HARMONISED EUROPEAN STANDARD
CYBER; CRA; Essential cybersecurity requirements for Browsers

Reference <Workitem> Keywords <keywords>

ETSI 650 Route des Lucioles F-06921 Sophia Antipolis Cedex - FRANCE Tel.: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16 Siret N° 348 623 562 00017 - APE 7112B Association à but non lucratif enregistrée à la Sous-préfecture de Grasse (06) N° w061004871

Important notice

The present document may be made available in electronic versions and/or in print. The content of any electronic and/or print versions of the present document shall not be modified without the prior written authorization of ETSI. In case of any existing or perceived difference in contents between such versions and/or in print, the prevailing version of an ETSI deliverable is the one made publicly available in PDF format on ETSI deliver repository.

Users should be aware that the present document may be revised or have its status changed, this information is available in the Milestones listing.

If you find errors in the present document, please send your comments to the relevant service listed under Committee Support Staff.

If you find a security vulnerability in the present document, please report it through our Coordinated Vulnerability Disclosure (CVD) program.

Notice of disclaimer & limitation of liability

The information provided in the present deliverable is directed solely to professionals who have the appropriate degree of experience to understand and interpret its content in accordance with generally accepted engineering or other professional standard and applicable regulations.

No recommendation as to products and services or vendors is made or should be implied.

No representation or warranty is made that this deliverable is technically accurate or sufficient or conforms to any law and/or governmental rule and/or regulation and further, no representation or warranty is made of merchantability or fitness for any particular purpose or against infringement of intellectual property rights.

In no event shall ETSI be held liable for loss of profits or any other incidental or consequential damages.

Any software contained in this deliverable is provided "AS IS" with no warranties, express or implied, including but not limited to, the warranties of merchantability, fitness for a particular purpose and non-infringement of intellectual property rights and ETSI shall not be held liable in any event for any damages whatsoever (including, without limitation, damages for loss of profits, business interruption, loss of information, or any other pecuniary loss) arising out of or related to the use of or inability to use the software.

Copyright Notification

No part may be reproduced or utilised in any form or by any means, electronic or mechanical, including photocopying and microfilm except as authorised by written permission of ETSI. The content of the PDF version shall not be modified without the written authorization of ETSI. The copyright and the foregoing restriction extend to reproduction in all media.

© ETSI yyyy.

All rights reserved.

Contents

Contents

Contents ii

Intellectual Property Rights xii

Foreword xii

Modal verbs terminology 1

Executive summary 1

Introduction 1

1 Scope 1
1.1 Browser . . . 2
1.1.1 Standalone . . . 2
1.1.2 Embedded . . . 2
1.1.3 Progressive Web Apps (PWA) . . . 2
1.1.4 Browser Extensions . . . 2
1.2 Derivative Browsers and Manufacturer Obligations . . . 3
1.2.1 Open Source Browser Engines and Derivative Products . . . 3
1.2.2 Spectrum of Derivative Modifications . . . 3
1.2.3 Manufacturer Responsibilities for Derivative Products . . . 4
1.2.4 Trust in Upstream Security Implementations . . . 4
1.2.5 Application of This Standard to Derivative Browsers . . . 4
1.2.6 State of the Art: Industry Testing and Security Practices . . . 5

2 References 7
2.1 Normative references . . . 7
2.2 Informative references . . . 7

3 Definition of terms, symbols and abbreviations 8
3.1 Terms . . . 8
3.2 Symbols . . . 10

4 Product Context 10
4.1 General . . . 10
4.2 Out of scope use/environments . . . 10
4.3 In-Scope Components . . . 11
4.3.1 In-Scope components standalone browser . . . 11
4.3.2 In-Scope components embedded browser . . . 12
4.4 Use Cases . . . 14
4.4.1 Application to Conformity Assessment . . . 15
4.4.2 Use Cases for Browsers . . . 15
4.5 Product overview and architecture . . . 18
4.5.1 Product Definition . . . 18
4.5.2 Architectural Overview . . . 18
4.5.2.1 Core Architecture Components . . . 18
4.5.2.2 Security Architecture . . . 18
4.5.2.3 Extension Architecture . . . 18
4.5.3 Trust Boundaries and Threat Model . . . 19
4.5.3.1 Trust Zones . . . 19
4.5.3.2 Attack Surface . . . 19
4.5.4 Deployment Contexts . . . 19
4.5.4.1 Consumer Environment . . . 19

4.5.4.2 Enterprise Environment . . . 19
4.5.4.3 Specialized Environments . . . 20
4.5.5 Security-Relevant Characteristics . . . 20
4.5.5.1 Dynamic Threat Landscape . . . 20
4.5.5.2 Compatibility Requirements . . . 20
4.5.5.3 Performance Constraints . . . 20
4.5.5.4 User Agency and Autonomy . . . 20
4.6 Essential functions . . . 20
4.6.1 Core Essential Functions . . . 21
4.6.2 Security-Related Essential Functions . . . 21
4.6.3 Embedded Browser-Specific Security Functions . . . 22
4.6.4 Functions NOT Considered Essential . . . 23
4.7 Operational Environment . . . 23
4.7.1 Technical Environment . . . 23
4.7.2 Physical Environment . . . 24
4.7.3 Organizational Environment . . . 24
4.7.4 Threat Environment . . . 25
4.7.5 Lifecycle Environment . . . 25
4.8 Users . . . 25
4.8.1 User Categories . . . 25
4.8.2 User Behavior Patterns . . . 26
4.8.3 User Needs and Expectations . . . 26
4.8.4 User Assistance and Responsibilities . . . 27
4.8.5 Accessibility Considerations . . . 27

5 Browser-Specific Risk Factors **28**
5.1 Isolation Mechanisms . . . 28
5.2 Extension System Security . . . 30
5.3 Encryption Implementation . . . 31
5.4 Diagnostic and Monitoring Systems . . . 34
5.5 Update Delivery Mechanisms . . . 37
5.6 Protocol Handler Security . . . 39
5.7 Core Component Security . . . 41
5.8 Embedded Browser Security . . . 44
5.8.1 Overview . . . 44
5.8.2 Host Application Boundary Security . . . 44
5.8.3 Content Source Trust Management . . . 46

6 Technical Security Assessments **51**
6.1 Domain and Origin Isolation Assessments . . . 51
Assessment: DOM-REQ-1 (Process-per-site isolation) . . . 51
Assessment: DOM-REQ-2 (Cross-origin read blocking) . . . 52
Assessment: DOM-REQ-3 (Strict origin policy enforcement) . . . 53
Assessment: DOM-REQ-4 (CORS preflight enforcement) . . . 53
Assessment: DOM-REQ-5 (Cookie SameSite attribute enforcement) . . . 54
Assessment: DOM-REQ-6 (Origin-bound storage isolation) . . . 55
Assessment: DOM-REQ-7 (Frame sandboxing support) . . . 56
Assessment: DOM-REQ-8 (Opaque origin handling) . . . 57
Assessment: DOM-REQ-9 (CORP for cross-origin isolation) . . . 57
Assessment: DOM-REQ-10 (COOP enforcement) . . . 58
Assessment: DOM-REQ-11 (COEP enforcement) . . . 59
Assessment: DOM-REQ-12 (Document.domain deprecation) . . . 60
Assessment: DOM-REQ-13 (Enterprise origin isolation policy configuration) . . . 61
Assessment: DOM-REQ-14 (Logging of policy-based isolation exceptions) . . . 61

Assessment: DOM-REQ-15 (Compatibility mode isolation integrity) . . . 62
Assessment: DOM-REQ-16 (Third-party integration isolation) . . . 63
Assessment: DOM-REQ-17 (Documentation and logging of compatibility exceptions) . . . 63
Assessment: DOM-REQ-18 (Embedded component storage isolation) . . . 64
6.2 Extension System Security Assessments . . . 65
Assessment: EXT-REQ-1 (Permission model for extensions) . . . 65
Assessment: EXT-REQ-2 (Content script isolation) . . . 66
Assessment: EXT-REQ-3 (Extension API access control) . . . 67
Assessment: EXT-REQ-4 (Manifest validation) . . . 68
Assessment: EXT-REQ-5 (Extension sandboxing) . . . 69
Assessment: EXT-REQ-6 (Cross-extension isolation) . . . 70
Assessment: EXT-REQ-7 (Host permissions validation) . . . 71
Assessment: EXT-REQ-8 (CSP for extensions) . . . 71
Assessment: EXT-REQ-9 (WebRequest API security) . . . 72
Assessment: EXT-REQ-10 (Extension update verification) . . . 73
Assessment: EXT-REQ-11 (Extension storage isolation) . . . 74
Assessment: EXT-REQ-12 (Background script restrictions) . . . 75
Assessment: EXT-REQ-13 (Manifest V3 compliance) . . . 76
Assessment: EXT-REQ-14 (Native messaging security) . . . 77
Assessment: EXT-REQ-15 (Extension-controlled web content) . . . 78
Assessment: EXT-REQ-16 (Extension telemetry privacy) . . . 79
Assessment: EXT-REQ-17 (Extension signature validation) . . . 80
Assessment: EXT-REQ-18 (Extension permissions UI transparency) . . . 81
Assessment: EXT-REQ-19 (No extension support enforcement) . . . 82
Assessment: EXT-REQ-20 (Extension code loading prevention) . . . 83
Assessment: EXT-REQ-21 (Extension subsystem removal) . . . 83
Assessment: EXT-REQ-22 (Official extension store restriction) . . . 84
Assessment: EXT-REQ-23 (Extension security review requirement) . . . 85
Assessment: EXT-REQ-24 (Developer mode activation security) . . . 86
Assessment: EXT-REQ-25 (Developer mode visual indicators) . . . 86
Assessment: EXT-REQ-26 (Developer mode update disablement) . . . 87
Assessment: EXT-REQ-27 (Developer mode activity logging) . . . 88
Assessment: EXT-REQ-28 (Enterprise developer mode control) . . . 89
Assessment: EXT-REQ-29 (Sideloaded extension warnings) . . . 90
Assessment: EXT-REQ-30 (User extension permission controls) . . . 90
Assessment: EXT-REQ-31 (Sideloaded extension malware scanning) . . . 91
Assessment: EXT-REQ-32 (Extension security event logging) . . . 92
6.3 Cryptographic Security Assessments . . . 93
Assessment: ENC-REQ-1 (TLS 1.3+ support) . . . 93
Assessment: ENC-REQ-2 (Certificate validation) . . . 94
Assessment: ENC-REQ-3 (Certificate pinning support) . . . 95
Assessment: ENC-REQ-4 (HSTS enforcement) . . . 96
Assessment: ENC-REQ-5 (Mixed content blocking) . . . 96
Assessment: ENC-REQ-6 (Certificate Transparency) . . . 97
Assessment: ENC-REQ-7 (OCSP stapling) . . . 98
Assessment: ENC-REQ-8 (Cipher suite restrictions) . . . 99
Assessment: ENC-REQ-9 (Perfect forward secrecy) . . . 100
Assessment: ENC-REQ-10 (Revocation checking) . . . 100
Assessment: ENC-REQ-11 (Web Crypto API compliance) . . . 101
Assessment: ENC-REQ-12 (Secure random number generation) . . . 102
Assessment: ENC-REQ-13 (SubResource Integrity) . . . 103
Assessment: ENC-REQ-14 (Encrypted SNI) . . . 104
Assessment: ENC-REQ-15 (Certificate error UI) . . . 105
Assessment: ENC-REQ-16 (HTTPS-first mode) . . . 106

Assessment: ENC-REQ-17 (Certificate pinning bypass detection) . . . 107
Assessment: ENC-REQ-18 (TLS downgrade protection) . . . 107
Assessment: ENC-REQ-19 (Legacy crypto deprecation) . . . 108
Assessment: ENC-REQ-20 (Cryptographic key isolation) . . . 109
Assessment: ENC-REQ-21 (Certificate store security) . . . 110
Assessment: ENC-REQ-22 (Certificate pinning override prohibition) . . . 111
Assessment: ENC-REQ-23 (Hardware-backed cryptographic operations) . . . 112
Assessment: ENC-REQ-24 (Encryption strictness configuration) . . . 113
Assessment: ENC-REQ-25 (Connection security visual indicators) . . . 113
Assessment: ENC-REQ-26 (User encryption settings control) . . . 114
Assessment: ENC-REQ-27 (Encryption status UI indicators) . . . 115
Assessment: ENC-REQ-28 (Legacy protocol compatibility mode) . . . 116
Assessment: ENC-REQ-29 (Complete encryption policy control) . . . 117
Assessment: ENC-REQ-30 (User-enabled legacy protocol support) . . . 118
6.4 Security Event Logging Assessments . . . 118
Assessment: LOG-REQ-1 (Security event logging) . . . 119
Assessment: LOG-REQ-2 (Certificate error logging) . . . 119
Assessment: LOG-REQ-3 (Extension security events) . . . 120
Assessment: LOG-REQ-4 (CSP violation reporting) . . . 121
Assessment: LOG-REQ-5 (Network Error Logging - NEL) . . . 122
Assessment: LOG-REQ-6 (Crash reporting) . . . 123
Assessment: LOG-REQ-7 (Log data minimization) . . . 123
Assessment: LOG-REQ-8 (Log anonymization) . . . 124
Assessment: LOG-REQ-9 (User consent for telemetry) . . . 125
Assessment: LOG-REQ-10 (Secure log transmission) . . . 126
Assessment: LOG-REQ-11 (Log integrity protection) . . . 127
Assessment: LOG-REQ-12 (Log retention policies) . . . 127
Assessment: LOG-REQ-13 (Security dashboard) . . . 128
Assessment: LOG-REQ-14 (Incident detection) . . . 129
Assessment: LOG-REQ-15 (Audit trail completeness) . . . 130
Assessment: LOG-REQ-16 (Real-time security alerts) . . . 131
Assessment: LOG-REQ-17 (Forensic log export) . . . 132
Assessment: LOG-REQ-18 (Privacy-preserving analytics) . . . 132
Assessment: LOG-REQ-19 (Compliance logging) . . . 133
Assessment: LOG-REQ-20 (Log access controls) . . . 134
Assessment: LOG-REQ-21 (No remote telemetry transmission) . . . 135
Assessment: LOG-REQ-22 (Opt-in local logging) . . . 136
Assessment: LOG-REQ-23 (User log management) . . . 137
Assessment: LOG-REQ-24 (Telemetry opt-in requirement) . . . 137
Assessment: LOG-REQ-25 (Granular telemetry control) . . . 138
Assessment: LOG-REQ-26 (Telemetry data documentation) . . . 139
Assessment: LOG-REQ-27 (Telemetry disable capability) . . . 140
Assessment: LOG-REQ-28 (Telemetry status display) . . . 141
Assessment: LOG-REQ-29 (Enterprise logging enforcement) . . . 141
Assessment: LOG-REQ-30 (Enterprise monitoring integration) . . . 142
Assessment: LOG-REQ-31 (Tamper-evident logging) . . . 143
6.5 Update Mechanism Security Assessments . . . 144
Assessment: UPD-REQ-1 (Automatic update mechanism) . . . 144
Assessment: UPD-REQ-2 (Update signature verification) . . . 145
Assessment: UPD-REQ-3 (HTTPS-only update delivery) . . . 146
Assessment: UPD-REQ-4 (Update manifest integrity) . . . 147
Assessment: UPD-REQ-5 (Rollback protection) . . . 148
Assessment: UPD-REQ-6 (Update channel isolation) . . . 149
Assessment: UPD-REQ-7 (Component update support) . . . 150

Assessment: UPD-REQ-8 (Emergency update capability) . . . 151
Assessment: UPD-REQ-9 (Update verification before installation) . . . 152
Assessment: UPD-REQ-10 (Update failure recovery) . . . 152
Assessment: UPD-REQ-11 (Update transparency logging) . . . 153
Assessment: UPD-REQ-12 (Delta update security) . . . 154
Assessment: UPD-REQ-13 (Update server authentication) . . . 155
Assessment: UPD-REQ-14 (Update timing jitter) . . . 156
Assessment: UPD-REQ-15 (Background update enforcement) . . . 157
Assessment: UPD-REQ-16 (Update notification UI) . . . 158
Assessment: UPD-REQ-17 (Forced update for critical vulnerabilities) . . . 159
Assessment: UPD-REQ-18 (Update verification chain) . . . 160
Assessment: UPD-REQ-19 (Update source pinning) . . . 161
Assessment: UPD-REQ-20 (Update integrity verification) . . . 162
Assessment: UPD-REQ-21 (Staged rollout support) . . . 163
Assessment: UPD-REQ-22 (Update domain validation) . . . 163
Assessment: UPD-REQ-23 (Update binary reproducibility) . . . 164
Assessment: UPD-REQ-24 (Automatic update application without user intervention) . . . 165
Assessment: UPD-REQ-25 (Automatic browser restart after critical updates) . . . 166
Assessment: UPD-REQ-26 (Update postponement with time limits) . . . 167
Assessment: UPD-REQ-27 (Critical update postponement restrictions) . . . 167
Assessment: UPD-REQ-28 (Update severity indication) . . . 168
Assessment: UPD-REQ-29 (Automatic updates disabled by default) . . . 169
Assessment: UPD-REQ-30 (Automatic update recommendations) . . . 170
Assessment: UPD-REQ-31 (Security warnings for available updates) . . . 170
Assessment: UPD-REQ-32 (Easy update check and install mechanism) . . . 171
Assessment: UPD-REQ-33 (Full user control over update timing) . . . 172
Assessment: UPD-REQ-34 (Manual update check mechanism) . . . 173
Assessment: UPD-REQ-35 (Outdated version security warnings) . . . 174
Assessment: UPD-REQ-36 (Clear indication of available security updates) . . . 175
Assessment: UPD-REQ-37 (Explicit user initiation for all updates) . . . 176
Assessment: UPD-REQ-38 (Update changelog and security impact display) . . . 176
6.6 Protocol Handler Security Assessments . . . 177
Assessment: PRO-REQ-1 (Protocol handler registration validation) . . . 178
Assessment: PRO-REQ-2 (User consent for custom protocols) . . . 178
Assessment: PRO-REQ-3 (Protocol allowlist enforcement) . . . 179
Assessment: PRO-REQ-4 (Scheme hijacking prevention) . . . 180
Assessment: PRO-REQ-5 (Protocol parameter sanitization) . . . 181
Assessment: PRO-REQ-6 (External protocol handler security) . . . 182
Assessment: PRO-REQ-7 (Protocol handler UI transparency) . . . 183
Assessment: PRO-REQ-8 (Protocol downgrade protection) . . . 184
Assessment: PRO-REQ-9 (Protocol handler logging) . . . 185
Assessment: PRO-REQ-10 (Web+custom scheme support) . . . 185
Assessment: PRO-REQ-11 (Protocol handler persistence) . . . 186
Assessment: PRO-REQ-12 (Protocol confusion mitigation) . . . 187
Assessment: PRO-REQ-13 (Handler capability restrictions) . . . 188
Assessment: PRO-REQ-14 (Protocol handler revocation) . . . 189
Assessment: PRO-REQ-15 (Cross-origin protocol restrictions) . . . 190
Assessment: PRO-REQ-16 (Protocol handler manifest validation) . . . 191
Assessment: PRO-REQ-17 (Intent URL security - mobile) . . . 191
Assessment: PRO-REQ-18 (Universal Links security - iOS) . . . 192
Assessment: PRO-REQ-19 (Deep linking validation) . . . 193
Assessment: PRO-REQ-20 (Protocol handler CSP integration) . . . 194
Assessment: PRO-REQ-21 (Handler registration audit trail) . . . 195
Assessment: PRO-REQ-22 (Protocol handler update security) . . . 196

Assessment: PRO-REQ-23 (Handler isolation enforcement) . . . 197
Assessment: PRO-REQ-24 (HTTP/HTTPS-only protocol restriction) . . . 198
Assessment: PRO-REQ-25 (Custom protocol handler registration rejection) . . . 199
Assessment: PRO-REQ-26 (Non-standard URL scheme blocking) . . . 200
Assessment: PRO-REQ-27 (Protocol handler API removal) . . . 200
Assessment: PRO-REQ-28 (Non-HTTP/HTTPS navigation rejection) . . . 201
Assessment: PRO-REQ-29 (Enterprise protocol handler policy controls) . . . 202
Assessment: PRO-REQ-30 (Custom scheme registration without web+ prefix) . . . 203
Assessment: PRO-REQ-31 (Non-standard protocol handler security warnings) . . . 203
Assessment: PRO-REQ-32 (Protocol handler review interface) . . . 204
Assessment: PRO-REQ-33 (Custom handler vulnerability scanning) . . . 205
Assessment: PRO-REQ-34 (Protocol handler security audit logging) . . . 206
6.7 System Resource Access Security Assessments . . . 207
Assessment: SYS-REQ-1 (Process sandbox enforcement) . . . 207
Assessment: SYS-REQ-2 (Renderer process isolation) . . . 208
Assessment: SYS-REQ-3 (GPU process isolation) . . . 209
Assessment: SYS-REQ-4 (Network service isolation) . . . 210
Assessment: SYS-REQ-5 (Filesystem access control) . . . 211
Assessment: SYS-REQ-6 (Device API permissions) . . . 211
Assessment: SYS-REQ-7 (PWA permission management) . . . 212
Assessment: SYS-REQ-8 (Geolocation permission enforcement) . . . 213
Assessment: SYS-REQ-9 (Camera/microphone access control) . . . 214
Assessment: SYS-REQ-10 (Clipboard access restrictions) . . . 215
Assessment: SYS-REQ-11 (Notification permission management) . . . 216
Assessment: SYS-REQ-12 (USB device access security) . . . 216
Assessment: SYS-REQ-13 (Bluetooth permission enforcement) . . . 217
Assessment: SYS-REQ-14 (File System Access API security) . . . 218
Assessment: SYS-REQ-15 (WebUSB security controls) . . . 219
Assessment: SYS-REQ-16 (WebBluetooth security) . . . 220
Assessment: SYS-REQ-17 (WebNFC permission management) . . . 221
Assessment: SYS-REQ-18 (Sensor API permissions) . . . 222
Assessment: SYS-REQ-19 (Battery Status API restrictions) . . . 223
Assessment: SYS-REQ-20 (Hardware resource limits) . . . 223
Assessment: SYS-REQ-21 (Memory isolation enforcement) . . . 224
Assessment: SYS-REQ-22 (CPU resource quotas) . . . 225
Assessment: SYS-REQ-23 (Network bandwidth limits) . . . 226
Assessment: SYS-REQ-24 (Storage quota enforcement) . . . 227
Assessment: SYS-REQ-25 (Process priority management) . . . 227
Assessment: SYS-REQ-26 (Sandbox escape prevention) . . . 228
Assessment: SYS-REQ-27 (Speculative execution mitigations) . . . 229
Assessment: SYS-REQ-28 (Side-channel attack mitigations) . . . 230
Assessment: SYS-REQ-29 (Hardware token security) . . . 231
Assessment: SYS-REQ-30 (Accessibility API security) . . . 232
Assessment: SYS-REQ-31 (Native messaging restrictions) . . . 233
Assessment: SYS-REQ-32 (Host OS integration security) . . . 233
Assessment: SYS-REQ-33 (Block all device API access at SYS-0) . . . 234
Assessment: SYS-REQ-34 (Block native messaging interfaces at SYS-0) . . . 235
Assessment: SYS-REQ-35 (Block host OS integration at SYS-0) . . . 236
Assessment: SYS-REQ-36 (PWA permission scope restrictions) . . . 237
Assessment: SYS-REQ-37 (PWA installation permission restrictions) . . . 238
Assessment: SYS-REQ-38 (PWA uninstallation permission revocation) . . . 239
Assessment: SYS-REQ-39 (PWA permission audit logging) . . . 239
Assessment: SYS-REQ-40 (Enterprise PWA permission policy controls) . . . 240
Assessment: SYS-REQ-41 (Native-equivalent API access controls at SYS-3) . . . 241

Assessment: SYS-REQ-42 (Transparent UI for native integration at SYS-3) 242
Assessment: SYS-REQ-43 (Security warnings for privileged API access at SYS-3) 243
Assessment: SYS-REQ-44 (Extended system access audit logging at SYS-3) 244
Assessment: SYS-REQ-45 (User permission review and revocation at SYS-3) 245
Assessment: SYS-REQ-46 (Enterprise restriction of native-equivalent features at SYS-3) . . . 246
Assessment: SYS-REQ-47 (Native integration audit documentation at SYS-3) 246
6.8 Embedded Browser Security Assessments 247
Assessment: EMB-REQ-1 (JavaScript bridge API allowlists) 247
Assessment: EMB-REQ-2 (JavaScript bridge input validation) 248
Assessment: EMB-REQ-3 (JavaScript bridge logging) 249
Assessment: EMB-REQ-4 (Context isolation) 250
Assessment: EMB-REQ-5 (User consent for sensitive APIs) 251
Assessment: EMB-REQ-6 (No system-level API exposure without controls) 251
Assessment: EMB-REQ-7 (Immutable bridge configuration) 252
Assessment: EMB-REQ-8 (Host credential protection) 253
Assessment: EMB-REQ-9 (JavaScript bridge security review) 254
Assessment: EMB-REQ-10 (Bridge API rate limiting) 255
Assessment: EMB-REQ-11 (Granular capability-based permissions) 256
Assessment: EMB-REQ-12 (Storage isolation from host) 256
Assessment: EMB-REQ-13 (CSP enforcement for embedded content) 257
Assessment: EMB-REQ-14 (Encrypted cross-process bridge) 258
Assessment: EMB-REQ-15 (Native UI overlay prevention) 259
Assessment: EMB-REQ-16 (API surface allowlisting over denylisting) 260
Assessment: EMB-REQ-17 (Certificate validation for remote content) 260
Assessment: EMB-REQ-18 (Trusted origin allowlisting) 261
Assessment: EMB-REQ-19 (Subresource Integrity for external scripts) 262
Assessment: EMB-REQ-20 (Certificate pinning with backup pins) 263
Assessment: EMB-REQ-21 (Mixed content prevention) 264
Assessment: EMB-REQ-22 (Trust decision logging) 264
Assessment: EMB-REQ-23 (Cryptographic signature verification for local content) 265
Assessment: EMB-REQ-24 (Redirect chain trust enforcement) 266
Assessment: EMB-REQ-25 (HSTS enforcement for trusted origins) 267
Assessment: EMB-REQ-26 (Certificate validation failure notification) 268
Assessment: EMB-REQ-27 (Network security configuration) 268
Assessment: EMB-REQ-28 (CSP enforcement for third-party content) 269
Assessment: EMB-REQ-29 (Per-instance trust policies) 270
Assessment: EMB-REQ-30 (Certificate Transparency verification) 271
Assessment: EMB-REQ-31 (DNS rebinding attack prevention) 272
Assessment: EMB-REQ-32 (Trust boundary violation security events) 272
Assessment: EMB-REQ-33 (Block web content access to host application objects at EMB-0) 273
Assessment: EMB-REQ-34 (No JavaScript bridge exposure at EMB-0) 274
Assessment: EMB-REQ-35 (Treat all content as untrusted at EMB-0) 275
Assessment: EMB-REQ-36 (Allowlist-only bridge API access at EMB-1) 276
Assessment: EMB-REQ-37 (Minimal and documented bridge API allowlist at EMB-1) 277
Assessment: EMB-REQ-38 (Bidirectional bridge security controls at EMB-2) 278
Assessment: EMB-REQ-39 (Web content callback handler validation at EMB-2) 279
Assessment: EMB-REQ-40 (Bridge message queuing with integrity protection at EMB-2) . . 280
Assessment: EMB-REQ-41 (Bridge traffic anomaly monitoring at EMB-2) 280
Assessment: EMB-REQ-42 (Enterprise bridge API policy configuration at EMB-2) 281
Assessment: EMB-REQ-43 (Core security boundaries preserved at EMB-3) 282
Assessment: EMB-REQ-44 (User awareness of native capabilities at EMB-3) 283
Assessment: EMB-REQ-45 (User permission review and revocation at EMB-3) 284
Assessment: EMB-REQ-46 (Native integration audit documentation at EMB-3) 285
Assessment: EMB-REQ-47 (Enterprise native integration restrictions at EMB-3) 286

6.9 Remote Data Processing Systems Security Assessments 287
Assessment: RDPS-REQ-1 (Offline functionality documentation) 287
Assessment: RDPS-REQ-2 (Data classification and inventory) 288
Assessment: RDPS-REQ-3 (Data criticality classification) 288
Assessment: RDPS-REQ-4 (TLS 1.3 encryption for data transmission) 289
Assessment: RDPS-REQ-5 (RDPS endpoint certificate validation) 290
Assessment: RDPS-REQ-6 (Retry mechanisms with exponential backoff) 291
Assessment: RDPS-REQ-7 (Local data caching for offline operation) 292
Assessment: RDPS-REQ-8 (Secure authentication for RDPS access) 292
Assessment: RDPS-REQ-9 (Certificate pinning for RDPS) 293
Assessment: RDPS-REQ-10 (RDPS connection timeout controls) 294
Assessment: RDPS-REQ-11 (RDPS connectivity failure logging) 295
Assessment: RDPS-REQ-12 (Graceful functionality degradation when RDPS unavailable) . . 295
Assessment: RDPS-REQ-13 (Credentials protection from RDPS exposure) 296
Assessment: RDPS-REQ-14 (RDPS request rate limiting) 297
Assessment: RDPS-REQ-15 (RDPS data validation before processing) 298
Assessment: RDPS-REQ-16 (Data at rest encryption in RDPS storage) 298
Assessment: RDPS-REQ-17 (Mutual TLS authentication for RDPS) 299
Assessment: RDPS-REQ-18 (Redundant data copies for recovery) 300
Assessment: RDPS-REQ-19 (Data recovery from backups with integrity verification) 301
Assessment: RDPS-REQ-20 (Data retention policies with secure deletion) 301
Assessment: RDPS-REQ-21 (Per-user per-origin access controls) 302
Assessment: RDPS-REQ-22 (RDPS access and modification auditing) 303
Assessment: RDPS-REQ-23 (Data integrity verification using cryptographic hashes) 303
Assessment: RDPS-REQ-24 (RDPS endpoint substitution attack protection) 304
Assessment: RDPS-REQ-25 (Replay attack defense for RDPS communications) 305
Assessment: RDPS-REQ-26 (Data minimization principles for RDPS transmissions) 306
Assessment: RDPS-REQ-27 (User controls for RDPS data synchronization) 306
Assessment: RDPS-REQ-28 (Secure data export from RDPS for data portability) 307
Assessment: RDPS-REQ-29 (User-configurable RDPS endpoint security settings) 308
Assessment: RDPS-REQ-30 (RDPS service availability verification before critical operations) 308
Assessment: RDPS-REQ-31 (Connection pooling with security controls for RDPS) 309
Assessment: RDPS-REQ-32 (RDPS authentication token protection from extraction and theft) 310
Assessment: RDPS-REQ-33 (End-to-end encryption for all critical data in RDPS) 311
Assessment: RDPS-REQ-34 (Hardware-backed key storage for RDPS encryption keys) 312
Assessment: RDPS-REQ-35 (High-availability RDPS architecture with failover) 313
Assessment: RDPS-REQ-36 (RDPS disaster recovery procedures documentation and testing) 314
Assessment: RDPS-REQ-37 (Real-time RDPS integrity monitoring) 314
Assessment: RDPS-REQ-38 (RDPS security event logging with SIEM integration) 315
Assessment: RDPS-REQ-39 (Geographic data residency requirements enforcement) 316
Assessment: RDPS-REQ-40 (Zero-trust architecture for RDPS access) 317
Assessment: RDPS-REQ-41 (Regulatory compliance logging for RDPS operations) 318
Assessment: RDPS-REQ-42 (Automated RDPS security scanning and vulnerability detection) 319
Assessment: RDPS-REQ-43 (Cryptographic proof of RDPS data integrity) 320
Assessment: RDPS-REQ-44 (Secure multi-tenancy with data isolation in RDPS) 321
Assessment: RDPS-REQ-45 (RDPS security incident response procedures) 322
Assessment: RDPS-REQ-46 (RDPS access revocation mechanisms) 322
Assessment: RDPS-REQ-47 (RDPS transparency reporting for data access) 323
Assessment: RDPS-REQ-48 (Forward secrecy for RDPS communications) 324
Assessment: RDPS-REQ-49 (User notification of RDPS security events) 325
Assessment: RDPS-REQ-50 (Service discontinuation and data migration documentation) . . 326
Assessment: RDPS-REQ-51 (Enterprise administrator RDPS security policy configuration) . 327
Assessment: RDPS-REQ-52 (Fully offline browser operation without remote connectivity) . . 328
Assessment: RDPS-REQ-53 (All user data stored locally without remote synchronization) . . 329

Assessment: RDPS-REQ-54 (No telemetry, diagnostics, or usage data transmission) 330
Assessment: RDPS-REQ-55 (No degradation when network connectivity unavailable) 331
Assessment: RDPS-REQ-56 (No remote authentication or authorization services required) . . 331
Assessment: RDPS-REQ-57 (Local-only operation capabilities and limitations documentation) 332
Assessment: RDPS-REQ-58 (User notification that no data leaves local system) 333
Assessment: RDPS-REQ-59 (All RDPS-1 requirements implemented for RDPS-2) 334
Assessment: RDPS-REQ-60 (All RDPS-2 requirements implemented for RDPS-3) 335

Annex A (informative): Mapping between the present document and CRA requirements 336

Annex B (informative): Mapping of Use Cases to Capabilities and Requirements 341
B.1 Use Case Mapping Methodology 341
B.2 Use Case to Capability Mappings 341
UC-B1: General Purpose Web Browsing (Risk Level: Standard) 341
UC-B2: Development and Testing Environments (Risk Level: High) 342
UC-B3: Kiosks and Shared Terminals (Risk Level: High) 342
UC-B4: Financial Services Access (Risk Level: High) 342
UC-B5: Healthcare and Medical Systems (Risk Level: High) 343
UC-B6: E-Government Services Access (Risk Level: High) 343
UC-B7: Enterprise Applications (Risk Level: High) 344
UC-B8: Critical Infrastructure (Risk Level: CRITICAL) 344
UC-B9: Security Research (Risk Level: CRITICAL) 344
UC-B10: Adapted Browser with Modified Features (Risk Level: Standard to High) 345
B.3 Capability Condition Level Selection Guide 345
B.4 Cross-Reference to Assessments 346
B.5 Remote Data Processing Systems (RDPS) Mapping 346
RDPS-0: No Remote Data Processing (Fully Local Operation) 346
RDPS-1: Limited Remote Processing (Non-Sensitive Data) 346
RDPS-2: Extended Remote Processing (Sensitive Data) 347
RDPS-3: Full Remote Processing (Critical Data - Maximum Security) 347
RDPS Capability Selection Matrix by Use Case 347

Annex C (informative): Relationship between the present document and any related ETSI standards (if any) 348

Annex D (informative): Risk identification and assessment methodology 348
D.1 Assets 348
D.1.1 Data 348
D.1.2 Product functions 349
D.2 Threats 349
D.3 Assumptions 349
D.4 Risk assessments of threats 349

Annex E (informative): Risk evaluation guidance 349
E.1 Mapping of risks to requirements 349
E.2 Risks not treated by the requirements 349
E.3 Risk acceptance criteria 349
E.4 Residual risks 350

Annex K 350

Annex L (informative): Relationship between the present document and the requirements of EU Regulation 2024/2847 350

Annex : Change history 350

Intellectual Property Rights

Essential patents

IPRs essential or potentially essential to normative deliverables may have been declared to ETSI. The declarations pertaining to these essential IPRs, if any, are publicly available for **ETSI members and non-members** , and can be found in ETSI SR 000 314: *"Intellectual Property Rights (IPRs); Essential, or potentially Essential, IPRs notified to ETSI in respect of ETSI standards"* , which is available from the ETSI Secretariat. Latest updates are available on the ETSI IPR online database.

Pursuant to the ETSI Directives including the ETSI IPR Policy, no investigation regarding the essentiality of IPRs, including IPR searches, has been carried out by ETSI. No guarantee can be given as to the existence of other IPRs not referenced in ETSI SR 000 314 (or the updates on the ETSI Web server) which are, or may be, or may become, essential to the present document.

Trademarks

The present document may include trademarks and/or tradenames which are asserted and/or registered by their owners. ETSI claims no ownership of these except for any which are indicated as being the property of ETSI, and conveys no right to use or reproduce any trademark and/or tradename. Mention of those trademarks in the present document does not constitute an endorsement by ETSI of products, services or organizations associated with those trademarks.

DECT™, **PLUGTESTS™**, **UMTS™** and the ETSI logo are trademarks of ETSI registered for the benefit of its Members. **3GPP™**, **LTE™** and **5G™** logo are trademarks of ETSI registered for the benefit of its Members and of the 3GPP Organizational Partners. **oneM2M™** logo is a trademark of ETSI registered for the benefit of its Members and of the oneM2M Partners. **GSM®** and the GSM logo are trademarks registered and owned by the GSM Association.

Foreword

DRAFT FOREWORD - DO NOT CONSIDER THE CONTENT

This draft Harmonised European Standard (EN) has been produced by ETSI Technical Committee Cyber Working Group for EUSR (CYBER-EUSR), and is now submitted for the combined Public Enquiry and Vote phase of the ETSI Standardisation Request deliverable Approval Procedure (SRdAP).

`The present document has been prepared under the Commission's standardisation request C(2025) 618 final`

Once the present document is cited in the Official Journal of the European Union under that Regulation, compliance with the normative clauses of the present document given in table A.1 confers, within the limits of the scope of the present document, a presumption of conformity with the corresponding requirements of that Regulation and associated EFTA regulations.

Transposition table

The Harmonised Standard shall have appropriate transposition periods specified. A Harmonised Standard confers presumption of conformity when it has been published in the Official Journal of the European Union (OJEU) and transposed by a member state.

The Technical Body may propose different dates to the default ones (3, 6, 18). Technical Bodies who wish to propose different dates are advised to indicate this clearly in the approved committee draft.

Proposed national transposition dates	
Date of latest announcement of this EN (doa):	3 months after ETSI publication
Date of latest publication of new National Standard	

Proposed national transposition dates	
or endorsement of this EN (dop/e):	6 months after doa
Date of withdrawal of any conflicting National Standard (dow):	18 months after doa

The Technical Body should advise the ETSI Secretariat if the above default national transposition dates are inappropriate for the particular standard.

Modal verbs terminology

In the present document "**should**", "**should not**", "**may**", "**need not**", "**will**", "**will not**", "**can**" and "**cannot**" are to be interpreted as described in clause 3.2 of the ETSI Drafting Rules (Verbal forms for the expression of provisions).

"**must**" and "**must not**" are **NOT** allowed in ETSI deliverables except when used in direct citation.

Executive summary

Browsers represent one of the most complex and security-critical software products in modern computing, serving as the primary gateway between users and internet resources while processing untrusted content from millions of sources daily. The browser's architecture encompasses multiple interconnected subsystems - including rendering engines, JavaScript/WebAssembly execution environments, network stacks, and extension frameworks, each presenting distinct attack surfaces that shall be defended while maintaining performance, compatibility with legacy web content, and user autonomy.

Unlike traditional security products that can enforce restrictive controls, browsers shall balance protection against an evolving threat landscape with respect for user choice, creating unique challenges where users may deliberately choose to visit malicious sites, install risky extensions, or disable security features. The browser's multi-layered trust model, spanning from the highly privileged browser core through semi-trusted extensions to completely untrusted web content, requires sophisticated isolation mechanisms, granular permission systems, and careful mediation of system resource access.

Given browsers' ubiquitous deployment across consumer, enterprise, and specialized environments, their role as platforms for Progressive Web Applications, and their position as primary targets for nation-state and criminal actors, establishing proportionate security requirements under the Cyber Resilience Act demands careful consideration of the inherent tensions between security, functionality, performance, and user agency that define the modern web browsing experience.

Introduction

This European harmonised standard defines cybersecurity requirements applicable to browsers.

This document will provide security requirements and assessment criteria covering all elements defined in CRA Annex I Part 1 and Part 2 for stand alone browsers, as mentioned in CRA Annex III Class I important products. This work item intends to produce an EN as candidate for harmonisation, under the standardisation request in support of the implementation of the CRA (M/606).

1 Scope

This standard focuses on browsers, both standalone and embedded. Browsers are software products with digital elements that enable end users to access and interact with web content hosted on servers that are connected to local and remote networks.

Within the context of an operating system, browsers are user-applications with a primary function and probable daily use. They are often leveraged as means of accessing remote authentication (single-sign-on) or even as a bridge (deep-link) to another application that has already been installed. In both cases, all systems have the notion of a "default browser" that can then be instrumented by other applications to navigate to a website or perform such an activity.

The activity of browsing can be defined in the following steps:

1. A machine accesses remote resources and source code, such as HTML, JavaScript/WebAssembly, and CSS.
2. This source is represented visually, acoustically, or in some other form.
3. The user interacts with the rendered representation through input and output interfaces, including visual observation, text entry, pointer interaction, or other supported input modalities.

1.1 Browser

1.1.1 Standalone

Standalone browsers are applications that fulfil the functions of browsing.

Web browsers are software applications that access, retrieve, and interact with information and resources addressed by URLs. A standalone browser may be used for everyday tasks such as reading email, managing a calendar, or consuming the news.

Such programs commonly have interfaces for managing multiple websites, browsing history, bookmarks, user identities, passwords, and other settings.

They can commonly be extended with browser extensions, which are products with digital elements that have the ability to read, store, and modify the websites that users interact with.

1.1.2 Embedded

Embedded browsers are browsing services that are integrated into another system or application.

As such, they are programs using the same baseline technology of browsing but are commonly used for "single purpose" browsing. This means that instead of opening the user's preferred standalone browser, the hosting application will open an embedded browser to keep the user's attention. It is not common for a user to be able to change the configuration of an embedded browser.

1.1.3 Progressive Web Apps (PWA)

Progressive Web Apps are web applications that can be installed to a user's device from a standalone browser and subsequently operate in a dedicated application-like context.

PWAs leverage browser capabilities including service workers, application manifests, and isolated storage to provide offline functionality, push notifications, and integration with operating system features such as the application launcher and task switcher. When installed, they execute within the browser's process architecture but present themselves to the user as distinct applications with their own windows, icons, and settings.

Unlike traditional web pages, installed PWAs maintain separate configuration contexts from the main browser, including distinct storage partitions, permission grants, and display modes. They may register custom protocol handlers, manage their own cache strategies through service workers, and receive operating system events such as share targets or file handlers. Despite this application-like presentation, PWAs remain fundamentally web applications subject to the same security boundaries and web platform APIs as content rendered in standard browser tabs.

1.1.4 Browser Extensions

Browser extensions are third-party software components that integrate with and extend the functionality of standalone browsers.

Extensions operate with elevated privileges compared to standard web content, enabling them to intercept and modify network requests, inject scripts into web pages, access cross-origin resources, interact with browser APIs, and persist data across browsing sessions. They are distributed through vendor-operated extension stores or side-loaded through developer modes, and are subject to varying degrees of review, validation, and ongoing monitoring depending on the browser vendor's policies.

Unlike web applications that execute within the constraints of the same-origin policy, extensions declare their required permissions through manifest files and, once granted, operate with capabilities that span multiple origins and browser contexts. They may consist of background scripts or service workers for persistent logic, content scripts that execute within web page contexts, popup interfaces, options pages, and other components. The security model of extensions creates a unique trust boundary where extensions act as intermediaries between the browser core and web content, requiring careful permission management, isolation mechanisms, and code signing to prevent abuse while enabling legitimate functionality enhancements.

1.2 Derivative Browsers and Manufacturer Obligations

A significant proportion of browsers placed on the market are derivative products based on open source browser engines or substantially complete browser implementations. Understanding the manufacturer's obligations for such derivative products is essential to the proportionate application of this standard.

1.2.1 Open Source Browser Engines and Derivative Products

Open source browser projects such as Chromium, Gecko (Firefox), and WebKit provide complete or near-complete browser implementations that serve as the foundation for derivative products. These upstream projects are stewarded by organizations that maintain the core rendering engines, JavaScript execution environments, network stacks, and security architectures, but the projects themselves do not constitute products placed on the EU market with CE marking.

When an economic operator takes such an open source project, applies modifications (whether substantial or minor), and places the resulting browser on the market under their own brand or distribution channel, that operator becomes a manufacturer under the Cyber Resilience Act [i.1]. This classification applies regardless of the extent of modification - from minor branding and default configuration changes to substantial feature additions, custom user interfaces, or integration of proprietary services.

1.2.2 Spectrum of Derivative Modifications

Derivative browsers exist along a spectrum of modification, each with implications for conformity assessment:

Minor Modifications: Browsers that modify only branding elements, default search providers, homepage settings, bundled bookmarks, or visual themes while maintaining the upstream codebase's security architecture, update mechanisms, and core functionality. Examples include rebranded releases for specific markets or partnerships.

Configuration-Level Modifications: Browsers that alter default privacy settings, tracking protection levels, extension policies, or feature flags to differentiate the product while preserving the underlying implementation. Such modifications may strengthen or weaken security postures relative to the upstream project.

Feature Additions: Browsers that integrate additional capabilities such as built-in VPN services, cryptocurrency wallets, AI assistants, proprietary synchronization services, or vertical-specific toolbars. These additions create new attack surfaces and data processing considerations beyond those present in the upstream project.

Architectural Modifications: Browsers that modify process architecture, sandbox implementations, network request routing, certificate validation logic, or other security-critical components. Such changes may fundamentally alter the security properties inherited from the upstream project.

1.2.3 Manufacturer Responsibilities for Derivative Products

Economic operators placing derivative browsers on the market bear full manufacturer obligations under the CRA, regardless of their reliance on upstream security implementations. These obligations include:

Security Requirement Compliance: Demonstrating that the derivative product, in its modified form, satisfies the essential cybersecurity requirements of Annex I of the CRA. While manufacturers may rely on the security properties of unmodified upstream components, any modifications should be assessed for their impact on those security properties.

Vulnerability Management: Establishing processes to monitor both upstream security advisories and vulnerabilities specific to the manufacturer's modifications. Timely integration of upstream security patches is a critical manufacturer responsibility, as delays in patch integration extend the exposure window for known vulnerabilities affecting end users.

Conformity Assessment: Conducting or commissioning technical assessments that address both the inherited security properties from the upstream project and the security implications of the manufacturer's specific modifications. The assessment should consider whether modifications have weakened, maintained, or strengthened the security posture.

Technical Documentation: Maintaining documentation that clearly delineates which components are inherited from the upstream project versus manufacturer modifications, security reviews conducted on modifications, processes for integrating upstream updates, and any divergences from upstream security defaults.

Update Delivery: Ensuring that security updates reach end users in a timely manner. For derivative browsers, this includes both the integration of upstream security patches into the manufacturer's codebase and the delivery of updated builds to end users through the manufacturer's distribution and update infrastructure.

1.2.4 Trust in Upstream Security Implementations

Manufacturers of derivative browsers commonly rely on the security implementations provided by upstream projects for foundational requirements such as TLS protocol implementation, cryptographic library usage, certificate validation, same-origin policy enforcement, and sandbox architecture. This reliance is reasonable provided that:

Upstream Security Processes are Verifiable: The upstream project demonstrates transparent security practices including public vulnerability disclosure, security-focused development processes, regular security audits, and timely patch releases.

Modifications Do Not Undermine Upstream Security: The manufacturer's changes do not bypass, weaken, or interfere with the security mechanisms inherited from the upstream project. For example, modifications that disable certificate validation, weaken content security policies, or reduce sandbox restrictions would constitute substantial security regressions requiring additional justification and compensating controls.

Integration Timeliness is Maintained: The manufacturer maintains a process to integrate upstream security patches within a reasonable timeframe. Extended delays between upstream patch availability and manufacturer distribution create unnecessary risk exposure for end users.

Deviation Points are Documented and Assessed: Where the manufacturer intentionally diverges from upstream security defaults (e.g., enabling features disabled upstream for security reasons, or modifying cryptographic configurations), these deviations are documented with security rationale and risk assessment.

1.2.5 Application of This Standard to Derivative Browsers

When applying the requirements of this standard to derivative browsers, manufacturers and assessors should consider:

Inherited vs. Modified Components: Requirements addressing components that remain unmodified from the upstream project may be satisfied by demonstrating that the upstream implementation meets the

requirement, provided the manufacturer's integration does not interfere with that implementation.

Modification-Specific Assessment: Requirements addressing areas where the manufacturer has made modifications require direct assessment of those modifications. This includes manufacturer-added features, modified defaults, integrated services, and any changes to security-critical code paths.

Update Mechanism Obligations: Even where a manufacturer relies on the upstream project's update mechanism architecture, the manufacturer remains responsible for ensuring that updates reach end users. This includes operating update servers, signing update packages, managing update channels, and ensuring update delivery reliability.

Use Case Alignment: Derivative browsers should be assessed against the use cases (Chapter 4.4) that align with their intended deployment contexts. A derivative browser marketed for general consumer use would align with UC-B1, while one marketed for enterprise deployment with proprietary features would align with UC-B7, regardless of their shared upstream heritage.

Derivative browsers represent a practical and economically significant category of products within the browser market. This standard recognizes that reliance on well-maintained upstream security implementations is a valid engineering approach, while maintaining that manufacturers placing derivative products on the market retain full responsibility for the security properties of the products they distribute.

1.2.6 State of the Art: Industry Testing and Security Practices

The state of the art for browser development and security validation encompasses organizational practices, industry standards, and comprehensive testing regimes that manufacturers should demonstrate to establish the quality and security of their browser implementations, whether original or derivative.

Organizational Practices and Resources:

Reputable browser manufacturers, both upstream projects and derivative product vendors, demonstrate their commitment to security through:

- **Adequate staffing**: Employment of sufficient numbers of developers and security personnel with expertise in browser architecture, web standards, cryptography, and vulnerability research
- **Security-focused development**: Dedicated security teams, secure development lifecycle practices, code review processes, and security architecture oversight
- **Transparency and communication**: Public disclosure of security policies, vulnerability handling procedures, and regular security bulletins
- **User commitment**: Published statements of commitment to user security and privacy, including privacy policies, data handling practices, and user control mechanisms
- **Update cadence**: Regular release schedules for security updates and patches, with clear timelines for critical vulnerability remediation

Industry Standards Compliance:

Browsers are expected to comply with applicable industry standards including but not limited to:

- **Web standards**: W3C specifications (HTML, CSS, JavaScript/ECMAScript, DOM, Fetch, etc.), WHATWG living standards
- **Security standards**: IETF RFCs for TLS, HTTP, WebAuthn, and related protocols; CA/Browser Forum Baseline Requirements
- **Accessibility standards**: WCAG (Web Content Accessibility Guidelines), ARIA (Accessible Rich Internet Applications)
- **Privacy standards**: Do Not Track, Global Privacy Control, tracking protection standards

Industry-Recognized Testing Frameworks:

Browsers should undergo testing using recognized industry test suites and frameworks:

Standards Conformance Tests:

- **Web Platform Tests (WPT)**: Comprehensive cross-browser test suite maintained by W3C and browser vendors, with public dashboard available at https://wpt.fyi/ showing conformance across implementations
- **Test262**: Official ECMAScript conformance test suite maintained by Ecma TC39, verifying JavaScript/ECMAScript specification compliance
- **W3C Test Suites**: Individual test suites for specific W3C specifications (CSS Working Group tests, HTML5 tests, etc.)
- **Acid Tests**: Historical but influential browser standards compliance tests (Acid1, Acid2, Acid3) developed by the Web Standards Project

Functional and Compatibility Testing:

- **Selenium**: Open-source testing framework for browser automation, widely used for functional testing and regression testing across browsers
- **BrowserStack**: Cloud-based cross-browser testing platform enabling compatibility verification across browser versions, operating systems, and devices
- **Playwright/Puppeteer**: Modern browser automation and testing frameworks providing programmatic control for automated testing

Security-Specific Testing and Validation:

- **CA/Browser Forum participation**: Engagement with the CA/Browser Forum (https://cabforum.org/working-groups/server/charter/) which establishes baseline requirements for certificate authorities and browser trust store policies
- **TLS/Certificate validation testing**: Testing against standard certificate validation scenarios, revocation checking, and TLS protocol compliance
- **Vulnerability disclosure programs**: Participation in responsible disclosure programs, bug bounty programs, and coordinated vulnerability disclosure processes
- **Penetration testing**: Regular security assessments by internal security teams or external security researchers

Standards Body Participation:

Active participation in standards bodies demonstrates commitment to interoperability and security best practices:

- **W3C (World Wide Web Consortium)**: Participation in working groups defining web standards
- **WHATWG (Web Hypertext Application Technology Working Group)**: Collaboration on living standards for HTML, DOM, and related specifications
- **IETF (Internet Engineering Task Force)**: Engagement in protocol standardization (TLS, HTTP, WebRTC, etc.)
- **Ecma International**: Participation in ECMAScript (JavaScript) standardization through TC39
- **CA/Browser Forum**: Involvement in establishing requirements for certificate authorities and browser root programs
- **FIDO Alliance**: Participation in authentication standards development (WebAuthn, FIDO2)

Implications for Derivative Browsers:

Derivative browser manufacturers should demonstrate:

1. **Upstream testing inheritance**: Evidence that the upstream project undergoes comprehensive testing via WPT, Test262, and other industry-standard test suites, with results publicly available
2. **Modification testing**: Testing of manufacturer-specific modifications using appropriate test frameworks to ensure modifications do not introduce regressions or security vulnerabilities
3. **Integration testing**: Validation that the integration of upstream components with manufacturer additions maintains standards compliance and security properties
4. **Security review process**: Documentation of security review procedures for modifications, including code review, security testing, and vulnerability assessment

5. **Update testing**: Verification that upstream updates are tested before distribution to ensure compatibility with manufacturer modifications

Manufacturers may demonstrate compliance with industry testing practices by referencing:

- Publicly available test results on wpt.fyi or similar dashboards
- Participation in open-source testing efforts
- Documentation of testing methodologies and results
- Third-party security assessments or certifications
- Membership in relevant standards bodies and working groups

The state of the art represents a comprehensive approach to browser quality and security, combining organizational commitment, standards compliance, extensive automated testing, security-focused practices, and community engagement. Derivative browser manufacturers should demonstrate that their products meet or exceed these industry norms, either through inheritance from well-maintained upstream projects or through their own testing and validation processes.

2 References

2.1 Normative references

In Harmonised Standards these references shall be specific *(identified by date of publication and/or edition number or version number)* ***publicly available and in English, except in exceptional circumstances making sure that impacts have been evaluated and explanations have been given on how any negative implications should be avoided*** *. See clauses 2.10.1 and 8.4 of the EDRs and the communiqué on "References in ETSI Deliverables".*

Guidance for selecting normative references in harmonised standards is given in clause 2.8.3 of the Vademecum on European standardisation. Please ***systematically consult with your Technical Officer*** *for the latest guidance on normative references other than to ENs, ISO/IEC standards, notably to prevent the risk of non-acceptance.*

Legal acts can never be used as normative references.

It is recommended that the number of references be limited to the minimum needed for the implementation/application of the ETSI Deliverables. References not directly concerned with the implementation/application/understanding of the ETSI Deliverable shall be listed in the Bibliography annex.

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the referenced document (including any amendments) applies.

Referenced documents which are not found to be publicly available in the expected location might be found in the ETSI docbox.

> NOTE: While any hyperlinks included in this clause were valid at the time of publication, ETSI cannot guarantee their long-term validity.

The following referenced documents are necessary for the application of the present document.

- [1] <Standard Organization acronym> <document number> (<version number>): "<Title>".

2.2 Informative references

References are either specific (identified by date of publication and/or edition number or version number) or nonspecific. For specific references, only the cited version applies. For non-specific references, the latest version of the referenced document (including any amendments) applies.

> NOTE: While any hyperlinks included in this clause were valid at the time of publication, ETSI cannot guarantee their long-term validity.

The following referenced documents may be useful in implementing an ETSI deliverable or add to the reader's understanding but are not required for conformance to the present document.

- [i.1] Regulation (EU) 2024/2847 of the European Parliament and of the Council of 23 October 2024 on horizontal cybersecurity requirements for products with digital elements and amending Regulations (EU) No 168/2013 and (EU) 2019/1020 and Directive (EU) 2020/1828 (Cyber Resilience Act).
- [i.2] NIST SP 800-128 (2011) Guide for Security-Focused Configuration Management of Information Systems
- [i.x] <Standard Organization acronym> <document number> (<version number>): "<Title>".

3 Definition of terms, symbols and abbreviations

3.1 Terms

The terms below are important for understanding the purpose and usage of browsers. For the purposes of the present document, the following terms apply:

Term	Definition
Access	The capability to retrieve, load, and display web content from servers through network protocols, including establishing connections, downloading resources, and rendering content for user consumption.
Accessing Web Content	The complete process by which browsers retrieve, process, and present web resources to end users, encompassing network communication, content parsing, rendering, and user interface presentation.
Browser Extensions	Software modules that augment browser functionality by adding features, modifying behavior, or enhancing user experience beyond the browser's core capabilities, typically installed and managed through the browser's extension system.
Browsers	Software products with digital elements that enable end users to access and interact with web content hosted on servers that are connected to local and remote networks. *Note: Expert group definition - In the context of this category of products, browsers are software products with digital elements that enable end users to access and interact with web content hosted on servers that are connected to networks such as the Internet.*
Certificate	A digital document issued by a Certificate Authority that validates the identity of a website and enables encrypted HTTPS connections, verified by browsers through cryptographic signature validation and certificate chain trust evaluation to prevent man-in-the-middle attacks.
Content Security Policy (CSP)	A browser security mechanism that allows web applications to declare approved sources for executable scripts, styles, and other resources through HTTP headers or meta tags, mitigating cross-site scripting attacks by preventing execution of unauthorized code.
Cross-Site Scripting (XSS)	A security vulnerability that allows attackers to inject malicious scripts into trusted websites viewed by other users, potentially stealing credentials, session tokens, or sensitive data by executing attacker-controlled code in the victim's browser context.
Custom Protocol	Non-standard or application-specific communication protocols that browsers may support for specialized content access or functionality, extending beyond traditional web protocols like HTTP/HTTPS.
Embedded Browsers	Browsers that are intended for integration into another system or application.

Term	Definition
End Users	Natural persons who utilize browsers to access web content for personal, professional, or other purposes, including but not limited to browsing, reading, viewing multimedia content, and interacting with web applications.
Exploit	A technique, code, or sequence of actions that takes advantage of a vulnerability to achieve unauthorized behavior, such as arbitrary code execution, privilege escalation, sandbox escape, or information disclosure.
Extension API	Programming interfaces exposed by browsers that enable extensions to access browser functionality, modify web content, intercept network requests, or integrate with browser features, subject to declared permissions and security policies.
Interact	The critical activity that defines browsing, encompassing user actions such as clicking hyperlinks, submitting forms, executing scripts, manipulating page elements, and engaging with dynamic web content through input devices.
Man-in-the-Middle (MITM) Attack	An attack where an adversary intercepts and potentially modifies network communication between a browser and server, often exploiting weak encryption, invalid certificates, or unencrypted HTTP connections to eavesdrop on or manipulate data transmission.
Networks	Communication infrastructures that enable data transmission between browsers and servers, encompassing local area networks (LANs), wide area networks (WANs), and the global Internet.
Origin	A fundamental security boundary defined by the combination of scheme (protocol), host (domain), and port of a URL, forming the basis for Same-Origin Policy enforcement and determining which web content can access shared resources, storage, and APIs.
Permission	A user-granted authorization that allows web content to access sensitive browser capabilities or device hardware (camera, microphone, location, notifications, etc.), managed through explicit user consent prompts and revocable through browser settings.
Process Isolation	The architectural pattern of separating browser components and web content into distinct operating system processes with independent memory spaces and restricted inter-process communication, containing the impact of security vulnerabilities and preventing cross-context data leakage.
Progressive Web Applications	Web-based applications that operate within the browser environment, leveraging advanced browser APIs and capabilities to provide enhanced functionality including offline operation, background synchronization, push notifications, and device hardware access, while remaining fundamentally dependent on the browser's runtime and security model for execution and user interaction.
Raw Content	Unprocessed source code and data formats delivered by servers, including but not limited to XML, JSON, JavaScript, HTML, CSS, and other markup or programming languages before browser interpretation.
Renderer Process	A sandboxed browser process responsible for parsing, executing, and displaying web content including HTML, CSS, and JavaScript, isolated from other content and the browser core to contain potential exploits within a restricted security boundary.

Term	Definition
Same-Origin Policy	The core browser security model that restricts how documents and scripts from one origin can interact with resources from another origin, preventing malicious websites from reading sensitive data or performing unauthorized actions on behalf of users across different web applications.
Sandbox	An operating system-level security mechanism that restricts the capabilities and system access of browser processes, limiting damage from compromised web content by preventing unauthorized filesystem access, system call execution, or privilege escalation beyond defined boundaries.
Servers	Computer systems or software applications that store, process, and deliver web content to browsers via network protocols, responding to browser requests with appropriate resources and data.
Standalone Browsers	Standalone applications that fulfil the functions of browsers.
Telemetry	Automated collection and transmission of browser usage data, performance metrics, crash reports, and diagnostic information to browser manufacturers for product improvement, typically requiring user consent and subject to privacy controls and data minimization principles.
Vulnerability	A weakness or flaw in browser implementation that can be exploited by malicious actors to bypass security controls, execute arbitrary code, access unauthorized data, or compromise system integrity, typically addressed through security updates and patches.
Web Content	The displayed and rendered representation of raw content, transformed by browsers into human-perceivable formats including text, images, videos, interactive elements, and structured layouts as intended by content creators.
WebView	A platform-specific embedded browser component that enables applications to display web content within their user interface, providing a subset of full browser functionality while operating under the security context and lifecycle of the host application. Common implementations include Android WebView, iOS WKWebView, Windows WebView2, and cross-platform frameworks such as Electron, Tauri, and Chromium Embedded Framework (CEF).

3.2 Symbols

For the purposes of the present document, the [following] symbols [given in ... and the following] apply:

[to be added]

4 Product Context

4.1 General

4.2 Out of scope use/environments

List uses/environments covered by other legislation or standards (critical, industrial, medical, etc.). Hoping to have a reusable generic list of these soon.

The types of product with digital elements listed in the section do not fall within the scope of the Cyber Resilience Act [i.1], and are not covered by this standard:

1. Services, except for the remote data processing solutions for a covered product as defined in CRA recitals 11-12; article 3, 2 [i.1];
2. Products specifically designed or procured for national security and defence purpose as defined in CRA recitals 14 and 26; article 2, 7-8 [i.1];
3. Products developed for or used exclusively for internal use by public administration as defined in CRA recital 16; article 5, 2 [i.1];
4. Non-commercial free and open source software as defined in CRA recitals 17-21; article 13, 5 [i.1];
5. Medical Devices and Software as defined in CRA recital 25; article 2, 2 [a-b] [i.1];
6. Vehicles, including aviation and marine equipment as defined in CRA recital 27; article 2, 2.c "vehicles"; recital 27; article 2, 3 "aviation"; article 2, 4 "marine equipment" [i.1];
7. Spare and used parts as defined in CRA recital 29; article 2, 6 [i.1];
8. Refurbished, repaired, and upgraded products that have not been substantially modifiedas defined in recitals 39 - 42 [i.1];

The following types of products have reduced or varied requirements under the Cyber Resilience Act [i.1] and can only be partially covered by this standard.

1. High Risk AI as defined in CRA recital 51; article 12 [i.1];
2. Testing and unfinished versions as defined in recital 37; Article 4, 2-3 [i.1];
3. Products Placed on the Market Prior to December 11, 2027 as defined in CRA article 69 [i.1].

4.3 In-Scope Components

4.3.1 In-Scope components standalone browser

For the purposes of this standard, a standalone browser consists of the following in-scope security-relevant components:

Core Browser Components:

1. **Rendering Engine**: HTML parser, CSS engine, layout system, and DOM implementation responsible for processing and displaying web content.
2. **JavaScript Engine**: JavaScript runtime, JIT compiler, garbage collector, and execution context management providing the environment for web application code execution.
3. **Network Stack**: HTTP/HTTPS client implementation, certificate validation, connection management, caching subsystem, and protocol handlers (WebSocket, WebRTC, etc.).
4. **Process Architecture**: Multi-process isolation model including browser process, renderer processes, GPU process, network process, and inter-process communication (IPC) mechanisms.
5. **Storage Subsystem**: Cookie management, localStorage, sessionStorage, IndexedDB, Cache API, origin-partitioned storage, and persistent storage quota management.
6. **Permission System**: Runtime permission prompts, permission state management, permission policy enforcement, and user consent UI for sensitive capabilities (camera, microphone, location, notifications, etc.).
7. **Sandbox Implementation**: Operating system-level process sandboxing, seccomp/AppContainer restrictions, filesystem access controls, and system call filtering.
8. **Security Policy Engines**: Same-Origin Policy enforcement, Cross-Origin Resource Sharing (CORS) validation, Content Security Policy (CSP) parser and enforcer, and Mixed Content blocking.

Extension System Components (if present):

9. **Extension Runtime**: Extension process management, manifest validation, permission enforcement for extension APIs, and content script injection mechanism.
10. **Extension API Layer**: Browser APIs exposed to extensions (webRequest, tabs, storage, etc.), permission-based access controls, and extension-to-browser IPC.

Update and Maintenance Components:

11. **Update System**: Automatic update mechanism, update signature verification, update rollback capability, and background update process.
12. **Diagnostic and Telemetry**: Crash reporting, error logging, usage metrics collection (where implemented with user consent), and debug logging infrastructure.

User Interface Components:

13. **Security Indicators**: HTTPS lock icon, certificate viewer, permission indicators, malicious site warnings, and phishing/malware protection UI.
14. **User Consent UI**: Permission prompts, download confirmations, external protocol handler registration prompts, and security warnings.

Certificate and Trust Components:

15. **Certificate Management**: Root certificate store, certificate validation logic, OCSP/CRL checking, Certificate Transparency verification, and certificate pinning.
16. **Trust Decisions**: Safe Browsing integration, malicious site detection, phishing protection, download scanning integration, and security warnings.

Out-of-Scope Components:

The following components are explicitly excluded from the security requirements of this standard:

- Third-party websites and web applications accessed through the browser
- Server-side infrastructure operated by the browser manufacturer (sync services, account systems) except where they deliver security-critical updates
- Operating system components and system libraries not distributed as part of the browser package
- Third-party extensions and plugins developed outside the browser manufacturer's control
- User-generated bookmarks, preferences, and configuration data
- Remote attestation or DRM modules that operate under separate security models

4.3.2 In-Scope components embedded browser

For the purposes of this standard, an embedded browser (WebView component or integrated browser engine) consists of the following in-scope security-relevant components in addition to or as variations of the standalone browser components listed in 4.3.1:

Embedded Browser Core Components:

1. **WebView Engine**: The embedded rendering engine (e.g., Android WebView, iOS WKWebView, Electron, CEF, WebView2) including HTML/CSS/JavaScript processing capabilities adapted for host application integration.
2. **Host Application Boundary**: Security boundary enforcement between web content running in the WebView and the native host application code, including context isolation and privilege separation.
3. **JavaScript Bridge**: Native-to-web and web-to-native communication interface allowing controlled interaction between JavaScript running in web content and native application APIs, including message passing and function exposure mechanisms.
4. **Custom URL Scheme Handlers**: Registration and handling of application-specific URL schemes (e.g., app://, custom-protocol://) that trigger native code execution or data retrieval from web content.
5. **WebView Configuration API**: Host application APIs for configuring WebView security properties (JavaScript enablement, file access permissions, content security settings, network access controls).

Embedded-Specific Security Components:

6. **Content Source Policy**: Allowlisting and trust management for content sources loaded into the WebView (local files, remote URLs, bundled assets), including validation of content origin and integrity.
7. **Storage Isolation**: Separation of WebView storage (cookies, localStorage, IndexedDB) from both the host application's native storage and from other WebView instances or standalone browsers on the same device.
8. **Permission Delegation**: Mechanism for handling web API permission requests (camera, microphone, location, etc.) where the WebView delegates permission decisions to the host application.
9. **Navigation Controls**: Host application controls over WebView navigation including URL allowlisting, navigation interception, redirect validation, and prevention of unintended navigation to external sites.
10. **Script Injection Controls**: Security mechanisms governing the injection of JavaScript into web content by the host application, including timing, isolation, and privilege level of injected scripts.

Host Integration Components:

11. **Native API Exposure Management**: Controlled exposure of native device capabilities (filesystem, camera, sensors, biometrics, secure storage) to web content through the JavaScript bridge with appropriate authorization and validation.
12. **Data Sharing Controls**: Mechanisms for secure data transfer between web content and native application context, including input validation, output encoding, and data sanitization at the boundary.
13. **Event Handling**: WebView event system for navigation events, load events, error events, and security-relevant events (certificate errors, SSL warnings, mixed content detection) with propagation to host application.
14. **Native UI Integration**: Security considerations for WebView rendering within native UI containers, including overlay protection, screenshot prevention for sensitive content, and secure display of trust indicators.

Update and Maintenance Components (Embedded-Specific):

15. **WebView Update Mechanism**: System for updating the embedded browser engine independently of the host application (platform-provided WebView updates) or bundled with application updates (Electron, CEF).
16. **Compatibility Validation**: Testing and validation that WebView security properties are maintained across engine updates and remain compatible with host application security requirements.

Additional Embedded Browser Considerations:

17. **Process Architecture**: Whether the WebView runs in-process with the host application or in a separate process, and the IPC mechanisms used for communication if separated.
18. **TLS/Certificate Management**: Handling of TLS connections including whether the WebView uses system certificate stores or custom certificate validation, and integration with certificate pinning.
19. **Debugging Interfaces**: Security controls around WebView debugging capabilities (browser developer tools, web inspector interfaces) to prevent unauthorized debugging of production applications.

Differences from Standalone Browsers:

Unlike standalone browsers where all web content is considered untrusted, embedded browsers shall:

- Establish selective trust relationships with certain content sources (bundled HTML, local files, specific remote origins) while maintaining security boundaries
- Mediate permissions through the host application rather than direct user prompts
- Share or isolate storage and state management with the host application
- Navigate the security boundary between web content privileges and native application privileges
- Consider the combined attack surface of both the WebView engine and the host application

Out-of-Scope Components for Embedded Browsers:

The following components are explicitly excluded from the security requirements for embedded browsers:

- Host application code outside the WebView integration points
- Server-side infrastructure serving content to the WebView (unless operated by the WebView engine provider for security updates)
- Third-party web content loaded into the WebView beyond the control of the host application developer
- Platform WebView implementations provided by the operating system (Android System WebView, iOS WKWebView) where the host application developer has no control over the engine implementation
- Native dependencies and libraries used by the host application but not related to WebView functionality

4.4 Use Cases

This clause defines representative use cases that illustrate the diverse operational contexts in which browsers and embedded browser components are deployed. These use cases serve multiple purposes within the conformity assessment framework:

1. **Risk Contextualization**: Each use case is associated with a risk level (Standard, High, or Critical) that reflects the potential impact of security failures in that deployment context, derived from the risk assessment methodology detailed in Annex A.
2. **Requirement Selection Guidance**: The use cases inform the selection of appropriate security capability levels from Chapter 5, helping manufacturers determine which condition levels are suitable for their intended deployment contexts.
3. **Proportionality Principle**: In accordance with the CRA's proportionality principle, these use cases demonstrate how security requirements scale with risk, ensuring that security controls are commensurate with the threats and impacts relevant to each deployment scenario.
4. **Shared Understanding**: The use cases provide a common vocabulary for manufacturers, assessors, and regulators to discuss browser security requirements in relation to real-world deployment contexts.

Scope and Applicability:

The use cases defined in this clause encompass both standalone browsers and embedded browser components (WebViews). While the fundamental security capabilities defined in Chapter 5 apply to both categories, the specific threats, deployment environments, and risk profiles differ:

- **Standalone browsers** (UC-B1 through UC-B8) operate as independent applications with direct user interaction and comprehensive security controls managed by the browser itself.
- **Embedded browsers** are represented in UC-B10 and in aspects of other use cases where browser engines are integrated into host applications, requiring consideration of host-web security boundaries and trust relationships.
- **Progressive Web Applications (PWAs)**, while representing an important deployment model, inherit their security properties from the underlying browser (standalone or embedded) in which they execute, and thus are covered by the applicable use case for that browser deployment.

Use Case Structure:

Each use case provides:

- **Description**: The primary purpose and scope of the deployment
- **Typical workflows**: Common user interactions and usage patterns
- **Typical environments**: Physical and network contexts in which the browser operates
- **Security considerations**: Key threats, vulnerabilities, and security controls relevant to the use case
- **Risk level**: Overall risk classification (Standard, High, or Critical)
- **Rationale**: Justification for the assigned risk level based on threat landscape and potential impact

Risk Levels Explained:

- **Standard Risk**: General-purpose deployments where security failures primarily affect individual users, with limited financial or societal impact. Standard security capabilities are appropriate.
- **High Risk**: Deployments involving sensitive personal data, financial transactions, health information, organizational data, or authenticated access to critical services. Enhanced security capabilities and stricter condition levels are recommended.
- **Critical Risk**: Deployments where security failures could result in significant physical harm, disruption of essential services, compromise of critical infrastructure, or large-scale societal impact. Maximum security capabilities with the strictest condition levels are required.

4.4.1 Application to Conformity Assessment

Manufacturers shall use these use cases as follows:

1. **Identify Applicable Use Cases**: Determine which use case(s) best represent the intended purpose and deployment context of the browser or embedded browser component.
2. **Assess Risk Level**: Evaluate whether the assigned risk level (Standard, High, Critical) aligns with the manufacturer's risk assessment for their specific deployment. Where multiple use cases apply, the highest applicable risk level should be considered.
3. **Select Security Capabilities**: Use Annex B to identify the recommended security capability condition levels for the applicable use case(s). These recommendations represent typical configurations; manufacturers may select stricter conditions based on their risk assessment or specific deployment requirements.
4. **Document Rationale**: In conformity assessment documentation, manufacturers should clearly identify which use case(s) apply to their product and provide justification for any deviations from recommended capability levels.
5. **Consider Use Case Combinations**: Many deployments will span multiple use cases (e.g., a browser used for both general web browsing and enterprise applications). In such cases, manufacturers should satisfy requirements for all applicable use cases or implement use-case-specific profiles.

Note: These use cases are representative and not exhaustive. Manufacturers deploying browsers in contexts not explicitly covered by these use cases should conduct a detailed risk assessment per Annex A and select security capability levels appropriate to the identified risks.

4.4.2 Use Cases for Browsers

UC-B1: General Purpose Web Browsing

- Description: Browsing of public websites for news, social media, entertainment, shopping, streaming, and general information. Excludes authenticated access to sensitive personal or organizational systems.
- Typical workflows: High tab count with frequent context switching; passive consumption (reading/watching); form fills; long idle sessions.
- Typical environments: Personal devices used in homes, cafes, transit, or public spaces. No physical security controls; high exposure to shoulder surfing, untrusted networks, or device theft.
- Security considerations: Tracking protection; HTTPS-only mode; blocking of malicious sites; access to camera/microphone/location; exposure to browser extensions; integrated password management; profile separation.
- Risk level: Standard
- Rationale: Primary threats originate from web content (malware, trackers, phishing) rather than targeted compromise.

UC-B2: Development and Testing Environments

- Description: Browser usage by developers, QA, and testers for building, debugging, and validating web applications, including compatibility testing with pre-release (canary/nightly/beta) browser builds.

- Typical workflows: Frequent navigation to localhost/internal IPs; manual and automated interaction with untrusted or malformed code; usage of developer tools; HAR/network capture; testing auth flows.
- Typical environments: Developer workstations in office or home offices; may be BYOD or corporate-managed; moderate physical security.
- Security considerations: Isolation using ephemeral profiles or web browser private mode, allowlists for extensions, supply chain auditing, curation and anonymization of test data.
- Risk level: High
- Rationale: Exposure to untrusted code, experimental browser features, and misconfiguration increases likelihood of exploit execution or leakage of credentials.

UC-B3: Kiosks and Shared Terminals

- Description: Multi-user public or semi-public devices for check-in, customer service, library access, clinic intake, or retail assistance.
- Typical workflows: Short, single-purpose sessions (check-in, lookup, form submission); no authentication or credential entry.
- Typical environments: Fixed-location terminals in lobbies, libraries, clinics, classrooms or stores.
- Security considerations: Strict domain allowlist; no access to camera/mic/location/notifications; block credential saving and autofill; remote wipe and health monitoring; encryption of cached assets.
- Risk level: High
- Rationale: High turnover of untrusted users combined with potential for physical tampering.

UC-B4: Financial Services Access

- Description: Access to online banking, brokerage, payments, crypto exchanges, wallets, and insurance portals involving monetary transactions or sensitive financial data.
- Typical workflows: Daily logins; balance checks; initiating transfers/payments; uploading documents.
- Typical environments: Primarily personal devices in uncontrolled locations; corporate devices in secure facilities
- Security considerations: HTTPS-only mode; credential monitoring for breaches.
- Risk level: High
- Rationale: Exposure to financial fraud and account takeover; session hijacking; MITM;

UC-B5: Healthcare and Medical Systems

- Description: Browser access to EHRs, telemedicine platforms, patient portals, prescription systems or health insurance.
- Typical workflows: Patient record access/modification; remote consultations; e-prescribing; uploading/downloading diagnostic files.
- Typical environments: #TODO: Similar to home + hospital workstations?
- Security considerations: Session re-auth for sensitive actions, auto-timeout after inactivity, data encryption at rest/in transit.
- Risk level: High
- Rationale: Regulatory penalties, reputational damage, and potential patient safety risks.

UC-B6: E-Government Services Access

- Description: Access to citizen-facing or administrative government portals for taxes, benefits, licenses, identity verification, legal filings, or voting systems.
- Typical workflows: Infrequent but critical sessions, form filling, document uploads, digital signature application.
- Typical environments: #TODO: Personal devices or public terminals.
- Security considerations: Strong authentication; digital signature validation; certificate management.
- Risk level: High
- Rationale: Compromise can lead to identity theft, benefit fraud, election interference, or erosion of civic trust.

UC-B7: Enterprise Applications

- Description: Internal browser-based tools for CRM, ERP, HRMS, document collaboration, project management and BI,
- Typical workflows: Daily CRUD operations of records, document management,
- Typical environments: Corporate laptops, desktops, or BYOD devices used remotely or on-premise.
- Security considerations: SSO, DLP (control copy/paste/print/download actions), allowlist for extensions, integration with SIEM, containerization of BYOD.
- Risk level: High
- Rationale: Exposure of intellectual property, customer data, or internal operations to exfiltration or insider threat.

UC-B8: Critical Infrastructure

- Description: Web interfaces for SCADA, energy grid, water treatment, transportation control, or emergency dispatch.
- Typical workflows: Administered access, scoped to individual user accounts.
- Typical environments: Secure control rooms, data centers, or field stations - physically access-controlled, often air-gapped or operating within segmented network.
- Security considerations: Certificate management; zero trust architecture; mTLS; RBAC; supply chain controls; immutable logging.
- Risk level: Critical
- Rationale: Successful attack could disrupt essential services, cause physical damage, or endanger human life.

UC-B9: Security Research #TODO: Perhaps this one is too much

- Description: Intentional browsing to analyze phishing pages, malware or malicious extensions.
- Typical workflows: Automated or manual navigation to malicious URLs; downloading/executing payloads in sandbox; DOM inspection; screenshot/HAR capture; behavioral logging.
- Typical environments: Dedicated lab machines or air-gapped workstations, often behind shielded network zones.
- Security considerations: Isolation using disposable VMs, capture of all artifacts and network traffic;
- Risk level: Critical
- Rationale: Deliberate exposure to live threats; failure can lead to host compromise, lateral movement, or data exfiltration.

UC-B10: Adapted Browser with Modified Features

- Description: A manufacturer creates a product based on an existing open-source browser (e.g., Chromium, Firefox) by adding custom features, modifying default configurations, integrating proprietary services, or tailoring the browser for specific market segments, enterprise deployments, or regional requirements.
- Typical workflows: Standard web browsing workflows similar to general-purpose browsers, but with manufacturer-specific features such as custom home pages, integrated search providers, proprietary sync services, enhanced privacy controls, vertical-specific toolbars, or pre-configured extension bundles.
- Typical environments: All environments applicable to the underlying browser engine (personal devices, corporate workstations, mobile devices, kiosks), but with manufacturer-controlled default configurations and update channels.
- Security considerations: Inheritance of upstream browser vulnerabilities; security review of added features and modifications; validation that customizations do not weaken existing security controls; secure management of manufacturer-operated services (sync, accounts, analytics); timely integration of upstream security patches; transparency regarding data collection by added features; supply chain security for bundled extensions or services; verification that modifications maintain sandboxing and isolation properties; compliance with baseline browser security requirements while accounting for manufacturer additions.
- Risk level: Standard to High (depends on extent of modifications and deployment context)
- Rationale: The security posture depends on both the upstream browser's security and the manufacturer's implementation quality. Added features introduce additional attack surface and potential vulnerabilities. Delayed or incomplete integration of upstream patches can extend exposure to known vulnerabilities.

Manufacturer-operated services create additional trust dependencies and data processing considerations. However, when properly implemented, adapted browsers can maintain equivalent security to their upstream base while providing differentiated user value. The risk level increases when modifications are extensive, when manufacturer services handle sensitive data, or when the browser is deployed in high-risk contexts (UC-B4 through UC-B8).

4.5 Product overview and architecture

4.5.1 Product Definition

A standalone browser is a software application that enables users to access, retrieve, and interact with content on the World Wide Web and other network resources. Unlike embedded browsers or WebView components integrated into other applications, standalone browsers operate as independent applications with direct user interfaces, comprehensive feature sets, and autonomous update mechanisms.

4.5.2 Architectural Overview

4.5.2.1 Core Architecture Components

The browser architecture consists of several interconnected subsystems:

Rendering Engine: Processes HTML, CSS, and JavaScript to display web content. This includes the DOM parser, CSS engine, and layout system that transforms web resources into visual representations.

JavaScript Engine: Executes JavaScript code in isolated contexts, providing the runtime environment for dynamic web applications while maintaining security boundaries between different execution contexts.

Network Stack: Manages all network communications including HTTP/HTTPS requests, WebSocket connections, and other protocols. Implements connection pooling, caching, and security features such as certificate validation.

Process Architecture: Modern browsers employ multi-process architectures where the main browser process is separated from renderer processes, plugin processes, and GPU processes. This isolation prevents compromise of one process from affecting others.

Storage Subsystem: Manages various forms of local data storage including cookies, localStorage, IndexedDB, and cache storage, each with distinct security properties and access controls.

4.5.2.2 Security Architecture

Sandbox Model: Web content executes within restricted sandboxes that limit access to system resources. The sandbox is enforced at multiple levels - process isolation at the OS level, and API restrictions at the browser level.

Permission System: Browsers mediate access to sensitive capabilities (camera, microphone, location, notifications) through a permission system that requires explicit user consent and provides ongoing usage indicators.

Content Security Boundaries: The Same-Origin Policy (SOP) forms the fundamental security boundary, ensuring that content from one origin cannot access resources from another origin without explicit permission.

Certificate and Trust Management: Browsers maintain root certificate stores and implement certificate validation, including mechanisms for certificate pinning, HSTS (HTTP Strict Transport Security), and Certificate Transparency.

4.5.2.3 Extension Architecture

Browser extensions operate in a unique position within the security model:

- **Privileged Execution Context**: Extensions run with elevated privileges compared to web content, able to intercept and modify network requests, access cross-origin resources, and interact with browser APIs.
- **Manifest-Based Permissions**: Extensions declare required permissions in manifests, which are presented to users during installation. However, the granularity and understandability of these permissions remain challenges.
- **Content Script Injection**: Extensions can inject scripts into web pages, creating a three-way trust relationship between the browser, the extension, and the web content.

4.5.3 Trust Boundaries and Threat Model

4.5.3.1 Trust Zones

1. **Browser Core** (Highest Trust): The browser executable and core libraries, typically signed and verified by the operating system.
2. **Browser Extensions** (Elevated Trust): Third-party code with significant privileges, operating between web content and browser core.
3. **Web Applications** (Limited Trust): JavaScript applications running within the browser sandbox with restricted capabilities.
4. **Web Content** (Untrusted): General web content including advertisements, user-generated content, and third-party resources.
5. **Network** (Untrusted): All network communications are considered potentially hostile, requiring encryption and validation.

4.5.3.2 Attack Surface

The browser presents multiple attack surfaces:

- **Network Interface**: Exposed to malicious servers, man-in-the-middle attacks, and protocol-level exploits
- **Rendering Engine**: Subject to parsing vulnerabilities, memory corruption, and logic bugs
- **JavaScript Engine**: Target for JIT compiler bugs, type confusion, and sandbox escapes
- **Extension APIs**: Abused by malicious extensions or compromised legitimate extensions
- **User Interface**: Social engineering through spoofing, phishing, and permission fatigue
- **Local Storage**: Cross-site tracking, data exfiltration, and persistence mechanisms

4.5.4 Deployment Contexts

4.5.4.1 Consumer Environment

Individual users installing browsers on personal devices, typically with:

- Direct internet connectivity
- Mixed personal and professional usage
- User-managed security decisions
- Varied technical expertise levels

4.5.4.2 Enterprise Environment

Organizational deployments requiring:

- Centralized policy management
- Compliance with regulatory requirements
- Integration with security infrastructure (proxies, SIEM systems)

- Controlled update cycles
- Restriction of certain features or extensions

4.5.4.3 Specialized Environments

- **Kiosk Mode**: Public-facing browsers with locked-down configurations
- **Development Mode**: Browsers with relaxed security for testing
- **Privacy-Focused**: Configurations emphasizing anonymity and tracking prevention
- **Isolated Browsing**: Air-gapped or heavily restricted network access

4.5.5 Security-Relevant Characteristics

4.5.5.1 Dynamic Threat Landscape

Browsers face constantly evolving threats as new web standards introduce new capabilities, zero-day vulnerabilities are discovered, and attack techniques advance. The browser serves as the primary interface between users and potentially hostile internet content.

4.5.5.2 Compatibility Requirements

Browsers shall maintain backward compatibility with legacy web content while implementing new security features, creating tension between security and functionality. This includes supporting older TLS versions, maintaining compatibility with enterprise applications, and handling non-standard web content. Should the usage of legacy web content impact the security of the browser, then the content shall be ignored.

4.5.5.3 Performance Constraints

Security measures shall be balanced against performance requirements. Users expect instantaneous page loads and smooth interactions, limiting the computational overhead available for security checks. This affects decisions around sandboxing granularity, encryption methods, and validation procedures.

4.5.5.4 User Agency and Autonomy

Unlike many security products, browsers shall respect user choice while protecting against threats. Users may choose to:

- Visit dangerous websites despite warnings
- Install risky extensions
- Disable security features for compatibility
- Share sensitive information voluntarily

This requirement for user agency fundamentally shapes the browser security model, requiring informed consent mechanisms rather than purely restrictive controls.

4.6 Essential functions

The essential functions of a browser, as defined for the purposes of this standard, are those capabilities that shall remain operational to fulfill the browser's primary purpose of enabling secure access to web content and services. These functions form the baseline against which security requirements are assessed.

This section addresses both **standalone browsers** (independent applications with direct user interfaces) and **embedded browsers** (browser engines integrated into host applications, commonly referred to as WebViews, browser components, or embedded web rendering engines).

4.6.1 Core Essential Functions

Content Retrieval and Rendering: The browser shall retrieve web resources via network protocols (primarily HTTP/HTTPS) and render them into a visual and interactive presentation for the user. This includes:

- HTML parsing and DOM construction
- CSS styling and layout computation
- JavaScript and WebAssembly execution within secure sandboxes
- Rendering of images, media, and other embedded content

For embedded browsers: The host application may provide additional constraints on content sources, implement content filtering, or restrict certain rendering features. However, the fundamental capability to securely parse and render web content remains essential. See reference: WebView Security Best Practices.

Navigation and Session Management: The browser shall enable navigation between web resources and maintain session state including navigation history and form data.

For standalone browsers: This includes managing multiple concurrent browsing contexts (tabs, windows) with independent session states.

For embedded browsers: Navigation may be constrained by the host application to approved domains or URL patterns. The host application is responsible for implementing navigation controls and may delegate or restrict back/forward navigation. Session management may be simplified or controlled by the host application. See reference: Android WebView Security.

Cryptographic Communication: The browser shall establish encrypted connections to web servers using TLS/SSL protocols (TLS 1.2 minimum, TLS 1.3 recommended), validate server certificates, and provide indicators of connection security status.

For standalone browsers: Visual security indicators (padlock icons, address bar coloring) shall be prominently displayed in the user interface.

For embedded browsers: Security indicators may be delegated to the host application's UI. The host application shall be responsible for surfacing certificate validation status to end users when handling sensitive data. Certificate pinning may be implemented at the host application level. See references: Certificate Pinning, TLS Best Practices.

Data Storage: The browser shall provide controlled mechanisms for web applications to store data locally, including cookies, localStorage, IndexedDB, and cache storage, subject to security policies and user controls.

For embedded browsers: Storage isolation from the host application is critical. The embedded browser shall maintain separate storage contexts that are not directly accessible to host application code without explicit bridging. Storage may be partitioned per embedded browser instance to prevent data leakage between different uses within the same application. See reference: WebView Data Isolation.

User Input and Interaction: The browser shall accept and process user inputs (keyboard, mouse, touch) and deliver them securely to web content, while protecting against input injection attacks and unauthorized access to input devices.

For embedded browsers: Input handling shall prevent the host application from injecting synthetic events that could bypass user consent (e.g., programmatically triggering clicks on permission prompts). The boundary between host-provided input and user-generated input shall be clearly maintained. See reference: Input Validation in WebViews.

4.6.2 Security-Related Essential Functions

Isolation and Sandboxing: The browser shall maintain security boundaries between different origins, processes, and execution contexts to prevent unauthorized access and limit the impact of compromised content.

For embedded browsers: A critical additional boundary exists between the embedded browser content and the host application. This boundary shall prevent:

- Web content from accessing host application memory, files, or system resources without explicit bridging
- Web content from executing arbitrary code in the host application context
- Host application code from arbitrarily accessing or manipulating web content internal state

The host application shall implement a secure bridge or message-passing interface for controlled communication between web content and native code. See references: WebView Isolation, iOS WKWebView Security.

Permission Management: The browser shall mediate access to sensitive capabilities (camera, microphone, location, notifications, clipboard) through a permission system that requires user consent and provides ongoing visibility.

For embedded browsers: Permission requests may be delegated to the host application, which is then responsible for obtaining appropriate user consent and respecting platform-level permission grants. The host application shall not automatically grant permissions to embedded web content without user awareness. Permissions granted to the host application shall not automatically extend to all web content loaded in embedded browsers. See reference: WebView Permissions Model.

Update Mechanism: The browser shall maintain the capability to receive, verify, and apply security updates to address vulnerabilities and maintain security effectiveness over the product lifecycle.

For standalone browsers: Direct update mechanisms with user notification and control.

For embedded browsers: Updates are typically delivered as part of operating system updates (for system WebViews) or application updates (for bundled browser engines). The host application developer is responsible for:

- Using up-to-date versions of embedded browser components
- Monitoring security advisories for the embedded browser engine
- Deploying application updates that include patched browser components
- Not pinning to outdated versions of browser engines with known vulnerabilities

See references: Android System WebView Updates, Electron Security Updates.

Certificate Validation: The browser shall validate server certificates against trusted root certificate authorities and enforce certificate transparency requirements to detect mis-issued certificates.

For embedded browsers: Certificate validation shall use the platform's trusted certificate store unless explicit certificate pinning is implemented by the host application. The host application shall not disable certificate validation except in clearly documented development/testing scenarios that are disabled in production builds. See reference: Certificate Validation Best Practices.

4.6.3 Embedded Browser-Specific Security Functions

JavaScript Bridge Security: For embedded browsers that expose native functionality to web content through JavaScript bridges (e.g., `addJavascriptInterface` in Android WebView, `WKScriptMessageHandler` in iOS), the following are essential:

- **Input Validation**: All data passed from web content to native code shall be validated and sanitized to prevent injection attacks
- **Output Encoding**: Data passed from native code to web content shall be properly encoded to prevent XSS
- **Minimal Exposed Surface**: Only necessary functionality shall be exposed; avoid exposing powerful or sensitive APIs
- **Authentication**: Bridge calls that perform sensitive operations shall verify origin and intent
- **Intent Validation**: For URL schemes and intent handlers, validate and sanitize all parameters

See references: Android WebView JavaScript Interface Security, iOS JavaScript Bridge Security.

Content Source Validation: Embedded browsers shall validate and restrict content sources:

- **Allowlist Enforcement**: Maintain allowlists of approved domains/URLs when loading sensitive functionality
- **Local Content Handling**: When loading local HTML/JS resources, ensure they cannot be overwritten or manipulated by untrusted code
- **Deep Link Validation**: Validate and sanitize deep links that cause content to load in embedded browsers

See reference: Mobile Application Security Testing Guide - WebViews.

Host Application Data Exposure Prevention: The embedded browser shall prevent web content from accessing host application data through:

- **File System Isolation**: Preventing JavaScript file:// access to arbitrary application files
- **Database Isolation**: Preventing web content from accessing native application databases
- **Shared Preferences/User Defaults Isolation**: Protecting native application settings from web access
- **Intent/URL Scheme Filtering**: Validating and restricting URL scheme invocations that could manipulate the host application

See reference: OWASP Mobile Security Testing Guide.

4.6.4 Functions NOT Considered Essential

The following functions, while commonly present in browsers, are not considered essential for the core purpose and may be disabled or restricted in high-security configurations without fundamentally impairing browser functionality:

For standalone browsers:

- Extension/add-on support
- Synchronization of browsing data across devices
- Built-in password management
- Autofill of forms
- Access to hardware devices beyond display and basic input
- Developer tools and debugging interfaces (except in designated development builds)
- Support for legacy protocols or deprecated web standards
- Integration with external applications or services beyond standard web APIs

For embedded browsers (in addition to the above):

- Multiple tab/window management (may be simplified to single content view)
- Browser history UI (may be delegated to host application)
- Download management UI (host application typically handles downloads)
- Print functionality (host application may provide)
- Find-in-page functionality (host application may provide)
- Reader mode or content simplification features

4.7 Operational Environment

The operational environment encompasses the technical, physical, and organizational contexts in which browsers are deployed and operated. Understanding these environments is essential for appropriate security configuration and risk assessment.

4.7.1 Technical Environment

Platform Diversity: Browsers operate across heterogeneous platforms including:

- Desktop operating systems (Windows, macOS, Linux)

- Mobile operating systems (iOS, Android)
- Embedded systems and IoT devices
- Virtualized and containerized environments

Each platform provides different security primitives (process isolation, memory protection, file system controls) that browsers leverage for security enforcement.

Network Conditions: Browsers shall operate across varying network environments:

- Trusted corporate networks with security controls and monitoring
- Home networks with varying security configurations
- Public Wi-Fi networks with potential adversarial presence
- Mobile networks with carrier-level controls
- Air-gapped or isolated networks in high-security environments

Network conditions directly impact threat exposure, with untrusted networks requiring enhanced protection against network-level attacks (MITM, eavesdropping, traffic manipulation).

Hardware Capabilities: The operational environment includes diverse hardware with varying security features:

- Devices with Trusted Platform Modules (TPM) or secure enclaves
- Systems with hardware-enforced memory protection (DEP, ASLR)
- Devices with biometric authentication capabilities
- Systems with varying performance characteristics affecting security feature feasibility

4.7.2 Physical Environment

Physical Security Controls: Browser deployment contexts vary significantly in physical security:

- **Controlled Environments**: Corporate offices, data centers, and secure facilities with physical access controls, surveillance, and authorized personnel only
- **Semi-Controlled Environments**: Home offices, shared workspaces with moderate physical security
- **Uncontrolled Environments**: Public spaces, cafes, transit where device theft, shoulder surfing, and physical tampering are realistic threats
- **Hostile Environments**: Border crossings, adversarial jurisdictions where targeted physical attacks or device seizure may occur

Device Ownership and Control:

- **Corporate-Owned Devices**: Centrally managed with enforced policies, monitoring, and remote management capabilities
- **Bring Your Own Device (BYOD)**: Personal devices used for work purposes with limited organizational control
- **Shared Devices**: Multi-user systems (kiosks, family computers) where isolation between users is required
- **Personal Devices**: Individually owned and managed with user-determined security configurations

4.7.3 Organizational Environment

Governance and Compliance: Organizations deploying browsers may be subject to:

- Regulatory requirements (GDPR, EU Data Act, sector-specific regulations)
- Industry standards and certification requirements
- Internal security policies and acceptable use policies
- Contractual obligations to customers or partners

Security Maturity: Organizations vary in security capabilities:

- **Advanced**: Dedicated security teams, SIEM integration, threat intelligence, security orchestration
- **Intermediate**: Basic security tools, patch management, logging and monitoring

- **Basic**: Default configurations, reactive security posture, limited security resources

User Training and Awareness: The effectiveness of browser security controls depends on user security awareness:

- Ability to recognize phishing attempts and malicious websites
- Understanding of permission prompts and security indicators
- Adherence to security policies and best practices
- Reporting of security incidents

4.7.4 Threat Environment

The operational environment includes varying threat actor capabilities and motivations:

Opportunistic Threats: Automated attacks, commodity malware, and broad-spectrum phishing affecting general internet users

Targeted Threats: Attacks directed at specific organizations, industries, or individuals including:

- Corporate espionage and intellectual property theft
- Financial fraud and account takeover
- Nation-state surveillance and intelligence gathering
- Ransomware and extortion campaigns

Insider Threats: Risks from authorized users with legitimate access who may act maliciously or negligently

Supply Chain Threats: Compromise of browser components, extensions, or integrated services during development, distribution, or update processes

4.7.5 Lifecycle Environment

Development and Testing: Browsers used in development environments encounter untrusted code, pre-release features, and non-standard configurations that increase risk exposure.

Production Deployment: Operational browsers serving end-users with expected availability, performance, and security requirements.

Decommissioning: End-of-life browsers that may still contain sensitive data requiring secure disposal or migration processes.

Update and Maintenance: Browsers require ongoing updates that shall be delivered securely, tested for stability, and deployed without service interruption.

4.8 Users

Browser users encompass a diverse population with varying technical expertise, security awareness, and usage patterns. Understanding user characteristics is essential for designing effective security controls that users can understand and properly utilize.

4.8.1 User Categories

General Consumers: The largest user population, including:

- Minimal to moderate technical expertise
- Primary activities: web browsing, social media, shopping, entertainment, personal email
- Security awareness varies widely; may not recognize threats
- Expect seamless user experience; security friction reduces adoption
- Limited ability to troubleshoot security issues independently

Professional Users: Individuals using browsers for work-related activities:

- Moderate technical expertise with domain-specific knowledge
- Activities: SaaS applications, web-based productivity tools, professional research
- Subject to organizational security policies
- Generally higher security awareness due to training
- Balance productivity needs with security requirements

Developers and Technical Users: Advanced users with deep technical knowledge:

- High technical expertise; understand browser internals and web standards
- Activities: web development, testing, debugging, performance analysis
- Require access to developer tools and advanced features
- Higher tolerance for security friction; may disable controls intentionally
- Capable of understanding and responding to complex security warnings

Enterprise Administrators: IT personnel managing browser deployments:

- High technical expertise in systems administration
- Responsible for configuring, deploying, and managing browsers at scale
- Implement group policies and security configurations
- Monitor browser usage and security events
- Balance security requirements with user productivity

Children and Protected Users: Vulnerable populations requiring enhanced protection:

- Limited technical expertise and security awareness
- Susceptible to social engineering and inappropriate content
- Require parental controls and content filtering
- May not understand consequences of security decisions
- Need simplified security interfaces with reduced decision-making burden

4.8.2 User Behavior Patterns

Security Decision-Making: Users exhibit common patterns in security decisions:

- **Alert Fatigue**: Repeated security prompts lead to automatic dismissal without reading
- **Optimism Bias**: Underestimation of personal risk ("it won't happen to me")
- **Immediate Gratification**: Preference for immediate access over security delay
- **Trust Misplacement**: Difficulty distinguishing legitimate from malicious content
- **Habituation**: Security behaviors become routine without conscious evaluation

Permission Granting: When presented with permission requests, users typically:

- Grant permissions to accomplish immediate tasks without evaluating necessity
- Lack understanding of implications of granted permissions
- Rarely review or revoke previously granted permissions
- May not notice when permissions are being actively used

Update Behavior: User approaches to browser updates vary:

- Some users apply updates immediately when prompted
- Others defer updates indefinitely to avoid interruption
- Many users unaware of update status or importance
- Corporate environments often enforce centralized update management

4.8.3 User Needs and Expectations

Usability: Users expect browsers to be:

- Intuitive and easy to use without extensive training
- Responsive and fast, with minimal performance impact from security features
- Compatible with their required websites and applications

- Consistent across sessions and devices

Privacy: Users increasingly expect:

- Control over personal data collection and usage
- Protection from tracking by advertisers and third parties
- Transparency about data practices
- Compliance with privacy regulations

Security: While security awareness varies, users generally expect:

- Protection from malware and phishing without constant manual intervention
- Clear warnings when accessing dangerous sites
- Secure handling of passwords and payment information
- Prevention of unauthorized access to sensitive data

Transparency: Users need:

- Understandable explanations of security features and warnings
- Visibility into what data is being collected and shared
- Clear indication of security status (encrypted connections, permissions in use)
- Accessible security settings and controls

4.8.4 User Assistance and Responsibilities

User Responsibilities: Effective browser security requires users to:

- Keep browsers updated to address vulnerabilities
- Exercise caution with extension installation
- Recognize and avoid phishing attempts
- Use strong, unique passwords for web accounts
- Respond appropriately to security warnings
- Report security incidents when encountered

Browser Responsibilities to Users: Browsers shall support users by:

- Providing clear, actionable security warnings
- Using understandable language avoiding technical jargon
- Offering contextual help and explanations
- Defaulting to secure configurations
- Making security controls discoverable and accessible
- Minimizing security decision burden through sound defaults

4.8.5 Accessibility Considerations

Security features shall be accessible to users with disabilities:

- **Visual Impairments**: Security indicators shall be perceivable by screen readers; warnings shall not rely solely on color
- **Cognitive Disabilities**: Security messages shall be simple and clear; complex security decisions should be minimized
- **Motor Impairments**: Security controls shall be operable with alternative input methods
- **Hearing Impairments**: Audio-based security notifications shall have visual alternatives

Accessibility requirements shall not compromise security effectiveness, but security features should not exclude users with disabilities from safe browser usage.

5 Browser-Specific Risk Factors

5.1 Isolation Mechanisms

5.1.1 Domain and Origin Isolation **[DOM]** The manufacturer shall ensure that execution contexts belonging to different origins are securely isolated to prevent unauthorized data access, code execution, or state manipulation across boundaries. Isolation shall include process separation, independent storage and cache spaces, and validation of all cross-origin communication through standardized, browser-mediated mechanisms such as Cross-Origin Resource Sharing (CORS), which allows controlled sharing of resources between origins through validated HTTP headers, and postMessage, which provides a secure message-passing interface between isolated contexts (e.g. frames or windows). Any relaxation of isolation shall be explicitly authorized, documented, and monitored to prevent data leakage or privilege escalation.

Capability: Browser enforces isolation between domains and origins (defined by scheme, host, and port) to protect integrity and confidentiality of data and execution.

Conditions:

- DOM-0: Full isolation: Each origin is strictly separated. No mechanism exists for cross-origin access or relaxation.
- DOM-1: Controlled isolation: Isolation is enforced by default but may be selectively relaxed through standardized, browser-mediated mechanisms (e.g. CORS or postMessage) with explicit validation.
- DOM-2: Configurable isolation: Isolation is enforced by default, but users or administrators can define exceptions via explicit configuration or policy.
- DOM-3: Integrated isolation: Isolation remains in place, but third-party integrations, compatibility modes, or embedded components may introduce controlled exceptions under defined policies.

Threats:

- Cross-site scripting (XSS) attacks
- Cross-site request forgery (CSRF)
- Data exfiltration between origins
- Session hijacking
- Clickjacking

Risk: HIGH - Failure of isolation mechanisms enables immediate compromise of user data across multiple web applications

Requirements:

DOM-0 Requirements (Full isolation)

- **DOM-0-REQ-1**: Browser shall implement process-per-site isolation for all origins → Assessment: DOM-REQ-1
- **DOM-0-REQ-2**: Browser shall enforce Cross-Origin Read Blocking (CORB) for all cross-origin resource loads → Assessment: DOM-REQ-2
- **DOM-0-REQ-3**: Browser shall prevent all cross-origin DOM access without exception → Assessment: DOM-REQ-3
- **DOM-0-REQ-4**: Browser shall isolate localStorage and IndexedDB per origin with no sharing mechanism → Assessment: DOM-REQ-6
- **DOM-0-REQ-5**: Browser shall treat all sandboxed and data: origins as opaque with no relaxation → Assessment: DOM-REQ-8
- **DOM-0-REQ-6**: Browser shall enforce document.domain restrictions without any override mechanism → Assessment: DOM-REQ-12

DOM-1 Requirements (Controlled isolation)

- **DOM-1-REQ-1**: Browser shall implement process-per-site isolation → Assessment: DOM-REQ-1

- **DOM-1-REQ-2**: Browser shall enforce CORB with exceptions only for properly configured CORS headers → Assessment: DOM-REQ-2
- **DOM-1-REQ-3**: Browser shall prevent cross-origin DOM access except via postMessage → Assessment: DOM-REQ-3
- **DOM-1-REQ-4**: Browser shall enforce CORS preflight for all non-simple cross-origin requests → Assessment: DOM-REQ-4
- **DOM-1-REQ-5**: Browser shall enforce SameSite cookie attribute with Lax as default → Assessment: DOM-REQ-5
- **DOM-1-REQ-6**: Browser shall isolate storage per origin → Assessment: DOM-REQ-6
- **DOM-1-REQ-7**: Browser shall support iframe sandbox attribute with granular tokens → Assessment: DOM-REQ-7
- **DOM-1-REQ-8**: Browser shall treat sandboxed and data: origins as opaque → Assessment: DOM-REQ-8
- **DOM-1-REQ-9**: Browser shall restrict document.domain setter by default (require Origin-Agent-Cluster opt-out) → Assessment: DOM-REQ-12

DOM-2 Requirements (Configurable isolation)

- **DOM-2-REQ-1**: Browser shall implement process-per-site isolation → Assessment: DOM-REQ-1
- **DOM-2-REQ-2**: Browser shall enforce CORB → Assessment: DOM-REQ-2
- **DOM-2-REQ-3**: Browser shall prevent cross-origin DOM access except via CORS and postMessage → Assessment: DOM-REQ-3
- **DOM-2-REQ-4**: Browser shall enforce CORS preflight → Assessment: DOM-REQ-4
- **DOM-2-REQ-5**: Browser shall enforce SameSite cookie policies → Assessment: DOM-REQ-5
- **DOM-2-REQ-6**: Browser shall isolate storage per origin → Assessment: DOM-REQ-6
- **DOM-2-REQ-7**: Browser shall support iframe sandboxing → Assessment: DOM-REQ-7
- **DOM-2-REQ-8**: Browser shall support Cross-Origin-Resource-Policy (CORP) header → Assessment: DOM-REQ-9
- **DOM-2-REQ-9**: Browser shall enforce Cross-Origin-Opener-Policy (COOP) → Assessment: DOM-REQ-10
- **DOM-2-REQ-10**: Browser shall enforce Cross-Origin-Embedder-Policy (COEP) → Assessment: DOM-REQ-11
- **DOM-2-REQ-11**: Administrators shall be able to configure origin isolation policies via enterprise policy → Assessment: DOM-REQ-13
- **DOM-2-REQ-12**: Browser shall log all policy-based isolation exceptions → Assessment: DOM-REQ-14

DOM-3 Requirements (Integrated isolation)

- **DOM-3-REQ-1**: Browser shall implement baseline process-per-site isolation → Assessment: DOM-REQ-1
- **DOM-3-REQ-2**: Browser shall enforce CORB with documented exceptions for compatibility → Assessment: DOM-REQ-2
- **DOM-3-REQ-3**: Browser shall enforce CORS policies → Assessment: DOM-REQ-4
- **DOM-3-REQ-4**: Browser shall enforce SameSite cookie policies → Assessment: DOM-REQ-5
- **DOM-3-REQ-5**: Browser shall isolate storage per origin → Assessment: DOM-REQ-6
- **DOM-3-REQ-6**: Browser shall support CORP, COOP, and COEP headers → Assessments: DOM-REQ-9, DOM-REQ-10, DOM-REQ-11
- **DOM-3-REQ-7**: Compatibility modes shall not weaken core isolation boundaries → Assessment: DOM-REQ-15
- **DOM-3-REQ-8**: Third-party integrations shall be subject to same origin isolation policies → Assessment: DOM-REQ-16
- **DOM-3-REQ-9**: All isolation exceptions for compatibility shall be documented and logged → Assessment: DOM-REQ-17
- **DOM-3-REQ-10**: Embedded components shall maintain storage isolation from embedding context → Assessment: DOM-REQ-18

5.2 Extension System Security

5.2.1 Third-Party Code Execution [**EXT**] The manufacturer shall implement controls for third-party extensions that can modify browser behavior, considering that certified extensions may still exhibit malicious behavior.

Capability: Browser extension system with APIs for third-party code augmentation

Conditions:

- EXT-0: No extension support
- EXT-1: Curated extension store only
- EXT-2: Curated store with developer mode
- EXT-3: Unrestricted extension installation

Threats:

- Malicious extensions harvesting user data
- Extension-based cryptomining
- Browser fingerprinting
- Privilege escalation through extension APIs
- Supply chain attacks on popular extensions

Risk: HIGH - Extensions operate as semi-trusted code with significant access to browser functionality and user data

Requirements:

EXT-0 Requirements (No extension support)

- **EXT-0-REQ-1**: Browser shall not provide any extension installation or execution capability → Assessment: EXT-REQ-19
- **EXT-0-REQ-2**: Browser shall block all attempts to load extension code → Assessment: EXT-REQ-20
- **EXT-0-REQ-3**: Browser build shall not include extension subsystem components → Assessment: EXT-REQ-21

EXT-1 Requirements (Curated extension store only)

- **EXT-1-REQ-1**: Browser shall implement granular permission model for extensions → Assessment: EXT-REQ-1
- **EXT-1-REQ-2**: Browser shall isolate extension content scripts from page scripts → Assessment: EXT-REQ-2
- **EXT-1-REQ-3**: Browser shall enforce extension API access control based on declared permissions → Assessment: EXT-REQ-3
- **EXT-1-REQ-4**: Browser shall validate extension manifests before installation → Assessment: EXT-REQ-4
- **EXT-1-REQ-5**: Browser shall sandbox extension execution environments → Assessment: EXT-REQ-5
- **EXT-1-REQ-6**: Browser shall isolate extensions from each other → Assessment: EXT-REQ-6
- **EXT-1-REQ-7**: Browser shall validate host permissions against declared patterns → Assessment: EXT-REQ-7
- **EXT-1-REQ-8**: Browser shall enforce Content Security Policy for extensions → Assessment: EXT-REQ-8
- **EXT-1-REQ-9**: Browser shall verify extension update signatures from official store → Assessment: EXT-REQ-10
- **EXT-1-REQ-10**: Browser shall enforce Manifest V3 compliance for new extensions → Assessment: EXT-REQ-13
- **EXT-1-REQ-11**: Browser shall validate extension signatures before installation → Assessment: EXT-REQ-17

- **EXT-1-REQ-12**: Browser shall provide transparent permissions UI during installation → Assessment: EXT-REQ-18
- **EXT-1-REQ-13**: Browser shall only allow installation from official curated store → Assessment: EXT-REQ-22
- **EXT-1-REQ-14**: Extensions shall undergo security review before store publication → Assessment: EXT-REQ-23

EXT-2 Requirements (Curated store with developer mode)

- **EXT-2-REQ-1**: Browser shall implement all EXT-1 requirements for store-installed extensions
- **EXT-2-REQ-2**: Browser shall isolate extension storage per extension → Assessment: EXT-REQ-11
- **EXT-2-REQ-3**: Browser shall restrict background script capabilities → Assessment: EXT-REQ-12
- **EXT-2-REQ-4**: Browser shall enforce WebRequest API security controls → Assessment: EXT-REQ-9
- **EXT-2-REQ-5**: Browser shall monitor extension-controlled web content → Assessment: EXT-REQ-15
- **EXT-2-REQ-6**: Developer mode shall require explicit user activation with security warnings → Assessment: EXT-REQ-24
- **EXT-2-REQ-7**: Developer mode extensions shall display persistent visual indicators → Assessment: EXT-REQ-25
- **EXT-2-REQ-8**: Developer mode shall disable automatic extension updates → Assessment: EXT-REQ-26
- **EXT-2-REQ-9**: Browser shall log all developer mode extension activities → Assessment: EXT-REQ-27
- **EXT-2-REQ-10**: Enterprise policies shall be able to disable developer mode → Assessment: EXT-REQ-28

EXT-3 Requirements (Unrestricted extension installation)

- **EXT-3-REQ-1**: Browser shall implement baseline extension permission model → Assessment: EXT-REQ-1
- **EXT-3-REQ-2**: Browser shall enforce content script isolation → Assessment: EXT-REQ-2
- **EXT-3-REQ-3**: Browser shall control extension API access → Assessment: EXT-REQ-3
- **EXT-3-REQ-4**: Browser shall validate extension manifests → Assessment: EXT-REQ-4
- **EXT-3-REQ-5**: Browser shall sandbox extensions → Assessment: EXT-REQ-5
- **EXT-3-REQ-6**: Browser shall isolate extensions from each other → Assessment: EXT-REQ-6
- **EXT-3-REQ-7**: Browser shall enforce native messaging security controls → Assessment: EXT-REQ-14
- **EXT-3-REQ-8**: Browser shall monitor extension telemetry with privacy protections → Assessment: EXT-REQ-16
- **EXT-3-REQ-9**: Sideloaded extensions shall display prominent security warnings → Assessment: EXT-REQ-29
- **EXT-3-REQ-10**: Browser shall provide user controls to review and revoke extension permissions → Assessment: EXT-REQ-30
- **EXT-3-REQ-11**: Browser shall scan sideloaded extensions for known malware signatures → Assessment: EXT-REQ-31
- **EXT-3-REQ-12**: All extension security events shall be logged for review → Assessment: EXT-REQ-32

5.3 Encryption Implementation

5.3.1 Data Protection Layers **[ENC]** The manufacturer shall implement comprehensive encryption across all data states and communication channels.

Capability: Multi-layer encryption for data in transit, at rest, and during synchronization

Conditions:

- ENC-0: Full encryption with hardware security module support
- ENC-1: Standard encryption with software-based key management
- ENC-2: Selective encryption based on data sensitivity

- ENC-3: Basic encryption with user-optional enhanced protection
- ENC-4: Minimal encryption

Threats:

- Man-in-the-middle attacks
- Data breach of stored credentials
- Synchronization data interception
- Update channel compromise
- Certificate validation bypass

Risk: CRITICAL - Encryption failures expose all user data and communications

Requirements:

ENC-0 Requirements (Full encryption with HSM support)

- **ENC-0-REQ-1**: Browser shall support TLS 1.3 or higher exclusively for all network communications → Assessment: ENC-REQ-1
- **ENC-0-REQ-2**: Browser shall perform complete certificate chain validation including expiry, revocation, and trust anchor verification → Assessment: ENC-REQ-2
- **ENC-0-REQ-3**: Browser shall enforce HTTP Public Key Pinning (HPKP) or alternative certificate pinning mechanisms → Assessment: ENC-REQ-3
- **ENC-0-REQ-4**: Browser shall enforce HTTP Strict Transport Security (HSTS) with preload support → Assessment: ENC-REQ-4
- **ENC-0-REQ-5**: Browser shall block all mixed content without exception → Assessment: ENC-REQ-5
- **ENC-0-REQ-6**: Browser shall enforce Certificate Transparency requirements for all certificates → Assessment: ENC-REQ-6
- **ENC-0-REQ-7**: Browser shall support and validate OCSP stapling responses → Assessment: ENC-REQ-7
- **ENC-0-REQ-8**: Browser shall restrict cipher suites to strong, modern algorithms only → Assessment: ENC-REQ-8
- **ENC-0-REQ-9**: Browser shall enforce perfect forward secrecy for all TLS connections → Assessment: ENC-REQ-9
- **ENC-0-REQ-10**: Browser shall perform real-time certificate revocation checking → Assessment: ENC-REQ-10
- **ENC-0-REQ-11**: Browser shall implement Web Crypto API with full W3C compliance → Assessment: ENC-REQ-11
- **ENC-0-REQ-12**: Browser shall provide cryptographically secure random number generation → Assessment: ENC-REQ-12
- **ENC-0-REQ-13**: Browser shall enforce SubResource Integrity (SRI) validation → Assessment: ENC-REQ-13
- **ENC-0-REQ-14**: Browser shall support Encrypted SNI (ESNI) or Encrypted Client Hello (ECH) → Assessment: ENC-REQ-14
- **ENC-0-REQ-15**: Browser shall display prominent, non-bypassable certificate error UI → Assessment: ENC-REQ-15
- **ENC-0-REQ-16**: Browser shall operate in HTTPS-first mode with automatic upgrade → Assessment: ENC-REQ-16
- **ENC-0-REQ-17**: Browser shall detect and alert on certificate pinning bypass attempts → Assessment: ENC-REQ-17
- **ENC-0-REQ-18**: Browser shall protect against TLS downgrade attacks → Assessment: ENC-REQ-18
- **ENC-0-REQ-19**: Browser shall deprecate and disable legacy cryptographic protocols (SSL, TLS 1.0/1.1) → Assessment: ENC-REQ-19
- **ENC-0-REQ-20**: Browser shall isolate cryptographic keys in hardware security modules when available → Assessment: ENC-REQ-20

- **ENC-0-REQ-21**: Browser shall implement secure certificate store with integrity protection → Assessment: ENC-REQ-21
- **ENC-0-REQ-22**: Browser shall not allow user override of certificate pinning failures → Assessment: ENC-REQ-22
- **ENC-0-REQ-23**: All cryptographic operations shall be performed in hardware-backed secure enclaves when available → Assessment: ENC-REQ-23

ENC-1 Requirements (Standard encryption with software key management)

- **ENC-1-REQ-1**: Browser shall support TLS 1.3 or higher for all network communications → Assessment: ENC-REQ-1
- **ENC-1-REQ-2**: Browser shall perform complete certificate chain validation → Assessment: ENC-REQ-2
- **ENC-1-REQ-3**: Browser shall support certificate pinning mechanisms → Assessment: ENC-REQ-3
- **ENC-1-REQ-4**: Browser shall enforce HSTS to prevent protocol downgrade attacks → Assessment: ENC-REQ-4
- **ENC-1-REQ-5**: Browser shall block mixed content with opt-in for passive content → Assessment: ENC-REQ-5
- **ENC-1-REQ-6**: Browser shall validate Certificate Transparency logs → Assessment: ENC-REQ-6
- **ENC-1-REQ-7**: Browser shall support OCSP stapling → Assessment: ENC-REQ-7
- **ENC-1-REQ-8**: Browser shall restrict cipher suites to industry-standard secure algorithms → Assessment: ENC-REQ-8
- **ENC-1-REQ-9**: Browser shall prefer perfect forward secrecy cipher suites → Assessment: ENC-REQ-9
- **ENC-1-REQ-10**: Browser shall perform certificate revocation checking (OCSP or CRLSets) → Assessment: ENC-REQ-10
- **ENC-1-REQ-11**: Browser shall implement Web Crypto API → Assessment: ENC-REQ-11
- **ENC-1-REQ-12**: Browser shall provide secure random number generation → Assessment: ENC-REQ-12
- **ENC-1-REQ-13**: Browser shall support SubResource Integrity validation → Assessment: ENC-REQ-13
- **ENC-1-REQ-14**: Browser shall display clear certificate error UI with bypass warnings → Assessment: ENC-REQ-15
- **ENC-1-REQ-15**: Browser shall support HTTPS-first mode as user option → Assessment: ENC-REQ-16
- **ENC-1-REQ-16**: Browser shall protect against TLS downgrade attacks → Assessment: ENC-REQ-18
- **ENC-1-REQ-17**: Browser shall disable legacy cryptographic protocols by default → Assessment: ENC-REQ-19
- **ENC-1-REQ-18**: Browser shall isolate cryptographic keys in process-level sandboxes → Assessment: ENC-REQ-20
- **ENC-1-REQ-19**: Browser shall implement secure certificate store → Assessment: ENC-REQ-21

ENC-2 Requirements (Selective encryption)

- **ENC-2-REQ-1**: Browser shall support TLS 1.3 and TLS 1.2 → Assessment: ENC-REQ-1
- **ENC-2-REQ-2**: Browser shall perform certificate chain validation → Assessment: ENC-REQ-2
- **ENC-2-REQ-3**: Browser shall support certificate pinning for critical domains → Assessment: ENC-REQ-3
- **ENC-2-REQ-4**: Browser shall enforce HSTS when declared by server → Assessment: ENC-REQ-4
- **ENC-2-REQ-5**: Browser shall warn about mixed content → Assessment: ENC-REQ-5
- **ENC-2-REQ-6**: Browser shall support Certificate Transparency → Assessment: ENC-REQ-6
- **ENC-2-REQ-7**: Browser shall support OCSP stapling → Assessment: ENC-REQ-7
- **ENC-2-REQ-8**: Browser shall support standard cipher suites → Assessment: ENC-REQ-8
- **ENC-2-REQ-9**: Browser shall support forward secrecy cipher suites → Assessment: ENC-REQ-9
- **ENC-2-REQ-10**: Browser shall perform revocation checking with soft-fail option → Assessment: ENC-REQ-10

- **ENC-2-REQ-11**: Browser shall implement Web Crypto API → Assessment: ENC-REQ-11
- **ENC-2-REQ-12**: Browser shall provide secure random number generation → Assessment: ENC-REQ-12
- **ENC-2-REQ-13**: Browser shall support SubResource Integrity → Assessment: ENC-REQ-13
- **ENC-2-REQ-14**: Browser shall display certificate errors with bypass option → Assessment: ENC-REQ-15
- **ENC-2-REQ-15**: Browser shall provide HTTPS upgrade suggestions → Assessment: ENC-REQ-16
- **ENC-2-REQ-16**: Browser shall maintain secure certificate store → Assessment: ENC-REQ-21
- **ENC-2-REQ-17**: Users shall be able to configure encryption strictness levels → Assessment: ENC-REQ-24
- **ENC-2-REQ-18**: Browser shall provide visual indicators for connection security status → Assessment: ENC-REQ-25

ENC-3 Requirements (Basic encryption with optional enhanced protection)

- **ENC-3-REQ-1**: Browser shall support TLS 1.2 or higher → Assessment: ENC-REQ-1
- **ENC-3-REQ-2**: Browser shall perform basic certificate validation → Assessment: ENC-REQ-2
- **ENC-3-REQ-3**: Browser shall enforce HSTS when enabled → Assessment: ENC-REQ-4
- **ENC-3-REQ-4**: Browser shall notify users of mixed content → Assessment: ENC-REQ-5
- **ENC-3-REQ-5**: Browser shall support common cipher suites → Assessment: ENC-REQ-8
- **ENC-3-REQ-6**: Browser shall support certificate revocation checking → Assessment: ENC-REQ-10
- **ENC-3-REQ-7**: Browser shall implement Web Crypto API → Assessment: ENC-REQ-11
- **ENC-3-REQ-8**: Browser shall provide secure random number generation → Assessment: ENC-REQ-12
- **ENC-3-REQ-9**: Browser shall display certificate errors with clear bypass options → Assessment: ENC-REQ-15
- **ENC-3-REQ-10**: Browser shall support basic certificate store operations → Assessment: ENC-REQ-21
- **ENC-3-REQ-11**: Users shall have full control over encryption settings → Assessment: ENC-REQ-26
- **ENC-3-REQ-12**: Browser shall provide encryption status indicators in UI → Assessment: ENC-REQ-27
- **ENC-3-REQ-13**: Legacy protocol support may be enabled for compatibility with user consent → Assessment: ENC-REQ-28

ENC-4 Requirements (Minimal encryption)

- **ENC-4-REQ-1**: Browser shall support TLS 1.2 → Assessment: ENC-REQ-1
- **ENC-4-REQ-2**: Browser shall perform basic certificate validation → Assessment: ENC-REQ-2
- **ENC-4-REQ-3**: Browser shall display certificate warnings → Assessment: ENC-REQ-15
- **ENC-4-REQ-4**: Browser shall implement Web Crypto API for web applications → Assessment: ENC-REQ-11
- **ENC-4-REQ-5**: Browser shall provide secure random number generation → Assessment: ENC-REQ-12
- **ENC-4-REQ-6**: Users shall have complete control over all encryption policies → Assessment: ENC-REQ-29
- **ENC-4-REQ-7**: Browser shall support legacy protocols when explicitly enabled by user → Assessment: ENC-REQ-30

5.4 Diagnostic and Monitoring Systems

5.4.1 Logging and Crash Reporting **[LOG]** The manufacturer shall balance diagnostic capabilities with privacy protection in logging and monitoring systems.

Capability: Crash dumps, audit trails, activity logs, and enterprise integration capabilities

Conditions:

- LOG-0: No logging or local-only logging
- LOG-1: Opt-in telemetry with anonymization

- LOG-2: Default telemetry with opt-out
- LOG-3: Mandatory telemetry for enterprise management

Threats:

- Information disclosure through logs
- Privacy violations via telemetry
- Log tampering or deletion
- Unauthorized access to diagnostic data
- Correlation attacks using telemetry data

Risk: MEDIUM - Diagnostic systems may leak sensitive information while being necessary for security monitoring

Requirements:

LOG-0 Requirements (No logging or local-only)

- **LOG-0-REQ-1**: Browser shall not transmit any telemetry or diagnostic data to remote servers → Assessment: LOG-REQ-21
- **LOG-0-REQ-2**: Browser shall maintain local security event logs for audit purposes → Assessment: LOG-REQ-1
- **LOG-0-REQ-3**: Local logs shall be stored with integrity protection → Assessment: LOG-REQ-11
- **LOG-0-REQ-4**: Local logs shall implement data minimization principles → Assessment: LOG-REQ-7
- **LOG-0-REQ-5**: Browser shall provide local security dashboard for log review → Assessment: LOG-REQ-13
- **LOG-0-REQ-6**: Local logs shall support forensic export for security analysis → Assessment: LOG-REQ-17
- **LOG-0-REQ-7**: Browser shall enforce strict access controls on local log files → Assessment: LOG-REQ-20
- **LOG-0-REQ-8**: All logging shall be disabled by default with opt-in for local logging → Assessment: LOG-REQ-22
- **LOG-0-REQ-9**: Users shall be able to view and delete all local logs at any time → Assessment: LOG-REQ-23

LOG-1 Requirements (Opt-in telemetry with anonymization)

- **LOG-1-REQ-1**: Browser shall log security events locally → Assessment: LOG-REQ-1
- **LOG-1-REQ-2**: Browser shall log certificate errors and validation failures → Assessment: LOG-REQ-2
- **LOG-1-REQ-3**: Browser shall log extension security events → Assessment: LOG-REQ-3
- **LOG-1-REQ-4**: Browser shall support CSP violation reporting → Assessment: LOG-REQ-4
- **LOG-1-REQ-5**: Browser shall support Network Error Logging (NEL) → Assessment: LOG-REQ-5
- **LOG-1-REQ-6**: Browser shall implement crash reporting with user consent → Assessment: LOG-REQ-6
- **LOG-1-REQ-7**: Browser shall minimize data collection in logs → Assessment: LOG-REQ-7
- **LOG-1-REQ-8**: Browser shall anonymize telemetry data before transmission → Assessment: LOG-REQ-8
- **LOG-1-REQ-9**: Browser shall require explicit user consent for all telemetry → Assessment: LOG-REQ-9
- **LOG-1-REQ-10**: Browser shall transmit logs over secure channels only → Assessment: LOG-REQ-10
- **LOG-1-REQ-11**: Browser shall protect log integrity with cryptographic signatures → Assessment: LOG-REQ-11
- **LOG-1-REQ-12**: Browser shall enforce log retention policies → Assessment: LOG-REQ-12
- **LOG-1-REQ-13**: Browser shall provide security dashboard for log review → Assessment: LOG-REQ-13
- **LOG-1-REQ-14**: Browser shall use privacy-preserving analytics techniques → Assessment: LOG-REQ-18
- **LOG-1-REQ-15**: Browser shall enforce access controls on diagnostic data → Assessment: LOG-REQ-20

- **LOG-1-REQ-16**: Telemetry shall be disabled by default and require explicit opt-in → Assessment: LOG-REQ-24
- **LOG-1-REQ-17**: Users shall have granular control over telemetry categories → Assessment: LOG-REQ-25
- **LOG-1-REQ-18**: Browser shall provide clear documentation of all collected data → Assessment: LOG-REQ-26

LOG-2 Requirements (Default telemetry with opt-out)

- **LOG-2-REQ-1**: Browser shall log security events → Assessment: LOG-REQ-1
- **LOG-2-REQ-2**: Browser shall log certificate errors → Assessment: LOG-REQ-2
- **LOG-2-REQ-3**: Browser shall log extension security events → Assessment: LOG-REQ-3
- **LOG-2-REQ-4**: Browser shall support CSP violation reporting → Assessment: LOG-REQ-4
- **LOG-2-REQ-5**: Browser shall support Network Error Logging → Assessment: LOG-REQ-5
- **LOG-2-REQ-6**: Browser shall implement crash reporting → Assessment: LOG-REQ-6
- **LOG-2-REQ-7**: Browser shall minimize logged data → Assessment: LOG-REQ-7
- **LOG-2-REQ-8**: Browser shall anonymize telemetry data → Assessment: LOG-REQ-8
- **LOG-2-REQ-9**: Browser shall provide clear opt-out mechanism during first run → Assessment: LOG-REQ-9
- **LOG-2-REQ-10**: Browser shall transmit logs securely → Assessment: LOG-REQ-10
- **LOG-2-REQ-11**: Browser shall protect log integrity → Assessment: LOG-REQ-11
- **LOG-2-REQ-12**: Browser shall enforce retention policies → Assessment: LOG-REQ-12
- **LOG-2-REQ-13**: Browser shall provide security dashboard → Assessment: LOG-REQ-13
- **LOG-2-REQ-14**: Browser shall support incident detection → Assessment: LOG-REQ-14
- **LOG-2-REQ-15**: Browser shall maintain complete audit trail → Assessment: LOG-REQ-15
- **LOG-2-REQ-16**: Browser shall use privacy-preserving analytics → Assessment: LOG-REQ-18
- **LOG-2-REQ-17**: Browser shall support compliance logging → Assessment: LOG-REQ-19
- **LOG-2-REQ-18**: Browser shall enforce log access controls → Assessment: LOG-REQ-20
- **LOG-2-REQ-19**: Users shall be able to disable telemetry at any time → Assessment: LOG-REQ-27
- **LOG-2-REQ-20**: Browser shall display telemetry status in settings UI → Assessment: LOG-REQ-28

LOG-3 Requirements (Mandatory telemetry for enterprise)

- **LOG-3-REQ-1**: Browser shall log all security events → Assessment: LOG-REQ-1
- **LOG-3-REQ-2**: Browser shall log certificate errors and security warnings → Assessment: LOG-REQ-2
- **LOG-3-REQ-3**: Browser shall log all extension security events → Assessment: LOG-REQ-3
- **LOG-3-REQ-4**: Browser shall support CSP violation reporting → Assessment: LOG-REQ-4
- **LOG-3-REQ-5**: Browser shall support Network Error Logging → Assessment: LOG-REQ-5
- **LOG-3-REQ-6**: Browser shall implement comprehensive crash reporting → Assessment: LOG-REQ-6
- **LOG-3-REQ-7**: Browser shall log data with minimal redaction for forensics → Assessment: LOG-REQ-7
- **LOG-3-REQ-8**: Browser shall transmit logs securely to enterprise SIEM → Assessment: LOG-REQ-10
- **LOG-3-REQ-9**: Browser shall protect log integrity with cryptographic controls → Assessment: LOG-REQ-11
- **LOG-3-REQ-10**: Browser shall enforce enterprise-defined retention policies → Assessment: LOG-REQ-12
- **LOG-3-REQ-11**: Browser shall provide comprehensive security dashboard → Assessment: LOG-REQ-13
- **LOG-3-REQ-12**: Browser shall support real-time incident detection → Assessment: LOG-REQ-14
- **LOG-3-REQ-13**: Browser shall maintain complete audit trail of all security events → Assessment: LOG-REQ-15
- **LOG-3-REQ-14**: Browser shall support real-time security alerts → Assessment: LOG-REQ-16
- **LOG-3-REQ-15**: Browser shall support forensic log export in standard formats → Assessment: LOG-REQ-17

- **LOG-3-REQ-16**: Browser shall implement compliance logging for regulatory requirements → Assessment: LOG-REQ-19
- **LOG-3-REQ-17**: Browser shall enforce role-based access controls for logs → Assessment: LOG-REQ-20
- **LOG-3-REQ-18**: Enterprise policies shall prevent users from disabling mandatory logging → Assessment: LOG-REQ-29
- **LOG-3-REQ-19**: Browser shall support integration with enterprise monitoring systems → Assessment: LOG-REQ-30
- **LOG-3-REQ-20**: Browser shall provide tamper-evident logging mechanisms → Assessment: LOG-REQ-31

5.5 Update Delivery Mechanisms

5.5.1 Security Update Management **[UPD]** The manufacturer shall implement secure update mechanisms that balance security needs with user autonomy.

Capability: Automatic security updates with user notification and control options

Conditions:

- UPD-0: Forced automatic updates without user control
- UPD-1: Automatic updates with postponement options
- UPD-2: Optional automatic updates (user should enable)
- UPD-3: Manual updates only

Threats:

- Exploitation of unpatched vulnerabilities
- Update channel compromise
- Malicious update injection
- Denial of service through forced updates
- Zero-day exploitation window

Risk: HIGH - Delayed or compromised updates leave browsers vulnerable to known exploits

Requirements:

UPD-0 Requirements (Forced automatic updates)

- **UPD-0-REQ-1**: Browser shall implement automatic update mechanism without user control → Assessment: UPD-REQ-1
- **UPD-0-REQ-2**: Browser shall verify digital signatures on all updates → Assessment: UPD-REQ-2
- **UPD-0-REQ-3**: Browser shall deliver updates exclusively over HTTPS → Assessment: UPD-REQ-3
- **UPD-0-REQ-4**: Browser shall validate update manifest integrity → Assessment: UPD-REQ-4
- **UPD-0-REQ-5**: Browser shall implement rollback protection to prevent downgrade attacks → Assessment: UPD-REQ-5
- **UPD-0-REQ-6**: Browser shall isolate update channels (stable, beta, dev) → Assessment: UPD-REQ-6
- **UPD-0-REQ-7**: Browser shall support component-level updates for security patches → Assessment: UPD-REQ-7
- **UPD-0-REQ-8**: Browser shall support emergency update capability for zero-day threats → Assessment: UPD-REQ-8
- **UPD-0-REQ-9**: Browser shall verify updates before installation → Assessment: UPD-REQ-9
- **UPD-0-REQ-10**: Browser shall implement failure recovery for failed updates → Assessment: UPD-REQ-10
- **UPD-0-REQ-11**: Browser shall log all updates to transparency log → Assessment: UPD-REQ-11
- **UPD-0-REQ-12**: Browser shall validate delta update security → Assessment: UPD-REQ-12
- **UPD-0-REQ-13**: Browser shall authenticate update servers with certificate pinning → Assessment: UPD-REQ-13

- **UPD-0-REQ-14**: Browser shall implement timing jitter to prevent fingerprinting → Assessment: UPD-REQ-14
- **UPD-0-REQ-15**: Browser shall enforce background update installation → Assessment: UPD-REQ-15
- **UPD-0-REQ-16**: Browser shall force critical security updates immediately → Assessment: UPD-REQ-17
- **UPD-0-REQ-17**: Browser shall verify complete update verification chain → Assessment: UPD-REQ-18
- **UPD-0-REQ-18**: Browser shall implement update source pinning → Assessment: UPD-REQ-19
- **UPD-0-REQ-19**: Browser shall verify update integrity with cryptographic hashes → Assessment: UPD-REQ-20
- **UPD-0-REQ-20**: Browser shall support staged rollout for risk mitigation → Assessment: UPD-REQ-21
- **UPD-0-REQ-21**: Browser shall validate update domain authenticity → Assessment: UPD-REQ-22
- **UPD-0-REQ-22**: Browser shall support binary reproducibility verification → Assessment: UPD-REQ-23
- **UPD-0-REQ-23**: Updates shall be applied without user intervention or postponement → Assessment: UPD-REQ-24
- **UPD-0-REQ-24**: Browser shall restart automatically after critical updates when safe → Assessment: UPD-REQ-25

UPD-1 Requirements (Automatic with postponement)

- **UPD-1-REQ-1**: Browser shall implement automatic update mechanism → Assessment: UPD-REQ-1
- **UPD-1-REQ-2**: Browser shall verify update signatures → Assessment: UPD-REQ-2
- **UPD-1-REQ-3**: Browser shall deliver updates over HTTPS only → Assessment: UPD-REQ-3
- **UPD-1-REQ-4**: Browser shall validate update manifest integrity → Assessment: UPD-REQ-4
- **UPD-1-REQ-5**: Browser shall implement rollback protection → Assessment: UPD-REQ-5
- **UPD-1-REQ-6**: Browser shall isolate update channels → Assessment: UPD-REQ-6
- **UPD-1-REQ-7**: Browser shall support component updates → Assessment: UPD-REQ-7
- **UPD-1-REQ-8**: Browser shall support emergency updates → Assessment: UPD-REQ-8
- **UPD-1-REQ-9**: Browser shall verify updates before installation → Assessment: UPD-REQ-9
- **UPD-1-REQ-10**: Browser shall implement update failure recovery → Assessment: UPD-REQ-10
- **UPD-1-REQ-11**: Browser shall log updates to transparency log → Assessment: UPD-REQ-11
- **UPD-1-REQ-12**: Browser shall secure delta updates → Assessment: UPD-REQ-12
- **UPD-1-REQ-13**: Browser shall authenticate update servers → Assessment: UPD-REQ-13
- **UPD-1-REQ-14**: Browser shall implement update timing jitter → Assessment: UPD-REQ-14
- **UPD-1-REQ-15**: Browser shall enforce background updates → Assessment: UPD-REQ-15
- **UPD-1-REQ-16**: Browser shall display clear update notifications → Assessment: UPD-REQ-16
- **UPD-1-REQ-17**: Browser shall force critical updates with limited postponement → Assessment: UPD-REQ-17
- **UPD-1-REQ-18**: Browser shall verify update verification chain → Assessment: UPD-REQ-18
- **UPD-1-REQ-19**: Browser shall implement update source pinning → Assessment: UPD-REQ-19
- **UPD-1-REQ-20**: Browser shall verify update integrity → Assessment: UPD-REQ-20
- **UPD-1-REQ-21**: Browser shall support staged rollouts → Assessment: UPD-REQ-21
- **UPD-1-REQ-22**: Browser shall validate update domains → Assessment: UPD-REQ-22
- **UPD-1-REQ-23**: Users shall be able to postpone non-critical updates for limited time (max 7 days) → Assessment: UPD-REQ-26
- **UPD-1-REQ-24**: Critical security updates shall not be postponable beyond 24 hours → Assessment: UPD-REQ-27
- **UPD-1-REQ-25**: Browser shall notify users of pending updates with severity indication → Assessment: UPD-REQ-28

UPD-2 Requirements (Optional automatic updates)

- **UPD-2-REQ-1**: Browser shall support automatic update mechanism when enabled → Assessment: UPD-REQ-1
- **UPD-2-REQ-2**: Browser shall verify update signatures → Assessment: UPD-REQ-2

- **UPD-2-REQ-3**: Browser shall deliver updates over HTTPS → Assessment: UPD-REQ-3
- **UPD-2-REQ-4**: Browser shall validate update manifest integrity → Assessment: UPD-REQ-4
- **UPD-2-REQ-5**: Browser shall implement rollback protection → Assessment: UPD-REQ-5
- **UPD-2-REQ-6**: Browser shall isolate update channels → Assessment: UPD-REQ-6
- **UPD-2-REQ-7**: Browser shall support component updates → Assessment: UPD-REQ-7
- **UPD-2-REQ-8**: Browser shall verify updates before installation → Assessment: UPD-REQ-9
- **UPD-2-REQ-9**: Browser shall implement update failure recovery → Assessment: UPD-REQ-10
- **UPD-2-REQ-10**: Browser shall authenticate update servers → Assessment: UPD-REQ-13
- **UPD-2-REQ-11**: Browser shall display update notifications → Assessment: UPD-REQ-16
- **UPD-2-REQ-12**: Browser shall verify update integrity → Assessment: UPD-REQ-20
- **UPD-2-REQ-13**: Browser shall validate update domains → Assessment: UPD-REQ-22
- **UPD-2-REQ-14**: Automatic updates shall be disabled by default → Assessment: UPD-REQ-29
- **UPD-2-REQ-15**: Browser shall prominently recommend enabling automatic updates → Assessment: UPD-REQ-30
- **UPD-2-REQ-16**: Browser shall display security warnings when updates are available → Assessment: UPD-REQ-31
- **UPD-2-REQ-17**: Browser shall provide easy mechanism to check for and install updates → Assessment: UPD-REQ-32
- **UPD-2-REQ-18**: Users shall have full control over update timing and installation → Assessment: UPD-REQ-33

UPD-3 Requirements (Manual updates only)

- **UPD-3-REQ-1**: Browser shall verify update signatures when manually triggered → Assessment: UPD-REQ-2
- **UPD-3-REQ-2**: Browser shall deliver updates over HTTPS → Assessment: UPD-REQ-3
- **UPD-3-REQ-3**: Browser shall validate update manifest integrity → Assessment: UPD-REQ-4
- **UPD-3-REQ-4**: Browser shall implement rollback protection → Assessment: UPD-REQ-5
- **UPD-3-REQ-5**: Browser shall verify updates before installation → Assessment: UPD-REQ-9
- **UPD-3-REQ-6**: Browser shall implement update failure recovery → Assessment: UPD-REQ-10
- **UPD-3-REQ-7**: Browser shall authenticate update servers → Assessment: UPD-REQ-13
- **UPD-3-REQ-8**: Browser shall display update availability notifications → Assessment: UPD-REQ-16
- **UPD-3-REQ-9**: Browser shall verify update integrity → Assessment: UPD-REQ-20
- **UPD-3-REQ-10**: Browser shall validate update domains → Assessment: UPD-REQ-22
- **UPD-3-REQ-11**: Browser shall provide manual update check mechanism → Assessment: UPD-REQ-34
- **UPD-3-REQ-12**: Browser shall display security warnings for outdated versions → Assessment: UPD-REQ-35
- **UPD-3-REQ-13**: Browser shall provide clear indication of available security updates → Assessment: UPD-REQ-36
- **UPD-3-REQ-14**: All updates shall require explicit user initiation → Assessment: UPD-REQ-37
- **UPD-3-REQ-15**: Browser shall display update changelog and security impact → Assessment: UPD-REQ-38

5.6 Protocol Handler Security

5.6.1 Custom Protocol Management **[PRO]** The manufacturer shall securely handle various communication protocols beyond standard HTTP/HTTPS.

Capability: Support for custom schemes, WebSocket, WebRTC, and emerging web standards

Conditions:

- PRO-0: HTTP/HTTPS only
- PRO-1: Standard web protocols with strict validation
- PRO-2: Custom protocols with registration system
- PRO-3: Unrestricted protocol handler registration

Threats:

- Protocol confusion attacks
- Scheme hijacking
- Bypass of security controls via custom protocols
- Data leakage through protocol handlers
- Local application exploitation via URL schemes

Risk: MEDIUM - Custom protocols can bypass standard web security controls

Requirements:

PRO-0 Requirements (HTTP/HTTPS only)

- **PRO-0-REQ-1**: Browser shall only support HTTP and HTTPS protocols → Assessment: PRO-REQ-24
- **PRO-0-REQ-2**: Browser shall reject all custom protocol handler registration attempts → Assessment: PRO-REQ-25
- **PRO-0-REQ-3**: Browser shall block access to non-standard URL schemes (file://, data://, javascript://, etc.) → Assessment: PRO-REQ-26
- **PRO-0-REQ-4**: Browser shall not provide registerProtocolHandler() API or equivalent functionality → Assessment: PRO-REQ-27
- **PRO-0-REQ-5**: Browser shall reject navigation to any non-HTTP/HTTPS protocols → Assessment: PRO-REQ-28

PRO-1 Requirements (Standard web protocols with strict validation)

- **PRO-1-REQ-1**: Browser shall validate all custom protocol handler registrations → Assessment: PRO-REQ-1
- **PRO-1-REQ-2**: Browser shall obtain explicit user consent before activating custom protocol handlers → Assessment: PRO-REQ-2
- **PRO-1-REQ-3**: Browser shall enforce protocol allowlists that restrict which custom schemes can be registered → Assessment: PRO-REQ-3
- **PRO-1-REQ-4**: Browser shall prevent scheme hijacking attacks → Assessment: PRO-REQ-4
- **PRO-1-REQ-5**: Browser shall sanitize protocol URL parameters before passing to handlers → Assessment: PRO-REQ-5
- **PRO-1-REQ-6**: Browser shall implement security controls for external protocol handlers → Assessment: PRO-REQ-6
- **PRO-1-REQ-7**: Browser shall provide transparent UI indicating protocol handler registration and invocation → Assessment: PRO-REQ-7
- **PRO-1-REQ-8**: Browser shall prevent protocol downgrade attacks → Assessment: PRO-REQ-8
- **PRO-1-REQ-9**: Browser shall log protocol handler registration, modification, and invocation events → Assessment: PRO-REQ-9
- **PRO-1-REQ-10**: Browser shall support web+custom scheme conventions → Assessment: PRO-REQ-10
- **PRO-1-REQ-11**: Browser shall enforce handler capability restrictions → Assessment: PRO-REQ-13
- **PRO-1-REQ-12**: Browser shall provide protocol handler revocation mechanisms → Assessment: PRO-REQ-14
- **PRO-1-REQ-13**: Browser shall enforce cross-origin protocol restrictions → Assessment: PRO-REQ-15

PRO-2 Requirements (Custom protocols with registration system)

- **PRO-2-REQ-1**: Browser shall implement all PRO-1 requirements
- **PRO-2-REQ-2**: Browser shall validate protocol handler persistence across sessions → Assessment: PRO-REQ-11
- **PRO-2-REQ-3**: Browser shall mitigate protocol confusion attacks → Assessment: PRO-REQ-12

- **PRO-2-REQ-4**: Browser shall validate protocol handler manifests → Assessment: PRO-REQ-16
- **PRO-2-REQ-5**: Browser shall integrate protocol handlers with Content Security Policy → Assessment: PRO-REQ-20
- **PRO-2-REQ-6**: Browser shall maintain audit trail for handler registrations → Assessment: PRO-REQ-21
- **PRO-2-REQ-7**: Browser shall enforce secure handler update mechanisms → Assessment: PRO-REQ-22
- **PRO-2-REQ-8**: Browser shall enforce handler isolation between origins → Assessment: PRO-REQ-23
- **PRO-2-REQ-9**: Browser shall support Intent URL security on Android platforms → Assessment: PRO-REQ-17
- **PRO-2-REQ-10**: Browser shall support Universal Links security on iOS platforms → Assessment: PRO-REQ-18
- **PRO-2-REQ-11**: Browser shall validate mobile deep linking security → Assessment: PRO-REQ-19
- **PRO-2-REQ-12**: Enterprise administrators shall be able to configure protocol handler allowlists and blocklists → Assessment: PRO-REQ-29

PRO-3 Requirements (Unrestricted protocol registration)

- **PRO-3-REQ-1**: Browser shall implement baseline protocol handler validation → Assessment: PRO-REQ-1
- **PRO-3-REQ-2**: Browser shall obtain user consent for protocol handler activation → Assessment: PRO-REQ-2
- **PRO-3-REQ-3**: Browser shall sanitize protocol parameters → Assessment: PRO-REQ-5
- **PRO-3-REQ-4**: Browser shall implement external handler security controls → Assessment: PRO-REQ-6
- **PRO-3-REQ-5**: Browser shall provide handler management UI → Assessment: PRO-REQ-7
- **PRO-3-REQ-6**: Browser shall log protocol handler security events → Assessment: PRO-REQ-9
- **PRO-3-REQ-7**: Browser shall allow registration of custom schemes without web+ prefix → Assessment: PRO-REQ-30
- **PRO-3-REQ-8**: Browser shall display security warnings for non-standard protocol handlers → Assessment: PRO-REQ-31
- **PRO-3-REQ-9**: Browser shall provide user-accessible handler revocation controls → Assessment: PRO-REQ-14
- **PRO-3-REQ-10**: Users shall be able to review all registered protocol handlers in browser settings → Assessment: PRO-REQ-32
- **PRO-3-REQ-11**: Browser shall scan custom handlers for known security vulnerabilities → Assessment: PRO-REQ-33
- **PRO-3-REQ-12**: All protocol handler security exceptions shall be logged and auditable → Assessment: PRO-REQ-34

5.7 Core Component Security

5.7.1 System Resource Access **[SYS]** The manufacturer shall implement secure boundaries between web content and system resources through Hardware Abstraction Layers and API mediation.

Capability: HAL implementation, PWA support, accessibility features, and local storage mechanisms

Conditions:

- SYS-0: Fully sandboxed with no system access
- SYS-1: Limited system access with strict permissions
- SYS-2: Extended system access for PWAs
- SYS-3: Native-equivalent system access

Threats:

- Sandbox escape attacks
- Unauthorized file system access

- Hardware fingerprinting
- Resource exhaustion attacks
- Accessibility feature abuse for screen reading

Risk: CRITICAL - System access breaches can compromise the entire host environment

Requirements:

SYS-0 Requirements (Fully sandboxed, no system access)

- **SYS-0-REQ-1**: Browser shall enforce process-level sandboxing for all web content → Assessment: SYS-REQ-1
- **SYS-0-REQ-2**: Browser shall isolate renderer processes from each other and browser core → Assessment: SYS-REQ-2
- **SYS-0-REQ-3**: Browser shall isolate GPU rendering in separate sandboxed process → Assessment: SYS-REQ-3
- **SYS-0-REQ-4**: Browser shall isolate network operations in separate process or service → Assessment: SYS-REQ-4
- **SYS-0-REQ-5**: Browser shall block all filesystem access from web content → Assessment: SYS-REQ-5
- **SYS-0-REQ-6**: Browser shall block all device API access (no camera, microphone, location, etc.) → Assessment: SYS-REQ-33
- **SYS-0-REQ-7**: Browser shall enforce strict hardware resource limits → Assessment: SYS-REQ-20
- **SYS-0-REQ-8**: Browser shall enforce memory isolation between processes → Assessment: SYS-REQ-21
- **SYS-0-REQ-9**: Browser shall implement sandbox escape prevention mechanisms → Assessment: SYS-REQ-26
- **SYS-0-REQ-10**: Browser shall implement Spectre/Meltdown mitigations → Assessment: SYS-REQ-27
- **SYS-0-REQ-11**: Browser shall implement side-channel attack mitigations → Assessment: SYS-REQ-28
- **SYS-0-REQ-12**: Browser shall not expose any native messaging interfaces → Assessment: SYS-REQ-34
- **SYS-0-REQ-13**: Browser shall block all host OS integration features → Assessment: SYS-REQ-35

SYS-1 Requirements (Limited system access with strict permissions)

- **SYS-1-REQ-1**: Browser shall enforce process-level sandboxing → Assessment: SYS-REQ-1
- **SYS-1-REQ-2**: Browser shall isolate renderer processes → Assessment: SYS-REQ-2
- **SYS-1-REQ-3**: Browser shall isolate GPU process → Assessment: SYS-REQ-3
- **SYS-1-REQ-4**: Browser shall isolate network service → Assessment: SYS-REQ-4
- **SYS-1-REQ-5**: Browser shall enforce strict filesystem access controls → Assessment: SYS-REQ-5
- **SYS-1-REQ-6**: Browser shall implement permission controls for device hardware APIs → Assessment: SYS-REQ-6
- **SYS-1-REQ-7**: Browser shall enforce geolocation permission requirements → Assessment: SYS-REQ-8
- **SYS-1-REQ-8**: Browser shall enforce camera/microphone access controls with visible indicators → Assessment: SYS-REQ-9
- **SYS-1-REQ-9**: Browser shall restrict clipboard access to require user interaction → Assessment: SYS-REQ-10
- **SYS-1-REQ-10**: Browser shall enforce notification permission management → Assessment: SYS-REQ-11
- **SYS-1-REQ-11**: Browser shall restrict Sensor API access with permissions → Assessment: SYS-REQ-18
- **SYS-1-REQ-12**: Browser shall restrict Battery Status API to prevent fingerprinting → Assessment: SYS-REQ-19
- **SYS-1-REQ-13**: Browser shall enforce hardware resource limits → Assessment: SYS-REQ-20
- **SYS-1-REQ-14**: Browser shall enforce memory isolation → Assessment: SYS-REQ-21
- **SYS-1-REQ-15**: Browser shall implement CPU resource quotas → Assessment: SYS-REQ-22
- **SYS-1-REQ-16**: Browser shall enforce network bandwidth limits → Assessment: SYS-REQ-23
- **SYS-1-REQ-17**: Browser shall enforce storage quota limits → Assessment: SYS-REQ-24
- **SYS-1-REQ-18**: Browser shall implement process priority management → Assessment: SYS-REQ-25

- **SYS-1-REQ-19**: Browser shall prevent sandbox escapes → Assessment: SYS-REQ-26
- **SYS-1-REQ-20**: Browser shall implement speculative execution mitigations → Assessment: SYS-REQ-27
- **SYS-1-REQ-21**: Browser shall implement side-channel mitigations → Assessment: SYS-REQ-28
- **SYS-1-REQ-22**: Browser shall implement accessibility API security controls → Assessment: SYS-REQ-30

SYS-2 Requirements (Extended system access for PWAs)

- **SYS-2-REQ-1**: Browser shall implement all SYS-1 requirements
- **SYS-2-REQ-2**: Browser shall enforce equivalent permission controls for PWAs → Assessment: SYS-REQ-7
- **SYS-2-REQ-3**: Browser shall implement USB device access security → Assessment: SYS-REQ-12
- **SYS-2-REQ-4**: Browser shall enforce Bluetooth permission controls → Assessment: SYS-REQ-13
- **SYS-2-REQ-5**: Browser shall implement File System Access API security → Assessment: SYS-REQ-14
- **SYS-2-REQ-6**: Browser shall enforce WebUSB security controls → Assessment: SYS-REQ-15
- **SYS-2-REQ-7**: Browser shall enforce WebBluetooth security → Assessment: SYS-REQ-16
- **SYS-2-REQ-8**: Browser shall implement WebNFC permission management → Assessment: SYS-REQ-17
- **SYS-2-REQ-9**: Browser shall implement hardware token security → Assessment: SYS-REQ-29
- **SYS-2-REQ-10**: Browser shall enforce restricted native messaging with security controls → Assessment: SYS-REQ-31
- **SYS-2-REQ-11**: PWA permissions shall not exceed web context permissions → Assessment: SYS-REQ-36
- **SYS-2-REQ-12**: PWA installation shall not auto-grant extended permissions → Assessment: SYS-REQ-37
- **SYS-2-REQ-13**: PWA uninstallation shall revoke all granted permissions → Assessment: SYS-REQ-38
- **SYS-2-REQ-14**: Browser shall maintain audit log of all PWA permission grants and revocations → Assessment: SYS-REQ-39
- **SYS-2-REQ-15**: Enterprise administrators shall be able to configure PWA permission policies → Assessment: SYS-REQ-40

SYS-3 Requirements (Native-equivalent system access)

- **SYS-3-REQ-1**: Browser shall implement baseline sandboxing for renderer processes → Assessment: SYS-REQ-1
- **SYS-3-REQ-2**: Browser shall enforce process isolation → Assessment: SYS-REQ-2
- **SYS-3-REQ-3**: Browser shall implement baseline permission controls for device APIs → Assessment: SYS-REQ-6
- **SYS-3-REQ-4**: Browser shall enforce filesystem access controls → Assessment: SYS-REQ-5
- **SYS-3-REQ-5**: Browser shall implement resource limits to prevent exhaustion → Assessment: SYS-REQ-20
- **SYS-3-REQ-6**: Browser shall implement memory isolation → Assessment: SYS-REQ-21
- **SYS-3-REQ-7**: Browser shall implement CPU quotas → Assessment: SYS-REQ-22
- **SYS-3-REQ-8**: Browser shall enforce storage quotas → Assessment: SYS-REQ-24
- **SYS-3-REQ-9**: Browser shall implement speculative execution mitigations → Assessment: SYS-REQ-27
- **SYS-3-REQ-10**: Browser shall implement side-channel mitigations → Assessment: SYS-REQ-28
- **SYS-3-REQ-11**: Browser shall enforce host OS integration security → Assessment: SYS-REQ-32
- **SYS-3-REQ-12**: Browser shall allow native-equivalent API access with user consent → Assessment: SYS-REQ-41
- **SYS-3-REQ-13**: Browser shall provide transparent UI for all native integration features → Assessment: SYS-REQ-42
- **SYS-3-REQ-14**: Browser shall display security warnings for privileged API access → Assessment: SYS-REQ-43

- **SYS-3-REQ-15**: Browser shall log all extended system access for security auditing → Assessment: SYS-REQ-44
- **SYS-3-REQ-16**: Users shall be able to review and revoke all system permissions → Assessment: SYS-REQ-45
- **SYS-3-REQ-17**: Enterprise administrators shall be able to restrict native-equivalent features → Assessment: SYS-REQ-46
- **SYS-3-REQ-18**: All native integration exceptions shall be documented and auditable → Assessment: SYS-REQ-47

5.8 Embedded Browser Security

5.8.1 Overview

Embedded browsers (WebView components, browser engines integrated into native applications) present unique security challenges distinct from standalone browsers. While standalone browsers operate as independent applications with their own security boundaries, embedded browsers exist within a host application context where they navigate the security boundary between web content and the host application.

Key Security Challenges:

1. **Host-Code Injection**: Malicious web content may attempt to inject code into or manipulate the host application through JavaScript bridge interfaces, custom URL scheme handlers, or exploitation of WebView API vulnerabilities.
2. **User Data Exfiltration**: Web content loaded in an embedded browser may access sensitive data from the host application through insecure bridge configurations, shared storage, or insufficient isolation between web and native contexts.
3. **Insufficient Isolation**: Unlike standalone browsers where all web content is untrusted, embedded browsers often establish trust relationships with certain content sources while maintaining security boundaries, creating complex policy enforcement challenges.

Risk Classification: CRITICAL - Vulnerabilities in embedded browser security can lead to complete compromise of the host application and exfiltration of all user data accessible to the host.

Applicable Use Cases: UC-B08 (Embedded Browser Component), UC-B09 (WebView Component in Native Applications), and aspects of UC-B10 (Adapted Browser with Modified Features) when the adaptation includes native component integration.

References:

- OWASP Mobile Security Testing Guide - WebViews: https://mas.owasp.org/MASTG/tests/android/MASVS-PLATFORM/MASTG-TEST-0028/
- Android WebView Security Best Practices: https://developer.android.com/develop/ui/views/layout/webapps/webview
- iOS WKWebView Security: https://developer.apple.com/documentation/webkit/wkwebview
- Chromium Embedded Framework Security: https://bitbucket.org/chromiumembedded/cef/wiki/GeneralUsage#markdow header-security
- Electron Security Best Practices: https://www.electronjs.org/docs/latest/tutorial/security

5.8.2 Host Application Boundary Security

[**EMB**] The manufacturer shall implement secure isolation boundaries between embedded browser content and host application code, data, and resources.

Capability: JavaScript bridge security, native API exposure control, host data protection, and context isolation

Conditions:

- **EMB-0**: No JavaScript bridge or native API exposure (isolated WebView)

- **EMB-1**: Limited JavaScript bridge with explicit allowlist of safe APIs
- **EMB-2**: Extended JavaScript bridge with bidirectional communication
- **EMB-3**: Full integration with access to native capabilities and host data

Threats:

- JavaScript injection into host application context
- Unauthorized access to native APIs through bridge exploitation
- Code execution in host process via WebView vulnerabilities
- Cross-context data leakage between web and native layers
- Exploitation of insecure JavaScript bridge configurations
- Bypass of content security policies via native bridges
- Host application credential theft via bridge access
- Native code injection via crafted web content

Risk: CRITICAL - Compromise of the host-browser boundary can lead to complete application takeover and data exfiltration

Requirements:

EMB-0 Requirements (No JavaScript bridge or native API exposure)

- **EMB-0-REQ-1**: Embedded browser shall implement complete context isolation → Assessment: EMB-REQ-4
- **EMB-0-REQ-2**: Embedded browser shall isolate all storage from host application → Assessment: EMB-REQ-12
- **EMB-0-REQ-3**: Embedded browser shall enforce CSP for all web content → Assessment: EMB-REQ-13
- **EMB-0-REQ-4**: Embedded browser shall prevent all web content access to host application objects → Assessment: EMB-REQ-33
- **EMB-0-REQ-5**: No JavaScript bridge or native API exposure mechanism shall exist → Assessment: EMB-REQ-34
- **EMB-0-REQ-6**: Embedded browser shall treat all loaded content as completely untrusted → Assessment: EMB-REQ-35
- **EMB-0-REQ-7**: Host application credentials shall be completely isolated from web context → Assessment: EMB-REQ-8

EMB-1 Requirements (Limited JavaScript bridge with explicit allowlist)

- **EMB-1-REQ-1**: JavaScript bridge shall implement explicit API allowlists with per-API access controls → Assessment: EMB-REQ-1
- **EMB-1-REQ-2**: All bridge data shall be validated, sanitized, and type-checked on native side → Assessment: EMB-REQ-2
- **EMB-1-REQ-3**: JavaScript bridge communications shall be logged → Assessment: EMB-REQ-3
- **EMB-1-REQ-4**: Embedded browser shall implement context isolation → Assessment: EMB-REQ-4
- **EMB-1-REQ-5**: Sensitive native APIs shall require explicit user consent → Assessment: EMB-REQ-5
- **EMB-1-REQ-6**: System-level APIs shall not be exposed without additional security controls → Assessment: EMB-REQ-6
- **EMB-1-REQ-7**: JavaScript bridge configuration shall be immutable after initialization → Assessment: EMB-REQ-7
- **EMB-1-REQ-8**: Web content shall not access host credentials, tokens, or keys → Assessment: EMB-REQ-8
- **EMB-1-REQ-9**: Bridge implementations shall be reviewed for injection vulnerabilities → Assessment: EMB-REQ-9
- **EMB-1-REQ-10**: Host shall implement rate limiting on bridge API calls → Assessment: EMB-REQ-10
- **EMB-1-REQ-11**: Bridge shall support granular capability-based permissions → Assessment: EMB-REQ-11

- **EMB-1-REQ-12**: Embedded browser shall isolate storage from host → Assessment: EMB-REQ-12
- **EMB-1-REQ-13**: Host shall enforce CSP for all embedded content → Assessment: EMB-REQ-13
- **EMB-1-REQ-14**: Web content shall not trigger misleading native UI → Assessment: EMB-REQ-15
- **EMB-1-REQ-15**: Host shall implement allowlists rather than denylists for bridge APIs → Assessment: EMB-REQ-16
- **EMB-1-REQ-16**: Only explicitly allowlisted APIs shall be accessible from web content → Assessment: EMB-REQ-36
- **EMB-1-REQ-17**: Bridge API allowlist shall be minimal and documented → Assessment: EMB-REQ-37

EMB-2 Requirements (Extended JavaScript bridge with bidirectional communication)

- **EMB-2-REQ-1**: All EMB-1 requirements shall be implemented
- **EMB-2-REQ-2**: Bridge communications crossing process boundaries shall be encrypted → Assessment: EMB-REQ-14
- **EMB-2-REQ-3**: Bridge shall implement granular permissions → Assessment: EMB-REQ-11
- **EMB-2-REQ-4**: Host shall implement comprehensive rate limiting → Assessment: EMB-REQ-10
- **EMB-2-REQ-5**: All bridge communications shall be logged with full audit trails → Assessment: EMB-REQ-3
- **EMB-2-REQ-6**: Bidirectional bridge calls shall maintain same security controls in both directions → Assessment: EMB-REQ-38
- **EMB-2-REQ-7**: Web content callback handlers shall be validated before invocation → Assessment: EMB-REQ-39
- **EMB-2-REQ-8**: Bridge shall implement message queuing with integrity protection → Assessment: EMB-REQ-40
- **EMB-2-REQ-9**: Host shall monitor bridge traffic for anomalies → Assessment: EMB-REQ-41
- **EMB-2-REQ-10**: Enterprise administrators shall be able to configure bridge API policies → Assessment: EMB-REQ-42

EMB-3 Requirements (Full integration with native capabilities)

- **EMB-3-REQ-1**: Baseline EMB-1 security controls shall be maintained
- **EMB-3-REQ-2**: Bridge shall implement comprehensive input validation → Assessment: EMB-REQ-2
- **EMB-3-REQ-3**: Bridge shall log all operations with security context → Assessment: EMB-REQ-3
- **EMB-3-REQ-4**: Sensitive native operations shall require user consent → Assessment: EMB-REQ-5
- **EMB-3-REQ-5**: System-level APIs shall have strict additional controls → Assessment: EMB-REQ-6
- **EMB-3-REQ-6**: Bridge implementations shall undergo security review → Assessment: EMB-REQ-9
- **EMB-3-REQ-7**: Rate limiting shall prevent bridge API abuse → Assessment: EMB-REQ-10
- **EMB-3-REQ-8**: Full integration shall not bypass core security boundaries → Assessment: EMB-REQ-43
- **EMB-3-REQ-9**: User shall be informed of all native capabilities granted to web content → Assessment: EMB-REQ-44
- **EMB-3-REQ-10**: User shall be able to review and revoke native API access → Assessment: EMB-REQ-45
- **EMB-3-REQ-11**: All native integrations shall be documented and auditable → Assessment: EMB-REQ-46
- **EMB-3-REQ-12**: Enterprise policies shall be able to restrict native integration scope → Assessment: EMB-REQ-47

5.8.3 Content Source Trust Management

[**EMB**] The manufacturer shall implement mechanisms to establish and enforce trust relationships between embedded browser content sources and the host application.

Capability: Content source validation, certificate pinning for embedded content, subresource integrity, and trust boundary enforcement

Conditions:

- **EMB-0**: All content treated as untrusted (public internet)
- **EMB-1**: Trusted domains with certificate validation
- **EMB-2**: Certificate pinning for specific trusted origins
- **EMB-3**: Local/bundled content with cryptographic verification

Threats:

- Man-in-the-middle attacks against trusted content sources
- Loading of malicious content from compromised trusted domains
- Bypass of trust boundaries through redirect chains
- Subresource substitution attacks
- DNS hijacking of embedded content sources
- Certificate authority compromise affecting trusted origins
- Mixed content attacks (trusted page loading untrusted resources)
- Cache poisoning affecting embedded content

Risk: CRITICAL - Compromise of trusted content sources can lead to injection of malicious code with elevated privileges

Requirements:

EMB-0 Requirements (All content treated as untrusted - public internet)

- **EMB-0-REQ-1**: All content shall be treated as completely untrusted
- **EMB-0-REQ-2**: Embedded browser shall validate SSL/TLS certificates for all remote content → Assessment: EMB-REQ-17
- **EMB-0-REQ-3**: Embedded browser shall prevent all mixed content → Assessment: EMB-REQ-21
- **EMB-0-REQ-4**: Certificate validation failures shall block content loading → Assessment: EMB-REQ-26
- **EMB-0-REQ-5**: Network security configuration shall prevent cleartext traffic → Assessment: EMB-REQ-27
- **EMB-0-REQ-6**: Trust boundary violations shall trigger security events → Assessment: EMB-REQ-32
- **EMB-0-REQ-7**: No content origin shall have privileged access
- **EMB-0-REQ-8**: All CSP policies shall be strictly enforced without exceptions

EMB-1 Requirements (Trusted domains with certificate validation)

- **EMB-1-REQ-1**: Embedded browser shall validate SSL/TLS certificates for all remote content → Assessment: EMB-REQ-17
- **EMB-1-REQ-2**: Host shall implement allowlist of trusted content origins → Assessment: EMB-REQ-18
- **EMB-1-REQ-3**: Embedded browser shall prevent mixed content → Assessment: EMB-REQ-21
- **EMB-1-REQ-4**: Trust decisions shall be logged with full context → Assessment: EMB-REQ-22
- **EMB-1-REQ-5**: Host shall implement redirect chain validation → Assessment: EMB-REQ-24
- **EMB-1-REQ-6**: Embedded browser shall enforce HSTS for trusted origins → Assessment: EMB-REQ-25
- **EMB-1-REQ-7**: Certificate failures shall trigger immediate notification and blocking → Assessment: EMB-REQ-26
- **EMB-1-REQ-8**: Network security config shall prevent cleartext to trusted domains → Assessment: EMB-REQ-27
- **EMB-1-REQ-9**: Trusted content shall not load untrusted third-party content without CSP → Assessment: EMB-REQ-28
- **EMB-1-REQ-10**: Trust boundary violations shall trigger security events → Assessment: EMB-REQ-32
- **EMB-1-REQ-11**: Only explicitly allowlisted origins shall be considered trusted
- **EMB-1-REQ-12**: Trust allowlist shall be immutable by web content

EMB-2 Requirements (Certificate pinning for specific trusted origins)

- **EMB-2-REQ-1**: All EMB-1 requirements shall be implemented
- **EMB-2-REQ-2**: Embedded browser shall implement certificate pinning for critical origins → Assessment: EMB-REQ-17
- **EMB-2-REQ-3**: Embedded browser shall enforce SRI for external scripts from trusted content → Assessment: EMB-REQ-19
- **EMB-2-REQ-4**: Certificate pinning shall include backup pins and rotation mechanisms → Assessment: EMB-REQ-20
- **EMB-2-REQ-5**: Trust policies shall be configurable per browser instance → Assessment: EMB-REQ-29
- **EMB-2-REQ-6**: Embedded browser shall implement certificate transparency verification → Assessment: EMB-REQ-30
- **EMB-2-REQ-7**: Host shall detect and prevent DNS rebinding attacks → Assessment: EMB-REQ-31
- **EMB-2-REQ-8**: Pin configuration shall be immutable after initialization
- **EMB-2-REQ-9**: Pinning violations shall immediately block content loading
- **EMB-2-REQ-10**: Pin rotation procedures shall be documented and tested

EMB-3 Requirements (Local/bundled content with cryptographic verification)

- **EMB-3-REQ-1**: Baseline EMB-1 certificate validation shall apply to all remote content
- **EMB-3-REQ-2**: Embedded browser shall verify cryptographic signatures for local/bundled content → Assessment: EMB-REQ-23
- **EMB-3-REQ-3**: Embedded browser shall enforce SRI for all external scripts → Assessment: EMB-REQ-19
- **EMB-3-REQ-4**: Certificate pinning shall be enforced for remote trusted origins → Assessment: EMB-REQ-17, EMB-REQ-20
- **EMB-3-REQ-5**: Trust decisions shall be logged comprehensively → Assessment: EMB-REQ-22
- **EMB-3-REQ-6**: Trust policies shall be configurable per instance → Assessment: EMB-REQ-29
- **EMB-3-REQ-7**: DNS rebinding prevention shall be enforced → Assessment: EMB-REQ-31
- **EMB-3-REQ-8**: Trust boundary violations shall trigger detailed security events → Assessment: EMB-REQ-32
- **EMB-3-REQ-9**: Local content signature verification shall use secure algorithms (RSA-2048+, ECDSA P-256+)
- **EMB-3-REQ-10**: Modified local content shall fail signature verification and be rejected
- **EMB-3-REQ-11**: Signing keys for local content shall be protected from extraction
- **EMB-3-REQ-12**: Hybrid deployments (local + remote) shall maintain strictest security controls for each content type

References:

- OWASP Mobile Top 10 - M1: Improper Platform Usage: https://owasp.org/www-project-mobile-top-10/
- CWE-749: Exposed Dangerous Method or Function: https://cwe.mitre.org/data/definitions/749.html
- CWE-940: Improper Verification of Source of a Communication Channel: https://cwe.mitre.org/data/definitions/940.html
- Android Network Security Configuration: https://developer.android.com/training/articles/security-config
- iOS App Transport Security: https://developer.apple.com/documentation/security/preventing_insecure_network_conne
- Certificate Pinning Best Practices: https://owasp.org/www-community/controls/Certificate_and_Public_Key_Pinning
- Electron Context Isolation: https://www.electronjs.org/docs/latest/tutorial/context-isolation

5.9 Remote Data Processing Systems [RDPS] The manufacturer shall implement secure remote data processing systems that protect data confidentiality, integrity, and availability while maintaining product functionality.

Overview:

A remote data processing solution (RDPS) is a component of a product that has an essential role in one or more of a product's functions and performs that role remotely from the local components of the product.

While many product update methods may fit into the definition of an RDPS, product update requirements are considered separately in Section 5.5 Update Delivery Mechanisms of this document.

All remote data processing solutions are components of the product, but their use is independent of the product's deployment environment. While a product's employment of an RDPS may create additional security requirements related to its remote nature and affect the tools available to an assessor, use of an RDPS does not significantly change the security requirements associated with the product's use or functions.

A remote data processing solution is a component of the product when it has been designed by or for the manufacturer or is in the manufacturer's control, and it is necessary for the product to perform its intended functions. Remote data processing solutions can perform any function of the product and are not limited to a product's core functions. Common uses of RDPS include remote storage of profile or configuration data, often to enable similar devices to use or access the data from a web interface. The system can also be a log storage or metrics collection endpoint, which is part of the product design, but those have dedicated sections in the present document with detailed requirements.

CRA Applicability: The CRA has a subtle difference in how the application is designed. A website that is accessed with a browser is not in scope, but an online service which is used from an installed application is in scope. The CRA Article 3(2) defines that an RDPS is under the responsibility of the manufacturer. Therefore, if the product default configuration is overwritten with local amendments by the user of the product, it is important that similar protection can be achieved as described in this section.

Capability: Secure remote data processing with encryption, authentication, availability controls, and data protection

Conditions:

- **RDPS-0**: No remote data processing (fully local operation)
- **RDPS-1**: Limited remote processing for non-sensitive data (configuration, preferences)
- **RDPS-2**: Extended remote processing including sensitive data with strong security controls
- **RDPS-3**: Full remote processing with critical data requiring maximum security

Threats:

- Eavesdropping and man-in-the-middle attacks during data transmission
- Unauthorized access to remote systems and data
- Data integrity compromise and unauthorized modification
- Denial of service attacks affecting service availability
- Multitenant data leakage in shared infrastructure
- Insider abuse and unauthorized access by service operators
- Data residency and sovereignty violations
- Insufficient data protection at rest
- Inadequate backup and recovery capabilities
- Vendor lock-in preventing data portability
- Service discontinuation without migration path
- Configuration tampering when user-defined RDPS endpoints allowed

Risk: HIGH - Compromised RDPS can lead to data theft, service disruption, privacy violations, and complete product functionality loss

Requirements:

RDPS-0 Requirements (No remote data processing)

- **RDPS-0-REQ-1**: Browser shall operate fully offline without requiring remote connectivity
- **RDPS-0-REQ-2**: All user data shall be stored locally without remote synchronization
- **RDPS-0-REQ-3**: Browser shall not transmit telemetry, diagnostics, or usage data to remote servers
- **RDPS-0-REQ-4**: Browser shall function without degradation when network connectivity unavailable
- **RDPS-0-REQ-5**: No remote authentication or authorization services shall be required

- **RDPS-0-REQ-6**: Browser shall document all local-only operation capabilities and limitations
- **RDPS-0-REQ-7**: Users shall be informed that no data leaves the local system

RDPS-1 Requirements (Limited remote processing for non-sensitive data)

- **RDPS-1-REQ-1**: Browser shall document product functionality when RDPS connectivity unavailable
- **RDPS-1-REQ-2**: Browser shall define all data processed or stored in RDPS with data classification
- **RDPS-1-REQ-3**: Browser shall classify criticality of all RDPS-processed data
- **RDPS-1-REQ-4**: Browser shall encrypt all data transmissions to RDPS using TLS 1.3 or higher
- **RDPS-1-REQ-5**: Browser shall authenticate RDPS endpoints using certificate validation
- **RDPS-1-REQ-6**: Browser shall implement retry mechanisms with exponential backoff for RDPS failures
- **RDPS-1-REQ-7**: Browser shall cache critical data locally for offline operation
- **RDPS-1-REQ-8**: Browser shall implement secure authentication for RDPS access
- **RDPS-1-REQ-9**: Browser shall validate server certificates and enforce certificate pinning for RDPS
- **RDPS-1-REQ-10**: Browser shall implement timeout controls for RDPS connections
- **RDPS-1-REQ-11**: Browser shall log RDPS connectivity failures and errors
- **RDPS-1-REQ-12**: Browser shall gracefully degrade functionality when RDPS unavailable
- **RDPS-1-REQ-13**: Browser shall not expose sensitive authentication credentials to RDPS
- **RDPS-1-REQ-14**: Browser shall implement rate limiting for RDPS requests
- **RDPS-1-REQ-15**: Browser shall validate all data received from RDPS before processing

RDPS-2 Requirements (Extended remote processing with sensitive data)

- **RDPS-2-REQ-1**: All RDPS-1 requirements shall be implemented
- **RDPS-2-REQ-2**: Browser shall encrypt sensitive data at rest in RDPS storage
- **RDPS-2-REQ-3**: Browser shall implement mutual TLS authentication for RDPS connections
- **RDPS-2-REQ-4**: Browser shall maintain redundant copies of critical data for recovery
- **RDPS-2-REQ-5**: Browser shall support data recovery from backups with integrity verification
- **RDPS-2-REQ-6**: Browser shall implement data retention policies with secure deletion
- **RDPS-2-REQ-7**: Browser shall enforce access controls on RDPS data per-user and per-origin
- **RDPS-2-REQ-8**: Browser shall audit all RDPS access and modifications
- **RDPS-2-REQ-9**: Browser shall implement data integrity verification using cryptographic hashes
- **RDPS-2-REQ-10**: Browser shall protect against RDPS endpoint substitution attacks
- **RDPS-2-REQ-11**: Browser shall implement defense against replay attacks on RDPS communications
- **RDPS-2-REQ-12**: Browser shall enforce data minimization principles for RDPS transmissions
- **RDPS-2-REQ-13**: Browser shall provide user controls for RDPS data synchronization
- **RDPS-2-REQ-14**: Browser shall implement secure data export from RDPS for data portability
- **RDPS-2-REQ-15**: When user-configurable RDPS endpoints provided, all associated security settings shall be configurable
- **RDPS-2-REQ-16**: Browser shall verify RDPS service availability before critical operations
- **RDPS-2-REQ-17**: Browser shall implement connection pooling with security controls for RDPS
- **RDPS-2-REQ-18**: Browser shall protect RDPS authentication tokens from extraction and theft

RDPS-3 Requirements (Full remote processing with critical data)

- **RDPS-3-REQ-1**: All RDPS-2 requirements shall be implemented
- **RDPS-3-REQ-2**: Browser shall implement end-to-end encryption for all critical data in RDPS
- **RDPS-3-REQ-3**: Browser shall use hardware-backed key storage for RDPS encryption keys
- **RDPS-3-REQ-4**: Browser shall implement high-availability RDPS architecture with failover
- **RDPS-3-REQ-5**: Browser shall document and test RDPS disaster recovery procedures
- **RDPS-3-REQ-6**: Browser shall implement real-time RDPS integrity monitoring
- **RDPS-3-REQ-7**: Browser shall provide RDPS security event logging with SIEM integration
- **RDPS-3-REQ-8**: Browser shall enforce geographic data residency requirements when configured
- **RDPS-3-REQ-9**: Browser shall implement zero-trust architecture for RDPS access

- **RDPS-3-REQ-10**: Browser shall support regulatory compliance logging for RDPS operations
- **RDPS-3-REQ-11**: Browser shall implement automated RDPS security scanning and vulnerability detection
- **RDPS-3-REQ-12**: Browser shall provide cryptographic proof of RDPS data integrity
- **RDPS-3-REQ-13**: Browser shall implement secure multi-tenancy with data isolation in RDPS
- **RDPS-3-REQ-14**: Browser shall provide incident response procedures for RDPS breaches
- **RDPS-3-REQ-15**: Browser shall implement RDPS access revocation mechanisms
- **RDPS-3-REQ-16**: Browser shall provide transparency reporting for RDPS data access
- **RDPS-3-REQ-17**: Browser shall implement forward secrecy for RDPS communications
- **RDPS-3-REQ-18**: Browser shall provide user notification of RDPS security events
- **RDPS-3-REQ-19**: Browser shall document service discontinuation and data migration procedures
- **RDPS-3-REQ-20**: Enterprise administrators shall be able to configure RDPS security policies

6 Technical Security Assessments

This chapter provides detailed technical assessment procedures for verifying conformance with the security requirements specified in Chapter 5. Each assessment follows a structured methodology designed to produce objective, repeatable results suitable for third-party conformity assessment.

6.1 Domain and Origin Isolation Assessments

This section covers assessment procedures for requirements DOM-REQ-1 through DOM-REQ-12, addressing site isolation, origin-based security boundaries, cross-origin resource sharing, and related isolation mechanisms.

Assessment: DOM-REQ-1 (Process-per-site isolation)

Reference: DOM-REQ-1 - Browser shall implement process-per-site isolation

Given: A conformant browser with site isolation capabilities (DOM-1 or higher)

Task: Verify that the browser enforces operating system-level process isolation between distinct sites to prevent compromise of one site from affecting other sites, and to enable OS-level security mechanisms (ASLR, sandboxing, memory protection) to provide defense-in-depth against web-based attacks.

Verification:

1. Open the browser and navigate to three distinct origins: https://example.com, https://test.com, and https://example.org → Create cross-origin iframes (example.com embedding test.com) and verify they run in separate processes
2. In each origin, open the browser's task manager or use platform process monitoring tools (Process Explorer on Windows, Activity Monitor on macOS, ps/top on Linux) → Confirm that each distinct site (eTLD+1) is allocated a separate renderer process
3. Identify the renderer processes associated with each origin → Verify that same-site navigations reuse the same renderer process
4. Record the Process IDs (PIDs) for each origin's renderer process → Check that cross-origin iframes are isolated in separate processes from their embedder
5. Navigate from https://example.com to https://example.com/page2 and verify the PID remains the same → Validate that process separation is maintained throughout the browser session
6. Navigate from https://example.com to https://different.example.com (different subdomain, same site) and verify the PID remains the same or changes according to documented site isolation policy → Review browser task manager or process monitoring tools to confirm distinct processes for distinct sites

Pass Criteria: All distinct sites use separate renderer processes AND cross-origin iframes are isolated in separate processes

Fail Criteria: Any two distinct sites share a renderer process OR cross-origin iframe runs in the same process as embedder

Evidence: Screenshots of browser task manager showing distinct PIDs for distinct sites, process tree diagrams, video recordings of process creation during navigation

References:

- Chromium Site Isolation Architecture: https://www.chromium.org/Home/chromium-security/site-isolation/
- Process Models for Web Browsers: https://www.chromium.org/developers/design-documents/process-models/
- Firefox Project Fission: https://wiki.mozilla.org/Project_Fission

Assessment: DOM-REQ-2 (Cross-origin read blocking)

Reference: DOM-REQ-2 - Browser shall enforce Cross-Origin Read Blocking (CORB)

Given: A conformant browser with DOM-1 or higher capability

Task: Verify that the browser prevents attackers from using script/image/stylesheet tags to read sensitive cross-origin data (JSON, HTML, XML) that lacks CORS headers, protecting against Spectre-style attacks and cross-site data leakage where malicious pages attempt to load victim resources into contexts that could expose response data through side channels.

Verification:

1. Set up a test web server serving resources with various MIME types (HTML, JSON, XML, images, scripts) → Test edge cases: nosniff headers, ambiguous MIME types, empty responses
2. Create a test page at https://attacker.com that attempts to load resources from https://victim.com using various methods: → Confirm that JSON, HTML, and XML responses without CORS headers are blocked from cross-origin script contexts
   - `<script src="https://victim.com/sensitive.json">`
   - `<img src="https://victim.com/sensitive.json">`
   - `<link rel="stylesheet" href="https://victim.com/sensitive.html">`
   - `fetch()` requests without CORS headers
3. Configure https://victim.com to serve JSON/HTML/XML resources without CORS headers → Verify that browser console shows CORB warning messages with details of blocked resources
4. Monitor network traffic using browser DevTools to capture CORB decisions → Check that network panel shows resources blocked with CORB designation
5. Verify that cross-origin requests for JSON/HTML/XML are blocked when served with incorrect MIME types → Validate that legitimate cross-origin subresources (images, scripts with correct MIME types) load successfully
6. Verify that legitimate cross-origin image/script/CSS loads still succeed → Confirm that `X-Content-Type-Options: nosniff` is respected in CORB decisions

Pass Criteria: All cross-origin HTML/JSON/XML loads without CORS headers are blocked from script contexts AND browser logs CORB blocking events

Fail Criteria: Any JSON/HTML/XML resource is readable cross-origin without CORS headers OR no CORB enforcement is observable

Evidence: Browser console screenshots showing CORB warnings, network panel captures showing blocked resources, packet captures demonstrating data was fetched but not exposed to scripts, test server logs

References:

- CORB Specification: https://chromium.googlesource.com/chromium/src/+/master/services/network/cross_origin_read
- Fetch Metadata Request Headers: https://www.w3.org/TR/fetch-metadata/
- MIME Sniffing Standard: https://mimesniff.spec.whatwg.org/

Assessment: DOM-REQ-3 (Strict origin policy enforcement)

Reference: DOM-REQ-3 - Browser shall prevent cross-origin DOM access without explicit consent

Given: A conformant browser with DOM-0 or higher capability

Task: Verify that the browser enforces the Same-Origin Policy to prevent scripts from one origin from reading or manipulating the DOM of another origin, protecting against cross-site scripting attacks where malicious sites attempt to steal sensitive data or hijack user sessions by accessing cross-origin window objects, documents, or storage.

Verification:

1. Create two test pages: https://site-a.com/test.html and https://site-b.com/test.html → Verify that `Location` object allows limited cross-origin access (href setter only) but not href getter
2. From site-a, open site-b in a new window using `window.open()` → Confirm that all cross-origin DOM access attempts throw SecurityError exceptions
3. Attempt cross-origin DOM access from site-a to site-b window: → Verify that browser console logs SecurityError with clear origin mismatch messages
   - `otherWindow.document` - attempt to access document
   - `otherWindow.localStorage` - attempt to access storage
   - `otherWindow.location.href = "javascript:..."` - attempt navigation hijacking
   - `otherWindow.frames[0]` - attempt to access frames
4. Verify that all cross-origin DOM access attempts throw SecurityError exceptions → Check that same-origin window access succeeds without errors
5. Test that same-origin window access succeeds → Validate that postMessage() works correctly for cross-origin communication
6. Test `window.postMessage()` as the legitimate cross-origin communication channel → Confirm that Location.href can be set cross-origin but not read

Pass Criteria: All cross-origin DOM property access attempts throw SecurityError AND postMessage provides functional cross-origin communication

Fail Criteria: Any cross-origin DOM property is readable/writable OR no exception is thrown

Evidence: Browser console screenshots showing SecurityError exceptions, video demonstration of test execution, automated test results from Web Platform Tests

References:

- HTML Standard - Cross-origin objects: https://html.spec.whatwg.org/multipage/browsers.html#cross-origin-objects
- Same-Origin Policy (MDN): https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy
- Web Platform Tests - Origin isolation: https://github.com/web-platform-tests/wpt/tree/master/html/browsers/origin

Assessment: DOM-REQ-4 (CORS preflight enforcement)

Reference: DOM-REQ-4 - Browser shall enforce CORS preflight for non-simple requests

Given: A conformant browser with DOM-1 or higher capability

Task: Verify that the browser enforces CORS preflight checks for potentially dangerous cross-origin requests (non-simple methods, custom headers) to prevent attackers from triggering unauthorized state-changing operations on victim servers, ensuring that servers have an opportunity to reject requests before they execute and protecting against CSRF-style attacks that bypass simple request restrictions.

Verification:

1. Set up a test server that logs all incoming requests including OPTIONS requests → Test preflight caching behavior

2. Create test pages that make various fetch() requests to cross-origin servers: → Confirm that OPTIONS preflight requests are sent before all non-simple cross-origin requests
   - Simple requests (GET with simple headers)
   - Non-simple requests (PUT, DELETE, PATCH methods)
   - Requests with custom headers (X-Custom-Header)
   - Requests with credentials (cookies)
3. Monitor network traffic to verify preflight OPTIONS requests are sent before non-simple requests → Verify that actual requests only proceed after successful preflight response with matching CORS headers
4. Configure the server to respond with various CORS header combinations: → Check that browser blocks requests when preflight fails or returns incorrect headers
   - Correct CORS headers (Access-Control-Allow-Origin, Allow-Methods, Allow-Headers)
   - Missing CORS headers
   - Incorrect origin in CORS headers
   - Expired preflight cache (Access-Control-Max-Age: 0)
5. Verify that actual requests only proceed after successful preflight → Validate that preflight responses are cached according to Access-Control-Max-Age

Pass Criteria: All non-simple requests are preceded by OPTIONS preflight AND requests fail when preflight response lacks appropriate CORS headers

Fail Criteria: Non-simple requests proceed without preflight OR requests succeed despite missing CORS headers

Evidence: Network panel screenshots showing OPTIONS requests before actual requests, server logs demonstrating preflight sequence, packet captures with timing analysis

References:

- Fetch Standard - CORS protocol: https://fetch.spec.whatwg.org/#http-cors-protocol
- CORS (MDN): https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS
- Preflight request specification: https://fetch.spec.whatwg.org/#cors-preflight-request

Assessment: DOM-REQ-5 (Cookie SameSite attribute enforcement)

Reference: DOM-REQ-5 - Browser shall enforce SameSite cookie attribute

Given: A conformant browser with DOM-1 or higher capability

Task: Verify that the browser enforces SameSite cookie restrictions to prevent Cross-Site Request Forgery (CSRF) attacks where malicious sites trigger authenticated requests to victim applications by controlling when cookies are sent in cross-site contexts, with Strict preventing all cross-site transmission, Lax allowing safe top-level navigations, and None requiring explicit Secure flag.

Verification:

1. Set up two test domains: https://site-a.com and https://site-b.com → Verify that SameSite=None requires Secure attribute
2. Configure site-a to set cookies with various SameSite attributes: → Confirm that SameSite=Strict cookies are never sent in cross-site contexts
   - `Set-Cookie: session=abc123; SameSite=Strict; Secure`
   - `Set-Cookie: tracking=xyz789; SameSite=Lax; Secure`
   - `Set-Cookie: legacy=old; Secure` (no SameSite)
   - `Set-Cookie: none=test; SameSite=None; Secure`
3. From site-b, perform various cross-site requests to site-a: → Verify that SameSite=Lax cookies are sent only in top-level navigation (GET)
   - Top-level navigation (clicking link)
   - Embedded resources (images, iframes)
   - JavaScript fetch() POST request
   - Form submission (GET and POST)

4. Monitor network traffic to verify which cookies are sent in each scenario → Check that SameSite=None cookies are sent in all contexts but require Secure attribute
5. Test the default SameSite behavior for cookies without explicit attribute (should be Lax) → Validate that cookies without SameSite attribute default to Lax behavior

Pass Criteria: Cookie transmission matches SameSite attribute policy for all test cases AND default behavior is Lax

Fail Criteria: Any cookie is sent in violation of its SameSite policy OR SameSite=None works without Secure

Evidence: Network panel screenshots showing Cookie headers in different contexts, DevTools Application tab showing cookie attributes, test server logs of received cookies

References:

- RFC 6265bis - SameSite Cookies: https://datatracker.ietf.org/doc/html/draft-ietf-httpbis-rfc6265bis
- SameSite Cookies Explained (web.dev): https://web.dev/samesite-cookies-explained/
- Cookie SameSite attribute (MDN): https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Set-Cookie/SameSite

Assessment: DOM-REQ-6 (Origin-bound storage isolation)

Reference: DOM-REQ-6 - Browser shall isolate localStorage and IndexedDB per origin

Given: A conformant browser with DOM-0 or higher capability

Task: Verify that the browser enforces complete storage isolation between origins to prevent malicious sites from reading sensitive user data (session tokens, personal information) stored by other applications, ensuring that origin boundaries (scheme, host, port) create impermeable barriers for localStorage and IndexedDB access.

Verification:

1. Create test pages at multiple origins: https://example.com, https://example.org, http://example.com, https://sub.example.com → Test port isolation (example.com:443 vs example.com:8080)
2. In each origin, write distinct data to localStorage: → Clear storage for one origin and verify other origins' storage is unaffected

   ```
   localStorage.setItem('origin-test', location.origin);
   localStorage.setItem('timestamp', Date.now());
   ```

3. In each origin, create an IndexedDB database with a distinct name and store origin-specific data → Confirm that each origin has completely isolated localStorage namespace
4. Attempt to read localStorage and IndexedDB from each origin → Verify that each origin has completely isolated IndexedDB namespace
5. Verify that each origin only sees its own storage → Check that different subdomains cannot access each other's storage
6. Test subdomain isolation (example.com vs sub.example.com) → Validate that different protocols (http vs https) have separate storage
7. Test protocol isolation (https vs http) → Confirm that different ports have separate storage

Pass Criteria: No origin can read another origin's localStorage or IndexedDB AND all origin components (scheme, host, port) contribute to isolation boundary

Fail Criteria: Any cross-origin storage access succeeds OR incomplete origin matching (e.g., ignoring port)

Evidence: Browser DevTools Application tab screenshots showing storage contents per origin, console logs demonstrating isolation, automated test results

References:

- Web Storage API specification: https://html.spec.whatwg.org/multipage/webstorage.html
- IndexedDB API specification: https://w3c.github.io/IndexedDB/
- Origin definition: https://html.spec.whatwg.org/multipage/origin.html#concept-origin

Assessment: DOM-REQ-7 (Frame sandboxing support)

Reference: DOM-REQ-7 - Browser shall support iframe sandbox attribute

Given: A conformant browser with DOM-1 or higher capability

Task: Verify that the browser implements iframe sandbox restrictions to mitigate risks from untrusted content by allowing developers to apply least-privilege principles to embedded frames, preventing malicious iframes from executing scripts, accessing parent windows, navigating the top frame, or abusing other dangerous capabilities unless explicitly permitted.

Verification:

1. Create test pages with iframes using various sandbox configurations: → Confirm that sandbox attribute restricts capabilities according to specified tokens

   ```
   <iframe sandbox src="test.html"></iframe>
   <iframe sandbox="allow-scripts" src="test.html"></iframe>
   <iframe sandbox="allow-scripts allow-same-origin" src="test.html"></iframe>
   <iframe sandbox="allow-forms allow-popups" src="test.html"></iframe>
   ```

2. In each sandboxed iframe, attempt various actions: → Verify that scripts are blocked unless allow-scripts is present

   - JavaScript execution (alert, console.log)
   - Form submission
   - Opening popups (window.open)
   - Accessing parent window
   - Accessing localStorage
   - Top navigation (top.location = . . .)

3. Verify that only explicitly allowed capabilities work → Check that same-origin access is blocked unless allow-same-origin is present

4. Test that sandbox="" (empty) applies strictest restrictions → Validate that sandboxed frames without allow-same-origin have unique opaque origin

5. Test CSP sandbox directive equivalence → Confirm that browser console logs security errors for blocked actions

6. Verify unique origin treatment for sandboxed iframes without allow-same-origin → Verify that CSP sandbox directive provides equivalent restrictions

Pass Criteria: All tested restrictions are enforced according to sandbox tokens AND browser logs security errors for blocked actions

Fail Criteria: Any capability works without corresponding allow-* token OR no restrictions are observed

Evidence: Browser console screenshots showing blocked actions, DevTools showing unique origin for sandboxed frames, test results demonstrating each sandbox token

References:

- HTML Standard - Sandboxing: https://html.spec.whatwg.org/multipage/iframe-embed-object.html#attr-iframe-sandbox
- CSP sandbox directive: https://www.w3.org/TR/CSP3/#directive-sandbox

- iframe sandbox (MDN): https://developer.mozilla.org/en-US/docs/Web/HTML/Element/iframe#attr-sandbox

Assessment: DOM-REQ-8 (Opaque origin handling)

Reference: DOM-REQ-8 - Browser shall treat sandboxed and data: origins as opaque

Given: A conformant browser with DOM-0 or higher capability

Task: Verify that the browser treats sandboxed iframes and data: URLs as having opaque (unique, unguessable) origins that cannot access storage or credentials, preventing untrusted content from stealing sensitive data or establishing persistent state, while ensuring that each opaque origin is internally unique to prevent two untrusted contexts from communicating even though both serialize as "null".

Verification:

1. Create test scenarios for opaque origins:
   - Sandboxed iframe without allow-same-origin: `<iframe sandbox="allow-scripts" src="...">`
   - data: URL navigation: `window.open('data:text/html,<h1>Test</h1>')`
   - Blob URL: `URL.createObjectURL(new Blob(['...'], {type: 'text/html'}))`
2. In each opaque origin context, attempt to:
   - Access localStorage/sessionStorage (should throw SecurityError)
   - Access IndexedDB (should throw SecurityError)
   - Make fetch() requests (should succeed but not send credentials)
   - Access parent window (should be blocked for sandboxed frames)
3. Verify that opaque origins serialize as "null" in `window.origin`
4. Test that two distinct opaque origins cannot access each other even though both serialize as "null"
5. Verify that cookies are not sent/received from opaque origins
6. Confirm that opaque origins report window.origin as "null"
7. Verify that opaque origins cannot access localStorage, sessionStorage, or IndexedDB
8. Check that opaque origins do not send or receive cookies
9. Validate that each opaque origin is unique and cannot access other opaque origins
10. Confirm that fetch requests from opaque origins work but without credentials

Pass Criteria: All storage APIs throw SecurityError in opaque origins AND window.origin reports "null" AND cookies are not sent

Fail Criteria: Any storage access succeeds from opaque origin OR cookies are sent/received

Evidence: Console screenshots showing SecurityError exceptions, network captures showing missing cookies, DevTools Application tab showing empty storage

References:

- HTML Standard - Opaque origins: https://html.spec.whatwg.org/multipage/origin.html#concept-origin-opaque
- data: URL security: https://datatracker.ietf.org/doc/html/rfc2397
- Fetch - CORS and credentials: https://fetch.spec.whatwg.org/#http-cors-protocol

Assessment: DOM-REQ-9 (CORP for cross-origin isolation)

Reference: DOM-REQ-9 - Browser shall support Cross-Origin-Resource-Policy header

Given: A conformant browser with DOM-2 or higher capability

Task: Verify that the browser enforces Cross-Origin-Resource-Policy (CORP) headers to allow servers to protect their resources from being loaded by cross-origin pages, defending against Spectre-style attacks where malicious sites embed victim resources to leak data through side channels, and enabling servers to opt into stronger isolation guarantees for sensitive content.

Verification:

1. Set up test servers on multiple origins (site-a.com, site-b.com, cdn.example.com) → Test CORP enforcement in cross-origin isolated contexts (COOP+COEP)
2. Configure site-a to serve resources with various CORP headers: → Confirm that resources with CORP: same-origin are blocked from cross-origin loads
   - `Cross-Origin-Resource-Policy: same-origin`
   - `Cross-Origin-Resource-Policy: same-site`
   - `Cross-Origin-Resource-Policy: cross-origin`
3. From site-b, attempt to load resources from site-a: → Verify that resources with CORP: same-site are blocked from cross-site loads but allowed same-site

   ```
   <img src="https://site-a.com/image-same-origin.png">
   <script src="https://site-a.com/script-same-site.js"></script>
   <iframe src="https://site-a.com/frame-cross-origin.html"></iframe>
   ```
4. Verify blocking behavior based on CORP header → Check that resources with CORP: cross-origin load from any origin
5. Test CORP interaction with CORS headers → Validate that browser console shows CORP blocking errors with clear messages
6. Verify that CORP applies to all resource types (images, scripts, frames, fetch) → Confirm that CORP is enforced for all resource types

Pass Criteria: All CORP policies are enforced according to specification AND browser logs blocking errors

Fail Criteria: Any resource loads in violation of its CORP header OR no CORP enforcement is observable

Evidence: Network panel showing blocked resources, console screenshots showing CORP errors, test results demonstrating same-origin/same-site/cross-origin behavior

References:

- CORP Specification: https://fetch.spec.whatwg.org/#cross-origin-resource-policy-header
- Cross-Origin Isolation guide: https://web.dev/coop-coep/
- CORP (MDN): https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Cross-Origin-Resource-Policy

Assessment: DOM-REQ-10 (COOP enforcement)

Reference: DOM-REQ-10 - Browser shall enforce Cross-Origin-Opener-Policy

Given: A conformant browser with DOM-2 or higher capability

Task: Verify that the browser enforces Cross-Origin-Opener-Policy (COOP) to prevent cross-origin documents from sharing browsing context groups and accessing each other through window.opener references, protecting against Spectre-style attacks by enabling process isolation and allowing sites to opt into cross-origin isolation that grants access to powerful features like SharedArrayBuffer.

Verification:

1. Create test pages with various COOP headers: → Test COOP reporting endpoint functionality
   - `Cross-Origin-Opener-Policy: same-origin`
   - `Cross-Origin-Opener-Policy: same-origin-allow-popups`
   - `Cross-Origin-Opener-Policy: unsafe-none` (default)
2. Test window.opener relationships: → Confirm that COOP: same-origin severs opener relationship with cross-origin pages
   - Page A (COOP: same-origin) opens Page B (no COOP) → opener should be null
   - Page A (no COOP) opens Page B (COOP: same-origin) → opener should be null
   - Page A (COOP: same-origin) opens Page B (same-origin with COOP) → opener should work

3. Verify browsing context group isolation → Verify that COOP: same-origin-allow-popups preserves opener for popups but not navigations
4. Test that cross-origin-isolated pages cannot be in the same browsing context group as non-isolated pages → Check that cross-origin isolated pages (COOP + COEP) get access to high-resolution timers and SharedArrayBuffer
5. Verify SharedArrayBuffer availability in cross-origin isolated contexts → Validate that browser process allocation reflects browsing context group isolation

Pass Criteria: Opener relationship is severed as specified by COOP policy AND cross-origin isolation enables SharedArrayBuffer

Fail Criteria: Opener relationship persists in violation of COOP policy OR SharedArrayBuffer unavailable in properly isolated context

Evidence: Console logs showing null window.opener, DevTools showing browsing context groups, demonstration of SharedArrayBuffer availability, network captures of violation reports

References:

- COOP Specification: https://html.spec.whatwg.org/multipage/origin.html#cross-origin-opener-policies
- Cross-Origin Isolation guide: https://web.dev/coop-coep/
- SharedArrayBuffer and cross-origin isolation: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Glob

Assessment: DOM-REQ-11 (COEP enforcement)

Reference: DOM-REQ-11 - Browser shall enforce Cross-Origin-Embedder-Policy

Given: A conformant browser with DOM-2 or higher capability

Task: Verify that the browser enforces Cross-Origin-Embedder-Policy (COEP) to ensure that all cross-origin resources loaded by a document have explicitly opted in via CORP or CORS headers, preventing the document from inadvertently loading attacker-controlled resources that could be used in Spectre-style side-channel attacks, and enabling cross-origin isolation when combined with COOP.

Verification:

1. Create a test page with COEP header: `Cross-Origin-Embedder-Policy: require-corp` → Confirm that resources without CORP or CORS are blocked when page has COEP: require-corp
2. From this page, attempt to load various cross-origin resources: → Verify that resources with CORP: cross-origin load successfully
   - Image without CORP/CORS: `<img src="https://cross-origin.com/image.png">`
   - Image with CORP: `<img src="https://cross-origin.com/image-with-corp.png">` (CORP: cross-origin)
   - Script without CORS: `<script src="https://cross-origin.com/script.js">`
   - Script with CORS: `<script src="..." crossorigin="anonymous">` (with proper CORS headers)
   - Iframe without COEP: `<iframe src="https://cross-origin.com/page.html">`
   - Iframe with COEP: `<iframe src="...">` (page has COEP header)
3. Verify blocking of resources without CORP/CORS → Check that resources with CORS and crossorigin attribute load successfully
4. Test COEP: credentialless as alternative to require-corp → Validate that iframes without COEP are blocked from embedding in COEP page
5. Verify that combining COOP + COEP enables cross-origin isolation → Confirm that browser console shows COEP blocking errors
6. Test COEP violation reporting → Verify that COOP + COEP combination enables self.crossOriginIsolated === true

Pass Criteria: All resources without CORP/CORS are blocked AND cross-origin isolation is achieved with COOP+COEP

Fail Criteria: Any resource without CORP/CORS loads successfully OR cross-origin isolation not achieved despite proper headers

Evidence: Network panel showing blocked resources, console screenshots showing COEP errors, JavaScript console showing self.crossOriginIsolated === true

References:

- COEP Specification: https://html.spec.whatwg.org/multipage/origin.html#coep
- Cross-Origin Isolation: https://web.dev/coop-coep/
- COEP (MDN): https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Cross-Origin-Embedder-Policy

Assessment: DOM-REQ-12 (Document.domain deprecation)

Reference: DOM-REQ-12 - Browser shall restrict or remove document.domain setter

Given: A conformant browser with DOM-1 or higher capability

Task: Verify that the browser restricts or removes the document.domain setter to eliminate a legacy same-origin policy relaxation mechanism that undermines site isolation, prevents origin-keyed agent clustering for performance, and creates security risks by allowing subdomains to arbitrarily merge their security boundaries, requiring sites to explicitly opt-out of modern isolation to use this deprecated feature.

Verification:

1. Create test pages on related subdomains: https://sub1.example.com and https://sub2.example.com → Test browser console warnings about document.domain deprecation
2. Attempt to relax same-origin policy using document.domain: → Confirm that document.domain setter is unavailable or no-op by default

   ```
   // On both sub1.example.com and sub2.example.com
   document.domain = 'example.com';
   ```

3. Verify that document.domain setter is either: → Verify that pages can only set document.domain if they explicitly opt-out via Origin-Agent-Cluster: ?0
   - Removed entirely (throws error or no-op)
   - Gated behind Origin-Agent-Cluster: ?0 header
4. Test that pages with `Origin-Agent-Cluster: ?1` (default) cannot set document.domain → Check that browser console shows deprecation warnings when document.domain is accessed
5. Verify that failing to set document.domain prevents cross-subdomain DOM access → Validate that cross-subdomain access fails when document.domain is disabled

Pass Criteria: document.domain is restricted by default (requires explicit opt-out) AND browser shows deprecation warnings

Fail Criteria: document.domain works unconditionally OR no deprecation warnings shown

Evidence: Console screenshots showing errors/warnings, test results showing cross-subdomain access blocked, DevTools showing Origin-Agent-Cluster header processing

References:

- document.domain deprecation: https://developer.chrome.com/blog/immutable-document-domain/
- Origin-Agent-Cluster header: https://html.spec.whatwg.org/multipage/origin.html#origin-agent-cluster
- HTML Standard - Origin-keyed agent clusters: https://html.spec.whatwg.org/multipage/origin.html#origin-keyed-agent-clusters

Assessment: DOM-REQ-13 (Enterprise origin isolation policy configuration)

Reference: DOM-2-REQ-11 - Administrators shall be able to configure origin isolation policies via enterprise policy

Given: A conformant browser with DOM-2 or higher capability in an enterprise environment

Task: Verify that enterprise administrators can configure origin isolation policies to accommodate legitimate business requirements for cross-origin interactions while maintaining security boundaries, enabling organizations to define exceptions for trusted domains or internal applications without requiring code changes or browser rebuilds, supporting controlled relaxation of isolation for specific business scenarios.

Verification:

1. Access the browser's enterprise policy management interface or configuration file → Verify that origins not covered by the policy maintain default isolation
2. Identify available policies related to origin isolation (e.g., SitePerProcessMode, IsolateOriginsMode, CrossOriginPolicyExceptions) → Test that policy changes require browser restart or profile reload to take effect
3. Configure a policy to modify default isolation behavior for specific origins or domains → Confirm that policy-configured exceptions are logged in browser diagnostic logs
4. Deploy the policy through enterprise management tools (Group Policy, MDM, configuration management) → Verify that users cannot override enterprise-configured isolation policies
5. Verify that the browser applies the configured policy on startup → Test that malformed policies are rejected with clear error messages
6. Test that the configured isolation exceptions work as intended for specified origins → Validate that policy configuration is documented in enterprise administration guides

Pass Criteria: Enterprise policies for origin isolation are available AND policies can be centrally deployed AND configured exceptions work as specified AND non-configured origins maintain default isolation

Fail Criteria: No enterprise policy mechanism exists for origin isolation OR policies cannot be enforced OR users can override enterprise policies OR policy application is inconsistent

Evidence: Enterprise policy configuration screenshots, browser policy status page showing applied policies, test results demonstrating policy-based isolation exceptions, diagnostic logs showing policy application

References:

- HTML Standard - Origin-keyed agent clusters: https://html.spec.whatwg.org/multipage/origin.html#origin-keyed-agent-clusters
- Site Isolation: https://www.chromium.org/Home/chromium-security/site-isolation/

Assessment: DOM-REQ-14 (Logging of policy-based isolation exceptions)

Reference: DOM-2-REQ-12 - Browser shall log all policy-based isolation exceptions

Given: A conformant browser with DOM-2 or higher capability with configured origin isolation policies

Task: Verify that the browser logs all policy-based isolation exceptions to provide security teams with visibility into when and where isolation boundaries are relaxed, enabling detection of policy misconfiguration, monitoring of exception usage, and investigation of potential security incidents involving cross-origin access that was permitted through administrative policy rather than standard browser security controls.

Verification:

1. Configure enterprise policies that create isolation exceptions for specific origins → Test that log entries distinguish between policy-based exceptions and standard CORS/postMessage
2. Access the browser's logging interface (console logs, diagnostic logs, or audit logs) → Verify that logs can be exported or forwarded to SIEM systems

3. Trigger cross-origin interactions that invoke policy-based exceptions → Confirm that logging does not expose sensitive user data
4. Verify that each exception usage is logged with relevant details: → Test that log volume is reasonable and does not impact browser performance
    - Timestamp of the exception
    - Source and target origins involved
    - Policy name or identifier that permitted the exception
    - Type of isolation relaxation (storage access, DOM access, etc.)
5. Test that logs include sufficient information for security auditing → Verify that logs persist across browser restarts
6. Verify that logs are written to a location accessible to security teams → Validate that disabled logging can be detected by monitoring systems

Pass Criteria: All policy-based isolation exceptions are logged AND logs contain sufficient detail for auditing AND logs are accessible to security teams AND logging does not expose sensitive data

Fail Criteria: Isolation exceptions are not logged OR logs lack necessary detail OR logs are inaccessible OR logging exposes sensitive user information

Evidence: Log file excerpts showing isolation exception entries, log analysis demonstrating completeness, SIEM integration test results, documentation of log format and fields

References:

- CWE-778: Insufficient Logging: https://cwe.mitre.org/data/definitions/778.html
- OWASP Logging Cheat Sheet: https://cheatsheetseries.owasp.org/cheatsheets/Logging_Cheat_Sheet.html

Assessment: DOM-REQ-15 (Compatibility mode isolation integrity)

Reference: DOM-3-REQ-7 - Compatibility modes shall not weaken core isolation boundaries

Given: A conformant browser with DOM-3 capability and compatibility modes enabled

Task: Verify that browser compatibility modes designed to support legacy web content do not compromise fundamental origin isolation boundaries, ensuring that even when compatibility features are enabled for older sites or applications, core security mechanisms like process isolation, storage separation, and cross-origin restrictions remain effective, preventing compatibility modes from becoming vectors for cross-origin attacks.

Verification:

1. Identify available compatibility modes in the browser (e.g., quirks mode, document mode, legacy rendering mode) → Test that CORS and SameSite policies are enforced in compatibility mode
2. Enable compatibility mode through meta tags, HTTP headers, or browser settings → Verify that sandbox attributes on iframes work correctly in compatibility mode
3. Load test pages with compatibility mode activated → Test that postMessage remains the only valid cross-origin communication mechanism
4. Verify that process-per-site isolation remains active under compatibility mode → Confirm that compatibility mode does not bypass Content Security Policy
5. Test that storage (localStorage, sessionStorage, IndexedDB) remains origin-isolated in compatibility mode → Verify that cross-origin cookies are still subject to SameSite restrictions
6. Attempt cross-origin DOM access in compatibility mode and verify it is blocked → Test that compatibility mode does not enable deprecated features like unrestricted document.domain

Pass Criteria: Core isolation mechanisms remain enforced in compatibility mode AND process isolation is maintained AND storage isolation is preserved AND cross-origin restrictions are not weakened

Fail Criteria: Compatibility mode weakens any core isolation boundary OR enables cross-origin access not available in standard mode OR bypasses security policies

Evidence: Test results demonstrating isolation enforcement in compatibility mode, process inspection showing maintained process separation, DevTools showing storage isolation, security policy enforcement verification

References:

- HTML Standard - Quirks mode: https://html.spec.whatwg.org/multipage/dom.html#concept-document-quirks
- Same-origin policy: https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy

Assessment: DOM-REQ-16 (Third-party integration isolation)

Reference: DOM-3-REQ-8 - Third-party integrations shall be subject to same origin isolation policies

Given: A conformant browser with DOM-3 capability supporting third-party integrations

Task: Verify that third-party browser integrations such as plugins, extensions, or embedded components are subject to the same origin isolation policies as standard web content, preventing privileged integrations from bypassing security boundaries to access cross-origin data, ensuring that even trusted third-party code operates within the browser's security model and cannot be exploited to violate origin isolation.

Verification:

1. Install or enable third-party integrations (extensions, plugins, or native components) → Verify that third-party integrations run in appropriately isolated processes
2. Load web content from multiple origins (https://example.com, https://test.com) → Test that integration code injected into pages respects Content Security Policy
3. Verify that third-party integrations cannot access cross-origin DOM without proper permissions → Confirm that integrations cannot bypass SameSite cookie restrictions
4. Test that third-party code cannot read localStorage or cookies from other origins → Verify that third-party code cannot relax origin isolation through internal APIs
5. Verify that integrations requiring cross-origin access declare explicit permissions → Test that integration crashes or failures do not compromise other origins
6. Test that integration-provided APIs are subject to CORS when accessed cross-origin → Validate that integration permissions are clearly disclosed to users

Pass Criteria: Third-party integrations are subject to origin isolation AND cannot bypass cross-origin restrictions AND require explicit permissions for cross-origin access AND run in isolated contexts

Fail Criteria: Third-party integrations can bypass origin isolation OR access cross-origin data without permissions OR are not properly isolated

Evidence: Integration security test results, permission disclosure screenshots, process isolation verification, cross-origin access attempt logs showing blocks

References:

- Web Extensions API: https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions
- Same-origin policy: https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy

Assessment: DOM-REQ-17 (Documentation and logging of compatibility exceptions)

Reference: DOM-3-REQ-9 - All isolation exceptions for compatibility shall be documented and logged

Given: A conformant browser with DOM-3 capability with compatibility-related isolation exceptions

Task: Verify that all isolation exceptions implemented for compatibility reasons are comprehensively documented for developers and administrators, and that runtime usage of these exceptions is logged for security monitoring, ensuring transparency about when and why isolation boundaries are relaxed and enabling detection of unexpected or malicious use of compatibility features to bypass security controls.

Verification:

1. Review browser documentation for a complete list of compatibility-related isolation exceptions
2. Verify that each exception is documented with:
   - Clear description of the exception behavior
   - Justification for why the exception exists
   - Affected origins or scenarios
   - Security implications and mitigations
   - Deprecation timeline (if applicable)
3. Enable browser diagnostic logging or audit logging
4. Trigger compatibility features that involve isolation exceptions
5. Verify that each exception usage is logged in real-time with:
   - Timestamp of exception invocation
   - Origin(s) involved
   - Type of isolation exception (e.g., cross-origin storage access, DOM access)
   - Compatibility feature that triggered the exception
6. Test that logs include context for security analysis
7. Verify that logs can be reviewed by security teams
8. Confirm that exception documentation is accessible to web developers
9. Test that undocumented exceptions are flagged as potential issues
10. Validate that log entries are distinct from standard CORS/postMessage usage

Pass Criteria: All compatibility exceptions are documented AND documentation includes security implications AND runtime usage is logged AND logs provide sufficient detail for security analysis

Fail Criteria: Exceptions are undocumented OR documentation lacks security details OR exception usage is not logged OR logs are insufficient for analysis

Evidence: Documentation excerpts describing exceptions, log samples showing exception usage, security team access verification, developer documentation review

References:

- CWE-778: Insufficient Logging: https://cwe.mitre.org/data/definitions/778.html
- OWASP Application Security Verification Standard: https://owasp.org/www-project-application-security-verification-standard/

Assessment: DOM-REQ-18 (Embedded component storage isolation)

Reference: DOM-3-REQ-10 - Embedded components shall maintain storage isolation from embedding context

Given: A conformant browser with DOM-3 capability supporting embedded browser components

Task: Verify that embedded browser components (such as WebView controls in native applications or embedded iframes) maintain strict storage isolation from their embedding context, preventing the embedding application or parent frame from directly accessing the embedded component's localStorage, cookies, IndexedDB, or other origin-scoped storage, ensuring that embedded web content cannot be compromised through storage manipulation by the host application.

Verification:

1. Create an embedded browser component (WebView, iframe) in a host application or page → Test that the embedded component cannot access the embedding application's storage
2. Load web content in the embedded component from origin https://example.com → Verify that storage isolation is maintained even if origins match
3. From the embedding context, attempt to access the embedded component's storage: → Test that only secure message-passing APIs can exchange data between contexts
   - Try to read localStorage from the embedded origin
   - Attempt to access cookies belonging to the embedded origin
   - Try to open IndexedDB databases from the embedded origin

4. Verify that all direct storage access attempts from embedding context are blocked → Confirm that clearing embedding context storage does not affect embedded component storage
5. Load the same origin (https://example.com) in a separate tab or window → Verify that embedded component storage persists independently
6. Verify that storage is isolated between embedded component and separate browsing context → Test that storage isolation applies to all storage mechanisms (localStorage, sessionStorage, IndexedDB, Cache API, cookies)

Pass Criteria: Embedded components maintain complete storage isolation AND embedding context cannot access embedded storage AND only secure APIs enable data exchange AND isolation applies to all storage types

Fail Criteria: Embedding context can access embedded component storage OR storage isolation is incomplete OR isolation can be bypassed

Evidence: Test results showing blocked storage access attempts, storage isolation verification across contexts, secure API usage examples, developer documentation of isolation boundaries

References:

- Web Storage API: https://html.spec.whatwg.org/multipage/webstorage.html
- IndexedDB API: https://w3c.github.io/IndexedDB/
- Cookies: https://httpwg.org/specs/rfc6265.html

6.2 Extension System Security Assessments

This section covers assessment procedures for requirements EXT-REQ-1 through EXT-REQ-18, addressing browser extension security including permissions, content script isolation, extension API access control, manifest validation, and extension update security.

Assessment: EXT-REQ-1 (Permission model for extensions)

Reference: EXT-REQ-1 - Browser shall implement a permission model that restricts extension capabilities to explicitly declared permissions

Given: A conformant browser with EXT-1 or higher capability

Task: Verify that the browser implements a least-privilege permission model for extensions to prevent malicious or compromised extensions from accessing sensitive APIs and user data beyond their declared functionality, protecting users from over-privileged extensions that could steal credentials, intercept network traffic, or exfiltrate browsing history without explicit user consent.

Verification:

1. Create a test extension with a minimal manifest.json declaring only basic permissions (e.g., "storage", "tabs") → Verify that extensions cannot dynamically request permissions not declared in optional_permissions
2. Attempt to use APIs that require undeclared permissions: → Test that host permissions are enforced for content script injection and webRequest interception
   - Cookie management interface (requires "cookies" permission)
   - Network request interception interface (requires "webRequest" permission)
   - File download interface (requires "downloads" permission)
   - Access to specific host patterns not declared in host_permissions
3. Verify that the browser blocks API access and throws exceptions for undeclared permissions → Confirm that extensions cannot access APIs without corresponding manifest permissions
4. Monitor browser console for permission-related error messages → Verify that browser throws clear error messages when undeclared APIs are accessed (e.g., "Cannot access cookie management interface without 'cookies' permission")

5. Add the required permissions to manifest.json and reload the extension → Check that permission prompts at install time accurately reflect all requested permissions
6. Verify that previously blocked APIs now function correctly → Validate that host permissions restrict content script injection to declared patterns
7. Test that permission requests at install time accurately reflect manifest declarations → Confirm that optional permissions can only be requested if declared in manifest

Pass Criteria: All API access is blocked when permissions are not declared AND clear error messages are shown AND permission grants are persistent across sessions

Fail Criteria: Any API access succeeds without declared permission OR permission system can be bypassed OR no error messages are shown

Evidence: Console screenshots showing permission errors, test results demonstrating API blocking, DevTools Extension panel showing active permissions, permission prompt screenshots

References:

- Chrome Extension Permissions: https://developer.chrome.com/docs/extensions/mv3/declare_permissions/
- Mozilla WebExtensions Permissions: https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/manifest.json/permissions
- Manifest V3 Permissions: https://developer.chrome.com/docs/extensions/mv3/permission_warnings/
- CWE-250: Execution with Unnecessary Privileges: https://cwe.mitre.org/data/definitions/250.html
- Optional Permissions API: https://developer.chrome.com/docs/extensions/reference/permissions/
- WebExtensions Security Best Practices: https://extensionworkshop.com/documentation/develop/build-a-secure-extension/

Assessment: EXT-REQ-2 (Content script isolation)

Reference: EXT-REQ-2 - Browser shall isolate content scripts from web page JavaScript contexts

Given: A conformant browser with EXT-1 or higher capability

Task: Verify that the browser isolates content scripts in a separate JavaScript execution world from the host web page to prevent malicious pages from stealing extension secrets, intercepting extension message passing, or using prototype pollution to compromise the extension's security, while allowing content scripts to safely manipulate the DOM for legitimate functionality.

Verification:

1. Create a test extension with a content script that: → Test protection against prototype pollution attacks from page context to content script
   - Defines a global variable: `var extensionSecret = "sensitive_data"`
   - Attempts to access variables defined by the web page
   - Uses messaging to communicate with the background script
2. Create a test web page that: → Confirm that content scripts and web page JavaScript execute in separate JavaScript worlds
   - Defines its own global variables: `var pageVariable = "page_data"`
   - Attempts to access variables defined by the content script
   - Attempts to intercept or modify content script message passing
3. Verify that the content script cannot directly access web page variables and vice versa → Verify that variables and functions defined in one context are not accessible from the other
4. Test that content scripts run in an isolated JavaScript world with separate global scope → Check that content scripts can manipulate the DOM but cannot access JavaScript objects from page context
5. Verify that DOM modifications are visible to both contexts but JavaScript objects are not shared → Validate that message passing between content scripts and extension background is not interceptable by web page
6. Test that the web page cannot intercept extension runtime message passing calls from content scripts → Confirm that prototype modifications in page context do not affect content script execution

7. Verify that content scripts cannot access page's inline event handlers or Function.prototype modifications → Verify that content scripts have access to clean browser APIs unmodified by page JavaScript

Pass Criteria: Complete JavaScript isolation between content script and web page contexts AND secure message passing AND protection from prototype pollution

Fail Criteria: Any JavaScript objects/variables leak between contexts OR message passing can be intercepted OR prototype pollution succeeds

Evidence: Console logs showing undefined variables across contexts, test results demonstrating isolation, browser DevTools showing separate execution contexts, security test results showing prototype pollution protection

References:

- Chrome Content Script Isolated Worlds: https://developer.chrome.com/docs/extensions/mv3/content_scripts/#isolated_
- Mozilla Content Script Execution Environment: https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/Content_scripts#execution_environment
- WebExtensions Content Script Security: https://developer.chrome.com/docs/extensions/mv3/security/
- CWE-501: Trust Boundary Violation: https://cwe.mitre.org/data/definitions/501.html
- Prototype Pollution Prevention: https://portswigger.net/web-security/prototype-pollution
- Content Script Communication: https://developer.chrome.com/docs/extensions/mv3/messaging/

Assessment: EXT-REQ-3 (Extension API access control)

Reference: EXT-REQ-3 - Browser shall enforce access control for sensitive extension APIs based on manifest declarations

Given: A conformant browser with EXT-1 or higher capability

Task: Verify that the browser restricts access to powerful extension APIs (debugger, privacy settings, system information) based on explicit manifest permissions to prevent malicious extensions from escalating privileges, debugging other extensions/tabs to steal data, modifying privacy settings, or accessing system information without user awareness and consent.

Verification:

1. Identify sensitive extension APIs that require specific permissions: → Verify access control is enforced consistently across background scripts, content scripts, and popup contexts
   - Debugger interface (requires "debugger" permission)
   - Extension management interface (requires "management" permission)
   - Privacy settings interface (requires "privacy" permission)
   - System information interfaces for CPU/memory/storage (requires "system.*" permissions)
   - Desktop capture interface (requires "desktopCapture" permission)
2. Create test extensions with varying permission sets → Confirm that sensitive APIs are blocked without appropriate permissions
3. Attempt to access each sensitive API without the required permission → Verify that browser throws descriptive errors when API access is denied
4. Verify that access is denied with clear error messages → Check that permission warnings during extension install clearly communicate sensitive capabilities
5. Test that powerful APIs like debugger and management show enhanced warnings during install → Validate that API access control is enforced uniformly across all extension contexts
6. Verify that certain APIs (e.g., debugger) cannot be used in published extensions on official extension stores → Confirm that no bypasses exist through code evaluation or dynamic loading
7. Test that API access control cannot be bypassed through indirect means (eval, dynamic code loading) → Verify that debugger and management APIs show enhanced security warnings

Pass Criteria: All sensitive APIs are access-controlled based on manifest permissions AND appropriate warnings are shown AND no bypass mechanisms exist

Fail Criteria: Any sensitive API accessible without permission OR warnings are insufficient OR bypass mechanisms exist

Evidence: Console screenshots showing API access errors, permission prompt screenshots showing warnings, test results across multiple extension contexts, official extension store policy enforcement verification

References:

- Chrome Extension API Reference: https://developer.chrome.com/docs/extensions/reference/
- Sensitive Permissions in Chrome: https://developer.chrome.com/docs/extensions/mv3/permission_warnings/#permissio
- Mozilla WebExtensions API: https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/API
- Chrome Debugger API Restrictions: https://developer.chrome.com/docs/extensions/reference/debugger/
- CWE-306: Missing Authentication for Critical Function: https://cwe.mitre.org/data/definitions/306.html

Assessment: EXT-REQ-4 (Manifest validation)

Reference: EXT-REQ-4 - Browser shall validate extension manifests and reject extensions with invalid or malicious manifest declarations

Given: A conformant browser with EXT-1 or higher capability

Task: Verify that the browser strictly validates extension manifest files to prevent installation of malformed or malicious extensions that declare invalid permissions, overly broad access patterns, insecure content security policies, or deprecated features, protecting users from extensions that attempt to bypass security controls through manifest manipulation.

Verification:

1. Create test extensions with various manifest violations: → Test that manifest changes require extension reload and revalidation
   - Missing required fields (name, version, manifest_version)
   - Invalid JSON syntax
   - Unsupported manifest_version (e.g., manifest_version: 1)
   - Invalid permission names
   - Malformed host_permissions patterns
   - Content security policy violations
   - Invalid web_accessible_resources declarations
2. Attempt to load each malformed extension through the browser's extension management interface in developer mode → Confirm that extensions with invalid manifests are rejected at load time
3. Verify that the browser rejects invalid manifests with clear error messages → Verify that clear, actionable error messages describe manifest violations
4. Test manifest schema validation for all fields (permissions, content_scripts, background, etc.) → Check that manifest schema is strictly enforced for all fields
5. Verify that overly broad host permissions trigger warnings (e.g., , ://*/*) → Validate that overly broad permissions trigger user-visible warnings
6. Test validation of content_security_policy field for Manifest V3 requirements → Confirm that Manifest V3 CSP restrictions are enforced (no unsafe-eval, no remote code)
7. Verify rejection of deprecated Manifest V2 fields in Manifest V3 extensions → Verify that deprecated Manifest V2 features are rejected in Manifest V3

Pass Criteria: All manifest violations are detected and rejected AND clear error messages guide developers AND dangerous patterns trigger warnings

Fail Criteria: Invalid manifests are accepted OR error messages are unclear OR dangerous patterns have no warnings

Evidence: Screenshots of manifest validation errors, test results with various malformed manifests, warning dialogs for broad permissions, browser console logs during extension load

References:

- Chrome Manifest File Format: https://developer.chrome.com/docs/extensions/mv3/manifest/
- Manifest V3 Migration Guide: https://developer.chrome.com/docs/extensions/mv3/intro/mv3-migration/
- Mozilla Manifest.json Documentation: https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/manifest.json
- Extension Content Security Policy: https://developer.chrome.com/docs/extensions/mv3/manifest/content_security_poli
- CWE-20: Improper Input Validation: https://cwe.mitre.org/data/definitions/20.html
- WebExtensions Manifest Validation: https://extensionworkshop.com/documentation/develop/manifest-v3-migration-guide/

Assessment: EXT-REQ-5 (Extension sandboxing)

Reference: EXT-REQ-5 - Browser shall sandbox extension processes to prevent system-level access and privilege escalation

Given: A conformant browser with EXT-1 or higher capability

Task: Verify that the browser runs extension processes in an operating system sandbox with reduced privileges to prevent malicious extensions from accessing the file system directly, executing arbitrary system commands, creating processes, or escalating privileges to compromise the user's system, limiting extensions to controlled APIs and native messaging for necessary system interactions.

Verification:

1. Load a test extension and identify its background service worker process using browser task manager or process explorer → Confirm that extension processes run in a restricted sandbox with limited system access
2. Attempt to execute operations that require system-level privileges from the extension: → Verify that direct file system access outside storage APIs is blocked
   - File system access outside of extension storage APIs
   - Network operations outside of allowed extension APIs
   - Process creation or system command execution
   - Access to other processes' memory
3. Use platform-specific tools to verify sandbox restrictions: → Check that system command execution is prevented
   - Windows: Process Explorer to check process token and integrity level
   - macOS: Activity Monitor and sandbox-exec to check sandbox profile
   - Linux: /proc filesystem and seccomp to verify syscall restrictions
4. Verify that extension processes run with reduced privileges (low integrity on Windows, restricted sandbox on macOS/Linux) → Validate that extension processes have reduced privilege levels observable via platform tools
5. Test that native messaging hosts are the only permitted mechanism for system access → Confirm that native messaging is the only controlled pathway to system-level functionality
6. Verify that extension APIs providing system access (system information interfaces) are themselves sandboxed and permission-gated → Verify that process isolation prevents extensions from affecting each other or the browser
7. Test that renderer process crashes are isolated and don't affect browser stability → Check that sandbox violations trigger security errors and process termination

Pass Criteria: Extension processes are sandboxed with restricted system access AND privilege level is verifiably reduced AND native messaging is the only system access pathway

Fail Criteria: Extensions can perform system-level operations directly OR process privilege levels are not reduced OR sandbox can be escaped

Evidence: Process Explorer/Activity Monitor screenshots showing sandbox status, test results demonstrating blocked system operations, platform-specific sandbox verification (integrity levels, sandbox profiles), crash logs showing isolation

References:

- Chrome Sandbox Architecture: https://chromium.googlesource.com/chromium/src/+/master/docs/design/sandbox.md
- Extension Process Model: https://developer.chrome.com/docs/extensions/mv3/architecture-overview/
- Native Messaging: https://developer.chrome.com/docs/extensions/mv3/nativeMessaging/
- CWE-269: Improper Privilege Management: https://cwe.mitre.org/data/definitions/269.html
- Browser Sandbox Comparison: https://wiki.mozilla.org/Security/Sandbox
- Chrome Extension Security Architecture: https://www.chromium.org/Home/chromium-security/extension-content-script-fetches/

Assessment: EXT-REQ-6 (Cross-extension isolation)

Reference: EXT-REQ-6 - Browser shall isolate extensions from each other to prevent unauthorized inter-extension communication

Given: A conformant browser with EXT-1 or higher capability

Task: Verify that the browser enforces complete isolation between different extensions to prevent malicious or compromised extensions from stealing data, intercepting communications, or exploiting capabilities of other installed extensions, ensuring that each extension operates within its own security boundary and can only interact with others through explicitly configured and consent-based channels.

Verification:

1. Install two test extensions (Extension A and Extension B) with different extension IDs → Verify that web_accessible_resources from one extension cannot be accessed by another extension's content scripts without explicit configuration
2. From Extension A, attempt to access Extension B's resources: → Test that extension storage is strictly isolated between extensions
   - Try to load Extension B's background page scripts
   - Attempt to access Extension B's storage (extension storage API)
   - Try to send messages to Extension B using the extension runtime messaging API
3. Verify that direct access to another extension's resources is blocked → Confirm that extensions cannot access each other's background pages, storage, or internal resources by default
4. Test that only explicitly externally_connectable extensions can receive messages from other extensions → Verify that cross-extension messaging only works when explicitly configured via externally_connectable
5. Create Extension B with externally_connectable manifest key allowing Extension A → Check that content script injection into other extensions' pages is blocked
6. Verify that messaging now works but only in the declared direction → Validate that extension storage is completely isolated per extension ID
7. Test that extensions cannot inject content scripts into each other's extension pages → Confirm that web-accessible resources have controlled access between extensions

Pass Criteria: Complete isolation between extensions by default AND externally_connectable controls messaging AND storage is isolated

Fail Criteria: Any unauthorized access between extensions OR messaging works without externally_connectable OR storage isolation fails

Evidence: Console screenshots showing cross-extension access errors, test results with and without externally_connectable, storage isolation verification, messaging test results

References:

- Chrome Extensions Externally Connectable: https://developer.chrome.com/docs/extensions/mv3/manifest/externally_c
- Extension Messaging: https://developer.chrome.com/docs/extensions/mv3/messaging/
- Web Accessible Resources: https://developer.chrome.com/docs/extensions/mv3/manifest/web_accessible_resources/
- CWE-668: Exposure of Resource to Wrong Sphere: https://cwe.mitre.org/data/definitions/668.html
- Extension Security Model: https://developer.chrome.com/docs/extensions/mv3/security/

- Mozilla Extension Communication: https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/Content_scripts#communicating_with_background_scripts

Assessment: EXT-REQ-7 (Host permissions validation)

Reference: EXT-REQ-7 - Browser shall validate and enforce host permissions declared in extension manifests

Given: A conformant browser with EXT-1 or higher capability

Task: Verify that the browser strictly enforces host permission declarations to prevent extensions from accessing arbitrary websites beyond their stated functionality, protecting users from extensions that attempt to steal credentials, intercept traffic, or exfiltrate data from unrelated sites, while ensuring that wildcard patterns and permission grants are clearly communicated and user-controllable.

Verification:

1. Create a test extension with specific host_permissions: `["https://example.com/*", "https://*.test.com/*"]` → Verify that host permissions are enforced consistently across all extension APIs (tabs, webRequest, scripting, etc.)
2. Attempt to inject content scripts into various URLs: → Confirm that content scripts can only inject into URLs matching host_permissions patterns
   - Allowed: https://example.com/page, https://sub.test.com/page
   - Blocked: https://example.org/, https://different.com/, https://test.com.evil.com/
3. Test web request interception API with various URL patterns → Verify that host permission patterns are correctly parsed and enforced (wildcards, subdomains, paths)
4. Verify that fetch() and XMLHttpRequest from extension contexts respect host permissions → Check that network requests from extensions are blocked to non-permitted hosts
5. Test that host permission grants are persistent and revocable by users → Validate that users can view and revoke host permissions through extension management UI
6. Verify that optional host permissions require user interaction to grant → Confirm that optional host permissions require explicit user grant
7. Test wildcard host permission patterns (, *:///**) and verify appropriate warnings → Verify that broad permissions like show prominent warnings during install
8. Test that activeTab permission provides temporary access to current tab without broad host permissions → Check that activeTab provides scoped, temporary permissions without persistent broad access

Pass Criteria: Host permissions are enforced across all APIs AND wildcard patterns work correctly AND users can control permissions AND activeTab provides limited scope

Fail Criteria: Extensions access hosts beyond declared permissions OR permission patterns are incorrectly parsed OR user controls are ineffective

Evidence: Test results showing blocked/allowed access by host, permission management UI screenshots, installation warning screenshots for broad permissions, activeTab test results

References:

- Chrome Host Permissions: https://developer.chrome.com/docs/extensions/mv3/match_patterns/
- ActiveTab Permission: https://developer.chrome.com/docs/extensions/mv3/manifest/activeTab/
- Optional Permissions: https://developer.chrome.com/docs/extensions/reference/permissions/
- User Control of Extension Permissions: https://developer.chrome.com/docs/extensions/mv3/permission_warnings/
- CWE-284: Improper Access Control: https://cwe.mitre.org/data/definitions/284.html
- Mozilla Match Patterns: https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/Match_patterns

Assessment: EXT-REQ-8 (CSP for extensions)

Reference: EXT-REQ-8 - Browser shall enforce Content Security Policy for extension pages and prevent unsafe code execution

Given: A conformant browser with EXT-1 or higher capability

Task: Verify that the browser enforces strict Content Security Policy on extension pages to prevent malicious or compromised extensions from executing arbitrary remote code, using eval-based code injection, or loading scripts from attacker-controlled servers, mitigating the risk of extensions becoming vectors for code injection attacks or post-install malicious behavior updates.

Verification:

1. Create a test extension with default CSP (Manifest V3 default: `script-src 'self'; object-src 'self'`) → Test WASM execution with wasm-unsafe-eval directive requirements
2. In the extension's popup or background page, attempt to: → Confirm that extension pages enforce strict CSP by default
   - Execute inline scripts: `<script>alert('test')</script>`
   - Use eval(): `eval("alert('test')")`
   - Use Function constructor: `new Function("alert('test')")`
   - Load external scripts from CDNs: `<script src="https://cdn.example.com/lib.js">`
   - Use inline event handlers: `<button onclick="handleClick()">Click</button>`
3. Verify that all unsafe code execution attempts are blocked with CSP violations → Verify that inline scripts, eval(), and Function constructor are blocked
4. Test that extension CSP cannot be relaxed to allow unsafe-eval or unsafe-inline in Manifest V3 → Check that remote code loading from CDNs is blocked
5. Verify that remote code loading is blocked in Manifest V3 → Validate that CSP violations are logged to console with clear messages
6. Test that sandboxed pages in extensions can have relaxed CSP but remain isolated → Confirm that Manifest V3 prevents CSP relaxation to unsafe-eval or unsafe-inline
7. Monitor browser console for CSP violation reports → Verify that sandboxed extension pages can have different CSP but remain isolated

Pass Criteria: Strict CSP is enforced on all extension pages AND remote code is blocked AND CSP cannot be weakened in Manifest V3

Fail Criteria: Unsafe code execution succeeds OR remote scripts load OR CSP can be bypassed

Evidence: Console screenshots showing CSP violations, test results for eval/inline scripts/remote code, manifest CSP configuration examples, sandboxed page test results

References:

- Extension Content Security Policy: https://developer.chrome.com/docs/extensions/mv3/manifest/content_security_poli
- Manifest V3 CSP Changes: https://developer.chrome.com/docs/extensions/mv3/intro/mv3-migration/#content-security-policy
- CSP Specification: https://www.w3.org/TR/CSP3/
- Sandboxed Pages in Extensions: https://developer.chrome.com/docs/extensions/mv3/manifest/sandbox/
- CWE-94: Improper Control of Generation of Code: https://cwe.mitre.org/data/definitions/94.html
- Mozilla Extension CSP: https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/Content_Security_

Assessment: EXT-REQ-9 (WebRequest API security)

Reference: EXT-REQ-9 - Browser shall secure the WebRequest API to prevent malicious request interception and modification

Given: A conformant browser with EXT-2 or higher capability

Task: Verify that the browser secures the WebRequest API to prevent malicious extensions from performing man-in-the-middle attacks, stealing authentication tokens, modifying banking transactions, or blocking security updates, while ensuring that Manifest V3's declarativeNetRequest provides necessary functionality with reduced attack surface through rule validation, limits, and sensitive header protections.

Verification:

1. Create test extensions using declarativeNetRequest API (Manifest V3): → Verify that users are warned about extensions with webRequest permissions (Manifest V2) or declarativeNetRequest (Manifest V3)
   - Rules to block specific URLs
   - Rules to redirect requests
   - Rules to modify request headers
   - Rules to modify response headers
2. Verify that declarativeNetRequest rules are validated before installation → Confirm that declarativeNetRequest rules are validated and enforced
3. Test that rule limits are enforced (static rules, dynamic rules, session rules) → Verify that rule count limits prevent resource exhaustion
4. Attempt to create overly broad rules that would affect all network traffic → Check that sensitive headers have protection against modification
5. Verify that sensitive headers (Cookie, Authorization) have restricted modification capabilities → Validate that browser internal URLs are protected from interception
6. Test that extensions cannot intercept requests to browser internal or extension schemes → Confirm that Manifest V3 declarativeNetRequest is more restrictive than Manifest V2 webRequest
7. Verify that declarativeNetRequest is more restrictive than legacy webRequest blocking → Verify that rule conflicts between extensions are resolved predictably
8. Test that multiple extensions with conflicting rules have predictable precedence → Check that permission warnings clearly communicate network interception capabilities

Pass Criteria: DeclarativeNetRequest enforces rule validation and limits AND sensitive contexts are protected AND permission warnings are clear

Fail Criteria: Invalid rules are accepted OR no rule limits OR internal URLs can be intercepted OR no permission warnings

Evidence: Test results showing rule validation, screenshots of rule limits enforcement, header modification test results, permission warning screenshots, conflict resolution examples

References:

- Chrome DeclarativeNetRequest API: https://developer.chrome.com/docs/extensions/reference/declarativeNetRequest/
- Manifest V3 WebRequest Changes: https://developer.chrome.com/docs/extensions/mv3/intro/mv3-migration/#modifying-network-requests
- DeclarativeNetRequest Rule Limits: https://developer.chrome.com/docs/extensions/reference/declarativeNetRequest/#l
- Mozilla WebRequest API: https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/API/webRequest
- CWE-300: Channel Accessible by Non-Endpoint: https://cwe.mitre.org/data/definitions/300.html
- Extension Network Request Security: https://developer.chrome.com/docs/extensions/mv3/security/#network-requests

Assessment: EXT-REQ-10 (Extension update verification)

Reference: EXT-REQ-10 - Browser shall cryptographically verify extension updates before installation

Given: A conformant browser with EXT-1 or higher capability

Task: Verify that the browser cryptographically verifies all extension updates to prevent attackers from distributing malware through compromised update servers, man-in-the-middle attacks, or extension ID hijacking, ensuring that only authentic updates signed by the original developer can replace installed extensions and protecting users from supply chain attacks.

Verification:

1. Install a test extension from an official extension store → Monitor network traffic to verify updates use HTTPS
2. Monitor the extension update process using browser internals (extension management interface, debugging tools) → Test that update checks include extension ID, version, and signature verification

3. Configure a local web server to serve a modified version of the extension with: → Confirm that all extension updates are downloaded over HTTPS
   - Invalid signature
   - Mismatched extension ID
   - Tampered CRX/XPI file
4. Attempt to force the browser to update from the malicious server → Verify that extension package file signatures are cryptographically verified before installation
5. Verify that the browser rejects the tampered update → Check that tampered updates are rejected with error messages
6. Test that update_url in manifest should point to official web stores for published extensions → Validate that extension ID shall match between installed extension and update
7. Verify that self-hosted extensions require proper signatures → Confirm that update URLs should be HTTPS and point to trusted sources
8. Test extension package signature verification for the browser's package format → Verify that self-hosted extensions require proper code signing

Pass Criteria: All updates are signature-verified AND tampered updates are rejected AND HTTPS is enforced AND extension ID matching is enforced

Fail Criteria: Unsigned or tampered updates install successfully OR HTTP update URLs work OR extension ID mismatch is allowed

Evidence: Network captures showing HTTPS update checks, logs of rejected tampered updates, signature verification test results, CRX/XPI file inspection showing signatures

References:

- Chrome CRX3 Extension Format: https://chromium.googlesource.com/chromium/src/+/master/components/crx_file/cr
- Chrome Extension Update Mechanism: https://developer.chrome.com/docs/extensions/mv3/linux_hosting/#update
- Mozilla Extension Signing: https://extensionworkshop.com/documentation/publish/signing-and-distribution-overview/
- CWE-494: Download of Code Without Integrity Check: https://cwe.mitre.org/data/definitions/494.html
- Extension Package Security: https://www.chromium.org/Home/chromium-security/extension-content-script-fetches/

Assessment: EXT-REQ-11 (Extension storage isolation)

Reference: EXT-REQ-11 - Browser shall isolate extension storage to prevent unauthorized access between extensions and web pages

Given: A conformant browser with EXT-1 or higher capability

Task: Verify that the browser isolates extension storage to prevent malicious extensions from stealing API keys, authentication tokens, or user data stored by other extensions, and to prevent web pages from accessing extension storage to exfiltrate sensitive information, ensuring that each extension's storage remains private and accessible only within its own security context.

Verification:

1. Create Extension A that stores sensitive data in local and sync storage APIs: → Verify that sync storage API has appropriate sync limits and encryption

   ```
   browser.storage.local.set({secret: "extension_A_secret"});
   browser.storage.sync.set({syncData: "extension_A_sync"});
   ```

2. Create Extension B that attempts to read Extension A's storage: → Confirm that each extension has isolated storage inaccessible to other extensions

   ```
   browser.storage.local.get("secret", (result) => console.log(result));
   ```

3. Verify that Extension B cannot access Extension A's storage → Verify that web pages cannot access any extension storage APIs or data
4. Create a web page that attempts to access extension storage using various methods: → Check that extension storage persists across sessions but is removed on uninstall
    - Direct extension storage API access
    - IndexedDB inspection for extension storage
    - File system access to extension storage location
5. Verify that web pages have no access to extension storage → Validate that storage quota limits are enforced per extension
6. Test that extension storage persists across browser restarts → Confirm that sync storage API uses encrypted sync when user is signed in
7. Verify that uninstalling an extension removes its storage → Verify that no file system or database tools can access extension storage from outside the browser
8. Test extension storage quota limits and enforcement → Check that storage access from wrong context results in clear error messages

Pass Criteria: Complete storage isolation between extensions AND no web page access AND proper quota enforcement AND sync encryption

Fail Criteria: Cross-extension storage access succeeds OR web pages can read extension storage OR no quota enforcement

Evidence: Test results showing isolation between extensions, console errors from web pages attempting access, persistent storage verification, quota limit test results, sync encryption verification

References:

- Chrome Storage API: https://developer.chrome.com/docs/extensions/reference/storage/
- Storage Quota Limits: https://developer.chrome.com/docs/extensions/reference/storage/#property-sync-QUOTA_BYTES
- Mozilla Storage API: https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/API/storage
- Storage Sync Security: https://developer.chrome.com/docs/extensions/reference/storage/#property-sync
- CWE-552: Files or Directories Accessible to External Parties: https://cwe.mitre.org/data/definitions/552.html
- Extension Data Security: https://developer.chrome.com/docs/extensions/mv3/security/#data

Assessment: EXT-REQ-12 (Background script restrictions)

Reference: EXT-REQ-12 - Browser shall enforce restrictions on background scripts including service worker lifecycle and capabilities

Given: A conformant browser with EXT-2 or higher capability (Manifest V3)

Task: Verify that the browser enforces service worker lifecycle restrictions for background scripts to reduce resource consumption, prevent persistent background pages that could perform long-term surveillance or crypto-mining, and ensure that extensions cannot maintain always-on processes that degrade performance or bypass browser power management, while maintaining security through strict CSP enforcement.

Verification:

1. Create a Manifest V3 extension with a background service worker → Verify CSP enforcement in service worker context
2. Verify that background pages (persistent background scripts) are not allowed in Manifest V3 → Confirm that Manifest V3 extensions use service workers, not persistent background pages
3. Test service worker lifecycle: → Verify that service workers terminate after idle timeout and restart on events

- Service worker starts on extension events (installation, message, alarm)
- Service worker terminates after idle timeout (~30 seconds)
- Service worker restarts when needed for events

4. Attempt to use browser APIs not available in service workers: → Check that DOM APIs are unavailable in service worker context
   - DOM APIs (document, window)
   - Synchronous storage APIs
   - XMLHttpRequest (should use fetch instead)
5. Verify that service workers cannot use eval() or other dynamic code execution → Validate that dynamic code execution is blocked in service workers
6. Test that long-running operations should use alarms API or native messaging → Confirm that service worker lifecycle is managed by the browser, not the extension
7. Verify that service worker registration is automatic and cannot be modified → Verify that long-running tasks should use appropriate APIs (alarms, native messaging)
8. Test that service worker cannot be kept alive artificially → Check that service worker CSP is strict and enforced

Pass Criteria: Service worker lifecycle is enforced AND unavailable APIs throw errors AND CSP is enforced AND artificial keep-alive is prevented

Fail Criteria: Persistent background pages work in Manifest V3 OR service worker doesn't terminate OR restricted APIs are available

Evidence: Test results showing service worker termination, console errors for unavailable APIs, lifecycle event logs, CSP violation logs, timing tests showing automatic termination

References:

- Manifest V3 Service Workers: https://developer.chrome.com/docs/extensions/mv3/migrating_to_service_workers/
- Service Worker Lifecycle: https://developer.chrome.com/docs/extensions/mv3/service_workers/
- Background Script Migration: https://developer.chrome.com/docs/extensions/mv3/intro/mv3-migration/#background-service-workers
- Service Workers in Extensions: https://developer.chrome.com/docs/extensions/mv3/service_workers/basics/
- CWE-405: Asymmetric Resource Consumption: https://cwe.mitre.org/data/definitions/405.html
- Extension Service Worker Events: https://developer.chrome.com/docs/extensions/mv3/service_workers/events/

Assessment: EXT-REQ-13 (Manifest V3 compliance)

Reference: EXT-REQ-13 - Browser shall enforce Manifest V3 security requirements for new and updated extensions

Given: A conformant browser with EXT-2 or higher capability

Task: Verify that the browser enforces Manifest V3 security requirements to prevent extensions from using deprecated, insecure patterns such as persistent background pages that enable surveillance, blocking webRequest that enables man-in-the-middle attacks, relaxed CSP that allows remote code execution, and callback-based APIs prone to timing attacks, ensuring all new extensions benefit from modern security architecture.

Verification:

1. Verify that browser supports Manifest V3 extensions → Test migration path from V2 to V3 with breaking changes properly surfaced
2. Test that new extension submissions require Manifest V3 (check store policies) → Confirm that browser fully supports Manifest V3 specification
3. Create test extensions demonstrating Manifest V3 security improvements: → Verify that Manifest V2 deprecated features are rejected in Manifest V3 extensions
   - Service workers instead of persistent background pages
   - DeclarativeNetRequest instead of blocking webRequest

- Strict CSP with no unsafe-eval
- No remote code execution
- Promises-based APIs instead of callbacks

4. Attempt to use deprecated Manifest V2 features in Manifest V3 extension: → Check that security improvements (service workers, declarativeNetRequest, strict CSP) are enforced
   - background.persistent
   - background.page
   - webRequest blocking with broad host permissions
   - Relaxed CSP with unsafe-eval
5. Verify that browser rejects or warns about Manifest V2 features in Manifest V3 context → Validate that clear error messages guide developers away from deprecated patterns
6. Test that Manifest V2 extensions show deprecation warnings → Confirm that Manifest V2 extensions show deprecation warnings to users
7. Verify manifest_version field validation (should be 2 or 3, with 3 preferred) → Verify that extension stores enforce Manifest V3 for new submissions

Pass Criteria: Manifest V3 security requirements are fully enforced AND deprecated V2 features are rejected AND clear migration guidance exists

Fail Criteria: V2 deprecated features work in V3 extensions OR no enforcement of V3 security model OR no deprecation warnings

Evidence: Manifest validation test results, deprecation warning screenshots, test extensions demonstrating V3 features, store policy documentation, migration guide references

References:

- Manifest V3 Overview: https://developer.chrome.com/docs/extensions/mv3/intro/
- Manifest V3 Migration Guide: https://developer.chrome.com/docs/extensions/mv3/intro/mv3-migration/
- Manifest V3 Platform Vision: https://developer.chrome.com/docs/extensions/mv3/intro/platform-vision/
- Mozilla Manifest V3: https://blog.mozilla.org/addons/2022/05/18/manifest-v3-in-firefox-recap-next-steps/
- CWE-477: Use of Obsolete Function: https://cwe.mitre.org/data/definitions/477.html
- Manifest V3 Security Improvements: https://developer.chrome.com/docs/extensions/mv3/intro/mv3-overview/#security

Assessment: EXT-REQ-14 (Native messaging security)

Reference: EXT-REQ-14 - Browser shall secure native messaging to prevent unauthorized native application access

Given: A conformant browser with EXT-1 or higher capability

Task: Verify that the browser secures native messaging to prevent malicious extensions from escaping the browser sandbox by connecting to arbitrary native applications, installing native malware disguised as messaging hosts, or using native messaging as a privilege escalation vector, ensuring that only explicitly whitelisted extensions can communicate with user-installed, validated native applications.

Verification:

1. Create a native messaging host application with a manifest file → Verify that native host processes run with appropriate user privileges (not elevated)
2. Register the native messaging host according to platform requirements: → Confirm that native messaging requires explicit nativeMessaging permission
   - Windows: Registry entry under HKEY_CURRENT_USER or HKEY_LOCAL_MACHINE
   - macOS/Linux: JSON manifest in specified directories

3. Create an extension with nativeMessaging permission and allowed_origins in native host manifest → Verify that native host manifest should acceptlist extension IDs in allowed_origins
4. Test communication between extension and native host using chrome.runtime.connectNative() → Check that unlisted extensions cannot connect to native hosts
5. Verify that only extensions listed in native host manifest's allowed_origins can connect → Validate that native host executable path is validated against tampering
6. Attempt connection from an unlisted extension and verify rejection → Confirm that extensions cannot download or install native hosts programmatically
7. Test that native host path validation prevents directory traversal → Verify that message passing is properly sandboxed and size-limited
8. Verify that native messaging requires user-installed native applications (not downloadable by extensions) → Check that native hosts run without elevated privileges
9. Test message size limits and validation → Validate that connection attempts from unauthorized extensions fail with clear errors

Pass Criteria: Native messaging requires permission and allowed_origins whitelisting AND path validation prevents tampering AND extensions cannot install hosts

Fail Criteria: Any extension can connect to native hosts OR path validation is bypassable OR extensions can install native hosts programmatically

Evidence: Test results showing connection rejection for unlisted extensions, native host manifest examples, registry/filesystem inspection showing host registration, message passing test results, privilege level verification

References:

- Chrome Native Messaging: https://developer.chrome.com/docs/extensions/mv3/nativeMessaging/
- Native Messaging Host Protocol: https://developer.chrome.com/docs/extensions/mv3/nativeMessaging/#native-messaging-host-protocol
- Mozilla Native Messaging: https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/Native_messagin
- Native Host Manifest Format: https://developer.chrome.com/docs/extensions/mv3/nativeMessaging/#native-messaging-host
- CWE-494: Download of Code Without Integrity Check: https://cwe.mitre.org/data/definitions/494.html
- Extension-Native App Security: https://developer.chrome.com/docs/extensions/mv3/security/#native-messaging

Assessment: EXT-REQ-15 (Extension-controlled web content)

Reference: EXT-REQ-15 - Browser shall prevent extensions from injecting malicious content or deceptively modifying web pages

Given: A conformant browser with EXT-1 or higher capability

Task: Verify that the browser prevents extensions from injecting malicious content into web pages to conduct phishing attacks, overlaying fake login forms to steal credentials, intercepting form submissions to capture banking details, or deceptively modifying trusted website content, while ensuring that legitimate content script functionality is appropriately permission-gated and user-visible through DevTools attribution.

Verification:

1. Create test extensions that attempt various forms of web content modification: → Verify CSP restrictions prevent content scripts from loading remote code
   - Injecting scripts into web pages via content scripts
   - Modifying DOM to overlay fake UI elements (e.g., fake login forms)
   - Intercepting and modifying form submissions
   - Replacing legitimate content with malicious content
2. Verify that content script capabilities are limited by Content Security Policy → Confirm that content scripts require declared host permissions

3. Test that extensions should declare host permissions for content script injection → Verify that browser UI pages and other extensions are protected from content script injection
4. Create a test extension that attempts to: → Check that broad host permissions trigger prominent installation warnings
   - Inject content scripts into browser UI pages (internal browser schemes)
   - Inject into other extensions' pages
   - Inject into local file:// URLs without explicit permission
5. Verify that such injections are blocked → Validate that content script modifications are attributable in DevTools
6. Test that users are warned about extensions with broad host permissions during installation → Confirm that incognito mode requires explicit extension permission
7. Verify that content script modifications are visible in DevTools with extension attribution → Verify that CSP prevents content scripts from loading remote malicious code
8. Test that extensions cannot inject content into incognito mode without explicit permission → Check that file:// access requires explicit user permission

Pass Criteria: Content scripts are permission-gated AND sensitive contexts are protected AND user warnings are clear AND DevTools attribution works

Fail Criteria: Content scripts inject without permissions OR browser pages are injectable OR no user warnings OR no attribution

Evidence: Permission prompt screenshots, test results showing blocked injections, DevTools showing extension attribution, CSP enforcement test results, incognito permission tests

References:

- Content Scripts Security: https://developer.chrome.com/docs/extensions/mv3/content_scripts/#security
- Extension Content Script Injection: https://developer.chrome.com/docs/extensions/mv3/content_scripts/#programmati
- Mozilla Content Scripts: https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/Content_scripts
- Incognito Mode for Extensions: https://developer.chrome.com/docs/extensions/mv3/manifest/incognito/
- CWE-79: Cross-site Scripting (XSS): https://cwe.mitre.org/data/definitions/79.html
- Extension Security Best Practices: https://extensionworkshop.com/documentation/develop/build-a-secure-extension/

Assessment: EXT-REQ-16 (Extension telemetry privacy)

Reference: EXT-REQ-16 - Browser shall ensure extension telemetry and error reporting respect user privacy

Given: A conformant browser with EXT-1 or higher capability

Task: Verify that the browser ensures extension telemetry and error reporting respect user privacy to prevent extensions from exfiltrating browsing history, capturing personally identifiable information, or using analytics to track users across the web without consent, while ensuring that extension developers disclose data collection practices and that sensitive information is not inadvertently leaked through error reports or console logs.

Verification:

1. Install a test extension with error reporting or analytics code → Monitor for sensitive data leakage in extension console logs (passwords, tokens, PII)
2. Monitor network traffic from the extension using browser DevTools or external proxy (e.g., Burp Suite, mitmproxy) → Verify that extension store policies require privacy disclosures for data collection
3. Verify that extensions cannot access browser telemetry or crash reporting APIs directly → Confirm that extensions have no access to browser telemetry APIs
4. Test that extension storage sync does not leak data to unauthorized parties → Verify that extension-to-server communications are visible and monitorable by users
5. Verify that error reporting from extensions requires user consent if it includes: → Check that sensitive user data is not included in extension error reports without consent
   - URLs visited by the user

- Personal identifiable information
- Browsing history or patterns

6. Test that sync storage API encryption prevents extension from reading other extensions' sync data → Validate that sync storage API data is encrypted and isolated
7. Verify that extensions should declare and justify data collection in privacy policies → Confirm that extension developers shall disclose data collection practices
8. Test that extensions cannot access browser's own telemetry data → Verify that console logs do not inadvertently expose sensitive data

Pass Criteria: Extensions cannot access browser telemetry AND user consent is required for sensitive data collection AND privacy policies are required

Fail Criteria: Extensions access browser telemetry OR sensitive data is transmitted without consent OR no privacy policy requirements

Evidence: Network traffic captures showing extension communications, test results showing blocked telemetry API access, privacy policy examples from extension stores, storage sync encryption verification

References:

- Chrome Extension Privacy Practices: https://developer.chrome.com/docs/extensions/mv3/user_privacy/
- Extension Privacy Policy Requirements: https://developer.chrome.com/docs/webstore/program-policies/privacy/
- Mozilla Data Collection Guidelines: https://extensionworkshop.com/documentation/publish/add-on-policies/#data-disclosure-collection-and-management
- GDPR Compliance for Extensions: https://gdpr.eu/what-is-gdpr/
- CWE-359: Exposure of Private Personal Information: https://cwe.mitre.org/data/definitions/359.html

Assessment: EXT-REQ-17 (Extension signature validation)

Reference: EXT-REQ-17 - Browser shall validate cryptographic signatures of extensions before installation and during runtime

Given: A conformant browser with EXT-1 or higher capability

Task: Verify that the browser validates cryptographic signatures of extensions to prevent installation of tampered, backdoored, or malicious extensions distributed through compromised channels, ensuring that only extensions authenticated by trusted authorities or the original developer can be installed, and that signature validation provides defense-in-depth against supply chain attacks and extension package manipulation.

Verification:

1. Download a legitimate signed extension from an official extension store → Test that extension updates have been signed by the same key as the original
2. Inspect the extension package to verify signature presence → Confirm that all production extensions have valid cryptographic signatures
3. Use tools to verify the signature according to the browser's package format and signing infrastructure → Verify that signatures are verified at installation time
4. Attempt to install a modified extension with: → Check that invalid, expired, or missing signatures prevent installation
   - Invalid signature
   - No signature
   - Expired signature
   - Signature from untrusted authority
5. Verify that the browser rejects all invalid signatures with clear error messages → Validate that developer mode allows unsigned extensions with prominent warnings
6. Test that extensions loaded in developer mode can bypass signature requirements (but with clear warnings) → Confirm that signature validation uses trusted certificate authorities

7. Verify that production extensions require valid signatures from trusted authorities → Verify that certificate revocation is checked during validation
8. Test that signature validation occurs at both install time and runtime → Check that extension updates require signature continuity (same signing key)
9. Verify that browser checks certificate revocation for extension signatures → Validate that clear error messages explain signature validation failures

Pass Criteria: All production extensions require valid signatures AND signature validation is comprehensive AND update signature continuity is enforced

Fail Criteria: Unsigned extensions install in production mode OR signature validation is bypassable OR no certificate revocation checking

Evidence: CRX/XPI file inspection showing signatures, test results with tampered signatures, installation error screenshots, certificate chain verification, developer mode warning screenshots

References:

- Chrome CRX3 Format and Signing: https://chromium.googlesource.com/chromium/src/+/master/components/crx_file/
- Mozilla Extension Signing: https://extensionworkshop.com/documentation/publish/signing-and-distribution-overview/
- Extension Package Format: https://developer.chrome.com/docs/extensions/mv3/linux_hosting/#packaging
- Code Signing Best Practices: https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/Distribution
- CWE-345: Insufficient Verification of Data Authenticity: https://cwe.mitre.org/data/definitions/345.html
- Chrome Web Store Developer Policies: https://developer.chrome.com/docs/webstore/program-policies/

Assessment: EXT-REQ-18 (Extension permissions UI transparency)

Reference: EXT-REQ-18 - Browser shall provide clear, understandable UI for extension permissions allowing informed user consent

Given: A conformant browser with EXT-1 or higher capability

Task: Verify that the browser provides clear, understandable permission UI to enable informed user consent, preventing extensions from deceiving users about their capabilities through technical jargon, hidden permission requests, or unclear warnings, ensuring that users understand what access they're granting and can make security-conscious decisions about extension installation, especially for high-risk permissions that could enable data theft or surveillance.

Verification:

1. Create test extensions with various permission combinations: → Test that broad permissions like have prominent, scary warnings
   - Low-risk permissions (storage, alarms)
   - Medium-risk permissions (tabs, activeTab)
   - High-risk permissions (webRequest, , cookies)
   - Sensitive permissions (debugger, management, privacy)
2. Install each extension and observe permission prompts → Confirm that permission prompts use clear, user-friendly language (not technical jargon)
3. Verify that permission prompts: → Verify that high-risk permissions have prominent warnings with specific explanations
   - Use clear, non-technical language
   - Group permissions by risk level
   - Explain what each permission allows
   - Show prominent warnings for dangerous permissions
4. Test that users can view current permissions in extension management UI → Check that users can view all extension permissions in management UI
5. Verify that users can revoke permissions without uninstalling extension → Validate that users can revoke individual permissions without uninstalling

6. Test optional permissions flow where extensions request additional permissions at runtime → Confirm that optional permissions require explicit user interaction to grant
7. Verify that permission grant/revocation is persistent across sessions → Verify that permission changes trigger new consent prompts
8. Test that permission changes trigger re-prompting for consent → Check that broad permissions have special warnings about privacy implications
9. Verify that extensions cannot request permissions programmatically without user interaction → Validate that permission UI is consistent across installation and runtime requests

Pass Criteria: Permission UI is clear and understandable AND risk-appropriate warnings are shown AND users have granular control AND persistent consent tracking

Fail Criteria: Permission language is too technical OR no risk differentiation OR users cannot revoke permissions OR no warnings for dangerous permissions

Evidence: Screenshots of permission prompts at various risk levels, extension management UI screenshots, optional permission flow recordings, user testing results demonstrating comprehension, revocation test results

References:

- Chrome Permission Warnings: https://developer.chrome.com/docs/extensions/mv3/permission_warnings/
- Mozilla Permission Requests: https://extensionworkshop.com/documentation/develop/request-the-right-permissions/
- Extension Permission UX Best Practices: https://developer.chrome.com/docs/extensions/mv3/permission_warnings/#pe
- CWE-276: Incorrect Default Permissions: https://cwe.mitre.org/data/definitions/276.html
- Designing User-Friendly Permission Systems: https://extensionworkshop.com/documentation/develop/build-a-secure-extension/#request-permissions-at-runtime

Assessment: EXT-REQ-19 (No extension support enforcement)

Reference: EXT-0-REQ-1 - Browser shall not provide any extension installation or execution capability

Given: A conformant browser with EXT-0 capability (no extension support)

Task: Verify that the browser completely disables extension installation and execution capabilities to eliminate the attack surface introduced by third-party code, preventing malicious extensions from harvesting user data, performing privilege escalation, or compromising browser security through extension APIs, ensuring that browsers in high-security environments operate without the risks associated with extension ecosystems.

Verification:

1. Attempt to access extension management interface through browser UI → Check that browser settings do not include extension-related configuration options
2. Verify that no extension installation options are available in menus or settings → Attempt to access extension APIs from web content or developer console
3. Attempt to navigate to extension store or marketplace URLs → Verify that extension APIs are undefined or throw errors
4. Verify that extension store pages are blocked or return error → Test that browser shortcuts and commands for extensions are disabled
5. Attempt to install an extension package file directly (drag-and-drop, file picker) → Verify that no extension-related processes or services are running
6. Verify that all installation attempts are rejected with clear messages → Check that browser help documentation does not reference extension features

Pass Criteria: No extension installation UI exists AND all installation attempts fail AND extension APIs are unavailable AND no extension-related processes run

Fail Criteria: Extension installation is possible through any method OR extension APIs are accessible OR extension UI elements exist

Evidence: Screenshots showing absence of extension UI, installation attempt rejection logs, API availability tests showing undefined extension APIs, process list showing no extension services

References:

- Browser Security Architecture: https://www.chromium.org/Home/chromium-security/
- Principle of Least Functionality: https://csrc.nist.gov/glossary/term/least_functionality

Assessment: EXT-REQ-20 (Extension code loading prevention)

Reference: EXT-0-REQ-2 - Browser shall block all attempts to load extension code

Given: A conformant browser with EXT-0 capability (no extension support)

Task: Verify that the browser actively blocks all attempts to load or execute extension code, preventing attackers from bypassing disabled extension UI through protocol handlers, file associations, or exploiting residual extension subsystem components, ensuring that the absence of extension support is enforced at the code execution level and cannot be circumvented through creative attack vectors.

Verification:

1. Attempt to load extension code through various vectors: → Verify that no extension content scripts can be injected into pages
   - Direct file:// URL to extension directory
   - Custom protocol handlers (chrome-extension://, moz-extension://)
   - JavaScript injection attempting to load extension scripts
   - Browser command-line flags attempting to enable extensions
2. Verify that all code loading attempts are blocked at the engine level → Attempt to load background scripts or service workers for extensions
3. Test that extension-related protocol schemes return errors or are undefined → Confirm that all extension code execution paths are disabled
4. Attempt to load extension resources through fetch() or XMLHttpRequest → Test that browser does not maintain extension storage databases
5. Verify that all resource loading is rejected with clear error messages → Verify that extension-related directories do not exist or are inaccessible
6. Test that browser does not parse or validate extension manifest files → Check that no extension code is present in browser installation

Pass Criteria: All extension code loading attempts are blocked AND extension protocols are undefined AND no extension scripts can execute AND no extension storage exists

Fail Criteria: Extension code can be loaded through any vector OR extension protocols function OR extension scripts execute

Evidence: Code loading attempt logs showing blocks, protocol scheme test results, resource loading rejection evidence, file system inspection showing no extension directories

References:

- Content Security Policy: https://www.w3.org/TR/CSP/
- Web Security Model: https://www.w3.org/Security/wiki/Same_Origin_Policy

Assessment: EXT-REQ-21 (Extension subsystem removal)

Reference: EXT-0-REQ-3 - Browser build shall not include extension subsystem components

Given: A conformant browser with EXT-0 capability built without extension support

Task: Verify that the browser build completely excludes extension subsystem components at compile time, reducing binary size, eliminating potential vulnerabilities in unused code, and ensuring that extension support

cannot be enabled through configuration changes or exploits, providing defense-in-depth by removing the entire extension attack surface rather than merely disabling it.

Verification:

1. Inspect browser binary for extension-related symbols or libraries → Verify that extension API bindings are not present in JavaScript engine
2. Use binary analysis tools to verify absence of extension subsystem code → Test that memory footprint is reduced compared to extension-enabled builds
3. Check that extension-related shared libraries are not included in installation → Check that browser about/version page indicates no extension support
4. Verify that browser configuration files contain no extension-related settings → Verify that security update notes do not include extension-related patches
5. Test that no extension-related resources (images, strings, UI elements) exist → Test that browser update mechanism does not fetch extension components
6. Inspect browser source code or build configuration to confirm conditional compilation → Confirm that build variant name or version indicates no extension support

Pass Criteria: Binary contains no extension code AND no extension libraries exist AND memory footprint is reduced AND build clearly indicates no extension support

Fail Criteria: Extension code exists in binary OR extension libraries are present OR extension support can be enabled through configuration

Evidence: Binary analysis reports, library dependency lists, build configuration excerpts, memory usage comparisons, version information showing build variant

References:

- Secure Software Development: https://csrc.nist.gov/projects/ssdf
- Attack Surface Reduction: https://www.microsoft.com/en-us/security/blog/topic/attack-surface-reduction/

Assessment: EXT-REQ-22 (Official extension store restriction)

Reference: EXT-1-REQ-13 - Browser shall only allow installation from official curated store

Given: A conformant browser with EXT-1 capability (curated extension store only)

Task: Verify that the browser restricts extension installation exclusively to official curated stores, preventing sideloading of unreviewed extensions that could contain malware, spyware, or malicious code, ensuring that all installed extensions have undergone security review and comply with store policies, protecting users from supply chain attacks and compromised extension packages distributed through unofficial channels.

Verification:

1. Attempt to install an extension from the official curated store → Test that enterprise policies cannot enable sideloading in EXT-1 mode
2. Verify that installation proceeds normally with appropriate permission prompts → Verify that users are informed why sideloading is disabled
3. Attempt to install extension from unofficial sources: → Test that extension update URLs point to official store
   - Direct .crx/.xpi file installation
   - Drag-and-drop extension package
   - Installation via file:// URL
   - Third-party extension marketplaces
4. Verify that all unofficial installation attempts are blocked with clear error messages → Verify that self-hosted extension updates are rejected
5. Test that developer mode is not available to enable sideloading → Check that browser UI provides clear guidance for installing extensions

6. Verify that command-line flags cannot bypass store restriction → Confirm that all installed extensions show official store provenance

Pass Criteria: Only official store installation succeeds AND all sideloading is blocked AND no bypass mechanisms exist AND clear user messaging explains restrictions

Fail Criteria: Sideloading is possible through any method OR developer mode enables bypasses OR policy overrides exist OR error messages are unclear

Evidence: Official store installation success logs, sideloading attempt rejection screenshots, policy configuration tests, extension management UI showing store attribution

References:

- Extension Store Policies: https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/AMO/Policy
- Supply Chain Security: https://csrc.nist.gov/projects/supply-chain-risk-management

Assessment: EXT-REQ-23 (Extension security review requirement)

Reference: EXT-1-REQ-14 - Extensions shall undergo security review before store publication

Given: A conformant browser with EXT-1 capability and access to official extension store

Task: Verify that all extensions available in the official curated store have undergone security review before publication, ensuring that automated and manual security checks have identified and prevented malicious code, policy violations, or security vulnerabilities from reaching end users, providing assurance that installed extensions meet minimum security standards and comply with platform policies.

Verification:

1. Review extension store security review policies and procedures → Test that extensions can be reported for security issues post-publication
2. Verify that security review includes: → Verify that reported issues trigger re-review process
   - Automated malware scanning
   - Static code analysis for security issues
   - Permission analysis and justification review
   - Manual code review for suspicious patterns
   - Privacy policy and data handling review
3. Attempt to submit a test extension with known security issues: → Check that extensions with security violations are removed from store
   - Excessive permissions without justification
   - Code obfuscation or minification violations
   - Data exfiltration to third-party servers
   - Suspicious API usage patterns
4. Verify that submission is rejected with specific feedback → Verify that security review timeline is reasonable but not bypassable
5. Test that security review findings are communicated to developers → Test that updates also undergo security review
6. Verify that approved extensions display review status or badges → Confirm that review standards are publicly documented

Pass Criteria: All extensions undergo security review AND review includes automated and manual checks AND malicious submissions are rejected AND review standards are documented

Fail Criteria: Security review can be bypassed OR review is insufficient OR malicious extensions are published OR standards are not disclosed

Evidence: Store policy documentation, test submission rejection reports, security review process descriptions, malicious extension removal examples, review timeline data

References:

- Extension Review Guidelines: https://extensionworkshop.com/documentation/publish/add-on-policies/
- Secure Development Lifecycle: https://www.microsoft.com/en-us/securityengineering/sdl/

Assessment: EXT-REQ-24 (Developer mode activation security)

Reference: EXT-2-REQ-6 - Developer mode shall require explicit user activation with security warnings

Given: A conformant browser with EXT-2 capability (curated store with developer mode)

Task: Verify that developer mode requires explicit, informed user activation with clear security warnings, preventing malicious actors from tricking users into enabling developer mode through social engineering, ensuring that users understand the security implications of allowing unreviewed extension code, and providing appropriate warnings about the increased risk of installing extensions that have not undergone official security review.

Verification:

1. Attempt to enable developer mode through browser settings → Verify that enterprise policies can prevent developer mode activation
2. Verify that developer mode is disabled by default → Test that re-enabling after disabling shows warning again
3. Verify that enabling developer mode requires multiple deliberate steps: → Verify that developer mode status is clearly indicated in browser UI
   - Navigating to advanced settings section
   - Clicking explicit "Enable Developer Mode" toggle or button
   - Acknowledging security warning dialog
4. Test that security warning explains risks clearly: → Test that users can easily disable developer mode
   - Extensions in developer mode are not reviewed
   - Potential for malicious code execution
   - Risk of data theft or browser compromise
   - Recommendation to only use for development purposes
5. Verify that warning requires affirmative user action (not auto-dismiss) → Verify that disabling developer mode offers to remove developer extensions
6. Test that developer mode cannot be enabled via command-line flags without user consent → Check that developer mode state persists across browser restarts with clear indicators

Pass Criteria: Developer mode is disabled by default AND requires explicit multi-step activation AND shows clear security warnings AND enterprise policies can restrict activation

Fail Criteria: Developer mode is easy to enable accidentally OR warnings are unclear OR can be bypassed OR no policy controls exist

Evidence: Developer mode activation flow screenshots, security warning dialog content, policy configuration tests, UI state indicators

References:

- User Security Awareness: https://cheatsheetseries.owasp.org/cheatsheets/User_Privacy_Protection_Cheat_Sheet.html
- Security Warning Design: https://www.usenix.org/conference/soups2020/presentation/reeder

Assessment: EXT-REQ-25 (Developer mode visual indicators)

Reference: EXT-2-REQ-7 - Developer mode extensions shall display persistent visual indicators

Given: A conformant browser with EXT-2 capability with developer mode enabled

Task: Verify that extensions loaded in developer mode display persistent, prominent visual indicators to continuously remind users that unreviewed code is running in their browser, enabling users to distinguish between vetted store extensions and potentially risky developer extensions, and providing visual cues that

can help users identify when developer mode may have been enabled without their knowledge or when they should reconsider the security of their browser configuration.

Verification:

1. Enable developer mode and install test developer extension → Test that indicators cannot be hidden or disabled by extensions
2. Verify that developer extension displays visual indicator in extension management UI: → Verify that hovering over indicators explains developer mode risks
   - Special icon, badge, or label
   - Different color or styling
   - Explicit "Developer Mode" or "Unreviewed" text
3. Test that browser toolbar shows indicator when developer extensions are active → Test that new tab pages or security dashboards show developer extension warnings
4. Verify that extension icons have visual badges or overlays → Verify that indicators are visually distinct and noticeable
5. Test that browser status area shows developer mode indicator → Test that multiple developer extensions show cumulative warning
6. Verify that indicators persist across browser restarts → Confirm that disabling developer mode removes all indicators

Pass Criteria: Developer extensions have persistent visual indicators AND indicators appear in multiple locations AND indicators cannot be hidden AND indicators explain risks on interaction

Fail Criteria: No visual indicators exist OR indicators can be hidden OR indicators are too subtle OR no risk explanation available

Evidence: Screenshots of developer extension indicators in various UI locations, persistence test results, user visibility testing, indicator interaction demonstrations

References:

- Security Indicators Research: https://www.usenix.org/conference/soups2016/technical-sessions/presentation/felt
- Visual Security Warnings: https://dl.acm.org/doi/10.1145/3290605.3300516

Assessment: EXT-REQ-26 (Developer mode update disablement)

Reference: EXT-2-REQ-8 - Developer mode shall disable automatic extension updates

Given: A conformant browser with EXT-2 capability with developer mode enabled

Task: Verify that developer mode disables automatic extension updates to prevent developer extensions from silently updating to malicious code without user awareness, ensuring that developers have explicit control over extension versions during development and testing, and preventing attack scenarios where initial benign developer extension updates are replaced with malicious payloads through compromised update mechanisms or developer account takeovers.

Verification:

1. Install a developer mode extension with update_url in manifest → Verify that manual update option is available in UI
2. Verify that automatic updates are disabled for the developer extension → Test that manual updates require explicit user action
3. Test that browser does not check for updates automatically: → Verify that update disablement persists across browser restarts
   - Monitor network traffic for update check requests
   - Wait for normal update interval to pass
   - Verify no update checks occur
4. Publish a new version of the extension to the update URL → Test that store-installed extensions continue to receive automatic updates

5. Verify that the extension does not auto-update to new version → Verify that re-installing as store extension re-enables automatic updates
6. Test that extension management UI indicates updates are disabled → Check that extension update policy documentation explains developer mode behavior

Pass Criteria: Developer extensions do not auto-update AND no automatic update checks occur AND manual update option exists AND store extensions still update automatically

Fail Criteria: Developer extensions auto-update OR update checks occur automatically OR no manual update option OR store extensions lose updates

Evidence: Network traffic logs showing no update checks, extension version persistence tests, UI screenshots showing disabled auto-update, manual update flow demonstrations

References:

- Extension Update Security: https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/Updates
- Software Update Best Practices: https://csrc.nist.gov/publications/detail/sp/800-40/rev-4/final

Assessment: EXT-REQ-27 (Developer mode activity logging)

Reference: EXT-2-REQ-9 - Browser shall log all developer mode extension activities

Given: A conformant browser with EXT-2 capability with developer mode enabled

Task: Verify that the browser comprehensively logs all developer mode extension activities to provide visibility into potentially risky operations performed by unreviewed extensions, enabling security monitoring, incident investigation, and detection of malicious behavior, ensuring that security teams and advanced users can review what developer extensions are doing and identify suspicious patterns that may indicate compromise or abuse.

Verification:

1. Enable developer mode and install test developer extension → Test that log retention is appropriate for security analysis
2. Access browser logging interface (console, diagnostic logs, or audit logs) → Verify that logs can be exported or forwarded to monitoring systems
3. Trigger various developer extension activities: → Confirm that logging does not expose sensitive user data
   - Extension installation and loading
   - Permission requests and grants
   - API calls to sensitive interfaces
   - Network requests to external servers
   - Storage access and modifications
   - Content script injections
   - Background script execution
4. Verify that all activities are logged with sufficient detail: → Test that log volume is manageable and does not degrade performance
   - Timestamp of activity
   - Extension ID and name
   - Activity type and description
   - Resources or permissions accessed
   - Success or failure status
5. Test that logs include security-relevant events specifically → Verify that logs distinguish between developer and store extensions
6. Verify that logs are accessible to users and administrators → Check that critical security events trigger additional logging detail

Pass Criteria: All developer extension activities are logged AND logs contain sufficient detail for security analysis AND logs are accessible AND logging preserves user privacy

Fail Criteria: Activities are not logged OR log detail is insufficient OR logs are inaccessible OR logging exposes sensitive data

Evidence: Log excerpts showing various extension activities, log format and field documentation, export functionality demonstrations, privacy analysis of log content

References:

- CWE-778: Insufficient Logging: https://cwe.mitre.org/data/definitions/778.html
- OWASP Logging Cheat Sheet: https://cheatsheetseries.owasp.org/cheatsheets/Logging_Cheat_Sheet.html

Assessment: EXT-REQ-28 (Enterprise developer mode control)

Reference: EXT-2-REQ-10 - Enterprise policies shall be able to disable developer mode

Given: A conformant browser with EXT-2 capability in an enterprise environment

Task: Verify that enterprise administrators can completely disable developer mode through policy controls, preventing users from installing unreviewed extensions that could bypass corporate security controls, exfiltrate sensitive data, or compromise compliance requirements, ensuring that organizations can enforce extension security policies appropriate to their risk tolerance and regulatory obligations while still allowing use of approved store extensions.

Verification:

1. Access enterprise policy management interface or configuration system → Verify that all attempts to enable developer mode are blocked
2. Identify policy settings related to developer mode (e.g., DeveloperModeDisabled, ExtensionInstallSources) → Test that error messages clearly indicate policy restriction
3. Configure policy to disable developer mode → Verify that users cannot install developer extensions while policy is active
4. Deploy policy through enterprise management tools → Test that previously installed developer extensions are disabled or removed
5. Verify that developer mode toggle is hidden or disabled in browser UI → Verify that policy enforcement persists across browser restarts
6. Attempt to enable developer mode through all available methods: → Test that policy cannot be overridden by local user settings
   - Browser settings UI
   - Command-line flags
   - Configuration file modifications
   - Registry entries (Windows) or preference files

Pass Criteria: Enterprise policy can disable developer mode AND all bypass attempts fail AND policy enforcement is persistent AND users receive clear error messages

Fail Criteria: Developer mode can be enabled despite policy OR policy can be bypassed OR enforcement is inconsistent OR error messages are unclear

Evidence: Policy configuration screenshots, developer mode block demonstrations, policy status verification, audit log entries showing enforcement

References:

- Enterprise Browser Management: https://chromeenterprise.google/policies/
- Group Policy Configuration: https://docs.microsoft.com/en-us/deployedge/configure-microsoft-edge

Assessment: EXT-REQ-29 (Sideloaded extension warnings)

Reference: EXT-3-REQ-9 - Sideloaded extensions shall display prominent security warnings

Given: A conformant browser with EXT-3 capability (unrestricted extension installation)

Task: Verify that sideloaded extensions (installed outside official stores) display prominent, persistent security warnings to inform users of increased risk, ensuring users understand that sideloaded extensions have not undergone official security review, may contain malicious code, and could compromise their privacy or security, enabling informed decisions about whether to continue using extensions from untrusted sources.

Verification:

1. Install a test extension via sideloading (direct file installation, not from store) → Test that warnings cannot be permanently dismissed or hidden
2. Verify that installation displays prominent security warning: → Verify that each browser session shows security status
   - Warning about unreviewed extension source
   - Explanation of potential security risks
   - Recommendation to only install trusted extensions
   - Option to cancel installation
3. Test that warning requires affirmative user acknowledgment → Test that extension permissions page emphasizes sideloaded status
4. Verify that installed sideloaded extension shows persistent indicators: → Verify that warning messages are clear and explain risks in user-friendly language
   - Visual badge or label in extension list
   - Warning icon on extension toolbar button
   - Status message in extension details
5. Test that browser status area indicates sideloaded extensions are present → Test that multiple sideloaded extensions show cumulative risk indicators
6. Verify that new tab page or security dashboard shows sideloaded extension warnings → Confirm that uninstalling sideloaded extension removes all warnings

Pass Criteria: Sideloaded extensions show prominent warnings at install AND persistent indicators remain visible AND warnings explain security risks clearly AND warnings cannot be hidden

Fail Criteria: No warnings shown OR warnings are dismissible OR risk explanation is unclear OR indicators are too subtle

Evidence: Installation warning screenshots, persistent indicator demonstrations, security dashboard views, warning message content analysis

References:

- Security Warning Effectiveness: https://www.usenix.org/conference/soups2013/proceedings/presentation/akhawe
- Extension Security Best Practices: https://extensionworkshop.com/documentation/develop/build-a-secure-extension/

Assessment: EXT-REQ-30 (User extension permission controls)

Reference: EXT-3-REQ-10 - Browser shall provide user controls to review and revoke extension permissions

Given: A conformant browser with EXT-3 capability with installed extensions

Task: Verify that users have granular controls to review all granted extension permissions and revoke specific permissions without uninstalling extensions, enabling users to adjust extension access rights as their security preferences evolve, reducing over-privileged extensions, and providing a mechanism to contain potentially compromised extensions by limiting their access to sensitive APIs or data while investigating security concerns.

Verification:

1. Install test extensions with various permission sets → Verify that revoking permissions takes effect immediately
2. Access extension management interface → Test that extensions handle permission revocation gracefully (no crashes)
3. Verify that each extension displays complete list of granted permissions → Verify that revoked permissions persist across browser restarts
4. Test that permission list is organized and easy to understand: → Test that extensions can request re-grant of revoked permissions with user consent
   - Grouped by category (site access, API permissions, etc.)
   - Clear descriptions of what each permission enables
   - Visual indicators for high-risk permissions
5. Verify that users can revoke individual permissions through UI controls → Verify that permission changes are logged for user review
6. Test permission revocation for various permission types: → Test that revoking critical permissions triggers warning about potential extension breakage
   - Host permissions (site access)
   - API permissions (cookies, webRequest, etc.)
   - Optional permissions previously granted

Pass Criteria: All extension permissions are reviewable AND permissions can be individually revoked AND revocation is immediate and persistent AND extensions handle revocation gracefully

Fail Criteria: Permissions are not reviewable OR cannot be revoked individually OR revocation causes crashes OR changes do not persist

Evidence: Extension permission UI screenshots, permission revocation demonstrations, persistence tests, extension error handling logs, user control workflow recordings

References:

- Runtime Permissions: https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/API/permissions
- User Control Principles: https://www.w3.org/TR/design-principles/#user-control

Assessment: EXT-REQ-31 (Sideloaded extension malware scanning)

Reference: EXT-3-REQ-11 - Browser shall scan sideloaded extensions for known malware signatures

Given: A conformant browser with EXT-3 capability allowing sideloaded extensions

Task: Verify that the browser automatically scans sideloaded extensions for known malware signatures before installation, providing baseline protection against well-known malicious extensions even in unrestricted installation environments, detecting common malware families, cryptominers, data stealers, and other threats that have been previously identified and catalogued, reducing the risk of users inadvertently installing extensions with confirmed malicious behavior.

Verification:

1. Obtain or create test extension packages with known malware signatures: → Test that scan results are logged for security review
   - EICAR test file embedded in extension
   - Known malicious extension samples (from malware repositories)
   - Extensions with known cryptominer code
   - Extensions with data exfiltration patterns
2. Attempt to install each malicious test extension via sideloading → Verify that malware definitions are updated regularly
3. Verify that browser performs pre-installation malware scan → Test that scan performance does not significantly delay installation
4. Verify that malicious extensions are blocked before installation with clear warnings: → Verify that users are informed about scan process

- Specific threat description (malware type detected)
- Recommendation to not proceed
- Option to report malicious extension

5. Test that scan covers all extension components: → Test that detected malware can be quarantined or deleted
   - JavaScript files
   - Background scripts and content scripts
   - Embedded libraries
   - WebAssembly modules
6. Verify that clean extensions pass scan and install normally → Confirm that scan cannot be disabled or bypassed by users

Pass Criteria: Malware scanning occurs before installation AND known malware is detected and blocked AND scan covers all extension components AND malware definitions are updated regularly

Fail Criteria: No malware scanning occurs OR known malware is not detected OR scan can be bypassed OR definitions are outdated

Evidence: Malware detection screenshots, scan log files, test results with various malware samples, malware definition update verification, performance impact analysis

References:

- Malware Detection in Browser Extensions: https://research.google/pubs/pub43824/
- Extension Security Analysis: https://www.usenix.org/conference/usenixsecurity12/technical-sessions/presentation/carlini

Assessment: EXT-REQ-32 (Extension security event logging)

Reference: EXT-3-REQ-12 - All extension security events shall be logged for review

Given: A conformant browser with EXT-3 capability with extensions installed

Task: Verify that all extension security events are comprehensively logged to provide complete audit trail for security monitoring, incident response, and forensic analysis, enabling detection of malicious extension behavior, permission abuse, or security policy violations, ensuring that administrators and security teams have visibility into extension activities that could indicate compromise or misuse in environments with unrestricted extension installation.

Verification:

1. Install various test extensions (store, sideloaded, developer mode) → Test that logs are retained for appropriate security analysis period
2. Access browser security logging interface → Verify that logs can be exported or forwarded to SIEM systems
3. Trigger various extension security events: → Confirm that logging does not expose user passwords or sensitive personal data
   - Extension installation from different sources
   - Permission requests and grants/denials
   - Sensitive API usage (debugger, management, privacy)
   - Cross-extension communication attempts
   - Native messaging connections
   - WebRequest interception and modification
   - Suspicious network requests to unusual domains
   - Extension update installations
   - Extension crashes or security errors
   - User permission revocations
4. Verify that all events are logged with comprehensive details: → Test that high-severity events trigger immediate alerts or notifications

- Timestamp with timezone
- Extension ID, name, and version
- Event type and severity level
- User actions or automated triggers
- Success/failure status
- Affected resources (URLs, permissions, APIs)
- Source of extension (store, sideloaded, developer)

5. Test that logs are structured for automated analysis (JSON, CSV, syslog format) → Verify that logs include contextual information for investigation
6. Verify that logs can be filtered by severity, extension, or event type → Test that log tampering is prevented or detected

Pass Criteria: All extension security events are logged AND logs contain comprehensive detail AND logs are structured and filterable AND logs can be exported to SIEM AND logging preserves privacy

Fail Criteria: Security events are not logged OR logs lack necessary detail OR logs cannot be analyzed OR logs expose sensitive data OR logs can be tampered with

Evidence: Log samples showing various event types, log schema documentation, SIEM integration demonstrations, privacy analysis, retention policy verification, alert configuration examples

References:

- CWE-778: Insufficient Logging: https://cwe.mitre.org/data/definitions/778.html
- OWASP Logging Cheat Sheet: https://cheatsheetseries.owasp.org/cheatsheets/Logging_Cheat_Sheet.html
- Security Information and Event Management: https://csrc.nist.gov/glossary/term/security_information_and_event_ma

6.3 Cryptographic Security Assessments

This section covers assessment procedures for requirements ENC-REQ-1 through ENC-REQ-21, addressing TLS/SSL configuration, certificate validation, cryptographic API usage, secure random number generation, and encryption key management.

Assessment: ENC-REQ-1 (TLS 1.3+ support)

Reference: ENC-REQ-1 - Browser shall support TLS 1.3 or higher for all network communications

Given: A conformant browser with encryption capability (ENC-1 or higher)

Task: Verify that the browser supports modern TLS 1.3 protocol to protect network communications from downgrade attacks, reduce latency through improved handshake efficiency, and provide forward secrecy through ephemeral key exchange, ensuring that user traffic benefits from the latest cryptographic protections against eavesdropping and man-in-the-middle attacks.

Verification:

1. Configure a test web server to support multiple TLS versions: TLS 1.0, TLS 1.1, TLS 1.2, and TLS 1.3 → Verify that TLS 1.3 cipher suites are properly supported (TLS_AES_128_GCM_SHA256, TLS_AES_256_GCM_SHA384, TLS_CHACHA20_POLY1305_SHA256)
2. Navigate the browser to each server endpoint and capture the TLS handshake using packet capture tools (Wireshark, tcpdump) → Confirm that browser successfully negotiates TLS 1.3 when server supports it
3. Verify that the browser successfully negotiates TLS 1.3 when available → Verify that TLS 1.3 is preferred over TLS 1.2 when both are available
4. Configure the server to offer only TLS 1.3 and verify successful connection → Check that browser Security panel shows "TLS 1.3" as the protocol version
5. Test TLS 1.3 features: 0-RTT resumption, encrypted SNI (ESNI/ECH), and post-handshake authentication → Validate that packet captures confirm TLS 1.3 handshake structure (single round-trip)
6. Use browser DevTools Security tab to verify TLS version for the connection → Confirm that TLS 1.3-specific features (0-RTT, encrypted extensions) function correctly

7. Test that browser prefers TLS 1.3 over older versions when both are available → Verify that browser supports all mandatory TLS 1.3 cipher suites

Pass Criteria: Browser successfully negotiates TLS 1.3 connections AND prefers TLS 1.3 over older versions when available

Fail Criteria: Browser cannot establish TLS 1.3 connections OR prefers older TLS versions when TLS 1.3 is available

Evidence: Wireshark packet captures showing TLS 1.3 handshake, DevTools Security tab screenshots, server logs showing negotiated protocol version, cipher suite listings

References:

- RFC 8446 - TLS 1.3 Specification: https://datatracker.ietf.org/doc/html/rfc8446
- Mozilla SSL Configuration Generator: https://ssl-config.mozilla.org/
- NIST SP 800-52 Rev. 2 - TLS Guidelines: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-52r2.pdf
- Qualys SSL Labs - SSL/TLS Best Practices: https://github.com/ssllabs/research/wiki/SSL-and-TLS-Deployment-Best-Practices
- Chrome Network Security: https://www.chromium.org/Home/chromium-security/education/tls/

Assessment: ENC-REQ-2 (Certificate validation)

Reference: ENC-REQ-2 - Browser shall perform complete certificate chain validation including expiry, revocation, and trust anchor verification

Given: A conformant browser with encryption capability (ENC-1 or higher)

Task: Verify that the browser performs comprehensive certificate chain validation to prevent man-in-the-middle attacks using expired, self-signed, or fraudulent certificates, ensuring that only connections to servers with valid certificates from trusted certificate authorities are established without warnings, protecting users from impersonation attacks and ensuring transport layer authentication.

Verification:

1. Set up test scenarios with various certificate configurations: → Confirm that valid certificates from trusted CAs are accepted without warnings
   - Valid certificate from trusted CA
   - Expired certificate
   - Self-signed certificate
   - Certificate with wrong hostname (CN/SAN mismatch)
   - Certificate with incomplete chain
   - Certificate from untrusted CA
   - Revoked certificate
2. Navigate to each test endpoint and observe browser behavior → Verify that expired certificates trigger security warnings and connection blocks
3. Verify that browser displays appropriate security warnings for invalid certificates → Check that self-signed certificates are rejected with clear error messages
4. Test that users cannot bypass critical certificate errors without explicit acknowledgment → Validate that hostname mismatches are detected and reported
5. Verify certificate chain validation using browser DevTools Security tab → Confirm that incomplete certificate chains are rejected
6. Test extended validation (EV) certificate display → Verify that certificates from untrusted CAs trigger warnings
7. Capture and analyze the certificate validation process → Check that users should explicitly acknowledge risk to bypass warnings
8. Test certificate validation with intermediate CA certificates → Validate that EV certificates display organization name in browser UI

Pass Criteria: All invalid certificates are detected and reported with appropriate warnings AND users cannot silently bypass critical errors

Fail Criteria: Any invalid certificate is accepted without warning OR users can bypass critical errors without explicit acknowledgment

Evidence: Screenshots of certificate warning dialogs, DevTools Security tab showing certificate validation status, error console logs, packet captures showing certificate exchange

References:

- RFC 5280 - X.509 Certificate and CRL Profile: https://datatracker.ietf.org/doc/html/rfc5280
- CA/Browser Forum Baseline Requirements: https://cabforum.org/baseline-requirements-documents/
- OWASP Transport Layer Protection Cheat Sheet: https://cheatsheetseries.owasp.org/cheatsheets/Transport_Layer_Prot
- Mozilla Root Store Policy: https://www.mozilla.org/en-US/about/governance/policies/security-group/certs/policy/
- Chrome Certificate Transparency Policy: https://github.com/chromium/ct-policy

Assessment: ENC-REQ-3 (Certificate pinning support)

Reference: ENC-REQ-3 - Browser shall support HTTP Public Key Pinning (HPKP) or alternative certificate pinning mechanisms

Given: A conformant browser with encryption capability (ENC-2 or higher)

Task: Verify that the browser enforces certificate pinning to prevent man-in-the-middle attacks against critical domains by rejecting connections to pinned sites that present certificates from unexpected certificate authorities, even if those CAs are otherwise trusted, ensuring that even with a compromised certificate authority, attackers cannot impersonate pinned origins through fraudulent certificate issuance.

Verification:

1. Configure a test server to send Public-Key-Pins or Expect-CT headers → Test static pins for built-in domains (e.g., Google properties)
2. Navigate to the test server and verify that pins are stored by the browser → Confirm that browser correctly stores certificate pins from HTTP headers
3. Attempt to connect with a certificate that violates the pin (different public key) → Verify that connections with mismatched certificates are rejected
4. Verify that the browser rejects the connection due to pin validation failure → Check that pin validation errors display clear security warnings
5. Test pin expiration and max-age directive behavior → Validate that pins expire according to max-age directive
6. Test includeSubDomains directive for pin inheritance → Confirm that includeSubDomains directive applies pins to subdomains
7. Test report-uri directive for pin violation reporting → Verify that pin violations are reported to specified report-uri
8. Examine browser's HPKP/pin storage (chrome://net-internals/#hsts for Chromium) → Check that browser maintains persistent pin storage across sessions

Pass Criteria: Browser enforces certificate pins correctly AND rejects connections that violate pin requirements

Fail Criteria: Browser ignores certificate pins OR allows connections that violate pin requirements

Evidence: Screenshots of pin validation errors, browser internal state showing stored pins, network logs showing rejected connections, pin violation reports

References:

- RFC 7469 - HTTP Public Key Pinning (HPKP): https://datatracker.ietf.org/doc/html/rfc7469
- Expect-CT Header: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Expect-CT

- OWASP Certificate Pinning: https://owasp.org/www-community/controls/Certificate_and_Public_Key_Pinning
- Chrome HPKP Guidance: https://www.chromium.org/hsts/

Assessment: ENC-REQ-4 (HSTS enforcement)

Reference: ENC-REQ-4 - Browser shall enforce HTTP Strict Transport Security (HSTS) to prevent protocol downgrade attacks

Given: A conformant browser with encryption capability (ENC-1 or higher)

Task: Verify that HSTS enforcement prevents protocol downgrade attacks where attackers strip TLS to intercept credentials, ensuring all connections to HSTS-enabled domains use HTTPS even if users or applications attempt insecure HTTP connections. This protection is critical for preventing man-in-the-middle attacks that exploit the initial HTTP connection window before users type "https://" or before redirects to HTTPS occur.

Verification:

1. Configure a test server to send Strict-Transport-Security header: `Strict-Transport-Security: max-age=31536000; includeSubDomains; preload` → Attempt to bypass HSTS with user override and verify it's prevented
2. Navigate to https://test.example.com and verify HSTS header is received → Browser receives and processes Strict-Transport-Security header
3. Attempt to navigate to http://test.example.com (non-HTTPS) and verify automatic upgrade to HTTPS → HTTP requests are automatically upgraded to HTTPS for HSTS domains
4. Test that browser stores HSTS policy persistently across sessions → HSTS policy persists across browser sessions
5. Verify includeSubDomains directive applies HSTS to all subdomains → includeSubDomains directive correctly applies to subdomains
6. Test HSTS policy expiration by setting short max-age values → HSTS policies expire according to max-age directive
7. Examine browser's HSTS storage (chrome://net-internals/#hsts) → Preloaded domains enforce HTTPS even on first visit
8. Test HSTS preload list for well-known domains → Users cannot bypass HSTS enforcement

Pass Criteria: All HTTP requests to HSTS domains are automatically upgraded to HTTPS AND HSTS policies are enforced persistently

Fail Criteria: HTTP requests are not upgraded OR HSTS policies can be bypassed OR policies do not persist

Evidence: Network logs showing HTTP to HTTPS upgrades, browser HSTS storage screenshots, packet captures showing only HTTPS connections, console logs

References:

- RFC 6797 - HTTP Strict Transport Security (HSTS): https://datatracker.ietf.org/doc/html/rfc6797
- HSTS Preload List: https://hstspreload.org/
- OWASP HSTS Cheat Sheet: https://cheatsheetseries.owasp.org/cheatsheets/HTTP_Strict_Transport_Security_Cheat_
- MDN HSTS Documentation: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Strict-Transport-Security
- Chromium HSTS Implementation: https://www.chromium.org/hsts/

Assessment: ENC-REQ-5 (Mixed content blocking)

Reference: ENC-REQ-5 - Browser shall block mixed content (HTTP resources on HTTPS pages) to prevent downgrade attacks

Given: A conformant browser with encryption capability (ENC-1 or higher)

Task: Verify that mixed content blocking prevents protocol downgrade attacks where attackers inject HTTP resources into HTTPS pages to intercept data or execute malicious scripts. Blocking mixed content maintains the security guarantees of HTTPS by preventing unencrypted channels that could be exploited by network-level attackers to compromise the page's integrity and confidentiality.

Verification:

1. Create an HTTPS test page (https://test.example.com) that loads various HTTP resources: → Active mixed content (scripts, stylesheets, iframes) is completely blocked
   - HTTP scripts: `<script src="http://example.com/script.js">`
   - HTTP stylesheets: `<link href="http://example.com/style.css">`
   - HTTP iframes: `<iframe src="http://example.com/page.html">`
   - HTTP images: `<img src="http://example.com/image.jpg">`
   - HTTP XHR/fetch requests
   - HTTP form submission actions
2. Navigate to the test page and observe browser behavior → Browser console displays clear mixed content blocking messages
3. Verify that active mixed content (scripts, stylesheets, iframes) is blocked → Passive mixed content may load with warnings (or be blocked in strict mode)
4. Test browser's handling of passive mixed content (images, audio, video) → Mixed content indicator appears in browser address bar/security indicator
5. Check browser console for mixed content warnings → CSP block-all-mixed-content directive blocks all mixed content
6. Test Content-Security-Policy: block-all-mixed-content directive → CSP upgrade-insecure-requests upgrades HTTP resources to HTTPS
7. Verify that upgrade-insecure-requests CSP directive upgrades HTTP to HTTPS → Form submissions to HTTP URLs are blocked or warned
8. Test mixed content indicator in browser address bar → Users are warned about security implications of mixed content

Pass Criteria: All active mixed content is blocked AND clear warnings are displayed in console and UI

Fail Criteria: Active mixed content loads successfully OR no warnings are displayed

Evidence: Browser console screenshots showing blocked resources, network panel showing blocked requests, security indicator screenshots, DevTools showing CSP violations

References:

- W3C Mixed Content Specification: https://w3c.github.io/webappsec-mixed-content/
- MDN Mixed Content Documentation: https://developer.mozilla.org/en-US/docs/Web/Security/Mixed_content
- Chrome Mixed Content Policy: https://blog.chromium.org/2019/10/no-more-mixed-messages-about-https.html
- Upgrade Insecure Requests: https://www.w3.org/TR/upgrade-insecure-requests/

Assessment: ENC-REQ-6 (Certificate Transparency)

Reference: ENC-REQ-6 - Browser shall enforce Certificate Transparency (CT) requirements for all publicly trusted certificates

Given: A conformant browser with encryption capability (ENC-2 or higher)

Task: Verify that Certificate Transparency enforcement detects rogue certificates issued by compromised or malicious certificate authorities, preventing attackers from using fraudulently issued certificates for man-in-the-middle attacks. CT ensures all publicly trusted certificates are logged in append-only public ledgers, making unauthorized certificate issuance detectable and accountable.

Verification:

1. Set up test servers with certificates that have different CT compliance levels: → Browser accepts certificates with valid SCTs
   - Certificate with embedded SCTs (Signed Certificate Timestamps)
   - Certificate with SCTs delivered via TLS extension
   - Certificate with SCTs delivered via OCSP stapling
   - Certificate without any SCTs
2. Navigate to each test server and observe browser behavior → Certificates without SCTs are rejected with security errors
3. Verify that browser requires CT for publicly trusted certificates → DevTools Security tab displays CT compliance status
4. Check DevTools Security tab for CT information → Browser validates SCT signatures from known logs
5. Test that certificates without valid SCTs are rejected → CT is enforced for all publicly trusted certificates
6. Verify that browser validates SCT signatures → Multiple SCT delivery mechanisms are supported
7. Test CT enforcement for different certificate types (DV, OV, EV) → Browser maintains list of trusted CT log servers
8. Examine browser's CT policy compliance → CT violations trigger certificate errors

Pass Criteria: Browser enforces CT requirements AND rejects certificates without valid SCTs

Fail Criteria: Browser accepts certificates without SCTs OR does not validate SCT signatures

Evidence: DevTools Security tab screenshots showing CT status, certificate error dialogs for non-CT certificates, network logs showing SCT validation, browser CT policy configuration

References:

- RFC 6962 - Certificate Transparency: https://datatracker.ietf.org/doc/html/rfc6962
- Chrome CT Policy: https://github.com/chromium/ct-policy/blob/master/ct_policy.md
- Certificate Transparency Overview: https://certificate.transparency.dev/
- Google CT Log List: https://www.gstatic.com/ct/log_list/v3/all_logs_list.json
- CT Monitoring: https://developers.google.com/web/updates/2017/10/certificate-transparency

Assessment: ENC-REQ-7 (OCSP stapling)

Reference: ENC-REQ-7 - Browser shall support OCSP stapling for efficient certificate revocation checking

Given: A conformant browser with encryption capability (ENC-1 or higher)

Task: Verify that OCSP stapling support enables efficient certificate revocation checking without privacy leaks, preventing attackers from using revoked certificates while avoiding OCSP responder queries that reveal user browsing patterns. Stapling moves the revocation check to the server, improving performance and privacy while maintaining security against compromised certificates.

Verification:

1. Configure a test server to support OCSP stapling (include OCSP response in TLS handshake) → Verify handling of expired or invalid OCSP responses
2. Navigate to the test server and capture the TLS handshake using packet capture tools → Browser successfully receives and validates stapled OCSP responses
3. Verify that browser receives and validates the stapled OCSP response → Packet captures show OCSP response in TLS handshake (CertificateStatus message)
4. Test server without OCSP stapling and verify browser falls back to OCSP responder query → Browser validates OCSP response signatures
5. Configure OCSP response indicating certificate is revoked and verify browser rejection → Revoked certificates are rejected based on OCSP response
6. Test OCSP response validation including signature verification → Browser falls back to direct OCSP query when stapling unavailable
7. Measure performance difference between stapled and non-stapled OCSP → OCSP Must-Staple extension is enforced (if present)

8. Test OCSP Must-Staple certificate extension enforcement → Performance metrics show faster connection with stapling

Pass Criteria: Browser correctly validates stapled OCSP responses AND rejects revoked certificates

Fail Criteria: Browser ignores OCSP responses OR accepts revoked certificates

Evidence: Packet captures showing stapled OCSP in TLS handshake, certificate error screenshots for revoked certificates, performance measurements, DevTools Security tab showing OCSP status

References:

- RFC 6066 - TLS Extension: OCSP Status Request: https://datatracker.ietf.org/doc/html/rfc6066#section-8
- RFC 6960 - Online Certificate Status Protocol (OCSP): https://datatracker.ietf.org/doc/html/rfc6960
- OCSP Stapling Explained: https://www.digicert.com/kb/ssl-support/what-is-ocsp-stapling.htm
- OCSP Must-Staple: https://scotthelme.co.uk/ocsp-must-staple/

Assessment: ENC-REQ-8 (Cipher suite restrictions)

Reference: ENC-REQ-8 - Browser shall support only secure cipher suites and disable weak or deprecated algorithms

Given: A conformant browser with encryption capability (ENC-1 or higher)

Task: Verify that cipher suite restrictions prevent cryptographic weaknesses from being exploited by attackers to break TLS encryption, ensuring only modern algorithms resistant to known attacks are used. Rejecting weak ciphers like RC4, DES, and export-grade algorithms prevents downgrade attacks and ensures forward security against future cryptanalysis.

Verification:

1. Configure test servers with various cipher suites: → Test cipher suite restrictions for different TLS versions
   - Strong: TLS_AES_256_GCM_SHA384, TLS_CHACHA20_POLY1305_SHA256
   - Acceptable: TLS_AES_128_GCM_SHA256, ECDHE-RSA-AES128-GCM-SHA256
   - Weak: RC4, DES, 3DES, NULL ciphers
   - Export-grade ciphers
2. Test browser connection to each server configuration → Browser successfully negotiates strong cipher suites
3. Verify that browser refuses weak cipher suites → Weak cipher suites (RC4, DES, 3DES, NULL) are rejected
4. Capture cipher suite negotiation using packet analysis → Connection fails when only weak ciphers are available
5. Test browser's cipher suite preference order → Browser prefers AEAD ciphers over CBC-mode ciphers
6. Verify that AEAD (Authenticated Encryption with Associated Data) ciphers are preferred → Cipher suite negotiation follows secure ordering
7. Test that CBC-mode ciphers are avoided or deprioritized → Export-grade ciphers are completely disabled
8. Examine browser's supported cipher suite list → DevTools shows negotiated cipher suite

Pass Criteria: Browser only accepts strong cipher suites AND rejects all weak or deprecated ciphers

Fail Criteria: Browser accepts any weak cipher suite OR fails to negotiate strong ciphers when available

Evidence: Packet captures showing cipher suite negotiation, DevTools Security tab showing negotiated cipher, server logs, connection error screenshots for weak cipher servers

References:

- Mozilla SSL Configuration Generator: https://ssl-config.mozilla.org/

- NIST SP 800-52 Rev. 2 - TLS Guidelines: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-52r2.pdf
- RFC 8446 - TLS 1.3 Cipher Suites: https://datatracker.ietf.org/doc/html/rfc8446#appendix-B.4
- IANA TLS Cipher Suite Registry: https://www.iana.org/assignments/tls-parameters/tls-parameters.xhtml#tls-parameters-4
- Qualys SSL Labs - Cipher Suite Guidance: https://github.com/ssllabs/research/wiki/SSL-and-TLS-Deployment-Best-Practices#23-use-secure-cipher-suites

Assessment: ENC-REQ-9 (Perfect forward secrecy)

Reference: ENC-REQ-9 - Browser shall support and prefer cipher suites with perfect forward secrecy (PFS)

Given: A conformant browser with encryption capability (ENC-2 or higher)

Task: Verify that perfect forward secrecy prevents retrospective decryption attacks where adversaries record encrypted traffic and later compromise server private keys to decrypt historical sessions. PFS ensures that even if long-term keys are compromised, past TLS sessions remain secure because session keys were ephemeral and never stored.

Verification:

1. Configure test servers with different key exchange mechanisms: → Test connection failure when only non-PFS ciphers are available
   - ECDHE (Elliptic Curve Diffie-Hellman Ephemeral) - provides PFS
   - DHE (Diffie-Hellman Ephemeral) - provides PFS
   - RSA key exchange - no PFS
2. Test browser connection to each server configuration → Browser successfully negotiates ECDHE/DHE cipher suites
3. Verify that browser prefers ECDHE/DHE over static RSA key exchange → Browser prefers PFS cipher suites over non-PFS alternatives
4. Capture TLS handshake to confirm ephemeral key exchange → Packet captures show ephemeral key exchange in handshake
5. Test that browser supports multiple elliptic curves (X25519, P-256, P-384) → Browser supports modern elliptic curves (X25519 preferred)
6. Verify proper DH parameter size enforcement (minimum 2048 bits) → DHE parameters meet minimum security requirements (2048+ bits)
7. Test TLS 1.3 behavior (all cipher suites provide PFS) → TLS 1.3 connections always provide PFS
8. Examine cipher suite preference order for PFS priority → DevTools Security tab indicates PFS status

Pass Criteria: Browser prefers PFS cipher suites when available AND successfully negotiates ephemeral key exchange

Fail Criteria: Browser prefers non-PFS ciphers over PFS alternatives OR fails to negotiate PFS key exchange

Evidence: Packet captures showing ECDHE/DHE key exchange, DevTools Security tab showing PFS cipher suite, connection test results, cipher suite preference analysis

References:

- RFC 8446 - TLS 1.3 (all cipher suites provide PFS): https://datatracker.ietf.org/doc/html/rfc8446
- Perfect Forward Secrecy Explained: https://scotthelme.co.uk/perfect-forward-secrecy/
- OWASP TLS Cheat Sheet - Forward Secrecy: https://cheatsheetseries.owasp.org/cheatsheets/Transport_Layer_Protectio secrecy
- Mozilla Server Side TLS Guidelines: https://wiki.mozilla.org/Security/Server_Side_TLS

Assessment: ENC-REQ-10 (Revocation checking)

Reference: ENC-REQ-10 - Browser shall perform certificate revocation checking via OCSP and/or CRL mechanisms

Given: A conformant browser with encryption capability (ENC-1 or higher)

Task: Verify that certificate revocation checking prevents attackers from using compromised or fraudulently obtained certificates that have been revoked by the issuing CA. Checking revocation status ensures browsers reject certificates known to be untrustworthy, protecting users from man-in-the-middle attacks using stolen or misused certificates even after they've been marked as invalid.

Verification:

1. Set up test certificates with different revocation configurations: → Test browser's revocation checking cache behavior
   - Valid certificate with OCSP responder URL
   - Valid certificate with CRL distribution point
   - Revoked certificate (marked as revoked in OCSP/CRL)
   - Certificate without revocation information
2. Navigate to servers using each certificate type → Browser performs OCSP queries for certificates with OCSP URLs
3. Verify that browser checks certificate revocation status → Browser downloads and processes CRLs when available
4. Test OCSP responder queries (use network monitoring) → Revoked certificates are rejected with clear error messages
5. Test CRL download and parsing → Browser caches revocation check results appropriately
6. Configure OCSP responder to return "revoked" status and verify browser rejection → OCSP responses are validated (signature, freshness)
7. Test soft-fail vs hard-fail behavior when revocation check is unavailable → CRL validity period is checked
8. Measure performance impact of revocation checking → Browser handles revocation check failures gracefully

Pass Criteria: Browser performs revocation checks AND rejects revoked certificates with clear error messages

Fail Criteria: Browser skips revocation checking OR accepts revoked certificates

Evidence: Network logs showing OCSP/CRL requests, certificate error screenshots for revoked certificates, packet captures, DevTools showing revocation status, performance measurements

References:

- RFC 6960 - Online Certificate Status Protocol (OCSP): https://datatracker.ietf.org/doc/html/rfc6960
- RFC 5280 - Certificate and CRL Profile: https://datatracker.ietf.org/doc/html/rfc5280
- Chrome CRLSet Mechanism: https://www.imperialviolet.org/2012/02/05/crlsets.html
- OWASP Certificate Revocation: https://owasp.org/www-community/controls/Certificate_and_Public_Key_Pinning

Assessment: ENC-REQ-11 (Web Crypto API compliance)

Reference: ENC-REQ-11 - Browser shall implement W3C Web Cryptography API for JavaScript cryptographic operations

Given: A conformant browser with encryption capability (ENC-1 or higher)

Task: Verify that Web Crypto API compliance provides secure, browser-native cryptographic operations preventing developers from implementing insecure custom cryptography that could be vulnerable to timing attacks, weak random number generation, or algorithmic flaws. Restricting the API to secure contexts ensures cryptographic keys and operations are only available over HTTPS, preventing network attackers from intercepting keys.

Verification:

1. Create test web page accessing window.crypto.subtle API → window.crypto.subtle is available in secure contexts only

2. Test key generation for various algorithms: → All mandatory algorithms are supported
   - RSA-OAEP, RSA-PSS (2048, 3072, 4096 bits)
   - ECDSA, ECDH (P-256, P-384, P-521 curves)
   - AES-GCM, AES-CBC (128, 256 bits)
   - HMAC with various hash functions
3. Test encryption/decryption operations with generated keys → Key generation produces keys of requested type and size
4. Test signing and verification operations → Encrypt/decrypt operations work correctly with proper padding
5. Test key derivation functions (PBKDF2, HKDF) → Sign/verify operations validate signatures correctly
6. Test key import/export in various formats (JWK, PKCS8, SPKI, raw) → Key derivation functions produce deterministic results
7. Verify that subtle crypto operations are only available in secure contexts (HTTPS) → Non-extractable keys cannot be exported
8. Test that cryptographic keys can be marked as non-extractable → Key usage constraints are enforced
9. Test key usage constraints (encrypt, decrypt, sign, verify, etc.) → Operations fail appropriately in insecure contexts (HTTP)
10. Verify proper error handling for invalid parameters → Error messages are clear and informative

Pass Criteria: All Web Crypto API operations function correctly AND API is restricted to secure contexts

Fail Criteria: Required algorithms are missing OR API is available in insecure contexts OR operations produce incorrect results

Evidence: Console logs showing successful crypto operations, test result screenshots, code demonstrating API usage, error messages for insecure context access

References:

- W3C Web Cryptography API Specification: https://www.w3.org/TR/WebCryptoAPI/
- MDN Web Crypto API Documentation: https://developer.mozilla.org/en-US/docs/Web/API/Web_Crypto_API
- Web Crypto API Examples: https://github.com/diafygi/webcrypto-examples
- OWASP Cryptographic Storage Cheat Sheet: https://cheatsheetseries.owasp.org/cheatsheets/Cryptographic_Storage_Cl
- Chrome Web Crypto Implementation: https://www.chromium.org/blink/webcrypto/

Assessment: ENC-REQ-12 (Secure random number generation)

Reference: ENC-REQ-12 - Browser shall provide cryptographically secure random number generation via Crypto.getRandomValues()

Given: A conformant browser with encryption capability (ENC-1 or higher)

Task: Verify that cryptographically secure random number generation prevents predictability attacks where weak randomness allows attackers to guess session tokens, cryptographic keys, or authentication nonces. High-quality entropy ensures generated values are unpredictable and cannot be reproduced, which is essential for secure key generation, token creation, and cryptographic operations.

Verification:

1. Create test page calling window.crypto.getRandomValues() with various typed arrays: → window.crypto.getRandomValues() is available and functional
   - Uint8Array, Uint16Array, Uint32Array
   - Different array sizes (1 byte to 65536 bytes)
2. Generate large samples of random data (millions of bytes) → Generated values pass statistical randomness tests
3. Test statistical properties of generated random data: → Repeated calls produce different, unpredictable values
   - Frequency test (chi-square)
   - Runs test

- Entropy analysis

4. Verify that repeated calls produce different values → All typed array types are supported
5. Test that maximum size limit (65536 bytes) is enforced → Maximum size limit (65536 bytes) is enforced
6. Verify proper error handling for invalid parameters → QuotaExceededError thrown for oversized requests
7. Test that random values are available in both secure and insecure contexts → Random values have sufficient entropy (>= 7.9 bits per byte)
8. Compare randomness quality across multiple browser sessions → Performance is acceptable (can generate MBs per second)
9. Test performance of random number generation → API is available in all contexts (secure and insecure)

Pass Criteria: Generated random values pass statistical tests AND have sufficient entropy for cryptographic use

Fail Criteria: Random values show statistical bias OR have insufficient entropy OR values are predictable

Evidence: Statistical test results, entropy analysis graphs, performance benchmarks, console logs showing API usage, test code with results

References:

- W3C Web Cryptography API - getRandomValues: https://www.w3.org/TR/WebCryptoAPI/#Crypto-method-getRandomValues
- MDN Crypto.getRandomValues(): https://developer.mozilla.org/en-US/docs/Web/API/Crypto/getRandomValues
- NIST SP 800-22 - Statistical Tests for Random Number Generators: https://csrc.nist.gov/publications/detail/sp/800-22/rev-1a/final
- Random.org - Statistical Tests: https://www.random.org/analysis/
- OWASP Cryptographic Storage - Random Number Generation: https://cheatsheetseries.owasp.org/cheatsheets/Cryptogra random-number-generation

Assessment: ENC-REQ-13 (SubResource Integrity)

Reference: ENC-REQ-13 - Browser shall enforce Subresource Integrity (SRI) for externally loaded scripts and stylesheets

Given: A conformant browser with encryption capability (ENC-1 or higher)

Task: Verify that Subresource Integrity enforcement prevents supply chain attacks where compromised CDNs or third-party servers deliver malicious modified scripts or stylesheets. SRI ensures that external resources match their expected cryptographic hashes, detecting unauthorized modifications that could inject malware, steal credentials, or compromise user data even if the hosting server is compromised.

Verification:

1. Create test page loading external resources with integrity attributes: → Resources with matching integrity hashes load and execute successfully

```
<script src="https://cdn.example.com/lib.js"
        integrity="sha384-oqVuAfXRKap7fdgcCY5uykM6+R9GqQ8K/uxy9rx7HNQlGYl1kPzQho1wx4JwY8wC"
        crossorigin="anonymous"></script>
<link href="https://cdn.example.com/style.css"
      integrity="sha384-T3c6CoIi6uLrA9TneNEoa7RxnatzjcDSCmG1MXxSR1GAsXEV/Dwwykc2MPK8M2HN"
      crossorigin="anonymous">
```

2. Verify that resources with correct integrity hashes load successfully → Resources with mismatched hashes are blocked from loading
3. Modify resource content to mismatch integrity hash and verify blocking → Browser console displays clear SRI violation errors
4. Test multiple hash algorithms: sha256, sha384, sha512 → Multiple hash algorithms (sha256, sha384, sha512) are supported

5. Test multiple integrity values (fallback hashes) → Multiple integrity values allow fallback verification
6. Test CORS requirement for cross-origin SRI resources → Cross-origin resources require proper CORS headers for SRI
7. Verify browser console errors for SRI failures → SRI failures prevent resource execution/application
8. Test SRI with different resource types (scripts, stylesheets, preload links) → Network panel shows blocked resources with SRI violations
9. Test that SRI failures prevent script execution/style application → Page functionality degrades gracefully when SRI blocks resources

Pass Criteria: Resources with correct integrity hashes load successfully AND resources with mismatched hashes are blocked with clear errors

Fail Criteria: Resources with mismatched hashes load and execute OR no error messages displayed

Evidence: Browser console screenshots showing SRI errors, network panel showing blocked resources, test page demonstrating SRI enforcement, DevTools showing integrity attribute validation

References:

- W3C Subresource Integrity Specification: https://www.w3.org/TR/SRI/
- MDN Subresource Integrity: https://developer.mozilla.org/en-US/docs/Web/Security/Subresource_Integrity
- SRI Hash Generator: https://www.srihash.org/
- OWASP SRI Guidance: https://cheatsheetseries.owasp.org/cheatsheets/Third_Party_Javascript_Management_Cheat_integrity
- Chrome SRI Implementation: https://www.chromium.org/Home/chromium-security/education/tls/

Assessment: ENC-REQ-14 (Encrypted SNI)

Reference: ENC-REQ-14 - Browser shall support Encrypted Server Name Indication (ESNI/ECH) to prevent SNI-based censorship and surveillance

Given: A conformant browser with encryption capability (ENC-3 or higher)

Task: Verify that Encrypted SNI support prevents network surveillance and censorship where adversaries monitor TLS handshakes to identify which websites users visit, enabling targeted blocking or surveillance. Encrypting SNI protects user privacy by hiding the destination hostname from network observers, preventing traffic analysis attacks that reveal browsing patterns even when connections use HTTPS.

Verification:

1. Configure a test server supporting Encrypted Client Hello (ECH) or ESNI → Examine browser settings for ECH/ESNI enablement
2. Publish ECH configuration in DNS (TLS HTTPS record type 65) → Browser successfully negotiates ECH/ESNI when available
3. Navigate browser to the test server → SNI extension is not visible in cleartext in packet captures
4. Capture TLS handshake using packet analysis tools → ClientHello contains encrypted_client_hello extension
5. Verify that SNI extension is encrypted in ClientHello message → Browser retrieves ECH configuration via DNS
6. Test fallback behavior when ECH is unavailable → Fallback to cleartext SNI works when ECH unavailable
7. Test that cleartext SNI is not visible in packet captures → DNS queries for ECH config are encrypted (DoH/DoT)
8. Verify browser DNS-over-HTTPS (DoH) or DNS-over-TLS (DoT) usage for ECH config retrieval → Browser DevTools or internal pages show ECH status
9. Test ECH with split-horizon DNS configurations → Connection succeeds with ECH-enabled servers

Pass Criteria: Browser encrypts SNI when ECH is available AND cleartext SNI is not visible in packet captures

Fail Criteria: SNI is transmitted in cleartext when ECH is available OR browser doesn't support ECH

Evidence: Packet captures showing encrypted ClientHello, Wireshark analysis showing absence of cleartext SNI, DNS query logs showing ECH config retrieval, browser configuration screenshots

References:

- RFC 8744 - Encrypted Server Name Indication (ESNI): https://datatracker.ietf.org/doc/html/rfc8744
- Encrypted Client Hello (ECH) Draft: https://datatracker.ietf.org/doc/html/draft-ietf-tls-esni
- Cloudflare ECH Announcement: https://blog.cloudflare.com/encrypted-client-hello/
- Mozilla ECH Implementation: https://support.mozilla.org/en-US/kb/understand-encrypted-client-hello
- Chrome ECH Status: https://chromestatus.com/feature/6196703843581952

Assessment: ENC-REQ-15 (Certificate error UI)

Reference: ENC-REQ-15 - Browser shall display clear, prominent warnings for certificate errors with appropriate risk communication

Given: A conformant browser with encryption capability (ENC-1 or higher)

Task: Verify that certificate error UI effectively warns users about man-in-the-middle attacks or compromised servers before they transmit sensitive data to untrusted connections. Clear, prominent warnings with appropriate risk communication prevent users from inadvertently trusting malicious certificates, while making bypass actions deliberately difficult discourages risky behavior that could expose credentials or personal information.

Verification:

1. Create test scenarios for various certificate errors: → Certificate errors trigger full-page interstitial warnings
   - Expired certificate
   - Untrusted CA
   - Hostname mismatch
   - Self-signed certificate
   - Revoked certificate
   - Weak signature algorithm (SHA-1)
   - Invalid certificate chain
2. Navigate to each error scenario and document browser UI response → Warning messages clearly explain the security risk
3. Verify that warning messages clearly communicate security risk → Users should take explicit action to bypass (not easy clickthrough)
4. Test that warnings are difficult to bypass (require explicit user action) → Technical details are accessible via "Advanced" or similar link
5. Evaluate warning message clarity for non-technical users → Certificate details can be viewed and inspected
6. Test that technical details are available (certificate viewer) → Warning UI uses appropriate visual indicators (red, warning icons)
7. Verify that bypass actions are clearly labeled with risk warnings → Bypass options are clearly labeled with risk warnings
8. Test mobile and desktop warning UI differences → Error types are distinguishable in UI messages
9. Verify that errors are logged in browser console → Console logs provide technical error details

Pass Criteria: All certificate errors display prominent warnings AND bypass requires explicit user acknowledgment of risk

Fail Criteria: Certificate errors can be bypassed silently OR warnings are unclear or easily dismissed

Evidence: Screenshots of warning UI for each error type, user testing feedback on clarity, console error logs, comparison with browser security UI guidelines

References:

- Google Transparency Report - HTTPS Security: https://transparencyreport.google.com/https/overview

Assessment: ENC-REQ-16 (HTTPS-first mode)

Reference: ENC-REQ-16 - Browser shall implement HTTPS-first mode to automatically upgrade HTTP connections to HTTPS when available

Given: A conformant browser with encryption capability (ENC-2 or higher)

Task: Verify that HTTPS-first mode protects users from accidental insecure connections where attackers perform SSL-stripping attacks to downgrade HTTPS to HTTP, enabling interception of credentials and session tokens. Automatic HTTPS upgrade eliminates the window of vulnerability before secure connections are established, while user warnings for HTTP-only sites ensure informed consent before transmitting data over insecure channels.

Verification:

1. Enable HTTPS-first mode in browser settings → Verify browser remembers HTTP-only sites to avoid repeated upgrade attempts
2. Navigate to HTTP URLs of sites that support HTTPS (http://example.com) → HTTP URLs are automatically upgraded to HTTPS
3. Verify that browser automatically upgrades to HTTPS → Address bar shows HTTPS protocol after upgrade
4. Test fallback behavior when HTTPS is unavailable: → Fallback to HTTP works with user consent when HTTPS unavailable
   - Site doesn't support HTTPS
   - HTTPS connection fails/times out
   - Certificate error on HTTPS version
5. Verify user is prompted before loading HTTP-only sites → User is warned before loading HTTP-only sites
6. Test HTTPS upgrade for embedded resources (images, scripts, iframes) → Browser remembers HTTP-only sites (cache/allowlist)
7. Test interaction with HSTS and upgrade-insecure-requests → Embedded resources are also upgraded
8. Measure performance impact of HTTPS upgrade attempts → HTTPS-first works alongside HSTS
9. Test HTTPS-first with different types of navigation (typed URL, bookmarks, links) → Network panel shows upgrade attempts

Pass Criteria: HTTP connections are automatically upgraded to HTTPS when available AND users are warned before HTTP-only sites load

Fail Criteria: HTTP URLs are not upgraded OR no warnings for HTTP-only sites OR fallback doesn't work

Evidence: Network logs showing HTTP to HTTPS upgrades, address bar screenshots, warning dialog screenshots, performance measurements, browser settings showing HTTPS-first configuration

References:

- Chrome HTTPS-First Mode: https://blog.chromium.org/2021/07/increasing-https-adoption.html
- Firefox HTTPS-Only Mode: https://support.mozilla.org/en-US/kb/https-only-prefs
- HTTPS Upgrade Mechanisms: https://www.w3.org/TR/upgrade-insecure-requests/
- EFF HTTPS Everywhere: https://www.eff.org/https-everywhere
- OWASP Transport Layer Protection: https://cheatsheetseries.owasp.org/cheatsheets/Transport_Layer_Protection_Chea

Assessment: ENC-REQ-17 (Certificate pinning bypass detection)

Reference: ENC-REQ-17 - Browser shall detect and prevent attempts to bypass certificate pinning protections

Given: A conformant browser with encryption capability (ENC-2 or higher)

Task: Verify that certificate pinning bypass detection prevents attackers or malware from installing rogue root certificates to perform man-in-the-middle attacks against pinned domains. Detecting bypass attempts protects browser vendor properties and high-security sites from SSL interception, while logging and user warnings ensure transparency when certificate validation is weakened by enterprise policies or security tools.

Verification:

1. Configure test environment with certificate pinning enabled → Static pins for built-in domains cannot be bypassed
2. Attempt various bypass techniques: → Custom root CA installation triggers user warnings
   - Installing custom root CA certificates
   - Using SSL/TLS interception proxies (corporate MITM)
   - Modifying browser certificate store
   - Using browser extensions to disable pinning
   - Command-line flags to disable certificate validation
3. Test browser's built-in static pins (Google, Mozilla properties) → SSL interception is detected and indicated in UI
4. Verify that pin bypass attempts are detected and logged → Browser logs pin bypass attempts
5. Test enterprise policy controls for pinning exceptions → Certificate store modifications are visible to users
6. Verify user notifications for certificate store modifications → Enterprise policies can override pins with explicit configuration
7. Test that developer tools can't silently bypass pinning → Developer tools respect pinning (or show clear bypass warnings)
8. Examine browser internal state for pin enforcement → Security indicators reflect weakened security when pins bypassed
9. Test interaction between pin bypass and security indicators → Console logs show certificate validation details

Pass Criteria: Static certificate pins cannot be bypassed without explicit user/admin action AND pin bypass attempts are logged and indicated

Fail Criteria: Pins can be silently bypassed OR no indication when certificate validation is weakened

Evidence: Console logs showing pin enforcement, certificate store modification warnings, test results from bypass attempts, enterprise policy documentation, security indicator screenshots

References:

- Chrome Certificate Pinning: https://www.chromium.org/Home/chromium-security/education/tls/
- OWASP Certificate Pinning: https://owasp.org/www-community/controls/Certificate_and_Public_Key_Pinning
- SSL Interception Detection: https://blog.mozilla.org/security/2015/04/30/deprecating-non-secure-http/
- Mozilla Pin Override: https://wiki.mozilla.org/SecurityEngineering/Public_Key_Pinning

Assessment: ENC-REQ-18 (TLS downgrade protection)

Reference: ENC-REQ-18 - Browser shall implement protections against TLS version and cipher suite downgrade attacks

Given: A conformant browser with encryption capability (ENC-1 or higher)

Task: Verify that TLS downgrade protection prevents man-in-the-middle attackers from forcing browsers to use older TLS versions or weaker cipher suites with known vulnerabilities like POODLE or BEAST.

Downgrade protection ensures that even when attackers intercept and modify handshake messages, the browser detects the manipulation and aborts the connection rather than proceeding with weakened cryptographic parameters.

Verification:

1. Set up test environment capable of simulating man-in-the-middle attacks → Browser signals maximum supported TLS version correctly
2. Configure server supporting TLS 1.3 and TLS 1.2 → TLS_FALLBACK_SCSV is included in fallback connections
3. Attempt to force downgrade from TLS 1.3 to TLS 1.2 by manipulating ClientHello → Version rollback attacks are detected and connection aborted
4. Test TLS_FALLBACK_SCSV signaling value (RFC 7507) → Cipher suite downgrade attempts trigger handshake failure
5. Attempt downgrade attacks during connection: → Browser validates ServerHello.random for downgrade sentinels
   - Version rollback to older TLS versions
   - Cipher suite downgrade to weaker algorithms
   - Extension stripping attacks
6. Verify browser detects and rejects downgrade attempts → Extension stripping is detected through transcript hash validation
7. Test that Finished message MAC includes all handshake messages → Finished message properly authenticates handshake
8. Verify TLS 1.3 downgrade protection sentinel values in ServerHello.random → Console shows error messages for detected downgrade attempts
9. Test protection against truncation attacks → Connection fails securely rather than completing with weakened security

Pass Criteria: All TLS downgrade attempts are detected AND connections fail rather than proceed with weakened security

Fail Criteria: Any downgrade attack succeeds OR browser accepts weakened connection parameters

Evidence: Packet captures showing downgrade attempts and rejection, Wireshark showing TLS_FALLBACK_SCSV, console error logs, test scripts demonstrating attack attempts

References:

- RFC 7507 - TLS Fallback SCSV: https://datatracker.ietf.org/doc/html/rfc7507
- RFC 8446 - TLS 1.3 Downgrade Protection: https://datatracker.ietf.org/doc/html/rfc8446#section-4.1.3
- POODLE Attack and Downgrade Prevention: https://www.openssl.org/~bodo/ssl-poodle.pdf
- Chrome TLS Implementation: https://www.chromium.org/Home/chromium-security/education/tls/

Assessment: ENC-REQ-19 (Legacy crypto deprecation)

Reference: ENC-REQ-19 - Browser shall deprecate and remove support for legacy cryptographic algorithms and protocols

Given: A conformant browser with encryption capability (ENC-1 or higher)

Task: Verify that legacy crypto deprecation prevents attackers from exploiting known cryptographic weaknesses in outdated algorithms like SHA-1 collision attacks, RC4 biases, or short RSA key factorization. Progressive deprecation with clear timelines gives organizations migration paths while ensuring browsers eventually reject severely compromised cryptography that no longer provides meaningful security guarantees.

Verification:

1. Test browser behavior with legacy cryptographic elements: → TLS 1.0 and 1.1 connections are rejected or show warnings

- TLS 1.0 and TLS 1.1 protocols
- SHA-1 certificates
- 1024-bit RSA keys
- MD5-based signatures
- RC4 cipher suite
- CBC-mode cipher suites
- DSA certificates

2. Verify that legacy protocols/algorithms are rejected or trigger warnings → SHA-1 certificates trigger security errors
3. Test deprecation timeline (when were features removed) → 1024-bit RSA keys are rejected
4. Verify that browser update notes document deprecated features → MD5 and RC4 are completely disabled
5. Test enterprise policy overrides for legacy support (temporary exceptions) → Legacy crypto rejections show clear error messages
6. Check browser developer documentation for deprecation roadmap → Browser documentation lists deprecated features with timelines
7. Test fallback behavior when modern crypto unavailable → Enterprise policies can temporarily enable legacy support (if necessary)
8. Verify that critical errors can't be bypassed for severely deprecated crypto → No silent fallback to insecure legacy protocols

Pass Criteria: All severely deprecated cryptographic elements are rejected AND users are warned about moderately deprecated features

Fail Criteria: Legacy crypto is accepted without warnings OR deprecated features work without indication

Evidence: Connection error screenshots for legacy servers, browser release notes documenting deprecations, console error logs, test results across browser versions showing deprecation timeline

References:

- Chrome Deprecation Timeline: https://www.chromium.org/Home/chromium-security/education/tls/
- Mozilla Security Roadmap: https://wiki.mozilla.org/Security/Server_Side_TLS
- RFC 8996 - Deprecating TLS 1.0 and TLS 1.1: https://datatracker.ietf.org/doc/html/rfc8996
- CA/B Forum - SHA-1 Deprecation: https://cabforum.org/2014/10/16/ballot-118-sha-1-sunset/
- NIST Cryptographic Algorithm Deprecation: https://csrc.nist.gov/projects/hash-functions

Assessment: ENC-REQ-20 (Cryptographic key isolation)

Reference: ENC-REQ-20 - Browser shall isolate cryptographic keys and prevent cross-origin key access

Given: A conformant browser with encryption capability (ENC-2 or higher)

Task: Verify that cryptographic key isolation enforces same-origin policy for Web Crypto API keys, preventing malicious cross-origin scripts from accessing or exfiltrating cryptographic keys generated by other origins. Key isolation ensures that even if an attacker compromises one origin, they cannot steal cryptographic keys belonging to other origins to impersonate users or decrypt sensitive data protected by those keys.

Verification:

1. Create test scenarios with Web Crypto API key generation on different origins → Test key isolation for different user profiles/contexts
2. Generate cryptographic keys on https://origin-a.com using Web Crypto API → Keys generated on one origin cannot be accessed from another origin
3. Attempt to access keys from https://origin-b.com → IndexedDB key storage respects same-origin policy
4. Test IndexedDB key storage isolation across origins → Non-extractable keys cannot be exported or extracted
5. Verify that keys marked as non-extractable cannot be extracted → Key usage restrictions are enforced (keys can't be used for unauthorized operations)

6. Test key export restrictions based on key usage → Key handles are opaque and cannot be transferred cross-origin
7. Test that key handles cannot be passed between origins via postMessage → Browser internal key storage is isolated per origin
8. Verify key isolation in browser's internal key storage → Hardware-backed keys are protected by platform security
9. Test hardware-backed key storage (if available, e.g., WebAuthn) → Different user profiles have separate key storage

Pass Criteria: Cryptographic keys are strictly isolated by origin AND non-extractable keys cannot be exported

Fail Criteria: Keys can be accessed across origins OR key usage restrictions can be bypassed

Evidence: Console logs showing SecurityError for cross-origin key access, test code demonstrating isolation, browser internal state showing key storage separation, WebAuthn test results

References:

- W3C Web Cryptography API - Key Storage: https://www.w3.org/TR/WebCryptoAPI/#concepts-key-storage
- MDN CryptoKey: https://developer.mozilla.org/en-US/docs/Web/API/CryptoKey
- Chrome Web Crypto Key Isolation: https://chromium.googlesource.com/chromium/src/+/master/components/webcrypt
- OWASP Key Management Cheat Sheet: https://cheatsheetseries.owasp.org/cheatsheets/Key_Management_Cheat_Shee
- WebAuthn Specification - Credential Storage: https://www.w3.org/TR/webauthn-2/#credential-storage

Assessment: ENC-REQ-21 (Certificate store security)

Reference: ENC-REQ-21 - Browser shall maintain secure certificate store with integrity protection and auditing

Given: A conformant browser with encryption capability (ENC-1 or higher)

Task: Verify that certificate store security prevents attackers from installing rogue root certificates to enable man-in-the-middle attacks against all TLS connections. Protected certificate stores with integrity checking and audit trails detect unauthorized modifications, while requiring explicit user consent for root CA changes prevents malware from silently compromising the foundation of TLS trust.

Verification:

1. Examine browser's certificate store location and access controls → Certificate store files have restrictive permissions (not world-readable)
2. Test that certificate store is protected with appropriate file system permissions → Certificate store cannot be modified without appropriate privileges
3. Attempt to modify certificate store while browser is running → Browser detects and handles certificate store corruption
4. Verify that certificate store changes are logged/auditable → Root CA additions/removals require explicit user consent
5. Test root CA certificate management: → Certificate store UI shows all installed root certificates
   - View installed root certificates
   - Add custom root CA (with user consent)
   - Remove root CA (with user consent)
   - Verify certificate trust settings
6. Test that system certificate store is used appropriately → System certificate store integration works correctly
7. Verify that enterprise-managed certificates are clearly indicated → Enterprise-managed certificates are visibly marked
8. Test certificate store integrity verification mechanisms → Certificate store modifications are logged

9. Test that certificate store updates are secure and authenticated → Users are notified of certificate store changes
10. Verify user notifications for certificate store modifications → Certificate store updates are signed and verified

Pass Criteria: Certificate store is protected with appropriate access controls AND modifications require user consent and are logged

Fail Criteria: Certificate store can be modified without user knowledge OR no audit trail for modifications

Evidence: File system permission analysis, certificate store UI screenshots, audit log samples, test results from modification attempts, enterprise policy documentation

References:

- Mozilla Root Store Program: https://www.mozilla.org/en-US/about/governance/policies/security-group/certs/
- Chrome Root Store: https://chromium.googlesource.com/chromium/src/+/main/net/data/ssl/chrome_root_store/
- Microsoft Trusted Root Program: https://learn.microsoft.com/en-us/security/trusted-root/program-requirements
- Apple Root Certificate Program: https://www.apple.com/certificateauthority/ca_program.html
- OWASP Certificate Management: https://cheatsheetseries.owasp.org/cheatsheets/Transport_Layer_Protection_Cheat_and-public-key-pinning

Assessment: ENC-REQ-22 (Certificate pinning override prohibition)

Reference: ENC-0-REQ-22 - Browser shall not allow user override of certificate pinning failures

Given: A conformant browser with ENC-0 capability (full encryption with HSM support)

Task: Verify that the browser prohibits user override of certificate pinning failures to prevent users from being socially engineered into accepting fraudulent certificates, ensuring that pinning violations - which indicate active man-in-the-middle attacks or certificate misissuance - are treated as hard failures that cannot be bypassed, protecting high-security environments where certificate pinning is a critical defense against targeted attacks on specific domains.

Verification:

1. Configure test server with certificate pinning via HTTP header or browser built-in pins → Test that advanced settings do not provide override mechanism
2. Establish baseline connection to verify pinning is active → Verify that enterprise policies cannot disable pinning for pinned domains
3. Modify server certificate to trigger pinning violation: → Test that developer tools cannot bypass pinning enforcement
   - Use different certificate authority
   - Use expired pin
   - Use self-signed certificate
   - Use certificate from wrong domain
4. Attempt to connect to server with pinning violation → Verify that command-line flags cannot disable pinning
5. Verify that browser blocks connection with clear error message → Test that error message explains pinning failure clearly
6. Verify that no "proceed anyway" or "add exception" option is available → Verify that pinning failures are logged for security monitoring

Pass Criteria: Pinning failures cannot be overridden through any mechanism AND no bypass options exist in UI OR settings AND error messages explain pinning violations AND failures are logged

Fail Criteria: Users can bypass pinning failures OR override mechanisms exist OR pinning can be disabled for pinned domains OR unclear error messages

Evidence: Pinning violation error screenshots showing no bypass option, settings inspection showing no override controls, policy test results, developer tools verification, log entries showing pinning failures

References:

- HTTP Public Key Pinning: https://www.rfc-editor.org/rfc/rfc7469
- Certificate Pinning Best Practices: https://owasp.org/www-community/controls/Certificate_and_Public_Key_Pinning
- CWE-295: Improper Certificate Validation: https://cwe.mitre.org/data/definitions/295.html

Assessment: ENC-REQ-23 (Hardware-backed cryptographic operations)

Reference: ENC-0-REQ-23 - All cryptographic operations shall be performed in hardware-backed secure enclaves when available

Given: A conformant browser with ENC-0 capability on hardware with secure enclave support

Task: Verify that all cryptographic operations utilize hardware-backed secure enclaves (such as TPM, Secure Enclave, or TEE) when available, ensuring that private keys and sensitive cryptographic material never exist in unencrypted form in main memory, protecting against memory dumps, cold boot attacks, and malware that attempts to extract cryptographic keys from software-based key storage, providing defense-in-depth through hardware isolation.

Verification:

1. Identify available hardware security features on test platform: → Verify that cryptographic operations occur within secure enclave
   - TPM (Trusted Platform Module) on Windows/Linux
   - Secure Enclave on macOS/iOS
   - TEE (Trusted Execution Environment) on Android
   - Hardware security modules (HSM)
2. Install browser on platform with hardware security support → Test that key material is not accessible via memory dumps
3. Verify that browser detects and initializes hardware security features → Verify that browser preferences or about page indicates hardware security status
4. Perform cryptographic operations requiring key generation or usage: → Test that browser falls back gracefully if hardware security is unavailable
   - TLS client certificates
   - Web Crypto API key generation
   - Password manager encryption keys
   - Browser sync encryption keys
5. Verify that keys are generated and stored in hardware-backed storage → Verify that enterprise policies can require hardware-backed crypto
6. Test that private keys cannot be exported from hardware storage → Test that hardware security failures are logged and reported to users

Pass Criteria: Cryptographic operations use hardware-backed storage when available AND private keys are non-exportable AND key material is protected from memory access AND hardware security status is visible to users

Fail Criteria: Keys are stored in software despite hardware availability OR keys can be exported OR no hardware security detection OR silent fallback without user notification

Evidence: Platform security feature verification, key storage location inspection, memory dump analysis showing no key exposure, browser status page screenshots, enterprise policy configuration tests

References:

- Trusted Platform Module: https://trustedcomputinggroup.org/resource/tpm-library-specification/
- Web Crypto API Key Storage: https://www.w3.org/TR/WebCryptoAPI/#concepts-key-storage
- Hardware Security Modules: https://csrc.nist.gov/glossary/term/hardware_security_module

Assessment: ENC-REQ-24 (Encryption strictness configuration)

Reference: ENC-2-REQ-17 - Users shall be able to configure encryption strictness levels

Given: A conformant browser with ENC-2 capability (selective encryption)

Task: Verify that users can configure encryption strictness levels to balance security requirements with compatibility needs, enabling organizations and advanced users to enforce stricter encryption policies than defaults (such as TLS 1.3-only, strong cipher suites only, or mandatory HSTS) or relax policies for specific legitimate scenarios (such as internal legacy systems), providing flexibility while maintaining security visibility and audit trails.

Verification:

1. Access browser encryption settings or configuration interface → Verify that enterprise policies can override user strictness settings
2. Identify available encryption strictness options: → Test that per-site exceptions can be configured for compatibility
   - Minimum TLS version selection (TLS 1.2, TLS 1.3 only)
   - Cipher suite policy (strong only, compatible, allow weak)
   - Mixed content policy (block all, block active only, allow)
   - Certificate validation strictness (strict, standard, permissive)
   - HSTS enforcement level
3. Test configuring each strictness level and verify enforcement: → Verify that strictness level changes are logged
   - Set TLS 1.3-only mode and verify TLS 1.2 connections fail
   - Configure strong cipher suites only and verify weak cipher rejection
   - Enable strict mixed content blocking and verify all mixed content blocked
   - Set strict certificate validation and verify self-signed cert rejection
4. Verify that strictness changes take effect immediately or after restart → Test that current strictness level is visible in security indicators
5. Test that configuration changes are persistent across browser sessions → Verify that documentation explains security implications of each level

Pass Criteria: Multiple encryption strictness levels are available AND configuration changes are enforced AND settings persist AND per-site exceptions are possible AND changes are logged

Fail Criteria: No strictness configuration available OR changes not enforced OR settings don't persist OR no audit trail OR no security baseline

Evidence: Settings UI screenshots showing strictness options, connection test results at different levels, persistence verification, per-site exception demonstrations, audit logs

References:

- TLS Configuration Recommendations: https://wiki.mozilla.org/Security/Server_Side_TLS
- OWASP Transport Layer Protection: https://cheatsheetseries.owasp.org/cheatsheets/Transport_Layer_Protection_Chea

Assessment: ENC-REQ-25 (Connection security visual indicators)

Reference: ENC-2-REQ-18 - Browser shall provide visual indicators for connection security status

Given: A conformant browser with ENC-2 capability (selective encryption)

Task: Verify that the browser provides clear, prominent visual indicators for connection security status, enabling users to quickly identify secure HTTPS connections, insecure HTTP connections, and connection security issues, preventing users from entering sensitive information on insecure sites and helping users recognize potential man-in-the-middle attacks or certificate problems through consistent, understandable security indicators in the browser UI.

Verification:

1. Navigate to various sites with different security states: → Verify that security indicator changes are immediate upon navigation
   - Valid HTTPS with strong encryption (https://example.com with TLS 1.3)
   - Valid HTTPS with weak encryption (https://site.com with TLS 1.2, weak cipher)
   - HTTP site (http://insecure.com)
   - HTTPS with certificate error (expired, self-signed, wrong domain)
   - HTTPS with mixed content (active and passive)
   - HTTPS with Extended Validation certificate
2. Verify that each security state has distinct visual indicator: → Test that security indicators are accessible (screen reader compatible)
   - Padlock icon or security indicator in address bar
   - Color coding (green for secure, red for insecure, warning for issues)
   - Text labels ("Secure", "Not Secure", "Connection is not secure")
   - Icon states (locked, unlocked, warning symbol)
3. Test that clicking security indicator shows detailed connection information: → Verify that indicators follow consistent design across all platforms
   - TLS version and cipher suite
   - Certificate details and validity
   - Mixed content warnings
   - Permission grants
4. Verify that indicators are visible in all browser UI states (fullscreen, focus mode, etc.) → Test that users can easily distinguish secure from insecure connections
5. Test that indicators cannot be spoofed by web content → Verify that connection security details are easy to access

Pass Criteria: Distinct visual indicators for each security state AND indicators are prominent and clear AND detailed information is accessible AND indicators cannot be spoofed AND consistent across platforms

Fail Criteria: Security indicators are unclear OR states are not visually distinct OR detailed information not available OR indicators can be spoofed OR inconsistent across platforms

Evidence: Screenshots of security indicators for each connection state, detailed security info panel screenshots, accessibility testing results, spoofing attempt tests, cross-platform consistency verification

References:

- Browser Security Indicators: https://www.w3.org/TR/design-principles/#priority-of-constituencies
- User Security Awareness: https://www.usenix.org/conference/soups2016/technical-sessions/presentation/felt

Assessment: ENC-REQ-26 (User encryption settings control)

Reference: ENC-3-REQ-11 - Users shall have full control over encryption settings

Given: A conformant browser with ENC-3 capability (basic encryption with optional enhanced protection)

Task: Verify that users have full control over all encryption settings, enabling advanced users and specific deployment scenarios to configure encryption behavior according to their unique requirements, such as enabling legacy protocols for compatibility with older systems, configuring custom cipher suite preferences, or managing certificate exceptions, while ensuring that users are informed of the security implications of relaxing encryption requirements.

Verification:

1. Access browser advanced encryption settings → Test that dangerous configurations trigger prominent warnings
2. Verify that comprehensive encryption controls are available: → Verify that encryption settings can be exported/imported for consistency
   - TLS version selection (TLS 1.0, 1.1, 1.2, 1.3)
   - Individual cipher suite enable/disable

- Certificate validation options
- Mixed content policy configuration
- HSTS bypass capabilities
- Certificate exception management
- Revocation checking options

3. Test modifying each encryption setting: → Test that enterprise policies can restrict user control over specific settings
   - Enable legacy TLS 1.0/1.1 and verify older servers accessible
   - Disable specific cipher suites and verify negotiation changes
   - Relax certificate validation and verify self-signed cert acceptance
   - Allow mixed content and verify passive content loads
   - Add certificate exceptions and verify accepted
4. Verify that each setting change shows security warning explaining risks → Verify that reset-to-defaults option exists for safe rollback
5. Test that changes require user acknowledgment of risks → Test that encryption setting changes are logged
6. Verify that current encryption configuration is clearly documented in UI → Verify that help documentation explains each setting clearly

Pass Criteria: All encryption settings are user-configurable AND security warnings accompany risky changes AND current configuration is visible AND reset option exists AND changes are logged

Fail Criteria: Encryption settings are not configurable OR no security warnings for risky changes OR configuration not visible OR no rollback option OR no audit trail

Evidence: Advanced settings UI screenshots, configuration change workflows, security warning examples, setting export/import tests, policy restriction demonstrations, audit logs

References:

- TLS Configuration: https://wiki.mozilla.org/Security/Server_Side_TLS
- User Control Principles: https://www.w3.org/TR/design-principles/#user-control

Assessment: ENC-REQ-27 (Encryption status UI indicators)

Reference: ENC-3-REQ-12 - Browser shall provide encryption status indicators in UI

Given: A conformant browser with ENC-3 capability (basic encryption with optional enhanced protection)

Task: Verify that the browser provides comprehensive encryption status indicators throughout the user interface, enabling users to understand the current encryption configuration and connection security state at any time, helping users make informed decisions about whether their current settings are appropriate for their security needs and alerting them to potential security issues that may result from configuration changes.

Verification:

1. Access browser with various encryption configurations active → Verify that weak encryption configurations trigger visual warnings
2. Verify that encryption status is visible in multiple UI locations: → Test that encryption status persists in browser session info
   - Address bar security indicator
   - Settings page encryption section
   - About/information pages
   - Developer tools security panel
3. Test that status indicators show: → Verify that status indicators are accessible via keyboard navigation
   - Current TLS version in use
   - Active cipher suite
   - Certificate validity status
   - Enabled/disabled encryption features (HSTS, CT, OCSP)

- Legacy protocol status (if enabled)
- Number of certificate exceptions

4. Navigate to sites with different security configurations → Test that indicators work in all browsing modes (normal, private, reader)
5. Verify that indicators update dynamically to reflect current connection → Verify that encryption dashboard or summary page exists
6. Test that clicking indicators provides detailed encryption information → Test that status information is exportable for security audits

Pass Criteria: Encryption status visible in multiple UI locations AND indicators show detailed configuration AND indicators update dynamically AND weak configurations trigger warnings AND status is exportable

Fail Criteria: Encryption status not visible OR indicators lack detail OR no dynamic updates OR no warnings for weak config OR status not exportable

Evidence: UI screenshots from various locations showing status, status detail panel captures, dynamic update demonstrations, weak config warning examples, audit export samples

References:

- Security Indicators Design: https://www.usenix.org/conference/soups2016/technical-sessions/presentation/felt
- Transparency in Browser Security: https://www.w3.org/TR/design-principles/#transparency

Assessment: ENC-REQ-28 (Legacy protocol compatibility mode)

Reference: ENC-3-REQ-13 - Legacy protocol support may be enabled for compatibility with user consent

Given: A conformant browser with ENC-3 capability (basic encryption with optional enhanced protection)

Task: Verify that legacy protocol support (SSL 3.0, TLS 1.0, TLS 1.1) can be enabled for compatibility with older systems when necessary, but requires explicit, informed user consent with clear security warnings, ensuring users understand the risks of downgrading to deprecated protocols while enabling access to legacy systems that cannot be immediately upgraded, such as embedded devices, industrial control systems, or archived services.

Verification:

1. Verify that legacy protocols are disabled by default → Verify that legacy protocol enablement is persistent across sessions
2. Attempt to connect to server requiring legacy protocol (TLS 1.0 or TLS 1.1) → Test that legacy protocol usage triggers visual indicator in address bar
3. Verify that connection fails with clear error message explaining protocol version mismatch → Verify that connections using legacy protocols show prominent warning
4. Access browser settings to enable legacy protocol support → Test that legacy protocol can be enabled for specific sites only (not globally)
5. Verify that enabling legacy protocols requires: → Verify that legacy protocol enablement is logged for security auditing
   - Navigation to advanced settings section
   - Explicit checkbox or toggle action
   - Acknowledgment of security warning dialog
6. Test that security warning clearly explains: → Test that reset to defaults disables legacy protocols
   - Legacy protocols have known vulnerabilities
   - Connections may be susceptible to downgrade attacks
   - Sensitive information should not be transmitted
   - Recommendation to upgrade server instead

Pass Criteria: Legacy protocols disabled by default AND enablement requires explicit consent AND clear security warnings shown AND usage is indicated visually AND site-specific enablement possible AND changes logged

Fail Criteria: Legacy protocols enabled by default OR no consent required OR warnings unclear OR no visual indicators OR only global enablement OR no audit trail

Evidence: Default protocol test results, enablement workflow screenshots, security warning content, visual indicator examples, site-specific configuration tests, audit log samples

References:

- Deprecating Legacy TLS: https://datatracker.ietf.org/doc/rfc8996/
- TLS 1.0 and 1.1 Deprecation: https://www.rfc-editor.org/info/rfc8996

Assessment: ENC-REQ-29 (Complete encryption policy control)

Reference: ENC-4-REQ-6 - Users shall have complete control over all encryption policies

Given: A conformant browser with ENC-4 capability (minimal encryption)

Task: Verify that users have unrestricted control over all encryption policies without artificial limitations, enabling specialized use cases such as security research, penetration testing, legacy system maintenance, or air-gapped environments where standard encryption enforcement may need to be completely disabled, while ensuring that the browser clearly indicates when insecure configurations are active and the risks associated with completely disabling encryption protections.

Verification:

1. Access all available encryption policy controls in browser → Test that insecure configuration status is highly visible in all UI
2. Verify that users can modify every encryption-related setting: → Verify that browser warns before entering sensitive input on insecure connections
   - Completely disable TLS/SSL enforcement
   - Allow HTTP for all connections
   - Disable all certificate validation
   - Accept any cipher suite including export-grade
   - Disable HSTS, OCSP, CT, and all security features
   - Allow unlimited certificate exceptions
   - Disable mixed content blocking entirely
   - Remove all protocol version restrictions
3. Test disabling each encryption protection individually → Test that configuration can be saved and restored
4. Verify that disabled protections do not activate even in dangerous scenarios → Verify that no enterprise policies can override user's complete control
5. Test that completely insecure configuration allows access to test servers: → Test that security researchers can use browser for SSL/TLS testing
   - HTTP-only servers
   - Self-signed certificate servers
   - Expired certificate servers
   - Weak cipher servers (3DES, RC4)
6. Verify that browser displays persistent warning when encryption is disabled → Verify that documentation explains appropriate use cases for insecure configs

Pass Criteria: All encryption policies are user-modifiable without restriction AND disabled protections do not activate AND insecure status is highly visible AND configuration is persistent AND appropriate for specialized use cases

Fail Criteria: Some encryption policies cannot be disabled OR disabled protections still activate OR insecure status not visible OR configuration doesn't persist OR inappropriate for intended use cases

Evidence: Complete settings access screenshots, disabled protection test results, insecure configuration visual indicators, specialized use case demonstrations, security researcher testimonials

References:

- Security Research Ethics: https://www.w3.org/Security/wiki/Improving_Web_Security
- Penetration Testing Requirements: https://owasp.org/www-community/Penetration_testing

Assessment: ENC-REQ-30 (User-enabled legacy protocol support)

Reference: ENC-4-REQ-7 - Browser shall support legacy protocols when explicitly enabled by user

Given: A conformant browser with ENC-4 capability (minimal encryption)

Task: Verify that browser supports legacy cryptographic protocols (SSL 3.0, TLS 1.0, TLS 1.1) when explicitly enabled by user, ensuring that legacy systems, embedded devices, or historical services that cannot be upgraded remain accessible through modern browsers in specialized scenarios, while making it clear that these protocols have known security vulnerabilities and should only be used when absolutely necessary with full understanding of the risks.

Verification:

1. Access browser protocol version settings → Test that protocol negotiation prefers newest enabled version
2. Verify that legacy protocol options are available: → Verify that legacy protocol connections show security warnings
   - SSL 3.0 enable/disable
   - TLS 1.0 enable/disable
   - TLS 1.1 enable/disable
3. Enable each legacy protocol individually → Test that mixed protocol versions work correctly (e.g., TLS 1.0 and 1.3 both enabled)
4. Test connection to servers requiring each legacy protocol: → Verify that legacy protocol usage is logged
   - SSL 3.0-only server (if available for testing)
   - TLS 1.0-only server
   - TLS 1.1-only server
5. Verify that connections succeed when protocol is enabled → Test that browser identifies which protocol version was used for each connection
6. Verify that connections fail when protocol is disabled → Verify that legacy protocol support can be toggled without browser restart

Pass Criteria: All legacy protocols can be enabled by user AND enabled protocols function correctly AND protocol negotiation works properly AND legacy usage is warned and logged AND risks are documented

Fail Criteria: Legacy protocols cannot be enabled OR enabled protocols don't work OR protocol negotiation fails OR no warnings/logging OR risks not explained

Evidence: Legacy protocol enablement screenshots, successful connection tests to legacy servers, protocol version negotiation logs, security warning examples, risk documentation excerpts

References:

- SSL 3.0 Vulnerabilities (POODLE): https://www.rfc-editor.org/rfc/rfc7568
- TLS 1.0 and 1.1 Deprecation: https://datatracker.ietf.org/doc/rfc8996/
- Legacy Protocol Security Risks: https://www.nist.gov/publications/transitioning-use-cryptographic-algorithms-and-key-lengths

6.4 Security Event Logging Assessments

This section covers assessment procedures for requirements LOG-REQ-1 through LOG-REQ-20, addressing security event logging, audit trails, privacy-preserving telemetry, log retention, and security monitoring capabilities.

Assessment: LOG-REQ-1 (Security event logging)

Reference: LOG-REQ-1 - Browser shall implement comprehensive security event logging for security-relevant events

Given: A conformant browser with LOG-1 or higher capability

Task: Verify that comprehensive security event logging enables detection and investigation of attacks, providing audit trails for security incidents like XSS attempts, CSP violations, and CORS bypasses. Complete logging with timestamps, origins, and outcomes allows security teams to identify attack patterns, correlate events across multiple targets, and respond to threats effectively while supporting compliance with security monitoring requirements.

Verification:

1. Configure the browser to enable security event logging in developer or enterprise mode → Verify that security events include sufficient context for investigation (URL, origin, error type, timestamp)
2. Access the browser's internal logging interfaces (chrome://net-internals/#events, about:networking, or equivalent) → Test that security events persist across browser restarts if configured for persistent logging
3. Navigate to a test page that triggers multiple security events (mixed content, certificate errors, XSS attempts) → All security policy violations are logged with accurate timestamps
4. Attempt cross-origin requests that violate CORS policies → Logs include sufficient context to identify the origin and nature of security events
5. Load test pages with Content Security Policy violations → Security events include: CSP violations, CORS failures, mixed content blocks, certificate errors, sandbox violations
6. Trigger sandbox violations by attempting to access restricted APIs → Logs distinguish between blocked and allowed actions with clear outcomes
7. Review the logged events to verify all security-relevant actions are captured → Event log format is structured and machine-readable (JSON or similar)
8. Export the security event log to verify it contains timestamps, event types, origins, and outcomes → Logs can be exported for analysis or forwarding to external systems

Pass Criteria: All tested security events are captured in logs with complete context (timestamp, origin, event type, outcome) AND logs are exportable in a structured format

Fail Criteria: Any security event fails to be logged OR logs lack critical context (timestamp, origin, or outcome) OR logs are not exportable

Evidence: Screenshots of security event logs showing various event types, exported log files in JSON/structured format, video recordings of security events being triggered and logged, comparison matrices showing event coverage

References:

- W3C Reporting API: https://www.w3.org/TR/reporting-1/
- OWASP Logging Cheat Sheet: https://cheatsheetseries.owasp.org/cheatsheets/Logging_Cheat_Sheet.html
- NIST SP 800-92 Guide to Computer Security Log Management: https://csrc.nist.gov/publications/detail/sp/800-92/final

Assessment: LOG-REQ-2 (Certificate error logging)

Reference: LOG-REQ-2 - Browser shall log all certificate validation failures with detailed error information

Given: A conformant browser with LOG-1 or higher capability

Task: Verify that certificate error logging enables detection of man-in-the-middle attacks and certificate misconfigurations, providing detailed information for security investigations. Comprehensive certificate logs with chain details, error types, and revocation status help identify patterns of TLS interception, rogue certificates, or systematic validation failures that could indicate ongoing attacks against users or infrastructure.

Verification:

1. Set up a test web server with various certificate issues (expired, self-signed, wrong hostname, revoked) → Verify that certificate transparency failures are logged when CT enforcement is enabled
2. Navigate to https://expired.badssl.com/ and verify the certificate error is logged → Test that HSTS violations involving certificates are properly logged
3. Navigate to https://wrong.host.badssl.com/ and verify hostname mismatch is logged → All certificate validation failures are logged with specific error codes
4. Navigate to https://self-signed.badssl.com/ and verify self-signed certificate is logged → Logs include certificate subject, issuer, validity period, and error reason
5. Navigate to https://revoked.badssl.com/ and verify revocation status is logged → Certificate chain information is captured in logs
6. Review the certificate error logs to verify they include: certificate chain, error type, validation date, origin → Revocation check results (OCSP/CRL) are included
7. Test certificate pinning failures by creating a pinning policy and violating it → Certificate pinning violations are logged separately

Pass Criteria: All certificate validation failures are logged with complete certificate details AND error reasons are specific and actionable

Fail Criteria: Any certificate error is not logged OR logs lack certificate details OR error reasons are generic/unhelpful

Evidence: Certificate error log entries showing various failure types, screenshots of BadSSL.com test results with corresponding logs, certificate chain dumps from logs, network traces showing certificate validation process

References:

- RFC 5280 X.509 Certificate Validation: https://www.rfc-editor.org/rfc/rfc5280
- Certificate Transparency RFC 6962: https://www.rfc-editor.org/rfc/rfc6962
- Chrome Certificate Error Logging: https://www.chromium.org/Home/chromium-security/certificate-transparency/
- OWASP Transport Layer Protection Cheat Sheet: https://cheatsheetseries.owasp.org/cheatsheets/Transport_Layer_Prot

Assessment: LOG-REQ-3 (Extension security events)

Reference: LOG-REQ-3 - Browser shall log extension-related security events including installation, permission changes, and security violations

Given: A conformant browser with EXT-1 or higher capability and LOG-1 or higher capability

Task: Verify that extension security event logging enables detection of malicious extensions that abuse permissions, access sensitive APIs without authorization, or attempt to bypass security policies. Detailed extension logs with lifecycle events, permission changes, and security violations help identify compromised or rogue extensions that could exfiltrate data, inject malicious scripts, or escalate privileges beyond their declared capabilities.

Verification:

1. Install a test extension and verify the installation event is logged with extension ID, name, and permissions → Simulate an extension attempting to bypass CSP and verify the violation is logged
2. Modify extension permissions and verify permission changes are logged → Test extension update events and verify version changes are logged
3. Create a test extension that attempts to access APIs without proper permissions → Extension lifecycle events (install, update, uninstall) are logged with metadata
4. Trigger extension content script injection and verify it's logged → Permission requests and grants are logged with timestamp and user action

5. Test extension network requests to verify they are logged separately from normal browsing → Extension security violations are logged separately from web page violations
6. Uninstall the extension and verify the removal event is logged → Extension API access attempts are logged with success/failure status
7. Test developer mode extension loading and verify it's flagged in logs → Developer mode extensions are clearly marked in logs

Pass Criteria: All extension lifecycle and security events are logged with complete metadata AND extension actions are distinguishable from page actions

Fail Criteria: Any extension security event is not logged OR extension actions cannot be distinguished from page actions OR permission changes are not logged

Evidence: Extension event logs showing lifecycle events, permission change logs, screenshots of extension security violations with corresponding log entries, comparison of extension vs. page event logs

References:

- Chrome Extension Security Architecture: https://developer.chrome.com/docs/extensions/mv3/security/
- WebExtensions API Security: https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/API
- Extension Permission Model: https://www.chromium.org/Home/chromium-security/extension-content-script-fetches/
- OWASP Browser Extension Security: https://owasp.org/www-community/vulnerabilities/Unsafe_Mobile_Code

Assessment: LOG-REQ-4 (CSP violation reporting)

Reference: LOG-REQ-4 - Browser shall implement Content Security Policy violation reporting and logging

Given: A conformant browser with LOG-1 or higher capability

Task: Verify that CSP violation reporting enables detection of XSS attempts and policy bypasses, providing actionable intelligence about attacks targeting web applications. Automated reporting with complete violation context allows security teams to identify attack vectors, adjust CSP policies, and detect systematic attempts to inject malicious scripts or load unauthorized resources that could compromise user data.

Verification:

1. Create a test page with a strict CSP policy: `Content-Security-Policy: default-src 'self'; report-uri /csp-report` → Test CSP reporting with report-only mode using Content-Security-Policy-Report-Only header
2. Attempt to load external scripts that violate the CSP policy → Verify that violation reports include: violated-directive, blocked-uri, source-file, line-number
3. Verify CSP violation reports are sent to the report-uri endpoint → All CSP violations trigger console warnings in developer tools
4. Check browser console for CSP violation messages → Violation reports are sent to configured report-uri endpoints
5. Test inline script violations and verify they are reported → Reports include complete context: violated directive, blocked resource, source location
6. Test eval() violations and verify they are blocked and reported → Report-only mode generates reports without blocking resources
7. Configure CSP with report-to directive and Report-To header for modern reporting → Modern Reporting API (report-to) is supported for CSP reporting

Pass Criteria: All CSP violations are reported to configured endpoints AND console warnings are displayed AND reports contain complete violation context

Fail Criteria: Any CSP violation is not reported OR reports lack critical information OR report-uri/report-to mechanisms don't function

Evidence: CSP violation reports in JSON format, server logs showing received reports, browser console screenshots with CSP warnings, network traces showing report transmission, comparison of report-only vs. enforce mode

References:

- Content Security Policy Level 3: https://www.w3.org/TR/CSP3/
- CSP Violation Reports: https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP#violation_reports
- W3C Reporting API: https://www.w3.org/TR/reporting-1/
- CSP Evaluator Tool: https://csp-evaluator.withgoogle.com/

Assessment: LOG-REQ-5 (Network Error Logging - NEL)

Reference: LOG-REQ-5 - Browser shall support Network Error Logging (NEL) for monitoring network failures

Given: A conformant browser with LOG-1 or higher capability

Task: Verify that Network Error Logging enables detection of network-level attacks and infrastructure issues, monitoring connection failures that could indicate DNS hijacking, TLS interception, or targeted denial of service. NEL provides visibility into network failures that occur before HTTP layer, helping identify systematic attacks, CDN failures, or connectivity issues affecting user security and availability.

Verification:

1. Set up a test server that sends NEL policy header: `NEL: {"report_to":"network-errors","max_age":86400,"succes`
   → Verify NEL reports include: type, url, server-ip, protocol, status-code, elapsed-time, phase
2. Configure Report-To header: `Report-To: {"group":"network-errors","max_age":86400,"endpoints":[{"url":"h`
   → NEL policy is correctly parsed from HTTP headers
3. Trigger DNS resolution failures by attempting to navigate to non-existent domains → Network failures trigger NEL report generation
4. Trigger connection timeout errors by connecting to a filtered port → Reports are sent to configured endpoints asynchronously
5. Trigger TLS handshake failures by using misconfigured certificates → Sampling fractions are respected for success/failure events
6. Trigger HTTP protocol errors by sending malformed responses → NEL reports include detailed failure context (phase, type, status)
7. Verify that NEL reports are generated and sent to the configured endpoint → Reports distinguish between DNS, connection, TLS, and HTTP errors
8. Check that success sampling works correctly (success_fraction parameter) → NEL reports are batched and sent efficiently

Pass Criteria: NEL policy is respected AND all configured failure types generate reports AND reports include complete network error context

Fail Criteria: NEL policy is ignored OR network failures don't generate reports OR reports lack critical failure information

Evidence: NEL policy headers from server, collected NEL reports in JSON format, server logs showing received reports, network traces demonstrating various failure types, NEL report timing analysis

References:

- Network Error Logging Specification: https://www.w3.org/TR/network-error-logging/
- Reporting API Specification: https://www.w3.org/TR/reporting-1/
- NEL Deployment Guide: https://developers.google.com/web/updates/2018/09/reportingapi
- MDN Network Error Logging: https://developer.mozilla.org/en-US/docs/Web/HTTP/Network_Error_Logging

Assessment: LOG-REQ-6 (Crash reporting)

Reference: LOG-REQ-6 - Browser shall implement privacy-preserving crash reporting with user consent

Given: A conformant browser with LOG-1 or higher capability

Task: Verify that privacy-preserving crash reporting enables browser vendors to identify and fix security vulnerabilities without exposing user browsing history or personal information. User consent requirements ensure transparency while crash report anonymization prevents data leakage, balancing the need for diagnostic information to address exploitable bugs with user privacy rights and regulatory compliance.

Verification:

1. Review browser settings for crash reporting consent options → Test opt-out functionality and verify no reports are sent when opted out
2. Enable crash reporting and verify user consent is obtained before activation → Verify renderer process crashes are reported separately from browser process crashes
3. Force a controlled browser crash using debugging tools or crash test pages → User consent is required before crash reporting is enabled
4. Verify a crash report is generated locally → Crash reports are generated for browser and renderer crashes
5. Check that crash reports are anonymized and don't contain browsing history or personal data → Reports include technical diagnostics (stack traces, crash signatures) but no personal data
6. Verify crash reports include: crash signature, stack trace (symbolicated), browser version, OS version → Users can review crash reports before submission
7. Test that crash reports are only uploaded after user consent → Crash reporting can be disabled and re-enabled in settings
8. Verify users can view and delete crash reports before submission → Crash reports are transmitted securely (HTTPS) to vendor endpoints

Pass Criteria: User consent is obtained before crash reporting AND crash reports exclude personal data AND users can review/delete reports

Fail Criteria: Crash reports are sent without consent OR reports contain personal/browsing data OR users cannot control crash reporting

Evidence: Crash report consent dialogs, sanitized crash reports showing included data, settings screenshots showing crash reporting controls, privacy policy documentation, crash report upload network traces

References:

- Breakpad Crash Reporting: https://chromium.googlesource.com/breakpad/breakpad/
- Firefox Crash Reporter: https://support.mozilla.org/en-US/kb/mozilla-crash-reporter
- GDPR Crash Reporting Compliance: https://gdpr.eu/what-is-gdpr/

Assessment: LOG-REQ-7 (Log data minimization)

Reference: LOG-REQ-7 - Browser shall minimize data collection in logs, collecting only information necessary for security purposes

Given: A conformant browser with LOG-1 or higher capability

Task: Verify that log data minimization prevents excessive collection of personal information, balancing security monitoring needs with privacy rights and regulatory compliance. Collecting only necessary security-relevant data with automatic retention limits reduces the risk of data breaches exposing user browsing history, credentials, or sensitive personal information while still enabling effective threat detection and incident response.

Verification:

1. Review all log categories to identify what data is collected (security events, network, performance, etc.) → Check that telemetry aggregates data rather than logging individual user actions
2. Examine security event logs to verify they don't contain unnecessary personal information → Logs contain only security-relevant events, not general browsing activity
3. Check that URLs in logs are sanitized (query parameters removed or hashed) → Personal identifiable information (PII) is redacted or hashed in logs
4. Verify that user credentials are never logged, even in error conditions → URL parameters that may contain session tokens are removed
5. Test that cookies and authentication tokens are redacted from network logs → Credentials, cookies, and authentication headers are never logged
6. Review crash reports to ensure they exclude browsing history and form data → IP addresses are anonymized or removed from logs
7. Verify that IP addresses in logs are anonymized (last octet removed) or hashed → Logs automatically expire based on retention policies
8. Test that logs have automatic retention limits (time-based and size-based) → Form input data is excluded from all logs
9. Verify that sensitive form fields (passwords, credit cards) are never logged → Aggregated metrics replace individual event logging where possible

Pass Criteria: All logs demonstrate data minimization (no unnecessary PII) AND sensitive data is consistently redacted AND retention limits are enforced

Fail Criteria: Logs contain unnecessary PII OR credentials/tokens appear in logs OR no retention limits exist

Evidence: Log samples showing redaction of sensitive data, privacy analysis of log contents, retention policy documentation, comparison of logged vs. available data showing minimization, code review of logging implementations

References:

- OWASP Logging Cheat Sheet - Data Sanitization: https://cheatsheetseries.owasp.org/cheatsheets/Logging_Cheat_Sheet
- NIST Privacy Framework: https://www.nist.gov/privacy-framework
- W3C Privacy Principles: https://www.w3.org/TR/privacy-principles/

Assessment: LOG-REQ-8 (Log anonymization)

Reference: LOG-REQ-8 - Browser shall implement anonymization techniques for logs that require user-related data

Given: A conformant browser with LOG-1 or higher capability

Task: Verify that log anonymization prevents re-identification of users from telemetry data, protecting user privacy while maintaining the ability to detect security incidents and diagnose technical issues. Proper anonymization defends against correlation attacks where adversaries combine multiple log entries to de-anonymize users, as well as against data breaches where stolen logs could reveal sensitive user information.

Verification:

1. Review telemetry logs to identify user-related fields (user ID, device ID, session ID) → Check that pseudonymous identifiers change across different log contexts
2. Verify that user identifiers are hashed with secure cryptographic hash functions → User identifiers are consistently hashed with strong cryptographic algorithms
3. Check that hash salts are rotated periodically to prevent correlation → Hash salts are documented and rotated on a defined schedule
4. Test that IP addresses are anonymized using techniques like IP truncation or hashing → IP addresses are truncated or hashed before storage
5. Verify that timestamps are rounded to reduce precision (hour or day level) where appropriate → Timestamps are appropriately rounded to reduce granularity

6. Test that geographic data is generalized (city level rather than GPS coordinates) → Geographic data is generalized to prevent precise location tracking
7. Review aggregation techniques to ensure k-anonymity (minimum group size) is maintained → Aggregated data maintains k-anonymity with k >= 5
8. Verify differential privacy techniques are applied to statistical queries on logs → Differential privacy noise is added to prevent individual identification
9. Test that user fingerprints cannot be reconstructed from anonymized logs → Cross-log correlation attacks are prevented through identifier rotation

Pass Criteria: All user-related data is anonymized using documented techniques AND re-identification is demonstrably prevented AND k-anonymity is maintained

Fail Criteria: User data is logged in plaintext OR anonymization is reversible OR re-identification is possible through correlation

Evidence: Anonymized log samples with hash values, salt rotation policy documentation, privacy analysis showing k-anonymity, differential privacy parameters, re-identification attack test results (negative results expected)

References:

- Differential Privacy: https://www.microsoft.com/en-us/research/publication/differential-privacy/
- K-Anonymity Model: https://epic.org/privacy/reidentification/Sweeney_Article.pdf
- NIST De-Identification Guidelines: https://nvlpubs.nist.gov/nistpubs/ir/2015/NIST.IR.8053.pdf
- Google's Privacy-Preserving Techniques: https://policies.google.com/technologies/anonymization

Assessment: LOG-REQ-9 (User consent for telemetry)

Reference: LOG-REQ-9 - Browser shall obtain explicit user consent before collecting and transmitting telemetry data

Given: A conformant browser with LOG-1 or higher capability

Task: Verify that explicit user consent for telemetry protects user privacy rights and complies with data protection regulations including GDPR and CPRA. Without proper consent mechanisms, browsers may violate privacy laws by collecting personal data without permission, and users are deprived of control over their information. Consent shall be freely given, specific, informed, and revocable to meet legal and ethical standards.

Verification:

1. Perform a fresh installation of the browser and observe the first-run experience → Check that consent is re-requested when telemetry data types or purposes change significantly
2. Verify that a clear consent prompt is displayed for telemetry collection → First-run consent prompt is clear, prominent, and explains data collection
3. Check that the consent prompt explains what data is collected and why → Users can freely choose to accept or decline without dark patterns
4. Verify users can decline telemetry without affecting core browser functionality → Declining telemetry doesn't degrade core browser functionality
5. Test that declining telemetry prevents all non-essential data collection → Telemetry settings are easily accessible in preferences/settings
6. Navigate to browser settings and verify telemetry preferences are accessible → Consent choices are persistent and respected across updates
7. Verify users can change their consent choice at any time in settings → Granular controls allow users to consent to specific telemetry types
8. Test that telemetry settings are granular (separate controls for crash reports, usage stats, etc.) → Changes to data collection practices trigger new consent requests
9. Verify that consent choices persist across browser sessions and updates → Consent records are maintained for compliance auditing

Pass Criteria: Explicit consent is obtained before telemetry collection AND users can easily manage consent preferences AND browser functions normally when telemetry is declined

Fail Criteria: Telemetry starts without consent OR consent cannot be withdrawn OR declining breaks browser functionality OR dark patterns are used

Evidence: Screenshots of consent prompts and settings UI, network traces showing no telemetry when declined, functional testing with telemetry disabled, consent flow video recordings, privacy policy documentation

References:

- GDPR Consent Requirements: https://gdpr.eu/gdpr-consent-requirements/
- ePrivacy Directive: https://eur-lex.europa.eu/legal-content/EN/ALL/?uri=CELEX:32002L0058
- W3C Privacy Principles - User Control: https://www.w3.org/TR/privacy-principles/#user-control

Assessment: LOG-REQ-10 (Secure log transmission)

Reference: LOG-REQ-10 - Browser shall transmit logs securely using encrypted channels with certificate validation

Given: A conformant browser with LOG-1 or higher capability

Task: Verify that secure log transmission prevents interception or modification of telemetry and crash reports in transit, protecting sensitive diagnostic data from network attackers. Without encrypted transmission and certificate validation, adversaries can eavesdrop on log data to gain insights into user behavior, browser vulnerabilities, or enterprise configurations, or perform man-in-the-middle attacks to inject false telemetry data.

Verification:

1. Enable telemetry and crash reporting in browser settings → Verify log transmission is batched and rate-limited to prevent network abuse
2. Trigger events that generate log transmissions (crash, CSP violation, NEL error) → All log transmissions use TLS 1.2 or higher encryption
3. Use network monitoring tools (Wireshark, mitmproxy) to capture log transmission traffic → Certificate validation is enforced for log collection servers
4. Verify all log transmissions use HTTPS (TLS 1.2 or higher) → Invalid or expired certificates prevent log transmission
5. Verify certificate validation is performed for log collection endpoints → Certificate pinning is applied to log endpoints where supported
6. Test that log transmission fails if the server certificate is invalid → No log data is ever transmitted in plaintext
7. Check that certificate pinning is used for log collection endpoints if available → Connection failures trigger retry with exponential backoff
8. Verify log data is not transmitted over insecure protocols (HTTP, FTP, unencrypted sockets) → Log batching reduces network overhead and improves privacy
9. Test that log transmission includes retry logic for temporary network failures → Rate limiting prevents log transmission from consuming excessive bandwidth

Pass Criteria: All log transmissions use TLS 1.2+ with certificate validation AND transmission fails for invalid certificates AND no plaintext transmission occurs

Fail Criteria: Any logs transmitted over plaintext protocols OR certificate validation is not enforced OR invalid certificates are accepted

Evidence: Network packet captures showing TLS-encrypted log traffic, certificate validation test results, failed transmission logs for invalid certificates, retry mechanism testing, bandwidth usage analysis

References:

- TLS 1.3 Specification RFC 8446: https://www.rfc-editor.org/rfc/rfc8446

- Certificate Pinning: https://owasp.org/www-community/controls/Certificate_and_Public_Key_Pinning
- OWASP Transport Layer Protection: https://cheatsheetseries.owasp.org/cheatsheets/Transport_Layer_Protection_Chea
- Mozilla TLS Configuration: https://wiki.mozilla.org/Security/Server_Side_TLS

Assessment: LOG-REQ-11 (Log integrity protection)

Reference: LOG-REQ-11 - Browser shall implement integrity protection for locally stored logs to prevent tampering

Given: A conformant browser with LOG-1 or higher capability

Task: Verify that log integrity protection prevents attackers from covering their tracks after compromising a system by tampering with security logs. Without integrity protection, malicious actors who gain local access can modify or delete log entries to hide evidence of their activities, making incident response and forensic investigation impossible. Cryptographic integrity mechanisms ensure that any tampering is detected.

Verification:

1. Enable local security logging in browser configuration or enterprise policy → Test that log integrity is checked before logs are exported or transmitted
2. Generate security events that create local log entries → Local logs include integrity protection mechanisms (signatures, MACs, or hashes)
3. Locate the local log storage files in the browser's data directory → Tampering with log contents is detected by the browser
4. Verify that log files include cryptographic signatures or message authentication codes (MACs) → Log files use OS-level protection where available (append-only, immutable flags)
5. Attempt to modify a log entry manually and verify the tampering is detected → Log rotation preserves integrity chains across files
6. Check that log files use append-only mechanisms where supported by the OS → Corrupted logs trigger alerts or warnings
7. Verify log rotation maintains integrity chains between rotated files → Enterprise deployments support strong integrity mechanisms (digital signatures)
8. Test that the browser detects and alerts on corrupted or tampered logs → Integrity checks occur before log export or transmission
9. Verify enterprise-mode logs support additional integrity mechanisms (digital signatures) → Integrity metadata is stored separately from log content for additional protection

Pass Criteria: Logs include integrity protection (signatures/MACs/hashes) AND tampering is detected AND alerts are generated for integrity violations

Fail Criteria: Logs lack integrity protection OR tampering is not detected OR no alerts for integrity violations

Evidence: Log file analysis showing integrity mechanisms, tampering test results demonstrating detection, alert screenshots for corrupted logs, documentation of integrity algorithms used, enterprise policy configurations

References:

- NIST FIPS 180-4 Secure Hash Standard: https://csrc.nist.gov/publications/detail/fips/180/4/final
- Log Integrity and Non-Repudiation: https://www.nist.gov/publications/guide-computer-security-log-management
- Merkle Tree for Log Integrity: https://en.wikipedia.org/wiki/Merkle_tree
- OWASP Logging Guide - Integrity: https://cheatsheetseries.owasp.org/cheatsheets/Logging_Cheat_Sheet.html

Assessment: LOG-REQ-12 (Log retention policies)

Reference: LOG-REQ-12 - Browser shall implement and enforce log retention policies that balance security needs with privacy requirements

Given: A conformant browser with LOG-1 or higher capability

Task: Verify that log retention policies balance security investigation needs against privacy rights by limiting how long personal data is stored. Excessive retention violates privacy regulations like GDPR which mandate data minimization, while insufficient retention hampers security incident investigation. Proper retention policies ensure logs are available for legitimate security purposes without becoming an indefinite privacy liability.

Verification:

1. Review browser documentation for default log retention policies → Verify that regulatory compliance requirements (GDPR, etc.) are considered in retention
2. Examine local log storage to identify retention periods for different log types → Default retention periods are documented for each log type
3. Verify that security logs have appropriate retention (30-90 days typical) → Security logs are retained longer than general telemetry (30-90 days vs. 7-30 days)
4. Test that crash dumps are automatically deleted after retention period → Automatic deletion occurs when retention period expires
5. Verify that telemetry data has shorter retention than security logs → Log rotation prevents disk exhaustion (size-based limits)
6. Check that enterprise mode supports configurable retention policies → Enterprise policies allow customization of retention periods
7. Test that log rotation occurs based on size and time criteria → Users can manually clear logs through settings or clear browsing data
8. Verify that users can manually clear logs before retention period expires → Retention enforcement continues even when browser is not running
9. Test that retention policies are enforced even when browser is closed → GDPR/privacy compliance is demonstrated through retention limits

Pass Criteria: Documented retention policies exist for all log types AND automatic deletion enforces retention AND policies comply with privacy regulations

Fail Criteria: No retention policies OR logs grow unbounded OR retention periods violate privacy regulations (too long)

Evidence: Retention policy documentation, log file age analysis, storage usage over time, automatic deletion test results, enterprise policy configuration examples, GDPR compliance analysis

References:

- ISO 27001 Log Management: https://www.iso.org/standard/27001
- NIST SP 800-92 Log Retention: https://csrc.nist.gov/publications/detail/sp/800-92/final

Assessment: LOG-REQ-13 (Security dashboard)

Reference: LOG-REQ-13 - Browser shall provide a security dashboard that presents security events and status to users

Given: A conformant browser with LOG-1 or higher capability

Task: Verify that a security dashboard empowers users to understand their security posture and respond to threats by providing clear visibility into security events and protection status. Without a dashboard, users remain unaware of ongoing attacks, misconfigurations, or compromised security settings, leaving them vulnerable. Transparent security status information enables informed security decisions and builds user trust.

Verification:

1. Access the browser's security dashboard through the browser's settings interface → Test that the dashboard supports filtering and searching of security events
2. Verify the dashboard displays current security status (safe/warning/critical) → Security dashboard is easily accessible from main settings

3. Check that recent security events are listed with timestamps and descriptions → Current security status is clearly displayed with visual indicators
4. Trigger a security event (certificate error, malware warning, etc.) and verify it appears in the dashboard → Recent security events are listed chronologically with timestamps
5. Test that the dashboard categorizes events by severity (critical, warning, info) → Events are categorized by severity level with appropriate visual coding
6. Verify the dashboard shows security settings status (HTTPS-only, Safe Browsing, etc.) → Each event includes actionable information and remediation guidance
7. Test that clicking on security events provides detailed information and remediation steps → Dashboard updates when new security events occur
8. Verify the dashboard updates in real-time or near-real-time when security events occur → Users can filter events by type, severity, or time period
9. Check that the dashboard is accessible from the main browser settings menu → Dashboard shows overall security posture (enabled protections)

Pass Criteria: Security dashboard is accessible AND displays recent security events with severity AND provides actionable remediation guidance

Fail Criteria: No security dashboard exists OR dashboard doesn't show events OR events lack context/remediation info

Evidence: Screenshots of security dashboard showing various states, video walkthrough of dashboard features, security event listings, user interface usability assessment, comparison with security best practices

References:

- Chrome Security Settings: https://support.google.com/chrome/answer/114836
- NIST Cybersecurity Framework - Detect: https://www.nist.gov/cyberframework
- User-Centered Security Design: https://www.usenix.org/conference/soups2019

Assessment: LOG-REQ-14 (Incident detection)

Reference: LOG-REQ-14 - Browser shall implement automated incident detection based on security event patterns

Given: A conformant browser with LOG-2 or higher capability

Task: Verify that automated incident detection identifies active attacks by correlating security event patterns that indicate malicious activity, enabling rapid response before significant damage occurs. Manual log review alone cannot detect sophisticated attacks that span multiple events or occur at scale. Automated detection using heuristics and pattern matching provides early warning of credential stuffing, reconnaissance, malware distribution, and other attack campaigns.

Verification:

1. Configure the browser for enhanced security monitoring (enterprise mode if required) → Test that correlation of multiple minor events escalates to incident status
2. Access browser's internal incident detection interfaces or logs → Automated detection identifies suspicious patterns (credential stuffing, scanning, etc.)
3. Simulate a credential stuffing attack by repeatedly entering wrong passwords → Incident detection uses heuristics and machine learning where appropriate
4. Verify that repeated authentication failures trigger an incident alert → Multiple low-severity events can aggregate to trigger incident alerts
5. Simulate a port scanning attack by navigating to many sequential ports on localhost → False positive rates are managed through tuning and whitelisting
6. Verify that unusual network activity patterns are detected → Incidents are logged with detailed context for investigation
7. Trigger multiple CSP violations in rapid succession and verify pattern detection → Users or administrators receive notifications for detected incidents

8. Test that suspicious extension behavior (excessive API calls) triggers alerts → Incident severity is calculated based on event type and frequency
9. Verify that malware download attempts are detected and blocked → Detection rules are updated regularly to address new attack patterns

Pass Criteria: Automated detection identifies at least 3 attack patterns (credential stuffing, scanning, malware) AND incidents are logged with context AND alerts are generated

Fail Criteria: No automated detection occurs OR fewer than 3 attack patterns detected OR no alerts generated

Evidence: Incident detection logs showing various attack patterns, alert notifications, false positive analysis, detection rule documentation, test results for simulated attacks, tuning methodology

References:

- MITRE ATT&CK Framework: https://attack.mitre.org/
- NIST Incident Response Guide SP 800-61: https://csrc.nist.gov/publications/detail/sp/800-61/rev-2/final
- Browser Security Indicators: https://www.w3.org/TR/security-privacy-questionnaire/

Assessment: LOG-REQ-15 (Audit trail completeness)

Reference: LOG-REQ-15 - Browser shall maintain complete audit trails for security-relevant administrative actions

Given: A conformant browser with LOG-2 or higher capability (enterprise mode)

Task: Verify that complete audit trails for administrative actions enable accountability and investigation of security policy changes, preventing unauthorized or malicious modifications from going unnoticed. Without comprehensive audit logging, insider threats or compromised administrator accounts can weaken security settings without detection. Complete audit trails create accountability and support forensic investigations when security incidents occur.

Verification:

1. Enable enterprise policy management for the browser → Export the audit log and verify it includes all tested actions with complete metadata
2. Change a security-critical setting (e.g., disable Safe Browsing, modify HTTPS-only mode) → All security-relevant configuration changes are logged
3. Verify the change is logged with: timestamp, user/admin identity, setting name, old value, new value → Logs include: timestamp, user/admin identity, action type, object affected, before/after values
4. Install or remove a browser extension and verify the action is logged → Both successful and failed administrative actions are logged
5. Modify certificate trust settings and verify the change is logged → Extension lifecycle events (install/update/remove) are included
6. Change cookie or site permission policies and verify logging → Certificate and trust anchor modifications are logged
7. Modify content security policies and verify logging → Policy enforcement events are captured
8. Test that policy enforcement (GPO, MDM) actions are logged → Audit logs are tamper-evident and include integrity protection
9. Verify that failed administrative actions (insufficient permissions) are also logged → Logs are exportable in standard formats (JSON, CSV, syslog)

Pass Criteria: All security-relevant administrative actions are logged with complete metadata AND failed actions are logged AND logs are exportable

Fail Criteria: Any security configuration change is not logged OR logs lack critical metadata OR logs are not exportable

Evidence: Audit log exports showing various administrative actions, log completeness analysis, integrity verification results, enterprise policy documentation, screenshots of logged events

References:

- NIST SP 800-53 Audit and Accountability: https://csrc.nist.gov/publications/detail/sp/800-53/rev-5/final
- ISO 27001 Audit Logging: https://www.iso.org/standard/27001
- CIS Controls - Audit Log Management: https://www.cisecurity.org/controls/

Assessment: LOG-REQ-16 (Real-time security alerts)

Reference: LOG-REQ-16 - Browser shall provide real-time security alerts for critical security events

Given: A conformant browser with LOG-1 or higher capability

Task: Verify that real-time security alerts prevent users from inadvertently exposing themselves to immediate threats by providing prominent warnings before dangerous actions occur. Delayed or passive alerts allow users to proceed with risky actions like visiting malware sites or ignoring certificate errors. Immediate, blocking alerts with clear threat information enable users to make informed security decisions and avoid common attack vectors.

Verification:

1. Navigate to a site with a revoked certificate and verify immediate alert is displayed → Test that alert severity levels affect presentation (critical vs. warning vs. info)
2. Navigate to a known malware site (using Safe Browsing test URLs) and verify blocking alert → Critical security events trigger immediate, modal alerts
3. Trigger a password breach detection (if supported) and verify immediate notification → Alerts are displayed before dangerous actions are allowed
4. Install a malicious extension (test extension) and verify warning is displayed → Alert content is clear, specific, and actionable
5. Attempt to download a known malicious file and verify real-time blocking alert → Users should explicitly acknowledge alerts to proceed
6. Test that alerts are displayed before allowing dangerous actions (not after) → Alerts distinguish between critical threats (malware) and warnings (certificate issues)
7. Verify alerts are prominent, modal, and cannot be easily dismissed accidentally → Visual design makes alerts prominent and attention-getting
8. Test that alerts provide clear information about the threat and recommended actions → Enterprise mode supports admin notifications for critical events
9. Verify enterprise mode supports additional real-time alerting (admin notifications) → Alert fatigue is avoided through appropriate severity calibration

Pass Criteria: Critical security events trigger immediate modal alerts AND alerts provide clear threat information AND users should acknowledge before proceeding

Fail Criteria: No real-time alerts for critical events OR alerts are easily dismissed OR alerts lack actionable information

Evidence: Screenshots of various security alerts, video recordings of alert timing, user studies on alert comprehensibility, enterprise admin notification examples, alert frequency analysis

References:

- NIST Usable Security: https://www.nist.gov/programs-projects/usable-cybersecurity
- Google Safe Browsing: https://safebrowsing.google.com/
- Security Warning Design: https://www.usenix.org/conference/soups2019
- Alert Fatigue Research: https://www.ndss-symposium.org/ndss-paper/auto-draft-188/

Assessment: LOG-REQ-17 (Forensic log export)

Reference: LOG-REQ-17 - Browser shall support forensic-quality log export for security investigations

Given: A conformant browser with LOG-2 or higher capability

Task: Verify that forensic log export enables detailed security investigations by providing complete, integrity-protected logs in standard formats that can be analyzed with industry-standard tools. Without proper export capabilities, security teams cannot perform comprehensive incident response or forensic analysis, limiting their ability to understand attack vectors, determine scope of compromise, or provide evidence for legal proceedings.

Verification:

1. Generate various security events across multiple sessions (certificate errors, CSP violations, etc.) → Verify that export process itself is logged for audit purposes
2. Access browser log export functionality (may require developer or enterprise mode) → Log export is available through settings or developer tools
3. Export security logs in multiple formats (JSON, CSV, syslog) → Multiple export formats are supported (JSON, CSV, syslog, CEF)
4. Verify exported logs include all events from the specified time period → Exported logs are complete and chronologically ordered
5. Check that exported logs maintain chronological ordering → Timestamps use standardized formats (ISO 8601, Unix epoch)
6. Verify exported logs include complete metadata (timestamps in ISO 8601 format, event IDs, etc.) → Event identifiers are included for correlation
7. Test that log export includes integrity information (signatures or hashes) → Integrity information accompanies exports (checksums or signatures)
8. Verify sensitive information is appropriately redacted in exported logs → Sensitive data is redacted appropriately
9. Test that exported logs are in formats compatible with SIEM tools (Splunk, ELK, etc.) → Exported formats are compatible with common SIEM platforms

Pass Criteria: Log export functionality exists AND multiple standard formats supported AND exported logs include complete metadata with integrity protection

Fail Criteria: No export functionality OR only proprietary formats OR exported logs lack metadata OR no integrity protection

Evidence: Exported log files in various formats, SIEM import test results, log completeness verification, integrity validation results, format specification documentation, screenshots of export interface

References:

- Common Event Format (CEF): https://www.microfocus.com/documentation/arcsight/arcsight-smartconnectors-8.3/cef-implementation-standard/
- Syslog Protocol RFC 5424: https://www.rfc-editor.org/rfc/rfc5424
- ELK Stack Log Analysis: https://www.elastic.co/what-is/elk-stack
- NIST SP 800-92 Log Management: https://csrc.nist.gov/publications/detail/sp/800-92/final

Assessment: LOG-REQ-18 (Privacy-preserving analytics)

Reference: LOG-REQ-18 - Browser shall use privacy-preserving techniques for analytics and aggregate reporting

Given: A conformant browser with LOG-1 or higher capability

Task: Verify that privacy-preserving analytics techniques enable browsers to gather valuable usage insights and improve security without compromising individual user privacy. Traditional analytics create re-identification risks by collecting detailed individual behavior. Differential privacy, local noise injection, and k-anonymity

allow aggregated insights while mathematically guaranteeing that individual users cannot be identified or their specific behaviors revealed.

Verification:

1. Review browser telemetry documentation for privacy-preserving techniques → Verify that privacy parameters (epsilon, delta) are documented and justified
2. Verify that differential privacy is used for usage statistics aggregation → Differential privacy is applied to aggregate statistics
3. Check that local differential privacy (LDP) adds noise before data leaves the device → Local differential privacy adds noise on-device before transmission
4. Test that RAPPOR (Randomized Aggregatable Privacy-Preserving Ordinal Response) or similar is used → RAPPOR or equivalent techniques are used for categorical data
5. Verify that aggregated metrics cannot be de-aggregated to identify individuals → Privacy budgets limit cumulative information disclosure
6. Test that feature usage statistics use k-anonymity (minimum group size) → K-anonymity ensures minimum group sizes (k >= 5)
7. Verify that privacy budgets limit information disclosure over time → Federated learning keeps training data local
8. Check that federated learning is used where applicable (e.g., next-word prediction) → Attribution Reporting API uses noise and aggregation
9. Test that aggregate reporting APIs (Attribution Reporting) use noise injection → Privacy parameters (epsilon, delta, k) are publicly documented

Pass Criteria: Differential privacy or equivalent techniques are used AND privacy parameters are documented AND individual users cannot be identified from aggregates

Fail Criteria: No privacy-preserving techniques used OR aggregate data allows individual identification OR privacy parameters undocumented

Evidence: Privacy technique documentation, epsilon/delta parameter specifications, de-identification attack test results (negative), differential privacy implementation code review, aggregate report samples, federated learning architecture diagrams

References:

- Differential Privacy: https://www.microsoft.com/en-us/research/publication/differential-privacy/
- RAPPOR: Randomized Aggregatable Privacy-Preserving Ordinal Response: https://research.google/pubs/pub42852/
- Attribution Reporting API: https://github.com/WICG/attribution-reporting-api
- Federated Learning: https://ai.googleblog.com/2017/04/federated-learning-collaborative.html
- Apple Differential Privacy: https://www.apple.com/privacy/docs/Differential_Privacy_Overview.pdf
- W3C Privacy Principles: https://www.w3.org/TR/privacy-principles/

Assessment: LOG-REQ-19 (Compliance logging)

Reference: LOG-REQ-19 - Browser shall provide logging capabilities to support regulatory compliance requirements (GDPR etc.)

Given: A conformant browser with LOG-2 or higher capability (enterprise mode)

Task: Verify that compliance logging enables organizations to demonstrate adherence to privacy regulations by maintaining comprehensive records of data processing activities, consent, and data subject rights fulfillment. Without proper compliance logging, organizations cannot prove they honor user rights, track data processing lawfulness, or respond to regulatory audits, leading to significant legal and financial penalties under GDPR and similar laws.

Verification:

1. Review browser documentation for compliance logging capabilities → Check that logs can demonstrate compliance during audits

2. Verify that data processing activities are logged (collection, storage, transmission) → Data processing activities are comprehensively logged
3. Test that user consent events are logged with timestamps and scope → Consent events capture: timestamp, user ID, data types, purposes, duration
4. Verify that data deletion requests are logged and honored → Data deletion events are logged with completion verification
5. Check that data subject access requests can be fulfilled from logs → Access request fulfillment is possible from log data
6. Test that cross-border data transfers are logged with destination regions → Cross-border transfers are logged with legal basis
7. Verify that third-party data sharing events are logged → Third-party data sharing is logged with recipient and purpose
8. Test that data breach detection events are logged with required metadata → Breach detection and notification events are logged
9. Verify that retention policies align with regulatory requirements → Retention aligns with GDPR (no longer than necessary) and other regulations

Pass Criteria: Compliance-relevant activities are logged (consent, deletion, access) AND logs support audit requirements AND retention aligns with regulations

Fail Criteria: Compliance activities not logged OR logs insufficient for audits OR retention violates regulations

Evidence: Compliance log exports, sample audit reports generated from logs, consent event logs, deletion request logs, data processing records, legal basis documentation, retention policy compliance analysis

References:

- GDPR Requirements: https://gdpr.eu/
- ISO 27001 Compliance Auditing: https://www.iso.org/standard/27001
- NIST Privacy Framework: https://www.nist.gov/privacy-framework
- ePrivacy Directive: https://eur-lex.europa.eu/legal-content/EN/ALL/?uri=CELEX:32002L0058

Assessment: LOG-REQ-20 (Log access controls)

Reference: LOG-REQ-20 - Browser shall implement access controls to protect logs from unauthorized access or modification

Given: A conformant browser with LOG-2 or higher capability (enterprise mode)

Task: Verify that log access controls protect sensitive security and diagnostic information from unauthorized disclosure or tampering, preserving both user privacy and forensic integrity. Unprotected logs can be read by malware or local attackers to gather intelligence about system configuration, security events, or user activities. Without write protection, attackers can tamper with logs to hide evidence of compromise.

Verification:

1. Review log storage locations and verify they use appropriate OS-level permissions → Check that encryption at rest is available for sensitive logs
2. Test that log files are readable only by the browser process and authorized users → Log files have restrictive OS permissions (user-only or admin-only read)
3. Verify that unprivileged processes cannot access browser log files → File ACLs prevent unauthorized access on multi-user systems
4. Test that log files use OS access control mechanisms (file permissions, ACLs, encryption) → Logs in user profiles are isolated from other user accounts
5. Verify that logs stored in user profile directories are protected from other users → Remote log transmission uses authenticated, encrypted channels
6. Test that remote log access (enterprise SIEM integration) requires authentication → Log export requires explicit user action or administrative privileges

7. Verify that log export functionality requires user confirmation or admin privileges → Log files use append-only or immutable attributes where supported
8. Test that log modification is prevented through append-only modes or immutable flags → Log access attempts are recorded in audit logs
9. Verify that log access attempts are themselves logged for audit → Encryption at rest protects logs on disk

Pass Criteria: Log files have restrictive permissions AND remote access requires authentication AND log modification is prevented AND access is audited

Fail Criteria: Logs are world-readable OR no access controls on remote access OR logs can be modified OR access not audited

Evidence: File permission analysis (ls -l, icacls), ACL configurations, multi-user access testing, remote access authentication tests, append-only flag verification, access audit log samples, encryption at rest verification

References:

- NIST SP 800-92 Log Protection: https://csrc.nist.gov/publications/detail/sp/800-92/final
- Linux File Permissions Best Practices: https://www.redhat.com/sysadmin/linux-file-permissions-explained
- Windows ACL Security: https://docs.microsoft.com/en-us/windows/security/identity-protection/access-control/access-control
- macOS File System Security: https://developer.apple.com/documentation/security
- OWASP Logging Guide - Protection: https://cheatsheetseries.owasp.org/cheatsheets/Logging_Cheat_Sheet.html
- ISO 27001 Log Access Controls: https://www.iso.org/standard/27001

Assessment: LOG-REQ-21 (No remote telemetry transmission)

Reference: LOG-0-REQ-1 - Browser shall not transmit any telemetry or diagnostic data to remote servers

Given: A conformant browser with LOG-0 capability (no logging or local-only)

Task: Verify that the browser completely disables all remote telemetry and diagnostic data transmission, ensuring that no usage statistics, crash reports, security events, or any other diagnostic information is sent to external servers, protecting user privacy in high-security or air-gapped environments where outbound data transmission could leak sensitive information about user activities, system configuration, or security posture.

Verification:

1. Install browser and complete initial setup without accepting any telemetry → Test that browser does not attempt telemetry even after updates
2. Use network monitoring tools to capture all browser traffic (Wireshark, tcpdump, proxy) → Verify that no telemetry data is queued for later transmission
3. Perform various browser activities that typically trigger telemetry: → Check that browser source/binary has no telemetry code or endpoints configured
   - Normal browsing to various sites
   - Triggering browser crashes or errors
   - Installing/removing extensions
   - Changing security settings
   - Certificate errors and warnings
   - Mixed content warnings
4. Verify that no telemetry endpoints are contacted: → Test behavior in various network conditions (online, offline, restricted)
   - No connections to analytics domains
   - No crash report uploads
   - No usage statistics transmission
   - No security event reporting

5. Inspect browser configuration files for telemetry settings → Verify that error messages do not suggest remote reporting
6. Verify that all telemetry features are disabled or removed → Check that browser help/support does not reference telemetry features

Pass Criteria: No telemetry or diagnostic data transmitted to remote servers AND no telemetry endpoints contacted AND no telemetry code active AND no queued data

Fail Criteria: Any telemetry transmission occurs OR telemetry endpoints contacted OR telemetry code present OR data queued for transmission

Evidence: Network capture logs showing no telemetry traffic, configuration file inspection, browser binary analysis, behavioral testing across various scenarios, offline operation verification

References:

- GDPR Data Minimization: https://gdpr-info.eu/art-5-gdpr/
- Zero-Trust Network Architecture: https://csrc.nist.gov/publications/detail/sp/800-207/final

Assessment: LOG-REQ-22 (Opt-in local logging)

Reference: LOG-0-REQ-8 - All logging shall be disabled by default with opt-in for local logging

Given: A conformant browser with LOG-0 capability (no logging or local-only)

Task: Verify that all logging is disabled by default to maximize user privacy, requiring users to explicitly opt-in before any local security event logs are created, ensuring that users who do not want any diagnostic information stored on their system are protected by default, while still allowing security-conscious users to enable local logging for audit and forensic purposes when desired.

Verification:

1. Perform fresh browser installation → Verify that logging is explicitly enabled by user
2. Verify that logging is disabled by default on first run → Test enabling logging through settings
3. Check that no log files are created during initial setup → Verify that opt-in requires clear user action (not pre-checked checkbox)
4. Perform various browser activities without enabling logging: → After enabling, verify that logs are created
   - Navigate to multiple websites
   - Trigger security warnings
   - Install extensions
   - Modify settings
5. Verify that no log files are created or populated → Test that opt-in persists across browser restarts
6. Access browser settings to find logging configuration → Verify that disabling logging stops log creation and offers deletion

Pass Criteria: Logging disabled by default AND no logs created without opt-in AND opt-in requires explicit user action AND privacy policy explains logging

Fail Criteria: Logging enabled by default OR logs created without consent OR opt-in is pre-selected OR unclear privacy policy

Evidence: Fresh installation testing, log directory inspection before/after opt-in, settings UI screenshots, privacy policy documentation, opt-in flow recordings

References:

- GDPR Consent Requirements: https://gdpr-info.eu/art-7-gdpr/
- Privacy by Default: https://ico.org.uk/for-organisations/guide-to-data-protection/guide-to-the-general-data-protection-regulation-gdpr/accountability-and-governance/data-protection-by-design-and-default/

- User Control Principles: https://www.w3.org/TR/design-principles/#user-control

Assessment: LOG-REQ-23 (User log management)

Reference: LOG-0-REQ-9 - Users shall be able to view and delete all local logs at any time

Given: A conformant browser with LOG-0 capability with local logging enabled

Task: Verify that users have complete control over their local logs, including the ability to view all logged data and permanently delete logs at any time, ensuring transparency about what information is being collected and empowering users to remove diagnostic data they no longer want stored, supporting user autonomy and privacy rights including the right to erasure.

Verification:

1. Enable local logging in browser settings → Test filtering and searching within logs
2. Generate various log entries through browser activities → Verify that "Delete Logs" or "Clear Logs" option is easily accessible
3. Access log viewing interface in browser settings or dedicated log viewer → Test deleting all logs and verify complete removal:
   - Check that log files are deleted from disk
   - Verify that log database is cleared
   - Check that no log remnants remain
4. Verify that all log categories are viewable: → Test selective log deletion (by category or date range)
   - Security events
   - Certificate errors
   - Extension activities
   - Crash reports
   - Audit trail entries
5. Test that log viewer displays logs in understandable format (not raw binary) → Verify that deletion is immediate and permanent
6. Verify that log entries include timestamps and event descriptions → Test that log deletion confirmation is clear about consequences

Pass Criteria: All logs are viewable in understandable format AND complete deletion is possible AND deletion is immediate and permanent AND selective deletion available

Fail Criteria: Logs not viewable OR incomplete deletion OR delayed deletion OR no selective options OR logs not in human-readable format

Evidence: Log viewer UI screenshots, log content examples, deletion workflow demonstrations, file system verification of deletion, selective deletion tests

References:

- GDPR Right to Erasure: https://gdpr-info.eu/art-17-gdpr/
- User Data Control: https://www.w3.org/TR/design-principles/#user-control
- Transparency Requirements: https://ico.org.uk/for-organisations/guide-to-data-protection/guide-to-the-general-data-protection-regulation-gdpr/individual-rights/right-to-erasure/

Assessment: LOG-REQ-24 (Telemetry opt-in requirement)

Reference: LOG-1-REQ-16 - Telemetry shall be disabled by default and require explicit opt-in

Given: A conformant browser with LOG-1 capability (opt-in telemetry)

Task: Verify that telemetry is disabled by default and requires explicit, informed opt-in consent before any diagnostic data is collected or transmitted, ensuring that users are not enrolled in telemetry programs without their knowledge, that consent is freely given and not bundled with other required actions, and that users understand what data will be collected and how it will be used before agreeing.

Verification:

1. Perform fresh browser installation → Verify that opt-in can be changed later in settings
2. Complete initial setup wizard → Test enabling telemetry post-installation
3. Verify that telemetry opt-in is presented clearly and separately: → Verify that enabling shows same clear information as initial setup
   - Not pre-checked or pre-selected
   - Separate from other required setup steps
   - Clear explanation of what data is collected
   - Clear explanation of how data is used
   - Link to detailed privacy policy
4. Test declining telemetry during setup → Test that partial opt-in is possible (selective categories)
5. Verify that browser functions normally without telemetry → Verify that opt-in status is clearly displayed in settings
6. Test that no telemetry is sent when declined → Test that revoking consent stops telemetry immediately

Pass Criteria: Telemetry disabled by default AND opt-in is explicit and not pre-checked AND clear data explanation provided AND consent can be revoked AND partial opt-in available

Fail Criteria: Telemetry enabled by default OR opt-in pre-checked OR unclear data description OR consent cannot be revoked OR all-or-nothing consent

Evidence: Installation wizard screenshots, telemetry opt-in dialog content, settings UI showing opt-in status, network traffic verification without consent, revocation workflow demonstrations

References:

- GDPR Consent Conditions: https://gdpr-info.eu/art-7-gdpr/
- Valid Consent Requirements: https://ico.org.uk/for-organisations/guide-to-data-protection/guide-to-the-general-data-protection-regulation-gdpr/consent/what-is-valid-consent/
- Privacy by Default: https://www.w3.org/TR/design-principles/#privacy

Assessment: LOG-REQ-25 (Granular telemetry control)

Reference: LOG-1-REQ-17 - Users shall have granular control over telemetry categories

Given: A conformant browser with LOG-1 capability with telemetry enabled

Task: Verify that users can exercise granular control over telemetry categories, enabling them to share some types of diagnostic data (such as crash reports) while withholding others (such as usage statistics), providing flexibility to balance privacy preferences with support for browser improvement, allowing users to contribute to security without sharing behavioral data.

Verification:

1. Access telemetry settings in browser → Verify that category changes take effect immediately
2. Verify that multiple telemetry categories are available: → Test that category selection persists across browser restarts
   - Crash reports and diagnostics
   - Security event reporting
   - Usage statistics and feature metrics
   - Performance measurements
   - Extension usage data
   - Error and warning reports
3. Test enabling/disabling each category independently → Verify that network traffic reflects only enabled categories
4. Verify that category descriptions clearly explain what data is included → Test that disabling all categories is equivalent to full opt-out

5. Test that selective enablement is respected: → Verify that category controls are easily accessible in settings
    - Enable only crash reports
    - Verify that only crashes are sent, not usage stats
    - Enable only security events
    - Verify that only security data is sent
6. Test various combinations of enabled categories → Check that documentation explains each category in detail

Pass Criteria: Multiple telemetry categories available AND independent control of each category AND clear category descriptions AND selective transmission respected AND changes immediate

Fail Criteria: No category separation OR all-or-nothing control OR unclear descriptions OR categories not independently controlled OR changes delayed

Evidence: Settings UI screenshots showing categories, category description content, selective enablement tests with network verification, documentation excerpts explaining categories

References:

- Privacy Control Granularity: https://www.w3.org/TR/design-principles/#user-control
- Data Minimization Principle: https://gdpr-info.eu/art-5-gdpr/
- Transparency and Choice: https://ico.org.uk/for-organisations/guide-to-data-protection/guide-to-the-general-data-protection-regulation-gdpr/principles/lawfulness-fairness-and-transparency/

Assessment: LOG-REQ-26 (Telemetry data documentation)

Reference: LOG-1-REQ-18 - Browser shall provide clear documentation of all collected data

Given: A conformant browser with LOG-1 capability

Task: Verify that the browser provides comprehensive, accessible documentation explaining exactly what telemetry data is collected, how it is used, where it is stored, how long it is retained, and who has access to it, enabling users to make informed decisions about consent and ensuring transparency about data practices in compliance with privacy regulations requiring clear information about data processing.

Verification:

1. Access browser privacy policy and telemetry documentation
2. Verify that documentation is easily findable:
    - Linked from settings
    - Linked from telemetry opt-in dialog
    - Available in help/support section
    - Accessible without creating account
3. Verify that documentation clearly explains:
    - Complete list of data types collected for each category
    - Purpose for collecting each data type
    - Legal basis for collection (consent, legitimate interest, etc.)
    - How data is anonymized or pseudonymized
    - Where data is stored (geographic location, infrastructure)
    - How long data is retained
    - Who has access to data (employees, third parties, partners)
    - How data is secured in transit and at rest
    - User rights regarding their data (access, deletion, portability)
4. Test that documentation is written in clear, non-technical language
5. Verify that technical details are available for advanced users
6. Test that documentation is available in multiple languages
7. Verify that documentation includes examples of actual data collected
8. Test that documentation is versioned and changes are tracked

9. Verify that users are notified of significant documentation changes
10. Check that contact information is provided for privacy questions

Pass Criteria: Comprehensive documentation available AND easily accessible AND clear non-technical language AND includes all required information AND examples provided AND changes tracked

Fail Criteria: Documentation incomplete OR hard to find OR overly technical OR missing key information OR no examples OR no change tracking

Evidence: Documentation excerpts covering all required areas, accessibility tests from various entry points, language analysis showing clarity, example data samples, version history, user notification examples

References:

- GDPR Transparency Requirements: https://gdpr-info.eu/art-12-gdpr/
- Privacy Policy Best Practices: https://www.priv.gc.ca/en/privacy-topics/privacy-policies/02_05_d_56/
- Clear Communication Guidelines: https://ico.org.uk/for-organisations/guide-to-data-protection/guide-to-the-general-data-protection-regulation-gdpr/individual-rights/right-to-be-informed/

Assessment: LOG-REQ-27 (Telemetry disable capability)

Reference: LOG-2-REQ-19 - Users shall be able to disable telemetry at any time

Given: A conformant browser with LOG-2 capability (default telemetry with opt-out)

Task: Verify that users can disable telemetry at any time through browser settings, even when telemetry is enabled by default, ensuring that users retain control over diagnostic data collection regardless of defaults, that disabling is immediate and comprehensive, and that users are not pressured or tricked into keeping telemetry enabled through dark patterns or complicated procedures.

Verification:

1. Start with browser with default telemetry enabled (LOG-2 mode) → Verify that browser functions normally with telemetry disabled
2. Access telemetry settings in browser preferences → Test that no telemetry is queued for later transmission
3. Verify that telemetry disable option is clearly visible and accessible: → Verify that settings UI clearly shows disabled status
   - Not buried in advanced settings
   - Clear labeling ("Disable Telemetry", "Turn Off Data Collection")
   - Single click or toggle to disable
   - No multi-step confirmation loops
4. Test disabling telemetry → Test that re-enabling shows clear opt-in again
5. Verify that disabling is immediate: → Verify that disabling telemetry is not bundled with other destructive actions
   - No "will take effect after restart" delays
   - Network monitoring shows immediate stop of telemetry
   - No final batch of data sent
6. Test that disabled state persists across browser restarts → Check that no dark patterns discourage disabling (scary warnings, hard-to-find option)

Pass Criteria: Disable option is easily accessible AND disabling is immediate AND disabled state persists AND browser functions normally AND no dark patterns present

Fail Criteria: Disable option hard to find OR disabling delayed OR state doesn't persist OR browser degraded OR dark patterns discourage disabling

Evidence: Settings UI screenshots, disable workflow recordings, network traffic showing immediate stop, persistence tests, functionality verification with telemetry disabled, dark pattern analysis

References:

- User Control Rights: https://www.w3.org/TR/design-principles/#user-control
- Dark Pattern Prevention: https://www.deceptive.design/
- Right to Object (GDPR): https://gdpr-info.eu/art-21-gdpr/

Assessment: LOG-REQ-28 (Telemetry status display)

Reference: LOG-2-REQ-20 - Browser shall display telemetry status in settings UI

Given: A conformant browser with LOG-2 capability (default telemetry with opt-out)

Task: Verify that the browser prominently displays current telemetry status in settings UI, enabling users to quickly understand whether diagnostic data is being collected and transmitted, what categories are active, and when the last transmission occurred, providing transparency and easy access to telemetry controls without requiring users to navigate through complex menu structures or consult documentation.

Verification:

1. Access browser settings/preferences → Verify that status is visible without opening advanced settings
2. Verify that telemetry status is visible in main settings view: → Test that status indicator distinguishes between:
   - Privacy or Data Collection section
   - Clear "Telemetry: Enabled/Disabled" indicator
   - Visual icon or status badge
   - Fully enabled telemetry
   - Partially enabled (some categories)
   - Fully disabled
   - Pending (waiting to send)
3. Test that status indicator accurately reflects current state → Verify that status information is available in "About" page or system info
4. Verify that clicking indicator provides detailed information: → Test that status is accessible via search in settings
   - What categories are enabled
   - Last telemetry transmission timestamp
   - Amount of data sent (if available)
   - Quick access to enable/disable controls
5. Test that status updates immediately when telemetry is toggled → Verify that status indicator includes timestamp of last update

Pass Criteria: Status visible in main settings AND accurate real-time status AND detailed information accessible AND updates immediately AND distinguishes states clearly

Fail Criteria: Status hidden in advanced settings OR inaccurate status OR no detailed info OR delayed updates OR states not distinguishable

Evidence: Settings UI screenshots showing status indicator, status detail panel captures, real-time update demonstrations, state differentiation examples, search functionality tests

References:

- Transparency in Design: https://www.w3.org/TR/design-principles/#transparency
- User Awareness: https://ico.org.uk/for-organisations/guide-to-data-protection/guide-to-the-general-data-protection-regulation-gdpr/individual-rights/right-to-be-informed/

Assessment: LOG-REQ-29 (Enterprise logging enforcement)

Reference: LOG-3-REQ-18 - Enterprise policies shall prevent users from disabling mandatory logging

Given: A conformant browser with LOG-3 capability (mandatory telemetry for enterprise) in managed environment

Task: Verify that enterprise policies can enforce mandatory logging that users cannot disable, ensuring that organizations can maintain required security monitoring, compliance logging, and audit trails regardless of individual user preferences, while clearly indicating to users that logging is enterprise-mandated and providing transparency about organizational data collection even when user control is restricted.

Verification:

1. Deploy browser in enterprise environment with mandatory logging policy → Verify that logs continue to be generated and transmitted
2. Access telemetry settings as standard user → Test that browser clearly indicates enterprise management:
   - Management indicator in settings
   - Policy explanation accessible to users
   - Contact information for IT department
3. Verify that logging disable controls are not available: → Verify that policy can specify exactly what shall be logged
   - Disable toggle/checkbox is grayed out or hidden
   - Settings indicate "Managed by your organization"
   - Clear explanation that policy prevents disabling
4. Test attempting to disable logging through all available methods: → Test that policy enforcement persists across browser updates
   - UI controls
   - Configuration file edits
   - Command-line flags
   - Registry/preference modifications
5. Verify that all disable attempts are blocked → Verify that policy changes propagate to managed browsers
6. Test that policy-mandated logging cannot be bypassed → Check that users can view what data is being collected even if they can't disable it

Pass Criteria: Enterprise policy prevents user disable of logging AND clear management indicators shown AND bypass attempts blocked AND users can view collected data AND policy enforcement persistent

Fail Criteria: Users can disable mandated logging OR no management indicators OR bypass possible OR users can't see data OR policy not enforced consistently

Evidence: Policy deployment documentation, managed browser screenshots showing disabled controls, bypass attempt logs, management indicator examples, policy change propagation tests

References:

- Enterprise Browser Management: https://chromeenterprise.google/policies/
- GDPR Article 88 (Processing in Employment Context): https://gdpr-info.eu/art-88-gdpr/
- Workplace Privacy Balance: https://ico.org.uk/for-organisations/guide-to-data-protection/guide-to-the-general-data-protection-regulation-gdpr/lawful-basis-for-processing/legitimate-interests/

Assessment: LOG-REQ-30 (Enterprise monitoring integration)

Reference: LOG-3-REQ-19 - Browser shall support integration with enterprise monitoring systems

Given: A conformant browser with LOG-3 capability in enterprise environment

Task: Verify that the browser supports integration with enterprise security information and event management (SIEM) systems, log aggregators, and monitoring platforms, enabling centralized security monitoring, incident detection, and compliance reporting across organizational browser deployments, using standard protocols and formats that interoperate with existing enterprise security infrastructure.

Verification:

1. Identify enterprise monitoring integration capabilities: → Verify that integration handles connection failures gracefully:
   - Syslog protocol support (RFC 5424/5424)

- Windows Event Log integration
- RESTful API for log retrieval
- File-based log export with rotation
- SIEM-specific connectors (Splunk, ELK, etc.)
- Local queuing of logs during outage
- Automatic retry logic
- No log loss during temporary failures

2. Configure browser to send logs to enterprise SIEM → Test that integration supports filtering and log level configuration
3. Verify that logs are transmitted in real-time or near-real-time → Verify that enterprise can query browser for current status
4. Test log format compatibility: → Test that integration works across different platforms (Windows, macOS, Linux)
   - Structured logs (JSON, XML, CEF)
   - Standardized field names
   - Consistent timestamps (ISO 8601, UTC)
   - Severity levels mapped correctly
5. Verify that integration supports authentication and encryption: → Verify that documentation includes integration guides for popular SIEMs
   - TLS for network transmission
   - API keys or certificates for authentication
   - Mutual TLS support
6. Test that all log categories are available through integration: → Test that integration performance doesn't degrade browser operation
   - Security events
   - Audit trails
   - Certificate errors
   - Extension activities

Pass Criteria: Standard protocol support (syslog, API) AND real-time transmission AND structured log formats AND authenticated/encrypted transmission AND handles failures gracefully AND cross-platform support

Fail Criteria: No standard protocols OR delayed transmission OR unstructured logs OR no authentication/encryption OR loses logs on failure OR platform-specific only

Evidence: SIEM integration configuration examples, log format samples, real-time transmission verification, authentication setup, failure recovery tests, cross-platform testing, performance impact analysis

References:

- Syslog Protocol: https://datatracker.ietf.org/doc/html/rfc5424
- Common Event Format: https://www.microfocus.com/documentation/arcsight/arcsight-smartconnectors/pdfdoc/commo event-format-v25/common-event-format-v25.pdf
- NIST SP 800-92 Log Management: https://csrc.nist.gov/publications/detail/sp/800-92/final

Assessment: LOG-REQ-31 (Tamper-evident logging)

Reference: LOG-3-REQ-20 - Browser shall provide tamper-evident logging mechanisms

Given: A conformant browser with LOG-3 capability (mandatory enterprise logging)

Task: Verify that logging mechanisms provide tamper-evidence to detect unauthorized modification or deletion of log entries, ensuring that attackers cannot cover their tracks by altering security logs, supporting forensic investigations and compliance requirements that mandate tamper-proof audit trails, providing cryptographic assurance that logs have not been modified since creation.

Verification:

1. Enable tamper-evident logging in browser or enterprise policy → Test that tamper detection is automatic and immediate
2. Verify tamper-evidence mechanisms in use: → Verify that tampered logs are flagged clearly
   - Cryptographic hashing of log entries
   - Digital signatures on log files
   - Merkle tree or hash chain linking entries
   - Append-only log storage
   - Write-once storage media support
3. Generate test log entries → Test that tamper evidence survives log export/archival
4. Verify that each entry has tamper-evident protection: → Verify that verification tools are provided to check log integrity
   - Hash or signature computed at creation
   - Hash chains link to previous entries
   - Timestamps are trusted (NTP, secure time source)
5. Attempt to tamper with logs: → Test that tamper evidence works with log rotation and archival
   - Modify log entry content
   - Delete log entries
   - Reorder log entries
   - Modify timestamps
6. Verify that tampering is detected: → Verify that cryptographic keys for signatures are protected
   - Hash verification fails
   - Signature verification fails
   - Hash chain breaks
   - Audit shows modification attempt

Pass Criteria: Cryptographic tamper-evidence active AND tampering is detected reliably AND detection is immediate AND evidence survives export AND verification tools provided

Fail Criteria: No tamper-evidence OR tampering not detected OR delayed detection OR evidence lost on export OR no verification tools

Evidence: Tamper-evidence mechanism documentation, log entry with hash/signature examples, tampering test results showing detection, verification tool demonstrations, export integrity tests, key protection verification

References:

- NIST SP 800-92 Log Integrity: https://csrc.nist.gov/publications/detail/sp/800-92/final
- Merkle Trees: https://en.wikipedia.org/wiki/Merkle_tree
- Digital Signatures: https://csrc.nist.gov/projects/digital-signatures
- Tamper-Evident Storage: https://csrc.nist.gov/glossary/term/tamper_evident

6.5 Update Mechanism Security Assessments

This section covers assessment procedures for requirements UPD-REQ-1 through UPD-REQ-23, addressing secure update delivery, signature verification, rollback protection, update channels, and update transparency.

Assessment: UPD-REQ-1 (Automatic update mechanism)

Reference: UPD-REQ-1 - Browser shall implement an automatic update mechanism that checks for and applies security updates without user intervention

Given: A conformant browser with UPD-1 or higher capability

Task: Verify that automatic updates prevent users from running vulnerable browser versions by ensuring security patches are promptly applied without requiring user awareness or action. Manual update processes fail because users often ignore update notifications or delay patching, leaving exploitable vulnerabilities active

for extended periods. Automatic updates close this security gap by applying critical fixes as soon as they become available.

Verification:

1. Install the browser and configure it to allow automatic updates (verify this is the default setting) → Test that the browser prompts for restart only when necessary for update installation
2. Check the browser's update settings to confirm automatic updates are enabled → Verify that the automatic update mechanism continues to function across browser restarts and system reboots
3. Access the browser's internal update interface through the settings menu → Automatic update checks occur at regular intervals without user action
4. Monitor the browser's update check schedule by reviewing internal logs or using network monitoring tools → Updates are downloaded in the background without interrupting browsing
5. Simulate an available update by configuring a test update server or using the browser's internal testing mechanisms → Update installation is automated with minimal or no user interaction required
6. Verify that the browser automatically checks for updates at the configured interval (typically daily) → Browser update status is clearly displayed to the user
7. Confirm that updates are downloaded in the background without blocking browser operation → Update checks occur even when the browser is running in background mode
8. Verify that updates are applied either at browser restart or in the background without user intervention → Failed update attempts are retried automatically

Pass Criteria: Browser automatically checks for updates at least daily AND downloads and applies updates without mandatory user intervention AND update status is visible to users

Fail Criteria: Automatic updates require manual user action to initiate OR updates fail to check automatically OR update mechanism can be permanently disabled by users

Evidence: Screenshots of automatic update settings and status pages, network packet captures showing update check requests, browser internal logs showing update schedule, time-stamped video of automatic update process, configuration file exports showing update settings

References:

- Chrome Update Architecture: https://chromium.googlesource.com/chromium/src/+/main/docs/updater/functional_spe
- Firefox Update System: https://wiki.mozilla.org/Software_Update
- Microsoft Edge Update Policies: https://docs.microsoft.com/en-us/deployedge/microsoft-edge-update-policies
- NIST SP 800-40 Guide to Enterprise Patch Management: https://csrc.nist.gov/publications/detail/sp/800-40/rev-4/final
- CIS Browser Security Benchmarks: https://www.cisecurity.org/benchmark/google_chrome

Assessment: UPD-REQ-2 (Update signature verification)

Reference: UPD-REQ-2 - Browser shall verify cryptographic signatures of all updates before installation using trusted public keys

Given: A conformant browser with UPD-1 or higher capability

Task: Verify that update signature verification prevents distribution of malicious browser versions by ensuring only authentic vendor-signed packages can be installed, even if update infrastructure is compromised or man-in-the-middle attacks occur. Without cryptographic signature verification, attackers who compromise update servers or intercept update traffic can inject backdoored browser binaries, resulting in complete system compromise for all affected users.

Verification:

1. Obtain a legitimate browser update package from the official distribution channel → Test that the browser rejects updates with missing signatures

2. Extract the update package and locate the cryptographic signature file or embedded signature → Confirm that signature verification occurs before any update content is executed
3. Verify the signature algorithm used (should be RSA-4096, ECDSA-P384, or stronger) → All update packages include valid cryptographic signatures
4. Attempt to modify the update package contents and observe that signature verification fails → Signature verification uses strong cryptographic algorithms (RSA-4096, ECDSA-P384, or better)
5. Create a test update package signed with an untrusted key and attempt to install it → Modified update packages are rejected due to signature verification failure
6. Monitor the browser's update process using system call tracing tools to verify signature verification occurs → Updates signed with untrusted keys are rejected
7. Check that the browser's trusted public keys are embedded in the binary or stored in a protected location → Signature verification occurs before any update code execution
8. Verify that signature verification failures prevent update installation and generate error logs → Verification failures are logged and reported to the user

Pass Criteria: All updates are cryptographically signed AND signatures are verified before installation AND modified or incorrectly signed updates are rejected

Fail Criteria: Updates can be installed without valid signatures OR signature verification can be bypassed OR weak cryptographic algorithms are used (RSA-2048 or weaker)

Evidence: Update package signature files, signature verification logs, network traces of update downloads showing signature transmission, system call traces showing verification process, test results from modified update packages, cryptographic algorithm analysis

References:

- Mozilla Code Signing Policy: https://wiki.mozilla.org/Security/Binary_Transparency
- NIST FIPS 186-5 Digital Signature Standard: https://csrc.nist.gov/publications/detail/fips/186/5/final
- Authenticode Code Signing (Windows): https://docs.microsoft.com/en-us/windows-hardware/drivers/install/authenticod
- Apple Code Signing Guide: https://developer.apple.com/library/archive/documentation/Security/Conceptual/CodeSigni
- OWASP Code Signing Best Practices: https://cheatsheetseries.owasp.org/cheatsheets/Third_Party_Javascript_Manager

Assessment: UPD-REQ-3 (HTTPS-only update delivery)

Reference: UPD-REQ-3 - Browser shall download all updates exclusively over HTTPS with certificate validation and pinning

Given: A conformant browser with UPD-1 or higher capability

Task: Verify that HTTPS-only update delivery protects update packages from interception and modification during transit, preventing network attackers from injecting malicious code even before signature verification. While signature verification provides end-to-end authenticity, transport encryption prevents metadata leakage about which browser versions are vulnerable and adds defense-in-depth against implementation flaws in signature verification.

Verification:

1. Configure a network proxy to intercept browser update traffic → Test that the browser verifies the update server's hostname matches the certificate
2. Monitor network traffic during an update check to verify all connections use HTTPS → Confirm that update requests include proper TLS configurations (strong cipher suites, no deprecated protocols)
3. Attempt to redirect update requests to HTTP endpoints using DNS spoofing or proxy manipulation → All update downloads occur exclusively over HTTPS connections
4. Verify that the browser rejects HTTP update URLs and only accepts HTTPS → HTTP update URLs are rejected or automatically upgraded to HTTPS
5. Test certificate validation by presenting an invalid certificate for the update server → TLS 1.2 or TLS 1.3 is enforced for all update traffic

6. Check if the browser implements certificate pinning for update domains → Certificate validation is performed with proper hostname verification
7. Attempt a man-in-the-middle attack on the update channel using a rogue certificate → Certificate pinning is implemented for update domains
8. Verify that TLS 1.2 or higher is used for all update connections → Man-in-the-middle attacks on update channels are detected and blocked

- Strong cipher suites are negotiated (AES-GCM, ChaCha20-Poly1305)
- Deprecated protocols (TLS 1.0, TLS 1.1, SSLv3) are rejected

Pass Criteria: All update traffic uses HTTPS with TLS 1.2+ AND certificate validation is enforced AND HTTP update URLs are rejected AND certificate pinning is implemented for critical update domains

Fail Criteria: Updates can be downloaded over HTTP OR TLS 1.1 or earlier is accepted OR certificate validation can be bypassed OR no certificate pinning is implemented

Evidence: Network packet captures showing HTTPS-only update traffic, TLS handshake analysis, certificate chain validation logs, test results from HTTP redirect attempts, man-in-the-middle attack test results, cipher suite negotiation logs

References:

- Chrome Update Server Pinning: https://www.chromium.org/Home/chromium-security/security-faq/#TOC-How-does-key-pinning-work-in-Chrome-
- Mozilla Update Server Security: https://wiki.mozilla.org/Security/Server_Side_TLS
- NIST SP 800-52 TLS Guidelines: https://csrc.nist.gov/publications/detail/sp/800-52/rev-2/final
- IETF RFC 8446 TLS 1.3: https://www.rfc-editor.org/rfc/rfc8446
- OWASP Transport Layer Security: https://cheatsheetseries.owasp.org/cheatsheets/Transport_Layer_Security_Cheat_S
- Certificate Pinning Best Practices: https://owasp.org/www-community/controls/Certificate_and_Public_Key_Pinning

Assessment: UPD-REQ-4 (Update manifest integrity)

Reference: UPD-REQ-4 - Browser shall verify the integrity of update manifests containing version information, file hashes, and metadata

Given: A conformant browser with UPD-1 or higher capability

Task: Verify that update manifest integrity protects the update metadata from tampering, ensuring that version information, file hashes, and download URLs cannot be manipulated by attackers to redirect updates to malicious files or bypass security checks. Compromised manifests can trick browsers into downloading wrong versions or accepting manipulated update files even when signature verification is implemented.

Verification:

1. Capture an update manifest file during a browser update check (typically JSON or XML format) → Confirm that manifest validation failures are logged and prevent update installation
2. Examine the manifest structure to verify it includes: version number, file hashes, file sizes, signature, and metadata → Update manifests are cryptographically signed and signatures are verified
3. Verify that the manifest itself is signed and the signature is validated before processing → Manifests include complete metadata: version, file hashes (SHA-256+), sizes, timestamps
4. Attempt to modify the manifest file (change version number or file hashes) and verify it is rejected → Modified manifests are rejected due to signature verification failures
5. Test that each file listed in the manifest includes a cryptographic hash (SHA-256 or stronger) → Each update file is verified against manifest hashes before installation
6. Verify that downloaded update files are checked against the manifest hashes before installation → Hash mismatches prevent installation and generate error logs
7. Test that mismatched file hashes prevent installation and trigger error handling → Manifests include rollback protection information (minimum version)

8. Check that the manifest includes timestamp information to prevent replay attacks → Timestamp validation prevents replay attacks
9. Verify that the manifest specifies minimum supported versions for rollback protection → Manifest structure follows a well-defined schema

Pass Criteria: Update manifests are signed and verified AND contain file hashes (SHA-256+) for all components AND hash verification is enforced before installation AND manifest tampering is detected and rejected

Fail Criteria: Manifests are not signed OR file hashes are missing or not verified OR modified manifests are accepted OR weak hash algorithms (MD5, SHA-1) are used

Evidence: Update manifest files in JSON/XML format, manifest signature verification logs, test results from modified manifests, hash verification logs, network captures showing manifest downloads, schema validation results

References:

- Chrome Update Manifest Format: https://chromium.googlesource.com/chromium/src/+/main/docs/updater/protocol_3
- The Update Framework (TUF) Specification: https://theupdateframework.github.io/specification/latest/
- NIST SP 800-107 Hash Function Recommendations: https://csrc.nist.gov/publications/detail/sp/800-107/rev-1/final
- Google Omaha Protocol: https://github.com/google/omaha/blob/main/doc/ServerProtocolV3.md
- OWASP Software Supply Chain Security: https://owasp.org/www-project-software-component-verification-standard/

Assessment: UPD-REQ-5 (Rollback protection)

Reference: UPD-REQ-5 - Browser shall implement rollback protection to prevent installation of older versions with known vulnerabilities

Given: A conformant browser with UPD-1 or higher capability

Task: Verify that rollback protection prevents attackers from downgrading browsers to older versions containing known exploitable vulnerabilities, blocking a common attack vector where adversaries force installation of vulnerable software to exploit publicly disclosed flaws. Without rollback protection, attackers who compromise update infrastructure or perform man-in-the-middle attacks can systematically downgrade browser installations to versions with known, weaponized exploits.

Verification:

1. Identify the currently installed browser version number → Confirm that rollback protection errors are logged with version information
2. Obtain an older version of the browser update package (at least 2-3 versions older) → Browser rejects installation of older versions through automatic update mechanism
3. Attempt to install the older update package through the automatic update mechanism → Minimum version numbers are enforced based on security requirements
4. Verify that the browser rejects the installation with an appropriate error message → Manual downgrade attempts are blocked or generate security warnings
5. Check the browser's update configuration for minimum version enforcement policies → Browser maintains persistent version history to detect rollback attempts
6. Test that the browser maintains a record of the highest version ever installed → Critical security milestone versions cannot be downgraded
7. Attempt to manually downgrade the browser by installing an older version package → Rollback protection applies to both full updates and component updates
8. Verify that critical security updates cannot be rolled back even by administrators → Clear error messages indicate rollback protection enforcement
9. Test that the rollback protection persists across browser reinstallations (if applicable) → Version comparison logic handles all version number formats correctly

Pass Criteria: Browser prevents installation of older versions through automatic updates AND maintains version history to detect rollbacks AND critical security versions cannot be downgraded

Fail Criteria: Older versions can be installed through update mechanism OR no minimum version enforcement exists OR rollback protection can be easily bypassed

Evidence: Update rejection logs showing version mismatch errors, test results from downgrade attempts, version history configuration files, error messages from rollback protection, policy configuration showing minimum version requirements

References:

- Chrome Version Enforcement: https://chromeenterprise.google/policies/#TargetVersionPrefix
- Android Rollback Protection: https://source.android.com/docs/security/features/verifiedboot/verified-boot#rollback-protection
- TUF Rollback Attack Prevention: https://theupdateframework.io/docs/security/
- NIST Cybersecurity Framework - Update Management: https://www.nist.gov/cyberframework
- Mozilla Update Security Guidelines: https://wiki.mozilla.org/Security/Server_Side_TLS
- Microsoft Update Rollback Protection: https://docs.microsoft.com/en-us/windows/deployment/update/waas-restart

Assessment: UPD-REQ-6 (Update channel isolation)

Reference: UPD-REQ-6 - Browser shall maintain isolation between update channels (stable, beta, dev) with independent signature verification

Given: A conformant browser with UPD-1 or higher capability supporting multiple channels

Task: Verify that update channel isolation prevents attackers from injecting unstable or experimental browser versions into production deployments by ensuring strict separation between stable, beta, and development update streams. Without channel isolation, compromised development infrastructure or testing servers could be leveraged to push untested code to stable channel users, introducing vulnerabilities or instability into production environments.

Verification:

1. Identify the available update channels for the browser (stable, beta, dev, canary) → Confirm that cross-channel update injection attempts are logged and rejected
2. Install the browser in stable channel mode and verify the update channel configuration → Each update channel has distinct update server endpoints
3. Examine the update server URLs for each channel to verify they are distinct → Channel-specific signature verification prevents cross-channel update injection
4. Attempt to inject a beta or dev channel update into a stable channel installation → Update manifests include channel identification that is verified
5. Verify that each channel uses separate signature verification keys or key policies → Channel switching requires explicit user consent and configuration changes
6. Test that switching between channels requires explicit user action and confirmation → Downgrading channels (stable to beta/dev) triggers security warnings
7. Check that channel metadata is included in update manifests and verified → Channel isolation prevents malicious update server redirection
8. Attempt to downgrade from stable to beta channel and verify the security implications are communicated → Each channel maintains independent version history and rollback protection
9. Verify that each channel has independent version numbering and rollback protection → Cross-channel update attempts are detected and logged

Pass Criteria: Update channels use separate server endpoints AND channel-specific signature verification prevents injection AND channel switching requires explicit user action AND cross-channel updates are rejected

Fail Criteria: Update channels share signature keys OR cross-channel updates can be injected OR no channel verification in update manifests OR channel switching occurs without user consent

Evidence: Update server URL configurations for each channel, signature verification key policies, channel metadata in update manifests, test results from cross-channel injection attempts, channel switching logs, network traces showing channel-specific endpoints

References:

- Chrome Release Channels: https://www.chromium.org/getting-involved/dev-channel/
- Firefox Release Management: https://wiki.mozilla.org/Release_Management/Release_Process
- Microsoft Edge Channels: https://docs.microsoft.com/en-us/deployedge/microsoft-edge-channels
- Google Chrome Enterprise Channel Management: https://support.google.com/chrome/a/answer/9982578
- NIST Secure Software Development Framework: https://csrc.nist.gov/Projects/ssdf

Assessment: UPD-REQ-7 (Component update support)

Reference: UPD-REQ-7 - Browser shall support independent security updates for components (rendering engine, JavaScript engine, libraries) without full browser updates

Given: A conformant browser with UPD-2 or higher capability

Task: Verify that component-level updates enable rapid patching of critical subsystems like JavaScript engines or rendering engines without waiting for full browser release cycles, reducing the window of exposure for component-specific vulnerabilities. Monolithic update systems delay security fixes because all components should be tested together, while independent component updates allow targeted, accelerated security patching for high-risk subsystems.

Verification:

1. Identify the browser's major components that support independent updates (e.g., V8, Chromium base, libraries) → Verify that component update status is visible in the browser's update interface
2. Monitor the update mechanism to detect component-specific update packages → Individual components can be updated independently of full browser updates
3. Verify that component updates include version information and dependency specifications → Component updates include version and dependency metadata
4. Test that component updates are applied without requiring a full browser restart when possible → Component updates are signed and verified like full browser updates
5. Attempt to install incompatible component versions and verify dependency checking prevents installation → Incompatible component versions are rejected based on dependency checking
6. Check that component updates follow the same security verification as full updates (signatures, HTTPS) → Component updates can be applied with minimal or no browser restart
7. Verify that component update manifests specify compatibility with browser versions → Critical security components can be updated on accelerated schedules
8. Test that critical component vulnerabilities can be patched independently of the full release schedule → Component update history is tracked separately from full browser versions
9. Confirm that component updates maintain rollback protection independently → Update interface displays component-level version information

Pass Criteria: Browser supports independent component updates with signature verification AND dependency checking prevents incompatible installations AND component updates follow same security verification as full updates

Fail Criteria: No support for component-level updates OR component updates bypass security verification OR no dependency checking for component compatibility

Evidence: Component update manifest files, component version listings from browser internals, update logs showing component-specific updates, dependency verification results, signature verification for component updates, test results from incompatible component installations

References:

- Chrome Component Updates: https://chromium.googlesource.com/chromium/src/+/main/components/component_upd
- V8 Engine Versioning: https://v8.dev/docs/version-numbers
- WebRTC Component Updates: https://webrtc.github.io/webrtc-org/release-notes/
- NIST Software Component Verification: https://csrc.nist.gov/Projects/cyber-supply-chain-risk-management
- OWASP Dependency Check: https://owasp.org/www-project-dependency-check/

Assessment: UPD-REQ-8 (Emergency update capability)

Reference: UPD-REQ-8 - Browser shall support emergency update mechanism for critical zero-day vulnerabilities with accelerated deployment

Given: A conformant browser with UPD-1 or higher capability

Task: Verify that emergency update mechanisms enable rapid deployment of critical zero-day patches to all users before exploit code becomes widely available, minimizing the window of vulnerability exploitation. Standard rollout schedules of days or weeks are unacceptable for actively exploited vulnerabilities, requiring accelerated deployment paths that bypass normal staged rollouts while maintaining security verification integrity.

Verification:

1. Review the browser's update documentation for emergency or critical update procedures → Confirm that emergency update mechanisms include additional verification to prevent abuse
2. Examine update check frequency configuration and verify it can be increased for critical updates → Browser supports accelerated update checks for critical security updates
3. Test that the browser can be configured to prioritize critical security updates over feature updates → Emergency updates include priority flags in update manifests
4. Simulate an emergency update scenario by configuring a test update with high-priority flags → Critical updates bypass staged rollout mechanisms for faster deployment
5. Verify that emergency updates bypass normal staged rollout delays → Users receive prominent notifications for emergency security updates
6. Test that users receive prominent notifications for critical security updates requiring immediate action → Emergency updates can trigger forced restarts with clear security justification
7. Check that emergency updates can trigger forced restarts with appropriate user warnings → Emergency update flags are authenticated and cannot be spoofed
8. Verify that emergency update flags cannot be spoofed by malicious update servers → Audit logs distinguish emergency updates from regular updates
9. Test that emergency updates are logged with high-priority markers for audit purposes → Emergency update mechanism includes safeguards against abuse

Pass Criteria: Browser supports emergency update mechanism with accelerated deployment AND emergency updates bypass normal rollout delays AND priority flags are authenticated AND users are clearly notified of critical updates

Fail Criteria: No emergency update mechanism exists OR emergency updates follow normal rollout schedule OR priority flags can be spoofed OR users cannot distinguish critical from regular updates

Evidence: Emergency update configuration documentation, test results from simulated critical updates, update priority flags in manifests, notification screenshots for critical updates, audit logs showing emergency update markers, rollout bypass verification

References:

- Chrome Critical Update Deployment: https://chromereleases.googleblog.com/
- Firefox Critical Updates: https://www.mozilla.org/en-US/security/advisories/
- NIST SP 800-40 Emergency Patching: https://csrc.nist.gov/publications/detail/sp/800-40/rev-4/final

- CERT Vulnerability Disclosure: https://vuls.cert.org/confluence/display/Wiki/Vulnerability+Disclosure+Policy
- Microsoft Security Response Center: https://www.microsoft.com/en-us/msrc

Assessment: UPD-REQ-9 (Update verification before installation)

Reference: UPD-REQ-9 - Browser shall perform complete integrity verification of all update files before beginning installation process

Given: A conformant browser with UPD-1 or higher capability

Task: Verify that update verification prevents installation of compromised or tampered browser builds, ensuring only authentic vendor-signed updates can be applied even if the update server or distribution network is compromised. This prevents attackers from injecting malicious code through man-in-the-middle attacks, compromised CDNs, or corrupted downloads that bypass cryptographic verification.

Verification:

1. Monitor the browser update process to identify the verification phase before installation → Verify that the verification process cannot be bypassed or interrupted
2. Capture update files during download and verify that hashes match the update manifest → All update files are verified against cryptographic hashes before installation
3. Corrupt an update file after download but before installation to test verification → Hash algorithms used are SHA-256 or stronger
4. Verify that corrupted files are detected and re-downloaded before installation proceeds → Corrupted or modified files are detected and rejected
5. Test that partial downloads are detected and not installed → Partial downloads are detected and completed before installation
6. Check that all update files are verified against the signed manifest before any installation steps → Installation begins only after complete verification of all update components
7. Verify that installation only begins after all files pass integrity checks → Verification failures trigger automatic re-download or error reporting
8. Test that verification failures trigger error handling and logging → Failed verifications are logged with specific error information
9. Confirm that failed verifications do not leave the browser in an unstable state → Browser remains in stable state if verification fails

Pass Criteria: All update files are verified with cryptographic hashes (SHA-256+) before installation AND corrupted files are detected and re-downloaded AND installation only proceeds after complete verification

Fail Criteria: Update files are installed without verification OR weak hash algorithms are used OR corrupted files can be installed OR verification can be bypassed

Evidence: Update verification logs showing hash checks, test results from corrupted file installations, network traces showing re-download behavior, hash algorithm analysis, installation process flow documentation, error handling logs

References:

- Chrome Update Verification: https://chromium.googlesource.com/chromium/src/+/main/docs/updater/functional_spec
- Mozilla Update Integrity Checks: https://wiki.mozilla.org/Software_Update#Integrity_Checks
- NIST Hash Function Security: https://csrc.nist.gov/projects/hash-functions
- TUF Consistent Snapshot Protection: https://theupdateframework.github.io/specification/latest/#consistent-snapshots
- ISO/IEC 29147 Vulnerability Disclosure: https://www.iso.org/standard/72311.html

Assessment: UPD-REQ-10 (Update failure recovery)

Reference: UPD-REQ-10 - Browser shall implement robust failure recovery mechanisms to restore functionality if update installation fails

Given: A conformant browser with UPD-1 or higher capability

Task: Verify that update failure recovery mechanisms maintain browser availability and security posture even when updates fail, preventing denial-of-service scenarios where failed updates render the browser unusable or leave it in a vulnerable state. This ensures users retain access to a functional, secure browser and prevents attackers from exploiting update failures to create persistent availability issues.

Verification:

1. Create a test scenario where update installation fails (disk full, permission denied, file corruption) → Confirm that recovery processes are logged for troubleshooting
2. Trigger an update and simulate installation failure at various stages (download, verification, installation) → Failed updates do not prevent browser from launching or functioning
3. Verify that the browser rolls back to the previous working version after installation failure → Browser automatically rolls back to previous version after installation failures
4. Test that the browser remains functional and can be launched after a failed update → User data and settings are preserved through failed update attempts
5. Verify that failed update state is detected and reported to the user with clear messaging → Clear error messages indicate update failure causes
6. Test that the browser automatically retries failed updates with exponential backoff → Automatic retry mechanisms with exponential backoff are implemented
7. Simulate power loss or system crash during update installation → Interrupted updates (power loss, crash) are detected and recovered on next launch
8. Verify that the browser can recover from interrupted updates on next launch → Recovery processes maintain browser stability and security
9. Test that update failure recovery preserves user data and settings → All recovery actions are logged with timestamps and error codes

Pass Criteria: Browser remains functional after update failures AND automatically rolls back to working version AND retries failed updates automatically AND preserves user data through failures

Fail Criteria: Failed updates prevent browser launch OR no rollback mechanism exists OR user data is corrupted by failed updates OR no automatic retry for failed updates

Evidence: Update failure logs with error codes, rollback process documentation, test results from simulated failures (disk full, crash, corruption), user data integrity verification, retry attempt logs with backoff timing, recovery process screenshots

References:

- Chrome Update Recovery: https://chromium.googlesource.com/chromium/src/+/main/docs/updater/functional_spec.m handling
- NIST Resilience Engineering: https://csrc.nist.gov/glossary/term/resilience
- Google Omaha Error Handling: https://github.com/google/omaha/blob/main/doc/ServerProtocolV3.md#error-codes

Assessment: UPD-REQ-11 (Update transparency logging)

Reference: UPD-REQ-11 - Browser shall implement update transparency logging to create auditable records of all update activities

Given: A conformant browser with UPD-1 or higher capability and LOG-1 or higher capability

Task: Verify that update transparency logging creates auditable records of all update activities, enabling detection of compromised update infrastructure, supply chain attacks, or unauthorized modifications to the browser. This provides forensic evidence for security incident investigation and enables organizations to verify that only legitimate, authorized updates were applied to their browser fleet.

Verification:

1. Enable update logging through browser configuration or developer tools → Test that update logs are protected from tampering and include integrity verification
2. Perform a complete update cycle and capture all generated logs → All update activities are logged with timestamps and version information
3. Verify that logs include: update check timestamps, available versions, download start/completion, verification results, installation status → Logs include download sources, file hashes, and signature verification results
4. Test that update logs include cryptographic hashes of installed components → Update failures are logged with detailed error information
5. Verify that signature verification results are logged with key identifiers → User actions related to updates are captured in logs
6. Check that update server URLs and responses are logged for audit purposes → Logs are structured and machine-readable (JSON or similar format)
7. Test that all update failures are logged with specific error codes and context → Update logs can be exported for compliance and audit purposes
8. Verify that logs include user actions related to updates (manual checks, deferrals, channel changes) → Logs include sufficient detail for security incident investigation
9. Confirm that update logs can be exported for external analysis or compliance reporting → Log integrity is protected through checksums or signing

Pass Criteria: All update activities are logged with complete details AND logs include hashes and verification results AND logs are exportable in structured format AND log integrity is protected

Fail Criteria: Update activities are not logged OR logs lack critical information (timestamps, versions, hashes) OR logs cannot be exported OR logs can be tampered with

Evidence: Complete update log files showing full update cycle, exported logs in JSON/structured format, log integrity verification results, test results showing failure logging, screenshots of update log interfaces, compliance report samples

References:

- Chrome Update Logging: https://chromium.googlesource.com/chromium/src/+/main/docs/updater/functional_spec.md
- Binary Transparency: https://wiki.mozilla.org/Security/Binary_Transparency
- NIST SP 800-92 Log Management: https://csrc.nist.gov/publications/detail/sp/800-92/final
- Google Binary Authorization: https://cloud.google.com/binary-authorization/docs
- Certificate Transparency RFC 6962: https://www.rfc-editor.org/rfc/rfc6962
- OWASP Logging Cheat Sheet: https://cheatsheetseries.owasp.org/cheatsheets/Logging_Cheat_Sheet.html

Assessment: UPD-REQ-12 (Delta update security)

Reference: UPD-REQ-12 - Browser shall securely implement delta updates (patches) with same security verification as full updates

Given: A conformant browser with UPD-2 or higher capability supporting delta updates

Task: Verify that delta update security prevents attackers from exploiting differential patching mechanisms to inject malicious code through crafted patch files that bypass full binary verification. Delta updates introduce unique attack vectors where malicious patches could transform a legitimate binary into a compromised one if the patch itself and the resulting binary are not both cryptographically verified against known-good hashes.

Verification:

1. Monitor the browser update mechanism to detect delta update packages (smaller than full updates) → Confirm that delta update security matches or exceeds full update security
2. Verify that delta updates are offered only when the current version is compatible → Delta updates are cryptographically signed and verified before application
3. Capture a delta update package and examine its structure and signature → Source version verification ensures delta is compatible with installed version

4. Verify that delta updates are cryptographically signed independently from full updates → Resulting files after delta application are verified against target hashes
5. Test that applying a delta update includes verification of the resulting files against expected hashes → Incompatible delta updates are rejected with fallback to full updates
6. Attempt to apply a delta update to an incompatible base version and verify it is rejected → Delta update application is atomic (complete or rollback)
7. Test that delta update application includes atomic operations (all-or-nothing installation) → Both patch integrity and result integrity are verified
8. Verify that failed delta updates can fall back to full update downloads → Delta updates use same or stronger cryptography as full updates
9. Check that delta updates include integrity checks for both the patch and the result → Delta update failures trigger automatic fallback mechanisms

Pass Criteria: Delta updates are signed and verified independently AND source version compatibility is checked AND resulting files are verified against target hashes AND fallback to full updates on failure

Fail Criteria: Delta updates bypass security verification OR no source version checking OR result files not verified OR no fallback mechanism for failed deltas

Evidence: Delta update package analysis showing signatures, delta vs full update size comparisons, verification logs for delta application, test results from incompatible delta applications, fallback mechanism demonstrations, atomic operation verification

References:

- Google Courgette (Delta Compression): https://www.chromium.org/developers/design-documents/software-updates-courgette/
- Binary Diff Security: https://theupdateframework.github.io/specification/latest/#targets-metadata
- Mozilla MAR Format (Mozilla Archive): https://wiki.mozilla.org/Software_Update:MAR
- Microsoft Delta Updates: https://docs.microsoft.com/en-us/windows/deployment/update/psfxwhitepaper
- NIST Software Patch Security: https://csrc.nist.gov/glossary/term/patch

Assessment: UPD-REQ-13 (Update server authentication)

Reference: UPD-REQ-13 - Browser shall authenticate update servers using certificate validation, pinning, and domain verification

Given: A conformant browser with UPD-1 or higher capability

Task: Verify that update server authentication prevents man-in-the-middle attacks and DNS hijacking attempts that could redirect browsers to malicious update servers distributing compromised builds. Certificate pinning and domain verification ensure that even if an attacker obtains a valid TLS certificate through a compromised Certificate Authority or DNS spoofing, the browser will reject connections to fraudulent update servers.

Verification:

1. Capture network traffic during update checks to identify update server domains → Confirm that failed authentication prevents update checks and is logged
2. Examine TLS certificates presented by update servers for validity and chain of trust → Update servers present valid TLS certificates with complete chain of trust
3. Verify that update server domains match expected values hardcoded in browser or configuration → Certificate pinning is implemented for update server domains
4. Test certificate pinning by attempting to present a valid but unpinned certificate for update domain → Update server domains are verified against expected values
5. Attempt DNS spoofing to redirect update requests to malicious servers → DNS spoofing attempts are detected through certificate pinning
6. Verify that update server certificate validation includes OCSP/CRL checks → Certificate revocation status is checked via OCSP or CRL

7. Test that expired or revoked certificates for update servers are rejected → Expired or revoked certificates cause update failures
8. Check that update server authentication occurs before any update data is processed → HSTS is enforced for update server domains
9. Verify that update server domains use HSTS to prevent protocol downgrade attacks → Authentication failures are logged and prevent update downloads

Pass Criteria: Update servers are authenticated with certificate validation AND certificate pinning is implemented AND domain verification prevents spoofing AND revocation checking is performed

Fail Criteria: No certificate pinning for update servers OR domain verification can be bypassed OR revocation checking is not performed OR expired certificates are accepted

Evidence: Update server TLS certificates and chains, certificate pinning configurations, network traces showing authentication, test results from DNS spoofing attempts, OCSP/CRL check logs, HSTS policy verification, authentication failure logs

References:

- Chrome Certificate Pinning: https://www.chromium.org/Home/chromium-security/security-faq/#TOC-How-does-key-pinning-work-in-Chrome-
- Certificate Transparency Monitoring: https://certificate.transparency.dev/
- IETF RFC 7469 Public Key Pinning: https://www.rfc-editor.org/rfc/rfc7469
- OCSP Stapling: https://www.rfc-editor.org/rfc/rfc6066#section-8
- Mozilla Update Server Security: https://wiki.mozilla.org/Security/Server_Side_TLS
- OWASP Certificate Pinning: https://owasp.org/www-community/controls/Certificate_and_Public_Key_Pinning

Assessment: UPD-REQ-14 (Update timing jitter)

Reference: UPD-REQ-14 - Browser shall implement randomized timing jitter for update checks to prevent server load spikes and timing analysis

Given: A conformant browser with UPD-1 or higher capability

Task: Verify that update timing jitter prevents thundering herd problems that could enable denial-of-service attacks against update infrastructure or timing analysis attacks that reveal browser deployment patterns. Synchronized update checks from millions of browsers could overwhelm update servers or allow attackers to identify organizational update policies through traffic analysis, making randomized jitter a critical availability and privacy protection.

Verification:

1. Configure multiple browser instances (at least 10) with synchronized clocks → Confirm that jitter is cryptographically random, not predictable
2. Monitor update check timing for each instance over a 24-hour period → Update checks include randomized timing jitter to distribute server load
3. Calculate the distribution of update check times relative to the scheduled interval → Jitter range is appropriate for the update check frequency (typically ±10-20%)
4. Verify that update checks are not synchronized across all instances → Multiple instances do not synchronize update checks at same time
5. Measure the jitter range (randomization window) applied to update checks → Jitter uses cryptographically secure random number generation
6. Test that jitter is applied even when manual update checks are performed → Critical updates can bypass jitter for immediate deployment when needed
7. Verify that jitter does not delay critical security updates beyond acceptable windows → Jitter prevents timing analysis attacks on update behavior
8. Check that jitter implementation prevents timing analysis attacks → Jitter does not excessively delay important security updates

9. Test that jitter ranges are appropriate (e.g., ±1-2 hours for daily checks) → Update check distribution follows expected random distribution

Pass Criteria: Update checks include random jitter of at least ±10% of check interval AND jitter uses cryptographic randomness AND multiple instances show distributed check times AND critical updates can bypass jitter

Fail Criteria: No timing jitter implemented OR jitter is predictable OR all instances synchronize checks OR critical updates are delayed by jitter

Evidence: Update check timing logs from multiple instances, statistical analysis of check time distribution, jitter range configuration, randomness quality analysis, test results showing de-synchronized checks, critical update bypass demonstrations

References:

- Chrome Update Timing: https://chromium.googlesource.com/chromium/src/+/main/docs/updater/functional_spec.md
checks
- Thundering Herd Problem: https://en.wikipedia.org/wiki/Thundering_herd_problem
- NIST Randomness Recommendations: https://csrc.nist.gov/projects/random-bit-generation
- Google Omaha Protocol Timing: https://github.com/google/omaha/blob/main/doc/ServerProtocolV3.md#update-check-timing
- Load Balancing Best Practices: https://aws.amazon.com/architecture/well-architected/

Assessment: UPD-REQ-15 (Background update enforcement)

Reference: UPD-REQ-15 - Browser shall enforce background update processes that continue even when browser is not actively running

Given: A conformant browser with UPD-2 or higher capability

Task: Verify that background update enforcement ensures security updates are applied even when users rarely launch the browser, preventing scenarios where unpatched browsers accumulate critical vulnerabilities. Without background updates, attackers can target users who infrequently use their browsers but still have them installed, exploiting the extended window of vulnerability between releases and actual patching.

Verification:

1. Close all browser windows completely to ensure browser is not running → Confirm that background updates respect system resource constraints (network metering, battery status)
2. Monitor system processes to verify background update service remains active → Background update service remains active when browser is closed
3. Wait for the scheduled update check interval with browser closed → Update checks occur on schedule regardless of browser running state
4. Verify that update checks occur even when browser is not running → Background service starts automatically at system boot
5. Test that background update service starts automatically at system boot → Updates can download and install without browser being open
6. Simulate an available update and verify it downloads in background while browser is closed → Background service runs with minimal necessary privileges
7. Test that background updates can wake the system from sleep if configured (platform-dependent) → Service cannot be easily disabled by users or malicious software
8. Verify that background update service has appropriate system permissions but runs with minimal privileges → Resource-aware update behavior (respects metered connections, battery)
9. Check that background update process is resistant to termination by users or malware → Background service restarts automatically if terminated abnormally

Pass Criteria: Background update service runs independently of browser AND performs update checks on schedule when browser closed AND downloads updates in background AND restarts automatically if

terminated

Fail Criteria: Update service requires browser to be running OR no background update checks when closed OR service can be easily disabled OR does not restart after termination

Evidence: System process listings showing background service, update logs showing checks while browser closed, service configuration and permissions, automatic restart verification, resource usage monitoring, test results from service termination attempts

References:

- Chrome Background Updates: https://chromium.googlesource.com/chromium/src/+/main/docs/updater/functional_spe mode
- Windows Update Service Architecture: https://docs.microsoft.com/en-us/windows/deployment/update/how-windows-update-works
- macOS Launch Agents: https://developer.apple.com/library/archive/documentation/MacOSX/Conceptual/BPSystemSta
- Linux systemd Services: https://www.freedesktop.org/software/systemd/man/systemd.service.html
- Firefox Background Update Service: https://support.mozilla.org/en-US/kb/enable-background-updates-firefox-windows
- NIST Automated Patch Management: https://csrc.nist.gov/publications/detail/sp/800-40/rev-4/final

Assessment: UPD-REQ-16 (Update notification UI)

Reference: UPD-REQ-16 - Browser shall provide clear, user-friendly notifications about available updates and security status without enabling user suppression of critical updates

Given: A conformant browser with UPD-1 or higher capability

Task: Verify that update notification UI balances security enforcement with user experience, ensuring critical security updates cannot be suppressed while avoiding notification fatigue that causes users to ignore all update prompts. This prevents social engineering attacks where users habitually dismiss security notifications and ensures that truly critical updates demanding immediate attention are distinguished from routine maintenance updates.

Verification:

1. Trigger an available update and observe the notification mechanism → Confirm that update UI clearly distinguishes between "check for updates" and current update status
2. Verify that update notifications are visible but non-intrusive (e.g., menu icon, subtle indicator) → Update notifications are displayed prominently but do not block browsing
3. Test that critical security updates generate more prominent notifications than feature updates → Critical security updates have more prominent notifications than feature updates
4. Attempt to dismiss or suppress update notifications and verify critical updates cannot be permanently suppressed → Update type and importance are clearly communicated to users
5. Verify that update notifications include clear information about update type (security vs feature) → Critical security updates cannot be permanently dismissed or ignored
6. Test that users can defer non-critical updates but not critical security updates → Non-critical updates can be deferred by users with clear re-notification
7. Check that update status is always accessible through browser settings or help menu → Update status is always visible in browser settings/help menu
8. Verify that notifications include estimated update size and installation time → Notifications include helpful details (update size, type, installation requirements)
9. Test that update notifications are accessible (screen reader compatible, high contrast support) → Update UI is accessible to users with disabilities

Pass Criteria: Update notifications are clear and accessible AND critical updates cannot be permanently suppressed AND update type and importance are communicated AND users can defer non-critical updates

Fail Criteria: No update notifications OR critical updates can be permanently suppressed OR update type unclear OR notifications block browser usage OR inaccessible UI

Evidence: Screenshots of update notifications for various update types, test results from notification dismissal attempts, accessibility testing results (screen reader, high contrast), user flow documentation, notification timing and frequency logs

References:

- Chrome Update UI: https://support.google.com/chrome/answer/95414
- Firefox Update Preferences: https://support.mozilla.org/en-US/kb/update-firefox-latest-release
- WCAG 2.1 Accessibility Guidelines: https://www.w3.org/WAI/WCAG21/quickref/
- Microsoft UI Design Principles: https://docs.microsoft.com/en-us/windows/apps/design/
- NIST Usability and Security: https://www.nist.gov/itl/applied-cybersecurity/tig/back-basics-multi-factor-authentication

Assessment: UPD-REQ-17 (Forced update for critical vulnerabilities)

Reference: UPD-REQ-17 - Browser shall support forced update mechanisms for critical vulnerabilities that require immediate patching

Given: A conformant browser with UPD-1 or higher capability

Task: Verify that forced update mechanisms can override user preferences when actively exploited critical vulnerabilities require immediate patching, preventing scenarios where user inaction leaves browsers vulnerable to widespread attacks. This emergency response capability is to be protected against abuse through cryptographic verification while ensuring that users running vulnerable versions are protected even if they attempt to defer updates.

Verification:

1. Review browser documentation for forced update or killswitch mechanisms → Confirm that forced update events are logged with justification and admin override options (if applicable)
2. Simulate a critical vulnerability scenario requiring immediate update → Browser supports forced update mechanism for critical security issues
3. Verify that forced updates can override user deferral preferences → Forced updates override user preferences and deferral settings
4. Test that forced updates trigger mandatory browser restart with clear security messaging → Clear security messaging explains necessity of forced updates to users
5. Check that forced update status is indicated in update manifests with verifiable flags → Forced update flags in manifests are cryptographically verified
6. Attempt to bypass or cancel a forced update and verify it cannot be avoided → Grace periods provide countdown timers before mandatory restart
7. Verify that forced updates include grace periods with countdown timers before enforcement → Forced updates cannot be bypassed or permanently cancelled
8. Test that forced updates can disable the browser if update fails or is unavailable → Browser may be disabled if critical update cannot be applied
9. Check that forced update mechanism includes safeguards against abuse (signed manifests, trusted channels) → Forced update mechanism is protected against spoofing and abuse

Pass Criteria: Forced update mechanism exists for critical vulnerabilities AND overrides user deferrals AND provides clear security messaging AND forced update flags are cryptographically verified AND includes grace periods

Fail Criteria: No forced update mechanism OR can be bypassed by users OR no verification of forced update flags OR no security messaging explaining necessity

Evidence: Forced update configuration documentation, test results from simulated critical updates, user notification screenshots, forced update manifest flags, grace period timer demonstrations, bypass attempt

results, audit logs for forced updates

References:

- Chrome Component Killswitch: https://www.chromium.org/administrators/policy-list-3#ComponentUpdatesEnabled
- Firefox Blocklist System: https://wiki.mozilla.org/Blocklisting
- Microsoft Forced Update Policies: https://docs.microsoft.com/en-us/deployedge/microsoft-edge-update-policies#updatedefault
- CVE Critical Severity Guidelines: https://www.first.org/cvss/
- NIST Critical Patch Management: https://csrc.nist.gov/publications/detail/sp/800-40/rev-4/final

Assessment: UPD-REQ-18 (Update verification chain)

Reference: UPD-REQ-18 - Browser shall implement complete chain-of-trust verification from update manifest through component signatures to final installation

Given: A conformant browser with UPD-1 or higher capability

Task: Verify that complete chain-of-trust verification prevents sophisticated supply chain attacks where attackers compromise individual components of the update distribution system. By requiring cryptographic validation at every level from root keys through manifests to individual component signatures, the browser ensures that compromise of any single element in the distribution chain cannot result in installation of malicious updates.

Verification:

1. Trace the complete update verification chain from initial update check to installation → Confirm that any break in verification chain prevents installation and triggers errors
2. Verify that update manifest is signed by trusted root key embedded in browser → Complete chain-of-trust from root keys to installed components
3. Examine the signature chain: root key → manifest → component hashes → component signatures → Root keys for update verification are embedded in browser binary
4. Test that each level of the verification chain is validated before proceeding → Update manifests are signed and verified against trusted root keys
5. Attempt to break the chain by presenting valid components with invalid manifest → Component signatures are verified against manifest hashes
6. Verify that intermediate certificate revocation breaks the verification chain → Each verification step is performed in sequence with no shortcuts
7. Test that the root keys used for update verification are embedded in browser binary and protected → Any break in verification chain prevents installation
8. Check that chain-of-trust validation occurs even for component updates → Intermediate certificate revocation is detected and enforced
9. Verify that the entire verification chain is logged for audit purposes → Verification chain is logged with details at each step

Pass Criteria: Complete verification chain from root keys to components AND each level is validated before proceeding AND breaks in chain prevent installation AND root keys are embedded and protected

Fail Criteria: Incomplete verification chain OR steps can be skipped OR root keys not embedded OR chain breaks do not prevent installation

Evidence: Verification chain documentation and diagrams, root key extraction from browser binary, signature verification logs at each chain level, test results from chain break attempts, intermediate certificate revocation tests, audit logs showing complete chain verification

References:

- Chrome Root Certificate Program: https://www.chromium.org/Home/chromium-security/root-ca-policy/

- Code Signing Certificate Chains: https://docs.microsoft.com/en-us/windows-hardware/drivers/install/digital-signatures
- X.509 Certificate Path Validation: https://www.rfc-editor.org/rfc/rfc5280#section-6
- Mozilla Root Store Policy: https://www.mozilla.org/en-US/about/governance/policies/security-group/certs/policy/
- TUF Root of Trust: https://theupdateframework.github.io/specification/latest/#root-metadata
- NIST Trust Anchor Management: https://csrc.nist.gov/publications/detail/sp/800-57-part-1/rev-5/final

Assessment: UPD-REQ-19 (Update source pinning)

Reference: UPD-REQ-19 - Browser shall implement update source pinning to prevent malicious redirection to unauthorized update servers

Given: A conformant browser with UPD-1 or higher capability

Task: Verify that update source pinning prevents malicious redirection attacks that could deliver compromised updates through DNS hijacking, BGP routing attacks, or compromised enterprise proxies. Hardcoded, pinned update server domains ensure that even if network infrastructure is compromised, browsers will only accept updates from authentic vendor-controlled servers validated through certificate pinning.

Verification:

1. Identify the configured update server URLs embedded in browser binary or configuration → Confirm that update source pinning violations are logged and reported
2. Verify that update server domains are pinned and cannot be modified through configuration files → Update server domains are hardcoded in browser binary
3. Attempt to modify system DNS or hosts file to redirect update domains to malicious servers → Update source URLs cannot be modified through configuration or registry
4. Test that certificate pinning for update domains prevents MITM attacks → DNS redirection attacks are prevented through certificate pinning
5. Verify that update server URLs use multiple trusted domains with fallback mechanisms → Multiple trusted update domains with verified fallback mechanisms
6. Attempt to inject malicious update server URLs through registry/preference modifications → Enterprise policies cannot override update source pinning for security updates
7. Test that update source pinning is enforced even for enterprise-managed deployments → Network-based redirection attempts are detected and blocked
8. Check that hardcoded update domains cannot be overridden by network-based attacks → Fallback servers are subject to same pinning requirements
9. Verify that fallback update servers are also pinned and verified → Source pinning violations are logged as security events

Pass Criteria: Update server domains are hardcoded AND cannot be modified by configuration OR DNS attacks AND certificate pinning prevents redirection AND fallback servers are also pinned

Fail Criteria: Update servers can be modified through configuration OR DNS redirection succeeds OR no certificate pinning OR enterprise policies can override source pinning

Evidence: Decompiled browser binary showing hardcoded update URLs, test results from DNS redirection attempts, certificate pinning verification, configuration modification tests, enterprise policy override tests, source pinning violation logs

References:

- Chrome Update Server Infrastructure: https://chromium.googlesource.com/chromium/src/+/main/docs/updater/protoc server
- Certificate Pinning Implementation: https://owasp.org/www-community/controls/Certificate_and_Public_Key_Pinnin
- DNS Rebinding Protection: https://en.wikipedia.org/wiki/DNS_rebinding
- Mozilla Update Server Security: https://wiki.mozilla.org/Software_Update#Security
- Google Omaha Server Protocol: https://github.com/google/omaha/blob/main/doc/ServerProtocolV3.md

- NIST Supply Chain Risk Management: https://csrc.nist.gov/Projects/cyber-supply-chain-risk-management

Assessment: UPD-REQ-20 (Update integrity verification)

Reference: UPD-REQ-20 - Browser shall verify integrity of installed components after update application to detect corruption or tampering

Given: A conformant browser with UPD-1 or higher capability

Task: Verify that post-installation integrity verification detects tampering or corruption that occurs after updates are applied, protecting against attacks where malware modifies browser components between installation and execution. This closes the security window where installed files are vulnerable to modification before integrity checks run, ensuring that any tampering triggers immediate remediation.

Verification:

1. Perform a complete browser update and monitor the post-installation verification phase → Confirm that integrity verification uses same cryptographic strength as update verification (SHA-256+)
2. Verify that installed files are checked against expected cryptographic hashes after installation → Post-installation integrity verification checks all installed components
3. Attempt to modify an installed file immediately after update completion → Cryptographic hashes (SHA-256+) are used for integrity verification
4. Test that modified files are detected on next browser launch → Modified files are detected immediately after installation
5. Verify that integrity verification occurs both after installation and at browser startup → Startup integrity checks detect tampering between sessions
6. Check that integrity verification covers all critical components (binaries, libraries, resources) → Critical components are verified (binaries, libraries, key resources)
7. Test that integrity failures trigger repair or re-installation mechanisms → Integrity failures trigger automatic repair or update mechanisms
8. Verify that integrity verification results are logged with specific failure details → Browser refuses to launch if critical components fail verification
9. Test that browser refuses to launch if critical component integrity verification fails → Integrity verification results are logged with specific error details

Pass Criteria: Post-installation integrity verification is performed AND uses strong cryptographic hashes (SHA-256+) AND modified files are detected AND integrity failures trigger repair OR prevent launch

Fail Criteria: No post-installation verification OR weak hash algorithms OR modified files not detected OR integrity failures do not trigger corrective actions

Evidence: Post-installation verification logs, hash algorithm analysis, test results from file modification attacks, startup integrity check logs, repair mechanism demonstrations, critical component verification coverage analysis

References:

- Chrome Component Integrity Verification: https://www.chromium.org/Home/chromium-security/education/tls#TOC-Certificate-Verification
- Windows Code Integrity: https://docs.microsoft.com/en-us/windows/security/threat-protection/windows-defender-application-control/windows-defender-application-control
- macOS Gatekeeper and Code Signing: https://developer.apple.com/documentation/security/notarizing_macos_software
- Linux Integrity Measurement Architecture: https://sourceforge.net/p/linux-ima/wiki/Home/
- NIST File Integrity Monitoring: https://csrc.nist.gov/glossary/term/file_integrity_monitoring
- OWASP Software Component Verification: https://owasp.org/www-project-software-component-verification-standard/

Assessment: UPD-REQ-21 (Staged rollout support)

Reference: UPD-REQ-21 - Browser shall support staged rollout mechanisms to gradually deploy updates to user populations with rollback capability

Given: A conformant browser with UPD-2 or higher capability

Task: Verify that staged rollout mechanisms limit the blast radius of defective updates while maintaining rapid response capability for critical security fixes. Gradual deployment with rollback capability prevents scenarios where buggy updates simultaneously impact millions of users, while emergency bypass ensures that actively exploited vulnerabilities can still be patched immediately across the entire user base.

Verification:

1. Review browser update architecture for staged rollout or canary deployment support → Confirm that staged rollout status is logged for monitoring and analysis
2. Verify that update manifests include rollout percentage or cohort targeting information → Update system supports staged rollout with progressive deployment
3. Test that updates are delivered to progressively larger user populations over time → Rollout percentages are specified in update manifests and enforced server-side
4. Check that rollout stages are controlled by server-side configuration, not client manipulation → Users cannot manipulate rollout cohort assignment
5. Verify that users cannot force-join or opt-out of rollout cohorts → Rollout can be paused or reversed if issues are detected
6. Test that rollout can be paused or rolled back if issues are detected → Critical security updates can bypass staged rollout
7. Monitor multiple browser instances to observe different rollout timing → Cohort assignment is stable (same user gets consistent rollout timing)
8. Verify that critical security updates can bypass staged rollout for immediate deployment → Multiple instances show varied rollout timing based on cohort
9. Test that rollout cohorts are assigned using stable user identifiers (not random per check) → Rollout status and cohort information are logged

Pass Criteria: Staged rollout mechanism is implemented AND rollout percentages are server-controlled AND users cannot manipulate cohorts AND critical updates can bypass rollout AND rollback capability exists

Fail Criteria: No staged rollout support OR users can manipulate rollout OR all users receive updates simultaneously OR no rollback capability OR critical updates follow slow rollout

Evidence: Update manifest files showing rollout percentages, server-side rollout configuration documentation, test results from multiple instances showing varied timing, rollout logs with cohort information, critical update bypass demonstrations, rollback process documentation

References:

- Chrome Staged Rollout: https://chromium.googlesource.com/chromium/src/+/main/docs/updater/functional_spec.md#
  rollout
- Google Omaha Rollout Mechanisms: https://github.com/google/omaha/blob/main/doc/ServerProtocolV3.md#rollout
- Canary Deployments: https://martinfowler.com/bliki/CanaryRelease.html
- Progressive Delivery Patterns: https://www.split.io/glossary/progressive-delivery/
- NIST Phased Deployment: https://csrc.nist.gov/publications/detail/sp/800-40/rev-4/final

Assessment: UPD-REQ-22 (Update domain validation)

Reference: UPD-REQ-22 - Browser shall validate that update requests and responses originate from authorized domains with strict certificate requirements

Given: A conformant browser with UPD-1 or higher capability

Task: Verify that update domain validation prevents server-side request forgery and domain confusion attacks where attackers redirect update requests to malicious servers through HTTP redirects, DNS manipulation, or compromised network infrastructure. Strict domain allowlisting ensures that browsers only download updates from explicitly authorized vendor domains, rejecting any deviation regardless of certificate validity.

Verification:

1. Capture update network traffic to identify all domains involved in update process → Confirm that domain validation failures are logged with domain details
2. Verify that all update domains are whitelisted or hardcoded in browser configuration → All update domains are explicitly whitelisted or hardcoded
3. Examine TLS certificates for update domains to verify they meet strict requirements (EV or equivalent) → TLS certificates for update domains meet strict requirements
4. Attempt to redirect update requests to unauthorized domains using DNS attacks → Unauthorized domains are rejected even with valid TLS certificates
5. Test that update domain acceptlist cannot be modified by users or malware → Domain acceptlist cannot be modified through configuration or registry
6. Verify that update responses from unauthorized domains are rejected → Subdomain validation is strict (no wildcard matching)
7. Check that update domain validation includes subdomain restrictions (no wildcards) → HTTP redirects to unauthorized domains are blocked
8. Test that HTTP redirects to unauthorized domains during update process are rejected → Domain validation applies to all update-related traffic
9. Verify that update domain validation occurs for all update-related requests (manifests, downloads, telemetry) → Validation failures are logged with attempted domain information

Pass Criteria: Update domains are whitelisted/hardcoded AND unauthorized domains are rejected AND domain list cannot be modified AND validation applies to all update traffic AND redirects to unauthorized domains are blocked

Fail Criteria: No domain acceptlist OR unauthorized domains accepted OR domain list can be modified OR validation can be bypassed with redirects OR logging insufficient

Evidence: Network traffic captures showing update domains, domain acceptlist extraction from browser binary, test results from unauthorized domain redirects, TLS certificate analysis, domain validation logs, configuration modification attempt results

References:

- Chrome Update Domain Security: https://chromium.googlesource.com/chromium/src/+/main/docs/updater/protocol_3
- Mozilla Update Domain Policies: https://wiki.mozilla.org/Software_Update#Security
- DNS Security Extensions (DNSSEC): https://www.icann.org/resources/pages/dnssec-what-is-it-why-important-2019-03-05-en
- Extended Validation Certificates: https://en.wikipedia.org/wiki/Extended_Validation_Certificate
- OWASP Server-Side Request Forgery Prevention: https://cheatsheetseries.owasp.org/cheatsheets/Server_Side_Request_
- NIST Domain Validation Guidelines: https://csrc.nist.gov/publications/detail/sp/800-63/3/final

Assessment: UPD-REQ-23 (Update binary reproducibility)

Reference: UPD-REQ-23 - Browser shall support mechanisms to enable verification of binary reproducibility for update transparency and supply chain security

Given: A conformant browser with UPD-2 or higher capability

Task: Verify that binary reproducibility enables independent verification of update authenticity, protecting against sophisticated supply chain attacks where build infrastructure is compromised to inject backdoors. Reproducible builds allow security researchers and organizations to verify that distributed binaries match published source code, detecting unauthorized modifications introduced during compilation or packaging.

Verification:

1. Review browser build and release documentation for reproducible build support → Confirm that reproducibility documentation and verification tools are publicly available
2. Verify that update packages include build metadata (compiler version, build timestamp, source commit) → Browser supports reproducible builds with documented build process
3. Attempt to reproduce an official update binary from published source code using documented build process → Update packages include complete build metadata
4. Compare reproduced binary hash with official release hash to verify reproducibility → Independent verification of binary reproducibility is possible
5. Check that build process documentation includes all dependencies and toolchain versions → Build process documentation includes all dependencies and toolchain versions
6. Verify that update transparency logs include binary hashes for independent verification → Multiple builds from same source produce bit-identical binaries
7. Test that multiple independent parties can reproduce identical binaries from same source → Update transparency logs enable third-party verification
8. Check for availability of build attestations or signed build manifests → Non-deterministic build elements are minimized or documented
9. Verify that non-reproducible elements (timestamps, randomness) are minimized or eliminated → Build attestations or signed manifests are available

Pass Criteria: Reproducible build process is documented AND build metadata is included in updates AND independent parties can verify binary reproducibility AND update transparency supports third-party verification

Fail Criteria: No reproducible build support OR build process not documented OR independent verification not possible OR build metadata missing OR excessive non-deterministic elements

Evidence: Build process documentation, reproduced binary hash comparisons, build metadata extraction from update packages, transparency log entries, independent build verification results, build attestation signatures, reproducibility verification tool outputs

References:

- Reproducible Builds Project: https://reproducible-builds.org/
- Chromium Build Documentation: https://chromium.googlesource.com/chromium/src/+/main/docs/linux/build_instruct
- Debian Reproducible Builds: https://wiki.debian.org/ReproducibleBuilds
- NIST Secure Software Development Framework: https://csrc.nist.gov/Projects/ssdf
- SLSA Supply Chain Security Framework: https://slsa.dev/

Assessment: UPD-REQ-24 (Automatic update application without user intervention)

Reference: UPD-0-REQ-23 - Updates shall be applied without user intervention or postponement

Given: A conformant browser with UPD-0 capability (forced automatic updates)

Task: Verify that updates are applied automatically without any user intervention or ability to postpone, ensuring that security patches are deployed immediately to minimize the window of vulnerability exploitation, protecting users who might otherwise delay critical updates, while accepting the trade-off of reduced user control in exchange for maximum security posture.

Verification:

1. Configure browser in UPD-0 mode with forced automatic updates → Test that updates install in background without interrupting user workflows
2. Trigger an update availability (use test update server if available) → Verify that only browser restart (if needed) may be delayed to safe moment
3. Verify that update is downloaded automatically without user prompt → Test that critical updates force installation even if browser is in use

4. Verify that update is applied immediately without user confirmation → Verify that users cannot disable automatic update mechanism
5. Test that no UI options exist to postpone or decline update → Test that settings UI shows no postponement options
6. Verify that update application proceeds even during active browsing sessions → Verify that update process cannot be interrupted or canceled by user

Pass Criteria: Updates download automatically AND install without user confirmation AND no postponement options available AND settings prevent disabling automatic updates

Fail Criteria: Updates require user confirmation OR postponement is possible OR automatic updates can be disabled OR update process can be interrupted

Evidence: Update installation logs showing automatic application, UI screenshots showing no postponement options, settings verification showing forced updates, test scenarios demonstrating immediate installation

References:

- NIST Secure Software Development: https://csrc.nist.gov/projects/ssdf
- Automatic Security Updates Best Practices: https://www.us-cert.gov/ncas/tips/ST04-006

Assessment: UPD-REQ-25 (Automatic browser restart after critical updates)

Reference: UPD-0-REQ-24 - Browser shall restart automatically after critical updates when safe

Given: A conformant browser with UPD-0 capability after critical update installation

Task: Verify that the browser automatically restarts after critical security updates are installed, but only when safe to do so (when user is not in middle of sensitive tasks), ensuring updates become effective immediately while minimizing disruption, balancing security urgency against user experience by detecting safe restart windows and providing brief warnings.

Verification:

1. Install critical security update that requires browser restart → Test that brief warning is displayed before automatic restart (e.g., 60 seconds)
2. Verify that browser detects restart requirement → Verify that warning cannot be dismissed or canceled for critical updates
3. Test browser behavior in various states: → Test that browser session state is preserved across restart
   - Idle state (no active tabs or forms)
   - Active browsing (tabs open but no form input)
   - Form input in progress (unsaved data)
   - File downloads in progress
   - Media playback active
4. Verify that browser waits for safe moment to restart in each scenario → Verify that tabs, windows, and browsing history are restored
5. Test that unsaved form data prevents immediate restart → Test that restart occurs automatically if browser remains idle after update
6. Verify that active downloads are protected from restart → Verify that maximum delay before forced restart is reasonable (e.g., 8 hours)

Pass Criteria: Browser restarts automatically after critical updates AND waits for safe moments AND preserves user session AND provides brief warning AND forced restart occurs if delayed too long

Fail Criteria: No automatic restart OR unsafe restart timing OR session not preserved OR no warning provided OR indefinite delay possible

Evidence: Restart behavior logs, timing analysis across different browser states, session preservation verification, warning dialog screenshots, maximum delay testing

References:

- User Experience in Security Updates: https://www.usenix.org/conference/soups2016/technical-sessions/presentation/mathur
- Browser Session Management: https://developer.mozilla.org/en-US/docs/Web/API/Window/sessionStorage

Assessment: UPD-REQ-26 (Update postponement with time limits)

Reference: UPD-1-REQ-23 - Users shall be able to postpone non-critical updates for limited time (max 7 days)

Given: A conformant browser with UPD-1 capability (automatic with postponement)

Task: Verify that users can postpone non-critical updates for a limited time (maximum 7 days), providing flexibility for users to control update timing while ensuring that security patches are not delayed indefinitely, balancing user autonomy with security needs by enforcing reasonable time limits after which updates shall be applied.

Verification:

1. Trigger availability of non-critical update → Test that postponed update becomes mandatory when deadline approaches
2. Verify that update notification provides postponement option → Verify that warning increases in urgency as deadline nears
3. Test postponing update and verify it is not installed immediately → Test that update is automatically installed when maximum postponement expires
4. Verify that postponement options offer reasonable durations (1 day, 3 days, 7 days) → Verify that users cannot extend postponement beyond maximum limit
5. Test selecting maximum postponement period (7 days) → Test that multiple postponements don't extend total delay beyond limit
6. Verify that browser tracks postponement deadline → Verify that update notification persists until user takes action

Pass Criteria: Non-critical updates can be postponed AND maximum postponement is 7 days AND forced installation after deadline AND multiple postponements don't exceed limit AND critical updates have stricter limits

Fail Criteria: Indefinite postponement possible OR maximum exceeds 7 days OR no forced installation after deadline OR deadline can be extended indefinitely

Evidence: Postponement UI screenshots, deadline tracking verification, forced installation logs after expiry, multiple postponement tests, critical vs non-critical update handling comparison

References:

- Update Management Best Practices: https://www.cisecurity.org/insights/white-papers/software-update-and-patch-management-best-practices
- User Control in Security Updates: https://www.usenix.org/conference/soups2016/technical-sessions/presentation/mathur

Assessment: UPD-REQ-27 (Critical update postponement restrictions)

Reference: UPD-1-REQ-24 - Critical security updates shall not be postponable beyond 24 hours

Given: A conformant browser with UPD-1 capability with critical security update available

Task: Verify that critical security updates addressing actively exploited vulnerabilities or high-severity issues cannot be postponed beyond 24 hours, ensuring that urgent patches are deployed rapidly while still providing minimal user control, recognizing that critical vulnerabilities represent immediate threats requiring prompt mitigation despite user preference for delayed updates.

Verification:

1. Trigger availability of critical security update (e.g., zero-day patch) → Test that warning messages escalate as postponement deadline approaches
2. Verify that update is clearly marked as "Critical" or "Security Update" → Verify that browser displays prominent warning if critical update is postponed
3. Test that postponement options are limited for critical updates → Test that update is automatically installed when 24-hour deadline expires
4. Verify that maximum postponement for critical update is 24 hours or less → Verify that users cannot extend critical update postponement
5. Test that postponement UI emphasizes urgency and security risk → Test that "remind me later" options are limited (1 hour, 4 hours, 24 hours max)
6. Verify that security rationale is provided explaining criticality → Verify that background browser processes trigger update even if UI postponed

Pass Criteria: Critical updates identified separately AND maximum postponement is 24 hours AND forced installation after deadline AND prominent warnings displayed AND no extension possible

Fail Criteria: Critical updates can be postponed beyond 24 hours OR no distinction from non-critical OR insufficient warnings OR deadline can be extended OR no forced installation

Evidence: Critical update UI showing restricted postponement, urgency warnings, 24-hour deadline enforcement logs, automatic installation verification, security rationale display

References:

- CISA Known Exploited Vulnerabilities: https://www.cisa.gov/known-exploited-vulnerabilities-catalog
- Rapid Security Update Deployment: https://www.nist.gov/publications/guide-enterprise-patch-management-technologies

Assessment: UPD-REQ-28 (Update severity indication)

Reference: UPD-1-REQ-25 - Browser shall notify users of pending updates with severity indication

Given: A conformant browser with UPD-1 capability with updates available

Task: Verify that update notifications clearly indicate update severity (critical, high, medium, low) to help users understand urgency and make informed decisions about update timing, providing transparency about security impact and enabling users to prioritize critical patches while potentially deferring less urgent updates, supporting risk-aware update management.

Verification:

1. Trigger updates of various severity levels:
   - Critical security fix for actively exploited vulnerability
   - High severity security patch
   - Medium priority bug fixes and security improvements
   - Low priority feature updates
2. Verify that update notifications display severity level prominently
3. Test that severity is indicated through multiple cues:
   - Text labels ("Critical", "High", "Medium", "Low")
   - Color coding (red for critical, yellow for medium, etc.)
   - Icon indicators (warning symbols, shields)
   - Urgency language in notification text
4. Verify that severity explanation is provided when requested
5. Test that critical severity triggers most prominent notification
6. Verify that postponement options vary based on severity
7. Test that update details explain security impact:
   - CVE identifiers for security issues
   - Brief description of vulnerability addressed
   - Potential attack scenarios prevented

8. Verify that severity assessment is based on objective criteria
9. Test that multiple updates are ordered by severity in UI
10. Verify that severity indication is consistent across all notification channels

Pass Criteria: Severity clearly indicated through multiple UI cues AND explanation provided AND varies postponement options AND includes security impact details AND consistent across notifications

Fail Criteria: No severity indication OR unclear severity communication OR severity doesn't affect options OR no security impact details OR inconsistent indication

Evidence: Notification screenshots for each severity level, severity explanation text, postponement option variations, security impact descriptions, CVE references in updates

References:

- CVSS (Common Vulnerability Scoring System): https://www.first.org/cvss/
- Security Update Communication Best Practices: https://www.cisa.gov/coordinated-vulnerability-disclosure-process

Assessment: UPD-REQ-29 (Automatic updates disabled by default)

Reference: UPD-2-REQ-14 - Automatic updates shall be disabled by default

Given: A conformant browser with UPD-2 capability (optional automatic updates)

Task: Verify that automatic updates are disabled by default in UPD-2 mode, requiring explicit user opt-in before automated update installation occurs, respecting user autonomy and system control in environments where automatic changes may conflict with testing requirements, system stability needs, or bandwidth constraints, while clearly communicating the security risks of manual update management.

Verification:

1. Perform fresh browser installation in UPD-2 mode → Verify that settings clearly show automatic updates are disabled
2. Verify that automatic updates are disabled by default on first run → Test that manual update check is available
3. Check browser settings to confirm automatic update state → Verify that enabling automatic updates requires explicit user opt-in:
   - Navigate to settings
   - Locate automatic update toggle
   - Click to enable with confirmation
4. Trigger update availability with automatic updates disabled → Test that opt-in decision is persistent across browser restarts
5. Verify that update is not downloaded or installed automatically → Verify that first-run wizard or setup offers automatic update option with clear explanation
6. Test that update notification is displayed but requires user action → Test that security implications of disabled automatic updates are explained

Pass Criteria: Automatic updates disabled by default AND manual action required for updates AND explicit opt-in to enable automatic updates AND security implications explained AND opt-in persists

Fail Criteria: Automatic updates enabled by default OR automatic installation occurs without opt-in OR no clear opt-in mechanism OR no security explanation

Evidence: Fresh installation default state verification, settings screenshots showing disabled state, manual update workflow, opt-in process documentation, security warning examples

References:

- User Autonomy in Software Updates: https://www.w3.org/TR/design-principles/#user-control

- Update Policy Best Practices: https://www.cisecurity.org/insights/blog/patch-management-best-practices

Assessment: UPD-REQ-30 (Automatic update recommendations)

Reference: UPD-2-REQ-15 - Browser shall prominently recommend enabling automatic updates

Given: A conformant browser with UPD-2 capability with automatic updates disabled

Task: Verify that the browser prominently recommends enabling automatic security updates, educating users about security benefits while respecting their choice to manage updates manually, using clear messaging that explains risks of manual update management without employing dark patterns or coercive tactics, supporting informed decision-making about update policies.

Verification:

1. Access browser with automatic updates disabled
2. Verify that recommendation to enable automatic updates is displayed:
   - During initial browser setup/first run
   - In browser settings near update controls
   - In security or privacy dashboard
   - Periodically as gentle reminder (not nagging)
3. Test that recommendation clearly explains benefits:
   - Automatic security patches protect against threats
   - Reduced window of vulnerability exploitation
   - No need to remember to check for updates
   - Background installation minimizes disruption
4. Verify that recommendation is prominent but not intrusive:
   - Visible and easy to find
   - Not blocking workflow or annoying
   - Can be acknowledged/dismissed
   - Does not reappear constantly (dark pattern)
5. Test that enabling automatic updates from recommendation is easy (single click)
6. Verify that recommendation respects user's choice to decline:
   - Clear "No thanks" or "Keep manual" option
   - Choice is remembered
   - Not repeatedly prompted in same session
7. Test that recommendation is balanced and informative, not fear-mongering
8. Verify that recommendation links to detailed information if user wants to learn more

Pass Criteria: Recommendation is prominent and visible AND clearly explains benefits AND easy to enable from recommendation AND respects user's choice to decline AND not intrusive or nagging

Fail Criteria: No recommendation shown OR unclear benefits OR difficult to enable OR forced enablement OR dark patterns used OR constantly nagging user

Evidence: Recommendation display locations and frequency, message content analysis, enablement workflow, dismissal behavior, user choice respect verification

References:

- Ethical Design Principles: https://www.w3.org/TR/design-principles/
- Dark Pattern Prevention: https://www.deceptive.design/
- Security Awareness Communication: https://www.nist.gov/itl/applied-cybersecurity/nice/resources/online-learning-content

Assessment: UPD-REQ-31 (Security warnings for available updates)

Reference: UPD-2-REQ-16 - Browser shall display security warnings when updates are available

Given: A conformant browser with UPD-2 capability with security updates available

Task: Verify that the browser displays clear security warnings when security updates are available but not yet installed, informing users about increased risk from running outdated versions with known vulnerabilities, providing actionable information to encourage timely manual installation without resorting to scare tactics, supporting security awareness for users managing updates manually.

Verification:

1. Trigger availability of security update (patch for known vulnerability)
2. Verify that security warning is displayed prominently:
   - Banner or notification in browser UI
   - Icon or badge indicating security risk
   - Entry in security dashboard or settings
3. Test that warning clearly communicates security risk:
   - Updates address security vulnerabilities
   - Outdated version may be vulnerable to attacks
   - Recommendation to install updates promptly
   - Severity indicator if critical update
4. Verify that warning provides direct link to update installation
5. Test that warning persists until update is installed
6. Verify that warning visibility increases with time since update available:
   - Subtle notification initially
   - More prominent after several days
   - Prominent warning after one week
7. Test that warning is informative but not alarmist:
   - Factual language about security risks
   - No fear-mongering or exaggeration
   - Clear call to action
8. Verify that warning includes timestamp (update available since X days)
9. Test that multiple security updates aggregate into single clear warning
10. Verify that warning disappears immediately after update installation

Pass Criteria: Security warnings displayed prominently AND clearly communicate risk AND provide direct update link AND escalate with time AND informative without fear-mongering AND disappear after installation

Fail Criteria: No security warnings OR unclear risk communication OR no update link OR static visibility OR alarmist language OR warnings persist after update

Evidence: Security warning screenshots, warning visibility escalation timeline, language tone analysis, update link functionality, persistence behavior verification

References:

- Security Warning Design: https://www.usenix.org/conference/soups2016/technical-sessions/presentation/felt
- Effective Security Communication: https://www.usenix.org/conference/soups2013/proceedings/presentation/akhawe

Assessment: UPD-REQ-32 (Easy update check and install mechanism)

Reference: UPD-2-REQ-17 - Browser shall provide easy mechanism to check for and install updates

Given: A conformant browser with UPD-2 capability (optional automatic updates)

Task: Verify that users can easily check for available updates and install them manually through intuitive UI controls, ensuring that manual update management is practical and convenient for users who prefer explicit control, providing clear workflow from update check to installation without requiring technical knowledge or complex procedures.

Verification:

1. Access browser settings or help menu → Test that update installation is reliable and completes successfully
2. Verify that "Check for Updates" option is easily findable: → Verify that any restart requirement is clearly communicated
   - Located in expected place (About, Settings, Help menu)
   - Clear, understandable label
   - Accessible within 2-3 clicks from main UI
3. Test manually triggering update check → Test that update check and install work from multiple entry points
4. Verify that update check provides clear feedback: → Verify that update changelog or release notes are accessible
   - Checking animation or progress indicator
   - Clear result message (up-to-date or updates available)
   - Details about available updates
5. Test that update installation is straightforward: → Test that errors during update are handled gracefully with clear messages
   - Single "Install Update" or "Update Now" button
   - Clear indication of what will happen
   - Progress indicator during download and installation
6. Verify that update process doesn't require advanced technical knowledge → Verify that users can cancel update download if needed (before installation)

Pass Criteria: Update check easily accessible AND clear feedback provided AND simple installation process AND reliable completion AND restart requirements communicated AND multiple access points AND release notes available

Fail Criteria: Update check hard to find OR no feedback OR complex installation OR frequent failures OR unclear requirements OR only one access point OR no release information

Evidence: UI navigation path to update check, update check workflow screenshots, installation process recordings, error handling examples, accessibility from multiple locations

References:

- Usability in Security Features: https://www.usenix.org/conference/soups2006/proceedings
- User-Centered Design for Updates: https://www.w3.org/TR/design-principles/#user-control

Assessment: UPD-REQ-33 (Full user control over update timing)

Reference: UPD-2-REQ-18 - Users shall have full control over update timing and installation

Given: A conformant browser with UPD-2 capability with updates available

Task: Verify that users have complete control over when and whether to install updates, enabling users to manage update timing according to their needs, workflows, and system requirements, respecting user autonomy fully while acknowledging the security trade-offs, ensuring that users who choose manual update management have necessary tools and information to do so effectively.

Verification:

1. Trigger availability of various updates (critical and non-critical) → Test that users can selectively install some updates while skipping others
2. Verify that users can choose whether to install each update → Verify that skipping updates doesn't prevent checking for newer updates
3. Test that users can defer updates indefinitely: → Test that users can postpone updates without time limits
   - No forced installation deadlines
   - No automatic installation after time period
   - User always has option to decline

4. Verify that update notifications can be dismissed → Verify that even critical security updates can be declined by user
5. Test that dismissed updates don't automatically reinstall → Test that users understand consequences of their choices through clear warnings
6. Verify that users can choose specific installation timing: → Verify that update preferences are respected across browser sessions
    - Install now
    - Install on next restart
    - Install later (no deadline)
    - Don't install (skip update)

Pass Criteria: Users can defer updates indefinitely AND can decline any update AND can choose specific installation timing AND selective installation possible AND preferences persist AND consequences clearly communicated

Fail Criteria: Forced installation occurs OR time limits enforced OR cannot decline updates OR no selective installation OR preferences not persistent OR inadequate consequence communication

Evidence: Update control UI showing all options, indefinite deferral testing, selective installation demonstrations, critical update decline verification, preference persistence tests, warning message examples

References:

- User Autonomy in Software Design: https://www.w3.org/TR/design-principles/#user-control
- Update Management for Advanced Users: https://www.cisecurity.org/insights/white-papers/software-update-and-patch-management-best-practices

Assessment: UPD-REQ-34 (Manual update check mechanism)

Reference: UPD-3-REQ-11 - Browser shall provide manual update check mechanism

Given: A conformant browser with UPD-3 capability (manual updates only)

Task: Verify that the browser provides a reliable manual update check mechanism allowing users to check for available updates on demand, ensuring that users who have disabled all automatic update behaviors can still maintain security through manual update management, providing necessary tools for deliberate update workflows in controlled environments.

Verification:

1. Verify that automatic update checks are fully disabled in UPD-3 mode → Verify that update check can be performed multiple times without restriction
2. Access manual update check mechanism in browser UI → Test that check does not automatically download or install anything
3. Verify that "Check for Updates" is easily accessible: → Verify that check completes in reasonable time (< 10 seconds typically)
    - Available in About page, Settings, or Help menu
    - Clear labeling and prominent placement
    - Functional even with all automatic features disabled
4. Test triggering manual update check → Test that check works across different network conditions
5. Verify that check contacts update servers and retrieves current information: → Verify that check failure doesn't prevent retrying
    - Network request to update endpoint
    - Retrieval of latest version metadata
    - Comparison with current browser version
6. Test that check provides clear results: → Test that update metadata includes security information and changelog
    - "You are up to date" if current
    - "Update available" with version details if outdated

- Error message if check fails (network issue, server unavailable)

Pass Criteria: Manual check mechanism easily accessible AND reliably contacts update servers AND provides clear results AND doesn't auto-download/install AND works in various network conditions AND includes security information

Fail Criteria: No manual check OR difficult to access OR unreliable OR automatically downloads/installs OR frequent failures OR no security information

Evidence: Update check UI location and access path, network traffic during check, result message examples for various scenarios, reliability testing across networks, metadata content analysis

References:

- Manual Update Management: https://www.cisecurity.org/controls/v8/
- User-Initiated Security Updates: https://www.nist.gov/publications/guide-enterprise-patch-management-technologies

Assessment: UPD-REQ-35 (Outdated version security warnings)

Reference: UPD-3-REQ-12 - Browser shall display security warnings for outdated versions

Given: A conformant browser with UPD-3 capability running outdated version

Task: Verify that the browser displays persistent security warnings when running significantly outdated versions with known vulnerabilities, informing users about security risks of delayed updates and encouraging timely manual installation, balancing respect for user autonomy with responsibility to communicate security status, especially when manual update management results in vulnerable configurations.

Verification:

1. Run browser with outdated version (multiple updates behind current) → Test that warning can be acknowledged but returns periodically
2. Verify that security warning is displayed indicating outdated status: → Verify that warning does not block browser functionality (not modal dialog)
   - Visible banner, icon, or notification
   - Clear message about being out of date
   - Indication of how far behind current version
3. Test that warning communicates security implications: → Test that warning includes last update check timestamp
   - Known vulnerabilities in current version
   - Security fixes available in newer versions
   - Recommendation to update
4. Verify that warning escalates with age of browser version: → Verify that critically outdated versions show prominent, persistent warnings
   - Minor warning if one version behind
   - Moderate warning if several versions behind
   - Strong warning if critically outdated (>6 months, contains CVEs)
5. Test that warning provides direct path to update mechanism → Test that warning differentiates between minor feature updates and security patches
6. Verify that warning persists across browser sessions → Verify that warning disappears after update installation

Pass Criteria: Warnings displayed for outdated versions AND clearly communicate security risk AND escalate with version age AND provide update path AND persistent but not blocking AND distinguish security vs feature updates

Fail Criteria: No warnings for outdated versions OR unclear risk communication OR static visibility regardless of age OR no update path OR blocking workflow OR doesn't distinguish update types

Evidence: Warning screenshots at various version ages, warning escalation timeline, security messaging content, update path accessibility, persistence behavior across sessions

References:

- Security Communication Best Practices: https://www.usenix.org/conference/soups2016/technical-sessions/presentation/felt
- Vulnerability Disclosure: https://www.cisa.gov/coordinated-vulnerability-disclosure-process

Assessment: UPD-REQ-36 (Clear indication of available security updates)

Reference: UPD-3-REQ-13 - Browser shall provide clear indication of available security updates

Given: A conformant browser with UPD-3 capability with security updates available

Task: Verify that the browser clearly distinguishes security updates from feature updates, providing prominent indication when security patches are available, enabling users to prioritize security-critical updates over general improvements, supporting informed manual update decisions by highlighting updates that address vulnerabilities even when user prefers manual installation control.

Verification:

1. Trigger availability of mixed updates:
   - Security patch fixing vulnerabilities
   - Feature update adding new capabilities
   - Bug fix update improving stability
2. Verify that security updates are clearly marked:
   - "Security Update" label or badge
   - Security icon (shield, lock) indicator
   - Color coding (red or yellow for security)
   - Separate section in update list
3. Test that security update details explain vulnerability addressed:
   - CVE identifier if applicable
   - Brief vulnerability description
   - Severity rating
   - Potential attack scenarios prevented
4. Verify that security updates are visually prioritized in UI:
   - Listed first in update list
   - Larger or bold text
   - More prominent notification
5. Test that non-security updates are clearly differentiated:
   - "Feature Update" or "Bug Fix" labels
   - Different visual treatment
   - Lower priority in list
6. Verify that mixed updates (security + features) are accurately labeled
7. Test that security update indicators are consistent across all UI locations
8. Verify that users can filter or view only security updates
9. Test that security update urgency is communicated appropriately
10. Verify that users can choose to install only security updates, skipping features

Pass Criteria: Security updates clearly distinguished from other types AND vulnerability details provided AND visual prioritization AND consistent indicators AND filtering possible AND selective installation supported

Fail Criteria: No distinction between update types OR missing vulnerability details OR no visual differentiation OR inconsistent indicators OR no filtering OR cannot selectively install security updates

Evidence: Update list screenshots showing mixed update types, security indicators across various UI locations, CVE information display, filtering UI, selective installation workflow

References:

- CVE Program: https://cve.mitre.org/
- Security Update Best Practices: https://www.cisa.gov/sites/default/files/publications/CISA-Cyber-Essentials-Toolkit_Mar2022_508.pdf

Assessment: UPD-REQ-37 (Explicit user initiation for all updates)

Reference: UPD-3-REQ-14 - All updates shall require explicit user initiation

Given: A conformant browser with UPD-3 capability with updates available

Task: Verify that absolutely no updates occur without explicit user initiation, ensuring complete user control over browser modifications and system changes, respecting environments requiring change control procedures, stability priorities, or bandwidth management, confirming that manual-only update mode provides true autonomy without any automatic behaviors.

Verification:

1. Configure browser in UPD-3 (manual only) mode → Test that user should explicitly:
   - Manually check for updates
   - Manually trigger download
   - Manually initiate installation
   - Manually restart browser if needed
2. Trigger availability of various updates (critical security and non-critical) → Verify that even critical security updates require user initiation
3. Verify that no automatic update checks occur: → Test that no "soft" automatic behaviors occur (e.g., automatic check with manual install)
   - Monitor network traffic for update check requests
   - Confirm no scheduled background checks
   - Verify no update checks on browser startup
4. Test that updates are not downloaded automatically: → Verify that update configuration cannot be bypassed by browser or OS
   - Available updates not pre-downloaded
   - No background downloads
   - Network traffic shows no update downloads
5. Verify that updates are not installed without user action: → Test that enterprise policies cannot override manual-only setting
   - No automatic installation during idle time
   - No installation on browser restart
   - No forced installation for critical updates

Pass Criteria: No automatic update checks AND no automatic downloads AND no automatic installation AND all actions require explicit user initiation AND applies to all update types AND cannot be bypassed

Fail Criteria: Any automatic update behavior OR checks without user action OR downloads without initiation OR installation without explicit action OR critical updates bypass manual mode

Evidence: Network traffic analysis showing no automatic update activity, manual workflow verification for all update steps, critical update handling in manual mode, policy override testing, mode persistence across updates

References:

- Change Control Procedures: https://www.cisecurity.org/controls/v8/
- User Autonomy Principles: https://www.w3.org/TR/design-principles/#user-control

Assessment: UPD-REQ-38 (Update changelog and security impact display)

Reference: UPD-3-REQ-15 - Browser shall display update changelog and security impact

Given: A conformant browser with UPD-3 capability with updates available

Task: Verify that the browser provides comprehensive changelog and security impact information for updates, enabling informed manual update decisions by clearly documenting what changes each update introduces, what vulnerabilities are addressed, and what security improvements are included, supporting users who need detailed information before authorizing system changes.

Verification:

1. Trigger availability of update with mixed changes (security fixes, features, bug fixes) → Verify that changelog includes version numbers and dates
2. Access update details or changelog in browser UI → Test that changelog is accessible before update installation:
   - Available in update notification
   - Viewable in update check results
   - Accessible from browser help/about page
3. Verify that changelog is comprehensive and well-organized: → Verify that changelog links to detailed documentation if available
   - Grouped by category (security, features, bug fixes, performance)
   - Chronological or logical ordering
   - Clear, understandable language
4. Test that security section is prominent and detailed: → Test that changelog format is readable (not raw git commits)
   - List of CVEs addressed
   - Vulnerability descriptions
   - Severity ratings
   - Attack scenarios prevented
   - Credit to security researchers if applicable
5. Verify that feature changes are clearly documented: → Verify that security impact is emphasized visually in changelog
   - New capabilities added
   - Changed behaviors
   - Deprecated features
6. Test that bug fixes are listed with impact descriptions → Test that changelog is available in multiple languages if browser is localized

Pass Criteria: Comprehensive changelog provided AND security impact prominently detailed AND CVE information included AND accessible before installation AND well-organized and readable AND emphasized security sections

Fail Criteria: No changelog available OR minimal security details OR missing CVE information OR only accessible after installation OR poorly formatted OR security not emphasized

Evidence: Changelog screenshots showing all sections, security detail depth analysis, CVE reference verification, accessibility before installation, formatting quality assessment, localization testing

References:

- Security Disclosure Best Practices: https://www.cisa.gov/coordinated-vulnerability-disclosure-process
- Software Release Notes Standards: https://docs.gitlab.com/ee/development/changelog.html

6.6 Protocol Handler Security Assessments

This section covers assessment procedures for requirements PRO-REQ-1 through PRO-REQ-23, addressing custom protocol handler registration, scheme hijacking prevention, protocol allowlisting, URL scheme security, mobile deep linking, and handler isolation.

Assessment: PRO-REQ-1 (Protocol handler registration validation)

Reference: PRO-REQ-1 - Browser shall validate all custom protocol handler registrations to ensure they meet security requirements before registration

Given: A conformant browser with PRO-1 or higher capability

Task: Verify that protocol handler registration validation prevents malicious websites from hijacking critical URL schemes or injecting XSS vectors through handler templates. Without strict validation, attackers could register handlers for privileged schemes like javascript: or craft handler URLs containing script injection, enabling cross-site scripting attacks when victims click protocol links.

Verification:

1. Navigate to a test page that attempts to register a custom protocol handler using registerProtocolHandler() → Verify that successful registrations require user permission or confirmation
2. Test registration with a valid web+custom scheme (e.g., web+music, web+calendar) → Valid web+custom scheme registrations are accepted after validation
3. Attempt to register a handler for a dangerous built-in scheme (http, https, file, javascript) → Built-in privileged schemes (http, https, file, javascript, data, blob) cannot be registered
4. Verify that the browser rejects registration attempts for privileged schemes → Only HTTPS origins can register protocol handlers
5. Test registration with malformed URL templates (missing %s placeholder, invalid URL format) → URL templates shall include %s placeholder and be valid URLs
6. Verify that only HTTPS origins can register protocol handlers (HTTP origins should be rejected) → Malformed registrations are rejected with appropriate error messages
7. Test that the handler URL template is properly validated for XSS vectors → XSS vectors in handler URLs are sanitized or rejected
8. Attempt to register handlers with data: or blob: URLs and verify rejection → Registration from iframes follows same-origin policy
9. Test registration from an iframe and verify same-origin policy enforcement → User permission is required before handler activation

Pass Criteria: Browser validates protocol registrations AND rejects privileged schemes AND requires HTTPS origins AND validates URL templates AND requires user permission

Fail Criteria: Privileged schemes can be registered OR HTTP origins accepted OR invalid URL templates accepted OR no user permission required OR XSS vectors not sanitized

Evidence: Screenshots of registration attempts for various schemes, console error messages for rejected registrations, permission prompt screenshots, network traces showing HTTPS enforcement, XSS test results, developer console outputs

References:

- WHATWG HTML Standard - registerProtocolHandler: https://html.spec.whatwg.org/multipage/system-state.html#custom-handlers
- MDN registerProtocolHandler API: https://developer.mozilla.org/en-US/docs/Web/API/Navigator/registerProtocolHan
- Custom URL Scheme Security: https://tools.ietf.org/html/rfc8252#section-7.1
- Protocol Handler Security Best Practices: https://www.chromium.org/developers/design-documents/create-amazing-password-forms/
- OWASP URL Validation: https://cheatsheetseries.owasp.org/cheatsheets/Input_Validation_Cheat_Sheet.html
- Web Application Security Working Group: https://www.w3.org/2011/webappsec/

Assessment: PRO-REQ-2 (User consent for custom protocols)

Reference: PRO-REQ-2 - Browser shall obtain explicit user consent before activating custom protocol handlers

Given: A conformant browser with PRO-1 or higher capability

Task: Verify that protocol handler consent requirements prevent malicious websites from silently registering handlers that could launch local applications or exfiltrate data through custom protocols without user knowledge. Explicit consent ensures users understand which websites can intercept specific protocol schemes, protecting against protocol handler hijacking where attackers register handlers to intercept sensitive protocol activations.

Verification:

1. Register a custom protocol handler (web+test) on a test page → Verify that permission prompts cannot be spoofed or triggered without user action
2. Verify that registration triggers a permission prompt to the user → Protocol handler registration shows clear permission prompt
3. Test that the permission prompt clearly identifies the scheme and handling origin → Permission prompt identifies the custom scheme and requesting origin
4. Click a link with the custom protocol (web+test:example) and verify activation prompt → Activation of custom protocol shows confirmation before launching handler
5. Test that users can allow, deny, or remember the choice for the handler → Users can allow, deny, or set persistent preferences
6. Verify that denying handler activation falls back to default behavior or shows error → Permission choices persist across sessions
7. Test that remembered choices persist across browser sessions → Settings provide UI to view and revoke handler permissions
8. Navigate to browser settings and verify users can revoke protocol handler permissions → Each origin requires independent user consent
9. Test that each origin requires separate consent (cross-origin isolation) → Permission prompts are genuine browser UI (not web content)

Pass Criteria: Explicit user consent required for registration AND activation prompts shown before launching AND permissions are manageable in settings AND cross-origin isolation enforced

Fail Criteria: No consent prompts displayed OR handlers activate without user permission OR permissions cannot be revoked OR cross-origin handlers share permissions

Evidence: Screenshots of permission prompts (registration and activation), settings UI showing handler permissions, video recording of consent flow, persistent permission test results, cross-origin permission isolation tests, prompt timing analysis

References:

- WHATWG HTML Standard - User Activation: https://html.spec.whatwg.org/multipage/interaction.html#tracking-user-activation
- Permissions API Specification: https://www.w3.org/TR/permissions/
- User Consent Best Practices: https://www.w3.org/TR/security-privacy-questionnaire/
- GDPR Consent Requirements: https://gdpr.eu/gdpr-consent-requirements/
- Chrome Permission UX Guidelines: https://developer.chrome.com/docs/extensions/mv3/permission_warnings/
- Mozilla Permission Prompts: https://support.mozilla.org/en-US/kb/permissions-manager-give-ability-store-passwords-set-cookies-more

Assessment: PRO-REQ-3 (Protocol allowlist enforcement)

Reference: PRO-REQ-3 - Browser shall enforce protocol allowlists that restrict which custom schemes can be registered

Given: A conformant browser with PRO-1 or higher capability

Task: Verify that protocol allowlist enforcement prevents registration of dangerous schemes that could enable code execution, data exfiltration, or privilege escalation through protocol handlers. Strict allowlisting blocks attackers from registering handlers for privileged schemes like vbscript: or shell: that could execute arbitrary code when activated.

Verification:

1. Review browser documentation for allowed custom protocol scheme patterns → Verify that only approved safelist schemes bypass the web+ prefix requirement
2. Attempt to register a handler for web+validname and verify acceptance → Custom schemes use web+ prefix unless on explicit safelist
3. Test registration with schemes that don't follow web+ prefix convention → Safelisted schemes (mailto, tel, sms) can be registered without web+ prefix
4. Verify that safelisted schemes (mailto, tel, sms) can be registered with special handling → Dangerous schemes (javascript, data, vbscript, shell) are blocklisted
5. Attempt to register handlers for blocklisted dangerous schemes (vbscript, shell, etc.) → Scheme names follow DNS label conventions (alphanumeric, hyphens, no spaces)
6. Test enterprise policy controls for custom protocol allowlists → Scheme matching is case-insensitive
7. Verify that scheme names follow DNS label conventions (alphanumeric, hyphens) → Length limits prevent excessively long scheme names (e.g., 64 character limit)
8. Test that scheme names are case-insensitive during registration and matching → Enterprise policies can extend or restrict allowlists
9. Attempt registration with excessively long scheme names and verify length limits → Invalid scheme patterns are rejected with clear error messages

Pass Criteria: web+ prefix required for custom schemes AND safelist exceptions work correctly AND blocklist prevents dangerous schemes AND scheme validation follows standards

Fail Criteria: web+ prefix not enforced OR dangerous schemes accepted OR invalid scheme patterns allowed OR no length limits

Evidence: Registration test results for various scheme patterns, error messages for rejected schemes, enterprise policy configuration examples, scheme validation test matrix, documentation of allowlist and blocklist

References:

- WHATWG URL Standard - Schemes: https://url.spec.whatwg.org/#schemes
- RFC 3986 URI Generic Syntax: https://www.rfc-editor.org/rfc/rfc3986#section-3.1
- Custom URL Scheme Guidelines: https://www.iana.org/assignments/uri-schemes/uri-schemes.xhtml
- Chromium URL Scheme List: https://source.chromium.org/chromium/chromium/src/+/main:url/url_constants.cc
- Safari Custom Protocol Handlers: https://developer.apple.com/documentation/xcode/defining-a-custom-url-scheme-for-your-app
- Mozilla Protocol Handler Allowlist: https://searchfox.org/mozilla-central/source/dom/base/nsContentUtils.cpp

Assessment: PRO-REQ-4 (Scheme hijacking prevention)

Reference: PRO-REQ-4 - Browser shall prevent scheme hijacking attacks where malicious handlers override legitimate protocol handlers

Given: A conformant browser with PRO-1 or higher capability

Task: Verify that scheme hijacking prevention protects users from malicious websites that attempt to override legitimate protocol handlers to intercept sensitive protocol activations. Without protection, attackers could register handlers for schemes like mailto: or custom banking protocols to capture credentials, redirect users, or exfiltrate data when victims click protocol links.

Verification:

1. Register a legitimate protocol handler for web+test from https://trusted.example.com → Verify that resetting browser settings revokes all custom protocol handlers
2. Attempt to register a competing handler for web+test from https://malicious.example.com → Multiple handlers for same scheme trigger user choice rather than silent override
3. Verify that the browser either prevents the override or prompts user for choice → User can see and select from all registered handlers for a scheme

4. Test that the most recently used or user-preferred handler takes precedence → Built-in handlers maintain priority or require explicit user override
5. Verify that built-in handlers (mailto, tel) cannot be completely overridden without permission → Handler selection UI clearly displays origin and scheme information
6. Test that unregistering a handler doesn't automatically activate an attacker's handler → Private/incognito mode handlers are session-only
7. Attempt to register a handler that impersonates a well-known service → Browser reset revokes all custom handlers
8. Verify that handler selection UI clearly shows the origin of each handler → No automatic activation of handlers after unregistration
9. Test that handlers registered in private/incognito mode don't persist → Handler precedence is deterministic and user-controllable

Pass Criteria: Multiple handlers for same scheme require user selection AND origins clearly displayed AND built-in handlers protected AND private mode isolation enforced

Fail Criteria: Silent override of existing handlers OR origins not displayed OR built-in handlers easily hijacked OR private mode handlers persist

Evidence: Screenshots of handler selection UI, multi-handler registration test results, private mode isolation verification, browser reset test results, handler precedence documentation, user choice recording

References:

- Same-Origin Policy for Handlers: https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy
- URL Scheme Hijacking Prevention: https://portswigger.net/web-security/dom-based/open-redirection
- OWASP URL Redirection Cheat Sheet: https://cheatsheetseries.owasp.org/cheatsheets/Unvalidated_Redirects_and_For
- Browser Handler Precedence: https://html.spec.whatwg.org/multipage/system-state.html#concept-handler-precedence

Assessment: PRO-REQ-5 (Protocol parameter sanitization)

Reference: PRO-REQ-5 - Browser shall sanitize protocol URL parameters before passing them to protocol handlers

Given: A conformant browser with PRO-1 or higher capability

Task: Verify that protocol parameter sanitization prevents injection attacks where malicious protocol URLs containing special characters could exploit vulnerabilities in handler implementations. Without proper encoding, protocol parameters containing script tags, SQL commands, or command injection payloads could execute in the context of the handling website or application, enabling XSS, SQLi, or remote code execution.

Verification:

1. Register a test protocol handler with URL template: https://handler.example.com/handle?url=%s → Verify that Unicode characters are properly normalized and encoded
2. Create a test link with XSS payload: Click → All special characters in protocol URLs are properly URL-encoded
3. Click the link and verify that the script payload is URL-encoded when passed to handler → Script tags and JavaScript code are encoded, not executed
4. Test with SQL injection characters in protocol URL and verify proper encoding → SQL injection characters are escaped/encoded
5. Attempt protocol URL with newline/CRLF characters and verify sanitization → CRLF and newline characters are stripped or encoded
6. Test with null bytes and other control characters in protocol URL → Null bytes and control characters are removed or encoded
7. Verify that the %s placeholder is properly substituted with encoded parameter → Parameter substitution uses safe encoding (encodeURIComponent or equivalent)

8. Test with extremely long protocol parameters and verify truncation or rejection → Extremely long parameters are truncated or rejected
9. Attempt to include additional parameters beyond the defined template → Template structure is enforced (no parameter injection)

Pass Criteria: All special characters properly URL-encoded AND XSS payloads neutralized AND injection attacks prevented AND template structure enforced

Fail Criteria: Special characters not encoded OR XSS possible through protocol parameters OR injection successful OR template can be modified

Evidence: XSS test results showing encoding, SQL injection test results, CRLF injection tests, parameter encoding examples, extremely long parameter handling, Unicode normalization tests, network traces showing encoded parameters

References:

- OWASP XSS Prevention Cheat Sheet: https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_
- URL Encoding RFC 3986: https://www.rfc-editor.org/rfc/rfc3986#section-2.1
- JavaScript URL Encoding: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/encod
- Parameter Injection Attacks: https://owasp.org/www-community/attacks/Command_Injection
- Content Security Policy: https://www.w3.org/TR/CSP3/

Assessment: PRO-REQ-6 (External protocol handler security)

Reference: PRO-REQ-6 - Browser shall implement security controls when launching external (OS-level) protocol handlers

Given: A conformant browser with PRO-1 or higher capability

Task: Verify that external protocol handler security prevents malicious websites from silently launching local applications with crafted parameters that could enable command injection or arbitrary code execution. Confirmation prompts and parameter sanitization ensure users understand what application is being launched and prevent attackers from exploiting vulnerable OS-level handlers through injection attacks.

Verification:

1. Configure an OS-level protocol handler for a custom scheme (e.g., myapp://) → Test that repeated external handler launches don't bypass confirmation prompts
2. Create a web page with a link to the custom protocol: Launch → Confirmation prompt appears before launching any external protocol handler
3. Click the link and verify that a confirmation prompt appears before launching → Prompt identifies the target application and protocol scheme
4. Test that the confirmation clearly identifies the external application to be launched → User shall explicitly approve each launch (or set persistent preference)
5. Verify that parameters passed to external handlers are sanitized → Parameters are sanitized to prevent command injection
6. Test launching external handlers with malicious parameters (command injection attempts) → Automatic launch is prevented (requires user gesture)
7. Attempt to launch external handlers from iframes and verify restrictions → Iframe restrictions prevent silent external handler launches
8. Test that user should click or interact to trigger external handler (no automatic launch) → User preferences for external handlers are persistent and accessible
9. Verify that external handler launches respect user preferences (allow/deny lists) → Allow/deny lists work correctly for external protocols

Pass Criteria: Confirmation required for external handler launches AND application identified in prompt AND parameters sanitized AND user gesture required

Fail Criteria: External handlers launch without confirmation OR application not identified OR command injection possible OR automatic launch allowed

Evidence: Screenshots of external handler confirmation prompts, command injection test results, parameter sanitization verification, user gesture requirement tests, iframe restriction tests, preference persistence verification

References:

- External Protocol Handler Security: https://textslashplain.com/2019/08/28/browser-architecture-web-to-app-communication-overview/
- macOS URL Scheme Handling: https://developer.apple.com/documentation/xcode/defining-a-custom-url-scheme-for-your-app
- Linux Desktop Entry Specification: https://specifications.freedesktop.org/desktop-entry-spec/desktop-entry-spec-latest.html

Assessment: PRO-REQ-7 (Protocol handler UI transparency)

Reference: PRO-REQ-7 - Browser shall provide transparent UI that clearly indicates when protocol handlers are registered or invoked

Given: A conformant browser with PRO-1 or higher capability

Task: Verify that protocol handler UI transparency prevents deceptive handler registrations where users unknowingly grant protocol handling permissions without understanding the security implications. Transparent UI ensures users can identify, review, and revoke protocol handlers, protecting against social engineering attacks where malicious sites silently register handlers to intercept sensitive protocol activations.

Verification:

1. Navigate to a page and register a custom protocol handler → Verify that developer tools show protocol handler events for debugging
2. Verify that a clear notification or permission prompt appears during registration → Registration triggers visible notification or permission request
3. Check that the browser UI shows an indicator when handlers are registered (e.g., icon in address bar) → Browser UI indicates when handlers are registered for current origin
4. Click a custom protocol link and verify that the handler invocation is visible to user → Handler invocation shows clear user feedback (dialog, notification, or status)
5. Test that handler management UI is accessible from browser settings → Settings provide comprehensive handler management interface
6. Verify that the settings UI lists all registered handlers with origins and schemes → All registered handlers listed with scheme, origin, and URL template
7. Test that users can easily identify which handler will be invoked for a scheme → Handler selection and removal are user-friendly
8. Verify that handler removal is straightforward from the settings UI → Failed handler invocations show error messages
9. Test that the browser provides clear feedback when handler invocation fails → Developer console logs handler events

Pass Criteria: Registration and invocation visible to user AND settings provide handler management AND all handlers listed with details AND clear feedback for all operations

Fail Criteria: Silent handler operations OR no settings UI for management OR handlers hidden from user OR no feedback on invocation

Evidence: Screenshots of registration notifications, address bar indicators, settings UI showing handler list, handler invocation UI, error messages, developer console logs, video walkthrough of handler lifecycle

References:

- User Interface Security Principles: https://www.w3.org/TR/security-privacy-questionnaire/#questions
- Chrome Protocol Handler Settings: https://support.google.com/chrome/answer/114662
- Firefox Protocol Handler Management: https://support.mozilla.org/en-US/kb/change-program-used-open-email-links
- WCAG Accessibility Guidelines: https://www.w3.org/WAI/WCAG21/quickref/
- User-Centered Security Design: https://www.usenix.org/conference/soups2019

Assessment: PRO-REQ-8 (Protocol downgrade protection)

Reference: PRO-REQ-8 - Browser shall prevent protocol downgrade attacks where HTTPS handlers are replaced with HTTP handlers

Given: A conformant browser with PRO-1 or higher capability

Task: Verify that protocol downgrade protection prevents attackers from replacing secure HTTPS handlers with insecure HTTP handlers to enable man-in-the-middle attacks on protocol activations. Without downgrade protection, attackers could register HTTP handlers to intercept sensitive data passed through protocol URLs that were previously handled securely over HTTPS.

Verification:

1. Register a protocol handler from an HTTPS origin: https://secure.example.com/handler?url=%s → Verify that downgrade attempts are logged for security monitoring
2. Attempt to register a handler for the same scheme from an HTTP origin: http://insecure.example.com/handler?url=%s → HTTP origins cannot register handlers if HTTPS origin has already registered
3. Verify that the HTTP handler registration is rejected or requires explicit user confirmation → Handler URL templates are HTTPS for security-sensitive schemes
4. Test that handler URL templates are HTTPS if registered from HTTPS origins → Existing HTTPS handlers cannot be downgraded to HTTP
5. Attempt to update an existing HTTPS handler to use an HTTP URL template → HSTS policies prevent HTTP handler registration for HSTS hosts
6. Verify that downgrades are prevented or trigger security warnings → Mixed content blocking applies to handler URLs
7. Test that HSTS policies affect protocol handler registration → Upgrade-insecure-requests policy upgrades handler URLs when applicable
8. Verify that mixed content policies apply to handler URL templates → Downgrade attempts generate security warnings or errors
9. Test that handler invocation follows upgrade-insecure-requests policy → Security events are logged for downgrade attempts

Pass Criteria: HTTP handler registration blocked when HTTPS exists AND URL templates are HTTPS AND HSTS enforced for handlers AND downgrade attempts logged

Fail Criteria: HTTP handlers can override HTTPS handlers OR HTTP URL templates accepted OR HSTS not enforced OR no logging of downgrades

Evidence: Downgrade test results, HSTS enforcement verification, mixed content policy tests, security warning screenshots, event logs showing downgrade attempts, upgrade-insecure-requests verification

References:

- HTTP Strict Transport Security (HSTS) RFC 6797: https://www.rfc-editor.org/rfc/rfc6797
- Upgrade Insecure Requests: https://www.w3.org/TR/upgrade-insecure-requests/
- Mixed Content Specification: https://www.w3.org/TR/mixed-content/
- Chrome HTTPS Best Practices: https://developers.google.com/web/fundamentals/security/encrypt-in-transit/why-https
- Mozilla Web Security Guidelines: https://infosec.mozilla.org/guidelines/web_security

Assessment: PRO-REQ-9 (Protocol handler logging)

Reference: PRO-REQ-9 - Browser shall log protocol handler registration, modification, and invocation events for security auditing

Given: A conformant browser with PRO-1 or higher capability and LOG-1 or higher capability

Task: Verify that protocol handler logging creates comprehensive audit trails of handler lifecycle events, enabling detection of malicious handler registrations, unauthorized invocations, or suspicious patterns that could indicate compromise. Complete logging supports security investigations by providing forensic evidence of when handlers were registered, by whom, and how they were used.

Verification:

1. Enable security event logging in browser configuration → Export protocol handler logs and verify they are in structured format
2. Register a custom protocol handler and verify the event is logged → All handler lifecycle events are logged (registration, invocation, modification, removal)
3. Check that the log entry includes: timestamp, origin, scheme, handler URL template, user decision → Log entries include complete metadata: timestamp, origin, scheme, URL template, user action
4. Invoke a registered protocol handler and verify the invocation is logged → Failed registration attempts are logged with error details
5. Modify an existing handler (if supported) and verify the change is logged → External handler invocations are logged separately from web handlers
6. Unregister a protocol handler and verify the removal is logged → Logs distinguish between user-initiated and script-initiated events
7. Test that failed registration attempts are logged with error reasons → Log format is structured (JSON or similar) for analysis
8. Verify that external protocol handler launches are logged → Logs can be exported for security monitoring
9. Test that logs include sufficient context for security analysis → Log retention follows security event retention policies

Pass Criteria: All handler lifecycle events logged with complete metadata AND failed attempts logged AND logs exportable in structured format

Fail Criteria: Handler events not logged OR logs lack critical metadata OR failed attempts not logged OR logs not exportable

Evidence: Log exports showing handler events, log entry examples with metadata, failed registration logs, external handler invocation logs, log format documentation, retention policy verification

References:

- NIST SP 800-92 Log Management: https://csrc.nist.gov/publications/detail/sp/800-92/final
- W3C Reporting API: https://www.w3.org/TR/reporting-1/
- OWASP Logging Cheat Sheet: https://cheatsheetseries.owasp.org/cheatsheets/Logging_Cheat_Sheet.html
- Audit Trail Requirements: https://www.iso.org/standard/27001

Assessment: PRO-REQ-10 (Web+custom scheme support)

Reference: PRO-REQ-10 - Browser shall support web+custom scheme format for custom protocol handlers as specified by WHATWG

Given: A conformant browser with PRO-1 or higher capability

Task: Verify that web+custom scheme support enforces the WHATWG-specified naming convention that prevents namespace collisions with system-level or IANA-registered schemes. The web+ prefix requirement ensures custom web handlers are clearly distinguished from OS-level protocol handlers, preventing malicious sites from hijacking system protocols while enabling safe custom protocol functionality.

Verification:

1. Register a protocol handler using web+ prefix: registerProtocolHandler('web+music', 'https://handler.example.com/play? → Verify that web+ handlers work correctly across different browser contexts
2. Verify that the registration succeeds for properly formatted web+ schemes → web+ prefix is recognized and properly handled
3. Test that web+ schemes are case-insensitive (web+music equals WEB+MUSIC) → Schemes are case-insensitive during registration and matching
4. Create a link with web+music:track123 and verify it invokes the handler → web+ shall be followed by valid scheme name (alphanumeric, hyphens)
5. Test that schemes without web+ prefix (except safelisted) are rejected → web+ alone without suffix is invalid
6. Verify that web+ is followed by at least one alphanumeric character → Scheme names follow DNS label conventions
7. Test that web+ alone (without suffix) is rejected → Multiple web+ schemes can be registered from same origin
8. Verify that web+ schemes follow DNS label rules (no spaces, special chars limited) → Links with web+ schemes correctly invoke registered handlers
9. Test registration of multiple different web+ schemes from same origin → Error messages guide developers on correct web+ format

Pass Criteria: web+ prefix required and recognized AND case-insensitive matching AND DNS label rules enforced AND multiple schemes supported per origin

Fail Criteria: web+ prefix not recognized OR case-sensitive matching OR invalid scheme names accepted OR only one scheme per origin

Evidence: Registration test results for various web+ formats, case sensitivity tests, scheme validation test matrix, multi-scheme registration examples, error message documentation, cross-context invocation tests

References:

- WHATWG HTML Standard - web+ Schemes: https://html.spec.whatwg.org/multipage/system-state.html#normalize-protocol-handler-parameters
- RFC 3986 URI Scheme Syntax: https://www.rfc-editor.org/rfc/rfc3986#section-3.1
- Custom Protocol Handler Specification: https://html.spec.whatwg.org/multipage/system-state.html#custom-handlers
- MDN web+ Scheme Documentation: https://developer.mozilla.org/en-US/docs/Web/API/Navigator/registerProtocolHa
- DNS Label Syntax RFC 1035: https://www.rfc-editor.org/rfc/rfc1035
- Browser Protocol Handler Implementation: https://www.chromium.org/developers/design-documents/create-amazing-password-forms/

Assessment: PRO-REQ-11 (Protocol handler persistence)

Reference: PRO-REQ-11 - Browser shall persist protocol handler registrations across sessions while respecting privacy modes

Given: A conformant browser with PRO-1 or higher capability

Task: Verify that protocol handler persistence balances usability with privacy by maintaining registrations across normal browser sessions while ensuring private mode handlers do not leak across sessions. This prevents privacy violations where private browsing handlers could reveal user activity history while maintaining functionality for legitimate persistent handler registrations.

Verification:

1. Register a protocol handler in normal browsing mode → Verify that exported browser profiles include protocol handler settings
2. Close the browser and reopen it → Handler registrations persist across normal browser restarts
3. Verify that the protocol handler registration persists and is still functional → Registered handlers remain functional after session closure

4. Click a custom protocol link and verify the handler still works after restart → Private/incognito mode handlers are session-only (do not persist)
5. Register a protocol handler in private/incognito mode → Private mode handlers do not leak to normal mode or vice versa
6. Verify that the handler works during the private session → Clearing browsing data removes handler registrations
7. Close the private/incognito window and open a new one → Handler persistence respects user privacy preferences
8. Verify that the handler registered in private mode does NOT persist → Profile export/import includes handler configurations
9. Test that clearing browsing data removes protocol handler registrations → Handler storage is properly synchronized in multi-device scenarios

Pass Criteria: Normal mode handlers persist across sessions AND private mode handlers are session-only AND data clearing removes handlers AND profile export includes handlers

Fail Criteria: Normal handlers don't persist OR private handlers persist OR data clearing doesn't remove handlers OR handlers not in profile export

Evidence: Persistence test results across restarts, private mode isolation verification, data clearing test results, profile export/import examples, multi-device sync verification (if applicable)

References:

- Browser Storage Persistence: https://storage.spec.whatwg.org/
- Private Browsing Mode Specification: https://www.w3.org/TR/tracking-dnt/#private-browsing
- Clear Browsing Data Specification: https://www.w3.org/TR/clear-site-data/
- Firefox Private Browsing: https://support.mozilla.org/en-US/kb/private-browsing-use-firefox-without-history
- Chrome Incognito Mode: https://support.google.com/chrome/answer/95464

Assessment: PRO-REQ-12 (Protocol confusion mitigation)

Reference: PRO-REQ-12 - Browser shall mitigate protocol confusion attacks where similar-looking schemes are used to deceive users

Given: A conformant browser with PRO-1 or higher capability

Task: Verify that protocol confusion mitigation prevents homograph attacks where visually similar scheme names using Unicode characters deceive users into trusting malicious handlers. Attackers could register handlers for schemes that look identical to legitimate ones using Cyrillic or other non-Latin characters, tricking users into believing they're using trusted handlers while data is actually sent to attacker-controlled servers.

Verification:

1. Attempt to register a protocol handler using homoglyphs (e.g., web+test with Cyrillic 'e' instead of Latin 'e') → Test that schemes with different Unicode normalization forms are treated as different
2. Verify that the browser normalizes Unicode in scheme names or rejects homoglyphs → Scheme names are restricted to ASCII alphanumeric characters and hyphens
3. Test registration with mixed scripts in scheme names (Latin + Cyrillic) → Unicode homoglyphs in scheme names are rejected
4. Attempt to register schemes that visually resemble built-in schemes (e.g., web+https with Cyrillic characters) → Mixed script scheme names are not allowed
5. Verify that scheme names are restricted to ASCII alphanumeric and hyphens → Schemes visually similar to built-in schemes are flagged or rejected
6. Test that UI clearly displays scheme names without ambiguity → UI displays scheme names in monospace or disambiguating font

7. Verify that punycode or IDN homographs are not allowed in scheme names → Punycode/IDN encoding not allowed in scheme names
8. Test registration of schemes with confusable characters (0 vs O, 1 vs l) → Confusable character combinations are prevented or warned
9. Verify that handler selection UI disambiguates similar schemes → Different Unicode normalizations are properly handled

Pass Criteria: Scheme names restricted to ASCII AND homoglyphs rejected AND confusable schemes flagged AND UI clearly displays schemes

Fail Criteria: Unicode homoglyphs accepted OR mixed scripts allowed OR confusing schemes not flagged OR ambiguous UI display

Evidence: Homoglyph test results, Unicode normalization tests, confusable character test matrix, UI screenshots showing scheme display, rejection error messages, scheme validation code review

References:

- Unicode Security Considerations: https://www.unicode.org/reports/tr36/
- Homograph Attacks: https://en.wikipedia.org/wiki/IDN_homograph_attack
- ASCII URI Schemes RFC 3986: https://www.rfc-editor.org/rfc/rfc3986#section-3.1
- Chrome IDN Spoof Protection: https://chromium.googlesource.com/chromium/src/+/master/docs/idn.md
- OWASP Unicode Security: https://owasp.org/www-community/attacks/Unicode_Encoding

Assessment: PRO-REQ-13 (Handler capability restrictions)

Reference: PRO-REQ-13 - Browser shall restrict capabilities available to protocol handlers based on context and permissions

Given: A conformant browser with PRO-1 or higher capability

Task: Verify that handler capability restrictions prevent privilege escalation where malicious handlers exploit protocol invocation to gain unauthorized access to APIs or bypass security policies. Handlers shall operate within their origin's security context without inheriting privileges from the protocol scheme itself, preventing attackers from using custom protocols to circumvent same-origin policy or Content Security Policy restrictions.

Verification:

1. Register a protocol handler and invoke it with a custom protocol URL → Verify that handlers respect Permissions Policy (formerly Feature Policy)
2. Verify that the handler page inherits security context from its origin, not protocol URL → Handler security context is based on handler URL origin, not protocol
3. Test that handler cannot access privileged APIs without proper permissions → Privileged APIs require explicit permissions (not granted by handler status)
4. Verify that protocol parameters don't grant additional capabilities → CSP of handler origin is enforced
5. Test that handlers follow Content Security Policy of their origin → Protocol parameters cannot inject capabilities or bypass security
6. Attempt to access local files from handler and verify blocking → Local file access is blocked unless explicitly permitted
7. Test that handlers cannot bypass same-origin policy using protocol parameters → Same-origin policy is enforced for handlers
8. Verify that handlers in iframes have restricted capabilities → Iframe handlers have sandboxed capabilities
9. Test that handler invocation doesn't grant automatic permission escalation → No automatic permission grants upon handler invocation

Pass Criteria: Handler capabilities based on origin AND CSP enforced AND no privilege escalation via protocol AND same-origin policy maintained

Fail Criteria: Protocol grants additional capabilities OR CSP bypassed OR privilege escalation possible OR same-origin policy violated

Evidence: Capability restriction test results, CSP enforcement verification, permission escalation tests (negative), same-origin policy tests, iframe sandbox verification, Permissions Policy tests

References:

- Content Security Policy Level 3: https://www.w3.org/TR/CSP3/
- Permissions Policy: https://www.w3.org/TR/permissions-policy-1/
- Same-Origin Policy: https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy
- HTML Iframe Sandbox: https://html.spec.whatwg.org/multipage/iframe-embed-object.html#attr-iframe-sandbox
- Web API Permissions: https://www.w3.org/TR/permissions/
- Browser Security Model: https://www.chromium.org/Home/chromium-security/

Assessment: PRO-REQ-14 (Protocol handler revocation)

Reference: PRO-REQ-14 - Browser shall provide mechanisms to revoke protocol handler registrations

Given: A conformant browser with PRO-1 or higher capability

Task: Verify that protocol handler revocation mechanisms enable users to remove unwanted or malicious handlers that may have been registered through social engineering or deceptive UIs. Without effective revocation, users would be permanently stuck with handlers they didn't intend to register, allowing attackers to maintain persistent control over protocol scheme handling even after users discover the deception.

Verification:

1. Register multiple protocol handlers from different origins → Test that browser reset removes all custom protocol handlers
2. Navigate to browser settings and locate protocol handler management section → Settings UI provides clear handler management interface
3. Verify that all registered handlers are listed with scheme, origin, and URL template → All registered handlers are visible with complete details
4. Revoke a specific handler using the settings UI → Individual handlers can be selectively revoked
5. Verify that the revoked handler no longer appears in the handler list → Revoked handlers stop functioning immediately
6. Test that clicking custom protocol links no longer invokes the revoked handler → unregisterProtocolHandler() API works correctly
7. Verify that unregisterProtocolHandler() API can programmatically revoke handlers → Cross-origin revocation is prevented (only origin can unregister its handlers)
8. Test that only the registering origin can programmatically unregister its handlers → Site data clearing includes handler revocation
9. Verify that clearing site data revokes all handlers from that origin → Browser reset removes all handlers

Pass Criteria: Settings UI allows handler revocation AND unregisterProtocolHandler() API works AND cross-origin revocation prevented AND site clearing removes handlers

Fail Criteria: No revocation mechanism OR handlers persist after revocation OR cross-origin revocation possible OR site clearing doesn't affect handlers

Evidence: Settings UI screenshots showing handler management, revocation test results, unregisterProtocolHandler() examples, cross-origin revocation prevention tests, site data clearing verification, browser reset verification

References:

- WHATWG unregisterProtocolHandler API: https://html.spec.whatwg.org/multipage/system-state.html#dom-navigator-unregisterprotocolhandler
- Clear Site Data Specification: https://www.w3.org/TR/clear-site-data/
- Browser Settings Best Practices: https://www.w3.org/TR/security-privacy-questionnaire/
- Chrome Protocol Handler Settings: https://support.google.com/chrome/answer/114662

- Firefox Handler Management: https://support.mozilla.org/en-US/kb/change-program-used-open-email-links
- User Control and Transparency: https://www.w3.org/TR/privacy-principles/#user-control

Assessment: PRO-REQ-15 (Cross-origin protocol restrictions)

Reference: PRO-REQ-15 - Browser shall enforce cross-origin restrictions for protocol handler registration and invocation

Given: A conformant browser with PRO-1 or higher capability

Task: Verify that cross-origin protocol restrictions enforce same-origin policy for handler registration and management, preventing malicious iframes from registering handlers on behalf of parent frames or vice versa. This isolation ensures that each origin maintains independent control over its handlers, preventing cross-origin attacks where embedded content could hijack the parent's protocol handling or manipulate handlers from other origins.

Verification:

1. Create a page at https://origin-a.example.com that registers a protocol handler → Verify that subdomains are treated as separate origins for handler registration
2. Create an iframe at https://origin-b.example.com embedded in origin-a → Protocol handlers are attributed to the registering origin (not parent frame)
3. Attempt to register a protocol handler from the iframe → Cross-origin iframes register handlers under their own origin
4. Verify that the handler is attributed to origin-b (iframe origin), not origin-a (parent) → Handlers can only be modified/removed by their registering origin
5. Test that handlers registered by origin-a cannot be unregistered by origin-b → Handler URL templates are same-origin with registration origin
6. Create a cross-origin link to custom protocol and verify handler invocation → Cross-origin invocation works but respects security boundaries
7. Test that handler URL templates match the registering origin → postMessage doesn't bypass origin restrictions
8. Verify that postMessage cannot be used to bypass handler origin restrictions → CORS policies are orthogonal to handler registration
9. Test that CORS policies don't affect protocol handler registration → Subdomain isolation is enforced (sub.example.com != example.com)

Pass Criteria: Handlers attributed to registering origin AND cross-origin modification prevented AND URL templates same-origin AND subdomain isolation enforced

Fail Criteria: Handler origin attribution incorrect OR cross-origin modification allowed OR URL templates can be cross-origin OR subdomain isolation violated

Evidence: Cross-origin registration tests, iframe attribution tests, cross-origin modification prevention tests, URL template origin validation, subdomain isolation tests, public suffix list verification

References:

- Same-Origin Policy: https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy
- HTML Origin Specification: https://html.spec.whatwg.org/multipage/origin.html
- CORS Specification: https://fetch.spec.whatwg.org/#http-cors-protocol
- Public Suffix List: https://publicsuffix.org/
- Iframe Security: https://html.spec.whatwg.org/multipage/iframe-embed-object.html
- Subdomain Security: https://tools.ietf.org/html/rfc6265#section-5.1.3

Assessment: PRO-REQ-16 (Protocol handler manifest validation)

Reference: PRO-REQ-16 - Browser shall validate protocol handler manifests in Progressive Web Apps and installed web applications

Given: A conformant browser with PRO-1 or higher capability and support for Web App Manifests

Task: Progressive Web Apps can declare protocol handlers in their manifest files, creating a persistent attack surface that bypasses runtime API restrictions. Malicious PWAs could register handlers for privileged schemes, escape manifest scope restrictions, or persist handlers after uninstallation to maintain access to user data. Rigorous manifest validation ensures PWA protocol handlers meet the same security standards as API-registered handlers while preventing manifest-based handler abuse.

Verification:

1. Create a PWA with a manifest file declaring protocol handlers in protocol_handlers field → protocol_handlers field in manifest is parsed and validated
2. Install the PWA and verify that declared protocol handlers are registered → Declared handlers are registered upon app installation with user permission
3. Test manifest protocol handlers with invalid URL templates and verify rejection → Invalid handlers in manifest are rejected with clear error messages
4. Verify that manifest handlers require user permission before activation → Same security validations apply as registerProtocolHandler API
5. Test that manifest protocol handlers follow same security rules as registerProtocolHandler → Privileged schemes cannot be registered via manifest
6. Attempt to declare privileged schemes (http, https) in manifest and verify rejection → Handlers are removed when app is uninstalled
7. Test that manifest handlers are unregistered when app is uninstalled → Handlers point to URLs within app scope
8. Verify that manifest scope restrictions apply to protocol handlers → Manifest updates trigger handler re-validation
9. Test manifest handler updates when app manifest is updated → Developer console shows validation errors
10. Verify that manifest validation errors are reported to developer console → OS-level protocol handler registration occurs for installed apps

Pass Criteria: Manifest protocol_handlers validated AND user permission required AND same security rules as API AND uninstalled app removes handlers

Fail Criteria: Invalid manifests accepted OR no user permission OR weaker security than API OR handlers persist after uninstall

Evidence: Manifest examples with protocol handlers, installation flow screenshots, validation error messages, uninstallation verification, scope restriction tests, OS protocol handler registration proof, update handling tests

References:

- Web App Manifest Specification: https://www.w3.org/TR/appmanifest/
- PWA Protocol Handlers: https://web.dev/url-protocol-handler/
- URL Handlers in PWA: https://github.com/WICG/pwa-url-handler/blob/main/explainer.md
- Chrome PWA Installation: https://web.dev/install-criteria/
- MDN Web App Manifest: https://developer.mozilla.org/en-US/docs/Web/Manifest
- App Scope Specification: https://www.w3.org/TR/appmanifest/#scope-member

Assessment: PRO-REQ-17 (Intent URL security - mobile)

Reference: PRO-REQ-17 - Browser shall implement security controls for Android Intent URLs and prevent intent scheme attacks

Given: A conformant mobile browser with PRO-1 or higher capability running on Android

Task: Android Intent URLs allow web pages to launch native applications with arbitrary parameters, creating a powerful attack vector for launching privileged system components, bypassing app permissions, or exfiltrating sensitive data through maliciously crafted intent extras. Intent scheme attacks can trigger dangerous actions like sending SMS, making calls, or accessing sensitive device features without proper authorization. Comprehensive intent URL validation prevents these attacks while maintaining legitimate app integration functionality.

Verification:

1. Create a test page with an intent URL: Click → Intent URLs are parsed and validated before invocation
2. Verify that clicking the intent link triggers appropriate security checks → Privileged system components cannot be launched via intent URLs
3. Test that malicious intent URLs cannot launch privileged system components → Dangerous actions trigger user permission prompts or are blocked
4. Attempt intent URL with dangerous actions (SEND_SMS, CALL, etc.) and verify blocking or permission prompts → Intent extras are sanitized to prevent injection attacks
5. Test intent URLs with arbitrary extras and verify sanitization → Intent URLs cannot bypass app permission model
6. Verify that intent URLs cannot bypass app permissions → Package name validation prevents targeting of unintended apps
7. Test that browser validates package names in intent URLs → Data exfiltration via intent URLs is prevented
8. Attempt to use intent URLs to exfiltrate data and verify prevention → Iframe intent URLs have restricted capabilities
9. Test that intent URLs from iframes have additional restrictions → Intent invocations are logged with target package and action
10. Verify that intent URL invocation is logged for security auditing → User confirmation required for sensitive intent actions

Pass Criteria: Intent URLs validated before launch AND privileged actions blocked AND extras sanitized AND user confirmation for sensitive actions AND invocations logged

Fail Criteria: Intent URLs launch without validation OR privileged actions allowed OR no sanitization OR no confirmation for sensitive actions OR not logged

Evidence: Intent URL test results, privileged action blocking verification, extras sanitization tests, permission prompt screenshots, iframe restriction tests, security log entries, data exfiltration prevention tests

References:

- Android Intent Specification: https://developer.android.com/reference/android/content/Intent
- Intent URL Scheme Security: https://tools.ietf.org/html/rfc8252#appendix-B.2
- Chrome Intent URLs: https://developer.chrome.com/docs/multidevice/android/intents/
- Android App Links: https://developer.android.com/training/app-links
- OWASP Mobile Security: https://owasp.org/www-project-mobile-security-testing-guide/
- Intent Security Best Practices: https://developer.android.com/guide/components/intents-filters#SafeIntent

Assessment: PRO-REQ-18 (Universal Links security - iOS)

Reference: PRO-REQ-18 - Browser shall implement security controls for iOS Universal Links and prevent link hijacking

Given: A conformant mobile browser with PRO-1 or higher capability running on iOS

Task: iOS Universal Links enable seamless web-to-app transitions but create risks of link hijacking, phishing through malicious association files, and privacy violations through link interception tracking. Attackers

can serve fraudulent apple-app-site-association files over compromised connections or use HTTP downgrade attacks to bypass domain validation. Strict validation of association files over HTTPS with certificate verification prevents unauthorized app launches and protects user privacy while maintaining legitimate app integration.

Verification:

1. Set up a test website with an apple-app-site-association file for Universal Links → apple-app-site-association file is validated before Universal Link activation
2. Create a link to a URL that matches a Universal Link pattern → HTTPS required for both the association file and triggering URLs
3. Verify that the browser validates the apple-app-site-association file before allowing app opening → Certificate validation enforced when fetching association file
4. Test that HTTPS is required for Universal Links (HTTP domains should be rejected) → Domain validation prevents unauthorized app openings
5. Verify that the association file is fetched over HTTPS with certificate validation → Malformed or malicious association files are rejected
6. Test that only domains listed in the association file can trigger the app → User choice between app and browser is preserved
7. Attempt to use a malicious association file and verify rejection → User preferences for Universal Links persist
8. Test that users can choose to open in app or browser → No user tracking via Universal Link interception
9. Verify that Universal Link preferences persist across sessions → Association file caching follows secure practices
10. Test that Universal Links respect user privacy (no tracking via link interception) → Link preview shows correct destination before opening

Pass Criteria: Association file validated over HTTPS AND certificate validation enforced AND user choice respected AND preferences persist

Fail Criteria: No association file validation OR HTTP accepted OR certificate errors ignored OR no user choice OR preferences don't persist

Evidence: Association file validation tests, HTTPS enforcement verification, certificate validation tests, malformed file rejection tests, user choice UI screenshots, preference persistence tests, privacy analysis

References:

- Apple Universal Links Documentation: https://developer.apple.com/ios/universal-links/
- Supporting Associated Domains: https://developer.apple.com/documentation/xcode/supporting-associated-domains
- apple-app-site-association Format: https://developer.apple.com/documentation/bundleresources/applinks
- Universal Links Security: https://developer.apple.com/videos/play/wwdc2019/717/
- Deep Linking Security: https://tools.ietf.org/html/rfc8252
- iOS Security Guide: https://support.apple.com/guide/security/welcome/web

Assessment: PRO-REQ-19 (Deep linking validation)

Reference: PRO-REQ-19 - Browser shall validate deep links before navigation to prevent deep link hijacking and phishing

Given: A conformant browser with PRO-1 or higher capability

Task: Deep links enable direct navigation to specific app content but create attack vectors for parameter injection, XSS payload delivery, privilege escalation, and phishing through misleading app names or domains. Unvalidated deep links can bypass app sandboxes, exfiltrate data through malicious parameters, or launch unverified apps without user awareness. Comprehensive deep link validation with parameter sanitization, domain verification, and user confirmation for unverified apps prevents these attacks while maintaining legitimate app deep linking functionality.

Verification:

1. Create a deep link that targets a mobile app: myapp://user/profile?id=123 → Deep link format and structure validated before processing
2. Verify that the browser validates the deep link format before allowing navigation → Parameters in deep links are URL-encoded and sanitized
3. Test that parameters in deep links are sanitized to prevent injection → XSS payloads in deep links are neutralized
4. Attempt to use a deep link with XSS payload and verify sanitization → User confirmation required for unverified deep links
5. Test that deep links to unverified apps trigger user confirmation → Domain validation confirms app ownership of deep link domains
6. Verify that deep link domains are validated against app claims (App Links/Universal Links) → Deep links cannot escalate privileges or bypass sandbox
7. Test that deep links cannot bypass app sandbox or permissions → Phishing detection identifies misleading app names or domains
8. Attempt to use deep links for phishing (misleading app names) and verify warnings → HTTPS fallback works when target app not installed
9. Verify that HTTPS fallback URLs are used when app is not installed → Deep link invocations logged with destination and parameters
10. Test that deep link invocation is logged for security monitoring → Rate limiting prevents deep link abuse

Pass Criteria: Deep link format validated AND parameters sanitized AND user confirmation for unverified links AND domain validation enforced AND invocations logged

Fail Criteria: Invalid deep links accepted OR no parameter sanitization OR no confirmation for unverified links OR no domain validation OR not logged

Evidence: Deep link validation tests, parameter sanitization examples, XSS prevention tests, user confirmation screenshots, domain validation verification, phishing detection tests, fallback URL tests, security logs

References:

- Android App Links Verification: https://developer.android.com/training/app-links/verify-site-associations
- iOS Universal Links Validation: https://developer.apple.com/documentation/xcode/supporting-associated-domains
- Deep Link Security: https://tools.ietf.org/html/rfc8252#section-7.11
- OWASP Mobile Deep Link Security: https://owasp.org/www-project-mobile-security-testing-guide/
- Deep Link Phishing Prevention: https://attack.mitre.org/techniques/T1660/
- URL Scheme Best Practices: https://www.rfc-editor.org/rfc/rfc8252#appendix-B

Assessment: PRO-REQ-20 (Protocol handler CSP integration)

Reference: PRO-REQ-20 - Browser shall integrate protocol handlers with Content Security Policy to prevent handler-based attacks

Given: A conformant browser with PRO-1 or higher capability

Task: Protocol handlers can be weaponized to bypass Content Security Policy restrictions if not properly integrated with CSP directives, allowing attackers to navigate to restricted destinations, execute blocked scripts, or submit forms to prohibited targets through custom protocol invocations. Without CSP integration, handlers create a side channel that circumvents navigate-to, form-action, and script-src protections. Enforcing CSP directives on protocol handler operations ensures handlers cannot be used to violate the page's security policy.

Verification:

1. Create a page with a strict CSP that includes navigate-to directive → navigate-to CSP directive restricts protocol handler destinations
2. Register a protocol handler from this page → CSP violations during handler operations are properly reported
3. Verify that protocol handler invocation respects the navigate-to CSP directive → Handler URL templates that violate CSP are rejected
4. Test that CSP violations during handler registration are reported → script-src controls JavaScript that registers handlers
5. Create a handler URL template that would violate CSP and verify blocking → form-action applies to forms with custom protocol actions
6. Test that the script-src directive affects protocol handler registration scripts → connect-src applies to handler page, not registration
7. Verify that form-action CSP directive applies to forms targeting custom protocols → Handler pages inherit appropriate CSP from their origin
8. Test that connect-src doesn't restrict protocol handler registration but affects handler page → CSP violation reports include handler-specific context
9. Verify that CSP inheritance works correctly for handler pages → Unsafe-inline and unsafe-eval restrictions apply to handlers
10. Test that CSP reports include protocol handler context when violations occur → CSP nonces and hashes work with handler registration

Pass Criteria: navigate-to restricts handler destinations AND violations reported AND handler URLs validated against CSP AND form-action enforced

Fail Criteria: CSP not enforced for handlers OR violations not reported OR handler URLs bypass CSP OR form-action ignored

Evidence: CSP policy examples with handlers, navigate-to enforcement tests, violation reports with handler context, handler URL validation tests, form-action tests, CSP inheritance verification

References:

- Content Security Policy Level 3: https://www.w3.org/TR/CSP3/
- CSP navigate-to Directive: https://www.w3.org/TR/CSP3/#directive-navigate-to
- CSP form-action Directive: https://www.w3.org/TR/CSP3/#directive-form-action
- CSP Violation Reporting: https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP#violation_reports
- CSP Integration: https://w3c.github.io/webappsec-csp/
- Protocol Handler CSP Considerations: https://www.chromium.org/Home/chromium-security/

Assessment: PRO-REQ-21 (Handler registration audit trail)

Reference: PRO-REQ-21 - Browser shall maintain a complete audit trail of protocol handler registration and modification events

Given: A conformant browser with PRO-2 or higher capability (enterprise mode)

Task: Protocol handler registration and modification events create security-critical audit trails essential for detecting handler-based attacks, insider threats, and policy violations in enterprise environments. Without comprehensive auditing, attackers can register malicious handlers, modify existing handlers, or abuse handler permissions without detection, leaving no forensic evidence for incident response. Tamper-evident audit trails with complete lifecycle tracking enable security teams to detect, investigate, and respond to handler abuse while meeting compliance requirements.

Verification:

1. Enable enterprise audit logging for protocol handlers → Complete audit trail for all handler lifecycle events
2. Register a protocol handler and verify the registration is logged → Audit entries include: timestamp, identity, action, object, before/after state, outcome

3. Check that audit log includes: timestamp, origin, scheme, handler URL, user/admin identity → Failed operations logged with error details
4. Modify handler permissions and verify the change is audited → Permission changes audited with user decision
5. Unregister a handler and verify the removal is audited → Enterprise policy enforcement events included
6. Test that failed registration attempts are logged with error reasons → Audit logs are tamper-evident (signed or chained hashes)
7. Verify that permission grants/denials for handlers are audited → Logs exportable in standard formats (JSON, CEF, syslog)
8. Test that enterprise policy changes affecting handlers are logged → SIEM integration supported for centralized logging
9. Export the audit trail and verify it's in a tamper-evident format → Audit trail completeness can be verified
10. Verify that audit logs can be forwarded to enterprise SIEM systems → Log retention aligns with compliance requirements

Pass Criteria: All handler events audited with complete metadata AND failed attempts included AND logs tamper-evident AND SIEM integration supported

Fail Criteria: Incomplete audit trail OR missing metadata OR logs can be tampered OR no SIEM integration

Evidence: Audit log exports, log completeness analysis, tamper-evidence verification, SIEM integration examples, failed operation logs, enterprise policy audit examples, retention policy documentation

References:

- NIST SP 800-53 Audit and Accountability: https://csrc.nist.gov/publications/detail/sp/800-53/rev-5/final
- ISO 27001 Audit Trail Requirements: https://www.iso.org/standard/27001
- Common Event Format (CEF): https://www.microfocus.com/documentation/arcsight/arcsight-smartconnectors-8.3/cef-implementation-standard/
- Syslog Protocol RFC 5424: https://www.rfc-editor.org/rfc/rfc5424
- CIS Audit Log Management: https://www.cisecurity.org/controls/

Assessment: PRO-REQ-22 (Protocol handler update security)

Reference: PRO-REQ-22 - Browser shall securely handle updates to protocol handler registrations and prevent malicious modifications

Given: A conformant browser with PRO-1 or higher capability

Task: Protocol handler updates create opportunities for malicious modification attacks where attackers hijack existing trusted handlers by changing their URL templates to point to attacker-controlled destinations, dangerous URL schemes, or downgraded HTTP endpoints. Cross-origin handler updates could allow one domain to subvert another's handlers, while unvalidated PWA manifest updates could silently redirect protocol traffic. Secure update mechanisms with same-origin enforcement, dangerous scheme blocking, and downgrade protection prevent handler hijacking while maintaining legitimate update functionality.

Verification:

1. Register a protocol handler from https://trusted.example.com → Handler updates require same-origin or user permission
2. Attempt to update the handler URL template from the same origin → Cross-origin handler modification is prevented
3. Verify that the update requires user confirmation or follows original permissions → Active handler updates trigger user notification
4. Attempt to update the handler from a different origin (https://attacker.example.com) → PWA manifest handler updates follow secure update process

5. Verify that cross-origin updates are prevented → Handler URL changes are audited with before/after values
6. Test that handler updates during active use trigger security warnings → Updates to dangerous URL schemes (data:, javascript:) are blocked
7. Verify that automatic handler updates (e.g., PWA manifest updates) are validated → HTTPS-to-HTTP downgrades in handler URLs are prevented
8. Test that handler URL changes are logged in audit trail → Update frequency is rate-limited to prevent abuse
9. Attempt to update handler to point to data: or javascript: URLs and verify blocking → Failed update attempts are logged
10. Verify that downgrade protection applies to handler updates → Users can review and approve pending handler updates

Pass Criteria: Same-origin update restriction enforced AND cross-origin updates prevented AND dangerous schemes blocked AND updates audited

Fail Criteria: Cross-origin updates allowed OR dangerous schemes accepted OR no audit trail OR downgrade attacks possible

Evidence: Update test results showing same-origin enforcement, cross-origin prevention tests, dangerous scheme blocking verification, audit log entries for updates, downgrade prevention tests, PWA manifest update security verification

References:

- WHATWG HTML Handler Updates: https://html.spec.whatwg.org/multipage/system-state.html#custom-handlers
- Web App Manifest Updates: https://www.w3.org/TR/appmanifest/#updating
- Secure Software Updates: https://www.rfc-editor.org/rfc/rfc8240
- OWASP Secure Update Guidelines: https://cheatsheetseries.owasp.org/cheatsheets/Vulnerable_Dependency_Manageme
- Same-Origin Policy for Updates: https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy

Assessment: PRO-REQ-23 (Handler isolation enforcement)

Reference: PRO-REQ-23 - Browser shall enforce process isolation for protocol handlers to limit impact of handler compromise

Given: A conformant browser with PRO-2 or higher capability

Task: Protocol handlers execute in the browser context with potential access to sensitive user data and system resources, making them high-value targets for exploitation. Without process isolation, a compromised handler could access data from other origins, bypass sandbox restrictions, read cross-process memory through Spectre-type attacks, or crash the entire browser. Handler isolation through Site Isolation architecture with sandboxing, IPC validation, and Spectre mitigations contains the impact of handler compromise and prevents privilege escalation attacks.

Verification:

1. Register and invoke a protocol handler → Protocol handlers run in isolated renderer processes
2. Use browser internals (process inspection interface) to verify handler process isolation → Handler isolation follows Site Isolation architecture
3. Verify that handler pages run in separate renderer processes from the triggering page → Cross-origin data is not accessible from handler process
4. Test that compromised handler cannot access other origin's data → Handler processes are sandboxed with restricted capabilities
5. Verify that Site Isolation applies to protocol handler pages → IPC messages from handlers are validated and sanitized

6. Test that handler processes have appropriate sandbox restrictions → Handler crashes are isolated and don't affect browser stability
7. Verify that IPC between handler and browser process is validated → No elevated privileges granted to handler processes
8. Test that handler crash doesn't affect other tabs or browser stability → Memory isolation prevents cross-process memory access
9. Verify that handler processes don't have elevated privileges → Spectre/Meltdown mitigations apply to handler processes
10. Test that memory isolation prevents handler from reading other process memory → Process limit enforced to prevent resource exhaustion

Pass Criteria: Handler process isolation enforced AND Site Isolation applied AND sandboxing active AND IPC validated AND crashes isolated

Fail Criteria: Handlers in same process as other content OR no sandboxing OR IPC not validated OR crashes affect browser OR cross-process memory access possible

Evidence: Process isolation verification using browser internals, Site Isolation tests, sandbox capability analysis, IPC validation tests, crash isolation tests, memory isolation verification, Spectre mitigation confirmation

References:

- Chromium Site Isolation: https://www.chromium.org/Home/chromium-security/site-isolation/
- Browser Process Architecture: https://www.chromium.org/developers/design-documents/multi-process-architecture/
- Sandbox Architecture: https://chromium.googlesource.com/chromium/src/+/master/docs/design/sandbox.md
- IPC Security: https://www.chromium.org/developers/design-documents/inter-process-communication/
- Spectre Mitigations: https://www.chromium.org/Home/chromium-security/ssca/
- Process Isolation Best Practices: https://www.w3.org/TR/security-privacy-questionnaire/#isolation

Assessment: PRO-REQ-24 (HTTP/HTTPS-only protocol restriction)

Reference: PRO-0-REQ-1 - Browser shall only support HTTP and HTTPS protocols

Given: A conformant browser with PRO-0 capability (HTTP/HTTPS only)

Task: Verify that the browser restricts all network communication to HTTP and HTTPS protocols exclusively, eliminating the attack surface introduced by custom protocol handlers, WebSocket, WebRTC, and other non-standard schemes, ensuring maximum security in environments where custom protocols are not needed and could be exploited to bypass security controls, exfiltrate data, or trigger local application vulnerabilities.

Verification:

1. Attempt to navigate to URLs with non-HTTP/HTTPS schemes: → Test that WebSocket connections are blocked (ws:// and wss://)
   - file:// URLs
   - data:// URLs
   - javascript:// URLs
   - ftp:// URLs
   - ws:// and wss:// URLs
   - Custom schemes (mailto:, tel:, etc.)
2. Verify that all non-HTTP/HTTPS navigation attempts are blocked → Verify that WebRTC connections cannot be established
3. Test that blocked navigation displays clear error message → Test that FTP protocol is not supported
4. Verify that error message does not suggest workarounds → Verify that data: URIs are blocked or heavily restricted
5. Test that links with custom protocols do not activate handlers → Test that no custom protocol registration UI exists

6. Verify that registerProtocolHandler() API is not available → Verify that browser help documentation indicates protocol restrictions

Pass Criteria: Only HTTP and HTTPS protocols functional AND all other protocols blocked AND clear error messages AND no workarounds available AND no protocol registration API

Fail Criteria: Non-HTTP/HTTPS protocols accessible OR unclear error messages OR workarounds exist OR protocol registration possible

Evidence: Navigation test results for various protocol schemes, error message screenshots, API availability verification, WebSocket/WebRTC connection attempts, protocol registration UI absence

References:

- URL Standard: https://url.spec.whatwg.org/
- Attack Surface Reduction: https://www.microsoft.com/en-us/security/blog/topic/attack-surface-reduction/

Assessment: PRO-REQ-25 (Custom protocol handler registration rejection)

Reference: PRO-0-REQ-2 - Browser shall reject all custom protocol handler registration attempts

Given: A conformant browser with PRO-0 capability (HTTP/HTTPS only)

Task: Verify that the browser rejects all attempts to register custom protocol handlers through any mechanism, preventing web applications from registering handlers that could be exploited for phishing, command injection, or local application attacks, ensuring that protocol handling remains limited to built-in HTTP/HTTPS with no extensibility for custom schemes.

Verification:

1. Attempt to register custom protocol handler via registerProtocolHandler() API → Test that browser extensions cannot register protocol handlers
2. Verify that API call is rejected or API is undefined → Verify that enterprise policies cannot enable protocol registration
3. Test registration attempts for various schemes: → Test that command-line flags cannot bypass restriction
   - web+custom schemes
   - mailto: handler override
   - Custom application protocols
4. Verify that all registration attempts fail → Verify that error messages clearly indicate feature is disabled
5. Test that no protocol handler registration UI is available in settings → Test that web applications cannot detect registered handlers
6. Verify that manifest-based handler registration is not supported → Verify that no protocol handler storage or persistence exists

Pass Criteria: All protocol handler registration attempts rejected AND no registration API available AND no UI for registration AND extensions cannot register handlers AND no bypass mechanisms

Fail Criteria: Registration succeeds through any mechanism OR registration API accessible OR UI available OR extensions can register OR bypass possible

Evidence: API call rejection logs, registration attempt test results, settings UI verification, extension capability testing, policy override attempts, error messages

References:

- registerProtocolHandler API: https://developer.mozilla.org/en-US/docs/Web/API/Navigator/registerProtocolHandler
- Security by Reduction: https://csrc.nist.gov/glossary/term/least_functionality

Assessment: PRO-REQ-26 (Non-standard URL scheme blocking)

Reference: PRO-0-REQ-3 - Browser shall block access to non-standard URL schemes (file://, data://, javascript://, etc.)

Given: A conformant browser with PRO-0 capability (HTTP/HTTPS only)

Task: Verify that the browser blocks access to non-standard URL schemes that could be exploited for cross-site scripting (javascript:), local file disclosure (file://), or content injection (data:), restricting URL navigation and resource loading exclusively to HTTP and HTTPS to eliminate common attack vectors that abuse non-standard schemes to bypass same-origin policy or execute malicious code.

Verification:

1. Test blocking of javascript: URLs: → Test that custom app schemes (mailto:, tel:, sms:) are blocked
   - In address bar navigation
   - In links ()
   - In iframe src attributes
   - In form action attributes
2. Verify that data: URLs are blocked: → Verify that blocked schemes return consistent error messages
   - In navigation
   - In iframe src
   - In image src
   - In script src
3. Test that file: URLs are blocked: → Test that scheme blocking applies to all contexts (main frame, iframes, workers)
   - In navigation
   - In resource loading (images, scripts, styles)
   - In iframe src
4. Verify that blob: URLs are blocked or restricted → Verify that CSP cannot override scheme blocking
5. Test that about: URLs are limited to safe pages only → Test that redirects to non-standard schemes are blocked
6. Verify that view-source: scheme is blocked → Verify that meta refresh to non-standard schemes is blocked

Pass Criteria: All non-standard schemes blocked consistently AND blocking applies in all contexts AND clear error messages AND CSP cannot override AND redirects blocked

Fail Criteria: Non-standard schemes accessible in any context OR inconsistent blocking OR CSP overrides OR redirects succeed

Evidence: Scheme blocking test results for various contexts, error message examples, CSP interaction tests, redirect blocking verification, meta refresh handling

References:

- URL Standard Schemes: https://url.spec.whatwg.org/#special-scheme
- XSS via javascript: URLs: https://owasp.org/www-community/attacks/xss/
- data: URI Security: https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics_of_HTTP/Data_URLs#security

Assessment: PRO-REQ-27 (Protocol handler API removal)

Reference: PRO-0-REQ-4 - Browser shall not provide registerProtocolHandler() API or equivalent functionality

Given: A conformant browser with PRO-0 capability (HTTP/HTTPS only)

Task: Verify that the browser completely removes or disables the registerProtocolHandler() API and any equivalent protocol registration mechanisms, ensuring that web applications cannot even detect or attempt to

use custom protocol handler functionality, providing defense in depth by eliminating the API surface rather than merely blocking its effects.

Verification:

1. Test that navigator.registerProtocolHandler is undefined or throws error → Test that Content-Security-Policy cannot enable the API
2. Verify that attempting to call the API produces clear error → Verify that browser feature detection shows API as unavailable
3. Test that navigator.isProtocolHandlerRegistered() is also unavailable → Test that JavaScript cannot detect if API was previously available
4. Verify that navigator.unregisterProtocolHandler() is unavailable → Verify that web platform tests for protocol handlers are skipped
5. Test that Feature Policy cannot re-enable protocol handler API → Test that browser documentation indicates API is not supported
6. Verify that no polyfills or fallback mechanisms exist → Verify that removal is consistent across all browsing contexts (windows, workers, iframes)

Pass Criteria: registerProtocolHandler API completely unavailable AND related APIs also removed AND no detection possible AND consistent across contexts AND documentation indicates unavailability

Fail Criteria: API is accessible OR can be enabled through policies OR detection possible OR inconsistent availability OR undocumented removal

Evidence: API availability tests in various contexts, feature detection results, policy configuration attempts, browser documentation excerpts, error messages when attempting API access

References:

- registerProtocolHandler Specification: https://html.spec.whatwg.org/multipage/system-state.html#custom-handlers
- Feature Policy: https://w3c.github.io/webappsec-feature-policy/

Assessment: PRO-REQ-28 (Non-HTTP/HTTPS navigation rejection)

Reference: PRO-0-REQ-5 - Browser shall reject navigation to any non-HTTP/HTTPS protocols

Given: A conformant browser with PRO-0 capability (HTTP/HTTPS only)

Task: Verify that all navigation attempts to non-HTTP/HTTPS protocols are rejected regardless of how navigation is initiated, ensuring comprehensive protocol restriction that cannot be bypassed through various navigation mechanisms like user clicks, script navigation, form submission, or meta refresh, protecting against protocol-based attacks from any entry point.

Verification:

1. Test navigation rejection through various mechanisms: → Test that navigation history does not include blocked attempts
   - Direct address bar input
   - Link clicks (elements)
   - window.location assignments
   - window.open() calls
   - Form submissions
   - Meta refresh tags
   - HTTP redirects (3xx responses)
   - JavaScript location.href changes
2. Test rejection for various non-HTTP/HTTPS schemes: → Verify that referrer is not sent for blocked navigation
   - Custom protocols (myapp://)
   - File system access (file://)

- Data URIs (data://)
- JavaScript (javascript://)
- WebSocket (ws://, wss://)
- FTP (ftp://)

3. Verify that all navigation attempts are blocked before any handler invocation → Test that beforeunload event does not fire for blocked navigation
4. Test that blocked navigation shows user-friendly error page → Verify that page lifecycle continues normally after blocked navigation
5. Verify that error page does not expose protocol handler information → Test that HTTPS upgrade (HTTP -> HTTPS) still functions

Pass Criteria: All non-HTTP/HTTPS navigation blocked AND blocking occurs before handler invocation AND user-friendly errors AND history not polluted AND works across all navigation mechanisms

Fail Criteria: Navigation succeeds through any mechanism OR handlers invoked before blocking OR confusing errors OR history includes blocked attempts

Evidence: Navigation blocking tests for all mechanisms and schemes, error page screenshots, history inspection, lifecycle event verification, logging samples

References:

- Navigation and Browsing Contexts: https://html.spec.whatwg.org/multipage/browsing-the-web.html
- Secure Navigation: https://w3c.github.io/webappsec-secure-contexts/

Assessment: PRO-REQ-29 (Enterprise protocol handler policy controls)

Reference: PRO-2-REQ-12 - Enterprise administrators shall be able to configure protocol handler allowlists and blocklists

Given: A conformant browser with PRO-2 capability in enterprise environment

Task: Verify that enterprise administrators can configure protocol handler allowlists and blocklists through group policy or MDM to control which custom protocols are permitted in their organization, enabling organizations to approve specific business-critical protocol handlers (like internal application launchers) while blocking potentially dangerous handlers, supporting compliance and security requirements through centralized policy management.

Verification:

1. Access enterprise policy management interface → Test that user cannot override enterprise protocol policies
2. Identify protocol handler allowlist/blocklist policies → Verify that allowlist takes precedence when both are configured
3. Configure policy to allow specific protocol schemes (e.g., web+customapp) → Test that wildcard patterns work in policies (e.g., web+myorg*)
4. Configure policy to block specific schemes (e.g., web+dangerous) → Verify that policy applies to all browser profiles on managed device
5. Deploy policy to managed browser → Test that policy changes propagate immediately or on restart
6. Test that allowed protocols can be registered and used → Verify that policy status is visible in browser management UI
7. Verify that blocked protocols cannot be registered → Test that attempting blocked protocol shows enterprise policy message

Pass Criteria: Allowlist and blocklist policies available AND policies enforced correctly AND user cannot override AND wildcard support AND policy status visible AND enforcement logged

Fail Criteria: No policy controls OR policies not enforced OR user override possible OR no wildcard support OR no visibility OR no logging

Evidence: Policy configuration screenshots, enforcement test results with allowed and blocked protocols, user override attempts, wildcard pattern tests, policy status UI, audit logs

References:

- Enterprise Browser Management: https://chromeenterprise.google/policies/
- Group Policy Configuration: https://docs.microsoft.com/en-us/deployedge/configure-microsoft-edge
- Mobile Device Management: https://developer.apple.com/documentation/devicemanagement

Assessment: PRO-REQ-30 (Custom scheme registration without web+ prefix)

Reference: PRO-3-REQ-7 - Browser shall allow registration of custom schemes without web+ prefix

Given: A conformant browser with PRO-3 capability (unrestricted protocol registration)

Task: Verify that the browser allows registration of custom protocol schemes without requiring the web+ prefix, providing maximum flexibility for legacy applications and custom integrations that use established protocol schemes, while acknowledging increased security risk from schemes that don't follow web+ convention and may conflict with future browser features or OS-level protocol handlers.

Verification:

1. Attempt to register custom protocol without web+ prefix (e.g., "myapp") → Test that schemes conflicting with browser internals are rejected
2. Verify that registration succeeds without requiring web+ prefix → Verify that schemes matching standard protocols (http, ftp) are rejected
3. Test that registered handler functions correctly when invoked → Test that OS-level protocol conflicts are detected and warned about
4. Verify that schemes without web+ prefix are marked with security warning → Verify that documentation explains risks of non-web+ schemes
5. Test registration of common non-web+ schemes: → Test that handler management UI shows prefix status
   - Application launchers (myapp://)
   - Internal tools (tool://)
   - Legacy protocols (custom://)
6. Verify that registration still requires user consent → Verify that non-web+ handlers appear in security audits

Pass Criteria: Custom schemes without web+ can be registered AND security warnings shown AND conflicts detected AND user consent still required AND risks documented

Fail Criteria: Non-web+ registration fails OR no security warnings OR conflicts not detected OR no consent required OR undocumented risks

Evidence: Registration success examples for non-web+ schemes, security warning screenshots, conflict detection tests, user consent dialogs, documentation excerpts, audit trail entries

References:

- Custom Scheme and Content Handlers: https://html.spec.whatwg.org/multipage/system-state.html#custom-handlers
- Protocol Handler Security: https://developer.mozilla.org/en-US/docs/Web/API/Navigator/registerProtocolHandler#sec

Assessment: PRO-REQ-31 (Non-standard protocol handler security warnings)

Reference: PRO-3-REQ-8 - Browser shall display security warnings for non-standard protocol handlers

Given: A conformant browser with PRO-3 capability with custom protocol handlers

Task: Verify that the browser displays clear security warnings when non-standard protocol handlers are registered or invoked, informing users about potential risks of custom handlers including command injection,

local application exploitation, or data exfiltration, enabling informed decisions about accepting handlers in unrestricted environments where security is balanced against compatibility.

Verification:

1. Register a non-standard protocol handler (not following web+ convention)
2. Verify that security warning is displayed during registration:
   - Clear explanation of handler risks
   - Warning about non-standard scheme
   - Potential security implications
   - Recommendation to verify handler source
3. Test that warning requires explicit user acknowledgment
4. Verify that invoking non-standard handler shows warning:
   - Banner or notification when handler activates
   - Indication of which handler is being called
   - Option to cancel handler invocation
5. Test warnings for various risk levels:
   - High risk: local application launchers
   - Medium risk: web+ schemes to unknown origins
   - Low risk: handlers from trusted origins
6. Verify that warnings distinguish between:
   - First-time handler invocation (full warning)
   - Subsequent invocations (brief reminder)
7. Test that warning cannot be permanently dismissed for high-risk handlers
8. Verify that warning includes handler details (origin, scheme, target URL)
9. Test that warning links to security documentation
10. Verify that security dashboard shows active non-standard handlers

Pass Criteria: Security warnings displayed at registration and invocation AND warnings explain risks clearly AND explicit acknowledgment required AND warnings vary by risk level AND handler details provided

Fail Criteria: No warnings shown OR unclear risk explanation OR no acknowledgment required OR static warnings regardless of risk OR missing handler details

Evidence: Warning screenshots at registration and invocation, risk level differentiation examples, acknowledgment requirements, security dashboard views, documentation links

References:

- Security Warning Design: https://www.usenix.org/conference/soups2016/technical-sessions/presentation/felt
- Protocol Handler Risks: https://textslashplain.com/2019/08/28/browser-architecture-web-platform-security/

Assessment: PRO-REQ-32 (Protocol handler review interface)

Reference: PRO-3-REQ-10 - Users shall be able to review all registered protocol handlers in browser settings

Given: A conformant browser with PRO-3 capability with protocol handlers registered

Task: Verify that users can review all registered protocol handlers through accessible browser settings interface, providing transparency about which applications or sites can intercept custom protocols, enabling users to audit handler registrations, understand what each handler does, and identify potentially malicious or unwanted handlers for removal.

Verification:

1. Register multiple protocol handlers for different schemes → Test that list can be filtered or searched
2. Access browser settings to find protocol handler management → Verify that clicking handler shows detailed information:
   - Full handler URL template

- Registering origin
- Number of invocations
- Security warnings if applicable

3. Verify that handler list is easily accessible: → Test that handlers can be removed from this interface
   - Located in Privacy, Security, or Site Settings
   - Clear menu label ("Protocol Handlers" or "Custom Protocols")
   - Accessible within 2-3 clicks
4. Test that handler list shows complete information: → Verify that handler changes (add/remove) update list immediately
   - Protocol scheme (e.g., web+myapp)
   - Handler URL/origin
   - Registration date
   - Last invocation date
   - Handler status (active, disabled)
5. Verify that list includes all registered handlers: → Test that export/backup of handler list is available
   - Web+ prefixed schemes
   - Non-web+ custom schemes
   - Built-in handler overrides (mailto, etc.)

Pass Criteria: Handler list easily accessible AND shows complete information AND includes all handlers AND can be filtered AND detailed view available AND removal possible AND changes immediate

Fail Criteria: List hard to find OR incomplete information OR missing handlers OR no filtering OR no details OR cannot remove OR delayed updates

Evidence: Handler list UI screenshots showing various handlers, detailed view examples, filter/search functionality, removal workflow, export capability, help documentation

References:

- User Control and Transparency: https://www.w3.org/TR/design-principles/#user-control
- Browser Settings Best Practices: https://www.w3.org/TR/security-privacy-questionnaire/#user-interface

Assessment: PRO-REQ-33 (Custom handler vulnerability scanning)

Reference: PRO-3-REQ-11 - Browser shall scan custom handlers for known security vulnerabilities

Given: A conformant browser with PRO-3 capability with custom protocol handlers

Task: Verify that the browser scans custom protocol handlers for known security vulnerabilities and suspicious patterns before registration and periodically during use, detecting handlers that could be exploited for command injection, path traversal, or other attacks, providing automated security analysis to protect users who may not recognize malicious handler patterns in unrestricted registration environments.

Verification:

1. Attempt to register protocol handlers with known vulnerable patterns: → Test periodic rescanning of registered handlers:
   - Command injection patterns (e.g., URL with shell metacharacters)
   - Path traversal attempts (../ sequences)
   - SQL injection patterns
   - Script injection patterns (
     , javascript:)
   - Check if scan runs on browser updates (new vulnerability signatures)
   - Verify notification if previously-safe handler becomes vulnerable
2. Verify that browser detects and warns about vulnerable patterns → Verify that vulnerability database is updated regularly
3. Test that scan occurs before handler registration completes → Test that false positives can be reported

4. Verify that detected vulnerabilities are clearly explained to user → Verify that scan covers various attack vectors:
   - Local command execution
   - File system access
   - Network exfiltration
   - Cross-origin data access
5. Test that high-severity vulnerabilities block registration → Test that scan results are logged for security auditing
6. Verify that medium-severity issues show strong warning but allow registration → Verify that documentation explains vulnerability scanning process

Pass Criteria: Vulnerability scanning active AND detects known patterns AND scan before registration AND periodic rescans AND severity-based actions AND regular signature updates AND scan results logged

Fail Criteria: No scanning OR fails to detect known vulnerabilities OR no periodic rescans OR no severity differentiation OR outdated signatures OR no logging

Evidence: Vulnerability detection test results for various attack patterns, scan timing verification, periodic rescan demonstrations, severity handling examples, signature update verification, scan logs

References:

- OWASP Top 10: https://owasp.org/www-project-top-ten/
- Common Weakness Enumeration: https://cwe.mitre.org/
- Protocol Handler Attack Vectors: https://portswigger.net/web-security/cross-site-scripting/contexts

Assessment: PRO-REQ-34 (Protocol handler security audit logging)

Reference: PRO-3-REQ-12 - All protocol handler security exceptions shall be logged and auditable

Given: A conformant browser with PRO-3 capability with protocol handlers active

Task: Verify that all protocol handler security events are comprehensively logged to enable security monitoring, incident investigation, and compliance auditing in unrestricted environments where protocol handlers pose elevated risk, ensuring that security teams have visibility into handler registration, invocation, security warning dismissals, and vulnerability detections to identify potential compromise or policy violations.

Verification:

1. Perform various protocol handler security events: → Verify that logs can be exported or forwarded to SIEM
   - Handler registration (standard and non-standard)
   - Handler invocation
   - Security warning acknowledgments
   - Vulnerability detections during scanning
   - Failed registration attempts (blocked schemes)
   - Handler removal/revocation
   - Policy violations (if enterprise policies active)
2. Access browser security logs or audit trail → Test that log tampering is prevented or detected
3. Verify that all events are logged with comprehensive details: → Verify that high-severity events trigger immediate log entries
   - Timestamp (with timezone)
   - Event type (registration, invocation, warning, etc.)
   - Protocol scheme involved
   - Handler URL/origin
   - Registering origin
   - User action taken (accepted, declined, dismissed)
   - Security warnings shown
   - Vulnerability scan results

- IP address or network context if relevant

4. Test that logs are structured for automated analysis (JSON, CSV, syslog) → Test that logging does not expose sensitive user data
5. Verify that logs can be filtered by event type, scheme, or time range → Verify that logs are accessible to security administrators
6. Test that logs are retained for appropriate period (configurable) → Test that log volume is reasonable and doesn't impact performance

Pass Criteria: All security events logged AND comprehensive details captured AND structured format AND filterable AND exportable to SIEM AND tamper-protected AND privacy-preserving

Fail Criteria: Events not logged OR insufficient details OR unstructured logs OR no filtering OR cannot export OR logs can be tampered OR exposes sensitive data

Evidence: Log samples showing various event types, log schema documentation, filter/search demonstrations, SIEM export examples, tamper protection verification, privacy analysis

References:

- CWE-778: Insufficient Logging: https://cwe.mitre.org/data/definitions/778.html
- OWASP Logging Cheat Sheet: https://cheatsheetseries.owasp.org/cheatsheets/Logging_Cheat_Sheet.html
- Security Audit Trail Requirements: https://csrc.nist.gov/glossary/term/audit_trail

6.7 System Resource Access Security Assessments

This section covers assessment procedures for requirements SYS-REQ-1 through SYS-REQ-32, addressing sandbox enforcement, Hardware Abstraction Layer (HAL) security, PWA permissions, filesystem access, device API security, and system resource isolation.

Assessment: SYS-REQ-1 (Process sandbox enforcement)

Reference: SYS-REQ-1 - Browser shall enforce process-level sandboxing to isolate web content from the host operating system

Given: A conformant browser with SYS-1 or higher capability

Task: Renderer processes handle untrusted web content and are the primary attack surface for browser exploitation. Without OS-level sandboxing, successful renderer exploits grant attackers full system access to read files, execute processes, exfiltrate data, and install malware. Process-level sandboxing using platform-specific mechanisms (seccomp-bpf, sandbox-exec, Job Objects) with syscall filtering and privilege reduction contains renderer compromises, preventing sandbox escapes and limiting attack impact to the isolated process.

Verification:

1. Launch the browser with sandbox debugging enabled (–enable-sandbox-logging or equivalent) → Verify that sandbox violations are logged and processes are terminated on policy violations
2. Open the browser's internal process viewer through the browser's debugging interface or task manager → All renderer processes execute within OS-level sandbox (seccomp-bpf on Linux, sandbox-exec on macOS, Job Objects on Windows)
3. Navigate to a test web page and verify it runs in a sandboxed renderer process → Sandboxed processes cannot access filesystem outside designated cache directories
4. Attempt to execute system calls from JavaScript that should be blocked (file system access, process creation) → System call filtering is active and blocks dangerous syscalls (execve, fork, ptrace)
5. Use platform-specific tools to inspect process sandbox status (Process Explorer on Windows, ps with security flags on Linux/macOS) → Process privileges are reduced (no root, limited capabilities, restricted tokens)
6. Verify renderer processes run with reduced privileges using tools like icacls, getfacl, or sandbox-exec -p → Memory isolation prevents cross-process memory access

7. Test that sandboxed processes cannot access parent process memory → Network access is mediated through broker process
8. Attempt to escape sandbox through known attack vectors and verify containment → Sandbox escape attempts are blocked and logged
9. Monitor system calls using strace (Linux), dtruss (macOS), or Process Monitor (Windows) to verify syscall filtering → Process termination occurs on sandbox policy violations

Pass Criteria: All renderer processes execute in OS-level sandbox AND dangerous system calls are filtered AND privilege reduction is verified AND sandbox escapes are prevented

Fail Criteria: Any renderer process runs without sandbox OR system calls are not filtered OR privileges are not reduced OR sandbox escape succeeds

Evidence: Process sandbox status screenshots, syscall trace logs showing filtering, privilege analysis outputs (icacls, capabilities), sandbox violation logs, security tool reports (Process Explorer, sandbox-exec output)

References:

- Chromium Sandbox Design: https://chromium.googlesource.com/chromium/src/+/master/docs/design/sandbox.md
- Linux seccomp-bpf: https://www.kernel.org/doc/html/latest/userspace-api/seccomp_filter.html
- macOS Sandbox: https://developer.apple.com/documentation/security/app_sandbox
- Windows Sandbox: https://docs.microsoft.com/en-us/windows/security/threat-protection/windows-sandbox/windows-sandbox-overview

Assessment: SYS-REQ-2 (Renderer process isolation)

Reference: SYS-REQ-2 - Browser shall isolate renderer processes from each other and from browser core processes

Given: A conformant browser with SYS-1 or higher capability

Task: Renderer process isolation is fundamental to browser security architecture, preventing compromised renderers from accessing data belonging to other origins. Without process-per-origin isolation, a successful exploit in one tab could steal credentials, session tokens, and sensitive data from all other open tabs, violating Same-Origin Policy at the process level. Site Isolation with distinct processes, mediated IPC, and no shared memory prevents cross-origin data theft, Spectre attacks, and cascading process crashes.

Verification:

1. Open multiple tabs with different origins in the browser → Test process-per-site-instance isolation for enhanced security
2. Use the browser's process viewer to verify each origin runs in a separate renderer process → Each origin or site instance runs in a dedicated renderer process
3. Open developer tools and use performance profiling to identify process boundaries → Process IDs are distinct and verifiable through OS tools
4. Test Site Isolation by navigating to cross-origin iframes and verifying separate processes → Renderer process crashes are isolated and do not cascade
5. Attempt to access memory or data from one renderer process in another using side-channel attacks → No shared memory regions exist between different renderer processes
6. Verify that process IDs are distinct for different origins using OS tools (ps, Task Manager) → Inter-process communication uses secure, mediated IPC channels
7. Test that renderer crashes in one tab do not affect other tabs or the browser process → Browser core process (broker) is isolated from all renderers
8. Monitor inter-process communication to verify it goes through secure IPC channels → GPU process isolation is separate from renderer isolation
9. Use memory analysis tools to verify no shared memory regions between renderers → Side-channel attacks cannot leak data between renderer processes

Pass Criteria: Different origins run in separate processes AND processes have distinct PIDs AND crashes are isolated AND no memory sharing exists

Fail Criteria: Same process handles multiple origins OR process crash cascades OR shared memory exists OR IPC is not secured

Evidence: Process viewer screenshots showing multiple renderer processes, PID listings from OS tools, crash isolation test results, memory map analysis, IPC traffic logs, Site Isolation verification reports

References:

- Chromium Site Isolation: https://www.chromium.org/Home/chromium-security/site-isolation/
- Firefox Fission: https://wiki.mozilla.org/Project_Fission

Assessment: SYS-REQ-3 (GPU process isolation)

Reference: SYS-REQ-3 - Browser shall isolate GPU rendering operations in a separate sandboxed process

Given: A conformant browser with SYS-1 or higher capability

Task: GPU processes execute untrusted shader code and interact with complex graphics drivers that have historically been sources of vulnerabilities. Without GPU process isolation, exploits targeting graphics drivers or shader compilers could escape to access the filesystem, network, or other process memory, bypassing renderer sandbox protections. Isolated GPU processes with command buffer validation and sandboxing contain GPU-related exploits while enabling graceful degradation through software rendering fallbacks.

Verification:

1. Launch browser and navigate to the browser's GPU information interface to verify GPU process information → Verify that software rendering fallback maintains process isolation
2. Open a WebGL-intensive page (e.g., https://webglsamples.org/) and verify GPU process activation → GPU process runs as separate, distinct process with unique PID
3. Use OS process viewer to identify the GPU process and verify it's distinct from renderers → GPU process executes within OS-level sandbox with reduced privileges
4. Check GPU process sandbox status using platform-specific security tools → GPU command buffers are validated before submission to driver
5. Verify GPU process has limited capabilities and cannot access filesystem directly → GPU process cannot directly access filesystem or network
6. Test that GPU process crashes do not terminate the browser or renderer processes → Crashes in GPU process trigger graceful degradation (software rendering)
7. Monitor GPU command buffer submissions to verify they're sanitized and validated → Graphics driver access is mediated and monitored
8. Attempt to exploit GPU driver vulnerabilities and verify sandbox containment → Shader compilation occurs in isolated context
9. Use graphics debugging tools (apitrace, RenderDoc) to analyze GPU process isolation → GPU memory is isolated from CPU-accessible memory

Pass Criteria: GPU process is isolated with distinct PID AND sandbox is enforced AND command validation occurs AND crashes are contained

Fail Criteria: No GPU process isolation OR sandbox not enforced OR commands not validated OR crashes cascade

Evidence: GPU process information screenshots, PID verification, sandbox status reports, crash test results, GPU command trace logs, shader compilation logs, graphics debugging tool outputs

References:

- Chromium GPU Process Architecture: https://www.chromium.org/developers/design-documents/gpu-accelerated-compositing-in-chrome/

- GPU Sandbox: https://chromium.googlesource.com/chromium/src/+/master/docs/design/sandbox.md#gpu-process
- WebGL Security: https://www.khronos.org/registry/webgl/specs/latest/1.0/#security
- Angle Project Security: https://chromium.googlesource.com/angle/angle
- GPU Denylist and Security: https://chromium.googlesource.com/chromium/src/+/master/gpu/config/software_renderin

Assessment: SYS-REQ-4 (Network service isolation)

Reference: SYS-REQ-4 - Browser shall isolate network operations in a separate sandboxed process or service

Given: A conformant browser with SYS-1 or higher capability

Task: Network operations in renderer processes create attack vectors for certificate validation bypass, CORS violations, and direct socket access that could enable data exfiltration or network-based attacks. Without network service isolation, compromised renderers could directly manipulate network connections, bypass security policies, or exploit network stack vulnerabilities. Isolating network operations in a separate service with mediated access ensures certificate validation, CORS enforcement, and CSP compliance occur in a privileged, monitored context outside attacker control.

Verification:

1. Open browser internal pages to view process architecture through the process inspection interface → Test that network service crashes trigger appropriate error handling
2. Verify network service runs as separate process or is isolated within browser process → Network operations execute in isolated network service/process
3. Use network analysis tools (Wireshark, tcpdump) to monitor network requests from different processes → Renderer processes cannot directly access network APIs
4. Test that renderer processes cannot directly create network sockets → All network requests are mediated through network service
5. Verify all network requests are mediated through network service/process → Certificate validation occurs in privileged context
6. Attempt to bypass network service from renderer process and verify blocking → CORS and content security policies are enforced at network layer
7. Monitor network service sandbox status using platform security tools → Network service runs with minimal necessary privileges
8. Test certificate validation occurs in network service, not renderer → Socket creation is controlled and monitored
9. Verify CORS and CSP enforcement happens in network service layer → Network service crashes are handled gracefully

Pass Criteria: Network service is isolated AND renderers use IPC for network access AND certificate validation is isolated AND CORS/CSP enforced at network layer

Fail Criteria: Renderers have direct network access OR no network service isolation OR certificate validation in renderer OR enforcement bypassed

Evidence: Process architecture diagrams, network traffic captures, IPC logs showing network requests, sandbox status for network service, certificate validation traces, CORS enforcement logs

References:

- Chromium Network Service: https://www.chromium.org/developers/design-documents/network-stack/
- Network Sandbox: https://chromium.googlesource.com/chromium/src/+/master/services/network/README.md
- CORS and Fetch Standard: https://fetch.spec.whatwg.org/
- Certificate Transparency: https://www.certificate-transparency.org/
- Mozilla Network Security: https://wiki.mozilla.org/Security/Server_Side_TLS

Assessment: SYS-REQ-5 (Filesystem access control)

Reference: SYS-REQ-5 - Browser shall enforce strict access controls on filesystem operations, limiting access to user-approved locations

Given: A conformant browser with SYS-1 or higher capability

Task: Unrestricted filesystem access from web content enables attackers to read sensitive files, access system directories, exfiltrate browser credentials from profile directories, or write malicious files to startup locations. Without strict access controls, file:// URLs could read arbitrary local files, and File APIs could access system directories without user awareness. User-mediated filesystem access with sandboxed namespaces, IPC-brokered operations, and revocable per-origin permissions prevents unauthorized file access while enabling legitimate file operations.

Verification:

1. Attempt to read local files using file:// URLs and verify restrictions → Check that filesystem access permissions are revoked when tab closes
2. Test File API access from web content and verify it requires user gesture → file:// URL access is restricted or requires user opt-in
3. Use File System Access API to request directory access and verify user prompt appears → File API requires user gesture (click/tap) for access
4. Monitor filesystem access from renderer process using system tools (auditd, OpenBSM, Process Monitor) → File System Access API shows permission prompts
5. Verify browser cache and profile directories are protected from direct renderer access → Renderer processes cannot directly access filesystem
6. Test that sandboxed filesystem namespace limits visible paths → System directories are not accessible from web content
7. Attempt to access system directories (/etc, C:\Windows) from web content and verify blocking → Browser profile and cache protected from renderer access
8. Verify file uploads use secure IPC to broker process for filesystem access → File operations use IPC to privileged broker process
9. Test that downloaded files are stored in user-designated locations only → Filesystem permissions are per-origin and revocable

Pass Criteria: Filesystem access requires user permission AND system directories are blocked AND renderer uses IPC for file operations AND permissions are revocable

Fail Criteria: Direct filesystem access from renderer OR no user permission required OR system directories accessible OR permissions not revocable

Evidence: File access audit logs, permission prompt screenshots, filesystem monitoring traces, sandbox policy dumps, IPC logs for file operations, directory access test results

References:

- File System Access API: https://wicg.github.io/file-system-access/
- File API Specification: https://www.w3.org/TR/FileAPI/
- OWASP File Upload Security: https://cheatsheetseries.owasp.org/cheatsheets/File_Upload_Cheat_Sheet.html
- Same-Origin Policy for file: URLs: https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy#file_origins

Assessment: SYS-REQ-6 (Device API permissions)

Reference: SYS-REQ-6 - Browser shall implement permission controls for all device hardware access APIs

Given: A conformant browser with SYS-1 or higher capability

Task: Device hardware APIs provide access to sensitive capabilities like cameras, microphones, sensors, and location data that can be abused for surveillance, data theft, or privacy violations. Without permission

controls, malicious websites could silently activate cameras for spying, record audio, or track user location. Per-origin permission prompts with explicit user consent, revocability, and cross-origin isolation prevent unauthorized device access while enabling legitimate functionality for trusted origins.

Verification:

1. Navigate to test page that requests camera access using navigator.mediaDevices.getUserMedia() → Verify permissions can be permanently denied by user
2. Verify permission prompt appears and requires explicit user action → All device API access triggers permission prompts
3. Test microphone access and verify separate permission prompt → User shall explicitly grant permission (no auto-grant)
4. Check permission settings in browser UI through the content/privacy settings interface → Permissions are origin-scoped and isolated
5. Revoke camera permission and verify future access is blocked → Cross-origin iframe access is blocked by default
6. Test permission persistence across browser restarts → Permission state is persistent and survives restarts
7. Verify permissions are per-origin and not shared across origins → Users can revoke permissions at any time
8. Test permission inheritance in cross-origin iframes (should be blocked) → Denied permissions throw appropriate errors
9. Attempt to access device without permission and verify SecurityError thrown → Permission prompts include clear device/API information

Pass Criteria: Device access requires explicit permission AND prompts are clear AND permissions are per-origin AND revocation works

Fail Criteria: Device access without permission OR auto-grant occurs OR permissions not per-origin OR revocation doesn't work

Evidence: Permission prompt screenshots, settings UI showing permissions, console logs of SecurityErrors, cross-origin test results, permission persistence tests, revocation verification

References:

- Permissions API: https://www.w3.org/TR/permissions/
- Media Capture and Streams: https://www.w3.org/TR/mediacapture-streams/
- Permission Delegation: https://www.w3.org/TR/permissions-policy-1/
- MDN Permissions: https://developer.mozilla.org/en-US/docs/Web/API/Permissions_API

Assessment: SYS-REQ-7 (PWA permission management)

Reference: SYS-REQ-7 - Browser shall enforce equivalent permission controls for Progressive Web Apps as for regular web content

Given: A conformant browser with PWA-1 and SYS-1 or higher capability

Task: Progressive Web Apps installed as standalone applications may appear more trustworthy to users, creating opportunities for permission abuse if PWAs receive elevated privileges compared to web contexts. Auto-granting permissions during PWA installation would bypass informed consent, while allowing service workers to circumvent permission checks enables background surveillance. Enforcing equivalent permission controls for PWAs as web content prevents privilege escalation through installation while ensuring permission revocation upon uninstallation.

Verification:

1. Install a test PWA with manifest requesting various permissions → Verify PWA display mode (standalone, fullscreen) does not affect permission requirements
2. Verify that PWA installation does not auto-grant permissions → PWA installation does not auto-grant permissions

3. Launch PWA and trigger permission requests (camera, location, notifications) → Permission prompts appear for all sensitive APIs
4. Verify permission prompts appear identical to browser context → Permissions are per-origin, shared with web context
5. Check that PWA permissions are isolated per origin in browser settings → Uninstalling PWA revokes granted permissions
6. Test that uninstalling PWA revokes all granted permissions → Service workers cannot bypass permission checks
7. Verify PWA cannot request more permissions than web context → Display mode does not affect permission requirements
8. Test permission state is synchronized between PWA and browser views of same origin → PWA permissions visible in browser settings
9. Attempt to bypass permission via service worker and verify blocking → Permission state synchronized across contexts

Pass Criteria: PWA permissions equal to web permissions AND no auto-grant on install AND uninstall revokes permissions AND service workers respect permissions

Fail Criteria: PWA gets extra permissions OR auto-grant on install OR uninstall doesn't revoke OR service worker bypass

Evidence: PWA installation flow screenshots, permission prompt comparisons, settings showing PWA permissions, uninstall verification tests, service worker permission logs, display mode test results

References:

- Web App Manifest: https://www.w3.org/TR/appmanifest/
- PWA Permissions: https://web.dev/articles/install-criteria
- Service Worker Security: https://www.w3.org/TR/service-workers/#security-considerations
- Permissions Policy in PWAs: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Permissions-Policy

Assessment: SYS-REQ-8 (Geolocation permission enforcement)

Reference: SYS-REQ-8 - Browser shall enforce user permission requirements for geolocation API access

Given: A conformant browser with SYS-1 or higher capability

Task: Geolocation APIs expose precise user location data that enables physical tracking, stalking, burglary planning, and profiling of user movements and routines. Without HTTPS requirements and permission controls, attackers on insecure connections could intercept location data, while malicious sites could track users without consent. HTTPS enforcement, per-origin permission prompts, immediate revocation, and cross-origin isolation prevent unauthorized location tracking while enabling legitimate location-based services.

Verification:

1. Navigate to test page that calls navigator.geolocation.getCurrentPosition() → Verify location permission can be set to "ask every time"
2. Verify permission prompt appears before any location data is returned → Geolocation API requires HTTPS context (except localhost)
3. Test that HTTPS context is required for geolocation (HTTP should fail) → Permission prompt appears before any location data access
4. Grant permission and verify location data is returned → Permission is per-origin and persistent
5. Revoke permission and verify subsequent calls are denied → High-accuracy mode requires explicit permission
6. Test high-accuracy mode requires explicit permission → Cross-origin access blocked without delegation
7. Verify watchPosition() respects same permission model → Revoked permissions immediately block access

8. Test that cross-origin iframes require permission policy delegation → Users can set "ask every time" preference
9. Attempt geolocation access without user gesture and verify it still requires permission → Permission state is accessible via Permissions API

Pass Criteria: HTTPS required AND permission prompt appears AND per-origin isolation AND revocation works AND cross-origin blocked

Fail Criteria: HTTP allows access OR no permission prompt OR not per-origin OR revocation doesn't work OR cross-origin allowed

Evidence: Permission prompt screenshots, HTTPS requirement test results, permission settings showing geolocation, cross-origin test logs, revocation verification, console errors for denied access

References:

- Geolocation API: https://www.w3.org/TR/geolocation-API/
- Geolocation Security: https://w3c.github.io/geolocation-api/#security
- Secure Contexts: https://www.w3.org/TR/secure-contexts/
- MDN Geolocation: https://developer.mozilla.org/en-US/docs/Web/API/Geolocation_API

Assessment: SYS-REQ-9 (Camera/microphone access control)

Reference: SYS-REQ-9 - Browser shall enforce strict permission controls for camera and microphone access with user-visible indicators

Given: A conformant browser with SYS-1 or higher capability

Task: Camera and microphone access enables covert surveillance, recording private conversations, capturing sensitive visual information, and violating user privacy. Without visible indicators, malicious sites could secretly record audio/video for blackmail, espionage, or data theft. Permission prompts with device selection, persistent active-use indicators, immediate mid-stream revocation, and cross-origin blocking prevent unauthorized surveillance while providing user transparency and control over their devices.

Verification:

1. Navigate to test page that requests camera access via getUserMedia({video: true}) → Verify permission prompts identify requesting origin clearly
2. Verify permission prompt appears with device selection options → Separate permission prompts for camera and microphone
3. Grant permission and verify camera indicator appears in browser UI (red dot, icon) → Device selection available in permission prompt
4. Test microphone access separately and verify distinct permission prompt → Visual indicators appear when camera/microphone active
5. Request both camera and microphone and verify single combined prompt → Indicators remain visible for entire use duration
6. Verify active use indicators remain visible while devices are active → Stopping stream immediately removes indicators
7. Test that stopping media stream removes indicators → Mid-stream revocation immediately stops device access
8. Verify users can revoke permission mid-stream and devices immediately stop → Cross-origin iframe access blocked without delegation
9. Test that cross-origin iframes cannot inherit camera/microphone permissions → Permission prompts clearly show requesting origin

Pass Criteria: Permission prompts appear AND active-use indicators visible AND mid-stream revocation works AND cross-origin blocked

Fail Criteria: No permission prompt OR no indicators OR revocation doesn't stop devices OR cross-origin allowed

Evidence: Permission prompt screenshots, active camera/microphone indicator screenshots, device selection UI, cross-origin test results, mid-stream revocation tests, origin display verification

References:

- Media Capture and Streams: https://www.w3.org/TR/mediacapture-streams/
- getUserMedia Security: https://w3c.github.io/mediacapture-main/#security-and-permissions
- Firefox Camera Privacy: https://support.mozilla.org/en-US/kb/how-manage-your-camera-and-microphone-permissions

Assessment: SYS-REQ-10 (Clipboard access restrictions)

Reference: SYS-REQ-10 - Browser shall restrict clipboard access to require user interaction or explicit permission

Given: A conformant browser with SYS-1 or higher capability

Task: Clipboard access enables theft of sensitive data like passwords, credit card numbers, authentication tokens, and private communications that users copy. Unrestricted clipboard reading allows malicious sites to silently exfiltrate clipboard contents, while background clipboard access enables persistent monitoring. User gesture requirements for writing, permission prompts for reading, background access blocking, and cross-origin restrictions prevent clipboard-based data theft while enabling legitimate copy/paste functionality.

Verification:

1. Test document.execCommand('copy') and verify it requires user gesture → Verify clipboard history is not accessible without permission
2. Attempt clipboard write without user gesture and verify it's blocked → Legacy clipboard API requires user gesture
3. Test Async Clipboard API (navigator.clipboard.writeText()) and verify permission model → Async Clipboard API requires permission for reading
4. Attempt clipboard read using navigator.clipboard.readText() and verify permission prompt → Background clipboard access is blocked
5. Test clipboard access in background tab and verify it's blocked → Cross-origin access requires permission policy delegation
6. Verify cross-origin iframe clipboard access requires permission policy → Clipboard events only fire from user-initiated actions
7. Test that clipboard events (copy, cut, paste) are only triggered by user actions → Sensitive data types require explicit permission
8. Verify sensitive data types (images, rich text) require explicit permission → Service worker clipboard access is restricted
9. Test that clipboard access from service workers is restricted → No access to clipboard history without permission

Pass Criteria: User gesture required for write AND permission required for read AND background access blocked AND cross-origin requires delegation

Fail Criteria: Write without gesture OR read without permission OR background access allowed OR cross-origin not restricted

Evidence: Clipboard permission prompt screenshots, console logs showing blocked access, user gesture test results, cross-origin test logs, background tab test results, service worker restriction verification

References:

- Clipboard API: https://www.w3.org/TR/clipboard-apis/
- Async Clipboard API: https://w3c.github.io/clipboard-apis/
- Clipboard Security Model: https://w3c.github.io/clipboard-apis/#security
- MDN Clipboard API: https://developer.mozilla.org/en-US/docs/Web/API/Clipboard_API

Assessment: SYS-REQ-11 (Notification permission management)

Reference: SYS-REQ-11 - Browser shall enforce permission controls for web notifications with user-visible prompts

Given: A conformant browser with SYS-1 or higher capability

Task: Web notifications enable persistent user engagement but create vectors for notification spam, phishing through fake system alerts, social engineering attacks via deceptive messages, and user annoyance leading to permission fatigue. Without permission controls, malicious sites could bombard users with unwanted notifications or craft convincing fake alerts mimicking system messages. User gesture requirements, permission prompts, per-origin isolation, and service worker permission enforcement prevent notification abuse while enabling legitimate push messaging.

Verification:

1. Test Notification.requestPermission() and verify user prompt appears → Notification permission requires explicit user grant
2. Verify notification requests require user gesture (click/tap) → Permission prompt appears before any notification shown
3. Grant permission and test notification display using new Notification() → User gesture required to trigger permission prompt
4. Verify notifications from different origins are isolated → Permissions are per-origin and isolated
5. Test notification permission revocation and verify no more notifications appear → Service worker notifications use same permission
6. Verify service worker notifications respect same permission model → Cross-origin iframe access blocked without delegation
7. Test that cross-origin iframes cannot inherit notification permission → Permission revocation immediately prevents notifications
8. Verify permission state is accessible via Notification.permission → Notification.permission accurately reflects state
9. Test notification action buttons and verify they maintain security context → Action buttons maintain security context
10. Verify silent notifications (without sound/vibration) still require permission → All notification types require permission

Pass Criteria: Permission prompt required AND user gesture needed AND per-origin isolation AND service workers respect permissions

Fail Criteria: No permission prompt OR no user gesture required OR not per-origin OR service worker bypass

Evidence: Permission prompt screenshots, user gesture requirement tests, service worker notification tests, cross-origin test results, revocation verification, notification display examples

References:

- Notifications API: https://notifications.spec.whatwg.org/
- Notification Security: https://notifications.spec.whatwg.org/#security-and-privacy
- Push API: https://www.w3.org/TR/push-api/
- Service Worker Notifications: https://web.dev/articles/push-notifications-overview
- Chrome Notifications: https://developer.chrome.com/docs/extensions/reference/notifications/

Assessment: SYS-REQ-12 (USB device access security)

Reference: SYS-REQ-12 - Browser shall enforce strict permission and security controls for WebUSB device access

Given: A conformant browser with SYS-1 or higher capability and WebUSB support

Task: WebUSB provides direct hardware access to USB devices, creating risks of firmware attacks, data exfiltration through storage devices, keystroke logging via HID devices, and unauthorized control of sensitive peripherals. Without restrictions, malicious sites could access mass storage to read private files, reprogram device firmware, or communicate with security keys to bypass authentication. HTTPS requirements, device picker prompts, dangerous class filtering, and per-device permissions prevent USB-based attacks while enabling legitimate device interaction.

Verification:

1. Navigate to test page that calls navigator.usb.requestDevice() → WebUSB requires HTTPS context (except localhost)
2. Verify permission prompt appears with device picker showing available USB devices → Permission prompt shows device picker with clear device identification
3. Test that HTTPS context is required for WebUSB (HTTP should fail) → Only explicitly selected devices are accessible
4. Grant access to specific USB device and verify connection succeeds → User gesture required to trigger device selection
5. Verify that only explicitly granted device is accessible → Cross-origin access blocked without permission policy
6. Test device access from cross-origin iframe and verify it's blocked → Dangerous device classes (HID, storage) are not available
7. Attempt to access USB device without user gesture and verify it's blocked → Permission revocation immediately blocks device access
8. Revoke USB permission and verify device access is immediately blocked → Device access is per-origin and isolated
9. Test that dangerous device classes (HID, mass storage) are filtered from device picker → Device picker shows only appropriate devices
10. Verify device disconnect/reconnect requires re-authorization if permission was revoked → Reconnected devices respect permission state

Pass Criteria: HTTPS required AND device picker shown AND only selected devices accessible AND dangerous classes blocked

Fail Criteria: HTTP allows access OR no device picker OR all devices accessible OR dangerous classes available

Evidence: WebUSB permission prompt screenshots, device picker UI, HTTPS requirement tests, dangerous device class filtering tests, cross-origin test results, revocation verification

References:

- WebUSB API: https://wicg.github.io/webusb/
- WebUSB Security: https://wicg.github.io/webusb/#security-and-privacy
- USB Device Class Codes: https://www.usb.org/defined-class-codes
- Chrome WebUSB: https://developer.chrome.com/articles/build-for-webusb/

Assessment: SYS-REQ-13 (Bluetooth permission enforcement)

Reference: SYS-REQ-13 - Browser shall enforce permission controls and security restrictions for Web Bluetooth API

Given: A conformant browser with SYS-1 or higher capability and Web Bluetooth support

Task: Web Bluetooth enables wireless communication with Bluetooth devices, creating risks of unauthorized pairing with sensitive peripherals, GATT service exploitation to extract data or modify device settings, and attacks on Bluetooth-enabled security devices or medical equipment. Without controls, malicious sites could pair with fitness trackers to steal health data, connect to Bluetooth keyboards to log keystrokes, or interact with dangerous device types. HTTPS requirements, device picker prompts, service UUID filtering, and blocklist enforcement prevent Bluetooth-based attacks.

Verification:

1. Navigate to test page that calls navigator.bluetooth.requestDevice() → Web Bluetooth requires HTTPS context (except localhost)
2. Verify permission prompt appears with Bluetooth device picker → Permission prompt shows Bluetooth device picker
3. Test that HTTPS context is required for Web Bluetooth (HTTP should fail) → Only explicitly selected devices are accessible
4. Grant access to specific Bluetooth device and verify GATT connection → Service UUID filtering works correctly
5. Verify only explicitly granted device is accessible → User gesture required to trigger device selection
6. Test service UUID filtering in device picker → Cross-origin access blocked without delegation
7. Attempt Bluetooth access without user gesture and verify blocking → Dangerous device types blocked by blocklist
8. Test cross-origin iframe access and verify it's blocked → Permission revocation immediately blocks access
9. Revoke Bluetooth permission and verify device access is blocked → Device access is per-origin and isolated
10. Verify Bluetooth blocklist prevents access to dangerous device types → GATT operations respect permission boundaries

Pass Criteria: HTTPS required AND device picker shown AND only selected devices accessible AND blocklist enforced

Fail Criteria: HTTP allows access OR no device picker OR all devices accessible OR blocklist not enforced

Evidence: Bluetooth permission prompt screenshots, device picker UI, service UUID filtering tests, HTTPS requirement verification, blocklist enforcement tests, cross-origin test results

References:

- Web Bluetooth API: https://webbluetoothcg.github.io/web-bluetooth/
- Web Bluetooth Security: https://webbluetoothcg.github.io/web-bluetooth/#security-and-privacy
- Bluetooth GATT Services: https://www.bluetooth.com/specifications/gatt/
- Chrome Web Bluetooth: https://developer.chrome.com/articles/bluetooth/
- Web Bluetooth Blocklist: https://github.com/WebBluetoothCG/registries/blob/master/gatt_blocklist.txt

Assessment: SYS-REQ-14 (File System Access API security)

Reference: SYS-REQ-14 - Browser shall enforce strict security controls for File System Access API including user permission and path restrictions

Given: A conformant browser with SYS-1 or higher capability and File System Access API support

Task: File System Access API provides powerful capabilities to read and write local files and directories, creating risks of unauthorized data exfiltration, ransomware-style file encryption, malicious file modification, and access to sensitive system directories. Without strict controls, malicious sites could silently read user documents, modify critical files, or encrypt files for ransom. OS-native file pickers, separate write confirmation, system directory filtering, and per-access authorization prevent filesystem abuse while enabling legitimate file editing applications.

Verification:

1. Test window.showOpenFilePicker() and verify file picker dialog appears → OS file/directory picker appears for all access requests
2. Verify user should explicitly select files through OS file picker → User should explicitly select files/directories
3. Test window.showDirectoryPicker() and verify directory picker dialog → Write access requires separate confirmation

4. Grant directory access and verify files within are accessible → System directories are blocked or filtered from picker
5. Test write access requires separate user confirmation → File handles require permission on reuse after restart
6. Attempt to access system directories and verify blocking/filtering → Cross-origin access to file handles is blocked
7. Test that file handles persist and verify permission prompt on reuse → HTTPS required for persistent file handle permissions
8. Verify cross-origin iframes cannot access file handles → Permission revocation clears all granted handles
9. Test permission revocation clears all file handles → Each file/directory access is separately authorized
10. Verify HTTPS context required for persistent permissions → No programmatic file system enumeration possible

Pass Criteria: OS picker required AND write needs confirmation AND system directories blocked AND handles require reauthorization

Fail Criteria: No picker shown OR write without confirmation OR system directories accessible OR handles work without reauth

Evidence: File picker screenshots, directory picker UI, write confirmation prompts, system directory blocking tests, file handle persistence tests, cross-origin blocking verification

References:

- File System Access API: https://wicg.github.io/file-system-access/
- File System Access Security: https://wicg.github.io/file-system-access/#privacy-considerations
- Chrome File System Access: https://developer.chrome.com/articles/file-system-access/
- WHATWG File System: https://fs.spec.whatwg.org/
- OWASP File Security: https://cheatsheetseries.owasp.org/cheatsheets/File_Upload_Cheat_Sheet.html

Assessment: SYS-REQ-15 (WebUSB security controls)

Reference: SYS-REQ-15 - Browser shall implement comprehensive security controls for WebUSB including device filtering, permission management, and secure contexts

Given: A conformant browser with SYS-1 or higher capability and WebUSB support

Task: WebUSB's comprehensive device access requires layered security controls beyond basic permission prompts to prevent exploitation of protected device classes, dangerous control transfers, and vendor-sensitive devices. Without interface class filtering, attackers could claim HID interfaces to log keystrokes or access mass storage to exfiltrate files. Control transfer validation, protected class filtering, vendor opt-out respect, and secure context requirements create defense-in-depth protection for USB device interactions.

Verification:

1. Test navigator.usb.getDevices() and verify only previously authorized devices returned → Protected USB device classes are never shown in picker
2. Verify protected USB classes are filtered (HID keyboards/mice, mass storage, video, audio) → Only secure contexts can access WebUSB API
3. Test USB device access requires user activation (transient user gesture) → User activation required for device requests
4. Verify vendors can opt out devices via USB device descriptor → Previously authorized devices require getDevices() call
5. Test that WebUSB requires secure context (HTTPS or localhost) → Protected interface classes cannot be claimed
6. Attempt interface claiming on protected interface classes and verify blocking → Device connection events only for authorized devices
7. Test USB device connection events fire only for authorized devices → Control transfers are validated for safety

8. Verify control transfers are validated and potentially dangerous ones blocked → Permissions Policy successfully restricts WebUSB
9. Test that permissions-policy can restrict WebUSB in iframes → DevTools shows USB activity for debugging
10. Verify USB device access is auditable through DevTools protocol → Vendor opt-out mechanism is respected

Pass Criteria: Protected classes filtered AND secure context required AND user activation needed AND control transfers validated

Fail Criteria: Protected classes available OR insecure context works OR no user activation required OR dangerous transfers allowed

Evidence: Device picker showing filtered devices, secure context requirement tests, protected interface class blocking logs, control transfer validation tests, Permissions Policy test results

References:

- WebUSB Specification: https://wicg.github.io/webusb/
- WebUSB Protected Interface Classes: https://wicg.github.io/webusb/#protected-interface-classes
- USB Implementers Forum: https://www.usb.org/
- Chrome WebUSB Security: https://chromium.googlesource.com/chromium/src/+/refs/heads/main/docs/security/permis for-powerful-web-platform-features.md

Assessment: SYS-REQ-16 (WebBluetooth security)

Reference: SYS-REQ-16 - Browser shall implement security controls for Web Bluetooth including GATT blocklist, device filtering, and permission management

Given: A conformant browser with SYS-1 or higher capability and Web Bluetooth support

Task: Web Bluetooth GATT services provide deep access to device functionality, creating risks of HID service exploitation for keystroke injection, firmware update service abuse for device bricking or malware installation, and fingerprinting through device name enumeration. Without a comprehensive GATT blocklist, malicious sites could exploit dangerous services to compromise connected devices or user privacy. GATT blocklist enforcement, service UUID filtering, device name sanitization, and secure context requirements prevent Bluetooth-based attacks.

Verification:

1. Test navigator.bluetooth.getDevices() returns only previously authorized devices → Secure context (HTTPS/localhost) required for all Web Bluetooth APIs
2. Verify GATT blocklist prevents access to dangerous services (HID, firmware update) → User activation required for device requests
3. Test that Web Bluetooth requires secure context (HTTPS or localhost) → GATT blocklist prevents access to dangerous services/characteristics
4. Verify user activation required for requestDevice() calls → Service UUID filtering correctly limits accessible services
5. Test service UUID filters work correctly in device selection → Blocklisted characteristics return errors when accessed
6. Attempt to access blocklisted GATT characteristics and verify blocking → Optional services declared in requestDevice()
7. Test that optional services still require user awareness → Device names sanitized to prevent fingerprinting
8. Verify device name filtering prevents fingerprinting → Permissions Policy successfully restricts Web Bluetooth
9. Test permissions-policy restricts Web Bluetooth in cross-origin iframes → Bluetooth scanning requires separate permission
10. Verify Bluetooth scanning requires explicit user permission → Only previously granted devices in getDevices()

Pass Criteria: Secure context required AND GATT blocklist enforced AND user activation needed AND fingerprinting prevented

Fail Criteria: Insecure context works OR blocklist bypassed OR no user activation required OR fingerprinting possible

Evidence: Secure context requirement tests, GATT blocklist enforcement logs, service UUID filtering results, fingerprinting prevention tests, Permissions Policy test results

References:

- Web Bluetooth Specification: https://webbluetoothcg.github.io/web-bluetooth/
- Web Bluetooth GATT Blocklist: https://github.com/WebBluetoothCG/registries/blob/master/gatt_blocklist.txt
- Bluetooth GATT Specifications: https://www.bluetooth.com/specifications/specs/
- Web Bluetooth Security Model: https://webbluetoothcg.github.io/web-bluetooth/#security-and-privacy-considerations
- Chrome Web Bluetooth Security: https://sites.google.com/a/chromium.org/dev/developers/design-documents/bluetooth-design-docs

Assessment: SYS-REQ-17 (WebNFC permission management)

Reference: SYS-REQ-17 - Browser shall enforce permission controls for Web NFC API with user prompts and secure context requirements

Given: A conformant browser with SYS-1 or higher capability and Web NFC support

Task: Web NFC enables reading and writing NFC tags, creating risks of malicious tag writing to deploy phishing attacks, NFC relay attacks, unauthorized data collection from contactless payment cards, and privacy violations through persistent NFC scanning. Background NFC access could enable covert tag reading while users are unaware. Secure context requirements, permission prompts with user gestures, background operation blocking, and dangerous tag filtering prevent NFC-based attacks while enabling legitimate tag interactions.

Verification:

1. Test NDEFReader.scan() and verify permission prompt appears → Secure context (HTTPS/localhost) required for Web NFC
2. Verify Web NFC requires secure context (HTTPS or localhost) → Permission prompt appears before NFC access granted
3. Test that NFC access requires user gesture for permission prompt → User gesture required to trigger permission prompt
4. Grant NFC permission and verify scan operations work → Both scan and write operations respect same permission
5. Test NDEFReader.write() and verify it respects same permission → Cross-origin iframe access blocked without delegation
6. Verify cross-origin iframe NFC access is blocked without permission policy → Permission revocation stops active scans
7. Test permission revocation immediately stops NFC scanning → Background pages cannot perform NFC operations
8. Verify NFC operations blocked when page in background → Dangerous tag operations are restricted
9. Test that dangerous NFC tag types are filtered or sandboxed → Permissions are per-origin and isolated
10. Verify NFC access is per-origin and isolated → NFC indicators show when NFC is active

Pass Criteria: Secure context required AND permission prompt shown AND user gesture needed AND background access blocked

Fail Criteria: Insecure context works OR no permission prompt OR no user gesture required OR background access allowed

Evidence: NFC permission prompt screenshots, secure context requirement tests, user gesture verification, background access blocking tests, cross-origin test results, dangerous tag filtering verification

References:

- Web NFC API: https://w3c.github.io/web-nfc/
- Web NFC Security: https://w3c.github.io/web-nfc/#security-and-privacy
- NFC Forum Specifications: https://nfc-forum.org/our-work/specification-releases/
- Web NFC Explainer: https://github.com/w3c/web-nfc/blob/gh-pages/EXPLAINER.md

Assessment: SYS-REQ-18 (Sensor API permissions)

Reference: SYS-REQ-18 - Browser shall enforce permission controls for Generic Sensor APIs including accelerometer, gyroscope, and magnetometer

Given: A conformant browser with SYS-1 or higher capability and Sensor API support

Task: Generic Sensor APIs expose motion and environmental data that enable fingerprinting, keylogging through motion analysis, PIN theft via accelerometer side channels, and location tracking through magnetometer readings. High-frequency sensor access amplifies these attacks by providing precise timing data for cryptographic attacks. Secure context requirements, permission controls, background operation suspension, frequency limits, and Permissions Policy enforcement prevent sensor-based attacks while enabling legitimate motion and orientation detection.

Verification:

1. Test Accelerometer creation and verify permission prompt or policy enforcement → Secure context required for all Sensor APIs
2. Verify secure context required for sensor APIs → Permission prompts or policies apply before sensor access
3. Test Gyroscope access and verify same permission model → High-frequency access may require explicit permission
4. Create Magnetometer sensor and verify permissions → Sensors automatically pause in background
5. Test that high-frequency sensor access may require additional permissions → Cross-origin access requires Permissions Policy delegation
6. Verify sensors stop when page moves to background → Permissions are per-origin and isolated
7. Test cross-origin iframe sensor access requires permission policy delegation → Privacy-sensitive sensors have additional restrictions
8. Verify sensor permissions are per-origin → Permissions Policy can block sensor access
9. Test that ambient light sensor respects privacy considerations → Sensor frequency is limited to prevent fingerprinting
10. Verify sensor access can be blocked via Permissions Policy → Clear user controls for sensor permissions

Pass Criteria: Secure context required AND permissions enforced AND background pausing works AND Permissions Policy respected

Fail Criteria: Insecure context works OR no permissions enforced OR background access allowed OR policy ignored

Evidence: Sensor permission prompt screenshots, secure context requirement tests, background pausing verification, Permissions Policy test results, frequency limiting tests, cross-origin blocking verification

References:

- Generic Sensor API: https://www.w3.org/TR/generic-sensor/
- Sensor Security Model: https://www.w3.org/TR/generic-sensor/#security-and-privacy
- Accelerometer API: https://www.w3.org/TR/accelerometer/
- Gyroscope API: https://www.w3.org/TR/gyroscope/
- Permissions Policy: https://www.w3.org/TR/permissions-policy-1/

Assessment: SYS-REQ-19 (Battery Status API restrictions)

Reference: SYS-REQ-19 - Browser shall implement privacy restrictions for Battery Status API to prevent fingerprinting

Given: A conformant browser with SYS-1 or higher capability

Task: Battery Status API historically enabled precise device fingerprinting through battery level, charging time, and discharge rate patterns that uniquely identify devices across browsing sessions and origins. High-precision battery data combined with timing information creates a persistent tracking identifier resistant to cookie deletion. Rounding battery levels, quantizing timing data, throttling updates, and rate limiting prevent battery-based fingerprinting while providing sufficient information for legitimate power management features.

Verification:

1. Test navigator.getBattery() and observe battery information returned → Battery level rounded to coarse granularity (e.g., 1% or 5%)
2. Verify battery level is rounded to prevent high-precision fingerprinting → Timing information quantized to prevent precise measurements
3. Test battery timing information is quantized to prevent tracking → Update frequency throttled to prevent tracking
4. Verify battery status updates are throttled → Secure context recommended for battery API access
5. Test that battery information is not available in insecure contexts → Cross-origin access requires Permissions Policy
6. Verify battery status in cross-origin iframes requires permission policy → Rate limiting prevents rapid polling
7. Test that frequent battery queries are rate-limited → Battery API can be disabled by user/policy
8. Verify battery API can be disabled via browser settings or policy → Charging state changes debounced
9. Test that battery charging state changes are debounced → No historical battery data exposed
10. Verify no access to detailed battery analytics or history → API surface minimized for privacy

Pass Criteria: Battery data quantized AND updates throttled AND rate limiting enforced AND no detailed analytics exposed

Fail Criteria: Precise battery data OR no throttling OR no rate limiting OR historical data exposed

Evidence: Battery level precision tests, timing quantization measurements, update frequency analysis, rate limiting verification, cross-origin test results, privacy analysis reports

References:

- Battery Status API: https://www.w3.org/TR/battery-status/
- Battery API Privacy Concerns: https://www.w3.org/TR/battery-status/#privacy-considerations
- Web API Privacy: https://www.w3.org/TR/fingerprinting-guidance/
- Chrome Battery Status: https://chromestatus.com/feature/4537134732017664

Assessment: SYS-REQ-20 (Hardware resource limits)

Reference: SYS-REQ-20 - Browser shall enforce resource limits to prevent excessive consumption of CPU, memory, and system resources

Given: A conformant browser with SYS-1 or higher capability

Task: Unrestricted hardware resource consumption enables denial-of-service attacks that freeze browsers, crash systems, drain battery, or make devices unusable through memory exhaustion, CPU monopolization, or GPU resource depletion. Malicious scripts with infinite loops or excessive allocations can render browsers unresponsive. Per-origin resource limits, background throttling, script timeouts, and memory quotas prevent resource-based DoS attacks while maintaining browser and system responsiveness.

Verification:

1. Create test page that attempts to allocate excessive memory and verify limits → Memory allocation limits prevent excessive consumption
2. Test CPU-intensive operations and verify throttling or limits applied → CPU-intensive operations are throttled
3. Monitor browser resource usage with intensive JavaScript loops → Background tabs have reduced resource quotas
4. Test WebWorker resource limits and verify isolation → WebWorkers have separate resource limits
5. Verify background tab resource throttling is active → WebAssembly memory is bounded and enforced
6. Test WebAssembly memory limits and verify enforcement → GPU memory limits prevent resource exhaustion
7. Monitor GPU memory usage and verify limits on WebGL contexts → Script timeouts prevent infinite loops
8. Test that runaway scripts trigger timeout warnings or termination → Resource limits are per-origin or per-process
9. Verify resource limits apply per-origin or per-process → Browser UI remains responsive under load
10. Test that browser remains responsive under resource pressure → User can terminate runaway processes/tabs

Pass Criteria: Memory limits enforced AND CPU throttling active AND background throttling works AND script timeouts prevent hangs

Fail Criteria: No memory limits OR no CPU throttling OR background tabs not throttled OR scripts can hang indefinitely

Evidence: Memory allocation test results, CPU usage graphs showing throttling, background tab resource measurements, WebWorker limit tests, script timeout logs, browser responsiveness tests

References:

- WebAssembly Memory: https://webassembly.github.io/spec/core/syntax/modules.html#memories
- Script Execution Limits: https://html.spec.whatwg.org/multipage/webappapis.html#long-tasks
- Firefox Process Limits: https://wiki.mozilla.org/Project_Fission

Assessment: SYS-REQ-21 (Memory isolation enforcement)

Reference: SYS-REQ-21 - Browser shall enforce memory isolation between processes to prevent cross-process memory access

Given: A conformant browser with SYS-1 or higher capability

Task: Memory isolation failures enable cross-process data theft, Spectre-class attacks to read arbitrary memory, privilege escalation through memory corruption, and exploitation of use-after-free vulnerabilities. Without ASLR, SharedArrayBuffer restrictions, and Spectre mitigations, attackers can reliably exploit memory vulnerabilities to breach process boundaries. Memory isolation with ASLR, cross-origin isolation requirements, memory zeroing, and hardware protections prevents cross-process memory attacks.

Verification:

1. Launch browser with multiple processes and identify their memory spaces → Each process has isolated virtual memory space
2. Use memory analysis tools to verify process memory isolation (valgrind, windbg, lldb) → Renderer processes cannot read browser process memory
3. Test that renderer processes cannot access browser process memory → SharedArrayBuffer requires explicit CORS headers
4. Verify SharedArrayBuffer requires cross-origin isolation → ASLR randomizes memory addresses per process
5. Test that different origins cannot share memory without explicit mechanisms → Spectre/Meltdown mitigations prevent side-channel leaks
6. Verify ASLR (Address Space Layout Randomization) is enabled → Memory is zeroed on deallocation

7. Test that Spectre mitigations prevent speculative execution memory leaks → No cross-process memory leaks detectable
8. Verify memory is zeroed when deallocated and reused → Hardware NX/DEP prevents code execution in data pages
9. Test that memory dumps do not leak data between processes → Process crashes don't leak memory to other processes
10. Verify hardware memory protection (NX, DEP) is enabled → Memory forensics shows proper isolation

Pass Criteria: Process memory isolated AND SharedArrayBuffer restricted AND ASLR enabled AND Spectre mitigations active

Fail Criteria: Cross-process memory access possible OR SharedArrayBuffer unrestricted OR no ASLR OR Spectre vulnerable

Evidence: Memory map analysis, process memory dumps, SharedArrayBuffer test results, ASLR verification, Spectre/Meltdown mitigation tests, memory leak detection reports

References:

- SharedArrayBuffer Security: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Sha
- Spectre Mitigations: https://www.chromium.org/Home/chromium-security/ssca/
- ASLR: https://en.wikipedia.org/wiki/Address_space_layout_randomization
- Site Isolation: https://www.chromium.org/Home/chromium-security/site-isolation/

Assessment: SYS-REQ-22 (CPU resource quotas)

Reference: SYS-REQ-22 - Browser shall enforce CPU resource quotas to prevent any single origin from monopolizing processor resources

Given: A conformant browser with SYS-1 or higher capability

Task: CPU resource monopolization by malicious scripts enables cryptojacking attacks that steal processing power for cryptocurrency mining, denial-of-service through CPU exhaustion, battery drain on mobile devices, and system slowdowns affecting user productivity. Background tabs consuming CPU enable covert mining operations. CPU quotas with background throttling, timer throttling, requestAnimationFrame suspension, and per-origin limits prevent CPU monopolization while maintaining responsiveness for foreground content.

Verification:

1. Create test page with intensive CPU computation (tight loop or crypto operations) → Background tabs throttled to <1% CPU typically
2. Monitor CPU usage per process using OS tools (top, Task Manager, Activity Monitor) → Foreground tabs have priority CPU access
3. Test that background tabs have reduced CPU quotas → WebWorker CPU usage tracked per origin
4. Verify WebWorker CPU usage is counted toward origin quota → Invisible elements have reduced CPU quotas
5. Test CPU throttling for invisible pages (display:none, zero opacity) → Timers throttled in background (1Hz or lower)
6. Verify timer throttling for background processes (setTimeout, setInterval) → requestAnimationFrame paused when not visible
7. Test that requestAnimationFrame is paused for background tabs → Browser task manager shows per-tab CPU usage
8. Monitor overall system CPU and verify browser doesn't exceed limits → Users can terminate high-CPU tabs easily
9. Test that users can identify and terminate high-CPU tabs → System CPU usage remains reasonable under load
10. Verify CPU priority scheduling favors foreground/visible content → CPU scheduling prioritizes user-visible content

Pass Criteria: Background throttling active AND timers throttled AND rAF paused AND user can terminate high-CPU tabs

Fail Criteria: No background throttling OR timers not throttled OR rAF active in background OR no termination controls

Evidence: CPU usage graphs per tab, background throttling measurements, timer frequency tests, rAF execution logs, task manager screenshots, system CPU monitoring

References:

- Page Lifecycle API: https://developer.chrome.com/blog/page-lifecycle-api/
- Timer Throttling: https://developer.mozilla.org/en-US/docs/Web/API/setTimeout#throttling
- requestAnimationFrame: https://developer.mozilla.org/en-US/docs/Web/API/window/requestAnimationFrame
- Firefox Process Priority: https://wiki.mozilla.org/Project_Fission#Process_prioritization

Assessment: SYS-REQ-23 (Network bandwidth limits)

Reference: SYS-REQ-23 - Browser shall enforce network bandwidth limits to prevent excessive network resource consumption

Given: A conformant browser with SYS-1 or higher capability

Task: Uncontrolled network resource consumption enables bandwidth exhaustion attacks, connection pool depletion preventing legitimate requests, network-based denial-of-service, and excessive data usage on metered connections. Malicious origins opening unlimited connections can monopolize network resources. Per-origin connection limits, HTTP/2 stream limits, WebSocket quotas, and background deprioritization prevent network resource abuse while enabling efficient multiplexing and user control.

Verification:

1. Create test page that attempts high-volume network requests → Connection limits enforced per origin (6-10 connections)
2. Monitor network usage per origin using browser DevTools Network panel → HTTP/2 stream limits enforced (max concurrent streams)
3. Test connection limits per origin (typically 6-10 connections) → Background tabs have reduced network priority
4. Verify HTTP/2 and HTTP/3 multiplexing limits → WebSocket connections limited per origin
5. Test bandwidth throttling for background tabs → Large transfers pausable and cancellable
6. Verify WebSocket connection limits per origin → Connection pooling prevents excessive sockets
7. Test that large downloads can be paused or cancelled → Browser task manager shows network usage per tab
8. Monitor overall browser network usage and verify limits → Network throttling options available in DevTools
9. Test that fetch() requests respect connection pooling limits → Overall browser network reasonable under load
10. Verify users can identify and control network-heavy tabs → Users can control network-heavy operations

Pass Criteria: Connection limits enforced AND background deprioritization AND WebSocket limits AND user controls available

Fail Criteria: Unlimited connections OR no background priority OR unlimited WebSockets OR no user controls

Evidence: Network connection graphs, DevTools Network panel screenshots, connection limit test results, WebSocket limit verification, background priority measurements, task manager network stats

References:

- HTTP Connection Management: https://developer.mozilla.org/en-US/docs/Web/HTTP/Connection_management_in_
- HTTP/2 Specification: https://httpwg.org/specs/rfc7540.html

- Chrome Network Throttling: https://developer.chrome.com/docs/devtools/network/reference/#throttling
- WebSocket Limits: https://datatracker.ietf.org/doc/html/rfc6455
- Firefox Network Priority: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Priority

Assessment: SYS-REQ-24 (Storage quota enforcement)

Reference: SYS-REQ-24 - Browser shall enforce storage quotas for origin-scoped storage mechanisms including IndexedDB, Cache API, and local storage

Given: A conformant browser with SYS-1 or higher capability

Task: Unrestricted storage enables disk space exhaustion attacks, storage-based denial-of-service, tracking through persistent data storage, and system instability from full disks. Malicious origins filling storage prevent legitimate applications from functioning. Per-origin storage quotas, persistent storage permissions, QuotaExceededError enforcement, and separate incognito limits prevent storage abuse while enabling adequate space for web applications and maintaining system stability.

Verification:

1. Test StorageManager.estimate() to query available storage quota → Storage quotas enforced per-origin (typically 50-100MB default)
2. Attempt to exceed storage quota using IndexedDB and verify QuotaExceededError → StorageManager API accurately reports usage and quota
3. Test localStorage quota limits (typically 5-10MB) → QuotaExceededError thrown when limits reached
4. Verify Cache API respects storage quota → localStorage limited to 5-10MB per origin
5. Test that storage quota is per-origin and isolated → Cache API storage counted toward quota
6. Verify persistent storage requires explicit user permission → Persistent storage requires user permission
7. Test that storage usage is accurately reported in browser settings → Browser settings show per-origin storage usage
8. Verify storage can be cleared per-origin by user → Users can clear storage per-origin
9. Test that incognito/private mode has separate, limited storage → Incognito mode has separate, limited quota
10. Verify storage quota warnings or prompts for large allocations → Large storage requests trigger user prompts

Pass Criteria: Quotas enforced per-origin AND QuotaExceededError thrown AND persistent storage requires permission AND user controls available

Fail Criteria: No quota enforcement OR errors not thrown OR persistent storage auto-granted OR no user controls

Evidence: StorageManager.estimate() results, QuotaExceededError screenshots, localStorage limit tests, Cache API quota tests, browser settings storage UI, incognito mode quota verification

References:

- Storage API: https://storage.spec.whatwg.org/
- StorageManager: https://developer.mozilla.org/en-US/docs/Web/API/StorageManager
- IndexedDB Quota: https://www.w3.org/TR/IndexedDB/#dom-idbdatabase-transaction
- Chrome Storage Quotas: https://web.dev/articles/storage-for-the-web
- Firefox Storage Limits: https://developer.mozilla.org/en-US/docs/Web/API/Storage_API/Storage_quotas_and_evictio

Assessment: SYS-REQ-25 (Process priority management)

Reference: SYS-REQ-25 - Browser shall implement process priority management to ensure responsive user experience and fair resource allocation

Given: A conformant browser with SYS-1 or higher capability

Task: Without process priority management, background tabs can monopolize CPU resources causing foreground tab lag, audio playback in background can be interrupted unfairly, and browser responsiveness suffers from equal treatment of all processes. Priority inversion where background processes starve foreground operations degrades user experience. Dynamic process priority management ensuring foreground preference, audio continuity, and fair scheduling prevents resource contention while maintaining responsive user interaction.

Verification:

1. Launch browser with multiple tabs and identify process priorities using OS tools → Foreground tabs run at higher OS process priority
2. Verify foreground tab processes have higher priority than background → Background tabs run at reduced priority (low or below normal)
3. Test that audio-playing tabs maintain elevated priority → Audio-playing tabs maintain adequate priority
4. Verify visible tabs have higher priority than hidden tabs → Priority changes dynamically when switching tabs
5. Test process priority changes when switching tabs → CPU scheduler gives preference to high-priority processes
6. Monitor CPU scheduling and verify foreground processes get preference → Extension processes have appropriate priority
7. Test that extension processes have appropriate priority → Utility processes prioritized by criticality
8. Verify utility processes (network, GPU) have appropriate priority → Frozen tabs have lowest priority
9. Test that frozen background tabs have lowest priority → No process starvation occurs
10. Verify process priorities don't allow starvation → User-facing processes most responsive

Pass Criteria: Foreground tabs high priority AND background tabs low priority AND dynamic priority adjustment AND no starvation

Fail Criteria: All same priority OR no background reduction OR static priorities OR starvation occurs

Evidence: Process priority listings (nice values, Windows priority classes), CPU scheduling analysis, tab switching priority changes, audio playback priority verification, process responsiveness measurements

References:

- Chrome Process Model: https://www.chromium.org/developers/design-documents/multi-process-architecture/
- Linux Process Priority: https://man7.org/linux/man-pages/man2/nice.2.html
- Windows Process Priority: https://docs.microsoft.com/en-us/windows/win32/procthread/scheduling-priorities
- Page Lifecycle: https://developer.chrome.com/blog/page-lifecycle-api/

Assessment: SYS-REQ-26 (Sandbox escape prevention)

Reference: SYS-REQ-26 - Browser shall implement multiple layers of defense to prevent sandbox escape attacks

Given: A conformant browser with SYS-1 or higher capability

Task: Sandbox escapes are high-severity vulnerabilities that allow compromised renderers to break containment and gain full system access, enabling malware installation, data exfiltration, and system compromise. Single-layer sandboxes are vulnerable to bypass through exploitation of implementation bugs or kernel vulnerabilities. Defense-in-depth with multiple sandbox layers, syscall filtering, IPC validation, kernel exploit mitigations, and continuous fuzzing prevents sandbox escapes even when individual defenses are bypassed.

Verification:

1. Review browser's sandbox architecture documentation and layers of defense → Multiple sandbox layers (syscall filter, namespace isolation, capability dropping)
2. Test syscall filtering using strace/dtruss to verify blocked calls → Dangerous syscalls blocked (execve, ptrace, mount, etc.)

3. Attempt known sandbox escape techniques and verify they're mitigated → IPC messages validated in broker process
4. Verify IPC validation prevents malformed messages from escaping sandbox → Process debugging APIs blocked
5. Test that renderer processes cannot ptrace or debug other processes → Memory corruption contained within sandbox
6. Verify memory corruption in renderer doesn't allow privilege escalation → GPU sandbox independent from renderer
7. Test that GPU process sandbox is separate from renderer sandbox → Kernel exploit mitigations active (SMEP, SMAP, KPTI)
8. Verify kernel exploit mitigations (SMEP, SMAP, KPTI) are active → Regular fuzzing for sandbox bypasses
9. Test fuzzing results for sandbox bypass vulnerabilities → Security updates address known escapes
10. Verify security updates address discovered sandbox escapes → Defense-in-depth architecture

Pass Criteria: Multiple sandbox layers AND syscall filtering AND IPC validation AND kernel mitigations active

Fail Criteria: Single layer only OR syscalls not filtered OR IPC not validated OR kernel mitigations missing

Evidence: Sandbox architecture documentation, syscall trace logs, sandbox escape test results, IPC validation logs, kernel mitigation verification, fuzzing reports, security update changelogs

References:

- Chromium Sandbox: https://chromium.googlesource.com/chromium/src/+/master/docs/design/sandbox.md
- Linux Namespace Isolation: https://man7.org/linux/man-pages/man7/namespaces.7.html
- Seccomp BPF: https://www.kernel.org/doc/html/latest/userspace-api/seccomp_filter.html
- SMEP and SMAP: https://www.intel.com/content/www/us/en/developer/articles/technical/software-security-guidance.html
- Project Zero Sandbox Escapes: https://googleprojectzero.blogspot.com/

Assessment: SYS-REQ-27 (Speculative execution mitigations)

Reference: SYS-REQ-27 - Browser shall implement mitigations for speculative execution vulnerabilities including Spectre and Meltdown

Given: A conformant browser with SYS-1 or higher capability

Task: Spectre and Meltdown enable attackers to exploit speculative execution in modern CPUs to read arbitrary memory across security boundaries, including passwords, encryption keys, and cross-origin data. Without mitigations, malicious JavaScript can use timing side channels to leak sensitive data from other processes or origins. Site Isolation, timer quantization, SharedArrayBuffer restrictions, CORB, and COOP/COEP headers prevent speculative execution attacks by ensuring cross-origin data never shares address space.

Verification:

1. Verify Site Isolation is enabled using chrome://process-internals or equivalent → Site Isolation active for all sites or at least sensitive origins
2. Test that high-resolution timers are reduced precision (performance.now() granularity) → Timer precision reduced to 100 microseconds or coarser
3. Verify SharedArrayBuffer requires cross-origin isolation headers → SharedArrayBuffer requires COOP+COEP headers
4. Test that cross-origin data is not accessible in same process → Cross-origin data in separate processes
5. Verify CORB (Cross-Origin Read Blocking) prevents sensitive data leaks → CORB blocks cross-origin reads of sensitive MIME types
6. Test that speculative execution cannot leak cross-origin data → Spectre PoC exploits fail to leak data
7. Verify process-per-site-instance isolation for sensitive origins → Process-per-site-instance for banks, login pages

8. Test that COOP/COEP headers enable process isolation → COOP/COEP enable strict process isolation
9. Verify kernel page-table isolation (KPTI) is active on system → KPTI active on vulnerable systems
10. Test that Spectre PoC exploits are mitigated → Regular updates address new speculative execution vulnerabilities

Pass Criteria: Site Isolation enabled AND timer precision reduced AND SharedArrayBuffer restricted AND CORB active

Fail Criteria: No Site Isolation OR high-precision timers OR SharedArrayBuffer unrestricted OR CORB disabled

Evidence: Site Isolation status screenshots, timer precision measurements, SharedArrayBuffer tests with/without headers, CORB blocking logs, Spectre PoC test results, KPTI verification

References:

- Spectre Mitigations: https://www.chromium.org/Home/chromium-security/ssca/
- Site Isolation: https://www.chromium.org/Home/chromium-security/site-isolation/
- SharedArrayBuffer: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/SharedArray
- CORB: https://chromium.googlesource.com/chromium/src/+/master/services/network/cross_origin_read_blocking_ex
- Spectre Attack: https://spectreattack.com/

Assessment: SYS-REQ-28 (Side-channel attack mitigations)

Reference: SYS-REQ-28 - Browser shall implement mitigations for side-channel attacks including timing attacks, cache attacks, and fingerprinting

Given: A conformant browser with SYS-1 or higher capability

Task: Side-channel attacks exploit timing differences, cache behavior, rendering performance, and other observable effects to infer sensitive information like user history, cross-origin data, or cryptographic keys. High-precision timers enable microarchitectural attacks, CSS :visited timing leaks browsing history, and cache timing reveals cross-origin content. Timer quantization, process isolation for cache partitioning, cross-origin timing sanitization, and fingerprinting surface minimization prevent side-channel information leakage.

Verification:

1. Test timer precision reduction (performance.now(), Date.now()) to verify quantization → Timer precision reduced (100 microseconds or coarser)
2. Verify SharedArrayBuffer is disabled or requires isolation headers → SharedArrayBuffer requires COOP+COEP isolation
3. Test that cache timing attacks are mitigated through process isolation → Process isolation prevents cache-based side channels
4. Verify rendering timing cannot leak cross-origin information → Rendering timing doesn't leak cross-origin data
5. Test that event loop timing is not exploitable for side channels → Event loop timing not exploitable
6. Verify CSS timing attacks are mitigated (e.g., :visited link timing) → :visited link styling restricted to prevent timing attacks
7. Test that network timing doesn't leak cross-origin data → Network timing (Resource Timing API) sanitized for cross-origin
8. Verify SVG filter timing attacks are prevented → SVG filter timing attacks mitigated
9. Test that WebGL cannot be used for timing side channels → WebGL timing attacks prevented through various restrictions
10. Verify fingerprinting surfaces are minimized or permissions-gated → Fingerprinting APIs require permission or are restricted

Pass Criteria: Timer quantization active AND SharedArrayBuffer isolated AND process isolation prevents cache attacks AND cross-origin timing sanitized

Fail Criteria: High-precision timers OR SharedArrayBuffer unrestricted OR no cache isolation OR timing leaks exist

Evidence: Timer precision tests, SharedArrayBuffer isolation verification, cache timing attack tests, cross-origin timing measurements, fingerprinting surface analysis, side-channel PoC test results

References:

- Web Platform Security: https://www.w3.org/TR/fingerprinting-guidance/
- Timing Attacks: https://www.w3.org/TR/hr-time-2/#clock-resolution
- CSS :visited Privacy: https://developer.mozilla.org/en-US/docs/Web/CSS/Privacy_and_the_:visited_selector
- Resource Timing: https://www.w3.org/TR/resource-timing-2/
- Fingerprinting Resistance: https://brave.com/privacy-updates/3-fingerprint-randomization/

Assessment: SYS-REQ-29 (Hardware token security)

Reference: SYS-REQ-29 - Browser shall implement secure access controls for hardware security tokens and authenticators (WebAuthn)

Given: A conformant browser with SYS-1 or higher capability and WebAuthn support

Task: Hardware security tokens provide strong authentication but require secure browser integration to prevent phishing, credential theft, and authenticator cloning attacks. Without HTTPS requirements and origin binding, attackers could steal credentials or use fake authenticators. User verification and presence requirements prevent remote attacks. HTTPS enforcement, origin-bound credentials, attestation validation, user presence requirements, and extension isolation ensure WebAuthn security while enabling passwordless authentication.

Verification:

1. Test WebAuthn registration with hardware security key (e.g., YubiKey) → WebAuthn requires secure context (HTTPS or localhost)
2. Verify registration requires HTTPS context (except localhost) → User verification enforced when requested by relying party
3. Test that user verification (PIN/biometric) is enforced when required → Attestation validation prevents cloned/fake authenticators
4. Verify authenticator attestation is validated correctly → Credentials strictly origin-bound (site isolation)
5. Test that credentials are origin-bound and not accessible cross-origin → User presence required (physical interaction)
6. Verify user presence is required (physical touch/button press) → CTAP2 protocol correctly implemented
7. Test that CTAP2 protocol is properly implemented for FIDO2 devices → Extensions cannot intercept or modify WebAuthn
8. Verify browser extensions cannot intercept WebAuthn operations → Users choose authenticator from picker
9. Test that authenticator selection respects user choice → Platform authenticators use OS-level secure enclaves
10. Verify platform authenticators (Touch ID, Windows Hello) are isolated → Private keys never accessible to JavaScript

Pass Criteria: HTTPS required AND user verification enforced AND origin-bound credentials AND user presence required

Fail Criteria: HTTP allowed OR no user verification OR cross-origin access OR no user presence

Evidence: WebAuthn registration screenshots, attestation validation logs, cross-origin test results, user presence verification, CTAP2 protocol traces, platform authenticator tests

References:

- WebAuthn Specification: https://www.w3.org/TR/webauthn-2/

- FIDO2 CTAP: https://fidoalliance.org/specs/fido-v2.0-ps-20190130/fido-client-to-authenticator-protocol-v2.0-ps-20190130.html
- WebAuthn Security: https://www.w3.org/TR/webauthn-2/#sctn-security-considerations
- FIDO Security Reference: https://fidoalliance.org/specifications/

Assessment: SYS-REQ-30 (Accessibility API security)

Reference: SYS-REQ-30 - Browser shall implement security controls for accessibility APIs to prevent information leakage and unauthorized access

Given: A conformant browser with SYS-1 or higher capability

Task: Accessibility APIs expose detailed page structure, content, and user interactions to assistive technologies, creating vectors for information leakage, fingerprinting, unauthorized content scraping, and privacy violations if not properly controlled. Malicious software masquerading as assistive technology could harvest passwords, track user actions, or extract sensitive content. OS-level permission requirements, sensitive data masking, cross-origin restrictions, and remote access authentication prevent accessibility API abuse while serving users with disabilities.

Verification:

1. Enable OS accessibility features (screen reader, voice control) → Accessibility API access requires OS-level permission
2. Verify browser exposes accessibility tree to authorized assistive technology only → Only authorized assistive technology can access accessibility tree
3. Test that accessibility API requires user opt-in or OS-level permission → Sensitive content (passwords) properly masked in accessibility tree
4. Verify accessibility tree doesn't leak sensitive data (passwords, hidden content) → Cross-origin content has limited accessibility exposure
5. Test that cross-origin content has restricted accessibility exposure → Accessibility events sanitized to prevent leakage
6. Verify accessibility events don't leak timing or user interaction information → Remote accessibility requires authentication
7. Test that remote accessibility access (enterprise tools) requires authentication → Accessibility API cannot be used for fingerprinting
8. Verify accessibility API cannot be abused for fingerprinting → ARIA attributes don't leak sensitive information
9. Test that ARIA attributes don't leak security-sensitive information → Accessibility access auditable
10. Verify screen reader access is logged for security auditing → Browser communicates securely with assistive technology

Pass Criteria: OS permission required AND sensitive data masked AND cross-origin limited AND remote access authenticated

Fail Criteria: No permission required OR passwords exposed OR cross-origin fully accessible OR remote access unauthenticated

Evidence: Accessibility permission flows, accessibility tree dumps showing masking, cross-origin accessibility tests, remote access authentication verification, fingerprinting tests, audit logs

References:

- WAI-ARIA: https://www.w3.org/TR/wai-aria-1.2/
- Accessibility Object Model: https://wicg.github.io/aom/spec/
- Chrome Accessibility: https://www.chromium.org/developers/design-documents/accessibility/
- macOS Accessibility: https://developer.apple.com/documentation/accessibility
- Windows Accessibility: https://docs.microsoft.com/en-us/windows/win32/winauto/entry-uiauto-win32

Assessment: SYS-REQ-31 (Native messaging restrictions)

Reference: SYS-REQ-31 - Browser shall enforce security restrictions on native messaging between extensions and native applications

Given: A conformant browser with SYS-1 and EXT-1 or higher capability

Task: Native messaging bridges the browser extension sandbox and native applications, creating risks of sandbox escape, privilege escalation, arbitrary code execution, and malware installation if not properly controlled. Malicious extensions could exploit native messaging to execute system commands, access sensitive files, or install malware. Manifest permissions, host registration, extension ID validation, message validation, and privilege limitation prevent native messaging abuse while enabling legitimate browser-native integration.

Verification:

1. Create test extension with nativeMessaging permission → nativeMessaging permission required in manifest
2. Verify extension requires explicit permission declaration in manifest → Native host is registered in browser's native messaging directory
3. Test that native messaging host is registered with browser → Only permitted extensions (by ID) can connect to host
4. Verify only explicitly allowed extensions can communicate with native host → User should manually install native host application
5. Test that native messaging requires user installation of native component → Messages validated by browser before passing to/from host
6. Verify message validation occurs on browser side → Native host runs with limited OS privileges
7. Test that native host process runs with limited privileges → Extension sandbox not bypassed via native messaging
8. Verify native messaging cannot bypass extension sandbox → Extension uninstall terminates native connections
9. Test that uninstalling extension terminates native messaging connections → Enterprise policies can control native messaging
10. Verify enterprise policies can restrict or acceptlist native messaging → Native host path validation prevents arbitrary execution

Pass Criteria: Manifest permission required AND host registration required AND extension ID validation AND user installs host

Fail Criteria: No permission required OR no host registration OR any extension can connect OR automatic host installation

Evidence: Extension manifest examples, native host registration files, permission validation tests, message validation logs, privilege analysis of native host, enterprise policy tests

References:

- Chrome Native Messaging: https://developer.chrome.com/docs/extensions/develop/concepts/native-messaging
- Firefox Native Messaging: https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/Native_messagin

Assessment: SYS-REQ-32 (Host OS integration security)

Reference: SYS-REQ-32 - Browser shall securely integrate with host operating system features while maintaining sandbox boundaries and security isolation

Given: A conformant browser with SYS-1 or higher capability

Task: Host OS integration provides essential functionality but creates attack surfaces for credential theft, certificate validation bypass, policy evasion, and privilege escalation if not properly secured. Insecure credential access exposes passwords, certificate store bypass enables MITM attacks, and policy violations undermine

enterprise security. Secure IPC for credentials, OS certificate store usage, file association validation, policy compliance, and update signature verification maintain security boundaries while enabling OS integration.

Verification:

1. Test OS credential integration (Windows Credential Manager, macOS Keychain) and verify secure access → OS credential storage accessed securely (encrypted IPC)
2. Verify browser uses OS certificate store with proper validation → OS certificate store used with proper validation
3. Test OS notification integration respects permission model → OS notifications require permission and are per-origin
4. Verify file type association handlers are validated and sandboxed → File associations validated before handler invocation
5. Test protocol handler registration requires user confirmation → Protocol handlers require user confirmation
6. Verify OS-level print dialog prevents renderer access to print system → Print system accessed via broker, not renderer
7. Test that OS share APIs require user gesture and confirmation → OS share APIs require user gesture
8. Verify browser respects OS security policies (AppLocker, Gatekeeper, SELinux) → Browser respects OS security policies
9. Test that browser updates use OS-level signature verification → Updates verified with OS-level signatures
10. Verify browser profile directories use OS access controls → Profile directories protected with OS ACLs

Pass Criteria: Secure credential access AND certificate store used AND handlers validated AND security policies respected

Fail Criteria: Insecure credential access OR certificate store bypassed OR handlers not validated OR policies ignored

Evidence: Credential access flow analysis, certificate store usage verification, file association tests, protocol handler registration flows, OS policy compliance tests, update signature verification, ACL analysis

References:

- Windows Credential Manager: https://docs.microsoft.com/en-us/windows/security/identity-protection/credential-guard/
- macOS Keychain: https://developer.apple.com/documentation/security/keychain_services
- Code Signing: https://developer.apple.com/documentation/xcode/notarizing_macos_software_before_distribution
- Windows AppLocker: https://docs.microsoft.com/en-us/windows/security/threat-protection/windows-defender-application-control/applocker/applocker-overview
- SELinux: https://www.redhat.com/en/topics/linux/what-is-selinux

Assessment: SYS-REQ-33 (Block all device API access at SYS-0)

Reference: SYS-REQ-33 - Browser shall block all device API access (no camera, microphone, location, etc.) at SYS-0 capability level

Given: A conformant browser with SYS-0 capability (fully sandboxed, no system access)

Task: At SYS-0 capability, browsers shall completely deny access to all device hardware APIs to minimize attack surface and prevent fingerprinting, location tracking, surveillance, and privacy violations. Allowing camera, microphone, geolocation, or sensor APIs at this level contradicts the security posture of zero system access, enabling malicious sites to track users, capture audio/video without consent, or perform sophisticated fingerprinting attacks. Complete device API blocking with permission UI disabled, API surface removed, and feature policy enforcement ensures zero hardware access.

Verification:

1. Navigate to test pages that request camera access using browser media APIs and verify requests are blocked → All camera access requests fail without user prompting

2. Test microphone access requests and confirm they fail immediately without permission prompts → All microphone requests fail immediately
3. Attempt to access geolocation API and verify it returns an error without prompting → Geolocation API returns error without permission dialogs
4. Test device motion and orientation sensor APIs and confirm they are blocked → Motion and orientation sensors blocked
5. Attempt to access ambient light sensor API and verify it is unavailable → Ambient light sensor unavailable
6. Test battery status API access and confirm it is blocked → Battery status API blocked
7. Verify proximity sensor API is not available at SYS-0 level → Proximity sensor not exposed
8. Test vibration API and confirm it is blocked or returns immediately without effect → Vibration API disabled or no-op
9. Attempt to access NFC, Bluetooth, and USB APIs and verify they are completely unavailable → NFC, Bluetooth, USB APIs completely unavailable
10. Verify that permission UI for all device APIs is disabled and not rendered → No permission UI rendered for any device API

Pass Criteria: All device hardware APIs blocked AND no permission prompts shown AND API surface not exposed AND feature policy enforced

Fail Criteria: Any device API accessible OR permission prompts shown OR APIs available in JavaScript scope

Evidence: API access test results showing blocked requests, JavaScript console showing undefined API objects, network logs showing no device enumeration, permission UI absence verification, feature policy analysis

References:

- Media Capture and Streams API: https://www.w3.org/TR/mediacapture-streams/
- Geolocation API: https://www.w3.org/TR/geolocation-API/
- Generic Sensor API: https://www.w3.org/TR/generic-sensor/
- Web Bluetooth: https://webbluetoothcg.github.io/web-bluetooth/
- WebUSB: https://wicg.github.io/webusb/

Assessment: SYS-REQ-34 (Block native messaging interfaces at SYS-0)

Reference: SYS-REQ-34 - Browser shall not expose any native messaging interfaces at SYS-0 capability level

Given: A conformant browser with SYS-0 capability (fully sandboxed, no system access)

Task: Native messaging interfaces allow extensions and web content to communicate with native applications on the host system, creating privileged channels for file access, process execution, and system command invocation. At SYS-0, these interfaces shall be completely disabled to prevent sandbox escape, unauthorized native code execution, and host system compromise. Exposing native messaging contradicts zero system access policy, enabling malicious code to bypass browser security boundaries. Complete removal of native messaging APIs, configuration interfaces, and manifest support prevents these attack vectors.

Verification:

1. Search browser extension API documentation to verify native messaging APIs are not documented for SYS-0 → Native messaging APIs undefined in extension context
2. Attempt to load an extension with native messaging permissions and verify it is rejected → Extensions with native messaging permissions rejected at install
3. Test extension manifest with "nativeMessaging" permission and confirm browser refuses to load it → Manifests declaring nativeMessaging fail validation
4. Attempt to call native messaging APIs from extension context and verify they are undefined → Native messaging host directories not accessed

5. Search for native messaging host configuration directories and verify they are not created or accessed → No permission UI for native messaging shown
6. Test that extension installation UI does not show native messaging permission requests → Browser does not spawn native host processes
7. Verify browser process does not spawn or communicate with native messaging hosts → API documentation excludes native messaging at SYS-0
8. Review browser API surface and confirm native messaging functions are not exposed → Configuration UI lacks native messaging options
9. Test that browser configuration UI does not offer native messaging settings → Security logs show zero native messaging attempts
10. Verify browser security logs do not show any native messaging connection attempts → Feature completely absent from browser capabilities

Pass Criteria: Native messaging APIs undefined AND extensions with native messaging rejected AND no host processes spawned AND configuration disabled

Fail Criteria: Native messaging APIs accessible OR extensions with permission load OR native hosts spawned OR configuration options available

Evidence: Extension manifest rejection logs, API availability tests showing undefined objects, process monitoring showing no native messaging hosts, configuration UI screenshots, security log analysis

References:

- Chrome Native Messaging: https://developer.chrome.com/docs/apps/nativeMessaging/
- Mozilla Native Messaging: https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/Native_messagin
- Browser Extension Security: https://www.w3.org/community/webextensions/

Assessment: SYS-REQ-35 (Block host OS integration at SYS-0)

Reference: SYS-REQ-35 - Browser shall block all host OS integration features at SYS-0 capability level

Given: A conformant browser with SYS-0 capability (fully sandboxed, no system access)

Task: Host OS integration features including file associations, protocol handlers, OS notifications, credential storage access, certificate store integration, and system dialog invocation create pathways between sandboxed browser content and privileged OS functionality. At SYS-0, all integration shall be blocked to maintain complete isolation, preventing protocol handler hijacking, file association exploits, notification spam, credential theft, and OS-level privilege escalation. Disabling OS integration prevents malicious content from escaping browser boundaries to manipulate host system configuration or access sensitive OS services.

Verification:

1. Attempt to register custom protocol handlers and verify registration is blocked → Protocol handler registration fails
2. Test file type association requests and confirm they are rejected → File association requests rejected
3. Attempt to trigger OS notification API and verify it fails without permission prompts → OS notifications blocked without prompts
4. Test access to OS credential storage (keychain, credential manager) and confirm it is blocked → Credential storage not accessible
5. Verify browser does not access or validate certificates using OS certificate store → OS certificate store not used
6. Attempt to invoke OS print dialog and confirm it is unavailable or non-functional → Print dialog unavailable
7. Test OS share/send APIs and verify they are completely disabled → Share APIs disabled
8. Attempt to register as default browser and confirm the feature is unavailable → Default browser registration blocked
9. Test system dialog invocation (file picker, color picker) and verify dialogs do not appear → System dialogs do not invoke

10. Verify browser respects OS security policies but does not integrate with OS security features → No OS security feature integration

Pass Criteria: All protocol handlers blocked AND file associations rejected AND OS notifications disabled AND credential storage inaccessible AND system dialogs blocked

Fail Criteria: Protocol handlers register OR file associations created OR notifications work OR credential storage accessible OR system dialogs appear

Evidence: Protocol registration test results, file association rejection logs, notification API failure messages, credential storage access attempts showing errors, certificate validation method analysis, dialog invocation tests

References:

- Custom Protocol Handlers: https://developer.mozilla.org/en-US/docs/Web/API/Navigator/registerProtocolHandler
- Web App Manifest: https://www.w3.org/TR/appmanifest/
- Notifications API: https://notifications.spec.whatwg.org/
- Credential Management API: https://www.w3.org/TR/credential-management-1/

Assessment: SYS-REQ-36 (PWA permission scope restrictions)

Reference: SYS-REQ-36 - PWA permissions shall not exceed web context permissions at SYS-2 capability level

Given: A conformant browser with SYS-2 capability supporting Progressive Web Applications

Task: Progressive Web Applications installed from browsers should not gain additional permissions beyond what the same web content has in browser context, preventing PWA installation from becoming a privilege escalation vector. If PWAs automatically gain filesystem access, native API access, or reduced permission prompting not available to web contexts, malicious actors can exploit PWA installation as a persistence mechanism with elevated privileges. Enforcing permission parity ensures PWAs remain scoped to the same security model as their web origins, with identical permission requirements, prompting, and revocation.

Verification:

1. Open a web application in browser tab and document available API permissions → PWA and web contexts have identical API availability
2. Install the same application as PWA and enumerate available APIs in installed context → Camera permissions identical across contexts
3. Compare permission scopes between browser tab context and PWA context for camera API → Microphone prompts same for PWA and web
4. Test microphone access in both contexts and verify identical permission prompting behavior → Geolocation requires same consent
5. Verify geolocation API requires same user consent in PWA as in browser tab → Filesystem access equally restricted
6. Test filesystem access in browser tab and PWA, confirming equal restrictions → USB permissions identical
7. Attempt to access USB devices in both contexts and verify same permission model applies → Bluetooth requires same user consent
8. Test Bluetooth API access and confirm PWAs don't bypass permission requirements → Notification permissions consistent
9. Verify notification permissions require same user consent regardless of install state → PWA installation grants no automatic permissions
10. Test that PWA installation does not auto-grant any permissions that web context requires explicit consent for → Permission UI and flows identical

Pass Criteria: PWA permissions identical to web context AND no automatic permission grants on installation AND same permission prompting behavior

Fail Criteria: PWA has more permissions than web context OR installation auto-grants permissions OR PWA bypasses permission prompts

Evidence: Permission enumeration comparison between contexts, API availability tests, permission prompt screenshots, installation permission logs, permission revocation testing

References:

- Progressive Web Apps: https://web.dev/progressive-web-apps/
- Permissions API: https://www.w3.org/TR/permissions/
- PWA Security: https://www.w3.org/TR/appmanifest/#security-and-privacy

Assessment: SYS-REQ-37 (PWA installation permission restrictions)

Reference: SYS-REQ-37 - PWA installation shall not auto-grant extended permissions at SYS-2 capability level

Given: A conformant browser with SYS-2 capability supporting Progressive Web Applications

Task: PWA installation should not silently grant permissions that would normally require explicit user consent during runtime, as automatic permission grants bypass user agency and create security risks. If installing a PWA automatically grants camera, microphone, geolocation, filesystem, or device API access without user awareness, malicious applications can gain surveillance and data access capabilities through social engineering of the installation flow alone. All permissions shall require explicit runtime consent after installation, ensuring users understand and authorize each capability access.

Verification:

1. Review PWA manifest for requested permissions and capabilities → Installation completes without granting permissions
2. Install a PWA that declares camera, microphone, and location permissions in manifest → Camera requires runtime permission prompt
3. After installation completes, verify no permissions are automatically granted without user consent → Microphone requires explicit user consent after install
4. Launch the installed PWA and attempt to access camera, verifying permission prompt appears → Geolocation prompts user at first access
5. Test microphone access from installed PWA and confirm runtime permission required → Filesystem permissions not granted at install
6. Attempt geolocation access and verify user should explicitly grant consent → Permission management UI shows no auto-grants
7. Test filesystem access and confirm permission prompt required at runtime → Installation flow does not include permission bundling
8. Review browser permission management UI and verify no permissions granted at install time → Install and permission actions are separate
9. Test that PWA installation flow does not include bundled permission consent → All permissions require post-install runtime consent
10. Verify installation UI separates “install app” from “grant permissions” actions → Users can install PWA and deny all permissions

Pass Criteria: Installation grants no permissions AND all APIs require runtime consent AND installation UI separate from permission granting

Fail Criteria: Any permissions auto-granted at install OR APIs accessible without runtime prompts OR installation bundles permission consent

Evidence: PWA installation flow screenshots, post-install permission status checks, API access tests showing permission prompts, permission management UI state, installation logs

References:

- Web App Manifest: https://www.w3.org/TR/appmanifest/
- Permissions API: https://www.w3.org/TR/permissions/
- PWA Installation: https://web.dev/install-criteria/

Assessment: SYS-REQ-38 (PWA uninstallation permission revocation)

Reference: SYS-REQ-38 - PWA uninstallation shall revoke all granted permissions at SYS-2 capability level

Given: A conformant browser with SYS-2 capability supporting Progressive Web Applications

Task: When users uninstall Progressive Web Applications, all previously granted permissions shall be immediately revoked to prevent residual access to device capabilities, storage, and user data. If permissions persist after uninstallation, uninstalled malicious PWAs could retain background notification access, maintain filesystem handles, preserve authentication tokens, or continue data collection through background sync. Complete permission revocation with persistent storage clearing, credential removal, and registration cleanup ensures uninstallation fully removes PWA capabilities.

Verification:

1. Install a PWA and grant it multiple permissions (camera, microphone, notifications, location) → All permissions revoked after uninstall
2. Document all granted permissions through browser permission management interface → Camera access no longer available
3. Verify PWA has active notification subscriptions and background sync registrations → Microphone permissions removed
4. Test that PWA has access to persistent storage, service workers, and cached data → Notification subscriptions cancelled
5. Uninstall the PWA through browser UI → Background sync registrations removed
6. Immediately after uninstall, check permission management UI to verify all permissions revoked → Service workers unregistered
7. Attempt to access previously granted APIs and verify they are no longer accessible → Persistent storage cleared
8. Verify notification subscriptions are cancelled and no notifications delivered → IndexedDB data removed or inaccessible
9. Check that service workers are unregistered and no longer run in background → Cache storage cleared
10. Verify persistent storage, IndexedDB, and cache storage are cleared or inaccessible → No residual PWA capabilities remain

Pass Criteria: All permissions revoked on uninstall AND service workers unregistered AND persistent storage cleared AND no residual access

Fail Criteria: Any permissions remain after uninstall OR service workers continue running OR storage persists OR notifications still delivered

Evidence: Permission status before and after uninstall, service worker registration logs, storage inspection showing cleared data, notification delivery tests, background sync status checks

References:

- Service Workers: https://www.w3.org/TR/service-workers/
- Storage API: https://storage.spec.whatwg.org/
- Push API: https://www.w3.org/TR/push-api/
- Background Sync: https://wicg.github.io/background-sync/spec/

Assessment: SYS-REQ-39 (PWA permission audit logging)

Reference: SYS-REQ-39 - Browser shall maintain audit log of all PWA permission grants and revocations at SYS-2 capability level

Given: A conformant browser with SYS-2 capability supporting Progressive Web Applications

Task: Comprehensive audit logging of PWA permission lifecycle events enables security monitoring, forensic analysis, compliance verification, and attack detection. Without detailed logs of permission grants, revocations, installation, and uninstallation events, organizations cannot detect suspicious permission requests, identify compromised applications, investigate security incidents, or demonstrate regulatory compliance. Tamper-resistant audit logs with timestamps, origin identification, permission types, and user actions provide accountability and security visibility.

Verification:

1. Enable audit logging in browser configuration (if optional) or verify it is enabled by default → Installation events logged with timestamp and origin
2. Install a PWA and grant it camera and microphone permissions → Permission grants recorded per API type
3. Review audit logs and verify they contain installation event with timestamp and origin → User actions (grant/deny) captured in logs
4. Check logs for permission grant events showing camera and microphone with user action recorded → Permission revocations logged with details
5. Revoke microphone permission through browser settings → Uninstallation events recorded
6. Verify revocation event logged with timestamp, permission type, and user/admin action → All revocations on uninstall documented
7. Uninstall the PWA completely → Logs include complete metadata (time, origin, permission, action)
8. Confirm uninstallation event appears in logs with all associated permission revocations → Logs protected from tampering
9. Test that logs include sufficient detail: timestamp, origin, permission type, user action, result → Enterprise admins can export and analyze logs
10. Verify logs are protected from tampering by application code or unauthorized access → Log retention policy documented and enforced

Pass Criteria: All PWA permission events logged AND logs include complete metadata AND logs protected from tampering AND retention policy enforced

Fail Criteria: Permission events not logged OR logs lack critical metadata OR logs can be tampered with OR no retention policy

Evidence: Audit log exports showing permission lifecycle events, log integrity verification, log access control testing, retention policy documentation, log format analysis

References:

- Audit Logging Best Practices: https://owasp.org/www-community/Log_Injection
- NIST Audit Logging: https://csrc.nist.gov/publications/detail/sp/800-92/final
- Browser Security Logging: https://chromium.googlesource.com/chromium/src/+/master/docs/security/audit-logging.md

Assessment: SYS-REQ-40 (Enterprise PWA permission policy controls)

Reference: SYS-REQ-40 - Enterprise administrators shall be able to configure PWA permission policies at SYS-2 capability level

Given: A conformant browser with SYS-2 capability supporting Progressive Web Applications and enterprise policy management

Task: Enterprise environments require centralized control over PWA capabilities to enforce organizational security policies, prevent data exfiltration, comply with regulations, and manage risk. Without enterprise policy controls, users can install PWAs that violate corporate security standards, access sensitive device APIs, persist beyond management, or bypass security monitoring. Group policy integration with allowlist/blocklist controls, forced permission denials, and audit requirements enables IT administrators to govern PWA security posture.

Verification:

1. Access browser enterprise policy configuration interface (Group Policy, MDM, policy JSON) → Policy blocks non-allowlisted PWA installations
2. Configure policy to block PWA installation from all origins except allowlisted domains → Camera permissions enforced denied by policy
3. Attempt to install PWA from non-allowlisted origin and verify installation is blocked → Audit logging requirement cannot be bypassed
4. Configure policy to deny camera permissions for all PWAs enterprise-wide → Storage quotas enforced per policy
5. Install allowed PWA and attempt camera access, verifying it is blocked by policy → Specific capabilities blocked as configured
6. Configure policy requiring audit logging for all PWA permission events → Policy overrides user preferences
7. Verify audit logging is enforced and cannot be disabled by users → Policy overrides PWA manifest permissions
8. Test policy to set maximum storage quota for PWAs → Centralized policy management available
9. Configure policy blocking specific PWA capabilities (background sync, notifications) → Policy violations logged to enterprise monitoring
10. Verify policy overrides user preferences and PWA manifest requests → Users cannot circumvent policy restrictions

Pass Criteria: Enterprise policies enforced AND allowlist/blocklist controls work AND permission denials override user grants AND policy violations logged

Fail Criteria: Policies not enforced OR users can bypass restrictions OR PWAs override policy OR no violation logging

Evidence: Policy configuration documentation, installation blocking tests, permission denial verification, audit logs showing policy enforcement, user bypass attempt results, policy override testing

References:

- Chrome Enterprise Policies: https://chromeenterprise.google/policies/
- Firefox Enterprise Policies: https://github.com/mozilla/policy-templates
- Microsoft Edge Policies: https://docs.microsoft.com/en-us/deployedge/microsoft-edge-policies
- Mobile Device Management: https://www.w3.org/TR/app-manifest/#controlling-installation

Assessment: SYS-REQ-41 (Native-equivalent API access controls at SYS-3)

Reference: SYS-REQ-41 - Browser shall allow native-equivalent API access with user consent at SYS-3 capability level

Given: A conformant browser with SYS-3 capability (native-equivalent system access)

Task: At SYS-3 capability, browsers enable native-equivalent API access for advanced web applications requiring deep system integration, but this access should require explicit, informed user consent to prevent abuse. Native-equivalent APIs may include direct filesystem access, arbitrary USB/serial device communication, advanced Bluetooth capabilities, or system command execution, creating significant security risks if granted without user understanding. Clear consent flows with risk warnings, per-API authorization, revocability, and audit trails balance functionality with security.

Verification:

1. Identify native-equivalent APIs available at SYS-3 (File System Access API, WebUSB, Serial API, etc.) → Native-equivalent APIs require consent prompts
2. Attempt to access File System Access API and verify user consent prompt appears with clear warnings → Prompts include clear risk warnings

3. Test that consent prompt explains the risks and capabilities being granted → Per-API authorization required
4. Grant consent and verify API access is enabled for the origin → Permissions can be revoked through settings
5. Revoke permission through browser settings and verify API access is blocked → WebUSB requires per-device consent
6. Test WebUSB device access and confirm per-device consent prompts appear → Serial API requires per-port authorization
7. Verify consent for one USB device does not grant access to all devices → Enhanced Bluetooth requires additional consent
8. Test Serial API access and confirm user should explicitly authorize each serial port → All API usage logged comprehensively
9. Attempt Bluetooth access beyond standard Web Bluetooth scope and verify enhanced consent required → Consent prompts indicate SYS-3 privilege level
10. Verify all native-equivalent API usage logged for security auditing → Users can deny access without breaking basic web functionality

Pass Criteria: Native-equivalent APIs gated by consent AND prompts include risk warnings AND per-API authorization AND permissions revocable AND usage logged

Fail Criteria: APIs accessible without consent OR prompts lack risk warnings OR blanket authorization granted OR permissions not revocable

Evidence: Consent prompt screenshots with risk warnings, permission grant/revoke testing, API access logs, per-device authorization verification, permission management UI analysis

References:

- File System Access API: https://wicg.github.io/file-system-access/
- WebUSB API: https://wicg.github.io/webusb/
- Web Serial API: https://wicg.github.io/serial/
- Permissions API: https://www.w3.org/TR/permissions/

Assessment: SYS-REQ-42 (Transparent UI for native integration at SYS-3)

Reference: SYS-REQ-42 - Browser shall provide transparent UI for all native integration features at SYS-3 capability level

Given: A conformant browser with SYS-3 capability (native-equivalent system access)

Task: Users shall have complete visibility into which web applications have native-equivalent capabilities, what system resources they access, and when access occurs, enabling informed decisions and security awareness. Without transparent UI, malicious applications can silently access filesystem, devices, or system APIs without user knowledge, facilitating data theft, surveillance, and system compromise. Comprehensive UI with active access indicators, permission management interfaces, usage history, and clear capability explanations ensures user awareness and control.

Verification:

1. Grant filesystem access to a web application through File System Access API → Active filesystem access indicator visible
2. Verify browser UI displays active indicator when application accesses filesystem → Permission management UI lists all native capabilities
3. Access browser permission management interface and verify it lists all granted native capabilities → Capabilities explained in clear language
4. Test that permission UI clearly explains each granted capability in user-friendly language → Usage history accessible and detailed
5. Verify browser provides usage history showing when and how native APIs were accessed → USB access shows active indicator

6. Test active USB device access and confirm browser shows indicator (icon, notification, or status) → Serial connections display connection status
7. Open serial port connection and verify browser displays active connection indicator → Indicators link to detailed information
8. Test that users can click indicators to view details about active native API usage → Users can revoke permissions through UI
9. Verify permission management UI allows users to view, modify, and revoke native capabilities → Revocation immediately effective and reflected in UI
10. Test that revoking permission immediately stops API access and updates UI → No native access occurs without UI visibility

Pass Criteria: Active access indicators visible AND permission management comprehensive AND usage history available AND revocation through UI functional

Fail Criteria: No access indicators OR permission UI incomplete OR no usage history OR revocation not available

Evidence: Active indicator screenshots during API usage, permission management UI documentation, usage history exports, indicator interaction testing, revocation workflow verification

References:

- User Interface Security: https://www.w3.org/TR/security-privacy-questionnaire/
- Browser UI Design: https://chromium.googlesource.com/chromium/src/+/master/docs/ui/
- Permission UX Best Practices: https://web.dev/permissions-ux/

Assessment: SYS-REQ-43 (Security warnings for privileged API access at SYS-3)

Reference: SYS-REQ-43 - Browser shall display security warnings for privileged API access at SYS-3 capability level

Given: A conformant browser with SYS-3 capability (native-equivalent system access)

Task: Privileged API access at SYS-3 grants capabilities equivalent to native applications, including filesystem access, device control, and system integration, requiring prominent security warnings to prevent user confusion and social engineering attacks. Without clear warnings, users may not understand the security implications of granting native-equivalent access, enabling attackers to gain persistent filesystem access, device control, or system-level privileges through deceptive prompts. Warning UI with risk explanations, consequences, and alternatives helps users make informed security decisions.

Verification:

1. Trigger File System Access API permission request from a web application → Filesystem access prompts include clear warnings
2. Verify permission prompt includes clear security warning about filesystem access risks → Warnings explain read/write capabilities and risks
3. Confirm warning explains that the site will gain ability to read/write files directly → USB prompts warn about device control implications
4. Test WebUSB permission prompt and verify it warns about device control risks → USB warnings note firmware access potential
5. Confirm USB warning explains potential for device firmware access and modification → Serial API warnings explain hardware communication risks
6. Trigger Serial API permission and verify warning about direct hardware communication → Warnings visually distinct from standard prompts
7. Test that warnings are visually distinct (color, icons, prominence) from normal permission prompts → Dismissal requires deliberate user action
8. Verify warnings cannot be easily dismissed or bypassed through rapid clicking → Explicit risk acknowledgment required

9. Test that warnings require explicit "I understand the risks" acknowledgment → Warnings shown on first and periodic subsequent access
10. Verify warnings are shown on first access and periodically for ongoing privileged access → Warning language calibrated to user understanding level

Pass Criteria: Privileged APIs show security warnings AND warnings explain specific risks AND visual distinction from normal prompts AND explicit acknowledgment required

Fail Criteria: No warnings shown OR warnings lack risk details OR warnings visually identical to normal prompts OR easy bypass available

Evidence: Permission prompt screenshots with security warnings, warning text analysis, user interaction flow documentation, bypass attempt testing, warning frequency verification

References:

- Security Warning Design: https://www.usenix.org/conference/soups2019/presentation/reeder
- Effective Permission Prompts: https://web.dev/permission-ux/
- User Security Indicators: https://www.w3.org/TR/security-privacy-questionnaire/

Assessment: SYS-REQ-44 (Extended system access audit logging at SYS-3)

Reference: SYS-REQ-44 - Browser shall log all extended system access for security auditing at SYS-3 capability level

Given: A conformant browser with SYS-3 capability (native-equivalent system access)

Task: Native-equivalent system access at SYS-3 creates significant security risk requiring comprehensive audit logging for attack detection, forensic investigation, compliance verification, and insider threat monitoring. Without detailed logs capturing filesystem access, device interactions, API usage patterns, and permission changes, organizations cannot detect data exfiltration, identify compromised applications, investigate security incidents, or meet regulatory audit requirements. Tamper-resistant logs with complete metadata enable security visibility and accountability.

Verification:

1. Enable or verify audit logging is active for SYS-3 capability level → All filesystem operations logged with paths
2. Grant filesystem access to a web application and access multiple files → USB device interactions logged comprehensively
3. Review audit logs and verify they contain entries for each file access with path, operation, timestamp → Serial port access and data transmission recorded
4. Connect and communicate with a USB device through WebUSB → Permission changes logged with full context
5. Verify logs capture device connection, data transfer, and disconnection events → Logs include complete metadata for each event
6. Open serial port connection and transmit data → Log entries are timestamped with high precision
7. Confirm logs document serial port access, configuration, and data transmission → Logs protected from tampering or deletion
8. Revoke a permission through browser settings → Enterprise administrators can export logs
9. Verify permission revocation logged with timestamp, origin, and permission type → Log retention meets security and compliance needs
10. Test that logs include complete metadata: timestamp, origin, API, resource, operation, result, user context → Logs integrate with SIEM and security monitoring tools

Pass Criteria: All extended system access logged AND logs include complete metadata AND tamper protection enforced AND enterprise export available

Fail Criteria: System access not logged OR logs lack critical metadata OR logs can be tampered with OR no export capability

Evidence: Audit log exports showing system access events, log completeness verification, tamper protection testing, SIEM integration demonstration, retention policy documentation

References:

- Security Audit Logging: https://csrc.nist.gov/publications/detail/sp/800-92/final
- Browser Security Logs: https://chromium.googlesource.com/chromium/src/+/master/docs/security/audit-logging.md
- SIEM Integration: https://owasp.org/www-community/Log_Injection

Assessment: SYS-REQ-45 (User permission review and revocation at SYS-3)

Reference: SYS-REQ-45 - Users shall be able to review and revoke all system permissions at SYS-3 capability level

Given: A conformant browser with SYS-3 capability (native-equivalent system access)

Task: Users shall have complete visibility and control over all granted system permissions to manage their security posture, respond to threats, and maintain privacy. Without comprehensive permission review and revocation interfaces, users cannot identify over-privileged applications, remove access from compromised sites, or recover from accidental permission grants. Centralized permission management with per-origin listings, granular revocation, usage visibility, and immediate enforcement enables user control and security hygiene.

Verification:

1. Grant multiple system permissions (filesystem, USB, serial, notifications) to various web origins → All granted permissions visible in management UI
2. Navigate to browser permission management interface → Permissions organized by origin and type
3. Verify interface displays all granted permissions organized by origin and permission type → High-privilege permissions clearly marked
4. Test that interface clearly identifies high-privilege permissions (filesystem, device access) → Individual permission revocation functional
5. Select a permission and revoke it through the management interface → Revocation takes effect immediately
6. Immediately test that the web application can no longer access the revoked resource → Applications receive permission denial after revocation
7. Verify error or permission denial shown to application after revocation → Bulk revocation available per origin
8. Test bulk revocation of all permissions for a specific origin → Usage history shown for each permission
9. Confirm all permissions removed and application loses all system access → No hidden or unrevocable permissions exist
10. Verify permission management UI shows usage history for each permission → Interface accessible and user-friendly

Pass Criteria: All permissions visible in management UI AND individual revocation functional AND immediate enforcement AND usage history available

Fail Criteria: Permissions not listed OR revocation doesn't work OR delayed enforcement OR no usage visibility

Evidence: Permission management UI screenshots, revocation testing showing immediate effect, usage history exports, bulk revocation verification, hidden permission audit

References:

- Permissions API: https://www.w3.org/TR/permissions/
- Permission Management UX: https://web.dev/permission-ux/
- Browser Privacy Controls: https://www.w3.org/TR/privacy-controls/

Assessment: SYS-REQ-46 (Enterprise restriction of native-equivalent features at SYS-3)

Reference: SYS-REQ-46 - Enterprise administrators shall be able to restrict native-equivalent features at SYS-3 capability level

Given: A conformant browser with SYS-3 capability supporting native-equivalent system access and enterprise policy management

Task: Enterprise environments require centralized control to restrict native-equivalent features that pose unacceptable risk to organizational security, intellectual property, or compliance requirements. Without enterprise policy controls, users at SYS-3 can grant unrestricted filesystem access, device control, or system integration to web applications, enabling data exfiltration, malware installation, or regulatory violations. Group policy integration with capability blocking, origin restrictions, and forced denials enables IT security governance.

Verification:

1. Access enterprise policy configuration interface (Group Policy, MDM, policy JSON) → File System Access API blocked by policy
2. Configure policy to block File System Access API organization-wide → Policy blocks API with clear error message
3. Attempt to use File System Access API as user and verify policy blocks access with clear error → WebUSB restricted to allowlisted origins
4. Configure policy to allow WebUSB only for specific allowlisted origins → Non-allowlisted origins denied device access
5. Test WebUSB from non-allowlisted origin and verify policy denies access → Allowlisted origins function normally
6. Test WebUSB from allowlisted origin and verify it functions → Serial API completely disabled by policy
7. Configure policy to completely disable Serial API → Administrator approval workflow enforced
8. Attempt serial port access and verify it is blocked by policy → Users cannot bypass policy restrictions
9. Configure policy requiring administrator approval for any SYS-3 capability grants → Policy violations logged to enterprise monitoring
10. Test that user permission grants require admin authorization workflow → Policies centrally managed and enforced

Pass Criteria: Enterprise policies block native-equivalent features AND origin allowlists enforced AND administrator approval available AND policy violations logged

Fail Criteria: Policies not enforced OR users bypass restrictions OR allowlists ignored OR no violation logging

Evidence: Policy configuration documentation, API blocking verification, origin allowlist testing, admin approval workflow demonstration, policy bypass attempt results, violation logs

References:

- Chrome Enterprise Policies: https://chromeenterprise.google/policies/
- Firefox Enterprise Policies: https://github.com/mozilla/policy-templates
- Microsoft Edge Policies: https://docs.microsoft.com/en-us/deployedge/microsoft-edge-policies
- Enterprise Browser Management: https://www.chromium.org/administrators/

Assessment: SYS-REQ-47 (Native integration audit documentation at SYS-3)

Reference: SYS-REQ-47 - All native integration exceptions shall be documented and auditable at SYS-3 capability level

Given: A conformant browser with SYS-3 capability (native-equivalent system access)

Task: Native integration exceptions that deviate from standard web security model shall be comprehensively documented to enable security review, vulnerability assessment, compliance auditing, and risk management.

Without documentation of security boundaries relaxed, APIs exposed beyond web standards, or sandbox exceptions granted, security teams cannot assess risk posture, auditors cannot verify compliance, and developers cannot understand security implications. Architecture documentation with threat models, exception justifications, and mitigation controls enables informed security governance.

Verification:

1. Review browser security documentation to identify documented native integration exceptions → Native integration exceptions fully documented
2. Verify documentation explains each exception's purpose, risks, and mitigations → Documentation explains purpose, risks, and mitigations
3. Confirm documentation includes threat model for native-equivalent API surface → Threat models available for native APIs
4. Review code comments and design documents for File System Access API implementation → Implementation aligns with documentation
5. Verify implementation matches documented security model and exception handling → Runtime information available about active integration
6. Test that browser provides runtime information about active native integration (developer tools, about:security) → Enterprise admin documentation comprehensive
7. Verify enterprise administrators can access detailed documentation of native features → Audit logs align with documented behavior
8. Review security audit logs and confirm they align with documented native integration behavior → No undocumented exceptions exist
9. Test that documented exceptions are the only ones present (no undocumented backdoors) → Documentation kept current with releases
10. Verify documentation is maintained and updated with browser releases → Security review process documented

Pass Criteria: All exceptions documented AND documentation includes threat models AND implementation matches documentation AND no undocumented exceptions

Fail Criteria: Exceptions not documented OR no threat models OR implementation deviates OR undocumented exceptions found

Evidence: Security documentation review, threat model analysis, code implementation verification, runtime security interface examination, audit log alignment testing, exception enumeration

References:

- Secure Development Lifecycle: https://www.microsoft.com/en-us/securityengineering/sdl/
- Threat Modeling: https://owasp.org/www-community/Threat_Modeling
- Security Documentation: https://cheatsheetseries.owasp.org/cheatsheets/Security_Documentation_Checklist.html

6.8 Embedded Browser Security Assessments

This section covers assessment procedures for requirements EMB-REQ-1 through EMB-REQ-32, addressing the unique security challenges of embedded browsers (WebView components, browser engines integrated into native applications). These assessments focus on JavaScript bridge security, native API exposure control, content source trust management, and host application boundary protection.

Assessment: EMB-REQ-1 (JavaScript bridge API allowlists)

Reference: EMB-REQ-1 - JavaScript bridges shall implement explicit allowlists of exposed native APIs with per-API access controls

Given: A conformant embedded browser with JavaScript bridge capability (EMB-1 or higher)

Task: JavaScript bridges expose native device APIs to web content, creating critical attack surfaces where unrestricted API exposure enables privilege escalation, data exfiltration, malware installation, and complete

device compromise. Without explicit allowlists, malicious web content can invoke any native API through reflection or dynamic invocation. Explicit API allowlisting with per-API access controls, runtime immutability, reflection blocking, and comprehensive logging prevents bridge-based attacks while enabling controlled native functionality.

Verification:

1. Review the embedded browser's JavaScript bridge configuration and implementation code → Verify that the bridge logs all API access attempts including denied attempts
2. Identify all native APIs exposed to web content through the JavaScript bridge → JavaScript bridge exposes only explicitly allowlisted native APIs
3. Verify that an explicit allowlist mechanism exists (configuration file, API declaration, or programmatic registration) → Configuration files or code clearly declare which APIs are accessible
4. Attempt to call native APIs that are not in the allowlist from web content → Per-API access controls restrict which origins or contexts can call each API
5. Verify that per-API access controls exist (e.g., origin-based restrictions, permission requirements) → Attempts to call non-allowlisted APIs result in errors or exceptions
6. Test that the allowlist cannot be modified by web content at runtime → Web content cannot modify the API allowlist at runtime
7. Attempt to use reflection or dynamic invocation to bypass the allowlist → Reflection or dynamic method invocation does not bypass allowlist enforcement

Pass Criteria: Only explicitly allowlisted APIs are callable from web content AND per-API access controls are enforced AND allowlist cannot be modified by web content

Fail Criteria: Any non-allowlisted API is callable OR access controls are bypassable OR web content can modify the allowlist

Evidence: JavaScript bridge configuration files, implementation code review, test results showing blocked API calls, security logs showing denied access attempts, penetration test reports

References:

- OWASP Mobile Top 10 - M1: Improper Platform Usage: https://owasp.org/www-project-mobile-top-10/
- Android WebView addJavascriptInterface Security: https://developer.android.com/reference/android/webkit/WebView#
- iOS WKWebView Configuration: https://developer.apple.com/documentation/webkit/wkwebviewconfiguration
- Electron Context Bridge: https://www.electronjs.org/docs/latest/api/context-bridge
- CWE-749: Exposed Dangerous Method or Function: https://cwe.mitre.org/data/definitions/749.html

Assessment: EMB-REQ-2 (JavaScript bridge input validation)

Reference: EMB-REQ-2 - All data crossing the JavaScript bridge shall be validated, sanitized, and type-checked on the native side

Given: A conformant embedded browser with JavaScript bridge capability (EMB-1 or higher)

Task: JavaScript bridge parameters originate from untrusted web content and can contain injection payloads, type confusion attacks, buffer overflows, path traversal sequences, or prototype pollution that exploit native code vulnerabilities. Without comprehensive validation, attackers can achieve SQL injection into native databases, command injection for arbitrary code execution, or path traversal to access sensitive files. Native-side validation with type checking, sanitization, range checking, and path canonicalization prevents injection attacks across the trust boundary.

Verification:

1. Identify all native APIs exposed via JavaScript bridge that accept parameters → All bridge API parameters undergo validation on the native side before processing
2. For each API, review the native-side validation and sanitization logic → Type checking rejects parameters of incorrect types

3. Create test cases with malicious inputs: SQL injection strings, path traversal sequences, command injection payloads, excessively long strings, null bytes, Unicode exploits → String sanitization removes or escapes dangerous characters
4. Invoke bridge APIs with malformed data: wrong types, missing parameters, extra parameters, null values, undefined values → Numeric parameters are range-checked
5. Attempt to pass JavaScript objects with prototype pollution characteristics → File paths are validated and canonicalized
6. Test that numeric parameters are range-checked and validated → Malicious inputs trigger validation errors without causing security issues
7. Verify that file paths are canonicalized and validated before use → Validation failures are logged with sufficient detail for security monitoring
8. Confirm that validation failures trigger errors and logging rather than silent failures → No SQL injection, command injection, or path traversal vulnerabilities exist

Pass Criteria: All bridge parameters are validated, sanitized, and type-checked AND malicious inputs are rejected safely AND validation failures are logged

Fail Criteria: Any injection vulnerability exists OR type checking is missing OR validation can be bypassed

Evidence: Code review showing validation logic, penetration test results, fuzzing test results, validation error logs, security assessment reports

References:

- OWASP Input Validation Cheat Sheet: https://cheatsheetseries.owasp.org/cheatsheets/Input_Validation_Cheat_Sheet.h
- CWE-20: Improper Input Validation: https://cwe.mitre.org/data/definitions/20.html
- CWE-89: SQL Injection: https://cwe.mitre.org/data/definitions/89.html
- CWE-78: OS Command Injection: https://cwe.mitre.org/data/definitions/78.html
- Android WebView Security Best Practices: https://developer.android.com/develop/ui/views/layout/webapps/webview#s

Assessment: EMB-REQ-3 (JavaScript bridge logging)

Reference: EMB-REQ-3 - JavaScript bridge communications shall be logged with sufficient detail for security auditing

Given: A conformant embedded browser with JavaScript bridge capability (EMB-1 or higher)

Task: JavaScript bridge logging is critical for detecting and investigating attacks where malicious web content attempts to exploit native APIs through the bridge. Without comprehensive audit logs, security teams cannot detect unauthorized API access, track attack patterns, or investigate security incidents. Proper logging of all bridge communications enables security monitoring systems to identify suspicious patterns such as repeated failed API calls, attempts to access restricted functionality, or unusual parameter values that may indicate injection attacks or reconnaissance activities.

Verification:

1. Configure the embedded browser to enable JavaScript bridge logging → Confirm log retention policies are documented and enforced
2. Invoke various bridge APIs from web content → All JavaScript bridge API invocations are logged
3. Review the generated logs and verify they contain: API name, timestamp, calling origin, parameters (with sensitive data redacted), return values, success/failure status → Logs include sufficient context for security analysis (API, origin, timestamp)
4. Attempt to call blocked or restricted APIs and verify denials are logged → Both successful and failed API calls are logged
5. Generate high-volume bridge traffic and verify logging continues without loss → Sensitive parameters are redacted or hashed in logs
6. Test that sensitive data (passwords, tokens) is redacted from logs → Logs are tamper-resistant with integrity protection

7. Verify logs are stored securely with appropriate access controls → Log retention policies balance security needs and privacy requirements

Pass Criteria: All bridge API calls are logged with sufficient detail AND sensitive data is redacted AND logs have integrity protection

Fail Criteria: API calls are not logged OR insufficient detail is recorded OR sensitive data appears in plaintext

Evidence: Log samples showing bridge API calls, log configuration files, security monitoring dashboards, log retention policy documentation

References:

- OWASP Logging Cheat Sheet: https://cheatsheetseries.owasp.org/cheatsheets/Logging_Cheat_Sheet.html
- NIST SP 800-92: Guide to Computer Security Log Management: https://csrc.nist.gov/publications/detail/sp/800-92/final
- CWE-778: Insufficient Logging: https://cwe.mitre.org/data/definitions/778.html

Assessment: EMB-REQ-4 (Context isolation)

Reference: EMB-REQ-4 - Embedded browsers shall implement context isolation to prevent web content from accessing host application objects

Given: A conformant embedded browser with context isolation capability (EMB-1 or higher)

Task: Context isolation is essential to prevent malicious web content from breaking out of its sandbox and directly accessing host application objects, which could lead to privilege escalation, data theft, or code execution in the host application context. Without proper isolation, attackers can enumerate the JavaScript global namespace to discover and exploit unintended host APIs, access sensitive application data structures, or manipulate host application behavior through prototype pollution. This boundary violation represents one of the most critical security threats in embedded browser architectures.

Verification:

1. Load web content in the embedded browser → Verify that iframes inherit context isolation from the parent
2. From web content JavaScript, attempt to access global objects that might belong to the host: `window.hostApp`, `window.native`, `window.android`, `window.nativeInterface`, etc. → Web content cannot access host application objects directly
3. Attempt to enumerate window properties to discover host-exposed objects → Global namespace pollution from host application is prevented
4. Test if web content can access Node.js globals (if applicable): `process`, `require`, `Buffer`, `global` → Node.js globals (if applicable) are not accessible to web content
5. Verify that the JavaScript bridge API is the only official communication channel → Only explicitly exposed JavaScript bridge APIs are accessible
6. Test that prototype pollution cannot affect host application objects → Window property enumeration does not reveal internal objects
7. Confirm that context isolation is enforced even for trusted content origins → Prototype pollution attacks do not affect host application

Pass Criteria: Web content has no direct access to host objects AND only official bridge APIs are accessible

Fail Criteria: Any host application object is directly accessible from web content OR context isolation can be bypassed

Evidence: Penetration test results showing blocked access attempts, code review of context isolation implementation, test scripts attempting to access host objects, browser console showing security errors

References:

- Electron Context Isolation: https://www.electronjs.org/docs/latest/tutorial/context-isolation
- Chromium Isolated Worlds: https://chromium.googlesource.com/chromium/src/+/master/third_party/blink/renderer/b World
- iOS WKWebView JavaScript Isolation: https://developer.apple.com/documentation/webkit/wkwebview
- Android WebView Isolation: https://developer.android.com/reference/android/webkit/WebView

Assessment: EMB-REQ-5 (User consent for sensitive APIs)

Reference: EMB-REQ-5 - Native APIs exposed via JavaScript bridges shall require explicit user consent for sensitive operations

Given: A conformant embedded browser exposing sensitive APIs via JavaScript bridge (EMB-2 or higher)

Task: User consent for sensitive APIs is critical to prevent malicious web content from silently accessing privacy-sensitive functionality like camera, microphone, location, files, or contacts without the user's knowledge. Without explicit consent requirements, attackers can exploit JavaScript bridge APIs to exfiltrate sensitive data, track user location, or access private information. Proper consent mechanisms protect users from invisible attacks where compromised or malicious web content attempts unauthorized access to device capabilities that could violate user privacy or security expectations.

Verification:

1. Identify all sensitive operations exposed via the JavaScript bridge (e.g., file access, location, camera, contacts, payments) → Confirm that consent state persists across browser sessions according to user choice
2. Attempt to invoke sensitive APIs from web content without prior user consent → Sensitive API calls trigger user consent prompts before execution
3. Verify that a clear, non-dismissible consent dialog appears for each sensitive operation → Consent dialogs clearly identify the origin and permission type
4. Test that consent dialogs clearly identify the requesting origin and the specific permission being requested → Users can grant, deny, or revoke permissions
5. Verify that consent can be granted permanently, temporarily (session-only), or denied → Denied permissions prevent API execution
6. Test that denied permissions are enforced consistently → Consent state is properly persisted and enforced
7. Verify that users can revoke previously granted permissions → No sensitive operations occur without explicit user consent

Pass Criteria: All sensitive API calls require explicit user consent AND consent UI is clear and non-spoofable AND permission state is properly enforced

Fail Criteria: Any sensitive operation occurs without user consent OR consent UI is bypassable OR permissions are not enforced

Evidence: Screenshots of consent dialogs, test results showing blocked operations without consent, user consent audit logs, permission state persistence verification

References:

- Android Runtime Permissions: https://developer.android.com/training/permissions/requesting
- iOS Permission Prompts: https://developer.apple.com/design/human-interface-guidelines/patterns/accessing-private-data
- W3C Permissions API: https://www.w3.org/TR/permissions/

Assessment: EMB-REQ-6 (No system-level API exposure without controls)

Reference: EMB-REQ-6 - Host applications shall not expose system-level APIs (file system, process execution, native module loading) through JavaScript bridges without additional security controls

Given: A conformant embedded browser with JavaScript bridge capability (EMB-1 or higher)

Task: Exposing system-level APIs through JavaScript bridges without strict controls creates critical vulnerabilities that can lead to arbitrary code execution, data exfiltration, or complete system compromise. Malicious web content could exploit unrestricted file system access to read sensitive files, execute arbitrary commands to take control of the host system, or load native modules to escalate privileges. Path traversal attacks could bypass directory restrictions, and unbounded network socket access could enable attackers to pivot to internal systems, making proper security controls on system-level APIs absolutely essential for embedded browser security.

Verification:

1. Review all APIs exposed via the JavaScript bridge → System-level APIs either not exposed or have strict security controls
2. Identify any system-level capabilities: file system access, process/command execution, native module loading, network socket access, registry access → File system access is restricted to specific allowlisted directories
3. For each system-level API, verify additional security controls exist: capability-based permissions, path allowlisting, command allowlisting, sandboxing, user consent → Command execution is restricted to specific allowlisted commands or completely prohibited
4. Attempt to use exposed file system APIs to access sensitive paths (/etc/passwd, C:\Windows\System32, application data directories) → Native module loading is restricted or prohibited
5. Attempt to execute arbitrary commands or load arbitrary native modules → Path traversal attacks are prevented
6. Test that path traversal sequences (../, ..\) are blocked or neutralized → System-level operations require elevated permissions or user consent
7. Verify that system-level operations are logged extensively → All system-level operations are logged with full details
8. Confirm that least-privilege principles are enforced (minimal necessary access) → Security controls cannot be bypassed

Pass Criteria: No unrestricted system-level APIs are exposed AND all system APIs have additional security controls AND controls are enforced effectively

Fail Criteria: Any system-level API is exposed without restrictions OR security controls can be bypassed OR arbitrary file/command access is possible

Evidence: API documentation, code review of exposed APIs, penetration test results, logs showing blocked system access attempts, security architecture documentation

References:

- CWE-78: OS Command Injection: https://cwe.mitre.org/data/definitions/78.html
- CWE-22: Path Traversal: https://cwe.mitre.org/data/definitions/22.html
- OWASP Command Injection: https://owasp.org/www-community/attacks/Command_Injection
- Electron Security Best Practices - Shell Execution: https://www.electronjs.org/docs/latest/tutorial/security#6-do-not-use-shellexecuteshellopen-or-related-methods-with-user-content
- Android File System Security: https://developer.android.com/training/data-storage

Assessment: EMB-REQ-7 (Immutable bridge configuration)

Reference: EMB-REQ-7 - JavaScript bridge configurations shall be immutable after WebView initialization to prevent runtime tampering

Given: A conformant embedded browser with JavaScript bridge capability (EMB-1 or higher)

Task: Immutable bridge configuration prevents runtime tampering attacks where malicious code attempts to dynamically modify the JavaScript bridge to add new attack surfaces, remove security controls, or replace legitimate APIs with malicious implementations. If configuration can be changed after initialization, attackers could exploit race conditions, code injection vulnerabilities, or confused deputy scenarios to inject malicious

APIs, escalate privileges, or disable security restrictions. Configuration immutability ensures that the security boundary established at initialization time cannot be undermined during runtime.

Verification:

1. Initialize the embedded browser with a specific JavaScript bridge configuration → Verify that attempting to modify configuration triggers security logging
2. After initialization, attempt to modify the bridge configuration from host application code → Bridge configuration is set during initialization and cannot be changed afterward
3. From web content, attempt to add new APIs to the bridge dynamically → Host application cannot add or remove bridge APIs after initialization
4. Test if existing bridge APIs can be removed or replaced at runtime → Web content cannot influence bridge configuration
5. Verify that configuration changes after initialization either fail or require browser restart → Attempts to modify configuration fail with appropriate errors
6. Test that web content cannot influence bridge configuration through any means → Configuration immutability is enforced at the code level
7. Review code to confirm configuration is set during initialization phase only → Any attempted configuration changes are logged as security events

Pass Criteria: Bridge configuration is immutable after initialization AND modification attempts fail AND attempts are logged

Fail Criteria: Configuration can be modified after initialization OR web content can influence configuration

Evidence: Code review showing configuration immutability enforcement, test results showing failed modification attempts, security logs showing configuration change attempts, API documentation

References:

- Android WebView Configuration Best Practices: https://developer.android.com/reference/android/webkit/WebSettings
- iOS WKWebView Configuration Immutability: https://developer.apple.com/documentation/webkit/wkwebviewconfigura
- Electron Security - Immutable Configuration: https://www.electronjs.org/docs/latest/tutorial/security
- CWE-732: Incorrect Permission Assignment: https://cwe.mitre.org/data/definitions/732.html

Assessment: EMB-REQ-8 (Host credential protection)

Reference: EMB-REQ-8 - Embedded browsers shall prevent web content from accessing host application credentials, tokens, or cryptographic keys

Given: A conformant embedded browser with context isolation (EMB-1 or higher)

Task: Protecting host credentials from web content access is essential to prevent credential theft attacks where malicious web content attempts to steal authentication tokens, API keys, or cryptographic keys stored by the host application. If web content can access host credentials through JavaScript bridge APIs, shared storage, or side channels, attackers can impersonate the host application, access protected resources, or decrypt sensitive data. This credential isolation requirement prevents scenarios where a compromised web page loaded in the embedded browser could steal the host's authentication credentials and use them for unauthorized access.

Verification:

1. Store test credentials/tokens in the host application's secure storage (keychain, keystore, credential manager) → Test that cryptographic operations using host keys cannot be triggered arbitrarily by web content
2. Load web content in the embedded browser → Host application credentials are completely isolated from web content
3. Attempt to access host credentials through various methods: JavaScript bridge APIs, localStorage inspection, cookie inspection, memory inspection → No JavaScript bridge API exposes credentials directly or indirectly

4. Verify that credentials stored by the host are isolated from web content storage → Web content storage (localStorage, cookies, IndexedDB) is separate from host credential storage
5. Test that shared storage mechanisms (if any) do not leak credentials → Shared storage mechanisms do not leak credentials
6. Attempt to extract credentials by triggering host application features that use them → Authentication tokens are not exposed in error messages or debug output
7. Verify that authentication tokens are not exposed in error messages or logs accessible to web content → Host cryptographic keys cannot be accessed or used arbitrarily by web content

Pass Criteria: Web content has no access to host credentials AND credential storage is completely isolated AND cryptographic keys are protected

Fail Criteria: Any credential or token is accessible to web content OR cryptographic keys can be misused

Evidence: Penetration test results, code review of credential storage isolation, test attempts to access credentials, memory dumps showing isolation, security audit reports

References:

- Android Keystore System: https://developer.android.com/training/articles/keystore
- iOS Keychain Services: https://developer.apple.com/documentation/security/keychain_services
- CWE-522: Insufficiently Protected Credentials: https://cwe.mitre.org/data/definitions/522.html

Assessment: EMB-REQ-9 (JavaScript bridge security review)

Reference: EMB-REQ-9 - All JavaScript bridge implementations shall be reviewed for injection vulnerabilities before production deployment

Given: A conformant embedded browser with JavaScript bridge capability (EMB-1 or higher)

Task: Security reviews for JavaScript bridge implementations are critical because bridges represent the most dangerous attack surface in embedded browsers, where improper input validation can lead to injection attacks, privilege escalation, or arbitrary code execution. Without formal security reviews, subtle vulnerabilities like SQL injection through bridge parameters, command injection in system-level APIs, or XSS through improperly sanitized return values can go undetected. Comprehensive security assessments by qualified professionals ensure that all bridge APIs are properly validated, sanitized, and protected against common attack vectors before they can be exploited in production.

Verification:

1. Obtain documentation of the security review process for JavaScript bridge implementations → Verify that third-party security assessments were conducted (if applicable)
2. Verify that a formal security code review was conducted before production deployment → Documented security review process exists for JavaScript bridge code
3. Review security assessment reports for the JavaScript bridge implementation → Security code reviews were conducted by qualified security professionals
4. Verify that common injection vulnerabilities were tested: SQL injection, command injection, path traversal, XSS, code injection → Common injection vulnerability types were specifically tested
5. Confirm that automated security scanning tools were used (static analysis, dynamic analysis) → Automated security scanning tools were applied
6. Verify that findings from security reviews were remediated before deployment → Security findings were tracked and remediated
7. Confirm that security reviews are repeated after significant bridge changes → Re-reviews occur after significant code changes

Pass Criteria: Formal security review was conducted AND injection vulnerabilities were specifically tested AND findings were remediated

Fail Criteria: No security review was conducted OR injection vulnerabilities were not tested OR findings were not remediated

Evidence: Security review reports, code review checklists, penetration test reports, static analysis tool outputs, vulnerability remediation tracking records, security sign-off documentation

References:

- OWASP Code Review Guide: https://owasp.org/www-project-code-review-guide/
- CWE Top 25 Most Dangerous Software Weaknesses: https://cwe.mitre.org/top25/archive/2023/2023_top25_list.html
- NIST SP 800-53 - Security Assessment: https://csrc.nist.gov/publications/detail/sp/800-53/rev-5/final
- OWASP Mobile Application Security Verification Standard: https://github.com/OWASP/owasp-masvs

Assessment: EMB-REQ-10 (Bridge API rate limiting)

Reference: EMB-REQ-10 - Host applications shall implement rate limiting on JavaScript bridge API calls to prevent abuse

Given: A conformant embedded browser with JavaScript bridge capability (EMB-2 or higher)

Task: Rate limiting on JavaScript bridge APIs prevents abuse scenarios where malicious web content floods the bridge with excessive API calls to perform denial-of-service attacks, exhaust system resources, or brute-force security controls. Without rate limits, attackers can rapidly probe for vulnerabilities, overwhelm logging systems to hide their tracks, or cause performance degradation that impacts legitimate users. Proper rate limiting also prevents reconnaissance attacks where malicious scripts systematically enumerate all available APIs or attempt to bypass security controls through repeated high-frequency attempts.

Verification:

1. Identify rate limits for each JavaScript bridge API (requests per second, burst limits) → Rate limits are defined and documented for all bridge APIs
2. Create test scripts that invoke bridge APIs at high frequency → Excessive API calls are rejected or throttled
3. Verify that rate limits are enforced: excessive calls are rejected or throttled → Rate limit violations trigger security logging and alerts
4. Test that rate limit violations trigger logging and security alerts → Rate limits are enforced per-origin or per-context
5. Verify that rate limits are per-origin (different origins have independent quotas) → Rate limits reset on appropriate time boundaries
6. Test that rate limits reset appropriately (per second, per minute, per session) → Multiple contexts cannot bypass rate limits
7. Attempt to bypass rate limits by using multiple execution contexts (iframes, workers) → Legitimate high-frequency use cases are accommodated
8. Verify that legitimate use cases are not blocked by rate limits → Rate limit parameters are configurable by host application

Pass Criteria: Rate limits are enforced on all bridge APIs AND violations are logged AND limits cannot be bypassed

Fail Criteria: No rate limiting exists OR rate limits are bypassable OR legitimate use is blocked

Evidence: Rate limit configuration documentation, test results showing throttling behavior, security logs showing rate limit violations, performance test results

References:

- OWASP API Security - Rate Limiting: https://owasp.org/API-Security/editions/2023/en/0xa4-unrestricted-resource-consumption/
- CWE-770: Allocation of Resources Without Limits: https://cwe.mitre.org/data/definitions/770.html
- NIST SP 800-95: Guide to Secure Web Services: https://csrc.nist.gov/publications/detail/sp/800-95/final
- Rate Limiting Best Practices: https://cloud.google.com/architecture/rate-limiting-strategies-techniques

Assessment: EMB-REQ-11 (Granular capability-based permissions)

Reference: EMB-REQ-11 - JavaScript bridges shall support granular capability-based permissions rather than all-or-nothing access

Given: A conformant embedded browser with advanced bridge capabilities (EMB-2 or higher)

Task: Granular capability-based permissions prevent over-privileged access scenarios where granting access to one API inadvertently exposes all bridge functionality to potentially malicious web content. All-or-nothing permission models force developers to choose between denying all access or granting excessive privileges, creating security risks. Fine-grained permissions enable the principle of least privilege, allowing each origin to access only the specific APIs it needs while preventing lateral movement attacks where compromise of one feature leads to unauthorized access to unrelated sensitive functionality.

Verification:

1. Review the permission model for the JavaScript bridge → Permission system supports individual API-level grants
2. Verify that permissions can be granted at individual API level rather than all-or-nothing → Different origins can have different permission sets
3. Test different permission configurations: grant some APIs but deny others to the same origin → Permissions can be scoped by time, usage count, or other constraints
4. Verify that capability tokens or permissions can be scoped by origin, time, or usage count → Temporary permissions expire as configured
5. Test that temporary permissions expire correctly → Permission grants and revocations are logged
6. Verify that permission grants are logged with full context → Users/administrators have control over granular permissions
7. Test that users or administrators can configure granular permissions → Default configuration follows least-privilege principle
8. Verify that the principle of least privilege is enforced by default → Permission model is clearly documented

Pass Criteria: Granular per-API permissions are supported AND permissions are properly scoped AND least-privilege is enforced by default

Fail Criteria: Only all-or-nothing permissions exist OR no permission scoping is available OR overly permissive defaults

Evidence: Permission configuration files, permission model documentation, test results showing granular control, user/admin permission management interface screenshots, audit logs

References:

- Capability-Based Security: https://en.wikipedia.org/wiki/Capability-based_security
- Principle of Least Privilege (NIST): https://csrc.nist.gov/glossary/term/least_privilege
- W3C Permissions API: https://www.w3.org/TR/permissions/
- Android Permission Model: https://developer.android.com/guide/topics/permissions/overview

Assessment: EMB-REQ-12 (Storage isolation from host)

Reference: EMB-REQ-12 - Embedded browsers shall isolate storage (cookies, localStorage, IndexedDB) from the host application's native storage

Given: A conformant embedded browser (EMB-0 or higher)

Task: Storage isolation between embedded browsers and host applications prevents data leakage attacks where malicious web content attempts to access sensitive application data stored in native storage mechanisms. Without proper isolation, compromised web content could read authentication tokens from host storage, modify application configuration data, or exfiltrate user information. Storage namespace collisions could also enable web content to poison host application data, leading to privilege escalation or application malfunction.

Proper isolation ensures that the web security model and native application security model remain separate and cannot be used to undermine each other.

Verification:

1. Store data in the host application's native storage (SharedPreferences, UserDefaults, native database) → Test that malicious web content cannot pollute host storage namespace
2. Load web content and attempt to access host storage through web APIs (localStorage, IndexedDB, cookies) → Web content storage (localStorage, IndexedDB, cookies) is isolated from host native storage
3. Verify that web content cannot read native host storage → Host native storage is not accessible from web APIs
4. Store data in web content storage and attempt to access it from host application → Web storage is not directly accessible from host code (without explicit bridge)
5. Verify that host application code cannot directly access web storage without explicit bridge APIs → Multiple embedded browser instances have independent storage
6. Test that different embedded browser instances have isolated storage → Storage isolation is maintained even for same-origin content in different instances
7. Verify that clearing host application data does not affect web storage (unless explicitly configured) → Storage clearing operations are properly scoped

Pass Criteria: Complete storage isolation between web content and host application AND no cross-contamination is possible

Fail Criteria: Web content can access host storage OR host can access web storage without proper APIs OR storage namespaces collide

Evidence: Code review of storage isolation implementation, test results showing blocked access attempts, storage dump comparisons, penetration test results

References:

- Android WebView Data Storage: https://developer.android.com/reference/android/webkit/WebView#getDataDir()
- Chromium Embedded Framework Storage Isolation: https://bitbucket.org/chromiumembedded/cef/wiki/GeneralUsage
- Web Storage API: https://html.spec.whatwg.org/multipage/webstorage.html

Assessment: EMB-REQ-13 (CSP enforcement for embedded content)

Reference: EMB-REQ-13 - Host applications shall implement Content Security Policy (CSP) enforcement for all embedded web content

Given: A conformant embedded browser (EMB-1 or higher)

Task: Content Security Policy enforcement for embedded web content provides defense-in-depth protection against XSS attacks, malicious script injection, and unauthorized resource loading that could be used to exfiltrate data or establish command-and-control channels. Without CSP, compromised or malicious web content can inject inline scripts, load external malicious resources, or use eval() to execute attacker-controlled code. Host-configured CSP ensures that even if web content is compromised, the attack surface is constrained by preventing dangerous JavaScript patterns and restricting where content can be loaded from.

Verification:

1. Configure the embedded browser with a restrictive CSP policy → CSP policy is configurable by host application
2. Load web content and verify CSP is enforced: inline scripts blocked, external resource loading restricted → CSP is enforced for all embedded web content
3. Test that CSP violations trigger browser console errors and violation reports → Inline scripts, eval, and other dangerous features are blocked per policy
4. Verify that CSP cannot be bypassed by web content → CSP violations generate console errors and reports

5. Test that meta tag CSPs are respected but cannot weaken configured policy → Web content cannot bypass or weaken CSP policy
6. Verify that CSP applies to all content including iframes → CSP applies to all nested contexts (iframes)
7. Test CSP report-uri or report-to functionality → Violation reports are sent to configured endpoints
8. Verify that host application can configure CSP policy programmatically → Default CSP is restrictive following security best practices

Pass Criteria: Host-configured CSP is enforced for all content AND violations are reported AND policy cannot be bypassed

Fail Criteria: CSP is not enforced OR can be bypassed OR violations are not reported

Evidence: CSP configuration code, browser console showing CSP violations, violation report logs, test results demonstrating CSP enforcement, penetration test results

References:

- Content Security Policy Level 3: https://www.w3.org/TR/CSP3/
- CSP Best Practices (OWASP): https://cheatsheetseries.owasp.org/cheatsheets/Content_Security_Policy_Cheat_Sheet.h
- Android WebView CSP: https://developer.android.com/reference/android/webkit/WebSettings#setMixedContentMode(i
- iOS WKWebView CSP Support: https://webkit.org/blog/7929/content-security-policy-for-webkit/

Assessment: EMB-REQ-14 (Encrypted cross-process bridge)

Reference: EMB-REQ-14 - JavaScript bridge communications shall be encrypted when crossing process boundaries

Given: A conformant embedded browser with multi-process architecture (EMB-2 or higher)

Task: Encrypted cross-process bridge communications protect against local privilege escalation attacks where a malicious process on the same system attempts to intercept or tamper with JavaScript bridge messages passing between the browser process and host application process. Without encryption, attackers with local access could use debugging tools, IPC monitoring utilities, or memory inspection to capture sensitive bridge data including authentication tokens, API parameters, or user data. Message integrity protection prevents tampering attacks, while anti-replay mechanisms prevent attackers from capturing and reusing legitimate bridge messages to perform unauthorized operations.

Verification:

1. Identify whether the embedded browser uses a multi-process architecture → Verify that encryption keys are securely managed and rotated
2. Verify that JavaScript bridge messages cross process boundaries → Multi-process architecture uses encrypted IPC for bridge messages
3. Analyze inter-process communication (IPC) mechanisms used for bridge messages → Bridge messages are not observable in plaintext during IPC
4. Verify that IPC channels are encrypted or use secure transport → Message integrity is cryptographically protected
5. Attempt to intercept bridge messages using debugging tools or IPC monitoring → Replay attacks are prevented through anti-replay mechanisms
6. Verify that message integrity is protected (authentication, MACs) → Encryption keys are properly generated and managed
7. Test that replay attacks are prevented (nonces, sequence numbers) → IPC interception does not reveal sensitive bridge data

Pass Criteria: Bridge IPC is encrypted AND integrity-protected AND replay-protected when crossing process boundaries

Fail Criteria: Plaintext bridge messages observable in IPC OR no integrity protection OR replay attacks possible

Evidence: Architecture documentation showing multi-process design, IPC traces showing encrypted messages, code review of encryption implementation, penetration test results, cryptographic audit reports

References:

- Chromium Mojo IPC Security: https://chromium.googlesource.com/chromium/src/+/master/mojo/README.md
- Electron IPC Security: https://www.electronjs.org/docs/latest/tutorial/ipc
- NIST SP 800-52: TLS Guidelines: https://csrc.nist.gov/publications/detail/sp/800-52/rev-2/final
- CWE-319: Cleartext Transmission of Sensitive Information: https://cwe.mitre.org/data/definitions/319.html

Assessment: EMB-REQ-15 (Native UI overlay prevention)

Reference: EMB-REQ-15 - Embedded browsers shall prevent web content from triggering native UI overlays or permission prompts that could mislead users

Given: A conformant embedded browser (EMB-1 or higher)

Task: Native UI overlay prevention protects users from social engineering attacks where malicious web content attempts to display fake permission prompts, system dialogs, or security warnings that appear to originate from the operating system or host application. Without proper controls, attackers can create convincing spoofed UI elements to trick users into granting permissions, entering credentials, or taking dangerous actions. Preventing web content from triggering misleading native UI and ensuring clear origin identification helps users make informed trust decisions and prevents clickjacking attacks that overlay malicious content on top of legitimate security prompts.

Verification:

1. Attempt to trigger native UI elements from web content: system dialogs, permission prompts, notifications → Web content cannot trigger misleading native UI elements
2. Verify that all UI elements clearly identify their source (host app vs. web content) → All UI clearly indicates whether it originates from web content or host/OS
3. Test that web content cannot create UI elements that appear to be from the OS or host app → Permission prompts show the requesting origin clearly
4. Attempt clickjacking attacks using iframes and native UI → Clickjacking attempts are prevented by UI design
5. Verify that permission prompts clearly show the requesting origin → Rapid permission prompt abuse is prevented
6. Test that web content cannot programmatically trigger permission prompts in rapid succession → Native security indicators cannot be hidden or spoofed
7. Verify that native UI elements cannot be overlaid or obscured by web content → Fullscreen mode maintains critical security UI
8. Test that fullscreen mode does not hide native UI security indicators → Users can distinguish web content UI from native UI

Pass Criteria: Web content cannot trigger misleading native UI AND all prompts clearly show origin AND security indicators are protected

Fail Criteria: Misleading UI can be triggered OR origin is not clear OR security indicators can be hidden

Evidence: Screenshots of UI elements showing origin indicators, test results of UI spoofing attempts, clickjacking test results, user study results on UI clarity, security audit reports

References:

- OWASP Clickjacking Defense: https://cheatsheetseries.owasp.org/cheatsheets/Clickjacking_Defense_Cheat_Sheet.html
- Android UI Security: https://developer.android.com/topic/security/risks/tapjacking
- CWE-1021: Improper Restriction of Rendered UI Layers: https://cwe.mitre.org/data/definitions/1021.html

Assessment: EMB-REQ-16 (API surface allowlisting over denylisting)

Reference: EMB-REQ-16 - Host applications shall implement allowlists for JavaScript bridge API surface rather than denylists

Given: A conformant embedded browser with JavaScript bridge (EMB-1 or higher)

Task: Allowlist-based API surface control is critical for preventing unauthorized access to internal APIs that were never intended for web content exposure. Denylist approaches are inherently insecure because they require developers to anticipate and block every dangerous API, while inevitably missing newly added functionality or overlooking indirect access paths. With allowlisting, only explicitly registered APIs are accessible, ensuring that refactoring, new feature development, or internal API additions cannot accidentally expand the attack surface. This prevents attackers from discovering and exploiting unintended API endpoints through reflection, code analysis, or brute-force enumeration.

Verification:

1. Review the JavaScript bridge implementation to determine if it uses allowlist or denylist approach → JavaScript bridge uses allowlist-based access control
2. Verify that only explicitly registered APIs are accessible (allowlist approach) → Only explicitly allowlisted APIs are accessible from web content
3. Attempt to discover and call APIs that are not in the allowlist → APIs not in allowlist are not accessible regardless of denylist status
4. Test that new APIs require explicit registration before becoming accessible → New APIs require explicit allowlist registration
5. Verify that the absence of an API from a denylist does not make it accessible → Denylists (if present) are used only as an additional layer, not primary control
6. Review code to confirm allowlist-based access control logic → Reflection/introspection cannot discover non-allowlisted APIs
7. Test that reflection or introspection cannot enumerate unlisted APIs → Code review confirms allowlist-first architecture
8. Verify that the allowlist is the sole source of truth for API accessibility → Security documentation clearly describes allowlist approach

Pass Criteria: Allowlist-based access control is implemented AND only allowlisted APIs are accessible AND denylisting is not the primary control

Fail Criteria: Denylist-based access control is used as primary mechanism OR non-allowlisted APIs are accessible

Evidence: Code review showing allowlist implementation, security architecture documentation, test results showing blocked non-allowlisted API access, API registration procedures

References:

- OWASP Access Control: https://cheatsheetseries.owasp.org/cheatsheets/Access_Control_Cheat_Sheet.html
- Allowlist vs Denylist Best Practices: https://owasp.org/www-community/vulnerabilities/Improper_Data_Validation
- CWE-183: Permissive List of Allowed Inputs: https://cwe.mitre.org/data/definitions/183.html
- Electron Context Bridge Allowlisting: https://www.electronjs.org/docs/latest/api/context-bridge

Assessment: EMB-REQ-17 (Certificate validation for remote content)

Reference: EMB-REQ-17 - Embedded browsers shall validate SSL/TLS certificates for all remote content sources with certificate pinning for critical origins

Given: A conformant embedded browser loading remote content (EMB-1 or higher)

Task: Certificate validation and pinning for remote content protects against man-in-the-middle attacks where attackers intercept network traffic to inject malicious content into the embedded browser. Without proper certificate validation, attackers on compromised networks can present fraudulent certificates to serve

malicious JavaScript that exploits bridge APIs or steals sensitive data. Certificate pinning for critical origins provides additional protection against certificate authority compromises and ensures that only the expected server certificates are trusted, preventing sophisticated attacks using rogue or compromised CAs to issue valid but malicious certificates for trusted domains.

Verification:

1. Configure the embedded browser to load content from HTTPS origins → Verify that pin backups exist to prevent lockout
2. Attempt to load content from a server with an invalid certificate (expired, self-signed, wrong hostname) → All HTTPS content undergoes full certificate validation
3. Verify that certificate validation errors block content loading → Invalid certificates block content loading
4. Test that user/host cannot bypass certificate errors for critical origins → Certificate errors generate clear error messages and logs
5. Configure certificate pinning for critical origins used by the application → Certificate pinning is enforced for configured critical origins
6. Attempt to use a valid but non-pinned certificate for a pinned origin → Pinning violations prevent content loading and trigger security events
7. Verify that pinning violations block content loading and trigger alerts → Backup pins exist to allow pin rotation without lockout
8. Test that certificate transparency information is validated → Certificate Transparency validation is performed

Pass Criteria: Full certificate validation is enforced AND certificate pinning works for critical origins AND no bypasses exist for pinned origins

Fail Criteria: Certificate validation can be bypassed OR pinning is not enforced OR no backup pins exist

Evidence: Certificate validation test results, pinning configuration, test results with invalid certificates, security logs showing validation failures, Certificate Transparency logs

References:

- RFC 6797: HTTP Strict Transport Security (HSTS): https://datatracker.ietf.org/doc/html/rfc6797
- OWASP Certificate Pinning: https://owasp.org/www-community/controls/Certificate_and_Public_Key_Pinning
- Android Network Security Config: https://developer.android.com/training/articles/security-config
- iOS App Transport Security: https://developer.apple.com/documentation/security/preventing_insecure_network_conne
- RFC 6962: Certificate Transparency: https://datatracker.ietf.org/doc/html/rfc6962

Assessment: EMB-REQ-18 (Trusted origin allowlisting)

Reference: EMB-REQ-18 - Host applications shall implement allowlists of trusted content origins and reject content from unapproved sources

Given: A conformant embedded browser (EMB-1 or higher)

Task: Trusted origin allowlisting prevents the embedded browser from loading malicious or untrusted content that could exploit JavaScript bridge APIs or compromise user data. Without origin restrictions, the browser could be tricked into loading attacker-controlled content through phishing links, open redirects, or DNS rebinding attacks. Enforcing a strict allowlist ensures that only vetted, trusted origins can execute in the privileged embedded browser context, preventing attackers from injecting malicious web pages that would have access to sensitive bridge functionality or user information intended only for legitimate application content.

Verification:

1. Review the host application's origin allowlist configuration → Origin allowlist is configured and documented
2. Attempt to load content from allowlisted origins and verify it loads successfully → Only allowlisted origins can load in the embedded browser

3. Attempt to load content from non-allowlisted origins and verify it is blocked → Non-allowlisted origins are blocked with appropriate error messages
4. Test redirect chains that exit allowlisted domains and verify they are blocked → Redirect chains exiting allowlisted domains are blocked
5. Verify that allowlist enforcement applies to main frame, iframes, and XHR/fetch requests → Allowlist applies to all resource types and contexts
6. Test that URL parameters or fragments cannot be used to bypass allowlist → URL manipulation cannot bypass allowlist enforcement
7. Verify that allowlist configuration is immutable by web content → Allowlist violations generate security events and logs
8. Test that allowlist violations trigger logging and security alerts → Users/administrators can configure the allowlist

Pass Criteria: Origin allowlist is enforced for all content AND violations are blocked and logged AND allowlist cannot be bypassed

Fail Criteria: Non-allowlisted origins can load OR allowlist can be bypassed OR violations not logged

Evidence: Origin allowlist configuration, test results with blocked non-allowlisted origins, security logs showing violations, redirect chain test results, penetration test reports

References:

- OWASP Content Security Policy: https://owasp.org/www-community/controls/Content_Security_Policy
- Android WebView URL Filtering: https://developer.android.com/reference/android/webkit/WebViewClient#shouldOver
- iOS WKNavigationDelegate: https://developer.apple.com/documentation/webkit/wknavigationdelegate
- CSP frame-ancestors directive: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Security-Policy/frame-ancestors

Assessment: EMB-REQ-19 (Subresource Integrity for external scripts)

Reference: EMB-REQ-19 - Embedded browsers shall enforce Subresource Integrity (SRI) for all external scripts loaded by trusted content

Given: A conformant embedded browser loading trusted content with external dependencies (EMB-2 or higher)

Task: Subresource Integrity enforcement protects against supply chain attacks where compromised CDNs or third-party script providers serve malicious code to embedded browsers. Without SRI, attackers who compromise external script sources can inject malicious JavaScript that would execute with full access to JavaScript bridge APIs and user data. SRI ensures that external scripts match their expected cryptographic hashes, preventing execution of modified or substituted content even if the hosting server is compromised, providing critical defense-in-depth protection for embedded browsers that load resources from external origins.

Verification:

1. Configure trusted content pages to load external scripts from CDNs → Test that the host application can enforce SRI policy via CSP require-sri-for directive
2. Add SRI integrity attributes to script tags: `<script src="..." integrity="sha384-..." crossorigin="anonymous">` → SRI integrity checking is enforced for external scripts
3. Verify that scripts with correct integrity hashes load successfully → Scripts with correct integrity hashes load and execute
4. Modify the external script content and verify that integrity validation fails and blocks execution → Scripts with incorrect or missing hashes are blocked per policy
5. Attempt to load scripts without integrity attributes from external origins and verify policy enforcement → SRI violations generate console errors and logs
6. Test that integrity validation applies to CSS, scripts, and other subresources → SRI applies to scripts, stylesheets, and other subresources

7. Verify that SRI failures trigger console errors and CSP violation reports → Host application can mandate SRI via CSP policy

Pass Criteria: SRI is enforced for external scripts AND integrity violations block resource loading AND policy can be mandated by host

Fail Criteria: SRI is not enforced OR can be bypassed OR weak hash algorithms are accepted

Evidence: HTML source showing SRI attributes, browser console showing integrity check results, test results with modified scripts, CSP headers mandating SRI, security logs

References:

- Subresource Integrity specification: https://www.w3.org/TR/SRI/
- SRI Best Practices (MDN): https://developer.mozilla.org/en-US/docs/Web/Security/Subresource_Integrity
- CSP require-sri-for: https://www.w3.org/TR/CSP3/#directive-require-sri-for
- OWASP Secure Headers Project: https://owasp.org/www-project-secure-headers/

Assessment: EMB-REQ-20 (Certificate pinning with backup pins)

Reference: EMB-REQ-20 - Certificate pinning configurations shall include backup pins and pin rotation mechanisms to prevent lockout

Given: A conformant embedded browser with certificate pinning (EMB-2 or higher)

Task: Backup pins and pin rotation mechanisms are essential to prevent application lockout scenarios where certificate rotation or CA changes leave deployed applications unable to connect to pinned origins. Without backup pins, a single certificate expiration or key compromise could render all deployed applications non-functional until users install updates. Proper pin management with backup pins and documented rotation procedures ensures that certificate pinning provides strong security against MITM attacks while maintaining operational resilience, preventing scenarios where security controls inadvertently cause widespread application failures that force emergency unpinning and reduce overall security posture.

Verification:

1. Review certificate pinning configuration for all pinned origins → Each pinned origin has at least one backup pin configured
2. Verify that at least one backup pin exists for each pinned origin → Primary and backup pins are both enforced correctly
3. Test that the primary pin is enforced correctly → Certificate rotation can occur using backup pins without lockout
4. Simulate primary certificate rotation and verify backup pin is accepted → Pin rotation procedures are documented and tested
5. Verify that pin rotation procedures are documented → Pin expiration/rotation schedules are reasonable
6. Test that pin expiration dates are configured appropriately (not too distant future) → Pin updates can be delivered via app updates or dynamic configuration
7. Verify that pin updates can be delivered to deployed applications → Total pin failure blocks connections rather than falling back to no pinning
8. Test fallback behavior when all pins fail (should block, not fall back to unpinned) → Pins include Subject Public Key Info (SPKI) hashes, not just certificates

Pass Criteria: Backup pins exist for all pinned origins AND pin rotation is supported AND lockout scenarios are prevented

Fail Criteria: No backup pins exist OR pin rotation is not supported OR lockout is possible OR insecure fallback behavior

Evidence: Certificate pinning configuration files, pin backup documentation, pin rotation test results, pin expiration monitoring, security architecture documentation

References:

- OWASP Certificate Pinning Cheat Sheet: https://cheatsheetseries.owasp.org/cheatsheets/Pinning_Cheat_Sheet.html
- RFC 7469: Public Key Pinning Extension for HTTP: https://datatracker.ietf.org/doc/html/rfc7469
- Android Network Security Config - Pin Backup: https://developer.android.com/training/articles/security-config#CertificatePinning
- iOS Certificate Pinning Best Practices: https://developer.apple.com/news/?id=g9ejcf8y

Assessment: EMB-REQ-21 (Mixed content prevention)

Reference: EMB-REQ-21 - Embedded browsers shall prevent mixed content (HTTPS pages loading HTTP resources) in all configurations

Given: A conformant embedded browser (EMB-1 or higher)

Task: Mixed content prevention protects against downgrade attacks where attackers intercept insecure HTTP resources loaded by HTTPS pages to inject malicious code or steal sensitive data. Even a single HTTP script loaded by an HTTPS page undermines the entire security model, allowing network attackers to execute arbitrary JavaScript with full access to the page's privileges and JavaScript bridge APIs. Blocking mixed content ensures that the security guarantees of HTTPS connections cannot be bypassed through insecure subresources, preventing attackers from using unencrypted channels to compromise otherwise secure embedded browser sessions.

Verification:

1. Load an HTTPS page in the embedded browser → Verify that CSP upgrade-insecure-requests directive is supported
2. Attempt to load HTTP subresources from the HTTPS page: images, scripts, stylesheets, XHR, fetch → Active mixed content is always blocked
3. Verify that passive mixed content (images, media) is blocked or triggers warnings → Passive mixed content is blocked or triggers prominent warnings
4. Verify that active mixed content (scripts, stylesheets, iframes) is always blocked → Mixed content errors appear in browser console with details
5. Test that mixed content blocking cannot be disabled by web content → Web content cannot disable mixed content protection
6. Verify that host application cannot weaken mixed content protection → Host application cannot weaken mixed content blocking
7. Test that browser console shows clear mixed content errors → upgrade-insecure-requests CSP directive automatically upgrades HTTP to HTTPS

Pass Criteria: Active mixed content is always blocked AND passive mixed content protection is enforced AND cannot be disabled

Fail Criteria: Mixed content loads successfully OR protection can be disabled OR insufficient warnings

Evidence: Browser console showing mixed content errors, test results with blocked HTTP resources, CSP header analysis, security configuration review, network traffic captures

References:

- W3C Mixed Content specification: https://www.w3.org/TR/mixed-content/
- Mixed Content (MDN): https://developer.mozilla.org/en-US/docs/Web/Security/Mixed_content
- CSP upgrade-insecure-requests: https://www.w3.org/TR/upgrade-insecure-requests/
- Chromium Mixed Content Policy: https://www.chromium.org/Home/chromium-security/deprecating-powerful-features-on-insecure-origins

Assessment: EMB-REQ-22 (Trust decision logging)

Reference: EMB-REQ-22 - Trust decisions shall be logged with sufficient detail to detect trust boundary violations

Given: A conformant embedded browser with trust management (EMB-1 or higher)

Task: Trust decision logging is critical for detecting and investigating attacks that attempt to cross trust boundaries in embedded browsers, such as untrusted origins trying to access privileged JavaScript bridge APIs or certificate validation bypasses. Without comprehensive logging of trust decisions, security teams cannot identify patterns of attack attempts, investigate security incidents, or detect compromised trusted origins that begin exhibiting suspicious behavior. Detailed trust logs enable security monitoring systems to flag anomalies like repeated origin allowlist violations, certificate pinning failures, or unusual patterns of bridge API access that may indicate reconnaissance or active exploitation attempts.

Verification:

1. Configure logging for trust-related decisions in the embedded browser → Verify log retention policies for security-relevant events
2. Load content from trusted and untrusted origins → All trust decisions are logged with sufficient detail
3. Review logs and verify they contain: origin, timestamp, trust decision outcome, reason for decision → Logs include origin, timestamp, decision, and rationale
4. Attempt trust boundary violations (untrusted origin accessing privileged APIs) → Trust boundary violations are logged with full context
5. Verify that violations are logged with full context → Certificate validation results are logged
6. Test that certificate validation decisions are logged → Origin allowlist enforcement is logged
7. Verify that origin allowlist enforcement is logged → Logs are protected from tampering
8. Test that logs are stored securely and cannot be tampered with by web content → Log retention balances security needs and privacy

Pass Criteria: All trust decisions are logged with sufficient detail AND violations are logged AND logs are tamper-protected

Fail Criteria: Trust decisions are not logged OR insufficient detail OR logs can be tampered with

Evidence: Log samples showing trust decisions, log configuration documentation, test results showing violation logging, log integrity verification, log retention policy documentation

References:

- OWASP Logging Cheat Sheet: https://cheatsheetseries.owasp.org/cheatsheets/Logging_Cheat_Sheet.html
- NIST SP 800-92: Guide to Computer Security Log Management: https://csrc.nist.gov/publications/detail/sp/800-92/final
- CWE-778: Insufficient Logging: https://cwe.mitre.org/data/definitions/778.html

Assessment: EMB-REQ-23 (Cryptographic signature verification for local content)

Reference: EMB-REQ-23 - Embedded browsers loading local/bundled content shall verify cryptographic signatures before execution

Given: A conformant embedded browser loading bundled/local content (EMB-2 or higher)

Task: Cryptographic signature verification for local content protects against tampering attacks where attackers with file system access attempt to modify bundled HTML, JavaScript, or CSS files to inject malicious code. Without signature verification, local privilege escalation or compromised backup/restore processes could allow attackers to silently modify embedded browser content that executes with full JavaScript bridge API access. Digital signatures ensure content integrity from build time through deployment, detecting any unauthorized modifications and preventing execution of tampered local resources that could otherwise compromise the entire host application through bridge exploitation.

Verification:

1. Identify local/bundled content loaded by the embedded browser (HTML, JS, CSS files) → All local/bundled content has cryptographic signatures
2. Verify that all bundled content is cryptographically signed → Signature verification occurs before content execution

3. Review signature verification implementation in the browser initialization code → Modified content fails signature verification
4. Attempt to modify bundled content and verify that signature verification fails → Signature verification failures block content loading
5. Test that signature verification failures prevent content loading → Secure signature algorithms are used
6. Verify that signature verification uses secure algorithms (RSA-2048+, ECDSA P-256+) → Signature verification cannot be bypassed
7. Test that signature verification cannot be bypassed or disabled → Signing keys are protected from extraction
8. Verify that signing keys are properly protected in the host application → Verification process is logged

Pass Criteria: All local content is signed AND signatures are verified before execution AND verification cannot be bypassed

Fail Criteria: Unsigned content can be loaded OR signature verification is bypassable OR weak signature algorithms

Evidence: Signed content bundles, signature verification code review, test results with modified content, cryptographic algorithm analysis, key protection documentation

References:

- Code Signing Best Practices (NIST): https://csrc.nist.gov/publications/detail/sp/800-183/final
- Android APK Signature Scheme: https://source.android.com/docs/security/features/apksigning
- Authenticode for Windows: https://docs.microsoft.com/en-us/windows-hardware/drivers/install/authenticode

Assessment: EMB-REQ-24 (Redirect chain trust enforcement)

Reference: EMB-REQ-24 - Host applications shall implement mechanisms to detect and prevent redirect chains that exit trusted domains

Given: A conformant embedded browser with origin allowlist (EMB-1 or higher)

Task: Redirect chain trust enforcement prevents open redirect attacks where trusted origins inadvertently redirect to attacker-controlled domains, potentially exposing JavaScript bridge APIs to malicious content. Without redirect validation, attackers can exploit legitimate redirects on trusted sites to bypass origin allowlists, tricking the embedded browser into loading malicious content that appears to come from a trusted source. Multi-hop redirect chains are particularly dangerous because each hop in the chain could be manipulated to eventually land on an untrusted domain, requiring comprehensive validation of the entire redirect path to prevent trust boundary violations.

Verification:

1. Configure trusted origin allowlist in the embedded browser → Verify that JavaScript-based redirects are also subject to trust enforcement
2. Load a page from a trusted origin that redirects to another trusted origin → Redirect chain validation is enforced
3. Verify that trusted-to-trusted redirects are allowed → Trusted-to-trusted redirects are allowed
4. Load a page from a trusted origin that redirects to an untrusted origin → Any redirect to untrusted origin is blocked
5. Verify that trusted-to-untrusted redirects are blocked → Multi-hop redirect chains are fully validated
6. Test multi-hop redirect chains: trusted → trusted → untrusted → Both HTTP and JavaScript redirects are subject to enforcement
7. Verify that all redirects in the chain are validated against allowlist → Blocked redirects trigger error pages and security logs
8. Test that redirect blocking triggers security logging → Users are notified when redirects are blocked

Pass Criteria: Redirect chains are fully validated AND untrusted redirects are blocked AND enforcement covers all redirect types

Fail Criteria: Redirects to untrusted origins succeed OR JavaScript redirects bypass enforcement OR multi-hop validation is incomplete

Evidence: Redirect test results, browser console showing blocked redirects, security logs, network traffic captures showing redirect chains, error page screenshots

References:

- OWASP Open Redirect: https://cheatsheetseries.owasp.org/cheatsheets/Unvalidated_Redirects_and_Forwards_Cheat_
- CWE-601: URL Redirection to Untrusted Site: https://cwe.mitre.org/data/definitions/601.html
- Android WebView shouldOverrideUrlLoading: https://developer.android.com/reference/android/webkit/WebViewClient
- iOS WKNavigationDelegate decidePolicyFor: https://developer.apple.com/documentation/webkit/wknavigationdelegate/webview

Assessment: EMB-REQ-25 (HSTS enforcement for trusted origins)

Reference: EMB-REQ-25 - Embedded browsers shall enforce strict transport security (HSTS) for all trusted content origins

Given: A conformant embedded browser (EMB-1 or higher)

Task: HSTS enforcement for trusted origins protects against SSL stripping attacks where network attackers intercept initial HTTP requests and prevent upgrade to HTTPS, allowing interception of all subsequent traffic. Without HSTS, embedded browsers are vulnerable during the brief window between application launch and the first HTTPS connection, when attackers can downgrade connections to HTTP and conduct man-in-the-middle attacks. HSTS ensures that all connections to trusted origins use HTTPS, with non-bypassable certificate validation, preventing network-level attackers from intercepting or modifying content loaded by the embedded browser even on compromised networks.

Verification:

1. Configure trusted content origins with HSTS headers: `Strict-Transport-Security: max-age=31536000; includeSubDomains; preload` → HSTS headers are parsed and cached correctly
2. Make initial HTTPS request to trusted origin and verify HSTS header is received and cached → HTTP requests to HSTS hosts are automatically upgraded to HTTPS
3. Attempt to make HTTP request to same origin and verify automatic upgrade to HTTPS → HSTS state persists across browser sessions
4. Test that HSTS enforcement continues after browser restart (persistence) → includeSubDomains flag applies to all subdomains
5. Verify that HSTS includeSubDomains flag is respected for subdomains → HSTS preload list is maintained and enforced
6. Test that HSTS preload list is supported and enforced → HSTS cannot be bypassed through IP access or other means
7. Attempt to bypass HSTS through direct IP access or hosts file manipulation → Certificate errors on HSTS hosts are non-bypassable
8. Verify that certificate errors on HSTS hosts cannot be bypassed → HSTS enforcement is logged

Pass Criteria: HSTS is enforced for all trusted origins AND HTTP is automatically upgraded AND certificate errors are non-bypassable

Fail Criteria: HSTS can be bypassed OR HTTP connections succeed to HSTS hosts OR certificate errors are bypassable

Evidence: Network traffic captures showing HTTPS upgrades, HSTS cache inspection, test results with HTTP requests, certificate error bypass attempts, HSTS preload list verification

References:

- RFC 6797: HTTP Strict Transport Security (HSTS): https://datatracker.ietf.org/doc/html/rfc6797
- HSTS Preload List: https://hstspreload.org/

- OWASP HSTS: https://cheatsheetseries.owasp.org/cheatsheets/HTTP_Strict_Transport_Security_Cheat_Sheet.html
- Chromium HSTS Implementation: https://www.chromium.org/hsts

Assessment: EMB-REQ-26 (Certificate validation failure notification)

Reference: EMB-REQ-26 - Certificate validation failures for trusted content sources shall trigger immediate user notification and content blocking

Given: A conformant embedded browser loading trusted remote content (EMB-1 or higher)

Task: Immediate notification and blocking on certificate validation failures for trusted content prevents silent man-in-the-middle attacks where users unknowingly interact with compromised content. Without clear error notifications and strict blocking, users might not realize that certificate validation has failed, allowing attackers to serve malicious content that appears to come from trusted origins. Non-bypassable certificate error handling for trusted origins ensures that even sophisticated attacks using fraudulent certificates cannot trick the embedded browser into loading untrusted content, protecting the integrity of the JavaScript bridge API access model.

Verification:

1. Configure trusted content origins with valid certificates → Certificate validation failures are detected for all failure types
2. Simulate certificate validation failures: expired certificate, hostname mismatch, untrusted CA, revoked certificate → User receives immediate, clear error notification
3. Verify that each failure type triggers immediate error notification → Content loading is completely blocked on validation failure
4. Verify that content loading is completely blocked (no partial loading) → Error messages explain the specific certificate problem
5. Test that error notifications clearly explain the problem to users → No bypass mechanism exists for certificate errors on trusted origins
6. Verify that no "continue anyway" option exists for trusted origin failures → Certificate failures are logged with full details
7. Test that certificate failures are logged to security event log → Host application can register for certificate failure callbacks
8. Verify that host application is notified of certificate failures via callback → Failure handling differs for trusted vs. untrusted origins

Pass Criteria: All certificate failures block content AND trigger immediate user notification AND are non-bypassable for trusted origins

Fail Criteria: Any certificate failure allows content loading OR notifications are missing/unclear OR bypass options exist

Evidence: Error page screenshots, certificate failure logs, test results with various failure types, user notification interface documentation, host callback implementation

References:

- RFC 5280: Certificate Validation: https://datatracker.ietf.org/doc/html/rfc5280
- CA/Browser Forum Baseline Requirements: https://cabforum.org/baseline-requirements-documents/
- Android WebView onReceivedSslError: https://developer.android.com/reference/android/webkit/WebViewClient#onRec
- iOS WKNavigationDelegate didReceive challenge: https://developer.apple.com/documentation/webkit/wknavigationdeleg webview

Assessment: EMB-REQ-27 (Network security configuration)

Reference: EMB-REQ-27 - Host applications shall implement network security configurations that prevent cleartext traffic to trusted domains

Given: A conformant embedded browser (EMB-1 or higher)

Task: This assessment verifies that host applications implement declarative network security configurations (Android Network Security Config, iOS App Transport Security, or equivalent) that prevent cleartext HTTP traffic to trusted domains. Cleartext traffic exposes sensitive data to network eavesdropping and man-in-the-middle attacks. The configuration shall enforce HTTPS-only communication for trusted domains at the network layer, apply uniformly across all network APIs (WebView, XHR, fetch, WebSocket), integrate certificate pinning settings, be declarative and immutable at runtime (not modifiable by web content), and log blocked cleartext traffic attempts for security monitoring. Platform-specific security features shall be properly configured to prevent accidental or malicious downgrade to insecure HTTP connections.

Verification:

1. Review network security configuration files (Android NSConfig, iOS ATS) → Network security configuration enforces HTTPS-only for trusted domains
2. Verify cleartext HTTP traffic disabled for trusted domains → Cleartext HTTP traffic blocked at network layer
3. Attempt HTTP requests to trusted domains across all network APIs → HTTPS enforcement applies uniformly to all network APIs
4. Test runtime modification attempts of HTTPS enforcement → Configuration is declarative and immutable at runtime
5. Verify certificate pinning integration with network security config → Certificate pinning integrated into network security config
6. Test cleartext traffic blocking and verify logging → Blocked cleartext traffic is logged
7. Review configuration documentation and auditability → Configuration documented and auditable
8. Verify platform-specific security features properly utilized → Platform features (NSConfig, ATS) correctly configured

Pass Criteria: Cleartext traffic to trusted domains is blocked at network layer AND configuration is immutable AND applies to all network APIs

Fail Criteria: Cleartext traffic succeeds to trusted domains OR configuration can be modified OR inconsistent enforcement across APIs

Evidence: Network security configuration files, network traffic captures showing blocked HTTP, test results with various network APIs, configuration documentation, platform security feature usage audit

References:

- Android Network Security Configuration: https://developer.android.com/training/articles/security-config
- iOS App Transport Security: https://developer.apple.com/documentation/security/preventing_insecure_network_conne
- CWE-319: Cleartext Transmission of Sensitive Information: https://cwe.mitre.org/data/definitions/319.html

Assessment: EMB-REQ-28 (CSP enforcement for third-party content)

Reference: EMB-REQ-28 - Embedded browsers shall prevent trusted content from loading untrusted third-party content without explicit CSP declarations

Given: A conformant embedded browser with CSP support (EMB-1 or higher)

Task: This assessment verifies that embedded browsers enforce Content Security Policy (CSP) to prevent trusted content from loading untrusted third-party resources without explicit declarations. Uncontrolled third-party content loading enables XSS attacks, data exfiltration, and supply chain compromises. The CSP shall restrict third-party content loading to explicitly allowlisted domains, apply uniformly to all resource types (scripts, styles, images, frames, etc.), report violations to configured endpoints, be immutable by web content, and have host-configured policies take precedence over page-specified CSP. Default CSP should be restrictive (deny by default), requiring explicit allowlisting for all third-party resources.

Verification:

1. Configure trusted content with restrictive CSP allowing specific third-party domains → CSP restricts third-party loading to explicitly allowed domains
2. Test loading from CSP-allowed domains and verify success → Allowed third-party content loads successfully
3. Attempt loading from non-allowed domains and verify blocking → Non-allowed third-party content is blocked
4. Test inline scripts from third-party content blocked unless CSP-allowed → CSP applies consistently across all resource types
5. Verify CSP applies to all resource types (scripts, styles, images, frames) → CSP violations generate reports to configured endpoint
6. Test CSP violations reported via report-uri or report-to → Web content cannot bypass or weaken CSP restrictions
7. Verify web content cannot weaken or disable CSP policy → Host-configured CSP cannot be overridden by page CSP
8. Test host-configured CSP takes precedence over page-specified CSP → CSP enforcement is logged and default policy is restrictive

Pass Criteria: Third-party content loading is restricted by CSP AND violations are blocked and reported AND policy cannot be weakened by web content

Fail Criteria: Third-party content bypasses CSP OR policy can be weakened OR violations not reported

Evidence: CSP policy headers, browser console showing CSP violations, violation reports, test results with blocked third-party content, network traffic analysis

References:

- Content Security Policy Level 3: https://www.w3.org/TR/CSP3/
- CSP script-src directive: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Security-Policy/script-src
- CSP Reporting: https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP#reporting
- OWASP CSP Cheat Sheet: https://cheatsheetseries.owasp.org/cheatsheets/Content_Security_Policy_Cheat_Sheet.html

Assessment: EMB-REQ-29 (Per-instance trust policies)

Reference: EMB-REQ-29 - Trust policies shall be configurable per embedded browser instance to support multi-tenant applications

Given: A conformant embedded browser supporting multiple instances (EMB-2 or higher)

Task: This assessment verifies that embedded browsers support per-instance trust policy configuration, allowing each instance to enforce different security policies independently while maintaining complete isolation from other instances.

Verification:

1. Create multiple embedded browser instances within the same host application → Trust policies can be configured independently per instance
2. Configure different trust policies for each instance (different origin allowlists, certificate pins, CSP policies) → Each instance enforces its own trust policy correctly
3. Load content in each instance and verify that instance-specific policies are enforced → Instances are isolated from each other (no cross-instance access)
4. Verify that instances cannot access each other's storage or state → Trust policy configuration is scoped to individual instances
5. Test that trust policy configuration is isolated per instance → Instance creation accepts custom policy configuration
6. Attempt to modify one instance's trust policy from another instance → Instance destruction cleans up policy state

7. Verify that instance destruction cleans up trust policy state → No interference between instances' trust enforcement
8. Test that host application can dynamically create instances with custom policies → Multi-tenant scenarios are properly supported

Pass Criteria: Per-instance trust policies are supported AND enforced correctly AND instances are isolated from each other

Fail Criteria: Policies cannot be configured per-instance OR instances interfere with each other OR policy isolation is broken

Evidence: Code showing per-instance policy configuration, test results with multiple instances, storage isolation verification, policy enforcement logs per instance, multi-tenant test scenarios

References:

- Android WebView Multiple Instances: https://developer.android.com/reference/android/webkit/WebView
- iOS WKWebView Configuration: https://developer.apple.com/documentation/webkit/wkwebviewconfiguration
- Electron BrowserView Isolation: https://www.electronjs.org/docs/latest/api/browser-view
- Multi-tenancy Security Best Practices: https://owasp.org/www-project-application-security-verification-standard/

Assessment: EMB-REQ-30 (Certificate Transparency verification)

Reference: EMB-REQ-30 - Embedded browsers shall implement certificate transparency verification for trusted content origins

Given: A conformant embedded browser with Certificate Transparency support (EMB-2 or higher)

Task: This assessment verifies that embedded browsers implement and enforce Certificate Transparency verification for trusted content origins, validating Signed Certificate Timestamps (SCTs) and blocking content from origins with missing or invalid SCTs.

Verification:

1. Configure trusted content origins that require Certificate Transparency → Certificate Transparency verification is supported
2. Load content from origins with valid SCTs (Signed Certificate Timestamps) in certificates or via TLS extension → Valid SCTs allow content loading from CT-required origins
3. Verify that content loads successfully when valid SCTs are present → Missing or invalid SCTs block content loading
4. Attempt to load content with certificates lacking SCTs from CT-required origins → SCTs are validated against known CT logs
5. Verify that missing or invalid SCTs block content loading for CT-required origins → CT enforcement policy is configurable (per-origin or global)
6. Test that SCTs are validated against known CT logs → CT validation failures generate clear error messages and logs
7. Verify that CT enforcement can be configured per-origin or globally → CT log list is kept up-to-date
8. Test that CT validation failures are logged → CT enforcement integrates with certificate validation flow

Pass Criteria: CT verification is implemented AND enforced for configured origins AND invalid SCTs block content loading

Fail Criteria: CT verification not implemented OR can be bypassed OR invalid SCTs are accepted

Evidence: Network traffic captures showing SCTs, CT validation logs, test results with missing SCTs, CT log list version verification, CT enforcement configuration

References:

- RFC 6962: Certificate Transparency: https://datatracker.ietf.org/doc/html/rfc6962

- Chromium CT Policy: https://www.chromium.org/Home/chromium-security/certificate-transparency/
- Certificate Transparency Logs: https://www.certificate-transparency.org/
- Google CT Policy: https://github.com/GoogleChrome/CertificateTransparency/blob/master/ct_policy.md

Assessment: EMB-REQ-31 (DNS rebinding attack prevention)

Reference: EMB-REQ-31 - Host applications shall detect and prevent DNS rebinding attacks targeting trusted local/internal origins

Given: A conformant embedded browser with local/internal origin access (EMB-2 or higher)

Task: This assessment verifies that embedded browsers detect and prevent DNS rebinding attacks targeting trusted local and internal origins through DNS caching, Host header validation, and logging of rebinding attempts.

Verification:

1. Identify local/internal origins accessed by the embedded browser (localhost, 127.0.0.1, private IP ranges, .local domains) → DNS rebinding attacks are detected
2. Simulate DNS rebinding attack: external domain initially resolves to attacker IP, then rebinds to local/internal IP → Unexpected DNS resolution changes trigger blocking
3. Verify that DNS rebinding is detected and blocked → Private IP address access requires explicit configuration
4. Test DNS cache behavior and TTL enforcement → DNS cache prevents rapid rebinding
5. Verify that private IP address access requires explicit allowlisting → Host header validation is enforced
6. Test that DNS resolution results are validated against expected address ranges → Resolution results are validated against address allowlists
7. Verify that Host header validation prevents rebinding exploitation → Rebinding attempts are logged as security events
8. Test that DNS rebinding attempts trigger security logging → Local/internal origins have additional protection

Pass Criteria: DNS rebinding attacks are detected and blocked AND private IP access is restricted AND rebinding attempts are logged

Fail Criteria: DNS rebinding succeeds OR no private IP restrictions OR rebinding not detected

Evidence: DNS rebinding test results, DNS cache configuration, Host header validation tests, security logs showing rebinding attempts, private IP access policy documentation

References:

- DNS Rebinding Explained: https://en.wikipedia.org/wiki/DNS_rebinding
- CWE-346: Origin Validation Error: https://cwe.mitre.org/data/definitions/346.html
- Private Network Access specification: https://wicg.github.io/private-network-access/

Assessment: EMB-REQ-32 (Trust boundary violation security events)

Reference: EMB-REQ-32 - Trust boundary violations shall trigger security events with sufficient context for incident response

Given: A conformant embedded browser with comprehensive logging (EMB-1 or higher)

Task: This assessment verifies that trust boundary violations trigger comprehensive security events with sufficient context for incident response, including proper distinction from general logs, SIEM integration capability, and alert generation for high-severity violations.

Verification:

1. Configure security event logging for the embedded browser → Trust boundary violations generate security events

2. Trigger various trust boundary violations (accessing blocked origins, calling restricted APIs, certificate validation failures, CSP violations) → Events include comprehensive context: timestamp, type, origin, description, severity
3. Review generated security events and verify they contain timestamp, event type, origin/source, detailed description, severity level, stack trace/context → Events are distinguishable from general logs (separate channel or marked)
4. Verify that security events are distinct from general application logs → Events can be exported or forwarded to security monitoring systems
5. Test that security events can be forwarded to SIEM or security monitoring systems → High-severity events trigger immediate alerts
6. Verify that high-severity violations trigger immediate alerts → Events include actionable information for incident response
7. Test that security events include sufficient context for incident investigation → Event logging is reliable and does not impact app stability
8. Verify that event generation does not disrupt application functionality → Events are tamper-resistant

Pass Criteria: All boundary violations generate security events AND events include sufficient context AND can be forwarded to monitoring systems

Fail Criteria: Violations don't generate events OR insufficient context OR cannot be monitored externally

Evidence: Security event log samples, event schema documentation, SIEM integration configuration, alert configuration, incident response playbooks referencing events, event log integrity verification

References:

- NIST SP 800-61: Computer Security Incident Handling Guide: https://csrc.nist.gov/publications/detail/sp/800-61/rev-2/final
- OWASP Logging Cheat Sheet: https://cheatsheetseries.owasp.org/cheatsheets/Logging_Cheat_Sheet.html
- CEF (Common Event Format): https://www.microfocus.com/documentation/arcsight/arcsight-smartconnectors/pdfdoc/common-event-format-v25/common-event-format-v25.pdf
- SIEM Integration Best Practices: https://www.sans.org/white-papers/36427/

Assessment: EMB-REQ-33 (Block web content access to host application objects at EMB-0)

Reference: EMB-REQ-33 - Embedded browser shall prevent all web content access to host application objects at EMB-0 capability level

Given: A conformant embedded browser with EMB-0 capability (no JavaScript bridge or native API exposure)

Task: At EMB-0 capability, embedded browsers shall completely isolate web content from host application objects, methods, and data structures to prevent JavaScript injection attacks, host context manipulation, and privilege escalation. Allowing any bridge interface, custom JavaScript objects, or window property injection at this level contradicts the security posture of complete isolation, enabling malicious web content to escape sandbox boundaries and compromise the host application. Complete object isolation with no window augmentation, strict namespace separation, and frozen global scope ensures zero native access.

Verification:

1. Load untrusted web content in the embedded browser → All common bridge object names undefined
2. Attempt to access common JavaScript bridge object names (AndroidBridge, webkit, chrome, nativeInterface, etc.) and verify they are undefined → Window object contains only standard web platform APIs
3. Enumerate window object properties and confirm no host-injected objects exist → No host-injected properties in global scope
4. Test reflection APIs (Object.keys, Object.getOwnPropertyNames) on window and verify no custom properties → Reflection APIs show only standard properties

5. Attempt to override native JavaScript methods and verify host does not expose overridden methods → Native method overrides not exploitable for host communication
6. Test for custom event listeners added by host and confirm none exist → No custom event listeners from host
7. Attempt to access parent/opener contexts and verify they are inaccessible or null → Parent/opener contexts inaccessible
8. Verify window.external is undefined or blocked (legacy IE bridge interface) → window.external undefined or blocked
9. Test for prompt/alert/confirm hijacking that would allow host communication → Dialog APIs do not create host communication channels
10. Verify no custom URL schemes trigger host-side callbacks at EMB-0 → Custom URL schemes do not trigger host callbacks

Pass Criteria: All bridge objects undefined AND window object contains only standard APIs AND no host-injected properties AND no communication channels exist

Fail Criteria: Bridge objects accessible OR custom window properties present OR host communication possible OR custom schemes trigger callbacks

Evidence: JavaScript console enumeration of window properties, reflection API output showing standard APIs only, bridge access attempt logs showing errors, URL scheme handler testing, dialog API interaction verification

References:

- OWASP Mobile WebView Security: https://mas.owasp.org/MASTG/tests/android/MASVS-PLATFORM/MASTG-TEST-0028/
- Android WebView addJavascriptInterface: https://developer.android.com/reference/android/webkit/WebView#addJavas
- iOS WKWebView Script Message Handlers: https://developer.apple.com/documentation/webkit/wkusercontentcontroller
- Electron Context Isolation: https://www.electronjs.org/docs/latest/tutorial/context-isolation

Assessment: EMB-REQ-34 (No JavaScript bridge exposure at EMB-0)

Reference: EMB-REQ-34 - No JavaScript bridge or native API exposure mechanism shall exist at EMB-0 capability level

Given: A conformant embedded browser with EMB-0 capability (no JavaScript bridge or native API exposure)

Task: JavaScript bridges create privileged communication channels between web content and native host applications, fundamentally violating EMB-0's isolation requirement. Any bridge mechanism - addJavascript-Interface (Android), WKScriptMessageHandler (iOS), contextBridge (Electron), or custom implementations - enables web content to invoke native methods, access host data, and potentially execute arbitrary code in the native context. At EMB-0, all bridge registration APIs should be disabled, configuration immutable, and attempted bridge creation blocked to ensure complete isolation.

Verification:

1. Review embedded browser initialization code and verify no bridge registration calls present → No addJavascriptInterface calls in Android WebView initialization
2. Test Android WebView addJavascriptInterface and confirm the API is not called or is blocked → No WKScriptMessageHandler registered in iOS WKWebView
3. Test iOS WKWebView addScriptMessageHandler and verify no handlers are registered → Electron contextBridge not exposed to renderer
4. Verify Electron contextBridge is not exposed in renderer preload scripts → Runtime bridge registration attempts fail
5. Attempt to register custom bridge handlers at runtime and verify registration fails → window.external undefined or non-functional

6. Test legacy bridge mechanisms (window.external, custom protocol schemes) and confirm they are disabled → Custom protocol scheme handlers do not create bridge channels
7. Verify embedded browser configuration prevents dynamic bridge creation → Configuration prevents dynamic bridge creation
8. Review application code for bridge-related APIs and confirm none are used → Application code audit shows no bridge APIs used
9. Test that web content cannot trigger native method invocation through any mechanism → Web content cannot invoke native methods
10. Verify security logs show zero bridge registration or access attempts → Security logs confirm zero bridge activity

Pass Criteria: No bridge registration APIs called AND runtime bridge creation blocked AND no legacy bridge mechanisms AND web-to-native invocation impossible

Fail Criteria: Bridge APIs called OR dynamic registration possible OR legacy bridges functional OR native invocation succeeds

Evidence: Source code review showing no bridge registration, runtime bridge attempt logs showing failures, WebView/browser configuration showing bridge disabled, security log analysis showing zero bridge activity, web content isolation testing

References:

- Android WebView JavaScript Interface Security: https://developer.android.com/reference/android/webkit/WebView#ad
- iOS WKWebView Message Handlers: https://developer.apple.com/documentation/webkit/wkscriptmessagehandler
- Electron Context Isolation: https://www.electronjs.org/docs/latest/tutorial/context-isolation
- OWASP WebView Security: https://mas.owasp.org/MASTG/tests/android/MASVS-PLATFORM/MASTG-TEST-0028/

Assessment: EMB-REQ-35 (Treat all content as untrusted at EMB-0)

Reference: EMB-REQ-35 - Embedded browser shall treat all loaded content as completely untrusted at EMB-0 capability level

Given: A conformant embedded browser with EMB-0 capability (no JavaScript bridge or native API exposure)

Task: At EMB-0, embedded browsers shall apply maximum security restrictions to all web content regardless of source origin, loading mechanism, or content type, treating even application-bundled content as potentially malicious. Trusting any content source creates privilege escalation risks where attackers who compromise trusted origins gain disproportionate access. Universal untrusted treatment with strict CSP enforcement, feature policy restrictions, and capability lockdown ensures consistent security posture across all content, preventing attackers from exploiting trust assumptions to escape isolation boundaries.

Verification:

1. Load content from application bundle resources and verify same security restrictions as remote content → Bundled content has same restrictions as remote content
2. Test application-bundled HTML/JS and confirm no privileged access granted → No privileged access for application resources
3. Load content from localhost and verify it receives no special treatment → Localhost receives no elevated privileges
4. Test file:// protocol content and confirm strict security policies applied → file:// content strictly restricted
5. Verify Content Security Policy enforced uniformly for all content sources → CSP applied uniformly to all sources
6. Test that bundled content cannot access host APIs unavailable to remote content → Host APIs equally inaccessible from all origins
7. Verify feature policy restrictions apply equally to all origins → Feature policies enforced consistently

8. Load same content from different sources (bundle, localhost, remote) and verify identical API availability → API availability identical across content sources
9. Test that no "trusted origin" configuration grants elevated privileges at EMB-0 → No trusted origin exceptions at EMB-0
10. Verify permission prompts (if any) appear for all content regardless of source → Permission requirements uniform across origins

Pass Criteria: All content sources treated identically AND no privileged access for bundled content AND CSP enforced uniformly AND no trusted origin exceptions

Fail Criteria: Bundled content has elevated privileges OR localhost trusted OR file:// bypasses restrictions OR trusted origins exist

Evidence: Content source comparison testing showing identical restrictions, CSP policy verification across sources, API availability enumeration by source, bundled content security analysis, trusted origin configuration audit

References:

- Android WebView Local Content Security: https://developer.android.com/develop/ui/views/layout/webapps/webview#lo content
- Content Security Policy: https://www.w3.org/TR/CSP3/
- Feature Policy: https://www.w3.org/TR/permissions-policy-1/
- OWASP WebView Best Practices: https://mas.owasp.org/MASTG/tests/android/MASVS-PLATFORM/MASTG-TEST-0028/

Assessment: EMB-REQ-36 (Allowlist-only bridge API access at EMB-1)

Reference: EMB-REQ-36 - Only explicitly allowlisted APIs shall be accessible from web content at EMB-1 capability level

Given: A conformant embedded browser with EMB-1 capability (limited JavaScript bridge with explicit allowlist)

Task: This assessment verifies that JavaScript bridge implementations use explicit allowlisting to expose only pre-approved, security-reviewed APIs to web content, with runtime enforcement that blocks reflection-based attacks, dynamic invocation exploits, and unintended API exposure. The assessment confirms that bridge objects are immutable from web context, reflection APIs either show only allowlisted methods or are blocked entirely, and comprehensive API inventory documentation exists to prevent undocumented API access and host compromise.

Verification:

1. Review JavaScript bridge configuration and document complete allowlist of exposed APIs → Allowlisted APIs documented comprehensively
2. Attempt to invoke API methods not in the allowlist and verify calls are blocked with errors → Non-allowlisted method calls blocked with errors
3. Use reflection APIs in JavaScript to enumerate bridge object methods → Reflection shows only allowlisted methods or is blocked
4. Verify reflection returns only allowlisted methods (or reflection is blocked entirely) → Dynamic invocation limited to allowlist
5. Test dynamic invocation (bridgemethodName) and confirm only allowlisted methods callable → Prototype chain not exploitable for API discovery
6. Attempt to access bridge object prototype and verify it's not exploitable for additional access → Method name matching is strict
7. Test method name variations (case changes, encoding) and verify strict matching enforced → Bridge object immutable from web context
8. Attempt to add methods to bridge object at runtime and verify it's immutable → API additions require native code changes

9. Test that bridge API additions require host application code changes and cannot be made from web content → No undocumented APIs accessible
10. Verify documentation and implementation match (no undocumented APIs accessible) → Security review documentation exists for all allowlisted APIs

Pass Criteria: Only allowlisted APIs accessible AND reflection blocked or limited AND dynamic invocation restricted AND bridge immutable from web

Fail Criteria: Non-allowlisted APIs accessible OR reflection exposes unapproved methods OR dynamic invocation bypasses allowlist OR bridge mutable

Evidence: Bridge configuration showing explicit allowlist, reflection API output, method invocation logs showing blocked calls, immutability testing results, security review documentation for allowlisted APIs

References:

- OWASP API Security: https://owasp.org/www-project-api-security/
- Principle of Least Privilege: https://csrc.nist.gov/glossary/term/least_privilege
- Android WebView Interface Security: https://developer.android.com/reference/android/webkit/WebView#addJavascript
- Secure JavaScript Bridge Design: https://www.electronjs.org/docs/latest/tutorial/context-isolation

Assessment: EMB-REQ-37 (Minimal and documented bridge API allowlist at EMB-1)

Reference: EMB-REQ-37 - Bridge API allowlist shall be minimal and documented at EMB-1 capability level

Given: A conformant embedded browser with EMB-1 capability (limited JavaScript bridge with explicit allowlist)

Task: JavaScript bridge security requires minimizing exposed API surface to reduce attack vectors, limit exploit potential, and simplify security review. Each exposed API creates opportunities for parameter injection, logic bugs, race conditions, and privilege escalation. Comprehensive documentation with security implications, threat models, input constraints, and usage examples enables security assessment, developer guidance, and audit compliance. Minimal API surface with complete documentation balances functionality needs with security risk management.

Verification:

1. Review complete bridge API allowlist and count exposed methods/functions → Complete API inventory documented
2. For each exposed API, verify security documentation exists explaining: purpose, parameters, return values, security implications, threat model → Each API has security documentation
3. Verify exposed APIs represent minimum necessary for application functionality → Security docs include threat models
4. Review application code and confirm all exposed bridge APIs are actually used → API surface justified as minimal
5. Identify any exposed APIs that are unused and flag for removal → No unused APIs exposed
6. Verify documentation includes input validation requirements for each API → Input validation requirements documented
7. Test that developers can understand API security implications from documentation alone → Security implications clearly explained
8. Verify threat model documentation covers potential attack vectors for each API → Attack vectors identified per API
9. Confirm documentation includes safe usage examples and anti-patterns to avoid → Safe usage examples provided
10. Verify security review sign-off exists for the API surface → Security review approval documented

Pass Criteria: API surface minimal and justified AND complete security documentation AND threat models per API AND security review approved

Fail Criteria: Excessive API surface OR missing documentation OR no threat models OR unused APIs exposed OR no security review

Evidence: Bridge API inventory with justifications, security documentation for each API, threat model documents, usage analysis showing API necessity, security review approval records

References:

- Attack Surface Reduction: https://csrc.nist.gov/glossary/term/attack_surface
- Threat Modeling: https://owasp.org/www-community/Threat_Modeling
- API Documentation Best Practices: https://swagger.io/specification/
- OWASP API Security: https://owasp.org/www-project-api-security/

Assessment: EMB-REQ-38 (Bidirectional bridge security controls at EMB-2)

Reference: EMB-REQ-38 - Bidirectional bridge calls shall maintain same security controls in both directions at EMB-2 capability level

Given: A conformant embedded browser with EMB-2 capability (extended JavaScript bridge with bidirectional communication)

Task: Bidirectional JavaScript bridges enable both web-to-native and native-to-web communication, creating dual attack surfaces where malicious web content can exploit native-to-web callbacks and compromised native code can exploit web-to-native handlers. Asymmetric security controls create vulnerabilities where attackers bypass strict web-to-native validation by exploiting weaker native-to-web callback validation. Symmetric security enforcement with identical validation, logging, rate limiting, and permission checks in both directions prevents directional exploitation and maintains consistent security posture.

Verification:

1. Document all bidirectional bridge APIs (web-to-native calls and native-to-web callbacks) → Bidirectional APIs fully documented
2. Test input validation on web-to-native calls and document validation rules → Web-to-native calls validated comprehensively
3. Test input validation on native-to-web callbacks and verify identical validation strictness → Native-to-web callbacks equally validated
4. Compare validation, sanitization, and type checking between directions → Validation rules identical in strictness
5. Verify logging is comprehensive for both web-to-native and native-to-web communications → Logging comprehensive for both directions
6. Test rate limiting applies equally to both communication directions → Rate limiting symmetric
7. Verify permission checks enforced symmetrically → Permission checks enforced both ways
8. Attempt to bypass web-to-native validation by exploiting native-to-web callback path → Callback path cannot bypass validation
9. Test that callbacks cannot be used to inject unvalidated data into web context → Callbacks do not inject unvalidated data
10. Verify security audit events generated for both communication directions → Audit events generated for all communications

Pass Criteria: Validation symmetric across directions AND logging comprehensive for both AND rate limiting equal AND no bypass via callback path

Fail Criteria: Asymmetric validation OR logging gaps in one direction OR rate limiting inconsistent OR callback path bypasses controls

Evidence: Bidirectional API documentation, validation rule comparison, logging configuration showing symmetry, rate limiting policy analysis, bypass attempt test results, security audit logs

References:

- Secure IPC Design: https://chromium.googlesource.com/chromium/src/+/master/docs/security/mojo.md
- Bidirectional Communication Security: https://www.electronjs.org/docs/latest/tutorial/message-ports
- Input Validation: https://cheatsheetseries.owasp.org/cheatsheets/Input_Validation_Cheat_Sheet.html
- Defense in Depth: https://csrc.nist.gov/glossary/term/defense_in_depth

Assessment: EMB-REQ-39 (Web content callback handler validation at EMB-2)

Reference: EMB-REQ-39 - Web content callback handlers shall be validated before invocation at EMB-2 capability level

Given: A conformant embedded browser with EMB-2 capability (extended JavaScript bridge with bidirectional communication)

Task: Native-to-web callbacks invoke JavaScript functions provided by web content, creating code injection and XSS risks if callbacks execute attacker-controlled functions or pass unvalidated data. Without strict validation, malicious web content registers callback handlers that exploit native invocation context, inject payloads through callback parameters, or leverage privileged execution timing. Handler validation with type checking, origin verification, parameter sanitization, and safe invocation patterns prevents callback-based attacks while enabling legitimate bidirectional communication.

Verification:

1. Register web content callback handler with JavaScript bridge → Callback existence validated before invocation
2. Trigger native-to-web callback from host application → Non-function callbacks rejected or handled safely
3. Verify native code validates callback handler exists and is a function before invocation → Callback parameters validated on native side
4. Test registering non-function callback and verify native code rejects or handles safely → Type checking enforced for complex objects
5. Verify callback parameters are validated and sanitized on native side before passing to web → Callbacks cannot access privileged state
6. Test passing complex objects through callbacks and confirm type checking enforced → Error handling prevents stack trace exposure
7. Attempt to register callback that accesses privileged global state and verify isolation maintained → Callbacks execute in restricted context
8. Test callback error handling and confirm errors don't expose native stack traces → Origin verification performed before invocation
9. Verify callbacks execute in restricted context without elevated privileges → Callback invocation logged for audit
10. Test that callback invocation includes origin verification → Malicious callbacks detected and blocked

Pass Criteria: Callbacks validated before invocation AND parameters sanitized AND error handling safe AND origin verification performed

Fail Criteria: Callbacks invoked without validation OR parameters unsanitized OR errors expose details OR no origin verification

Evidence: Callback registration and invocation code review, parameter validation testing, error handling analysis, origin verification logs, malicious callback detection results

References:

- Callback Security: https://www.electronjs.org/docs/latest/tutorial/security#17-validate-the-options-of-the-webcontents-opendevtools-call
- XSS Prevention: https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html
- Safe JavaScript Invocation: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Func
- Input Validation: https://cheatsheetseries.owasp.org/cheatsheets/Input_Validation_Cheat_Sheet.html

Assessment: EMB-REQ-40 (Bridge message queuing with integrity protection at EMB-2)

Reference: EMB-REQ-40 - Bridge shall implement message queuing with integrity protection at EMB-2 capability level

Given: A conformant embedded browser with EMB-2 capability (extended JavaScript bridge with bidirectional communication)

Task: Asynchronous JavaScript bridge communication requires message queuing to decouple sender and receiver timing, but queues create attack surfaces for message tampering, injection, reordering, and replay attacks. Without integrity protection, attackers intercept queued messages to modify parameters, inject malicious messages, reorder operations to cause logic bugs, or replay messages to duplicate actions. Authenticated message queuing with sequence numbers, cryptographic MACs, and queue integrity verification prevents message manipulation while enabling reliable asynchronous communication.

Verification:

1. Trigger asynchronous JavaScript bridge communication requiring message queuing → Messages include sequence numbers or timestamps
2. Verify messages include sequence numbers or timestamps for ordering → Tampering detected through integrity checks
3. Attempt to tamper with queued message content and verify integrity check detects modification → Injected messages rejected due to authentication failure
4. Test message injection into queue and verify authentication prevents acceptance → Reordering detected and handled safely
5. Attempt to reorder messages in queue and verify sequence validation detects anomaly → Replay attacks prevented by sequence/nonce validation
6. Test message replay attack and verify sequence/nonce checking prevents duplicate processing → Cryptographic integrity protection used (MAC/signature)
7. Verify queue implementation uses cryptographic integrity protection (MAC, signature) → Queue overflow handled securely
8. Test queue overflow scenarios and verify graceful degradation without security compromise → Queue not manipulable by web content
9. Verify message queue is not accessible to web content for direct manipulation → Integrity failures logged as security events
10. Test that queue integrity failures trigger security events and logging → Queue implementation resists race conditions

Pass Criteria: Messages integrity protected AND sequence validation enforced AND tampering detected AND replay prevention implemented

Fail Criteria: No integrity protection OR sequence validation absent OR tampering undetected OR replay attacks succeed

Evidence: Message format documentation showing integrity fields, tampering test results, injection attempt logs, replay attack prevention verification, queue implementation security review

References:

- Message Queue Security: https://www.rabbitmq.com/security.html
- HMAC Integrity Protection: https://csrc.nist.gov/projects/hash-functions
- Replay Attack Prevention: https://cheatsheetseries.owasp.org/cheatsheets/Cryptographic_Storage_Cheat_Sheet.html
- Secure IPC: https://chromium.googlesource.com/chromium/src/+/master/docs/security/mojo.md

Assessment: EMB-REQ-41 (Bridge traffic anomaly monitoring at EMB-2)

Reference: EMB-REQ-41 - Host shall monitor bridge traffic for anomalies at EMB-2 capability level

Given: A conformant embedded browser with EMB-2 capability (extended JavaScript bridge with bidirectional communication)

Task: JavaScript bridge traffic exhibits predictable patterns during normal application usage, enabling anomaly detection to identify attacks, compromised content, or exploited vulnerabilities. Monitoring for rate anomalies, suspicious method sequences, unusual parameters, or exploit patterns enables early attack detection and incident response. Without traffic monitoring, attackers exploit bridges repeatedly, probe for vulnerabilities, or exfiltrate data without detection. Real-time anomaly detection with baseline profiling, statistical analysis, and automated alerting prevents prolonged bridge exploitation.

Verification:

1. Establish baseline bridge traffic profile during normal application usage → Baseline traffic profile established
2. Configure anomaly detection with thresholds for: message rate, method frequency, parameter patterns, error rates → Anomaly detection thresholds configured
3. Trigger normal bridge usage and verify no false positive alerts → Normal usage generates no false positives
4. Simulate attack scenarios: rapid API calls exceeding rate limits, unusual method combinations, malformed parameters → Attack scenarios detected reliably
5. Verify anomaly detection system identifies and alerts on attack traffic → Alerts include investigation context
6. Test that alerts include sufficient context for investigation (origin, methods called, parameters, timing) → SIEM integration functional
7. Verify anomaly detection integrates with application security monitoring/SIEM → Automated responses implemented
8. Test automated responses to anomalies (throttling, blocking, enhanced logging) → Baseline adapts to usage changes
9. Verify baseline profiles updated automatically to reflect legitimate usage changes → Monitoring overhead acceptable
10. Test that monitoring performance overhead is acceptable → Security team receives actionable alerts

Pass Criteria: Anomaly detection functional AND attack scenarios detected AND false positives minimal AND automated alerting works

Fail Criteria: No anomaly detection OR attacks undetected OR excessive false positives OR no alerting integration

Evidence: Baseline traffic profile documentation, anomaly detection configuration, attack simulation results showing detection, alert samples, SIEM integration logs, performance impact analysis

References:

- Security Monitoring: https://www.sans.org/white-papers/36472/
- SIEM Integration: https://owasp.org/www-community/Log_Injection
- Behavioral Analysis: https://csrc.nist.gov/publications/detail/sp/800-94/final

Assessment: EMB-REQ-42 (Enterprise bridge API policy configuration at EMB-2)

Reference: EMB-REQ-42 - Enterprise administrators shall be able to configure bridge API policies at EMB-2 capability level

Given: A conformant embedded browser with EMB-2 capability supporting enterprise policy management

Task: Enterprise deployments require centralized control over JavaScript bridge capabilities to enforce organizational security standards, prevent data exfiltration, comply with regulations, and manage risk. Without policy controls, users or developers expose sensitive APIs that violate corporate security policies, grant excessive privileges to web content, or enable unauthorized native functionality. Enterprise policy integration with API allowlist management, permission restrictions, and audit requirements enables IT governance over bridge security posture.

Verification:

1. Access enterprise policy configuration interface (MDM, configuration profile, policy file) → Enterprise policy interface accessible to IT administrators
2. Configure policy to restrict bridge API allowlist to specific approved methods → API allowlist restrictions enforceable via policy
3. Deploy policy to application and verify restricted methods are inaccessible to web content → Restricted APIs blocked with clear errors
4. Test that web content attempting to access policy-blocked APIs receives clear errors → Audit logging requirement enforceable
5. Configure policy requiring audit logging for all bridge communications → Logging cannot be disabled when policy requires it
6. Verify policy-mandated logging cannot be disabled by application developers → Rate limits configurable and enforced
7. Configure policy setting maximum rate limits for bridge API calls → Bridge functionality can be disabled by policy
8. Test that rate limits are enforced per policy configuration → Policy changes effective without recompilation
9. Verify policy can completely disable bridge functionality if required → Policy violations logged to enterprise monitoring
10. Test that policy changes take effect without requiring application recompilation → Administrators can audit current bridge configuration

Pass Criteria: Enterprise policies enforced AND API restrictions work AND logging mandatory when required AND policy updates dynamic

Fail Criteria: Policies not enforced OR API restrictions bypassed OR logging can be disabled OR policy updates require recompilation

Evidence: Enterprise policy configuration documentation, API restriction enforcement testing, audit logging verification, rate limit enforcement analysis, policy update procedures, compliance reports

References:

- Enterprise Mobile Management: https://developer.apple.com/documentation/devicemanagement
- Android Enterprise: https://developers.google.com/android/work
- Mobile Application Management: https://www.microsoft.com/en-us/security/business/threat-protection/mobile-application-management
- Security Policy Enforcement: https://csrc.nist.gov/glossary/term/policy_enforcement

Assessment: EMB-REQ-43 (Core security boundaries preserved at EMB-3)

Reference: EMB-REQ-43 - Full integration shall not bypass core security boundaries at EMB-3 capability level

Given: A conformant embedded browser with EMB-3 capability (full integration with native capabilities)

Task: Even with extensive native integration at EMB-3, fundamental security boundaries should remain intact to prevent complete application compromise. Core boundaries include: renderer process isolation from host process memory, web content sandboxing, origin-based security policies, and cryptographic key isolation. Bypassing these boundaries through bridge integration enables attackers who compromise web content to escalate directly to host privileges, access other origins' data, or extract cryptographic material. Maintaining core boundaries while enabling integration preserves defense-in-depth and limits exploit impact.

Verification:

1. Verify renderer process isolation maintained even with extensive bridge integration → Renderer process isolation intact
2. Test that web content cannot access host process memory directly through bridge APIs → Host process memory inaccessible from web content

3. Verify origin-based security model enforced (same-origin policy, CORS) despite native integration → Same-origin policy enforced
4. Test that bridge APIs cannot bypass origin restrictions to access other origins' data → Bridge cannot bypass origin restrictions
5. Verify cryptographic keys and credentials remain isolated from web content → Cryptographic keys isolated
6. Test that bridge cannot expose raw file system paths or handles that bypass sandbox → File system sandbox boundaries maintained
7. Verify CSP and other content security policies remain enforced → CSP enforced for all content
8. Test that full integration does not disable process-level sandboxing → Process sandboxing active
9. Verify network security policies (certificate validation, HSTS) cannot be bypassed through bridge → Network security policies cannot be bypassed
10. Test that bridge APIs cannot grant web content ability to load arbitrary native code → Native code loading restricted

Pass Criteria: Process isolation maintained AND origin policies enforced AND crypto keys isolated AND sandbox boundaries intact

Fail Criteria: Process isolation bypassed OR origin policies circumvented OR keys accessible OR sandbox escaped

Evidence: Process isolation testing, origin policy enforcement verification, cryptographic key isolation audit, sandbox boundary testing, security architecture review

References:

- Browser Security Architecture: https://www.chromium.org/developers/design-documents/multi-process-architecture/
- Process Isolation: https://www.chromium.org/developers/design-documents/site-isolation/
- Same-Origin Policy: https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy
- Defense in Depth: https://csrc.nist.gov/glossary/term/defense_in_depth

Assessment: EMB-REQ-44 (User awareness of native capabilities at EMB-3)

Reference: EMB-REQ-44 - User shall be informed of all native capabilities granted to web content at EMB-3 capability level

Given: A conformant embedded browser with EMB-3 capability (full integration with native capabilities)

Task: Users should understand which native capabilities are accessible to web content to make informed security and privacy decisions. Without transparency, malicious web content silently accesses camera, location, contacts, filesystem, or other sensitive native APIs through bridge integration, enabling surveillance, data theft, and privacy violations. Clear capability disclosure through permission prompts, settings interfaces, and usage indicators empowers users to grant appropriate access, revoke excessive permissions, and detect unauthorized native API usage.

Verification:

1. Identify all native capabilities accessible through JavaScript bridge at EMB-3 → Permission prompts appear for sensitive capabilities
2. Verify permission prompts appear when web content first requests sensitive native capabilities → Prompts clearly explain native access
3. Test that prompts clearly explain what native access is being granted (e.g., "camera access", "contact list access") → Settings show all granted capabilities
4. Access application settings and verify list of granted native capabilities is displayed → Capabilities explained in user-friendly terms
5. Verify settings interface explains each capability in user-friendly language → Web origins with capabilities visible

6. Test that users can see which web origins have been granted each capability → Active usage indicators function
7. Verify active capability usage indicators appear (e.g., camera indicator when camera accessed) → Indicators show which content is active
8. Test that users can click indicators to see which web content is using capabilities → Grants persist across restarts
9. Verify capability grants are persistent and visible across application restarts → Documentation explains permissions model
10. Test that application documentation explains bridge capabilities and permissions model → Users can make informed decisions

Pass Criteria: Permission prompts clear AND settings show all capabilities AND usage indicators present AND documentation complete

Fail Criteria: No permission prompts OR settings incomplete OR no usage indicators OR missing documentation

Evidence: Permission prompt screenshots, settings interface documentation, usage indicator demonstration, capability grant persistence testing, user documentation review

References:

- Permission UX Best Practices: https://web.dev/permission-ux/
- Mobile Permission Models: https://developer.android.com/guide/topics/permissions/overview
- iOS Permission Model: https://developer.apple.com/design/human-interface-guidelines/privacy
- User Privacy Controls: https://www.w3.org/TR/privacy-controls/

Assessment: EMB-REQ-45 (User permission review and revocation at EMB-3)

Reference: EMB-REQ-45 - User shall be able to review and revoke native API access at EMB-3 capability level

Given: A conformant embedded browser with EMB-3 capability (full integration with native capabilities)

Task: Users should have granular control to review all granted native API permissions and revoke access to manage security posture and respond to threats. Without revocation capabilities, users cannot remove access from compromised web content, recover from accidental over-privileged grants, or adjust permissions as trust relationships change. Permission management interface with per-origin, per-capability revocation, immediate enforcement, and usage history enables user control, security hygiene, and privacy protection.

Verification:

1. Grant multiple native capabilities (camera, location, contacts, storage) to web content through bridge → All granted capabilities visible in settings
2. Access application permission management settings interface → Per-capability revocation functional
3. Verify interface lists all granted capabilities organized by web origin and capability type → Revocation takes effect immediately
4. Select individual capability and revoke it through settings → Web content receives clear errors after revocation
5. Immediately test that web content can no longer access revoked capability → Bulk revocation available per origin
6. Verify web content receives clear error when attempting to use revoked capability → Usage history displayed for capabilities
7. Test bulk revocation of all capabilities for a specific web origin → Revocations persist across restarts
8. Verify permission management shows usage history (when capabilities were last accessed) → Re-granting requires new user consent
9. Test that revocations persist across application restarts → No hidden or unrevocable capabilities exist
10. Verify revoked permissions require new user consent if web content requests again → Interface is user-friendly and accessible

Pass Criteria: All capabilities reviewable AND revocation immediate AND per-capability control AND usage history visible

Fail Criteria: Capabilities hidden OR revocation delayed OR only bulk revocation OR no usage history

Evidence: Permission management UI screenshots, revocation testing showing immediate effect, usage history exports, persistence verification, hidden capability audit

References:

- Android Permission Management: https://developer.android.com/training/permissions/requesting
- iOS Permission Management: https://developer.apple.com/documentation/uikit/protecting_the_user_s_privacy
- Permission Management UX: https://web.dev/permission-ux/
- User Privacy Controls: https://www.w3.org/TR/privacy-controls/

Assessment: EMB-REQ-46 (Native integration audit documentation at EMB-3)

Reference: EMB-REQ-46 - All native integrations shall be documented and auditable at EMB-3 capability level

Given: A conformant embedded browser with EMB-3 capability (full integration with native capabilities)

Task: Extensive native integration at EMB-3 creates complex attack surfaces requiring comprehensive documentation for security review, vulnerability assessment, compliance auditing, and risk management. Without documentation of exposed native APIs, security boundaries relaxed, and integration patterns used, security teams cannot assess risk, auditors cannot verify compliance, and incident responders cannot investigate breaches effectively. Complete architecture documentation with threat models, security reviews, API inventories, and integration justifications enables informed security governance and compliance verification.

Verification:

1. Review application security documentation for native integration architecture → Native integration architecture documented
2. Verify complete inventory of all native APIs exposed through JavaScript bridge → Complete API inventory available
3. Confirm each exposed API has security documentation including: purpose, parameters, threat model, mitigations → Per-API security documentation exists
4. Review threat model documentation covering bridge integration attack vectors → Threat models cover bridge integration
5. Verify security review records exist for native integration design and implementation → Security review records available
6. Test that runtime diagnostics expose current bridge configuration for auditing → Runtime diagnostics expose configuration
7. Verify enterprise administrators can access detailed integration documentation → Enterprise documentation comprehensive
8. Review code comments and confirm they explain security-critical integration points → Code comments explain security-critical sections
9. Verify integration patterns and security controls are documented with examples → Integration patterns documented with examples
10. Test that documentation is maintained and updated with application versions → Documentation current with application version

Pass Criteria: Complete API inventory AND per-API security docs AND threat models AND security review records

Fail Criteria: Incomplete inventory OR missing security docs OR no threat models OR no security reviews

Evidence: Security documentation collection, API inventory with security annotations, threat model documents, security review approval records, runtime configuration dumps

References:

- Secure Development Lifecycle: https://www.microsoft.com/en-us/securityengineering/sdl/
- Threat Modeling: https://owasp.org/www-community/Threat_Modeling
- Security Documentation: https://cheatsheetseries.owasp.org/cheatsheets/Security_Documentation_Checklist.html
- API Security: https://owasp.org/www-project-api-security/

Assessment: EMB-REQ-47 (Enterprise native integration restrictions at EMB-3)

Reference: EMB-REQ-47 - Enterprise policies shall be able to restrict native integration scope at EMB-3 capability level

Given: A conformant embedded browser with EMB-3 capability supporting enterprise policy management

Task: Enterprise environments require centralized control to restrict native integration capabilities that pose unacceptable risk to organizational security, intellectual property, or compliance. Without policy controls, users or developers expose unrestricted camera access, filesystem operations, contact data, or other sensitive capabilities that enable data exfiltration or violate corporate policies. Enterprise policy integration with capability blocking, API allowlisting, and permission restrictions enables IT security governance over native integration posture.

Verification:

1. Access enterprise policy configuration interface (MDM, configuration profile, policy file) → Enterprise policy interface accessible
2. Configure policy to block specific native capabilities organization-wide (e.g., camera, contact access) → Capability blocking enforceable
3. Deploy policy and verify blocked capabilities are inaccessible to all web content → Blocked capabilities inaccessible with clear errors
4. Test that web content attempting to access policy-blocked capabilities receives clear policy violation errors → Origin allowlisting functional for sensitive APIs
5. Configure policy to allowlist specific web origins for sensitive capabilities → Non-allowlisted origins denied access
6. Test that only allowlisted origins can access sensitive capabilities → Maximum permission scope enforceable
7. Configure policy to restrict maximum permission scope for all web content → Users cannot exceed policy limits
8. Verify users cannot grant permissions exceeding policy-defined maximum → Bridge can be fully disabled by policy
9. Test that policy can completely disable JavaScript bridge if required → Policy violations logged to enterprise systems
10. Verify policy violations are logged to enterprise security monitoring → Administrators can audit current integration configuration

Pass Criteria: Capability blocking enforced AND origin allowlisting works AND permission scope limits effective AND violations logged

Fail Criteria: Policies not enforced OR blocking bypassed OR allowlists ignored OR no violation logging

Evidence: Enterprise policy configuration documentation, capability blocking testing, origin allowlist verification, permission scope limit testing, policy violation logs

References:

- Enterprise Mobile Management: https://developer.apple.com/documentation/devicemanagement
- Android Enterprise Policies: https://developers.google.com/android/work/requirements
- Mobile Application Management: https://www.microsoft.com/en-us/security/business/threat-protection/mobile-application-management
- Enterprise Security Governance: https://csrc.nist.gov/glossary/term/security_governance

6.9 Remote Data Processing Systems Security Assessments

This section covers assessment procedures for requirements RDPS-REQ-1 through RDPS-REQ-45, addressing secure remote data processing, encryption in transit and at rest, authentication and authorization, availability and disaster recovery, data minimization and protection, and user configuration security.

Assessment: RDPS-REQ-1 (Offline functionality documentation)

Reference: RDPS-REQ-1 - Browser shall document product functionality when RDPS connectivity unavailable

Given: A conformant browser with RDPS-1 or higher capability

Task: Users and administrators should understand how browser functionality changes when remote data processing systems become unavailable due to network failures, service outages, or infrastructure issues. Without clear documentation of offline behavior, users cannot assess business continuity risk, plan for service disruptions, or make informed deployment decisions. Comprehensive documentation with feature availability matrices, degradation scenarios, data synchronization behavior, and recovery procedures enables informed risk management and operational planning.

Verification:

1. Review browser documentation for RDPS offline functionality coverage → Documentation covers offline functionality comprehensively
2. Verify documentation explains which features remain functional without RDPS connectivity → Feature availability matrix provided (online vs offline)
3. Confirm documentation lists features that become degraded or unavailable offline → Degraded features clearly identified
4. Test browser behavior when RDPS connectivity lost and verify it matches documentation → Actual behavior matches documented offline behavior
5. Verify documentation explains data synchronization behavior after connectivity restoration → Synchronization behavior explained
6. Test that users are notified when RDPS becomes unavailable → User notifications for RDPS unavailability documented
7. Verify documentation includes troubleshooting steps for RDPS connectivity issues → Troubleshooting guidance provided
8. Test that critical features identified in documentation continue functioning offline → Critical features function offline as documented
9. Verify documentation explains local data caching behavior during RDPS outages → Caching behavior explained
10. Confirm documentation describes maximum offline operation duration → Offline duration limits documented

Pass Criteria: Comprehensive offline documentation AND feature matrix provided AND behavior matches documentation AND user notifications described

Fail Criteria: Missing offline documentation OR no feature matrix OR behavior doesn't match docs OR no notification guidance

Evidence: Documentation review showing offline functionality coverage, feature availability matrix, offline behavior testing results, user notification examples, synchronization behavior documentation

References:

- NIST SP 800-160 Vol. 2: Systems Security Engineering - Cyber Resiliency: https://csrc.nist.gov/publications/detail/sp/800-160/vol-2/final
- Business Continuity Planning: https://www.iso.org/standard/75106.html
- Resilient System Design: https://owasp.org/www-project-resilient-system-design/

Assessment: RDPS-REQ-2 (Data classification and inventory)

Reference: RDPS-REQ-2 - Browser shall define all data processed or stored in RDPS with data classification

Given: A conformant browser with RDPS-1 or higher capability

Task: Data classification is fundamental to RDPS security, enabling appropriate protection controls, compliance with regulations (GDPR, CCPA), and risk-informed security decisions. Without complete data inventory and classification, organizations cannot assess privacy risks, implement proportionate security controls, or demonstrate regulatory compliance. Comprehensive data catalog with sensitivity levels, data types, retention requirements, and processing purposes enables informed security governance and privacy protection.

Verification:

1. Review browser RDPS data inventory documentation → Complete data inventory exists
2. Verify inventory includes complete list of data types processed/stored remotely → All RDPS data types documented
3. Confirm each data type has assigned classification (public, internal, confidential, restricted) → Classification assigned to each type
4. Test that classification reflects actual sensitivity of data → Classification reflects actual sensitivity
5. Verify documentation explains purpose and necessity for each data type → Purpose documented for each data type
6. Confirm data inventory includes retention periods for each type → Retention periods specified
7. Review RDPS data flows and verify they match inventory → Data flows match inventory
8. Test that no undocumented data is transmitted to RDPS → No undocumented data transmission occurs
9. Verify classification system aligns with industry standards (ISO 27001, NIST) → Classification system follows standards
10. Confirm inventory is maintained and updated with product versions → Inventory kept current with product updates

Pass Criteria: Complete data inventory AND classification for all types AND purpose documented AND retention specified AND no undocumented data

Fail Criteria: Incomplete inventory OR missing classifications OR purpose undefined OR no retention periods OR undocumented data found

Evidence: Data inventory documentation, classification scheme, data flow diagrams, network traffic analysis showing only documented data, retention policy documentation

References:

- ISO/IEC 27001 Information Security Management: https://www.iso.org/standard/27001
- NIST SP 800-60 Guide for Mapping Types of Information: https://csrc.nist.gov/publications/detail/sp/800-60/vol-1-rev-1/final
- GDPR Data Protection: https://gdpr-info.eu/
- Data Classification Best Practices: https://www.sans.org/white-papers/36857/

Assessment: RDPS-REQ-3 (Data criticality classification)

Reference: RDPS-REQ-3 - Browser shall classify criticality of all RDPS-processed data

Given: A conformant browser with RDPS-1 or higher capability

Task: Data criticality classification determines appropriate availability requirements, backup strategies, and disaster recovery priorities for RDPS data. Without criticality assessment, organizations cannot allocate resources appropriately, prioritize recovery efforts during incidents, or implement risk-proportionate protection controls. Criticality classification with availability requirements, recovery objectives (RTO/RPO), and business impact analysis enables effective business continuity and disaster recovery planning.

Verification:

1. Review browser RDPS data criticality classification documentation → Criticality classification exists for all data types
2. Verify each data type has assigned criticality level (critical, important, standard, low) → Criticality levels assigned systematically
3. Confirm criticality reflects business impact of data loss or unavailability → Business impact drives criticality assessment
4. Verify documentation includes Recovery Time Objective (RTO) for critical data → RTO specified for critical data
5. Confirm documentation specifies Recovery Point Objective (RPO) for critical data → RPO documented for critical data
6. Test that backup frequency aligns with RPO requirements → Backup frequency meets RPO
7. Verify high-availability mechanisms deployed for critical data → High-availability for critical data
8. Test data recovery procedures and verify they meet RTO targets → Recovery meets RTO targets
9. Confirm business impact analysis justifies criticality classifications → Business impact analysis documented
10. Verify criticality classifications updated with product functionality changes → Classifications updated with product changes

Pass Criteria: Criticality classification for all data AND RTO/RPO specified for critical data AND backup frequency appropriate AND recovery tested

Fail Criteria: Missing criticality classification OR no RTO/RPO specified OR inadequate backup frequency OR recovery fails RTO

Evidence: Criticality classification documentation, RTO/RPO specifications, business impact analysis, backup frequency configuration, recovery test results

References:

- NIST SP 800-34 Contingency Planning Guide: https://csrc.nist.gov/publications/detail/sp/800-34/rev-1/final
- ISO 22301 Business Continuity: https://www.iso.org/standard/75106.html
- Disaster Recovery Planning: https://www.ready.gov/business/implementation/IT

Assessment: RDPS-REQ-4 (TLS 1.3 encryption for data transmission)

Reference: RDPS-REQ-4 - Browser shall encrypt all data transmissions to RDPS using TLS 1.3 or higher

Given: A conformant browser with RDPS-1 or higher capability

Task: Unencrypted RDPS communications expose data to eavesdropping, man-in-the-middle attacks, and credential theft, enabling attackers to intercept sensitive information, modify data in transit, or hijack sessions. TLS 1.3 provides strong encryption with perfect forward secrecy, modern cipher suites, and protection against downgrade attacks. Enforcing minimum TLS 1.3 for all RDPS communications prevents protocol vulnerabilities, ensures strong cryptographic protection, and maintains confidentiality and integrity of data transmission.

Verification:

1. Configure network monitoring to capture browser-RDPS traffic → All RDPS connections encrypted with TLS
2. Trigger RDPS operations requiring data transmission → TLS 1.3 or higher enforced
3. Analyze captured traffic and verify all connections use TLS → TLS 1.2 and lower rejected
4. Verify TLS version is 1.3 or higher (no TLS 1.2, 1.1, 1.0 allowed) → Modern cipher suites used
5. Confirm cipher suites used are TLS 1.3 compliant (AES-GCM, ChaCha20-Poly1305) → Perfect forward secrecy enabled
6. Test that browser rejects TLS 1.2 or lower connections to RDPS → Downgrade attacks prevented
7. Verify perfect forward secrecy (PFS) enabled through ephemeral key exchange → Certificate validation mandatory

8. Test that browser prevents TLS downgrade attacks → No plaintext transmission detected
9. Verify certificate validation enforced for RDPS endpoints → Handshake completes with TLS 1.3 parameters
10. Confirm no plaintext data transmission occurs → Connection parameters meet security requirements

Pass Criteria: TLS 1.3+ enforced for all RDPS AND older TLS rejected AND PFS enabled AND certificate validation mandatory

Fail Criteria: TLS 1.2 or lower allowed OR unencrypted transmissions OR no PFS OR certificate validation optional

Evidence: Network traffic captures showing TLS 1.3, TLS version enforcement testing, cipher suite analysis, downgrade attack test results, certificate validation verification

References:

- TLS 1.3 RFC 8446: https://www.rfc-editor.org/rfc/rfc8446
- NIST SP 800-52 Rev. 2 TLS Guidelines: https://csrc.nist.gov/publications/detail/sp/800-52/rev-2/final
- Mozilla TLS Configuration: https://wiki.mozilla.org/Security/Server_Side_TLS
- OWASP Transport Layer Protection: https://cheatsheetseries.owasp.org/cheatsheets/Transport_Layer_Protection_Chea

Assessment: RDPS-REQ-5 (RDPS endpoint certificate validation)

Reference: RDPS-REQ-5 - Browser shall authenticate RDPS endpoints using certificate validation

Given: A conformant browser with RDPS-1 or higher capability

Task: RDPS endpoint authentication prevents man-in-the-middle attacks, server impersonation, and phishing attacks targeting remote data processing infrastructure. Without proper certificate validation, attackers can intercept RDPS communications, steal credentials, modify data, or redirect traffic to malicious servers. Comprehensive certificate validation with chain verification, revocation checking, hostname verification, and certificate pinning ensures authentic, trusted RDPS endpoints and prevents connection hijacking.

Verification:

1. Test RDPS connection with valid, trusted certificate and verify connection succeeds → Valid certificates accepted
2. Test with expired certificate and verify connection blocked with clear error → Expired certificates rejected
3. Test with self-signed certificate and confirm connection rejected → Self-signed certificates blocked
4. Test with certificate for wrong hostname and verify hostname verification fails → Hostname verification enforced
5. Test with revoked certificate and confirm revocation check prevents connection → Revocation checking performed
6. Verify certificate chain validation enforced (intermediate and root CAs verified) → Certificate chain validated
7. Test certificate pinning if implemented and verify pinned certificates required → Certificate pinning enforced (if implemented)
8. Attempt MITM attack with attacker-controlled certificate and verify detection → MITM attacks detected through certificate mismatch
9. Verify browser displays clear error messages for certificate validation failures → Clear error messages displayed
10. Test that users cannot bypass certificate errors for RDPS connections → Certificate errors not bypassable

Pass Criteria: Certificate validation enforced AND revocation checked AND hostname verified AND chain validated AND errors not bypassable

Fail Criteria: Invalid certificates accepted OR no revocation checking OR hostname not verified OR chain not validated OR errors bypassable

Evidence: Certificate validation test results, revocation checking logs, hostname verification testing, MITM attack prevention demonstration, error message screenshots

References:

- RFC 5280 X.509 Certificate Validation: https://www.rfc-editor.org/rfc/rfc5280
- RFC 6962 Certificate Transparency: https://www.rfc-editor.org/rfc/rfc6962
- OWASP Certificate Pinning: https://owasp.org/www-community/controls/Certificate_and_Public_Key_Pinning
- Certificate Validation Best Practices: https://csrc.nist.gov/publications/detail/sp/800-52/rev-2/final

Assessment: RDPS-REQ-6 (Retry mechanisms with exponential backoff)

Reference: RDPS-REQ-6 - Browser shall implement retry mechanisms with exponential backoff for RDPS failures

Given: A conformant browser with RDPS-1 or higher capability

Task: RDPS connectivity failures are inevitable due to network issues, server outages, or rate limiting. Without intelligent retry mechanisms, browsers either fail immediately (poor user experience) or retry aggressively (amplifying outages, triggering rate limits). Exponential backoff with jitter provides graceful degradation by spacing retries progressively further apart, reducing server load during outages while maintaining eventual connectivity. Proper retry logic with maximum attempts, timeout bounds, and failure notification enables resilient RDPS operations.

Verification:

1. Simulate RDPS connectivity failure (network disconnection, server timeout) → Initial retry after short delay
2. Verify browser attempts initial retry after short delay (e.g., 1-2 seconds) → Exponential backoff applied to subsequent retries
3. Confirm subsequent retries use exponentially increasing delays (2s, 4s, 8s, 16s, etc.) → Jitter randomization prevents synchronized retries
4. Test that jitter is added to prevent thundering herd (randomized delay component) → Maximum retry attempts enforced
5. Verify maximum retry attempts limit exists (e.g., 5-10 attempts) → Total retry duration bounded
6. Confirm total retry duration has upper bound (e.g., maximum 5 minutes) → User notification after exhaustion
7. Test that user is notified after retry exhaustion → Manual retry option available
8. Verify browser provides manual retry option after automatic retries fail → Successful retry restores operation
9. Test that successful retry restores normal operation without user intervention → Retry state persists appropriately
10. Confirm retry state persists across browser restarts for critical operations → No infinite retry loops

Pass Criteria: Exponential backoff implemented AND jitter applied AND maximum attempts enforced AND user notified on failure

Fail Criteria: Linear retry timing OR no jitter OR infinite retries OR no user notification

Evidence: Retry timing logs showing exponential pattern, jitter analysis, maximum attempt enforcement testing, user notification screenshots, retry exhaustion behavior verification

References:

- Exponential Backoff Algorithm: https://en.wikipedia.org/wiki/Exponential_backoff
- AWS Architecture Best Practices: https://aws.amazon.com/architecture/well-architected/
- Google Cloud Retry Strategy: https://cloud.google.com/architecture/scalable-and-resilient-apps
- Circuit Breaker Pattern: https://martinfowler.com/bliki/CircuitBreaker.html

Assessment: RDPS-REQ-7 (Local data caching for offline operation)

Reference: RDPS-REQ-7 - Browser shall cache critical data locally for offline operation

Given: A conformant browser with RDPS-1 or higher capability

Task: Local caching enables browser functionality continuation during RDPS outages by storing critical data locally for offline access. Without caching, RDPS unavailability renders browser features completely non-functional, creating poor user experience and business continuity risks. Intelligent caching with staleness policies, cache invalidation, and synchronization on reconnection balances offline functionality with data freshness and storage constraints.

Verification:

1. Identify critical data types requiring local caching (preferences, configuration, essential state) → Critical data cached locally
2. Verify browser caches critical data locally when RDPS accessible → Cached data accessible offline
3. Disconnect RDPS connectivity and verify cached data remains accessible → Browser functions with cached data
4. Test that browser continues functioning with cached data during outage → Staleness policies enforced
5. Verify cache staleness policies implemented (e.g., maximum cache age) → Users notified of stale data
6. Test that stale cached data is marked and users notified of potential outdatedness → Cache synchronizes on reconnection
7. Restore RDPS connectivity and verify cache synchronization occurs → Conflict resolution implemented
8. Test conflict resolution when local cached data differs from RDPS data → Cache size limits enforced
9. Verify cache size limits enforced to prevent excessive local storage consumption → Eviction policies functional
10. Test cache eviction policies (LRU, priority-based) when limits reached → Cache integrity maintained

Pass Criteria: Critical data cached AND accessible offline AND staleness detection AND synchronization on reconnection

Fail Criteria: No caching OR cached data inaccessible offline OR no staleness detection OR synchronization fails

Evidence: Cache storage verification, offline functionality testing, staleness policy documentation, synchronization logs, conflict resolution testing, cache size limit enforcement

References:

- HTTP Caching: https://developer.mozilla.org/en-US/docs/Web/HTTP/Caching
- Cache-Control Best Practices: https://web.dev/http-cache/
- Offline First Design: https://offlinefirst.org/
- Service Workers Caching: https://web.dev/service-workers-cache-storage/

Assessment: RDPS-REQ-8 (Secure authentication for RDPS access)

Reference: RDPS-REQ-8 - Browser shall implement secure authentication for RDPS access

Given: A conformant browser with RDPS-1 or higher capability

Task: RDPS authentication prevents unauthorized access to user data, enforces multi-tenancy boundaries, and enables audit trails linking actions to identities. Weak authentication enables data breaches, privacy violations, and unauthorized modifications. Strong authentication with secure credential storage, session management, token refresh, and multi-factor support where appropriate ensures only authorized browsers access RDPS data.

Verification:

1. Verify browser implements secure authentication mechanism (OAuth 2.0, OIDC, or equivalent) → Secure authentication mechanism implemented

2. Test that authentication credentials stored securely (encrypted, OS keychain integration) → Credentials stored securely
3. Verify authentication tokens time-limited (expiration enforced) → Tokens time-limited with expiration
4. Test token refresh mechanism for long-lived sessions → Token refresh functional
5. Verify expired tokens handled gracefully (automatic refresh or re-authentication) → Expired token handling graceful
6. Test that authentication failures trigger clear user prompts → Authentication failures prompt user
7. Verify session binding to prevent token theft attacks → Session binding prevents theft
8. Test that authentication tokens transmitted only over encrypted channels → Tokens transmitted encrypted only
9. Verify logout functionality properly invalidates tokens → Logout invalidates tokens
10. Test multi-factor authentication support if required for sensitive data → MFA supported where required

Pass Criteria: Strong authentication mechanism AND secure credential storage AND token expiration AND encrypted transmission

Fail Criteria: Weak authentication OR plaintext credentials OR no token expiration OR unencrypted token transmission

Evidence: Authentication mechanism documentation, credential storage analysis, token lifecycle testing, session security verification, logout effectiveness testing

References:

- OAuth 2.0 RFC 6749: https://www.rfc-editor.org/rfc/rfc6749
- OpenID Connect: https://openid.net/connect/
- OWASP Authentication Cheat Sheet: https://cheatsheetseries.owasp.org/cheatsheets/Authentication_Cheat_Sheet.html
- Token-Based Authentication: https://auth0.com/learn/token-based-authentication-made-easy/

Assessment: RDPS-REQ-9 (Certificate pinning for RDPS)

Reference: RDPS-REQ-9 - Browser shall validate server certificates and enforce certificate pinning for RDPS

Given: A conformant browser with RDPS-1 or higher capability

Task: Certificate pinning prevents man-in-the-middle attacks exploiting compromised Certificate Authorities by validating RDPS endpoints against pre-configured expected certificates or public keys. Without pinning, attackers with rogue CA certificates can intercept RDPS communications despite TLS encryption. Certificate pinning with backup pins, pin rotation procedures, and failure reporting provides defense-in-depth against sophisticated attacks targeting RDPS infrastructure.

Verification:

1. Review browser configuration for RDPS certificate pins (leaf cert, intermediate, or public key hashes) → Certificate pins configured
2. Test successful RDPS connection with correctly pinned certificate → Correctly pinned certificates accepted
3. Attempt connection with valid but unpinned certificate and verify rejection → Unpinned certificates rejected
4. Test that pinning failure triggers clear error without allowing connection → Pinning failures block connection
5. Verify backup pins configured to prevent operational lockout → Backup pins prevent lockout
6. Test pin rotation procedure during certificate renewal → Pin rotation procedure documented
7. Verify pinning failures reported to manufacturer for monitoring → Failures reported for monitoring
8. Test that pin validation occurs before establishing data connection → Validation occurs before data connection
9. Verify pin configuration immutable by web content or local tampering → Pin configuration tamper-resistant

10. Test graceful degradation if pinning causes connectivity issues (with user consent) → Graceful degradation with user consent

Pass Criteria: Certificate pinning enforced AND backup pins configured AND rotation procedure documented AND failures reported

Fail Criteria: No pinning OR single point of failure (no backup pins) OR no rotation procedure OR failures not reported

Evidence: Certificate pin configuration, pinning enforcement testing, backup pin verification, rotation procedure documentation, failure reporting logs

References:

- RFC 7469 Public Key Pinning: https://www.rfc-editor.org/rfc/rfc7469
- OWASP Certificate Pinning: https://owasp.org/www-community/controls/Certificate_and_Public_Key_Pinning
- Certificate Pinning Best Practices: https://noncombatant.org/2015/05/01/about-http-public-key-pinning/
- Chrome Certificate Pinning: https://www.chromium.org/Home/chromium-security/security-faq/

Assessment: RDPS-REQ-10 (RDPS connection timeout controls)

Reference: RDPS-REQ-10 - Browser shall implement timeout controls for RDPS connections

Given: A conformant browser with RDPS-1 or higher capability

Task: Connection timeouts prevent browsers from hanging indefinitely on unresponsive RDPS endpoints due to network issues, server failures, or denial-of-service attacks. Without timeouts, user experience degrades as browser features become non-responsive waiting for RDPS responses. Appropriate timeout values balanced between network latency tolerance and responsiveness enable reliable RDPS operations with graceful failure handling.

Verification:

1. Review browser RDPS timeout configuration (connection timeout, read timeout) → Connection timeout configured appropriately
2. Test connection establishment timeout (e.g., 30 seconds for initial connection) → Read timeout enforced
3. Simulate slow server response and verify read timeout enforced (e.g., 60 seconds) → Timeouts trigger graceful errors
4. Test that timeout triggers graceful error handling (not crash or hang) → Users notified with actionable messages
5. Verify user notified of timeout with actionable message → Timeout values network-appropriate
6. Test timeout values appropriate for expected network conditions → Different timeouts for operation criticality
7. Verify different timeout values for critical vs non-critical operations → Valid slow operations not aborted
8. Test that timeouts don't prematurely abort valid slow operations → Enterprise timeout configuration available
9. Verify timeout configuration adjustable for enterprise deployments → Behavior consistent across networks
10. Test timeout behavior under various network conditions (WiFi, cellular, slow networks) → No hangs or crashes on timeout

Pass Criteria: Connection and read timeouts configured AND graceful error handling AND user notification AND enterprise configurability

Fail Criteria: No timeouts OR hangs on unresponsive servers OR no user notification OR timeouts too aggressive

Evidence: Timeout configuration documentation, timeout enforcement testing, error handling verification, user notification screenshots, network condition testing results

References:

- HTTP Timeouts: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Keep-Alive
- Network Timeout Best Practices: https://www.nginx.com/blog/performance-tuning-tips-tricks/
- Resilient System Design: https://aws.amazon.com/builders-library/timeouts-retries-and-backoff-with-jitter/

Assessment: RDPS-REQ-11 (RDPS connectivity failure logging)

Reference: RDPS-REQ-11 - Browser shall log RDPS connectivity failures and errors

Given: A conformant browser with RDPS-1 or higher capability

Task: Comprehensive RDPS failure logging enables troubleshooting, performance monitoring, security incident detection, and reliability improvement. Without detailed logs, diagnosing RDPS issues becomes impossible, preventing root cause analysis and remediation. Structured logging with error details, timestamps, retry attempts, and contextual information supports operational visibility and continuous improvement.

Verification:

1. Simulate various RDPS failure scenarios (network timeout, DNS failure, connection refused, TLS error) → All failure types logged
2. Verify each failure type logged with appropriate severity level → Appropriate severity levels assigned
3. Confirm logs include timestamp, error type, RDPS endpoint, and failure reason → Logs include complete metadata
4. Test that retry attempts logged with attempt number and delay → Retry attempts documented
5. Verify authentication failures logged separately with rate limiting (prevent log flooding) → Authentication failures logged with rate limiting
6. Test that logs accessible to administrators for troubleshooting → Logs accessible to administrators
7. Verify user privacy protected (no sensitive data in logs) → User privacy protected
8. Test log rotation to prevent unbounded growth → Log rotation implemented
9. Verify critical failures trigger alerts or prominent log markers → Critical failures marked/alerted
10. Test log export capability for analysis tools → Export capability available

Pass Criteria: All failure types logged AND complete metadata AND privacy protected AND log management implemented

Fail Criteria: Failures not logged OR insufficient metadata OR sensitive data exposed OR unbounded log growth

Evidence: Log samples for various failure types, log schema documentation, privacy analysis, log rotation verification, export functionality testing

References:

- OWASP Logging Cheat Sheet: https://cheatsheetseries.owasp.org/cheatsheets/Logging_Cheat_Sheet.html
- Log Management Best Practices: https://www.sans.org/white-papers/33528/

Assessment: RDPS-REQ-12 (Graceful functionality degradation when RDPS unavailable)

Reference: RDPS-REQ-12 - Browser shall gracefully degrade functionality when RDPS unavailable

Given: A conformant browser with RDPS-1 or higher capability

Task: Graceful degradation ensures browsers remain usable during RDPS outages by maintaining core functionality while clearly communicating reduced capabilities to users. Without graceful degradation, RDPS failures cause complete feature unavailability, error messages, or undefined behavior confusing users. Intelligent degradation with feature prioritization, offline alternatives, and status communication balances service continuity with user expectations during infrastructure disruptions.

Verification:

1. Identify browser features dependent on RDPS connectivity → Core browser functionality maintained during outage
2. Document expected degradation behavior for each RDPS-dependent feature → RDPS-dependent features degrade gracefully
3. Simulate RDPS unavailability and verify core browser functionality remains operational → Users notified clearly of reduced functionality
4. Test that RDPS-dependent features degrade gracefully (don't crash or show errors) → Offline alternatives activate automatically
5. Verify users receive clear notification of reduced functionality with explanation → Features restore automatically on reconnection
6. Test that cached/offline alternatives activate automatically when RDPS unavailable → Degradation state visible in UI
7. Verify degraded features automatically restore when RDPS connectivity returns → No data loss during degradation
8. Test that degradation state visible in browser UI (status indicator, settings) → Manual retry available for operations
9. Verify no data loss occurs during degradation period → Degradation behavior matches documentation
10. Test user ability to manually retry RDPS-dependent operations → User experience remains acceptable

Pass Criteria: Core functionality maintained AND graceful degradation implemented AND user notifications clear AND automatic restoration on reconnection

Fail Criteria: Browser crashes OR features fail with errors OR no user notification OR functionality doesn't restore

Evidence: Degradation behavior documentation, offline functionality testing, user notification screenshots, reconnection restoration verification, data integrity testing

References:

- Graceful Degradation Patterns: https://developer.mozilla.org/en-US/docs/Glossary/Graceful_degradation
- Resilient Web Design: https://resilientwebdesign.com/
- Offline First: https://offlinefirst.org/

Assessment: RDPS-REQ-13 (Credentials protection from RDPS exposure)

Reference: RDPS-REQ-13 - Browser shall not expose sensitive authentication credentials to RDPS

Given: A conformant browser with RDPS-1 or higher capability

Task: Authentication credentials (passwords, tokens, keys) shall never be transmitted to or stored in RDPS to prevent credential theft, unauthorized access, and account compromise. Even with encrypted transmission, storing credentials in RDPS creates centralized breach targets and insider threat risks. The application of zero-knowledge architecture where only derived authentication proofs or encrypted credentials (with client-side keys) are shared ensures RDPS compromise cannot directly expose user credentials.

Verification:

1. Review RDPS data inventory and verify no passwords or plaintext credentials transmitted → No plaintext credentials in RDPS traffic
2. Capture network traffic during authentication and verify credentials not sent to RDPS → Only tokens or derived proofs transmitted
3. Test that only authentication tokens or derived proofs transmitted to RDPS → Credentials encrypted if stored remotely
4. Verify RDPS receives hashed/encrypted credentials at most (not plaintext) → Encryption keys remain client-side
5. Test that cryptographic keys for credential encryption stored client-side only → Password changes handled locally

6. Verify password changes occur locally without RDPS involvement in plaintext handling → RDPS cannot authenticate independently
7. Test that RDPS cannot authenticate users without client cooperation → Recovery mechanisms protect credentials
8. Verify credential recovery/reset mechanisms don't expose credentials to RDPS → Breach simulation confirms credential safety
9. Test that RDPS data breach simulation doesn't reveal credentials → Documentation explicit about credential handling
10. Verify security documentation explicitly states credentials never sent to RDPS → Zero-knowledge architecture implemented

Pass Criteria: No plaintext credentials to RDPS AND only tokens/proofs transmitted AND encryption keys client-side AND zero-knowledge architecture

Fail Criteria: Credentials sent to RDPS OR RDPS can authenticate users OR encryption keys on server OR plaintext storage

Evidence: Network traffic analysis showing no credentials, RDPS data inventory review, encryption key location verification, breach simulation results, security architecture documentation

References:

- Zero-Knowledge Architecture: https://en.wikipedia.org/wiki/Zero-knowledge_proof
- Credential Storage Best Practices: https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html
- Client-Side Encryption: https://www.owasp.org/index.php/Cryptographic_Storage_Cheat_Sheet

Assessment: RDPS-REQ-14 (RDPS request rate limiting)

Reference: RDPS-REQ-14 - Browser shall implement rate limiting for RDPS requests

Given: A conformant browser with RDPS-1 or higher capability

Task: Rate limiting prevents browsers from overwhelming RDPS infrastructure with excessive requests due to bugs, loops, or malicious content, protecting service availability for all users. Without rate limiting, single misbehaving clients can cause denial of service, increase costs, and degrade performance for legitimate users. Client-side rate limiting with request throttling, burst allowances, and backoff on rate limit errors ensures responsible RDPS resource consumption.

Verification:

1. Review browser rate limiting configuration for RDPS requests → Rate limiting configured appropriately
2. Test normal operation remains within rate limits → Normal operation within limits
3. Trigger rapid RDPS requests (e.g., through script loop) and verify throttling applied → Excessive requests throttled
4. Test that rate limiting implemented per-operation type (different limits for different APIs) → Per-operation type limits enforced
5. Verify burst allowances permit short spikes without immediate throttling → Burst allowances functional
6. Test that rate limit exceeded triggers exponential backoff (not immediate retry) → Backoff on limit exceeded
7. Verify user notified when rate limits significantly impact functionality → User notification for significant impacts
8. Test that rate limits documented for developers/administrators → Rate limits documented
9. Verify enterprise deployments can adjust rate limits for their needs → Enterprise configurability available
10. Test that rate limiting doesn't prevent legitimate high-frequency operations → Legitimate operations not blocked

Pass Criteria: Rate limiting implemented AND per-operation limits AND burst handling AND backoff on exceeded limits

Fail Criteria: No rate limiting OR single global limit OR no burst handling OR immediate retry on limit

Evidence: Rate limiting configuration documentation, throttling test results, burst handling verification, backoff behavior analysis, enterprise configuration options

References:

- API Rate Limiting: https://cloud.google.com/architecture/rate-limiting-strategies-techniques
- Token Bucket Algorithm: https://en.wikipedia.org/wiki/Token_bucket
- Rate Limiting Best Practices: https://www.keycdn.com/support/rate-limiting

Assessment: RDPS-REQ-15 (RDPS data validation before processing)

Reference: RDPS-REQ-15 - Browser shall validate all data received from RDPS before processing

Given: A conformant browser with RDPS-1 or higher capability

Task: Comprehensive data validation prevents compromised or malicious RDPS from injecting harmful data into browsers, causing security vulnerabilities, crashes, or unexpected behavior. Without validation, attackers who compromise RDPS can exploit browsers by sending malformed data, injection attacks, or excessive payloads. Multi-layer validation with schema enforcement, type checking, size limits, and sanitization provides defense-in-depth against RDPS compromise.

Verification:

1. Review RDPS data validation implementation for all data types → Schema validation enforced
2. Test that browser validates data schema matches expected format → Type checking comprehensive
3. Verify type checking enforced (strings, numbers, booleans validated correctly) → Size limits prevent overflow
4. Test size limits prevent excessive data payloads from RDPS → Content sanitization applied
5. Verify data sanitization for HTML/JavaScript content from RDPS → Malformed data rejected gracefully
6. Test that malformed JSON/data rejected with appropriate errors → Unexpected fields handled safely
7. Verify unexpected fields in RDPS responses ignored or flagged → NULL/undefined handling secure
8. Test that NULL/undefined values handled safely → Numeric range validation implemented
9. Verify numeric ranges validated (no integer overflow, invalid values) → Validation failures logged
10. Test that validation failures logged for security monitoring → Defense-in-depth validation layers

Pass Criteria: Schema validation enforced AND type checking comprehensive AND size limits applied AND sanitization for risky content

Fail Criteria: No validation OR incomplete type checking OR no size limits OR no sanitization

Evidence: Validation implementation review, malformed data rejection testing, injection attempt results, size limit enforcement verification, validation failure logs

References:

- Input Validation: https://cheatsheetseries.owasp.org/cheatsheets/Input_Validation_Cheat_Sheet.html
- JSON Schema Validation: https://json-schema.org/
- Data Sanitization: https://cheatsheetseries.owasp.org/cheatsheets/XSS_Filter_Evasion_Cheat_Sheet.html

Assessment: RDPS-REQ-16 (Data at rest encryption in RDPS storage)

Reference: RDPS-REQ-16 - Browser shall encrypt sensitive data at rest in RDPS storage

Given: A conformant browser with RDPS-2 or higher capability

Task: Encryption at rest protects RDPS data from unauthorized access through physical media theft, backup compromises, or infrastructure breaches. Without encryption, attackers gaining physical or logical access to RDPS storage can read all user data directly. Strong encryption with secure key management, algorithm compliance (AES-256), and access controls ensures data confidentiality even if storage media compromised.

Verification:

1. Review RDPS storage architecture and encryption implementation → Sensitive data encrypted at rest
2. Verify sensitive data encrypted before writing to storage → Strong encryption algorithm used (AES-256)
3. Test that encryption uses strong algorithms (AES-256-GCM or equivalent) → Keys stored separately from data
4. Verify encryption keys stored separately from encrypted data → Secure key management system
5. Test that encryption keys managed through secure key management system → Key rotation implemented
6. Verify key rotation procedures documented and implemented → Backups also encrypted
7. Test that backups also encrypted with appropriate key management → Key access requires authentication
8. Verify access to encryption keys requires authentication and authorization → Documentation comprehensive
9. Test that encryption at rest documented in security architecture → Regulatory compliance verified
10. Verify compliance with regulatory requirements (GDPR, applicable sector regulations) → Encryption covers all sensitive data types

Pass Criteria: AES-256 or equivalent encryption AND separate key storage AND key management system AND backup encryption

Fail Criteria: No encryption OR weak algorithms OR keys with data OR no key management

Evidence: Encryption architecture documentation, algorithm verification, key storage analysis, key management system review, backup encryption testing, compliance attestation

References:

- NIST Encryption Standards: https://csrc.nist.gov/publications/detail/sp/800-175b/final
- Data at Rest Encryption: https://cloud.google.com/security/encryption-at-rest
- Key Management Best Practices: https://csrc.nist.gov/publications/detail/sp/800-57-part-1/rev-5/final

Assessment: RDPS-REQ-17 (Mutual TLS authentication for RDPS)

Reference: RDPS-REQ-17 - Browser shall implement mutual TLS authentication for RDPS connections

Given: A conformant browser with RDPS-2 or higher capability

Task: Mutual TLS (mTLS) provides bidirectional authentication where both browser and RDPS server verify each other's identity through certificates, preventing unauthorized clients from accessing RDPS and unauthorized servers from impersonating RDPS. Standard TLS only authenticates the server, allowing any client to connect. mTLS with client certificates, certificate validation, and revocation checking ensures only authorized browsers access RDPS infrastructure.

Verification:

1. Verify browser configured with client certificate for RDPS authentication → Browser has valid client certificate
2. Test successful mTLS connection with valid client and server certificates → mTLS connection successful with both certs
3. Attempt connection without client certificate and verify RDPS rejects connection → Missing client certificate rejected
4. Test with expired client certificate and confirm connection rejected → Expired client certificates rejected
5. Verify client certificate validation enforced on RDPS side → RDPS validates client certificates
6. Test client certificate revocation checking (CRL or OCSP) → Revocation checking functional
7. Verify client certificate securely stored (encrypted, OS keychain) → Client certificate stored securely
8. Test client certificate renewal process → Renewal process documented
9. Verify server still validates client certificate chain (intermediates, root) → Full chain validation performed
10. Test that mTLS protects against man-in-the-middle even with compromised CA → Enhanced MITM protection

Pass Criteria: Client certificates configured AND mTLS enforced AND revocation checking AND secure certificate storage

Fail Criteria: No client certificates OR mTLS not enforced OR no revocation checking OR insecure storage

Evidence: mTLS configuration documentation, connection testing with various certificate states, revocation checking verification, certificate storage analysis, MITM attack prevention testing

References:

- Mutual TLS: https://www.cloudflare.com/learning/access-management/what-is-mutual-tls/
- RFC 8446 TLS 1.3: https://www.rfc-editor.org/rfc/rfc8446
- Client Certificate Authentication: https://docs.microsoft.com/en-us/azure/application-gateway/mutual-authentication-overview

Assessment: RDPS-REQ-18 (Redundant data copies for recovery)

Reference: RDPS-REQ-18 - Browser shall maintain redundant copies of critical data for recovery

Given: A conformant browser with RDPS-2 or higher capability

Task: Redundant data storage protects against data loss from hardware failures, corruption, ransomware, or operational errors by maintaining multiple synchronized copies across independent storage systems. Without redundancy, single points of failure can cause permanent data loss, service disruption, and user impact. Multi-region or multi-datacenter replication with consistency guarantees, automatic failover, and integrity verification ensures data availability and durability.

Verification:

1. Review RDPS architecture documentation for redundancy implementation → Critical data has multiple replicas
2. Verify critical data replicated to at least 2 independent storage systems → Replicas in independent failure domains
3. Test that replicas maintained in different failure domains (servers, racks, datacenters) → Replication mechanism documented
4. Verify replication synchronization mechanism (synchronous or asynchronous) → Data consistency maintained
5. Test data consistency between replicas → Automatic failover functional
6. Simulate primary storage failure and verify automatic failover to replica → Recovery from replica successful
7. Test data recovery from replica maintains integrity → Replication lag monitored
8. Verify replication lag monitored and alerted if excessive → Corruption detection and correction
9. Test that replica corruption detected and corrected → Geo-distribution implemented if required
10. Verify geo-distribution of replicas if required for disaster recovery → Recovery tested regularly

Pass Criteria: Multiple independent replicas AND different failure domains AND automatic failover AND consistency maintained

Fail Criteria: Single copy only OR replicas in same failure domain OR no failover OR consistency not guaranteed

Evidence: Architecture diagrams showing redundancy, replica configuration documentation, failover testing results, consistency verification, recovery procedure testing

References:

- Database Replication: https://en.wikipedia.org/wiki/Replication_(computing)
- AWS Multi-Region Architecture: https://aws.amazon.com/blogs/architecture/disaster-recovery-dr-architecture-on-aws-part-i-strategies-for-recovery-in-the-cloud/
- Data Redundancy Best Practices: https://cloud.google.com/architecture/dr-scenarios-planning-guide

Assessment: RDPS-REQ-19 (Data recovery from backups with integrity verification)

Reference: RDPS-REQ-19 - Browser shall support data recovery from backups with integrity verification

Given: A conformant browser with RDPS-2 or higher capability

Task: Backup recovery enables restoration from data corruption, accidental deletion, ransomware, or catastrophic failures by maintaining historical data snapshots with integrity guarantees. Without verified backups, recovery attempts may restore corrupted data, incomplete datasets, or tampered backups. Automated backup with encryption, integrity verification, recovery testing, and documented procedures ensures reliable data restoration when needed.

Verification:

1. Review backup strategy documentation (frequency, retention, scope) → Automated backups on schedule
2. Verify backups created automatically on defined schedule → Integrity verification implemented
3. Test backup integrity verification using checksums or cryptographic hashes → Backups encrypted at rest
4. Verify backups encrypted at rest with separate key management → All critical data backed up
5. Test backup completeness (all critical data included) → Recovery successful in simulation
6. Simulate data loss scenario and perform recovery from backup → Recovered data integrity verified
7. Verify recovered data integrity matches pre-loss state → Point-in-time recovery functional
8. Test point-in-time recovery to specific timestamp → Retention policy enforced
9. Verify backup retention policy enforced (old backups purged appropriately) → Recovery procedures documented
10. Test that recovery procedures documented and tested regularly → Regular recovery testing performed

Pass Criteria: Automated backups AND integrity verification AND successful recovery testing AND encryption at rest

Fail Criteria: Manual backups only OR no integrity verification OR recovery not tested OR unencrypted backups

Evidence: Backup strategy documentation, integrity verification logs, recovery test results, encryption verification, retention policy configuration

References:

- Backup and Recovery: https://csrc.nist.gov/publications/detail/sp/800-34/rev-1/final
- 3-2-1 Backup Rule: https://www.backblaze.com/blog/the-3-2-1-backup-strategy/
- Backup Integrity: https://www.sans.org/white-papers/36607/

Assessment: RDPS-REQ-20 (Data retention policies with secure deletion)

Reference: RDPS-REQ-20 - Browser shall implement data retention policies with secure deletion

Given: A conformant browser with RDPS-2 or higher capability

Task: Data retention policies ensure compliance with regulations (GDPR right to erasure, data minimization), reduce security exposure from storing unnecessary data, and manage storage costs. Without enforced retention and secure deletion, RDPS accumulates excessive personal data, violates privacy regulations, and creates larger breach targets. Automated retention with secure multi-pass deletion, deletion verification, and audit logging ensures compliant data lifecycle management.

Verification:

1. Review data retention policy documentation for all RDPS data types → Retention policies documented comprehensively
2. Verify retention periods defined per data classification and regulatory requirements → Retention periods per data type defined
3. Test automated deletion after retention period expires → Automated deletion implemented

4. Verify secure deletion prevents data recovery (multi-pass overwrite or cryptographic erasure) → Secure deletion prevents recovery
5. Test that deletion requests from users processed within regulatory timeframes → User deletion requests honored timely
6. Verify deletion confirmation provided to users → Deletion confirmation provided
7. Test that deleted data removed from backups per retention policy → Backups also cleaned per policy
8. Verify deletion logged for audit and compliance purposes → Deletion audit trail maintained
9. Test that related data (indexes, caches, logs) also deleted → Related data deleted completely
10. Verify regulatory compliance (GDPR Article 17, CCPA) demonstrated → Regulatory compliance verified

Pass Criteria: Retention policies defined AND automated deletion AND secure erasure AND audit logging

Fail Criteria: No retention policies OR manual deletion only OR recoverable after deletion OR no audit trail

Evidence: Retention policy documentation, automated deletion verification, secure erasure testing, deletion audit logs, regulatory compliance attestation

References:

- GDPR Right to Erasure: https://gdpr-info.eu/art-17-gdpr/
- NIST Data Sanitization: https://csrc.nist.gov/publications/detail/sp/800-88/rev-1/final
- Secure Data Deletion: https://www.usenix.org/legacy/event/fast11/tech/full_papers/Wei.pdf

Assessment: RDPS-REQ-21 (Per-user per-origin access controls)

Reference: RDPS-REQ-21 - Browser shall enforce access controls on RDPS data per-user and per-origin

Given: A conformant browser with RDPS-2 or higher capability

Task: Granular access controls prevent unauthorized data access across user boundaries and origin boundaries, enforcing multi-tenancy isolation and same-origin policy in RDPS storage. Without strict access controls, compromised users or origins can access other users' data, violating privacy and security. Role-based access control (RBAC) or attribute-based access control (ABAC) with authentication, authorization, and audit trails ensures proper data isolation.

Verification:

1. Review RDPS access control model (RBAC, ABAC, or equivalent) → Access control model documented
2. Test that User A cannot access User B's RDPS data → Cross-user access blocked
3. Verify origin-based isolation (origin A cannot access origin B's data) → Cross-origin access prevented
4. Test authentication required before any RDPS data access → Authentication enforced universally
5. Verify authorization checks performed for every data operation → Authorization per-operation checked
6. Test access control enforcement at database/storage layer (not just application) → Storage-layer enforcement implemented
7. Verify access control bypass attempts logged and blocked → Bypass attempts logged and blocked
8. Test that administrators have appropriate elevated access with audit logging → Administrator access audited
9. Verify principle of least privilege applied (minimal necessary access granted) → Least privilege enforced
10. Test access controls survive privilege escalation attempts → Privilege escalation prevented

Pass Criteria: Per-user isolation enforced AND per-origin isolation enforced AND storage-layer controls AND bypass prevention

Fail Criteria: Cross-user access possible OR cross-origin access allowed OR application-only controls OR no bypass detection

Evidence: Access control model documentation, cross-user access testing, cross-origin isolation verification, privilege escalation testing, audit logs

References:

- RBAC: https://csrc.nist.gov/projects/role-based-access-control
- OWASP Access Control: https://cheatsheetseries.owasp.org/cheatsheets/Authorization_Cheat_Sheet.html
- Multi-Tenancy Security: https://www.microsoft.com/en-us/research/publication/multi-tenant-databases-for-software-as-a-service/

Assessment: RDPS-REQ-22 (RDPS access and modification auditing)

Reference: RDPS-REQ-22 - Browser shall audit all RDPS access and modifications

Given: A conformant browser with RDPS-2 or higher capability

Task: Comprehensive audit logging of RDPS operations enables security incident detection, forensic investigation, compliance verification, and insider threat monitoring. Without detailed audit trails, unauthorized access goes undetected, data breaches can't be traced, and compliance audits fail. Tamper-resistant audit logs with who, what, when, where details provide accountability and support security operations.

Verification:

1. Review audit logging implementation for RDPS operations → All read operations logged
2. Verify all data read operations logged with user, origin, timestamp, data type → Write operations logged with changes
3. Test that all write/modify operations logged with before/after values → Delete operations documented
4. Verify delete operations logged with deleted data reference → Failed attempts captured
5. Test that failed access attempts logged (authentication, authorization failures) → Administrative actions logged
6. Verify administrative operations logged with elevated privilege markers → Logs tamper-protected
7. Test audit log tamper protection (append-only, cryptographic signatures) → Retention requirements met
8. Verify audit logs retained per compliance requirements → SIEM export functional
9. Test audit log export for SIEM integration → Forensic detail sufficient
10. Verify audit logs include sufficient detail for forensic investigation → Audit log performance acceptable

Pass Criteria: All operations logged AND tamper protection AND retention compliance AND forensic detail sufficient

Fail Criteria: Operations not logged OR logs can be tampered OR insufficient retention OR missing critical details

Evidence: Audit log samples, tamper protection verification, retention policy documentation, SIEM integration testing, forensic analysis demonstration

References:

- Audit Logging: https://csrc.nist.gov/publications/detail/sp/800-92/final
- OWASP Logging Cheat Sheet: https://cheatsheetseries.owasp.org/cheatsheets/Logging_Cheat_Sheet.html
- Tamper-Evident Logging: https://www.usenix.org/legacy/event/sec09/tech/full_papers/waters.pdf

Assessment: RDPS-REQ-23 (Data integrity verification using cryptographic hashes)

Reference: RDPS-REQ-23 - Browser shall implement data integrity verification using cryptographic hashes

Given: A conformant browser with RDPS-2 or higher capability

Task: Cryptographic integrity verification detects unauthorized modifications, corruption, or tampering of RDPS data by comparing current hash values against stored reference hashes. Without integrity verification, silent data corruption, malicious modifications, or storage system failures go undetected until data loss occurs. Continuous integrity monitoring with strong hash algorithms (SHA-256), automatic verification, and corruption alerts ensures data trustworthiness.

Verification:

1. Review integrity verification implementation using cryptographic hashes → Strong cryptographic hash algorithm used
2. Verify hash algorithm is cryptographically strong (SHA-256, SHA-3, or better) → Hashes computed at write time
3. Test that hashes computed at data write and stored securely → Periodic verification performed
4. Verify periodic integrity verification checks data against stored hashes → Modifications update hashes correctly
5. Test that data modifications update hash values appropriately → Mismatches detected and alerted
6. Verify hash mismatches detected and alerted → Corruption detection functional
7. Test corruption detected through integrity verification → Automated remediation implemented
8. Verify automated remediation (restore from backup) on corruption detection → Hashes stored separately from data
9. Test that hash values stored separately from data (prevent correlated corruption) → All critical data covered
10. Verify integrity verification covers all critical data types → Performance impact acceptable

Pass Criteria: SHA-256 or stronger hashes AND periodic verification AND mismatch detection AND automated remediation

Fail Criteria: Weak hash algorithms OR no verification OR mismatches not detected OR no remediation

Evidence: Hash algorithm documentation, integrity verification configuration, corruption detection testing, remediation procedure verification, performance analysis

References:

- Cryptographic Hash Functions: https://csrc.nist.gov/projects/hash-functions
- Data Integrity: https://csrc.nist.gov/glossary/term/data_integrity
- SHA-256: https://en.wikipedia.org/wiki/SHA-2

Assessment: RDPS-REQ-24 (RDPS endpoint substitution attack protection)

Reference: RDPS-REQ-24 - Browser shall protect against RDPS endpoint substitution attacks

Given: A conformant browser with RDPS-2 or higher capability

Task: Endpoint substitution attacks occur when attackers redirect browser-RDPS communications to malicious servers through DNS poisoning, ARP spoofing, or configuration tampering, enabling data interception and manipulation. Without endpoint validation, browsers communicate with attacker-controlled servers believing they're legitimate RDPS. Certificate pinning, endpoint verification, domain validation, and secure configuration storage prevent endpoint substitution and ensure browsers connect only to legitimate RDPS infrastructure.

Verification:

1. Review RDPS endpoint configuration storage and protection mechanisms → Endpoints stored tamper-resistant
2. Verify endpoints stored in tamper-resistant configuration (encrypted, integrity-protected) → Modifications require authentication
3. Test that endpoint modifications require authentication and authorization → Certificate pinning enforced
4. Verify certificate pinning prevents connection to substituted endpoints with different certificates → DNS poisoning detected and blocked
5. Test DNS poisoning attack and verify browser detects and blocks malicious endpoint → Domain validation performed
6. Verify domain validation performed before establishing RDPS connections → Configuration changes logged
7. Test that endpoint configuration changes logged for audit → Fallback endpoints validated

8. Verify fallback endpoints also validated and pinned → MITM with substitution detected
9. Test that man-in-the-middle attacks with endpoint substitution detected → User notifications for security events
10. Verify users notified of endpoint changes or security warnings → Endpoint integrity maintained

Pass Criteria: Tamper-resistant endpoint storage AND certificate pinning AND domain validation AND MITM detection

Fail Criteria: Endpoints easily modified OR no certificate pinning OR no domain validation OR MITM undetected

Evidence: Endpoint configuration security analysis, certificate pinning verification, DNS poisoning test results, MITM attack prevention testing, audit logs

References:

- DNS Security: https://www.icann.org/resources/pages/dnssec-what-is-it-why-important-2019-03-05-en
- Certificate Pinning: https://owasp.org/www-community/controls/Certificate_and_Public_Key_Pinning
- Endpoint Security: https://cheatsheetseries.owasp.org/cheatsheets/Transport_Layer_Protection_Cheat_Sheet.html

Assessment: RDPS-REQ-25 (Replay attack defense for RDPS communications)

Reference: RDPS-REQ-25 - Browser shall implement defense against replay attacks on RDPS communications

Given: A conformant browser with RDPS-2 or higher capability

Task: Replay attacks allow attackers who intercept RDPS communications to re-transmit captured requests, potentially duplicating transactions, bypassing authentication, or causing unintended operations. Without replay protection, encrypted traffic remains vulnerable to replay even when confidentiality is maintained. Nonces, timestamps, sequence numbers, and request signing with replay detection prevent attackers from reusing captured network traffic.

Verification:

1. Review RDPS communication protocol for replay protection mechanisms → Nonces or unique IDs in requests
2. Verify requests include nonces (numbers used once) or unique request IDs → Duplicate nonces rejected
3. Test that server rejects duplicate nonces within time window → Timestamps validated server-side
4. Verify timestamps included in requests with server-side validation → Expired requests rejected
5. Test that expired timestamps rejected (e.g., requests older than 5 minutes) → Sequence number tracking implemented
6. Verify sequence numbers tracked to detect out-of-order or duplicate requests → Replay attacks detected in testing
7. Test replay attack by capturing and retransmitting valid request → Replayed requests rejected
8. Verify replayed request detected and rejected by RDPS → Detection events logged
9. Test that replay detection logged for security monitoring → Cryptographic binding prevents tampering
10. Verify cryptographic signatures bind requests to prevent modification during replay → Replay window appropriately sized

Pass Criteria: Nonces or timestamps enforced AND duplicate detection AND replay attacks blocked AND security logging

Fail Criteria: No replay protection OR duplicates accepted OR replay attacks succeed OR no logging

Evidence: Protocol documentation showing replay protection, nonce/timestamp implementation review, replay attack test results, detection logging verification

References:

- Replay Attack Prevention: https://en.wikipedia.org/wiki/Replay_attack

- Nonce: https://en.wikipedia.org/wiki/Cryptographic_nonce
- OWASP Session Management: https://cheatsheetseries.owasp.org/cheatsheets/Session_Management_Cheat_Sheet.html

Assessment: RDPS-REQ-26 (Data minimization principles for RDPS transmissions)

Reference: RDPS-REQ-26 - Browser shall enforce data minimization principles for RDPS transmissions

Given: A conformant browser with RDPS-2 or higher capability

Task: Data minimization limits RDPS data collection and transmission to only what is strictly necessary for functionality, reducing privacy risks, regulatory exposure, and breach impact. Without minimization, excessive data collection creates compliance violations (GDPR Article 5), larger attack surfaces, and user privacy concerns. Purpose limitation, necessity assessment, and data filtering ensure only required data transmitted to RDPS.

Verification:

1. Review RDPS data inventory and verify necessity justification for each data type → All transmitted data justified as necessary
2. Test that browser transmits only functionally required data to RDPS → No excessive data collection
3. Verify no excessive or speculative data collection occurs → PII minimized or pseudonymized
4. Test that personal identifiable information (PII) minimized or pseudonymized → Aggregation/anonymization applied
5. Verify data aggregation or anonymization applied where possible → Users can review transmitted data
6. Test that users can review what data transmitted to RDPS → Minimization documented publicly
7. Verify data minimization documented in privacy policy and data processing agreements → Unnecessary fields removed
8. Test that unused or unnecessary data fields removed from transmissions → Retention minimized
9. Verify data retention limited to minimum required period → GDPR compliance demonstrated
10. Test compliance with GDPR Article 5 data minimization principle → Regular minimization audits performed

Pass Criteria: Necessity justification for all data AND PII minimization AND user transparency AND GDPR compliance

Fail Criteria: Excessive data collected OR no justification OR PII not minimized OR GDPR non-compliance

Evidence: Data inventory with necessity justifications, PII minimization verification, user transparency features, GDPR compliance documentation, privacy policy review

References:

- GDPR Article 5 Data Minimization: https://gdpr-info.eu/art-5-gdpr/
- Data Minimization Best Practices: https://ico.org.uk/for-organisations/guide-to-data-protection/guide-to-the-general-data-protection-regulation-gdpr/principles/data-minimisation/

Assessment: RDPS-REQ-27 (User controls for RDPS data synchronization)

Reference: RDPS-REQ-27 - Browser shall provide user controls for RDPS data synchronization

Given: A conformant browser with RDPS-2 or higher capability

Task: User control over RDPS synchronization respects user autonomy, enables privacy management, and supports varied connectivity scenarios where users may prefer local-only operation. Without controls, users cannot disable synchronization for privacy, bandwidth conservation, or regulatory compliance (e.g., data localization requirements). Granular synchronization controls with clear UI, immediate effect, and data deletion options empower users to manage their data lifecycle.

Verification:

1. Verify browser provides UI for enabling/disabling RDPS synchronization → Global synchronization toggle available
2. Test that synchronization can be disabled globally → Per-data-type controls functional
3. Verify per-data-type synchronization controls (e.g., sync preferences but not history) → Changes take effect immediately
4. Test that disabling synchronization takes effect immediately → Data deletion option provided
5. Verify existing RDPS data optionally deletable when disabling synchronization → Settings persist across restarts
6. Test that synchronization state persists across browser restarts → UI accessible and discoverable
7. Verify synchronization settings accessible and discoverable in browser UI → Status clearly indicated
8. Test that synchronization status clearly indicated to users → Re-enable requires explicit action
9. Verify re-enabling synchronization requires explicit user action → Controls well-documented
10. Test that synchronization controls documented in user documentation → User autonomy respected

Pass Criteria: Global and per-type controls AND immediate effect AND data deletion option AND clear UI

Fail Criteria: No user controls OR changes delayed OR no deletion option OR hidden UI

Evidence: UI screenshots showing synchronization controls, settings persistence testing, data deletion verification, user documentation review

References:

- User Privacy Controls: https://www.w3.org/TR/privacy-controls/
- GDPR Right to Data Portability: https://gdpr-info.eu/art-20-gdpr/
- Privacy UX Best Practices: https://www.smashingmagazine.com/2019/04/privacy-ux-better-notifications-permission-requests/

Assessment: RDPS-REQ-28 (Secure data export from RDPS for data portability)

Reference: RDPS-REQ-28 - Browser shall implement secure data export from RDPS for data portability

Given: A conformant browser with RDPS-2 or higher capability

Task: Data portability enables users to export their RDPS data for backup, migration to alternative browsers, or regulatory compliance (GDPR Article 20), preventing vendor lock-in and respecting user data ownership. Without export capability, users cannot recover data, switch browsers, or exercise portability rights. Secure export with authentication, encryption, integrity verification, and standard formats empowers users while preventing unauthorized data extraction.

Verification:

1. Verify browser provides data export functionality for RDPS data → Export functionality available
2. Test that export requires user authentication before proceeding → Authentication required
3. Verify exported data includes all user-accessible RDPS data → All accessible data exported
4. Test export format is standard and machine-readable (JSON, XML, CSV) → Standard format used
5. Verify exported data integrity protected (checksums, signatures) → Integrity protection applied
6. Test that export encrypted during download if sensitive → Encryption during download
7. Verify export logged for audit and security monitoring → Export operations logged
8. Test export rate limiting to prevent abuse or data scraping → Rate limiting prevents abuse
9. Verify export includes metadata (timestamps, data types, sources) → Metadata included
10. Test that exported data importable to another browser (interoperability) → Cross-browser compatibility

Pass Criteria: Export functionality AND authentication required AND standard format AND integrity protection

Fail Criteria: No export capability OR no authentication OR proprietary format OR no integrity protection

Evidence: Export functionality demonstration, authentication verification, format documentation, integrity verification testing, interoperability testing

References:

- GDPR Data Portability: https://gdpr-info.eu/art-20-gdpr/
- Data Liberation: https://en.wikipedia.org/wiki/Google_Data_Liberation_Front
- JSON Format: https://www.json.org/json-en.html

Assessment: RDPS-REQ-29 (User-configurable RDPS endpoint security settings)

Reference: RDPS-REQ-29 - When user-configurable RDPS endpoints provided, all associated security settings shall be configurable

Given: A conformant browser with RDPS-2 or higher capability supporting user-configurable endpoints

Task: When browsers allow users to configure custom RDPS endpoints (e.g., enterprise deployments, self-hosted infrastructure), all security settings shall be configurable to maintain security equivalent to default RDPS. Without configurable security settings, users cannot enforce certificate pinning, encryption requirements, or authentication for custom endpoints, creating security gaps. Comprehensive security configuration with validation, documentation, and testing ensures custom endpoints maintain security posture.

Verification:

1. Verify browser supports user-configurable RDPS endpoints → Custom endpoint configuration available
2. Test that users can configure custom endpoint URLs → Certificate pinning for custom endpoints
3. Verify certificate pinning configurable for custom endpoints → TLS requirements configurable
4. Test that TLS version requirements configurable (enforce TLS 1.3+) → Authentication methods selectable
5. Verify authentication method configurable (OAuth, mTLS, API keys) → Timeout configuration available
6. Test that connection timeout values configurable → Retry parameters adjustable
7. Verify retry and backoff parameters configurable → Encryption settings configurable
8. Test that data encryption settings configurable → Secure defaults enforced
9. Verify all security settings have secure defaults → Insecure configs warned/blocked
10. Test that insecure configurations warned or blocked → Configuration documented

Pass Criteria: Comprehensive security configuration AND secure defaults AND insecure config warnings AND documentation complete

Fail Criteria: Limited configuration OR no defaults OR insecure configs allowed OR poor documentation

Evidence: Configuration UI documentation, security settings verification, secure defaults testing, insecure configuration prevention testing

References:

- Enterprise Configuration: https://chromeenterprise.google/policies/
- Secure Configuration: https://www.cisecurity.org/cis-benchmarks/
- Configuration Management: https://csrc.nist.gov/glossary/term/configuration_management

Assessment: RDPS-REQ-30 (RDPS service availability verification before critical operations)

Reference: RDPS-REQ-30 - Browser shall verify RDPS service availability before critical operations

Given: A conformant browser with RDPS-2 or higher capability performing critical operations dependent on remote data processing

Task: This assessment verifies that browsers check RDPS service availability before executing critical operations (authentication synchronization, encryption key retrieval, security policy validation) that depend on remote data processing. The assessment confirms that availability checks include timeout controls, the browser safely defers or fails operations when the service is unavailable, and users receive appropriate notifications.

Verification:

1. Identify all browser operations classified as critical and dependent on RDPS → Availability verification implemented for all critical operations
2. Test browser with RDPS fully available and responsive → Health check endpoints or APIs used for verification
3. Verify critical operations check service availability before execution → Timeout controls implemented (5-10 seconds)
4. Test browser with RDPS temporarily unavailable (network disconnected) → Retry logic with exponential backoff present
5. Verify browser detects unavailability before attempting critical operations → Circuit breaker pattern implemented to prevent cascade failures
6. Test browser with RDPS responding slowly (high latency) → Browser defers operations when service unavailable
7. Verify availability checks respect timeout limits (5-10 seconds typical) → Safe failure with error messaging when operation cannot be deferred
8. Test browser with RDPS returning unhealthy status codes (503, 500) → User notifications provided for deferred operations
9. Verify browser treats unhealthy responses as unavailable → Cached data used with integrity verification when applicable
10. Test that browser defers or safely fails critical operations when RDPS unavailable → Manual retry options available to users

Pass Criteria: Availability verification for all critical operations AND health checks implemented AND timeout controls (5-10s) AND deferred/safe failure when unavailable AND user notifications AND documentation complete

Fail Criteria: No availability verification OR critical operations attempted without checks OR no timeout controls OR unsafe failures OR no user notifications OR incomplete documentation

Evidence: Critical operations catalog, availability verification code review, health check endpoint documentation, timeout configuration, unavailability testing results, user notification examples, circuit breaker implementation

References:

- Health Check API Pattern: https://microservices.io/patterns/observability/health-check-api.html
- Circuit Breaker Pattern: https://martinfowler.com/bliki/CircuitBreaker.html
- Service Availability: https://sre.google/sre-book/availability-table/

Assessment: RDPS-REQ-31 (Connection pooling with security controls for RDPS)

Reference: RDPS-REQ-31 - Browser shall implement connection pooling with security controls for RDPS

Given: A conformant browser with RDPS-2 or higher capability implementing connection pooling for remote data processing systems

Task: This assessment verifies that browsers implement secure connection pooling for RDPS communications with per-user per-origin isolation, connection validation before reuse, eviction policies, pool size limits, and protection against connection smuggling attacks.

Verification:

1. Verify browser implements connection pooling for RDPS communications → Per-user per-origin connection pools implemented
2. Test that connections are pooled per-user and per-origin (no cross-user sharing) → Authentication state maintained per connection
3. Verify each pooled connection maintains separate authentication state → No credential sharing across contexts

4. Test that credentials never leak between different security contexts → Connection validation before reuse
5. Verify browser validates connections before reuse (certificate, TLS state) → Eviction policies implemented (idle timeout, max lifetime)
6. Test connection eviction policies (idle timeout, max lifetime, max request count) → Pool size limits enforced (6-8 per origin typical)
7. Verify connection pool size limits enforced (prevent resource exhaustion) → Stale connection detection and eviction
8. Test that browser detects and evicts stale or unhealthy connections → Connection smuggling attack prevention
9. Verify protection against connection smuggling attacks → Enterprise configuration options available
10. Test that enterprise administrators can configure or disable pooling → Connection pooling documented with security considerations

Pass Criteria: Per-user per-origin pooling AND authentication isolation AND connection validation before reuse AND eviction policies AND pool size limits AND smuggling prevention AND enterprise controls AND documentation complete

Fail Criteria: Cross-user connection sharing OR credential leakage OR no connection validation OR no eviction policies OR unlimited pool size OR smuggling vulnerabilities OR no enterprise controls OR incomplete documentation

Evidence: Connection pooling implementation code review, per-user per-origin isolation testing, credential isolation verification, connection validation testing, eviction policy configuration, pool size limit testing, connection smuggling attack testing, enterprise configuration documentation

References:

- HTTP Connection Management: https://developer.mozilla.org/en-US/docs/Web/HTTP/Connection_management_in_
- Connection Pooling Security: https://owasp.org/www-community/vulnerabilities/Insecure_Transport
- Request Smuggling: https://portswigger.net/web-security/request-smuggling

Assessment: RDPS-REQ-32 (RDPS authentication token protection from extraction and theft)

Reference: RDPS-REQ-32 - Browser shall protect RDPS authentication tokens from extraction and theft

Given: A conformant browser with RDPS-2 or higher capability storing authentication tokens for remote data processing systems

Task: This assessment verifies that browsers protect RDPS authentication tokens using OS-provided secure storage with hardware-backed encryption when available, implement token lifetime management and automatic rotation, prevent plaintext storage and memory extraction, and support remote token revocation by users and policy enforcement by administrators.

Verification:

1. Identify all RDPS authentication token types stored by browser → OS secure storage used for all token types
2. Verify tokens stored using OS secure storage mechanisms (Keychain, Credential Manager, Secret Service) → Hardware-backed encryption when available
3. Test that tokens encrypted with hardware-backed keys when available (TPM, Secure Enclave) → No plaintext token storage
4. Verify tokens never stored in plaintext on disk → Web content cannot access tokens
5. Test that tokens not accessible to web content via standard storage APIs → Memory protection implemented
6. Verify memory protection prevents token extraction via dumps or debugging → Token lifetime management present
7. Test token lifetime management (short access tokens, longer refresh tokens) → Automatic token rotation implemented

8. Verify automatic token rotation before expiration → Secure revocation on logout
9. Test secure token revocation on logout → Tokens never in URLs or HTTP headers
10. Verify token transmission uses TLS 1.3+ with certificate pinning → Remote token revocation interface available

Pass Criteria: OS secure storage for all tokens AND hardware-backed encryption when available AND no plaintext storage AND web content isolation AND memory protection AND token lifetime management AND automatic rotation AND secure revocation AND TLS 1.3+ transmission AND remote revocation UI AND documentation complete

Fail Criteria: Plaintext token storage OR web content access to tokens OR no memory protection OR no lifetime management OR no token rotation OR insecure revocation OR tokens in URLs OR no remote revocation OR incomplete documentation

Evidence: Token storage implementation code review, OS secure storage verification, hardware-backed encryption testing, web content isolation testing, memory protection verification, token lifetime configuration, rotation mechanism testing, revocation testing, TLS transmission verification, remote revocation UI documentation

References:

- OAuth 2.0 Security Best Practices: https://datatracker.ietf.org/doc/html/draft-ietf-oauth-security-topics
- Hardware-Backed Keystore: https://developer.android.com/training/articles/keystore

Assessment: RDPS-REQ-33 (End-to-end encryption for all critical data in RDPS)

Reference: RDPS-REQ-33 - Browser shall implement end-to-end encryption for all critical data in RDPS (RDPS-3 requirement)

Given: A conformant browser with RDPS-3 capability processing critical data through remote data processing systems

Task: This assessment verifies that browsers implement end-to-end encryption for all critical RDPS data with client-side key generation, authenticated encryption before transmission, server-side inability to decrypt, secure key backup and recovery mechanisms, metadata protection, and user education about E2EE trade-offs.

Verification:

1. Identify all critical data processed through RDPS by the browser → Client-side encryption key generation implemented
2. Verify encryption keys generated locally on client device using CSPRNG → Strong KDF used for key derivation (high iteration count)
3. Test that keys derived from user credentials use strong KDF (PBKDF2, Argon2, scrypt) → Authenticated encryption for all critical data
4. Verify all critical data encrypted on client before transmission → Data remains encrypted at rest remotely
5. Test that authenticated encryption algorithms used (AES-256-GCM, ChaCha20-Poly1305) → Server cannot decrypt data (verified via architecture review)
6. Verify encrypted data remains encrypted at rest in RDPS storage → Secure key backup/recovery with E2EE guarantees
7. Test that RDPS provider cannot decrypt data (server-side decryption impossible) → Client-side key rotation capability
8. Verify key backup and recovery mechanisms maintain E2EE guarantees → Metadata protection implemented
9. Test key rotation without server involvement → User education about E2EE trade-offs
10. Verify metadata protection implemented (encrypted filenames, padding, obfuscation) → Data loss warnings for lost keys without recovery

Pass Criteria: Client-side key generation AND strong KDF AND authenticated encryption for all critical data AND encrypted at rest AND server cannot decrypt AND secure key backup/recovery AND key rotation AND metadata protection AND user education AND documentation complete

Fail Criteria: Server-side key generation OR weak KDF OR unauthenticated encryption OR plaintext at rest OR server can decrypt OR insecure key backup OR no key rotation OR metadata leakage OR no user education OR incomplete documentation

Evidence: E2EE implementation architecture review, key generation code review, KDF configuration, encryption algorithm verification, server-side decryption impossibility proof, key backup/recovery documentation, key rotation testing, metadata protection verification, user interface documentation

References:

- End-to-End Encryption: https://www.w3.org/TR/webauthn-3/#end-to-end-encryption
- Signal Protocol: https://signal.org/docs/
- Authenticated Encryption: https://csrc.nist.gov/projects/block-cipher-techniques/bcm/modes-development

Assessment: RDPS-REQ-34 (Hardware-backed key storage for RDPS encryption keys)

Reference: RDPS-REQ-34 - Browser shall use hardware-backed key storage for RDPS encryption keys (RDPS-3 requirement)

Given: A conformant browser with RDPS-3 capability on platforms supporting hardware-backed key storage (TPM, Secure Enclave, TEE, HSM)

Task: This assessment verifies that browsers use hardware security modules (TPM, Secure Enclave, TrustZone, HSM) for RDPS encryption keys when available, with non-exportable device-bound keys, authentication requirements, secure hardware operations, key attestation support, software fallback warnings, and enterprise policy enforcement for hardware requirements.

Verification:

1. Identify platforms with hardware security modules (TPM, Secure Enclave, TrustZone, HSM) → Hardware security module detection implemented
2. Verify browser detects available hardware security features on each platform → Preferential use of hardware-backed storage
3. Test that browser preferentially uses hardware-backed storage when available → Non-exportable keys bound to device
4. Verify keys stored in hardware are non-exportable (bound to device) → Authentication required for key usage
5. Test that authentication required for hardware key usage (PIN, biometric) → Cryptographic operations in secure hardware
6. Verify cryptographic operations performed inside secure hardware → Key material never exposed to software
7. Test that key material never exposed to OS or browser process → Key attestation implemented
8. Verify key attestation provides proof of hardware-backed storage → Software fallback with security warnings
9. Test fallback to software storage shows clear security warnings → Enterprise policy enforcement for hardware requirement
10. Verify enterprise policies can require hardware-backed storage → Platform compatibility and limitations documented

Pass Criteria: Hardware detection on all supported platforms AND preferential hardware usage AND non-exportable keys AND authentication required AND operations in secure hardware AND key attestation AND fallback warnings AND enterprise policies AND documentation complete

Fail Criteria: No hardware detection OR no preferential hardware usage OR exportable keys OR no authentication OR operations in software OR no attestation OR silent fallback OR no enterprise policies OR incomplete documentation

Evidence: Hardware security module detection code review, key storage implementation verification, non-exportability testing, authentication requirement verification, secure hardware operation confirmation, attestation testing, fallback warning examples, enterprise policy documentation, platform compatibility matrix

References:

- TPM 2.0: https://trustedcomputinggroup.org/resource/tpm-library-specification/
- Apple Secure Enclave: https://support.apple.com/guide/security/secure-enclave-sec59b0b31ff/web
- Web Authentication API: https://www.w3.org/TR/webauthn-3/

Assessment: RDPS-REQ-35 (High-availability RDPS architecture with failover)

Reference: RDPS-REQ-35 - Browser shall implement high-availability RDPS architecture with failover (RDPS-3 requirement)

Given: A conformant browser with RDPS-3 capability requiring high availability for critical remote data processing operations

Task: This assessment verifies that browsers implement high-availability RDPS architecture with geographically distributed endpoints, automatic failover within 30-60 seconds, load balancing, session state preservation, graceful degradation for partial failures, and documented availability targets with MTTR specifications.

Verification:

1. Verify browser configuration supports multiple RDPS endpoints (primary/secondary) → Multiple RDPS endpoints configured (primary/secondary/tertiary)
2. Test that endpoints geographically distributed across regions → Geographic distribution across regions
3. Verify health monitoring detects endpoint failures → Health monitoring and failure detection
4. Test automatic failover switches to backup endpoint within 30-60 seconds → Failover within acceptable timeframe (30-60s)
5. Verify failed operations retried automatically without user intervention → Automatic operation retry
6. Test load balancing distributes traffic across healthy endpoints → Load balancing across healthy endpoints
7. Verify connection fallback sequence (primary → secondary → tertiary) → Session state preservation during failover
8. Test that session state preserved during failover → Graceful handling of partial failures
9. Verify partial failures handled gracefully (degraded service vs total failure) → Failover event logging and user notification
10. Test automatic recovery to primary endpoint after restoration → Availability targets and MTTR documented

Pass Criteria: Multiple geographically distributed endpoints AND health monitoring AND automatic failover (30-60s) AND operation retry AND load balancing AND session preservation AND graceful degradation AND automatic recovery AND logging/notifications AND availability targets documented

Fail Criteria: Single endpoint OR same geographic region OR no health monitoring OR manual failover OR no retry OR no load balancing OR session loss OR catastrophic failures OR no logging OR unavailability targets undocumented

Evidence: RDPS endpoint configuration documentation, geographic distribution verification, health monitoring implementation, failover timing tests, retry mechanism verification, load balancing testing, session preservation testing, partial failure handling tests, recovery testing, availability metrics and MTTR documentation

References:

- Failover Design Patterns: https://docs.microsoft.com/en-us/azure/architecture/patterns/category/resiliency
- SLA Targets: https://sre.google/sre-book/availability-table/

Assessment: RDPS-REQ-36 (RDPS disaster recovery procedures documentation and testing)

Reference: RDPS-REQ-36 - Browser shall document and test RDPS disaster recovery procedures (RDPS-3 requirement)

Given: A conformant browser with RDPS-3 capability requiring disaster recovery capabilities for critical remote data processing

Task: This assessment verifies that browsers have documented and regularly tested (annually minimum) disaster recovery procedures for RDPS with defined RTO/RPO targets, backup strategies, restoration procedures, alternative processing sites, role assignments, and validation across multiple disaster scenarios with documented improvements.

Verification:

1. Verify comprehensive DR procedures documented for RDPS infrastructure → Comprehensive DR documentation exists
2. Test that RTO and RPO defined for all critical RDPS components → RTO and RPO defined for critical components
3. Verify backup strategies documented (full, incremental, continuous replication) → Backup strategies documented (frequency, type, retention)
4. Test that restoration procedures provide step-by-step instructions → Step-by-step restoration procedures
5. Verify roles and responsibilities defined for disaster recovery → Roles and responsibilities clearly defined
6. Test that alternative processing sites identified and documented → Alternative processing sites identified
7. Verify backup frequency and retention periods specified → Regular DR testing performed (annual minimum, quarterly preferred)
8. Test that DR procedures tested at least annually (preferably quarterly) → Multiple disaster scenarios tested
9. Verify various disaster scenarios tested (datacenter outage, data corruption, breach) → Actual RTO/RPO measured and compared to targets
10. Test that actual RTO/RPO measured against targets during testing → Test results documented with improvements implemented

Pass Criteria: Comprehensive DR documentation AND defined RTO/RPO AND backup strategies AND restoration procedures AND regular testing (annual minimum) AND multiple scenarios tested AND measured RTO/RPO AND improvements documented AND user communication AND documentation complete

Fail Criteria: No DR documentation OR undefined RTO/RPO OR no backup strategy OR no restoration procedures OR no regular testing OR single scenario only OR unmeasured RTO/RPO OR no improvements OR no user communication OR incomplete documentation

Evidence: DR procedure documentation, RTO/RPO specifications, backup strategy documentation, restoration procedure manuals, DR test schedules and reports, test scenario documentation, RTO/RPO measurement results, improvement tracking, user communication materials

References:

- Disaster Recovery Planning: https://csrc.nist.gov/publications/detail/sp/800-34/rev-1/final
- Business Continuity: https://www.iso.org/standard/75106.html
- Backup and Recovery: https://aws.amazon.com/disaster-recovery/

Assessment: RDPS-REQ-37 (Real-time RDPS integrity monitoring)

Reference: RDPS-REQ-37 - Browser shall implement real-time RDPS integrity monitoring (RDPS-3 requirement)

Given: A conformant browser with RDPS-3 capability requiring continuous integrity monitoring of remote data processing systems

Task: This assessment verifies that browsers implement real-time RDPS data integrity monitoring using cryptographic verification (HMAC, signatures, Merkle trees), immutable audit trails, automated real-time violation detection and alerting, protected monitoring infrastructure, and comprehensive coverage across all RDPS components with user/administrator notifications.

Verification:

1. Verify cryptographic integrity verification implemented (HMAC, signatures, Merkle trees) → Cryptographic integrity verification implemented
2. Test that integrity checks cover all critical RDPS data (user data, configs, logs) → All critical data covered by integrity monitoring
3. Verify integrity monitoring runs continuously or at high frequency (minutes to hours) → Continuous or high-frequency monitoring (minutes to hours)
4. Test that immutable audit trails prevent retroactive tampering → Immutable audit trails implemented
5. Verify integrity violations detected in real-time or near-real-time (minutes) → Real-time or near-real-time detection (within minutes)
6. Test that security alerts generated automatically upon detection → Automated security alerts generated
7. Verify incident response procedures triggered for integrity violations → Incident response integration
8. Test that integrity monitoring infrastructure protected from compromise → Monitoring infrastructure protected from compromise
9. Verify comprehensive monitoring across all RDPS components → Comprehensive coverage of all RDPS components
10. Test that users and administrators notified of integrity violations → User and administrator notifications with status dashboards

Pass Criteria: Cryptographic integrity verification AND comprehensive data coverage AND continuous/high-frequency monitoring AND immutable audit trails AND real-time detection AND automated alerts AND incident response integration AND protected monitoring infrastructure AND user/admin notifications AND documentation complete

Fail Criteria: No cryptographic verification OR limited data coverage OR infrequent monitoring OR mutable audit trails OR delayed detection OR no alerts OR no incident response OR unprotected monitoring OR incomplete coverage OR no notifications OR incomplete documentation

Evidence: Integrity monitoring implementation code review, cryptographic verification testing, data coverage assessment, monitoring frequency configuration, immutable audit trail verification, detection latency measurement, alert generation testing, incident response integration documentation, monitoring infrastructure security review, notification system testing

References:

- File Integrity Monitoring: https://csrc.nist.gov/glossary/term/file_integrity_monitoring
- Tamper Detection: https://www.iso.org/standard/56328.html
- Merkle Trees: https://nakamoto.com/merkle-trees/

Assessment: RDPS-REQ-38 (RDPS security event logging with SIEM integration)

Reference: RDPS-REQ-38 - Browser shall provide RDPS security event logging with SIEM integration (RDPS-3 requirement)

Given: A conformant browser with RDPS-3 capability requiring enterprise-grade security event logging and analysis

Task: This assessment verifies that browsers implement comprehensive RDPS security event logging with SIEM integration support, standard log formats (CEF, LEEF, JSON, Syslog), encrypted transport protocols,

sufficient context per entry, reliable transmission with buffering, configurable verbosity levels, tamper protection, and enterprise configuration capabilities.

Verification:

1. Verify comprehensive security event logging for all RDPS operations → Comprehensive security event capture implemented
2. Test that logs capture authentication, authorization, data access, config changes → Authentication, authorization, access, and config events logged
3. Verify each log entry includes sufficient context (timestamp, user, IP, operation, outcome) → Sufficient context in each log entry
4. Test that standard log formats supported (CEF, LEEF, JSON, Syslog) → Standard log formats supported (multiple)
5. Verify standard transport protocols supported (TLS syslog, HTTPS, AMQP, Kafka) → Standard transport protocols supported (encrypted)
6. Test compatibility with major SIEM platforms (Splunk, QRadar, Sentinel, ELK) → Major SIEM platform compatibility verified
7. Verify reliable log transmission with buffering and retry mechanisms → Reliable transmission with buffering and retry
8. Test that log transmission encrypted and authenticated → Encrypted and authenticated log transmission
9. Verify configurable log verbosity levels (minimal, standard, detailed, debug) → Configurable verbosity levels available
10. Test tamper protection through cryptographic signing or immutable storage → Tamper protection implemented with enterprise configuration options

Pass Criteria: Comprehensive security event logging AND sufficient context per entry AND standard formats (CEF/LEEF/JSON/Syslog) AND standard protocols (TLS/HTTPS) AND SIEM compatibility AND reliable transmission AND encryption/authentication AND configurable verbosity AND tamper protection AND enterprise configuration AND documentation complete

Fail Criteria: Limited event logging OR insufficient context OR proprietary formats only OR insecure protocols OR no SIEM compatibility OR unreliable transmission OR unencrypted logs OR fixed verbosity OR no tamper protection OR no enterprise configuration OR incomplete documentation

Evidence: Security event catalog, log format specifications, transport protocol documentation, SIEM compatibility matrix, transmission reliability testing, encryption verification, verbosity configuration testing, tamper protection verification, enterprise configuration options, privacy assessment

References:

- SIEM Integration: https://www.splunk.com/en_us/blog/learn/siem-security-information-event-management.html
- Security Logging: https://cheatsheetseries.owasp.org/cheatsheets/Logging_Cheat_Sheet.html

Assessment: RDPS-REQ-39 (Geographic data residency requirements enforcement)

Reference: RDPS-REQ-39 - Browser shall enforce geographic data residency requirements when configured (RDPS-3 requirement)

Given: A conformant browser with RDPS-3 capability supporting configurable geographic data residency controls for regulatory compliance

Task: This assessment verifies that browsers implement geographic data residency enforcement with configurable regional restrictions, regional endpoint selection, cryptographic location verification, unauthorized transfer blocking, compliance monitoring and auditing, user transparency, and handling of edge cases (international travel, VPN, failover scenarios).

Verification:

1. Verify configurable geographic data residency options available → Geographic residency configuration options available
2. Test that enterprise administrators can specify allowed geographic regions → Enterprise administrators can specify allowed regions
3. Verify browser selects RDPS endpoints within allowed geographic regions → Regional RDPS endpoint selection implemented
4. Test that data storage respects geographic residency constraints → Data storage respects residency constraints
5. Verify data processing occurs only in allowed geographic regions → Data processing limited to allowed regions
6. Test that unauthorized geographic transfers blocked → Unauthorized transfers blocked
7. Verify cryptographic or attestation-based verification of server location → Cryptographic verification of server location
8. Test monitoring detects unauthorized geographic data flows → Monitoring detects policy violations
9. Verify documentation specifies which data types subject to residency controls → Compliance auditing capabilities available
10. Test that users informed about their data's geographic location → User transparency and control over data location

Pass Criteria: Geographic residency configuration AND regional endpoint selection AND storage/processing in allowed regions only AND unauthorized transfer blocking AND cryptographic location verification AND monitoring AND auditing AND user transparency AND edge case handling AND documentation complete

Fail Criteria: No residency configuration OR no regional selection OR storage/processing outside allowed regions OR no transfer blocking OR no location verification OR no monitoring OR no auditing OR no user transparency OR poor edge case handling OR incomplete documentation

Evidence: Geographic residency configuration documentation, regional endpoint selection testing, storage location verification, processing location verification, unauthorized transfer blocking tests, cryptographic attestation verification, monitoring system testing, compliance audit reports, user interface documentation, edge case handling testing

References:

- Cloud Data Location: https://www.microsoft.com/en-us/trust-center/privacy/data-location

Assessment: RDPS-REQ-40 (Zero-trust architecture for RDPS access)

Reference: RDPS-REQ-40 - Browser shall implement zero-trust architecture for RDPS access (RDPS-3 requirement)

Given: A conformant browser with RDPS-3 capability implementing zero-trust security principles for remote data processing access

Task: This assessment verifies that browsers implement zero-trust architecture for RDPS access with continuous per-request authentication/authorization, multi-factor context-aware identity verification, risk-based access decisions, least-privilege permissions, time-limited grants, anomaly monitoring, network micro-segmentation, and always-encrypted traffic.

Verification:

1. Verify continuous authentication and authorization for every RDPS request → Continuous authentication/authorization per request
2. Test that session-based trust not assumed (each request independently verified) → No session-based trust assumed
3. Verify multi-factor authentication required for RDPS access → Multi-factor authentication enforced
4. Test context-aware authentication considers device posture, location, behavior → Context-aware authentication (device, location, behavior)
5. Verify risk scoring aggregates multiple factors for access decisions → Risk-based access decisions

6. Test least-privilege access with fine-grained resource permissions → Least-privilege fine-grained permissions
7. Verify time-limited access grants expire and require renewal → Time-limited access grants
8. Test continuous monitoring detects access pattern anomalies → Anomaly detection and monitoring
9. Verify network micro-segmentation isolates RDPS components → Network micro-segmentation implemented
10. Test end-to-end encryption regardless of network location → End-to-end encryption always enforced

Pass Criteria: Continuous verification per request AND no session-based trust AND multi-factor authentication AND context-aware authentication AND risk-based decisions AND least-privilege AND time-limited grants AND anomaly monitoring AND micro-segmentation AND always-encrypted AND documentation complete

Fail Criteria: Session-based trust OR single-factor authentication OR no context awareness OR no risk scoring OR broad privileges OR indefinite grants OR no anomaly monitoring OR no segmentation OR conditional encryption OR incomplete documentation

Evidence: Zero-trust architecture design documentation, per-request verification testing, multi-factor authentication verification, context-aware authentication testing, risk scoring implementation review, permission granularity assessment, access grant expiration testing, anomaly detection verification, network segmentation documentation, encryption verification across all scenarios

References:

- Zero Trust Architecture: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-207.pdf
- Zero Trust Principles: https://www.microsoft.com/en-us/security/business/zero-trust
- Continuous Verification: https://www.cisa.gov/zero-trust-maturity-model

Assessment: RDPS-REQ-41 (Regulatory compliance logging for RDPS operations)

Reference: RDPS-REQ-41 - Browser shall support regulatory compliance logging for RDPS operations (RDPS-3 requirement)

Given: A conformant browser with RDPS-3 capability requiring compliance with regulatory audit and logging requirements

Task: This assessment verifies that browsers implement regulatory compliance logging for RDPS operations capturing data subject rights, consent, breaches, transfers, retention, and administrative actions with regulatory-required fields, immutable storage, regulation-specific retention periods (1-7 years), compliance reporting templates, and enterprise policy configuration.

Verification:

1. Verify compliance logging captures data subject rights exercises (GDPR Article 15-22) → Data subject rights exercises logged (access, erasure, portability, etc.)
2. Test that consent and withdrawal events logged with legal basis → Consent and legal basis tracked
3. Verify data breach detection and notification events logged → Breach detection and notification logged
4. Test that cross-border transfer events logged with legal mechanism → Cross-border transfers logged with legal mechanism
5. Verify data retention and deletion operations logged → Retention and deletion operations logged
6. Test that administrative access to sensitive data logged → Administrative access logged
7. Verify security control changes logged (encryption, access controls) → Regulatory-required fields in each entry
8. Test that each log entry includes regulatory-required fields → Immutable log storage implemented
9. Verify log immutability through append-only storage or cryptographic chaining → Regulation-specific retention periods enforced
10. Test that logs retained for regulation-specific periods (1-7 years typical) → Compliance reporting with templates and export capabilities

Pass Criteria: All regulatory events logged (rights, consent, breaches, transfers, retention) AND regulatory-required fields AND immutable storage AND regulation-specific retention AND compliance reporting AND export capabilities AND enterprise configuration AND documentation complete

Fail Criteria: Missing regulatory events OR insufficient fields OR mutable logs OR non-compliant retention OR no reporting OR no export OR no enterprise configuration OR incomplete documentation

Evidence: Compliance event catalog mapped to regulations, log schema with regulatory fields, immutability verification (append-only storage, cryptographic chaining), retention policy documentation, compliance report templates, export format specifications, enterprise configuration options, regulatory mapping documentation

References:

- GDPR Compliance Logging: https://gdpr.eu/
- ISO 27001 Audit Controls: https://www.iso.org/standard/27001

Assessment: RDPS-REQ-42 (Automated RDPS security scanning and vulnerability detection)

Reference: RDPS-REQ-42 - Browser shall implement automated RDPS security scanning and vulnerability detection (RDPS-3 requirement)

Given: A conformant browser with RDPS-3 capability requiring continuous security assessment of remote data processing infrastructure

Task: This assessment verifies that browsers implement automated RDPS security scanning covering vulnerability, configuration, dependency, and compliance domains with regular or event-triggered execution, vulnerability management workflow integration, automated prioritization, critical alerts (CVSS 9.0+), security dashboards, and remediation tracking.

Verification:

1. Verify automated vulnerability scanning for RDPS infrastructure components → Automated vulnerability scanning implemented
2. Test that vulnerability scanning identifies known CVEs in OS, applications, libraries → Configuration scanning for insecure settings
3. Verify configuration scanning detects insecure settings and weak crypto → Dependency scanning for supply chain risks
4. Test that dependency scanning identifies vulnerable third-party libraries → Compliance scanning (CIS, ISO 27001)
5. Verify compliance scanning checks adherence to security frameworks (CIS, ISO 27001) → Regular scanning schedule (daily/weekly) or event-triggered
6. Test that scanning performed regularly (daily/weekly) or event-triggered → Vulnerability management workflow integration
7. Verify scanning results integrated with vulnerability management workflows → Automated prioritization by severity and exploitability
8. Test automated prioritization based on severity (CVSS scores) and exploitability → Critical vulnerability immediate alerting
9. Verify critical vulnerabilities (CVSS 9.0+) trigger immediate alerts → Security dashboards with trends and metrics
10. Test that security dashboards show trends, metrics, compliance, and risk scores → False positive management and remediation tracking

Pass Criteria: Multi-domain scanning (vulnerability, config, dependency, compliance) AND regular/event-triggered scanning AND vulnerability management integration AND automated prioritization AND critical alerts AND security dashboards AND remediation tracking AND false positive management AND documentation complete

Fail Criteria: Single-domain scanning only OR infrequent scanning OR no workflow integration OR manual prioritization OR no critical alerts OR no dashboards OR no tracking OR poor false positive handling OR

incomplete documentation

Evidence: Security scanning tool documentation, vulnerability scan reports, configuration scan results, dependency scan output, compliance scan reports, scanning frequency configuration, vulnerability management workflow documentation, prioritization algorithm, critical alert examples, security dashboard screenshots, remediation tracking reports

References:

- Vulnerability Scanning: https://nvd.nist.gov/vuln/search
- CIS Benchmarks: https://www.cisecurity.org/cis-benchmarks/
- OWASP Dependency Check: https://owasp.org/www-project-dependency-check/

Assessment: RDPS-REQ-43 (Cryptographic proof of RDPS data integrity)

Reference: RDPS-REQ-43 - Browser shall provide cryptographic proof of RDPS data integrity (RDPS-3 requirement)

Given: A conformant browser with RDPS-3 capability requiring verifiable proof of data integrity for critical remote processing

Task: This assessment verifies that browsers implement cryptographic proof mechanisms for RDPS data integrity (signatures, HMAC, Merkle trees, authenticated encryption) with immediate proof generation, tamper-evident storage, efficient verification, batch support, audit trails, third-party verification capability, and user-initiated on-demand verification with proof downloads.

Verification:

1. Verify cryptographic integrity mechanisms implemented (signatures, HMAC, Merkle trees, authenticated encryption) → Cryptographic integrity mechanisms implemented (multiple types)
2. Test that integrity proofs generated immediately upon data creation/modification → Proofs generated immediately upon creation/modification
3. Verify proofs stored securely in tamper-evident storage → Tamper-evident proof storage
4. Test that verification possible at any time by authorized parties → Verification available at any time
5. Verify verification process efficient for regular checking (hourly/daily) → Efficient verification for regular checking
6. Test batch verification of multiple data items supported → Batch verification supported
7. Verify clear verification results provided (verified, tampered, unable to verify) → Clear verification results
8. Test that audit trail maintained for all verification events → Verification audit trail maintained
9. Verify third-party verification supported (transparency logs, notarization) → Third-party verification capability
10. Test that users can initiate on-demand verification and download proofs → User on-demand verification and proof download

Pass Criteria: Multiple cryptographic mechanisms AND immediate proof generation AND tamper-evident storage AND any-time verification AND efficient regular checking AND batch verification AND clear results AND audit trail AND third-party verification AND user controls AND documentation complete

Fail Criteria: Single mechanism only OR delayed proof generation OR insecure storage OR limited verification access OR inefficient verification OR no batch support OR unclear results OR no audit trail OR no third-party verification OR no user controls OR incomplete documentation

Evidence: Cryptographic mechanism implementation documentation, proof generation verification, tamper-evident storage testing, verification access control testing, performance benchmarks for verification, batch verification testing, verification result examples, audit trail documentation, third-party verification testing, user interface documentation

References:

- Digital Signatures: https://csrc.nist.gov/projects/digital-signatures
- Merkle Trees: https://nakamoto.com/merkle-trees/
- Certificate Transparency: https://certificate.transparency.dev/

Assessment: RDPS-REQ-44 (Secure multi-tenancy with data isolation in RDPS)

Reference: RDPS-REQ-44 - Browser shall implement secure multi-tenancy with data isolation in RDPS (RDPS-3 requirement)

Given: A conformant browser with RDPS-3 capability serving multiple independent users or organizations through shared infrastructure

Task: This assessment verifies that browsers implement secure RDPS multi-tenancy with per-tenant encryption keys, data-layer access controls, separate database schemas, query filtering, resource quotas, process/container isolation, network isolation, identity boundaries, separate audit logs and backups, resilience against side-channel attacks, and penetration testing of tenant boundaries.

Verification:

1. Verify per-tenant encryption keys prevent cross-tenant data decryption → Per-tenant encryption keys implemented
2. Test that per-tenant access controls enforced at data layer → Data-layer access control enforcement
3. Verify per-tenant database schemas or namespaces implemented → Separate database schemas or namespaces
4. Test that query results never include cross-tenant data → Query result filtering prevents cross-tenant data
5. Verify per-tenant resource quotas (storage, compute, bandwidth, rate limits) → Resource quotas enforced per tenant
6. Test process or container isolation with separate execution contexts → Process/container isolation per tenant
7. Verify network isolation prevents cross-tenant communication → Network isolation between tenants
8. Test strict tenant boundaries in identity and authentication systems → Identity system tenant boundary enforcement
9. Verify separate audit logs per tenant with no cross-tenant access → Separate audit logs per tenant
10. Test separate backup and recovery per tenant → Tenant-specific backup/recovery and configuration

Pass Criteria: Per-tenant encryption AND data-layer access controls AND separate schemas AND query filtering AND resource quotas AND process isolation AND network isolation AND identity boundaries AND separate audit logs AND separate backup/recovery AND penetration testing AND documentation complete

Fail Criteria: Shared encryption OR inadequate access controls OR shared schemas OR cross-tenant queries OR no quotas OR shared processes OR no network isolation OR weak identity boundaries OR shared logs OR shared backups OR no penetration testing OR incomplete documentation

Evidence: Multi-tenancy architecture documentation, per-tenant encryption verification, access control testing, database schema isolation verification, query filtering tests, resource quota enforcement testing, process isolation verification, network isolation testing, identity boundary testing, audit log separation verification, backup/recovery isolation testing, penetration test reports

References:

- Multi-Tenancy Security: https://csrc.nist.gov/publications/detail/sp/800-144/final
- Cloud Isolation: https://aws.amazon.com/compliance/data-privacy-faq/
- Tenant Isolation Patterns: https://docs.microsoft.com/en-us/azure/architecture/guide/multitenant/considerations/tenan models

Assessment: RDPS-REQ-45 (RDPS security incident response procedures)

Reference: RDPS-REQ-45 - Browser shall provide incident response procedures for RDPS breaches (RDPS-3 requirement)

Given: A conformant browser with RDPS-3 capability requiring documented incident response capabilities for security breaches

Task: This assessment verifies that browsers have documented and tested RDPS incident response procedures following established frameworks (NIST, SANS, ISO 27035) with incident classification, RDPS-specific scenarios, 24/7 capability, automated detection/alerting, incident tracking systems, regular drills (annual minimum), post-incident analysis, and regulatory-compliant user notification.

Verification:

1. Verify comprehensive incident response procedures documented following established frameworks → Comprehensive IR procedures documented (NIST, SANS, ISO frameworks)
2. Test incident classification criteria and escalation procedures defined → Incident classification and escalation procedures defined
3. Verify containment, investigation, remediation, and recovery procedures documented → Containment, investigation, remediation, recovery procedures
4. Test RDPS-specific incident scenarios documented with response procedures → RDPS-specific incident scenarios with response plans
5. Verify 24/7 incident response capability with on-call security team → 24/7 incident response capability
6. Test automated incident detection and alerting integrated with monitoring → Automated detection and alerting
7. Verify incident tracking and management system implemented → Incident tracking system
8. Test regular incident response drills performed (minimum annually) → Regular IR drills and exercises (annual minimum)
9. Verify post-incident lessons learned and root cause analysis documented → Post-incident analysis and improvements
10. Test that users notified of incidents according to regulatory timelines → User notification within regulatory timelines

Pass Criteria: Comprehensive documented procedures following frameworks AND incident classification AND RDPS-specific scenarios AND 24/7 capability AND automated detection AND tracking system AND regular drills (annual minimum) AND post-incident analysis AND regulatory-compliant user notification AND documentation complete

Fail Criteria: No documented procedures OR no classification OR generic scenarios only OR no 24/7 capability OR manual detection OR no tracking OR no drills OR no post-incident analysis OR delayed/missing user notification OR incomplete documentation

Evidence: Incident response procedure documentation, incident classification matrix, RDPS scenario playbooks, on-call schedule documentation, incident detection system configuration, incident tracking system, drill schedules and reports, post-incident review documentation, user notification templates and timelines, regulatory compliance verification

References:

- NIST Incident Response: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-61r2.pdf
- SANS Incident Response: https://www.sans.org/white-papers/33901/

Assessment: RDPS-REQ-46 (RDPS access revocation mechanisms)

Reference: RDPS-REQ-46 - Browser shall implement RDPS access revocation mechanisms (RDPS-3 requirement)

Given: A conformant browser with RDPS-3 capability requiring rapid access revocation for compromised accounts or terminated users

Task: This assessment verifies that browsers implement rapid RDPS access revocation mechanisms (tokens, credentials, devices, accounts) with immediate effectiveness (seconds to minutes) across all components, prevention of cached bypass, individual and bulk revocation, multi-channel administrative interfaces, audit trails, automated security-triggered revocation, and user notification.

Verification:

1. Verify multiple revocation mechanisms implemented (tokens, credentials, devices, accounts) → Multiple revocation mechanisms at different layers
2. Test that revocation effective within seconds to minutes across all RDPS components → Immediate revocation (seconds to minutes) across all components
3. Verify revoked access cannot be restored through cached credentials or stale state → Prevention of cached credential bypass
4. Test individual and bulk revocation capabilities → Individual and bulk revocation supported
5. Verify administrative interfaces enable revocation through multiple channels → Multi-channel administrative revocation interfaces
6. Test revocation effectiveness verification mechanisms → Revocation effectiveness verification
7. Verify audit trails maintained for all revocation events → Comprehensive revocation audit trails
8. Test automated revocation triggers respond to security events → Automated security-event-triggered revocation
9. Verify user notification of revocation (when appropriate) → User notification (when appropriate)
10. Test enterprise administrator configuration of revocation policies → Enterprise policy configuration and reporting

Pass Criteria: Multiple revocation mechanisms AND immediate effectiveness (seconds to minutes) AND no cached bypass AND individual/bulk revocation AND multi-channel interfaces AND effectiveness verification AND audit trails AND automated triggers AND user notification AND enterprise policies AND documentation complete

Fail Criteria: Single mechanism only OR delayed revocation OR cached bypass possible OR limited revocation scope OR single channel only OR no verification OR no audit trails OR no automation OR no user notification OR no enterprise policies OR incomplete documentation

Evidence: Revocation mechanism documentation, revocation latency measurements across all components, cached credential bypass testing, bulk revocation testing, administrative interface documentation, effectiveness verification testing, audit trail examples, automated trigger configuration, user notification templates, enterprise policy documentation

References:

- OAuth Token Revocation: https://datatracker.ietf.org/doc/html/rfc7009

Assessment: RDPS-REQ-47 (RDPS transparency reporting for data access)

Reference: RDPS-REQ-47 - Browser shall provide transparency reporting for RDPS data access (RDPS-3 requirement)

Given: A conformant browser with RDPS-3 capability requiring transparent reporting of data access for user trust and regulatory compliance

Task: This assessment verifies that browsers provide RDPS transparency reporting showing all data access types (client, administrative, third-party, automated) with sufficient entry detail, on-demand user access, standard format downloads (PDF, CSV, JSON), user-configurable notifications, government request reporting (where legal), enterprise aggregate reporting, and 1-year minimum retention with tampering protection.

Verification:

1. Verify user-facing transparency interfaces showing all RDPS data access → User-facing transparency interfaces implemented

2. Test that reports include browser client, administrative, third-party, and automated access → Comprehensive access coverage (client, admin, third-party, automated)
3. Verify each entry includes timestamp, accessor, type, data categories, justification, duration → Sufficient detail in each report entry (timestamp, accessor, type, categories, justification)
4. Test that users can access transparency reports on demand → On-demand user access to reports
5. Verify download capabilities in standard formats (PDF, CSV, JSON) → Standard format downloads available
6. Test user-configurable notifications for specific access types → User-configurable access notifications
7. Verify government data request transparency reporting (where legally permissible) → Government request transparency (where legal)
8. Test enterprise aggregate access reporting with anomaly identification → Enterprise aggregate reporting with anomaly detection
9. Verify balance between transparency and security (no security mechanism disclosure) → Security and privacy balanced appropriately
10. Test efficient report generation with minimal performance impact → Reports retained for minimum 1 year with tampering protection

Pass Criteria: Comprehensive transparency interfaces AND all access types covered AND sufficient entry detail AND on-demand access AND standard format downloads AND user notifications AND government request reporting (where legal) AND enterprise reporting AND appropriate security/privacy balance AND retention (1 year minimum) AND documentation complete

Fail Criteria: Limited transparency OR missing access types OR insufficient detail OR no on-demand access OR no downloads OR no notifications OR no government reporting OR no enterprise reporting OR security/privacy imbalance OR short retention OR incomplete documentation

Evidence: Transparency interface documentation, access type coverage assessment, report entry examples, user access mechanism testing, download format verification, notification configuration testing, government request reports (if available), enterprise reporting examples, security/privacy review, retention verification, user education materials

References:

- GDPR Article 15 Right of Access: https://gdpr.eu/article-15-right-of-access/
- Transparency Reporting: https://www.eff.org/issues/transparency
- Google Transparency Report: https://transparencyreport.google.com/

Assessment: RDPS-REQ-48 (Forward secrecy for RDPS communications)

Reference: RDPS-REQ-48 - Browser shall implement forward secrecy for RDPS communications (RDPS-3 requirement)

Given: A conformant browser with RDPS-3 capability requiring forward secrecy protection for remote data processing communications

Task: This assessment verifies that browsers implement forward secrecy for all RDPS communications using TLS 1.3 or TLS 1.2 with ECDHE/DHE cipher suites, disable non-forward-secret ciphers (RSA key exchange), verify endpoint support, securely destroy session keys, resist downgrade attacks, and provide enterprise monitoring and policy enforcement.

Verification:

1. Verify forward-secret cipher suites implemented for all RDPS communications → Forward-secret cipher suites implemented (ECDHE, DHE)
2. Test TLS 1.3 used (all cipher suites provide forward secrecy) or TLS 1.2 with ECDHE/DHE → TLS 1.3 or TLS 1.2 with FS cipher suites
3. Verify ECDHE prioritized over DHE for performance → ECDHE prioritized for performance
4. Test that non-forward-secret cipher suites (RSA key exchange) disabled → Non-FS cipher suites disabled (RSA key exchange)

5. Verify browser checks RDPS endpoint support for forward-secret ciphers → Endpoint forward secrecy verification
6. Test browser refuses or warns when endpoint lacks forward secrecy support → Connection refusal or prominent warnings for non-FS endpoints
7. Verify cipher suite selection logged for security auditing → Cipher suite selection logging
8. Test key material securely destroyed after session termination → Secure key destruction after sessions
9. Verify resistance to downgrade attacks forcing weaker ciphers → Downgrade attack resistance
10. Test enterprise administrators can verify and monitor cipher suite usage → Enterprise verification, monitoring, and policy enforcement

Pass Criteria: Forward-secret cipher suites for all RDPS AND TLS 1.3 or TLS 1.2 with ECDHE/DHE AND ECDHE prioritized AND non-FS disabled AND endpoint verification AND warnings/refusal AND logging AND secure key destruction AND downgrade resistance AND enterprise monitoring AND documentation complete

Fail Criteria: Non-FS cipher suites allowed OR no TLS 1.3 and weak TLS 1.2 OR DHE without ECDHE OR RSA key exchange enabled OR no endpoint verification OR silent non-FS acceptance OR no logging OR insecure key handling OR downgrade vulnerable OR no enterprise monitoring OR incomplete documentation

Evidence: Cipher suite configuration documentation, TLS connection testing showing FS cipher suites, endpoint verification testing, non-FS endpoint handling tests, cipher suite logging examples, key destruction verification, downgrade attack testing, enterprise monitoring interface documentation

References:

- Perfect Forward Secrecy: https://en.wikipedia.org/wiki/Forward_secrecy
- TLS 1.3 Specification: https://datatracker.ietf.org/doc/html/rfc8446
- OWASP TLS Cipher String Cheat Sheet: https://cheatsheetseries.owasp.org/cheatsheets/TLS_Cipher_String_Cheat_Sh

Assessment: RDPS-REQ-49 (User notification of RDPS security events)

Reference: RDPS-REQ-49 - Browser shall provide user notification of RDPS security events (RDPS-3 requirement)

Given: A conformant browser with RDPS-3 capability requiring user notification of security-relevant events

Task: This assessment verifies that browsers implement comprehensive RDPS security event notifications covering critical events (new device, authentication, data access, config changes, incidents, lifecycle) with multiple delivery channels (in-browser, email, push, SMS), user-configurable preferences, sufficient context, appropriate timing (immediate, near-real-time, digest), intelligent aggregation preventing fatigue, and notification history management.

Verification:

1. Verify notifications for critical security events (new device, location, authentication) → Critical security event notifications implemented
2. Test notifications for data access events (administrative access, third-party disclosure) → Data access and configuration change notifications
3. Verify notifications for security configuration changes → Security incident and data lifecycle notifications
4. Test notifications for security incidents and data lifecycle events → Multiple delivery channels with user preferences
5. Verify multiple delivery channels (in-browser, email, mobile push, SMS) → Sufficient context in notifications
6. Test user-configurable notification preferences → Appropriate timing (immediate, near-real-time, digest)
7. Verify sufficient context in each notification (what, when, where, data affected, actions) → Intelligent aggregation preventing notification fatigue
8. Test appropriate notification timing (immediate for critical, near-real-time for important) → Risk-based prioritization highlighting critical events

9. Verify intelligent aggregation and risk-based prioritization prevent fatigue → Notification history and preference management
10. Test notification history review and preference configuration → Enterprise policy configuration respecting user preferences

Pass Criteria: Comprehensive security event coverage AND multiple delivery channels AND user-configurable preferences AND sufficient context AND appropriate timing AND intelligent aggregation AND risk prioritization AND notification history AND user preference management AND enterprise policies AND documentation complete

Fail Criteria: Limited event coverage OR single channel only OR no user preferences OR insufficient context OR inappropriate timing OR no aggregation (fatigue) OR no prioritization OR no history OR no preference management OR no enterprise policies OR incomplete documentation

Evidence: Security event notification catalog, delivery channel documentation, notification preference UI, notification content examples, timing configuration, aggregation and prioritization algorithms, notification history interface, enterprise policy documentation, user education materials

References:

- User Notification Design: https://material.io/design/platform-guidance/android-notifications.html

Assessment: RDPS-REQ-50 (Service discontinuation and data migration documentation)

Reference: RDPS-REQ-50 - Browser shall document service discontinuation and data migration procedures (RDPS-3 requirement)

Given: A conformant browser with RDPS-3 capability requiring documented procedures for service discontinuation scenarios

Task: This assessment verifies that browsers document service discontinuation procedures with minimum notification timelines (90 days consumer, 180 days enterprise), data export in standard formats, migration assistance, transition periods, secure deletion, multiple discontinuation scenarios, anytime data export capability, complete data/metadata export, and enterprise enhanced protections (extended timelines, dedicated support, escrow options).

Verification:

1. Verify documented service discontinuation procedures including notification timelines → Service discontinuation procedures documented
2. Test that minimum notification periods specified (90 days consumer, 180 days enterprise) → Minimum notification timelines (90 days consumer, 180+ days enterprise)
3. Verify data export capabilities in standard formats (JSON, XML, CSV, original formats) → Data export in standard open formats
4. Test data migration assistance documentation and tools → Migration assistance documentation and tools
5. Verify service transition period with continued read-only access → Transition period with continued access
6. Test data deletion procedures after termination with secure deletion → Secure data deletion post-termination
7. Verify documentation addresses various discontinuation scenarios → Multiple discontinuation scenarios addressed
8. Test that users can export all their data at any time (not just during discontinuation) → Anytime data export capability
9. Verify export includes all user data and metadata with completeness verification → Complete data and metadata export with verification
10. Test enterprise enhanced protections (longer periods, dedicated support, escrow) → Enterprise enhanced protections (extended timelines, support, escrow options)

Pass Criteria: Comprehensive discontinuation documentation AND minimum notification periods (90/180 days) AND data export in standard formats AND migration assistance AND transition period AND secure deletion AND multiple scenarios addressed AND anytime export AND complete data export AND enterprise protections AND documentation complete

Fail Criteria: No discontinuation documentation OR insufficient notification periods OR proprietary export formats only OR no migration assistance OR no transition period OR insecure deletion OR single scenario only OR export only during discontinuation OR incomplete export OR no enterprise protections OR incomplete documentation

Evidence: Service discontinuation procedure documentation, notification timeline specifications, data export format documentation, migration tool documentation, transition period policies, data deletion procedures, discontinuation scenario documentation, anytime export verification, export completeness testing, enterprise protection agreements, data portability standards participation

References:

- GDPR Article 20 Data Portability: https://gdpr.eu/article-20-right-to-data-portability/
- Service Discontinuation Best Practices: https://www.eff.org/deeplinks/2019/10/adversarial-interoperability

Assessment: RDPS-REQ-51 (Enterprise administrator RDPS security policy configuration)

Reference: RDPS-REQ-51 - Enterprise administrators shall be able to configure RDPS security policies (RDPS-3 requirement)

Given: A conformant browser with RDPS-3 capability deployed in enterprise environments requiring centralized security policy management

Task: This assessment verifies that browsers provide centralized enterprise RDPS policy management for authentication, encryption, data residency, access control, monitoring, incident response, and data lifecycle with immediate enforcement across installations, policy compliance reporting, inheritance/versioning/testing capabilities, tamper prevention, and comprehensive configuration documentation.

Verification:

1. Verify centralized policy management interfaces for enterprise administrators → Centralized policy management interfaces available
2. Test authentication policy configuration (MFA, passwords, certificates, biometrics) → Authentication policies configurable (MFA, passwords, certs, biometrics)
3. Verify encryption policy configuration (TLS versions, cipher suites, E2EE, hardware keys) → Encryption policies configurable (TLS, ciphers, E2EE, hardware)
4. Test data residency policy configuration (geographic regions, cross-border controls) → Data residency policies configurable
5. Verify access control policy configuration (data access, admin access, restrictions) → Access control policies configurable
6. Test monitoring and logging policy configuration (verbosity, retention, SIEM, compliance) → Monitoring and logging policies configurable
7. Verify incident response policy configuration (notifications, escalation, automation) → Incident response and data lifecycle policies configurable
8. Test data lifecycle policy configuration (retention, deletion, backup, classification) → Immediate policy enforcement across installations
9. Verify immediate policy enforcement across all installations → Policy compliance dashboards with metrics and violation reporting
10. Test policy compliance reporting and violation prevention → Policy inheritance, versioning, testing, and comprehensive documentation

Pass Criteria: Centralized management interfaces AND comprehensive policy categories (authentication, encryption, residency, access, monitoring, incident response, lifecycle) AND immediate enforcement AND

compliance reporting AND policy inheritance/versioning AND testing capabilities AND tamper prevention AND documentation complete

Fail Criteria: No centralized management OR limited policy categories OR delayed enforcement OR no compliance reporting OR no inheritance/versioning OR no testing OR user tampering possible OR incomplete documentation

Evidence: Policy management interface documentation, authentication policy configuration testing, encryption policy testing, data residency policy testing, access control policy testing, monitoring policy testing, policy enforcement verification, compliance dashboard screenshots, policy inheritance testing, versioning and rollback testing, policy template documentation, integration guides

References:

- Chrome Enterprise Policies: https://chromeenterprise.google/policies/
- Microsoft Intune: https://docs.microsoft.com/en-us/mem/intune/
- NIST Enterprise Configuration Management: https://csrc.nist.gov/publications/detail/sp/800-70/rev-4/final

Assessment: RDPS-REQ-52 (Fully offline browser operation without remote connectivity)

Reference: RDPS-REQ-52 - Browser shall operate fully offline without requiring remote connectivity (RDPS-0 requirement)

Given: A conformant browser claiming RDPS-0 capability (no remote data processing) for air-gapped or offline-only deployments

Task: This assessment verifies that browsers operate completely offline with all core functionality (rendering, apps, bookmarks, settings, developer tools, security features), no network connectivity requirements for installation/setup/activation/updates, no error messages or degraded warnings, complete offline installers, and indefinite offline operation capability.

Verification:

1. Verify browser installs completely offline using offline installer → Complete offline installation with offline installer
2. Test that initial setup completes without network connectivity → No online activation or registration required
3. Verify all core functionality works offline (rendering, JavaScript, apps, bookmarks, settings) → All core functionality operational offline
4. Test that security features function without network (sandbox, encryption, isolation) → Security features fully functional without network
5. Verify browser never attempts network connections when offline → No network connection attempts when offline
6. Test that no error messages or warnings about missing network → No error messages or degraded functionality warnings
7. Verify offline-capable web applications work correctly (PWAs, service workers) → Offline web application support (PWAs, service workers)
8. Test that browser settings and configuration fully accessible offline → All settings and configuration accessible offline
9. Verify developer tools fully functional offline → Developer tools fully functional offline
10. Test that browser operates indefinitely without updates or online validation → Indefinite offline operation without degradation

Pass Criteria: Complete offline installation AND no online activation AND all core functionality offline AND all security features offline AND no network attempts AND no error/warning messages AND offline web app support AND indefinite operation AND documentation complete

Fail Criteria: Requires online installation OR online activation required OR limited functionality offline OR security features require network OR network connection attempts OR error/warning messages OR no offline app support OR time-limited offline OR incomplete documentation

Evidence: Offline installer verification, offline installation testing, network monitoring showing zero connection attempts, functionality testing without network, security feature verification offline, error message audit, offline web app testing, long-term offline operation testing, documentation review

References:

- Offline Web Applications: https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps/Offline_Service_work
- Air-Gapped System Security: https://csrc.nist.gov/glossary/term/air_gap
- Offline-First Design: https://offlinefirst.org/

Assessment: RDPS-REQ-53 (All user data stored locally without remote synchronization)

Reference: RDPS-REQ-53 - All user data shall be stored locally without remote synchronization (RDPS-0 requirement)

Given: A conformant browser claiming RDPS-0 capability (no remote data processing) for local-only data storage

Task: This assessment verifies that browsers store all user data types (history, bookmarks, passwords, autofill, cookies, settings, extensions, cached content) exclusively in local file system locations with no remote endpoints, no synchronization protocols, no sync features, zero network transmission, and documentation clearly stating local-only storage with user backup responsibilities and data portability guidance.

Verification:

1. Verify all user data types stored in local file system locations → All user data types stored locally (history, bookmarks, passwords, etc.)
2. Test that browsing history, bookmarks, passwords stored locally only → Local file system storage locations only
3. Verify form autofill, cookies, downloads, search history stored locally → No remote endpoints configured
4. Test that browser settings, extensions, cached content stored locally → No synchronization protocols present
5. Verify offline web app data (service workers, IndexedDB, localStorage) stored locally → No sync account or cloud backup features
6. Test that no remote endpoints configured for any data storage → Zero network transmission of user data (verified via monitoring)
7. Verify no synchronization protocols implemented (no sync code) → Documentation states local-only storage
8. Test that browser provides no remote synchronization features (no sync accounts) → User backup responsibility explained
9. Verify network monitoring shows zero data transmission for user data → Data loss risk warnings provided
10. Test documentation clearly states local-only storage and backup responsibilities → Data portability guidance (manual export/import)

Pass Criteria: All user data types local AND local file system storage AND no remote endpoints AND no sync protocols AND no sync features AND zero network transmission AND clear documentation AND backup guidance AND data loss warnings AND portability guidance

Fail Criteria: Any remote storage OR remote endpoints configured OR sync protocols present OR sync features available OR network data transmission OR unclear documentation OR no backup guidance OR no warnings OR no portability guidance

Evidence: File system analysis showing local-only storage, network monitoring showing zero user data transmission, code review showing no synchronization capabilities, documentation stating local-only storage

with backup responsibilities, data portability documentation

References:

- Local Storage Privacy: https://developer.mozilla.org/en-US/docs/Web/API/Window/localStorage
- Air-Gapped Data Management: https://csrc.nist.gov/glossary/term/air_gap

Assessment: RDPS-REQ-54 (No telemetry, diagnostics, or usage data transmission)

Reference: RDPS-REQ-54 - Browser shall not transmit telemetry, diagnostics, or usage data to remote servers (RDPS-0 requirement)

Given: A conformant browser claiming RDPS-0 capability (no remote data processing) for privacy-preserving operation

Task: This assessment verifies that browsers do not collect or transmit any telemetry, diagnostics, crash reports, usage analytics, or statistical data with no analytics SDKs or libraries, no crash reporting services, zero network transmission, verifiable through network monitoring and code review, and documentation clearly stating no telemetry collection.

Verification:

1. Verify no usage analytics collected or transmitted (page views, feature usage, session duration) → No usage analytics collection or transmission
2. Test that no performance metrics transmitted (rendering times, memory, CPU) → No performance metrics transmission
3. Verify no crash reports transmitted (stack traces, memory dumps, error logs) → No crash report transmission
4. Test that no debugging information transmitted (console logs, JavaScript errors) → No debugging information transmission
5. Verify no feature adoption metrics transmitted (feature usage, configuration) → No feature or system information transmission
6. Test that no system information transmitted (OS version, hardware, installed software) → No analytics SDKs or libraries present
7. Verify no network information transmitted (IP addresses, DNS queries, timing) → No crash reporting service integration
8. Test that no analytics SDKs or libraries included in browser → Zero telemetry transmission (verified via network monitoring)
9. Verify no crash reporting services integrated → Documentation states no telemetry
10. Test network monitoring shows zero telemetry transmission → Code review confirms no telemetry infrastructure

Pass Criteria: Zero telemetry of all types (usage, performance, crashes, debugging, features, system) AND no analytics SDKs AND no crash reporting services AND zero network transmission AND clear documentation AND code review verification

Fail Criteria: Any telemetry transmission OR analytics SDKs present OR crash reporting integrated OR network telemetry observed OR unclear documentation OR unverified code

Evidence: Network monitoring showing zero telemetry transmission, code review showing no analytics or telemetry libraries, binary analysis showing no telemetry endpoints, documentation stating no telemetry, privacy policy confirmation

References:

- Telemetry Privacy Risks: https://www.eff.org/deeplinks/2019/10/privacy-badger-now-fights-more-tracking-embedded-tweets
- GDPR Telemetry: https://gdpr.eu/what-is-gdpr/

Assessment: RDPS-REQ-55 (No degradation when network connectivity unavailable)

Reference: RDPS-REQ-55 - Browser shall function without degradation when network connectivity unavailable (RDPS-0 requirement)

Given: A conformant browser claiming RDPS-0 capability (no remote data processing) for reliable offline operation

Task: This assessment verifies that browsers provide identical functionality and performance whether online or offline for all core features (rendering, applications, browsing, settings, developer tools, extensions, passwords, local storage) with no connectivity checks gating features, no error/warning messages, and equivalent performance metrics when tested with cached content.

Verification:

1. Verify all rendering capabilities work identically offline (HTML, CSS, JS, images, media, fonts) → All rendering capabilities identical offline
2. Test that web applications and scripts execute normally offline → Web applications execute normally offline
3. Verify browsing features fully functional offline (tabs, windows, navigation, bookmarks, history) → All browsing features functional offline
4. Test browser settings and configuration accessible offline without limitations → Settings and configuration fully accessible offline
5. Verify developer tools fully functional offline → Developer tools fully functional offline
6. Test extension management works offline (previously installed extensions) → Extension management operational offline
7. Verify password management and autofill work offline with local data → Password management and autofill work offline
8. Test local data storage operations work identically offline (cookies, localStorage, IndexedDB) → Local storage operations identical offline
9. Verify no connectivity checks before enabling features → No connectivity checks gating features
10. Test that no connectivity-dependent UI elements or degradation warnings shown → Performance metrics equivalent online/offline (verified with cached content)

Pass Criteria: All features identical offline AND no performance reduction AND no error/warning messages AND no feature limitations AND no connectivity checks AND equivalent performance metrics AND clear documentation

Fail Criteria: Reduced functionality offline OR degraded performance offline OR error/warning messages OR limited features OR connectivity checks gating features OR slower offline performance OR unclear documentation

Evidence: Offline functionality testing for all features, performance benchmarking (online vs offline with cached content), network monitoring during offline operation, error message audit, feature availability audit, UI element inspection, performance metrics comparison

References:

- Offline-First Web Applications: https://offlinefirst.org/
- Service Worker Offline Support: https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps/Offline_Service_
- Progressive Enhancement: https://www.w3.org/wiki/Graceful_degradation_versus_progressive_enhancement

Assessment: RDPS-REQ-56 (No remote authentication or authorization services required)

Reference: RDPS-REQ-56 - No remote authentication or authorization services shall be required (RDPS-0 requirement)

Given: A conformant browser claiming RDPS-0 capability (no remote data processing) for local-only authentication

Task: This assessment verifies that browsers require no remote authentication or authorization for any functionality (installation, features, settings, data management, updates, security) with optional local authentication mechanisms only, local credential storage, identical online/offline behavior, and documentation confirming no remote authentication requirements.

Verification:

1. Verify no remote authentication required for installation and activation → No remote authentication for installation/activation
2. Test that all browser features accessible without online authentication → All features accessible without online authentication
3. Verify browser functionality works without remote authorization services → No remote authorization services required
4. Test local user data access requires no remote authentication → Local data access without remote authentication
5. Verify manual updates work without online validation → Manual updates without online validation
6. Test license validation, if any, does not require remote servers → No remote license validation required
7. Verify all security features operate without remote authentication → Security features independent of remote authentication
8. Test optional local authentication mechanisms work offline (master passwords, profiles, locks) → Optional local authentication works offline
9. Verify authentication data stored locally (encrypted database, OS credential manager) → Authentication data stored locally only
10. Test identical authentication behavior whether online or offline → Identical online/offline authentication behavior

Pass Criteria: Zero remote authentication requirements AND all features accessible without authentication AND local authentication only (if any) AND identical online/offline behavior AND local credential storage AND clear documentation

Fail Criteria: Any remote authentication required OR features gated by online authentication OR remote authorization services OR different online/offline behavior OR cloud-synced credentials OR unclear documentation

Evidence: Installation testing without network, feature access testing offline, authentication flow analysis showing local-only verification, network monitoring showing no authentication traffic, credential storage analysis showing local-only storage, online/offline behavior comparison, documentation review

References:

- Local Authentication: https://developer.mozilla.org/en-US/docs/Web/API/Web_Authentication_API
- Air-Gapped Authentication: https://csrc.nist.gov/glossary/term/authentication
- Privacy-Preserving Authentication: https://www.w3.org/TR/webauthn-3/

Assessment: RDPS-REQ-57 (Local-only operation capabilities and limitations documentation)

Reference: RDPS-REQ-57 - Browser shall document all local-only operation capabilities and limitations (RDPS-0 requirement)

Given: A conformant browser claiming RDPS-0 capability (no remote data processing) for transparent operation

Task: This assessment verifies that browsers provide comprehensive documentation of local-only operation covering offline capabilities (rendering, apps, storage, extensions, security, dev tools), clearly stated limitations (features unavailable offline, no sync, manual updates), operational guidance (installation, backup, troubleshooting, migration), and offline-accessible formats (bundled, PDF, plain text).

Verification:

1. Verify documentation comprehensively describes offline capabilities (rendering, apps, storage, extensions, security, dev tools) → Comprehensive offline capabilities documented (all feature categories)
2. Test that limitations clearly stated (features unavailable offline, no sync, manual updates) → Limitations and unavailable features clearly stated
3. Verify operational guidance provided (installation, backup, troubleshooting, migration) → Operational guidance provided (installation, backup, troubleshooting, migration)
4. Test documentation accessible offline (included with installation, PDF, plain text) → Documentation accessible offline (bundled, PDF, plain text)
5. Verify supported web standards and features documented → Supported web standards and features listed
6. Test that service worker and offline storage capabilities explained → Offline storage APIs and limits documented
7. Verify data management procedures documented (limits, backup, export) → Extension offline compatibility explained
8. Test extension compatibility and limitations clearly explained → Security feature offline operation confirmed
9. Verify security feature offline operation documented → Data backup procedures detailed
10. Test that offline installation procedures clearly described → Troubleshooting and migration guidance provided

Pass Criteria: Comprehensive capability documentation AND clear limitation statements AND operational guidance AND offline-accessible documentation AND web standards coverage AND storage documentation AND extension compatibility AND security documentation AND backup procedures AND troubleshooting guidance

Fail Criteria: Incomplete capability documentation OR unclear limitations OR no operational guidance OR online-only documentation OR missing standards coverage OR no storage documentation OR unclear extension support OR no security documentation OR no backup procedures OR no troubleshooting guidance

Evidence: Documentation completeness review covering all required areas, offline documentation accessibility verification (bundled docs, PDF availability), capability accuracy testing (verify documented features work as described), limitation verification (confirm stated limitations accurate), operational guidance evaluation (installation, backup, troubleshooting procedures)

References:

- Technical Documentation Best Practices: https://www.writethedocs.org/guide/
- Offline Documentation Standards: https://www.w3.org/TR/offline-webapps/
- Air-Gapped System Documentation: https://www.cisecurity.org/controls

Assessment: RDPS-REQ-58 (User notification that no data leaves local system)

Reference: RDPS-REQ-58 - Users shall be informed that no data leaves the local system (RDPS-0 requirement)

Given: A conformant browser claiming RDPS-0 capability (no remote data processing) for transparent privacy protection

Task: This assessment verifies that browsers clearly inform users that no data leaves the local system through multiple mechanisms (setup notification, settings statement, documentation, privacy policy) using clear non-technical language, prominent visibility, benefits explanation, technical verification methods for advanced users, and honest accurate implementation with easy ongoing access to privacy commitments.

Verification:

1. Verify clear notification during installation or first launch stating data remains local → Initial setup notification stating data remains local
2. Test prominent privacy statement in browser settings describing local-only operation → Prominent settings interface privacy statement

3. Verify about or help documentation explains no data transmission policy → About/help documentation explaining policy
4. Test privacy policy contains legally binding commitment to local-only operation → Privacy policy legal commitment to local-only operation
5. Verify optional visual indicators reinforcing local-only status → Clear, non-technical language used
6. Test language used is clear and non-technical (accessible to average users) → Prominent and visible notifications
7. Verify notifications prominent and visible (not buried in lengthy terms) → Benefits clearly explained
8. Test that benefits explained (privacy, offline operation, no dependency) → Contrast with cloud-dependent alternatives explained
9. Verify documentation includes network monitoring procedures for user verification → Technical verification methods documented
10. Test browser provides verification tools (network activity monitor, diagnostic mode) → Easy ongoing access to privacy commitments

Pass Criteria: Multiple notification mechanisms (setup, settings, docs, privacy policy) AND clear non-technical language AND prominent visibility AND benefits explained AND technical verification methods AND honest accurate claims AND easy ongoing access AND documentation complete

Fail Criteria: Single notification only OR technical jargon OR buried in terms OR benefits unexplained OR no verification methods OR inaccurate claims OR difficult to access OR incomplete documentation

Evidence: First-run notification screenshots, settings interface privacy statements, help documentation review, privacy policy legal commitment, language clarity assessment, notification prominence evaluation, verification method documentation, accuracy verification (network monitoring confirms zero transmission)

References:

- Privacy Notice Requirements: https://gdpr.eu/privacy-notice/
- Transparency and Consent: https://www.ftc.gov/business-guidance/privacy-security

Assessment: RDPS-REQ-59 (All RDPS-1 requirements implemented for RDPS-2)

Reference: RDPS-REQ-59 - All RDPS-1 requirements shall be implemented for RDPS-2 capability (RDPS-2-REQ-1 requirement)

Given: A conformant browser claiming RDPS-2 capability (extended remote processing with sensitive data) that should also meet all RDPS-1 requirements

Task: This assessment verifies that browsers claiming RDPS-2 capability fully implement all 15 RDPS-1 requirements (RDPS-REQ-1 through RDPS-REQ-15) as baseline foundation, each meeting specified criteria without degradation from RDPS-2 enhancements, with documentation confirming RDPS-2 includes all RDPS-1 protections and no gaps or exceptions.

Verification:

1. Verify RDPS-1-REQ-1 implemented (offline functionality documented) per RDPS-REQ-1 assessment → RDPS-1-REQ-11 implemented (failure logging) per RDPS-REQ-11
2. Test RDPS-1-REQ-2 implemented (data classification) per RDPS-REQ-2 assessment → RDPS-1-REQ-12 implemented (graceful degradation) per RDPS-REQ-12
3. Verify RDPS-1-REQ-3 implemented (data criticality) per RDPS-REQ-3 assessment → RDPS-1-REQ-13 implemented (credential protection) per RDPS-REQ-13
4. Test RDPS-1-REQ-4 implemented (TLS 1.3+ encryption) per RDPS-REQ-4 assessment → RDPS-1-REQ-14 implemented (rate limiting) per RDPS-REQ-14
5. Verify RDPS-1-REQ-5 implemented (certificate validation) per RDPS-REQ-5 assessment → RDPS-1-REQ-15 implemented (data validation) per RDPS-REQ-15
6. Test RDPS-1-REQ-6 implemented (retry with backoff) per RDPS-REQ-6 assessment → All 15 RDPS-1 requirements fully implemented

7. Verify RDPS-1-REQ-7 implemented (local caching) per RDPS-REQ-7 assessment → Each requirement meets specified criteria
8. Test RDPS-1-REQ-8 implemented (secure authentication) per RDPS-REQ-8 assessment → RDPS-2 enhancements do not compromise RDPS-1 baseline
9. Verify RDPS-1-REQ-9 implemented (certificate pinning) per RDPS-REQ-9 assessment → Documentation confirms RDPS-2 includes all RDPS-1 protections
10. Test RDPS-1-REQ-10 implemented (timeout controls) per RDPS-REQ-10 assessment → No gaps or exceptions in RDPS-1 requirement implementation

Pass Criteria: All 15 RDPS-1 requirements fully implemented AND each meets specified criteria AND no degradation from RDPS-2 enhancements AND documentation confirms inclusion AND no gaps or exceptions

Fail Criteria: Any RDPS-1 requirement not implemented OR any requirement fails criteria OR RDPS-2 compromises RDPS-1 baseline OR documentation does not confirm inclusion OR gaps or exceptions present

Evidence: RDPS-REQ-1 through RDPS-REQ-15 assessment results (all pass), comprehensive requirement coverage verification, baseline security preservation testing, documentation review confirming RDPS-2 includes all RDPS-1 protections, gap analysis showing complete RDPS-1 implementation

References:

- Defense in Depth: https://csrc.nist.gov/glossary/term/defense_in_depth
- Layered Security: https://www.nist.gov/cybersecurity
- Security Capability Levels: https://www.iso.org/standard/56328.html

Assessment: RDPS-REQ-60 (All RDPS-2 requirements implemented for RDPS-3)

Reference: RDPS-REQ-60 - All RDPS-2 requirements shall be implemented for RDPS-3 capability (RDPS-3-REQ-1 requirement)

Given: A conformant browser claiming RDPS-3 capability (full remote processing with critical data requiring maximum security) that should also meet all RDPS-2 requirements

Task: This assessment verifies that browsers claiming RDPS-3 capability fully implement all 18 RDPS-2 requirements (including 15 RDPS-1 requirements via RDPS-2-REQ-1) as verified through RDPS-REQ-1 through RDPS-REQ-32 assessments, each meeting specified criteria without degradation from RDPS-3 enhancements, with documentation confirming RDPS-3 includes all RDPS-2 and RDPS-1 protections and no gaps or exceptions.

Verification:

1. Verify all RDPS-1 requirements implemented via RDPS-REQ-59 assessment (15 requirements) → RDPS-2-REQ-11 (replay defense) per RDPS-REQ-25
2. Test RDPS-2-REQ-2 implemented (encryption at rest) per RDPS-REQ-16 assessment → RDPS-2-REQ-12 (data minimization) per RDPS-REQ-26
3. Verify RDPS-2-REQ-3 implemented (mutual TLS) per RDPS-REQ-17 assessment → RDPS-2-REQ-13 (sync controls) per RDPS-REQ-27
4. Test RDPS-2-REQ-4 implemented (redundant copies) per RDPS-REQ-18 assessment → RDPS-2-REQ-14 (data export) per RDPS-REQ-28
5. Verify RDPS-2-REQ-5 implemented (backup recovery) per RDPS-REQ-19 assessment → RDPS-2-REQ-15 (endpoint config) per RDPS-REQ-29
6. Test RDPS-2-REQ-6 implemented (retention policies) per RDPS-REQ-20 assessment → RDPS-2-REQ-16 (availability) per RDPS-REQ-30
7. Verify RDPS-2-REQ-7 implemented (access controls) per RDPS-REQ-21 assessment → RDPS-2-REQ-17 (pooling) per RDPS-REQ-31
8. Test RDPS-2-REQ-8 implemented (audit logging) per RDPS-REQ-22 assessment → RDPS-2-REQ-18 (token protection) per RDPS-REQ-32
9. Verify RDPS-2-REQ-9 implemented (integrity verification) per RDPS-REQ-23 assessment → All 18 RDPS-2 requirements fully implemented (includes RDPS-1 via RDPS-2-REQ-1)

10. Test RDPS-2-REQ-10 implemented (endpoint protection) per RDPS-REQ-24 assessment → RDPS-3 enhancements do not compromise RDPS-1/RDPS-2 baseline

Pass Criteria: All 18 RDPS-2 requirements fully implemented (including 15 RDPS-1 via RDPS-2-REQ-1) AND each meets specified criteria AND no degradation from RDPS-3 enhancements AND documentation confirms inclusion AND no gaps or exceptions

Fail Criteria: Any RDPS-2 requirement not implemented OR any requirement fails criteria OR RDPS-3 compromises RDPS-2/RDPS-1 baseline OR documentation does not confirm inclusion OR gaps or exceptions present

Evidence: RDPS-REQ-1 through RDPS-REQ-32 assessment results (all pass), comprehensive requirement coverage verification across all three capability levels, baseline security preservation testing, documentation review confirming RDPS-3 includes all RDPS-2 and RDPS-1 protections, gap analysis showing complete RDPS-2 implementation

References:

- Defense in Depth: https://csrc.nist.gov/glossary/term/defense_in_depth
- Layered Security Architecture: https://www.nist.gov/cybersecurity
- Enterprise Security Capability Maturity: https://www.iso.org/standard/56328.html

Annex A (informative): Mapping between the present document and CRA requirements

Table mapping technical security requirements from Section 5 of the present document to essential cybersecurity requirements in Annex I of the CRA. The purpose of this is to help identify missing technical security requirements.

CRA requirement	Technical security requirements(s)
No known exploitable vulnerabilities	UPD-0-REQ-1 through UPD-0-REQ-24 (Forced automatic updates), UPD-1-REQ-1 through UPD-1-REQ-25 (Automatic updates with postponement), LOG-REQ-14 (Incident detection), LOG-REQ-15 (Audit trail completeness), EMB-REQ-9 (JavaScript bridge security review), EXT-REQ-4 (Manifest validation), EXT-REQ-17 (Extension signature validation)
Secure design, development, production	EMB-REQ-9 (JavaScript bridge security review), EXT-REQ-4 (Manifest validation), EXT-REQ-17 (Extension signature validation), UPD-REQ-2 (Update signature verification), UPD-REQ-23 (Binary reproducibility), ENC-REQ-12 (Secure random number generation), SYS-REQ-26 (Sandbox escape prevention), SYS-REQ-27 (Spectre/Meltdown mitigations), SYS-REQ-28 (Side-channel mitigations)

CRA requirement	Technical security requirements(s)
Secure by default configuration	DOM-0-REQ-1 through DOM-0-REQ-6 (Full isolation by default), ENC-0-REQ-1 through ENC-0-REQ-23 (Full encryption by default), DOM-REQ-5 (SameSite=Lax default), DOM-REQ-12 (document.domain restricted by default), ENC-REQ-16 (HTTPS-first mode), UPD-0-REQ-1 (Automatic updates enabled by default), LOG-REQ-9 (User consent for telemetry), SYS-0-REQ-1 through SYS-0-REQ-13 (Sandboxed by default), EMB-0-REQ-1 through EMB-0-REQ-7 (Isolated by default)
Secure updates	UPD-REQ-1 (Automatic update mechanism), UPD-REQ-2 (Update signature verification), UPD-REQ-3 (HTTPS-only delivery), UPD-REQ-4 (Manifest integrity), UPD-REQ-5 (Rollback protection), UPD-REQ-6 (Channel isolation), UPD-REQ-7 (Component updates), UPD-REQ-8 (Emergency updates), UPD-REQ-9 (Verification before installation), UPD-REQ-10 (Failure recovery), UPD-REQ-11 (Transparency logging), UPD-REQ-12 (Delta update security), UPD-REQ-13 (Server authentication), UPD-REQ-14 (Timing jitter), UPD-REQ-15 (Background enforcement), UPD-REQ-16 (Notification UI), UPD-REQ-17 (Forced critical updates), UPD-REQ-18 (Verification chain), UPD-REQ-19 (Source pinning), UPD-REQ-20 (Integrity verification), UPD-REQ-21 (Staged rollout), UPD-REQ-22 (Domain validation), UPD-REQ-23 (Binary reproducibility), EXT-REQ-10 (Extension update verification)

CRA requirement	Technical security requirements(s)
Authentication and access control mechanisms	DOM-REQ-1 (Process-per-site isolation), DOM-REQ-3 (Cross-origin DOM access prevention), DOM-REQ-4 (CORS preflight), DOM-REQ-5 (SameSite cookies), DOM-REQ-6 (Storage isolation), EXT-REQ-1 (Extension permission model), EXT-REQ-3 (Extension API access control), EXT-REQ-7 (Host permissions validation), SYS-REQ-6 (Device API permissions), SYS-REQ-7 (PWA permission management), SYS-REQ-8 through SYS-REQ-19 (Device-specific permissions), EMB-REQ-1 (JavaScript bridge API allowlists), EMB-REQ-5 (User consent for sensitive operations), EMB-REQ-11 (Granular capability-based permissions), PRO-REQ-2 (User consent for custom protocols), PRO-REQ-3 (Protocol allowlist enforcement)
Confidentiality protection	ENC-REQ-1 (TLS 1.3+ support), ENC-REQ-2 (Certificate validation), ENC-REQ-3 (Certificate pinning), ENC-REQ-4 (HSTS enforcement), ENC-REQ-5 (Mixed content blocking), ENC-REQ-6 (Certificate Transparency), ENC-REQ-11 (Web Crypto API), ENC-REQ-13 (Subresource Integrity), ENC-REQ-14 (Encrypted SNI/ECH), ENC-REQ-16 (HTTPS-first mode), ENC-REQ-20 (Cryptographic key isolation), ENC-REQ-21 (Certificate store security), DOM-REQ-2 (CORB), DOM-REQ-6 (Storage isolation), EMB-REQ-4 (Context isolation), EMB-REQ-8 (Host credential protection), EMB-REQ-12 (Storage isolation from host), EMB-REQ-14 (Encrypted cross-process bridge), EMB-REQ-17 (Certificate validation for embedded content), EMB-REQ-21 (Mixed content prevention), EMB-REQ-27 (Network security configuration)

CRA requirement	Technical security requirements(s)
Integrity protection for data and configuration	ENC-REQ-2 (Certificate validation), ENC-REQ-13 (Subresource Integrity), UPD-REQ-2 (Update signature verification), UPD-REQ-4 (Update manifest integrity), UPD-REQ-5 (Rollback protection), UPD-REQ-20 (Update integrity verification), LOG-REQ-11 (Log integrity protection), EMB-REQ-2 (JavaScript bridge input validation), EMB-REQ-7 (Immutable bridge configuration), EMB-REQ-19 (SRI for embedded content), EMB-REQ-23 (Cryptographic signature verification for local content), EXT-REQ-4 (Manifest validation), EXT-REQ-17 (Extension signature validation), DOM-REQ-9 (CORP), DOM-REQ-11 (COEP)
Data minimization	LOG-REQ-7 (Log data minimization), LOG-REQ-8 (Log anonymization), LOG-REQ-12 (Log retention policies), LOG-REQ-18 (Privacy-preserving analytics), EXT-REQ-16 (Extension telemetry privacy), DOM-REQ-6 (Storage isolation limits data sharing), EMB-REQ-12 (Storage isolation from host), PRO-REQ-5 (Protocol parameter sanitization to prevent data leakage)
Availability protection	SYS-REQ-20 (Hardware resource limits), SYS-REQ-21 (Memory isolation), SYS-REQ-22 (CPU quotas), SYS-REQ-23 (Network bandwidth limits), SYS-REQ-24 (Storage quotas), SYS-REQ-25 (Process priority management), UPD-REQ-10 (Update failure recovery), UPD-REQ-21 (Staged rollout), EMB-REQ-10 (Bridge API rate limiting), EXT-REQ-5 (Extension sandboxing to prevent interference)
Minimize impact on other devices or services	DOM-REQ-1 (Process-per-site isolation), SYS-REQ-1 (Process sandbox enforcement), SYS-REQ-2 (Renderer process isolation), SYS-REQ-3 (GPU process isolation), SYS-REQ-4 (Network service isolation), SYS-REQ-20 (Resource limits), SYS-REQ-26 (Sandbox escape prevention), EXT-REQ-5 (Extension sandboxing), EXT-REQ-6 (Cross-extension isolation), EMB-REQ-4 (Context isolation), PRO-REQ-13 (Handler capability restrictions)

CRA requirement	Technical security requirements(s)
Limit attack surface	EXT-0-REQ-1 through EXT-0-REQ-3 (No extension support), SYS-0-REQ-1 through SYS-0-REQ-13 (Fully sandboxed), EMB-0-REQ-1 through EMB-0-REQ-7 (No JavaScript bridge), PRO-0-REQ-1 through PRO-0-REQ-5 (HTTP/HTTPS only), DOM-REQ-7 (iframe sandboxing), DOM-REQ-8 (Opaque origin handling), EMB-REQ-6 (No system-level API exposure), EMB-REQ-16 (Allowlists over denylists), EXT-REQ-12 (Background script restrictions), ENC-REQ-19 (Legacy crypto deprecation)
Exploit mitigation by limiting incident impact	DOM-REQ-1 (Process-per-site isolation limits cross-site impact), SYS-REQ-1 (Sandbox enforcement), SYS-REQ-2 (Process isolation), SYS-REQ-21 (Memory isolation), SYS-REQ-26 (Sandbox escape prevention), SYS-REQ-27 (Spectre/Meltdown mitigations), SYS-REQ-28 (Side-channel mitigations), DOM-REQ-2 (CORB), DOM-REQ-9 (CORP), DOM-REQ-10 (COOP), DOM-REQ-11 (COEP), EXT-REQ-2 (Content script isolation), EXT-REQ-5 (Extension sandboxing), EXT-REQ-6 (Cross-extension isolation), EMB-REQ-4 (Context isolation), UPD-REQ-5 (Rollback protection), UPD-REQ-21 (Staged rollout limits blast radius)
Logging and monitoring mechanisms	LOG-REQ-1 (Security event logging), LOG-REQ-2 (Certificate error logging), LOG-REQ-3 (Extension security events), LOG-REQ-4 (CSP violation reporting), LOG-REQ-5 (Network Error Logging), LOG-REQ-6 (Crash reporting), LOG-REQ-10 (Secure log transmission), LOG-REQ-11 (Log integrity protection), LOG-REQ-13 (Security dashboard), LOG-REQ-14 (Incident detection), LOG-REQ-15 (Audit trail completeness), LOG-REQ-16 (Real-time security alerts), LOG-REQ-17 (Forensic log export), LOG-REQ-19 (Compliance logging), LOG-REQ-20 (Log access controls), EMB-REQ-3 (JavaScript bridge logging), EMB-REQ-22 (Trust decision logging), EMB-REQ-32 (Trust boundary violation events), UPD-REQ-11 (Update transparency logging), PRO-REQ-9 (Protocol handler logging)

CRA requirement	Technical security requirements(s)
Secure deletion and data transfer	DOM-REQ-6 (Storage isolation enables secure per-origin deletion), ENC-REQ-1 (TLS 1.3+ for secure transfer), ENC-REQ-3 (Certificate pinning for critical transfers), ENC-REQ-5 (Mixed content blocking), EMB-REQ-17 (Certificate validation for embedded content transfers), EMB-REQ-21 (Mixed content prevention), EMB-REQ-27 (Network security configuration), SYS-REQ-24 (Storage quotas with cleanup mechanisms), EXT-REQ-11 (Extension storage isolation enables clean uninstall)

Annex B (informative): Mapping of Use Cases to Capabilities and Requirements

This annex provides a comprehensive mapping of each use case defined in Section 4.4 to the relevant browser capabilities and their associated requirement sets. This mapping helps manufacturers and assessors identify which requirements apply to specific deployment contexts.

B.1 Use Case Mapping Methodology

For each use case, the mapping identifies:

1. **Primary Capabilities**: Core security capabilities that are essential for the use case
2. **Recommended Condition Levels**: Specific condition levels (e.g., DOM-1, EXT-2) appropriate for the use case's risk profile
3. **Critical Requirements**: Specific requirement sets that are satisfied for the use case
4. **Optional Enhancements**: Additional requirements that may be appropriate based on deployment specifics

B.2 Use Case to Capability Mappings

UC-B1: General Purpose Web Browsing (Risk Level: Standard)

Primary Capabilities and Recommended Conditions:

- **DOM (Domain/Origin Isolation)**: DOM-1 (Controlled isolation)
- **EXT (Extension System)**: EXT-1 or EXT-2
- **ENC (Encryption)**: ENC-1
- **LOG (Logging/Monitoring)**: LOG-1
- **UPD (Updates)**: UPD-1
- **PRO (Protocol Handlers)**: PRO-1
- **SYS (System Resources)**: SYS-1

Critical Requirements: DOM-1-REQ-1 through DOM-1-REQ-9, ENC-1-REQ-1 through ENC-1-REQ-19, UPD-1-REQ-1 through UPD-1-REQ-25, EXT-1-REQ-1 through EXT-1-REQ-14, LOG-1-REQ-1 through LOG-1-REQ-18, SYS-1-REQ-1 through SYS-1-REQ-22

Assessment References: All DOM, ENC, UPD, EXT, LOG, PRO, SYS assessments apply

---

UC-B2: Development and Testing Environments (Risk Level: High)

Primary Capabilities and Recommended Conditions:

- **DOM**: DOM-2
- **EXT**: EXT-2
- **ENC**: ENC-1
- **LOG**: LOG-2
- **UPD**: UPD-1 or UPD-2
- **PRO**: PRO-2
- **SYS**: SYS-2

Critical Requirements: DOM-2-REQ-1 through DOM-2-REQ-12, EXT-2-REQ-1 through EXT-2-REQ-10, LOG-2-REQ-1 through LOG-2-REQ-20, PRO-2-REQ-1 through PRO-2-REQ-12, SYS-2-REQ-1 through SYS-2-REQ-15

Assessment References: All capability assessments, emphasis on EXT-REQ-9, DOM-REQ-9-11, SYS-REQ-14-17

UC-B3: Kiosks and Shared Terminals (Risk Level: High)

Primary Capabilities and Recommended Conditions:

- **DOM**: DOM-0 or DOM-1
- **EXT**: EXT-0
- **ENC**: ENC-0 or ENC-1
- **LOG**: LOG-3
- **UPD**: UPD-0
- **PRO**: PRO-0
- **SYS**: SYS-0
- **EMB** (if embedded): EMB-1 or EMB-2

Critical Requirements: DOM-0-REQ-1 through DOM-0-REQ-6, EXT-0-REQ-1 through EXT-0-REQ-3, ENC-0-REQ-1 through ENC-0-REQ-23, LOG-3-REQ-1 through LOG-3-REQ-20, UPD-0-REQ-1 through UPD-0-REQ-24, PRO-0-REQ-1 through PRO-0-REQ-5, SYS-0-REQ-1 through SYS-0-REQ-13

If Embedded: EMB-1-REQ-1 through EMB-1-REQ-17, EMB-REQ-8, EMB-REQ-3, EMB-REQ-22

Additional: Domain allowlisting, session auto-termination, no credential storage, remote monitoring

Assessment References: Strictest criteria; DOM-REQ-1-8, ENC-REQ-1-6, UPD-REQ-1-11, SYS-REQ-1-4, LOG-REQ-10-11

UC-B4: Financial Services Access (Risk Level: High)

Primary Capabilities and Recommended Conditions:

- **DOM**: DOM-1
- **EXT**: EXT-1
- **ENC**: ENC-0 or ENC-1
- **LOG**: LOG-1
- **UPD**: UPD-0 or UPD-1
- **PRO**: PRO-1
- **SYS**: SYS-1
- **EMB** (if embedded): EMB-1 or EMB-2

Critical Requirements: ENC-0-REQ-1 through ENC-0-REQ-23 OR ENC-1-REQ-1 through ENC-1-REQ-19, DOM-1-REQ-1 through DOM-1-REQ-9, EXT-1-REQ-1 through EXT-1-REQ-14, LOG-REQ-2, LOG-REQ-14

If Embedded: EMB-1-REQ-1 through EMB-1-REQ-17, EMB-REQ-17, EMB-REQ-20, EMB-REQ-2, EMB-REQ-8

Assessment References: ENC-REQ-1-7, ENC-REQ-17, DOM-REQ-5, LOG-REQ-2, EMB-REQ-1-10 (if embedded)

UC-B5: Healthcare and Medical Systems (Risk Level: High)

Primary Capabilities and Recommended Conditions:

- **DOM**: DOM-1 or DOM-2
- **EXT**: EXT-1
- **ENC**: ENC-0
- **LOG**: LOG-3
- **UPD**: UPD-0 or UPD-1
- **PRO**: PRO-1
- **SYS**: SYS-1
- **EMB** (if embedded): EMB-1 or EMB-2

Critical Requirements: ENC-0-REQ-1 through ENC-0-REQ-23, LOG-3-REQ-1 through LOG-3-REQ-20, LOG-REQ-7, LOG-REQ-8, LOG-REQ-19, DOM-1-REQ-1 through DOM-1-REQ-9, UPD-0-REQ-17, EXT-1-REQ-1 through EXT-1-REQ-14

If Embedded: EMB-1-REQ-1 through EMB-1-REQ-17, EMB-REQ-8, EMB-REQ-3, EMB-REQ-22

Compliance: GDPR data protection, session re-auth, auto-timeout, audit trails

Assessment References: ENC-REQ-1-7, LOG-REQ-7-9, LOG-REQ-19, LOG-REQ-11, EMB-REQ-3, EMB-REQ-22 (if embedded)

UC-B6: E-Government Services Access (Risk Level: High)

Primary Capabilities and Recommended Conditions:

- **DOM**: DOM-1
- **EXT**: EXT-1
- **ENC**: ENC-0
- **LOG**: LOG-2 or LOG-3
- **UPD**: UPD-0 or UPD-1
- **PRO**: PRO-1
- **SYS**: SYS-1 or SYS-2

Critical Requirements: ENC-0-REQ-1 through ENC-0-REQ-23, ENC-REQ-2, ENC-REQ-3, SYS-REQ-29, DOM-1-REQ-1 through DOM-1-REQ-9, LOG-REQ-15, UPD-0-REQ-1 through UPD-0-REQ-24

Special: Digital signatures, smart card integration, eIDAS compliance, legal non-repudiation

Assessment References: ENC-REQ-2-3, ENC-REQ-6, SYS-REQ-29, LOG-REQ-15

UC-B7: Enterprise Applications (Risk Level: High)

Primary Capabilities and Recommended Conditions:

- **DOM**: DOM-2
- **EXT**: EXT-1 or EXT-2
- **ENC**: ENC-1
- **LOG**: LOG-3
- **UPD**: UPD-0 or UPD-1
- **PRO**: PRO-2
- **SYS**: SYS-2
- **EMB** (if Electron/CEF/Tauri): EMB-2 or EMB-3

Critical Requirements: DOM-2-REQ-1 through DOM-2-REQ-12, LOG-3-REQ-1 through LOG-3-REQ-20, LOG-REQ-13, LOG-REQ-14, LOG-REQ-16, EXT-2-REQ-10, UPD-0-REQ-1 through UPD-0-REQ-24, PRO-2-REQ-11, SYS-2-REQ-11 through SYS-2-REQ-15

Enterprise Features: SSO, DLP, extension allowlisting, profile separation, BYOD containerization

Assessment References: DOM-REQ-9-11, LOG-REQ-13-16, LOG-REQ-19, EXT-REQ-3, PRO-REQ-3, SYS-REQ-7

UC-B8: Critical Infrastructure (Risk Level: CRITICAL)

Primary Capabilities and Recommended Conditions:

- **DOM**: DOM-0 or DOM-1
- **EXT**: EXT-0
- **ENC**: ENC-0
- **LOG**: LOG-3
- **UPD**: UPD-0
- **PRO**: PRO-0 or PRO-1
- **SYS**: SYS-0 or SYS-1
- **EMB** (if SCADA/ICS): EMB-0 or EMB-1

Critical Requirements: DOM-0-REQ-1 through DOM-0-REQ-6, EXT-0-REQ-1 through EXT-0-REQ-3, ENC-0-REQ-1 through ENC-0-REQ-23, LOG-3-REQ-1 through LOG-3-REQ-20, LOG-REQ-11, UPD-0-REQ-1 through UPD-0-REQ-24, UPD-REQ-5, UPD-REQ-11, PRO-0-REQ-1 through PRO-0-REQ-5, SYS-0-REQ-1 through SYS-0-REQ-13

If Embedded: EMB-0-REQ-1 through EMB-0-REQ-7, EMB-REQ-17, EMB-REQ-20, EMB-REQ-27, EMB-REQ-31

Additional: Zero trust, mTLS, RBAC, air-gapped deployment, supply chain controls, physical security

Assessment References: ALL assessments at strictest criteria; ENC-REQ-1-11, UPD-REQ-1-11, LOG-REQ-10-11, SYS-REQ-26-28, EMB-REQ-17-31 (if embedded)

UC-B9: Security Research (Risk Level: CRITICAL)

Primary Capabilities and Recommended Conditions:

- **DOM**: DOM-2 or DOM-3
- **EXT**: EXT-3
- **ENC**: ENC-1
- **LOG**: LOG-3
- **UPD**: UPD-2 or UPD-3

- **PRO**: PRO-3
- **SYS**: SYS-3

Critical Requirements: LOG-3-REQ-1 through LOG-3-REQ-20, LOG-REQ-6, LOG-REQ-17, DOM-3-REQ-9, EXT-3-REQ-9 through EXT-3-REQ-12, SYS-3-REQ-15

Environment Isolation: Disposable VMs, network capture, air-gapped zones, snapshot/rollback, behavioral logging

Important: Deploy ONLY in isolated research environments; NOT for production

Assessment References: LOG-REQ-1-20, LOG-REQ-17, EXT-REQ-4, all assessments in adversarial conditions

---

UC-B10: Adapted Browser with Modified Features (Risk Level: Standard to High)

Primary Capabilities and Recommended Conditions:

- **All capabilities**: Inherit from upstream browser
- **LOG**: May vary (LOG-1, LOG-2, or LOG-3)
- **UPD**: UPD-0 or UPD-1 (manufacturer-controlled)
- **EMB** (if native integration added): EMB-2 or EMB-3

Critical Requirements: All upstream requirements PLUS UPD-REQ-2, UPD-REQ-11, EMB-REQ-9, LOG-REQ-9, LOG-REQ-7, LOG-REQ-8

If Native Integration Added: EMB-2-REQ-1 through EMB-2-REQ-10 OR EMB-3-REQ-1 through EMB-3-REQ-12, EMB-REQ-1, EMB-REQ-2, EMB-REQ-3, EMB-REQ-9

For Bundled Extensions: EXT-REQ-4, EXT-REQ-17, supply chain security

Manufacturer Obligations: Timely upstream patches, security review, transparency, maintaining security controls, supply chain security

Risk Level: Standard (minimal modifications) to High (extensive modifications, sensitive data, high-risk deployment)

Assessment References: All upstream assessments PLUS EMB-REQ-9, UPD-REQ-2, LOG-REQ-9, EMB assessments if native integration

---

B.3 Capability Condition Level Selection Guide

Use Case Risk	DOM	EXT	ENC	LOG	UPD	PRO	SYS	EMB	RDPS
Standard	DOM-1	EXT-1/2	ENC-1	LOG-1	UPD-1	PRO-1	SYS-1	EMB-1	RDPS-0/1
High	DOM-1/2	EXT-0/1	ENC-0/1	LOG-2/3	UPD-0/1	PRO-0/1	SYS-0/1/2	EMB-0/1/2	RDPS-0/2
Critical	DOM-0/1	EXT-0	ENC-0	LOG-3	UPD-0	PRO-0	SYS-0/1	EMB-0/1	RDPS-0/3

Note: Specific deployments shall conduct detailed risk assessments per Annex D to determine appropriate condition levels.

RDPS Note: RDPS capability level selection depends on whether remote data processing is used and the sensitivity of data processed. RDPS-0 (no remote processing) is always acceptable and mandatory for air-gapped deployments. When remote processing is used, select RDPS level based on data sensitivity: RDPS-1 for non-sensitive data, RDPS-2 for sensitive data, RDPS-3 for critical data with regulatory requirements.

B.4 Cross-Reference to Assessments

All assessments in Chapter 6 map to requirements referenced in this annex:

- **Section 6.1**: DOM-REQ-1 through DOM-REQ-12
- **Section 6.2**: EXT-REQ-1 through EXT-REQ-18
- **Section 6.3**: ENC-REQ-1 through ENC-REQ-21
- **Section 6.4**: LOG-REQ-1 through LOG-REQ-20
- **Section 6.5**: UPD-REQ-1 through UPD-REQ-23
- **Section 6.6**: PRO-REQ-1 through PRO-REQ-23
- **Section 6.7**: SYS-REQ-1 through SYS-REQ-32
- **Section 6.8**: EMB-REQ-1 through EMB-REQ-32
- **Section 6.6.5**: RDPS-REQ-1 through RDPS-REQ-60

B.5 Remote Data Processing Systems (RDPS) Mapping

RDPS Capabilities are independent of deployment use cases but apply when browsers employ remote data processing for any functionality. The appropriate RDPS capability level should be selected based on data sensitivity and criticality:

RDPS-0: No Remote Data Processing (Fully Local Operation)

Applicable to: - **UC-B3**: Kiosks and Shared Terminals (air-gapped deployments) - **UC-B8**: Critical Infrastructure (air-gapped SCADA/ICS systems) - Any deployment requiring complete network isolation

Requirements: RDPS-0-REQ-1 through RDPS-0-REQ-7

Assessment References: RDPS-REQ-52 through RDPS-REQ-58

Key Characteristics: Zero network connectivity, all data local-only, no telemetry, no remote authentication, complete offline operation

---

RDPS-1: Limited Remote Processing (Non-Sensitive Data)

Applicable to: - **UC-B1**: General Purpose Web Browsing (preferences sync, bookmark sync) - **UC-B2**: Development and Testing Environments (extension sync, settings sync) - **UC-B10**: Adapted Browsers (non-sensitive preference synchronization)

Requirements: RDPS-1-REQ-1 through RDPS-1-REQ-15

Assessment References: RDPS-REQ-1 through RDPS-REQ-15

Key Characteristics: TLS 1.3+ encryption, certificate validation, graceful offline degradation, rate limiting, non-sensitive data only (configuration, preferences, non-critical bookmarks)

Data Examples: UI preferences, theme settings, non-sensitive bookmarks, display configuration, language preferences

---

RDPS-2: Extended Remote Processing (Sensitive Data)

Applicable to: - **UC-B4**: Financial Services (session state, transaction logs) - **UC-B5**: Healthcare and Medical Systems (audit logs, anonymized analytics) - **UC-B6**: E-Government Services (authentication state, encrypted form data) - **UC-B7**: Enterprise Applications (SSO tokens, policy sync, encrypted data sync)

Requirements: All RDPS-1 requirements PLUS RDPS-2-REQ-1 through RDPS-2-REQ-18

Assessment References: RDPS-REQ-1 through RDPS-REQ-32, RDPS-REQ-59

Key Characteristics: Data encryption at rest, mutual TLS, redundant backups, per-user per-origin access controls, audit logging, integrity verification, replay attack defense, data minimization

Data Examples: Authentication tokens, encrypted passwords, financial transaction logs, healthcare audit trails, enterprise policy data, encrypted user documents

Special Considerations: - GDPR compliance required for EU deployments - Sector-specific regulations for healthcare (UC-B5) - Financial services regulatory requirements (UC-B4) - Enterprise data residency requirements (UC-B7)

RDPS-3: Full Remote Processing (Critical Data - Maximum Security)

Applicable to: - **UC-B5**: Healthcare and Medical Systems (patient data, medical records - where remote processing is legally permitted) - **UC-B6**: E-Government Services (citizen PII, legal documents, classified data) - **UC-B7**: Enterprise Applications (trade secrets, financial records, strategic data) - **UC-B8**: Critical Infrastructure (control data, operational parameters - where remote processing is absolutely necessary and properly secured)

Requirements: All RDPS-1 and RDPS-2 requirements PLUS RDPS-3-REQ-1 through RDPS-3-REQ-20

Assessment References: RDPS-REQ-1 through RDPS-REQ-51, RDPS-REQ-59, RDPS-REQ-60

Key Characteristics: End-to-end encryption, hardware-backed keys, high availability with failover, disaster recovery, real-time integrity monitoring, SIEM integration, zero-trust architecture, compliance logging, automated security scanning, incident response procedures, access revocation, transparency reporting, forward secrecy, user notifications, enterprise policy enforcement

Data Examples: Medical records, patient health information, classified government data, trade secrets, financial statements, critical infrastructure operational data, personal identifiable information (PII)

Regulatory Compliance: - GDPR Article 32 (Security of Processing) - full compliance required - eIDAS Regulation (for e-government - UC-B6) - NIS2 Directive (for critical infrastructure - UC-B8) - Sector-specific EU regulations (healthcare, financial services) - ISO 27001/27017/27018 certifications recommended

Special Considerations: - Geographic data residency enforcement required - Multi-tenant isolation mandatory - Cryptographic proof of integrity - 24/7 incident response capability - Regular penetration testing and security audits - Documented disaster recovery with tested procedures - Enterprise administrator security policy controls

---

RDPS Capability Selection Matrix by Use Case

Use Case	Recommended RDPS Level	Data Types	Key Controls
UC-B1 (General Browsing)	RDPS-0 or RDPS-1	Preferences, bookmarks	Graceful offline, TLS 1.3+
UC-B2 (Development/Testing)	RDPS-1	Settings, extensions	Sync controls, rate limiting

Use Case	Recommended RDPS Level	Data Types	Key Controls
UC-B3 (Kiosks)	RDPS-0 (mandatory)	None (local only)	No remote processing
UC-B4 (Financial)	RDPS-2 or RDPS-3	Tokens, transactions	Encryption at rest, mTLS, audit logs
UC-B5 (Healthcare)	RDPS-2 or RDPS-3	Audit logs, patient data	Sector regulations, E2EE, DR
UC-B6 (E-Government)	RDPS-2 or RDPS-3	Citizen PII, documents	Data residency, zero-trust, compliance logging
UC-B7 (Enterprise)	RDPS-2 or RDPS-3	Enterprise data, policies	Enterprise controls, SIEM, HA
UC-B8 (Critical Infrastructure)	RDPS-0 (preferred) or RDPS-3	Control data	Air-gap preferred; if remote: max security
UC-B9 (Security Research)	RDPS-0 or RDPS-1	Research data	Isolated environments, no sensitive data
UC-B10 (Adapted Browser)	Inherit from use case	Depends on deployment	Match upstream + manufacturer obligations

Important Notes:

1. **RDPS-0 is mandatory** for air-gapped deployments (UC-B3 kiosks, UC-B8 critical infrastructure in isolated networks)
2. **RDPS capability levels are additive**: RDPS-2 includes all RDPS-1 requirements; RDPS-3 includes all RDPS-1 and RDPS-2 requirements
3. **Data classification drives RDPS level**: Manufacturers shall classify all remotely processed data and select appropriate RDPS level based on highest sensitivity
4. **Regulatory compliance**: RDPS-3 is recommended for all use cases with regulatory requirements (GDPR, NIS2, eIDAS, sector-specific regulations, etc.)
5. **User control**: For RDPS-1 and above, users should have transparency and control over what data is processed remotely
6. **Enterprise deployments**: UC-B7 should typically use RDPS-2 or RDPS-3 with enterprise policy controls (RDPS-3-REQ-20)

Annex C (informative): Relationship between the present document and any related ETSI standards (if any)

List any related ETSI standards and how they interact with the present document.

Annex D (informative): Risk identification and assessment methodology

D.1 Assets

D.1.1 Data

What data is stored on the product?

D.1.2 Product functions

See the functions in Section 4.4.

D.2 Threats

Based on the assets, what are the threats during:

- *Use for intended purpose or reasonably foreseeable use*
- *When integrated into another product*

Example threats can be found in the same documents suggested in the section on security requirements.

D.3 Assumptions

List assumptions that are relevant to the risk analysis for these threats. Everything is hackable if you try hard enough. What kinds of threats are in and out of scope? What are you assuming is the sophistication of attack? Relate to use cases. Some examples might include:

- *Not being attacked by a state actor*
- *Not using sophisticated or expensive hardware snooping techniques*
- *No secret hardware backdoors in other components*

D.4 Risk assessments of threats

For each threat identified above, use likelihood and magnitude of the threat to assess its risk in the context of use cases. The results should be consistent with the mapping of use cases to security levels.

Guidance from latest PT1 draft:

> *An analysis in terms of likelihood and magnitude of a product's threats is required to be able to determine the product's risks. NOTE 1 This document does not require a specific methodology for a cybersecurity risk analysis as long as the cybersecurity risk estimation is based on the likelihood of occurrence and magnitude of loss or disruption of cybersecurity risks. Thus, different approaches and models such as the fishbone model, event tree analysis or fault tree models can be used within the analysis of cybersecurity risks. NOTE 2 A qualitative estimation of the cybersecurity risks can be performed using risk matrices that map qualitative categories of the likelihood of occurrence and qualitative categories of magnitude of loss or disruption to cybersecurity risk categories. NOTE 3 A quantitative estimation of the cybersecurity risks can be performed using scoring systems that map qualitative categories of the likelihood of occurrence and qualitative categories of magnitude of loss or disruption to certain values.*

Annex E (informative): Risk evaluation guidance

E.1 Mapping of risks to requirements

Table mapping the identified risks to requirements

E.2 Risks not treated by the requirements

If any risks are not treated by the normative requirements, describe non-normative suggestions to mitigate them.

E.3 Risk acceptance criteria

Describe how to decide if residual risks are tolerable.

E.4 Residual risks

Describe how to treat any residual risks, for example by documenting them or informing the user.

Annex K

Crypto todo

https://certification.enisa.europa.eu/publications/eucc-guidelines-cryptography_en

Annex L (informative): Relationship between the present document and the requirements of EU Regulation 2024/2847

DRAFT ANNEX L - DO NOT CONSIDER THE CONTENT

The present document has been prepared under the Commission's standardisation request C(2025) 618 final to provide one voluntary means of conforming to the requirements of Regulation (EU) No 2024/2847 of the European Parliament and of the Council of 23 October 2024 on horizontal cybersecurity requirements for products with digital elements and amending Regulations (EU) No 168/2013 and (EU) No 2019/1020 and Directive (EU) 2020/1828 (Cyber Resilience Act). Once the present document is cited in the Official Journal of the European Union under that Regulation, compliance with the normative clauses of the present document given in table A.1 confers, within the limits of the scope of the present document, a presumption of conformity with the corresponding requirements of that Regulation and associated EFTA regulations. > NOTE: The above paragraphs have to be repeated in the Foreword.

The annex shall have a table for a clear indication of correspondence between normative clauses of the standard and the legal requirements aimed to be covered.

It should be evaluated - on the basis of the legal requirements supported and other information given in a harmonised standard - how detailed correspondence can be indicated between the normative elements of the harmonised standard and the legal requirements aimed to be covered. However, where this correspondence is expressed in too general terms, it could lead to a situation where the Commission cannot assess whether the Harmonised Standard satisfies the requirements, which it aims to cover, and subsequently publication of its references in the OJEU according to Article 10(6) of the Regulation is significantly delayed or is not possible at all.

Annex : Change history

Date	Version	Information about changes
<Month year>	<#>	<Changes made are listed in this cell>

History

Version	Date	Milestone
	<#>	

Version	Date	Milestone