Certified Web Defense & Cryptography Specialist

Examination Scope: 30 Questions | 35 Minutes | Passing Standard: 80%

1. EXECUTIVE OVERVIEW & CREDENTIAL STANDARDS

Advanced credential in enterprise-grade TLS 1.3 hardening, granular Content Security Policy (CSP), origin isolation (COOP/COEP/

2. KEY TOPICS & SYLLABUS

* TLS 1.3 cryptographic primitives and 0-RTT replay defenses
* Granular CSP: Nonce-based, hash-based, and strict-dynamic policies
* Cross-Origin Opener Policy (COOP) and Spectre mitigation
* Cross-Origin Embedder Policy (COEP) and Resource Policy (CORP)
* Permissions-Policy and browser feature restrictions
* HSTS Preload submission criteria and multi-domain implications

Examination Integrity & Anti-Cheat Protocols:

* Correct answer keys are evaluated strictly server-side and never sent to client devices.
* Questions and multiple-choice options are dynamically shuffled per individual session.
* Time limits are strictly enforced on the server; background blur/tab switches are audited.

CHAPTER 1: Modern Cryptography: TLS 1.3 Architecture

Forward secrecy, eliminating legacy ciphers, and 0-RTT replay risks.

TLS 1.3 (RFC 8446) fundamentally restructured transport security by dropping obsolete, fragile cryptographic mechanisms:

- Static RSA key exchange is removed: All TLS 1.3 sessions strictly mandate Ephemeral Diffie-Hellman (ECDHE/DHE), guaranteeing Perfect Forward Secrecy (PFS). If a server private key is compromised in the future, past traffic cannot be retroactively decrypted.
- Legacy symmetric ciphers (RC4, DES, 3DES, CBC-mode AES) are purged. Only authenticated encryption with associated data (AEAD) algorithms are allowed (e.g. AES-128-GCM, AES-256-GCM, CHACHA20-POLY1305).
- 0-RTT (Early Data): While 0-RTT accelerates resumption, it is vulnerable to network replay attacks. Sensitive state-changing endpoints must reject Early Data.

```
# OpenSSL / Nginx TLS 1.3 Cipher Suite
ssl_protocols TLSv1.3;
ssl_conf_command Ciphersuites TLS_AES_128_GCM_SHA256:TLS_AES_256_GCM_SHA384:TLS_CHACHA20_POLY1305_SHA256;
```

CHAPTER 2: Deep Content Security Policy (CSP)

Eliminating unsafe-inline with cryptographically random nonces and hashes.

A weak CSP that allows 'unsafe-inline' or 'unsafe-eval' provides virtually zero defense against Cross-Site Scripting (XSS). Modern enterprise CSP uses nonce-based or hash-based allowlisting:

1. Nonce-based: Generate a cryptographically random, base64-encoded string on every single HTTP response. Only scripts carrying the matching nonce attribute (<script nonce="rAnd0m...">) execute.
2. strict-dynamic: When combined with a nonce, strict-dynamic allows trusted root scripts to load additional dependencies dynamically without enumerating third-party CDN URLs in the policy.
3. frame-ancestors: Supersedes X-Frame-Options, offering fine-grained control over which external origins can embed the page.

```
Content-Security-Policy: default-src 'self'; script-src 'nonce-dGVzdDEyMzQ1Ng==' 'strict-dynamic'; object-
```

CHAPTER 3: Cross-Origin Isolation: COOP, COEP & CORP

Guarding memory boundaries against Spectre, Meltdown, and XS-Leaks.

Spectre-class microarchitectural side-channel attacks allow malicious code to read arbitrary data across memory boundaries. Browsers mitigated this by introducing Cross-Origin Isolation:

- Cross-Origin-Opener-Policy (COOP): Set to same-origin to isolate your top-level browsing context into its own operating system process, severing window.opener cross-origin references.
- Cross-Origin-Embedder-Policy (COEP): Set to require-corp to ensure all embedded subresources (images, scripts, iframes) explicitly opt-in to being loaded.
- Cross-Origin-Resource-Policy (CORP): Set to same-origin to prevent other origins from embedding your sensitive resources.

© 2026 SurfaceScan · CWDS Certification Curriculum

Enabling full cross-origin isolation unlocks powerful Web APIs like SharedArrayBuffer and high-resolution performance timers.

```
Cross-Origin-Opener-Policy: same-origin
Cross-Origin-Embedder-Policy: require-corp
Cross-Origin-Resource-Policy: same-origin
```

CHAPTER 4: Permissions-Policy & Feature Delegation

Restricting dangerous browser hardware APIs and sensors.

Permissions-Policy (formerly Feature-Policy) provides explicit control over which powerful browser features and hardware APIs may be used by the page and embedded iframes:

Restricting camera, microphone, geolocation, payment, usb, and accelerometer ensures that even if an attacker manages to inject malicious JavaScript, they cannot silently activate cameras, steal hardware tokens, or track the physical location of the user.

```
Permissions-Policy: camera=(), microphone=(), geolocation=(), payment=(), usb=()
```