Web Application Security & Secure Coding

Comprehensive Technical Study Notes: OWASP Top 10, SQL Injection, XSS, CSRF, and Secure Coding Guidelines

---

Module 1: OWASP Top 10 Framework Overview

The Open Web Application Security Project (OWASP) Top 10 is a globally recognized awareness standard for developers and web application security experts. It represents a broad consensus on the most critical security risks facing modern web applications.

1.1 Critical Vulnerability Categories

- **A01: Broken Access Control:** Failure to enforce permissions, allowing unauthorized access to restricted data, user accounts, or administrative actions.
- **A02: Cryptographic Failures:** Inadequate protection of sensitive data in transit or at rest (e.g., plain-text secrets, weak hashing algorithms like MD5/SHA1).
- **A03: Injection (SQLi, Command, XXE):** Untrusted input concatenated directly into interpreters, altering intended query or command execution flow.
- **A04: Insecure Design:** Architectural flaws and missing security controls during software design that cannot be patched by code implementation alone.
- **A05: Security Misconfiguration:** Default settings, open cloud storage buckets, permissive CORS headers, or unpatched software stack components.

1.2 Block Diagram: Security Testing & Risk Evaluation Lifecycle





> **SHIFT-LEFT SECURITY:** Shifting security left in the Software Development Life Cycle (SDLC) reduces vulnerability remediation costs by up to 10x compared to patching post-production breaches.

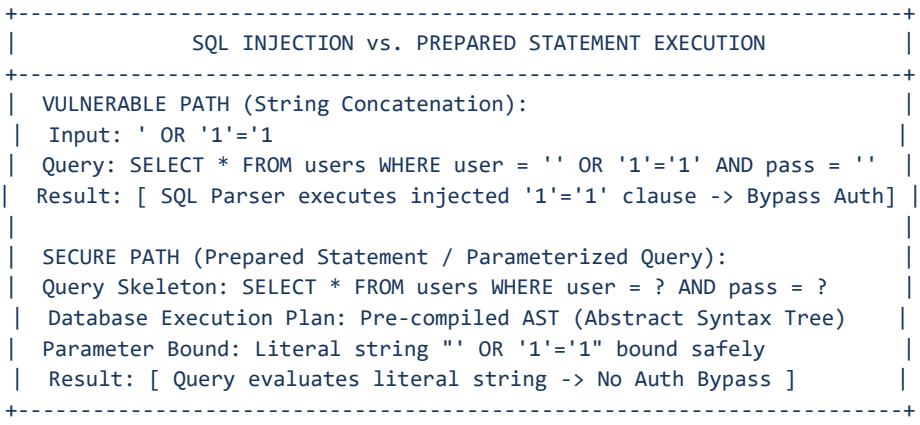
Module 2: SQL Injection (SQLi) Mechanics & Defenses

SQL Injection occurs when user-supplied input is directly concatenated into a dynamic database query string, allowing an attacker to break out of data context and manipulate query syntax.

2.1 Injection Vectors & Prevention via Parameterization

- **In-Band SQLi (Error-based / UNION-based):** The attacker extracts database schema and records directly in the web application response.
- **Inferential SQLi (Blind / Time-based):** The application does not output data directly, but attackers infer values by monitoring boolean response true/false or SQL SLEEP() delays.
- **Primary Defense (Parameterized Queries / Prepared Statements):** Ensures the database engine treats input strictly as data parameters, never as executable code.

2.2 Block Diagram: SQL Injection Attack Flow vs. Prepared Statements



Module 3: Cross-Site Scripting (XSS) Mechanics & Mitigation

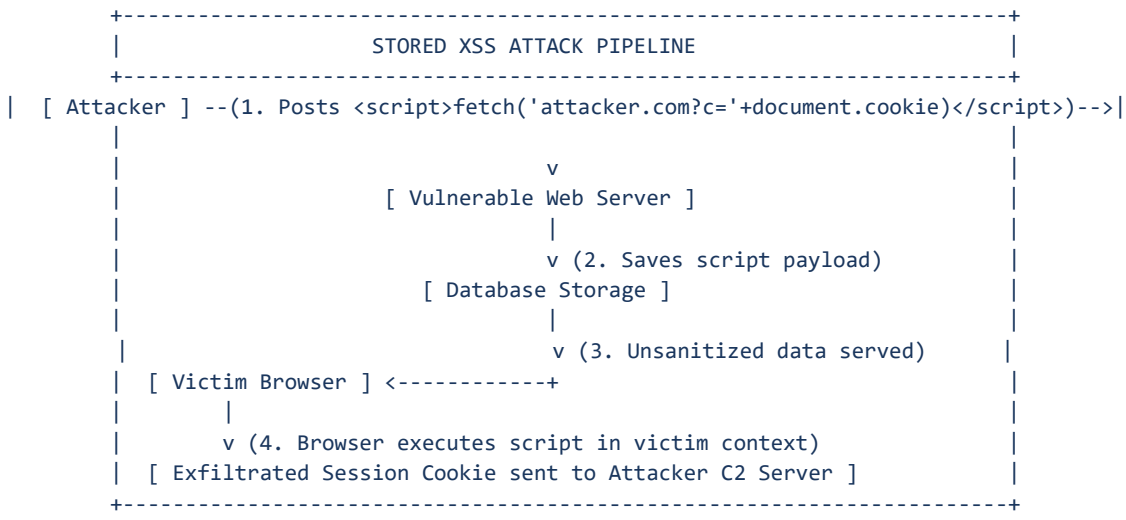
XSS vulnerabilities occur when web applications include untrusted data in web pages without proper contextual encoding or sanitization, executing malicious JavaScript in the victim's browser context.

3.1 XSS Variant Taxonomy

- **Stored XSS (Persistent):** Malicious payload is permanently saved in a database, forum, or comment field and served to every visiting user.

- **Reflected XSS (Non-Persistent):** Payload is embedded inside an HTTP request (e.g., search parameters) and immediately reflected in the server response.
- **DOM-based XSS:** Vulnerability resides entirely in client-side JavaScript execution where user input from source (e.g., location.hash) reaches an unsafe sink (e.g., innerHTML).

3.2 Block Diagram: Stored XSS Execution Pipeline



Module 4: Cross-Site Request Forgery (CSRF) & Defense

CSRF forces an authenticated user's web browser to execute unwanted actions on a trusted web application by exploiting ambient authority (automatic browser cookie transmission).

4.1 Anti-CSRF Token Mechanics & SameSite Cookie Attribute

- **Anti-CSRF Tokens (Synchronizer Token Pattern):** Cryptographically secure, unpredictable single-use tokens embedded in HTML forms/headers and validated server-side.
- **SameSite Cookie Attribute:** Strict or Lax setting preventing browsers from sending cookies along with cross-site requests.
- **Re-Authentication / OTP:** Requiring password confirmation for high-risk transactions (e.g., password change, fund transfer).

4.2 Block Diagram: CSRF Attack vs. Anti-CSRF Protection





Module 5: Secure Coding Practices Matrix

Vulnerability	Attack Mechanism	Primary Defense Control	Secondary Defense / Mitigation
SQL Injection	Dynamic SQL string concatenation	Parameterized Queries / Prepared Statements	Least Privilege Database Accounts
Cross-Site Scripting (XSS)	Injecting unescaped JS scripts	Context-Aware HTML Entity Encoding	Content Security Policy (CSP) & HttpOnly Cookies
CSRF	Exploiting automatic cookie sending	Anti-CSRF Synchronizer Tokens	SameSite=Strict/Lax Cookie Flags
Broken Access Control	Direct object reference manipulation	Server-Side RBAC / ABAC Enforcements	Disable Directory Browsing & Enforce UUIDs

> □ **SUMMARY CONCLUSION:** Secure coding requires defense-in-depth: zero trust input validation, strict context-aware encoding, parameterized database statements, anti-CSRF tokens, and defense perimeter WAF rules.