**1. Introduction**

Web applications are integral to modern businesses, but they are also prime targets for cyberattacks. Security threats such as **SQL Injection, Cross-Site Scripting (XSS), Cross-Site Request Forgery (CSRF), Session Hijacking, and Man-in-the-Middle (MITM) attacks** pose significant risks to data integrity, user privacy, and system availability. This report explores these threats, examines real-world incidents, and discusses effective mitigation strategies.

**2. SQL Injection**

**Definition & Mechanism**

SQL Injection (SQLi) occurs when an attacker inserts malicious SQL queries into input fields, manipulating the database to extract, modify, or delete sensitive data.

**Real-World Case Study: Sony Pictures (2011)**

In 2011, Sony Pictures suffered a massive SQL injection attack that exposed personal data of over **1 million users**, including passwords stored in plaintext. The attackers exploited weak input validation in Sony’s web forms.

**Mitigation Strategies**

* **Parameterized Queries (Prepared Statements):** Ensures inputs are treated as data, not executable code.
* **Input Validation & Sanitization:** Filters out malicious SQL characters.
* **Least Privilege Principle:** Database users should have minimal necessary permissions.
* **Web Application Firewalls (WAFs):** Blocks SQLi patterns.

**3. Cross-Site Scripting (XSS)**

**Definition & Types**

XSS involves injecting malicious scripts into web pages viewed by other users. Types include:

* **Stored XSS:** Malicious script stored on the server (e.g., in a comment).
* **Reflected XSS:** Script reflected off a web request (e.g., via URL parameters).
* **DOM-based XSS:** Manipulates the Document Object Model (DOM) in the browser.

**Real-World Case Study: eBay (2014)**

In 2014, attackers exploited a stored XSS vulnerability on eBay, injecting JavaScript into product listings. This redirected users to phishing sites, stealing login credentials.

**Mitigation Strategies**

* **Output Encoding:** Converts scripts into harmless text.
* **Content Security Policy (CSP):** Restricts script execution sources.
* **Input Sanitization:** Removes or neutralizes harmful scripts.

**4. Cross-Site Request Forgery (CSRF)**

**Definition & Mechanism**

CSRF tricks users into executing unwanted actions on a web app where they’re authenticated (e.g., changing passwords or transferring funds).

**Real-World Case Study: Netflix (2006)**

Attackers sent phishing emails with hidden image tags forcing logged-in Netflix users to add DVDs to their queues without consent.

**Mitigation Strategies**

* **CSRF Tokens:** Unique tokens validate legitimate requests.
* **SameSite Cookies:** Prevents cookies from being sent in cross-site requests.
* **Double-Submit Cookie:** Requires token in both cookie and form data.

**5. Session Hijacking**

**Definition & Techniques**

Attackers steal session IDs (via **session fixation, packet sniffing, or XSS**) to impersonate users.

**Real-World Case Study: Firesheep (2010)**

The **Firesheep** tool intercepted unencrypted session cookies on public Wi-Fi, allowing attackers to hijack Facebook, Twitter, and other accounts.

**Mitigation Strategies**

* **HTTPS Encryption:** Protects session IDs in transit.
* **Secure & HttpOnly Cookies:** Prevents JavaScript access.
* **Session Timeout & Regeneration:** Limits session validity.

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

**Definition & Methods**

MITM attackers intercept and alter communications between two parties (e.g., via **Wi-Fi eavesdropping, DNS spoofing, or SSL stripping**).

**Real-World Case Study: Superfish (2015)**

Lenovo pre-installed **Superfish** adware, which injected self-signed certificates, allowing MITM attacks on HTTPS traffic.

**Mitigation Strategies**

* **HTTPS (SSL/TLS):** Encrypts data in transit.
* **Certificate Pinning:** Ensures only trusted certificates are accepted.
* **VPN Usage:** Secures connections on public networks.

**7. Conclusion**

Modern web security threats are evolving, but robust defenses like **input validation, encryption, secure coding practices, and security headers** can mitigate risks. Developers must stay informed about emerging threats and adopt a **security-first approach** in web programming.