MITIGATIONS

1. SQL Injection (SQLi) Mitigations

The Goal: Prevent the database from executing user input as a command.

- **Primary: Prepared Statements (Parameterized Queries)**
  - **The Logic:** Instead of building a query string like a puzzle, you send the query structure to the database first (`SELECT * FROM users WHERE id = ?`). You then send the data separately.
  - **Why it works:** The database is "pre-compiled" with the structure. It treats the input strictly as data (a string or number), so even if an attacker enters `' OR 1=1`, the database just looks for a user whose name is literally that text string.
- **Secondary: Input Validation**
  - Use "Allow-lists." If a field expects a "User ID," the server should reject anything that isn't a number before it even touches the database.
- **Tertiary: Principle of Least Privilege**
  - The database user account used by the web app should not have `DROP TABLE` or `GRANT` permissions. It should only have access to the specific tables it needs to function.

---

2. Cross-Site Scripting (XSS) Mitigations

The Goal: Prevent the browser from executing malicious JavaScript.

- **Primary: Context-Aware Output Encoding**
  - **The Logic:** Before displaying user-supplied data on a page, convert "special" characters into their HTML entities.
  - **Example:** `<script>` becomes `&lt;script&gt;`.
  - **Why it works:** The browser sees the encoded version and thinks, "This is just text to show the user," instead of "This is a script I need to run."
- **Secondary: Content Security Policy (CSP)**
  - **The Logic:** A browser-level instruction (sent via HTTP headers) that tells the browser exactly which sources are allowed to run scripts.
  - **Why it works:** If a hacker successfully injects a script from `evil-hacker.com`, the browser will block it because it isn't on the "Approved List" defined in the CSP.
- **Tertiary: HttpOnly Cookie Flag**
  - This flag prevents JavaScript from accessing session cookies. Even if an XSS attack succeeds, the attacker cannot steal the user's login session.

---

3. Burp Suite: The Professional "Defender" Role

In an internship, you use Burp Suite not just to attack, but to **verify** that your mitigations actually work.

- **Verification with Repeater:** After a developer says they fixed a bug, you take the original "attack" request in **Repeater**, send it again, and observe the **Response**.
  - *Success:* The response should show the payload as plain text (encoded) rather than executing it.
- **Security Header Analysis:** Use Burp to inspect the **Headers** of the server's response. Check for:
  - `Content-Security-Policy`: Is it restrictive enough?
  - `X-Frame-Options`: Does it prevent Clickjacking?
  - `Strict-Transport-Security (HSTS)`: Does it force HTTPS?
- **Automated Scanning (Professional Tip):** Mention that in a corporate environment, **Burp Suite Professional** is used to run automated vulnerability scans periodically to catch "low-hanging fruit" before a human tester even starts.