**Name: Kabeer Ahmad**

Below Are The Steps I did for Securing a Web Application:

1. **Content Security Policy (CSP):**

Purpose: Prevent Cross-Site Scripting (XSS) by specifying which sources of content (scripts, styles, images) are allowed to be loaded.

Content-Security-Policy: default-src 'self'; script-src 'self' https://trusted-source.com; style-src 'self' <https://trusted-source.com>;

1. **HTTP Strict Transport Security (HSTS):**

**Purpose**: Forces browsers to interact with your server over HTTPS only, preventing man-in-the-middle (MITM) attacks.

Strict-Transport-Security: max-age=31536000; includeSubDomains; preload

1. **X-Frame-Options:**

**Purpose**: Prevent clickjacking by blocking your site from being framed in other sites.

X-Frame-Options: DENY

1. **X-XSS-Protection:**

**Purpose**: Enable Cross-Site Scripting (XSS) protection in browsers.

X-XSS-Protection: 1; mode=block

1. **SQL Injection:**

* **Issue**: SQL injection occurs when attackers insert malicious SQL queries into input fields, gaining unauthorized access to your database.
* **How to Prevent**:
  + **Parameterized Queries/Prepared Statements**: Always use parameterized queries or prepared statements when querying the database. Example in Node.js :

const query = 'SELECT \* FROM users WHERE id = $1';

const values = [userId];

client.query(query, values, (err, res) => {

// Handle response

});

1. **Cross-Site Scripting (XSS):**

* **Issue**: XSS occurs when an attacker injects malicious scripts into a website viewed by other users, enabling them to hijack sessions, steal cookies, or perform malicious actions.
* **How to Prevent**:
  + **Sanitize User Input**: Use libraries to sanitize input ( DOMPurify in JavaScript). It cleans input by removing any script or event handlers.

const sanitizedInput = DOMPurify.sanitize(userInput);

1. **Regularly Update and Patch:**

* Keep all dependencies, frameworks, and libraries up to date.
* Monitor for security patches and apply them as soon as possible.
* Automate vulnerability scanning of your dependencies with tools like **npm audit** (for Node.js projects) or **Dependabot** in GitHub.

1. **Implement Strong Authentication and Authorization:**

**a) Password Policy:**

* Enforce a strong password policy with a minimum length and complexity (e.g., 12 characters, mixing upper/lower case letters, numbers, and special characters).
* Store passwords using **strong hashing algorithms** like bcrypt or Argon2.

const bcrypt = require('bcrypt');

const saltRounds = 10;

const hash = await bcrypt.hash(password, saltRounds);

**b) Two-Factor Authentication (2FA):**

* Implement 2FA for sensitive actions, using services like Google Authenticator or SMS-based OTPs.

1. **Logging and Monitoring:**

* Set up logging for all critical actions, especially around authentication, authorization, and database interactions.
* Use tools like **ELK Stack (Elasticsearch, Logstash, Kibana)** or cloud-based logging services to monitor logs for suspicious activity.
* Set up alerting for any abnormal patterns, such as multiple failed login attempts.