SECURITY ASSESSMENT REPORT

BY: AHMED OWAIS

🛡 Security Assessment Report 🛡

Student Name: [Ahmed Owais]
University: [Air University Islamabad]
Date: [18/4/2025]

Step 1: Understanding the Application

Repository Used:

I developed a simple Node.js web application with User Management(Logout/Login) features from GitHub. The repository link is **https://github.com/ahmedowais520/Web-Project.git**

Setup Process:

1. Cloned the repository:
    - " git clone https://github.com/ahmedowais520/Web-Project.git "
2. Selected the correct Directory:
    - " cd Web-Project "
3. Installed dependencies:
    - " npm install "
4. Started the application:
    - " npm start "
5. Accessed the app at:
    - `http://localhost:3000`

Features Explored:

- Signup
- Login
- Profile management

I created dummy accounts to explore the user flows and confirmed all features were functioning as expected.

Step 2: Vulnerability Assessment

Tool 1: OWASP ZAP

- **Scan Performed:** Automated Scan targeting `http://localhost:3000`
- **Findings:**
  - **XSS (Cross-Site Scripting):** Reflected XSS found in the signup form (name field)
  - **Insecure Cookies:** Session cookies were missing security flags
  - **Content Security Policy (CSP):** CSP header was missing
  - **Directory Browsing:** Enabled on public directories

Tool 2: Browser Developer Tools (Manual XSS Test)

- **Test:** Entered `<script>alert('XSS');</script>` in the **signup form (name field)**.
- **Result:** A popup appeared confirming XSS vulnerability in the form.

Tool 3: SQL Injection Test

- **Test:** I entered [`admin' OR '1'='1`] in both username and password fields on the login page but the login was safe it didn't worked.
- **Test:** Again I entered [SELECT * FROM users WHERE username = '' OR 1=1-- ' AND password = 'foo']
- **Result:** Login was successful without valid credentials, indicating an " SQL Injection vulnerability ".

Step 3: Improvement Suggestions

---

1. **Input Validation:**
    - Sanitize and validate all user inputs to prevent XSS and SQL Injection.
2. **Use of Prepared Statements:**
    - Implement prepared statements or ORM methods to prevent SQL Injection.
3. **Password Security:**
    - Hash passwords using bcrypt or another strong hashing algorithm.
4. **Secure Headers:**
    - Add HTTP security headers, such as:
        - Content Security Policy (CSP)
        - X-Content-Type-Options
        - X-Frame-Options
5. **Cookie Security:**
    - Set `HttpOnly` and `Secure` flags for session cookies.
6. **Disable Directory Browsing:**
    - Turn off directory listing in the web server configuration.

Conclusion:

I have successfully completed the security assessment for the selected Node.js application. The app is functional but has critical vulnerabilities that need immediate attention. The recommended improvements should be prioritized to enhance the application's security posture.