Apex Planet Cybersecurity Internship:

Task 3 - Web Application Security

This repository contains the deliverables for Task 3 of the Apex Planet Cybersecurity & Ethical Hacking Internship, focusing on Web Application Security.

The project involved a security assessment of a vulnerable web application, Damn Vulnerable Web App (DVWA), to identify and exploit common vulnerabilities.

The vulnerabilities explored include:

- SQL Injection (SQLI)
- Cross-Site Scripting (XSS)
- Cross-Site Request Forgery (CSRF)

This repository serves as a detailed technical supplement to the official project report, which includes visual evidence and screenshots of the vulnerabilities discovered.

Table of Contents:

Methodology & Tools:

The following tools and methodologies were used in this project:

- **Kali Linux:** As the attacker machine to perform the tests.
- **Damn Vulnerable Web App (DVWA):** As the target application for the assessment.
- **Manual Testing:** Directly interacting with the web application's forms and URLs.
- **Burp Suite:** Used for intercepting and modifying requests.

Project Deliverables:

This repository contains the following deliverables as part of the Web Application Security task:

- **Attack Scenarios:** Detailed, step-by-step documentation of how each vulnerability was exploited.

- **Mitigation Notes:** Comprehensive guidance on how to fix and prevent each vulnerability.
- **Supporting Scripts/Files:** Any custom files used for the exploitation, such as the HTML file for the CSRF attack.

2. SQLI_Attack_Scenario.md

This file details the SQL Injection attack scenario and its mitigation.

SQL Injection (SQLI) - Attack Scenario & Mitigation:

Attack Scenario:

- **Vulnerable Component:**
  - The login form on the DVWA web application.
- **Vulnerability:**
  - The application is vulnerable to SQL Injection due to the lack of proper input sanitization on its login form.
- **Payload Used:**
  - ' OR 1=1--
- **Steps to Reproduce:**
  1. Navigate to the DVWA login page.
  2. In the "Username" field, enter ' or 1=1--.
  3. In the "Password" field, enter any random password.
  4. Click the "Login" button.
- **Expected Outcome:**
  - The malicious query manipulates the database query, allowing a login bypass. The user is granted access without needing a valid username and password, as demonstrated by the login success message.

Mitigation Notes

- **Principle:**
  - Never directly use user input in a database query.
- **Solution:**
  - The primary defence against SQL Injection is using **Prepared Statements** with parameterized queries. This method ensures that user input is treated as a literal value and not as executable SQL code.
- **Further Recommendations:**

    - Implement a Web Application Firewall (WAF) to filter malicious requests.
    - Follow the principle of least privilege for the database user, ensuring it only has the permissions required for the application's functions.

3. XSS_Attack_Scenario.md

This file details the Cross-Site Scripting attack and its mitigation.

Cross-Site Scripting (XSS) - Attack Scenario & Mitigation

Attack Scenario (Reflected XSS)

- **Vulnerable Component:**
    - The search bar or input field on a DVWA page.
- **Vulnerability:**
    - The application reflects user input back to the page without proper encoding, making it vulnerable to XSS.
- **Payload Used:**
    - <script>alert('XSS AttackSuccessful')</script>
- **Steps to Reproduce:**
    1. Navigate to the Reflected XSS page in DVWA.
    2. In the input field, type the payload <script>alert('XSS AttackSuccessful')</script>.
    3. Click "Submit."
- **Expected Outcome:**
    - A JavaScript alert box pops up with the message "XSS AttackSuccessful," confirming that the injected script was executed in the browser.

Mitigation Notes:

- **Principle:**
    - All user input must be considered malicious and should be properly handled before being rendered on a webpage.
- **Solution:**
    - Implement **Output Encoding** to convert special characters (<, >, " etc.) into HTML entities. This prevents the browser from interpreting the input as code.
- **Further Recommendations:**

    - Implement a **Content Security Policy (CSP)** header to restrict which domains are allowed to load scripts, styles, or other resources.
    - Use input validation to reject input that contains unexpected characters or patterns.

4. CSRF_Attack_Scenario.md

This file details the Cross-Site Request Forgery attack and its mitigation.

Cross-Site Request Forgery (CSRF) - Attack Scenario & Mitigation

Attack Scenario:

- **Vulnerable Component:**
    - A state-changing action, such as the "Change your admin password" function in DVWA.
- **Vulnerability:**
    - The web application does not verify if the request originated from the user's browser, allowing an attacker to forge a request and trick the user's browser into executing it.
- **Methodology:**
    - An attacker crafts a malicious HTML page with a hidden form and places it on a separate website. When an authenticated user visits the attacker's page, the hidden form automatically submits a request to the vulnerable application.
- **Steps to Reproduce:**
    1. An authenticated user is logged into the DVWA application.
    2. The user navigates to a malicious website created by the attacker.
    3. The malicious page contains an invisible <img> tag or form that points to the DVWA password change endpoint with new, attacker-defined password values.
    4. When the page loads, the browser automatically sends this request, and the password on the DVWA site is changed without the user's knowledge.

Mitigation Notes:

- **Principle:**
    - All state-changing requests must be verified to ensure they are initiated by a legitimate user with explicit intent.
- **Solution:**
    - Use **CSRF Tokens**. A unique, unpredictable token is generated by the server and embedded in a hidden field of the form. When the form is

submitted, the server verifies that the token matches the one in the user's session.

- **Further Recommendations:**
  - Use the Same Site attribute on cookies to prevent them from being sent with cross-site requests.
  - Implement a CAPTCHA or re-authentication for sensitive actions like changing passwords.