**Web Application Security Lab Report**

**Overview**

This lab explored practical aspects of ethical hacking and web application defense techniques. The objective was to simulate attacker behavior during reconnaissance and exploitation, and to implement secure coding practices for mitigating vulnerabilities. The tools used are common in both offensive security and secure development workflows.

**Tools and Technologies Used**

* Kali Linux (penetration testing OS)
* DVWA (Damn Vulnerable Web Application)
* Apache, MySQL/MariaDB, PHP
* Burp Suite (Community Edition)
* SQLMap
* Node.js with Express
* Middleware: csurf, cookie-parser, body-parser

**Task 1: Ethical Hacking Basics**

**Configuration and Setup**

DVWA was set up in a local environment. The Apache web server and MySQL were configured to serve DVWA, and the application was made accessible via http://localhost/dvwa/. Configuration files were modified to ensure proper database connection using default credentials for DVWA.

**Reconnaissance Activities**

**Nmap** and **Nikto** were used to identify services running on the test server and assess basic misconfigurations.

* **Nmap** identified open ports (e.g., 80 for HTTP, 3306 for MySQL) and provided version information for the services.
* **Nikto** returned findings related to outdated Apache modules and potential vulnerabilities in the web server setup, such as missing security headers and open directories.

**Findings (from tool output perspective):**

* Exposed services running default configurations.
* The web application lacked protections like X-Frame-Options or X-Content-Type-Options.
* Webroot accessible with no authentication controls, allowing for URL enumeration.

**Task 2: SQL Injection & Exploitation**

**Identifying SQL Injection**

A SQL injection point was found on the DVWA SQLi module under a user ID input field. Testing with payloads such as ' OR '1'='1 indicated that the application directly concatenated user input into SQL queries without validation.

**Capturing the Request**

Using Burp Suite, the vulnerable HTTP request was intercepted, showing a classic GET parameter vulnerable to SQL injection (id=1). This was saved and analyzed with SQLMap.

**Automated Exploitation**

Using **SQLMap**, the tool automatically:

* Confirmed the injection point.
* Enumerated available databases.
* Extracted table and user data from the dvwa database.

**Findings (from SQLMap perspective):**

* DVWA’s SQLi module was vulnerable to time-based and boolean-based blind SQL injection.
* SQLMap successfully extracted contents of the users table, including usernames and password hashes.

**Secure Coding Countermeasure**

The injection-prone code was mitigated using prepared statements with parameterized queries. This effectively neutralized the attack vector by separating data from the query structure.

**Example (Node.js):**

*const stmt = db.prepare("SELECT \* FROM users WHERE id = ?");*

*stmt.bind\_param("i", id);*

*stmt.execute();*

**Task 3: CSRF Protection and Testing**

**Implementation**

A basic Express application was created to demonstrate CSRF protection. The app used csurf middleware to embed a unique token in form requests and validate it on submission.

**CSRF Token Workflow**

1. When accessing /form, the server provided a unique CSRF token in the form.
2. Submitting the form via a browser sent the token back, which was validated server-side.
3. Any tampering with or removal of the token caused the request to fail validation.

**Testing with Burp Suite**

Requests were captured using Burp Intercept and sent to Repeater:

* When the token was removed from the POST body, the server returned a **403 Forbidden** response.
* This confirmed that CSRF protection was active and functioning as expected.

**Findings (from CSRF test perspective):**

* The application properly issued and validated CSRF tokens.
* Tampered requests without valid tokens were blocked.
* The approach successfully mitigated simulated CSRF attacks.

**Conclusion**

This lab effectively covered three major domains of web application security:

1. **Reconnaissance** showed how easily information can be gathered from misconfigured services.
2. **SQL Injection** was demonstrated both manually and through automation, revealing how critical secure coding practices like prepared statements are.
3. **CSRF Protection** illustrated the effectiveness of middleware-based safeguards against session abuse and unauthorized actions.

Each phase was carried out using common tools in a controlled environment, providing a foundational understanding of both offensive and defensive web security techniques.