Security Testing Report – DVWA

Intern Name: Tobechukwu Nicholas

Date: May 16, 2025

Application Tested: DVWA (Damn Vulnerable Web Application)

Testing Environment: Kali Linux (Docker DVWA Setup on localhost:8080)

Security Level: Low

# Objectives

- Conduct security testing on a sample web application (DVWA)

- Identify vulnerabilities like SQL Injection, XSS, and Authentication flaws

- Use penetration testing tools and manual testing techniques

- Document findings and recommend mitigation strategies

# Vulnerability Testing Results

## 1. SQL Injection

**Location**: http://localhost:8080/vulnerabilities/sqli/

**Test Method**: Manual input and payload injection in the User ID field

|  |  |
| --- | --- |
| **Payload Used** | **Result** |
| 1' OR '1'='1 | Application returned all users from the database, proving it's vulnerable to SQL injection. |
| 1' UNION SELECT null, version() # | Application revealed DB version: 10.5.18-MariaDB |
| 1' UNION SELECT null, database() # | Application revealed current database name: dvwa |
| 1' UNION SELECT null, user() # | Application revealed DB user: root@localhost |

**Impact**: Allows attackers to bypass authentication, extract data, and potentially access the underlying system.

**Mitigation**: Use prepared statements with bound parameters (e.g., PDO or mysqli in PHP). Avoid direct interpolation of user inputs in SQL queries.

## 2. Reflected XSS (Cross-Site Scripting)

**Location**: http://localhost:8080/vulnerabilities/xss\_r/

**Test Method**: Injected JavaScript in input field

|  |  |
| --- | --- |
| **Payload Used** | **Result** |
| <script>alert('XSS')</script> | Browser displayed JavaScript alert confirming XSS execution |
| <img src=x onerror=alert(1)> | Confirmed XSS via image-based payload |

**Impact**: An attacker could execute arbitrary scripts in the victim’s browser, steal session cookies, perform phishing, or alter site content.

**Mitigation**: Sanitize user inputs and encode outputs using libraries like OWASP ESAPI or built-in frameworks’ escaping methods.

## 3. Authentication Bypass via SQL Injection

**Location**: http://localhost:8080/login.php

**Test Method**: Logged in with a bypass payload

|  |  |
| --- | --- |
| **Credentials** | **Result** |
| **Username**: admin **Password**: ' OR '1'='1 | Logged in successfully without valid credentials |

**Impact**: Full account takeover by bypassing authentication controls.

**Mitigation**: Use secure authentication with parameterized SQL queries. Do not rely on string concatenation for login validation.

# Tools Used

- Manual Testing (Browser Input Fields)

- Kali Linux

- DVWA via Docker

# Summary of Findings

|  |  |  |  |
| --- | --- | --- | --- |
| **Vulnerability** | **Affected Page** | **Severity** | **Status** |
| SQL Injection | /sqli/ | High | Confirmed |
| Reflected XSS | /xss\_r/ | High | Confirmed |
| Auth Bypass (SQLi) | /login.php | High | Confirmed |

# Recommendations

1. Implement parameterized queries for all DB interactions.

2. Sanitize and encode all user input and output.

3. Use modern authentication libraries with hashing and salting.

4. Perform regular code reviews and penetration testing.

5. Set a stricter Content Security Policy (CSP) to reduce script injection risk.

# Conclusion

The DVWA application demonstrated multiple critical vulnerabilities, including SQL injection, XSS, and authentication flaws. These issues can be exploited to gain unauthorized access, extract sensitive data, and compromise the system. Developers should follow secure coding practices and regularly audit applications to mitigate these risks.