

# Web Application Security Testing Report

Target: Damn Vulnerable Web Application (DVWA)

Tester: Ponnam Manaswini

Environment: Kali Linux (Apache2, MySQL), Firefox Browser

Security Level: Low (confirmed via DVWA interface)

## Executive Summary:

This report documents the exploitation of four critical web vulnerabilities — Reflected Cross-Site Scripting (XSS), Stored Cross-Site Scripting (XSS), Cross-Site Request Forgery (CSRF), and SQL Injection (SQLi) — within DVWA. All tests were conducted in a controlled lab environment with DVWA .

## Vulnerability Findings:

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

- Module: XSS (Reflected)

- Payload Used: <script>alert(1)</script>

- Steps:

1. Navigated to XSS (Reflected) module

2. Entered payload in input field

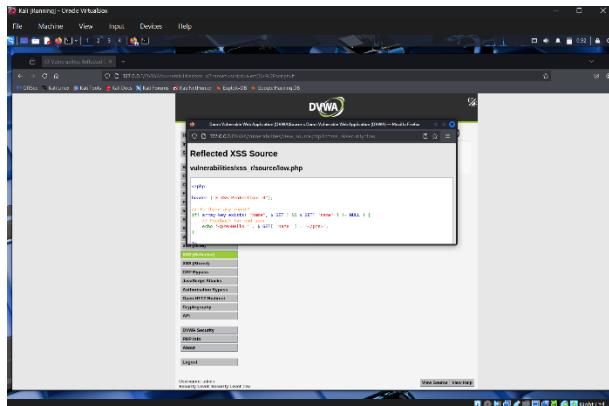
3. Observed alert popup in response

- Impact: Immediate script execution; phishing and session theft risk

- Remediation:

- Validate and encode input/output

- Avoid rendering user input directly in HTML



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

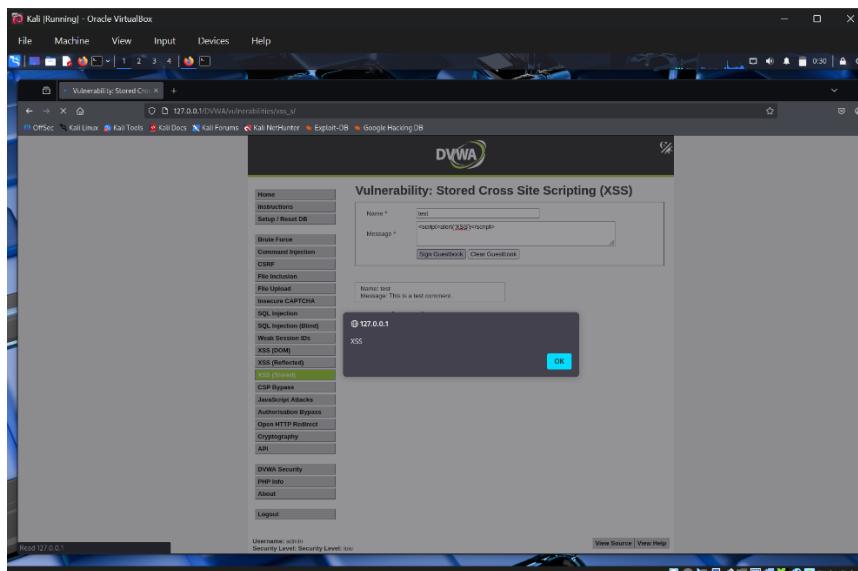
- Module: XSS (Stored)
- Payload Used: <script>alert('XSS')</script>
- Steps:

1. Navigated to XSS (Stored) module
2. Entered payload in the Name field and a message in the Message field
3. Submitted the form and observed alert popup upon page reload

- Impact: Malicious scripts are stored and executed whenever the vulnerable page is viewed, affecting all users

### - Remediation:

- Sanitize and encode user input before storing
- Encode output when rendering stored data
- Implement Content Security Policy (CSP) to restrict script execution



### **3.Cross-Site Request Forgery (CSRF)**

- Module: CSRF
  - Action Taken: Changed password manually via vulnerable form

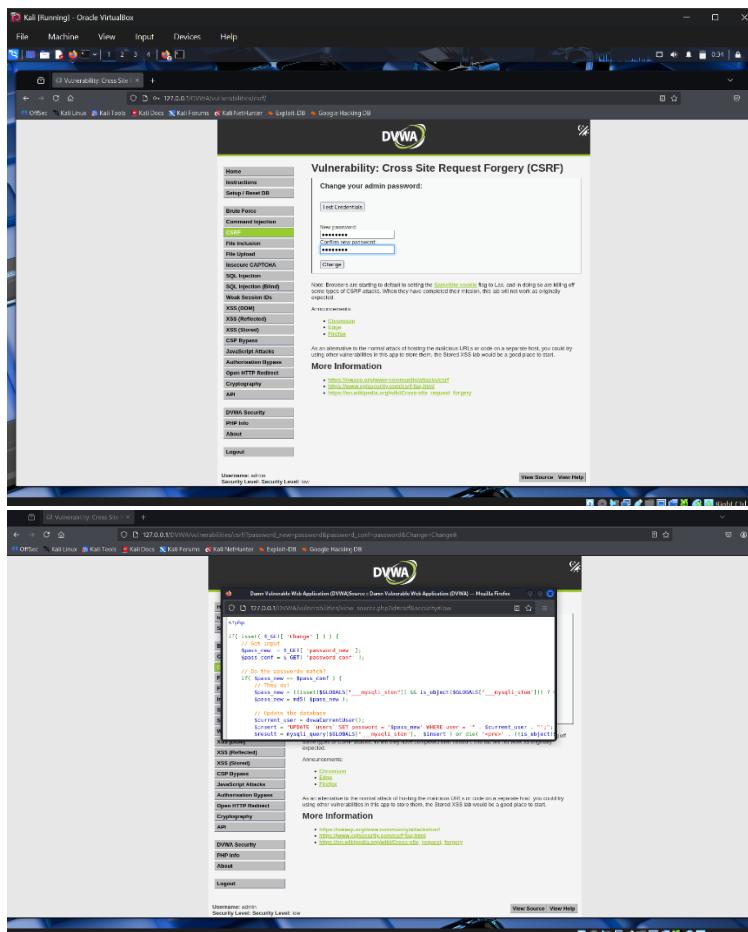
#### - Steps:

1. Navigated to CSRF module
  2. Entered new password and confirmed
  3. Submitted form and observed "Password Changed" message

- Impact: No CSRF token validation; attacker could replicate this via hidden form

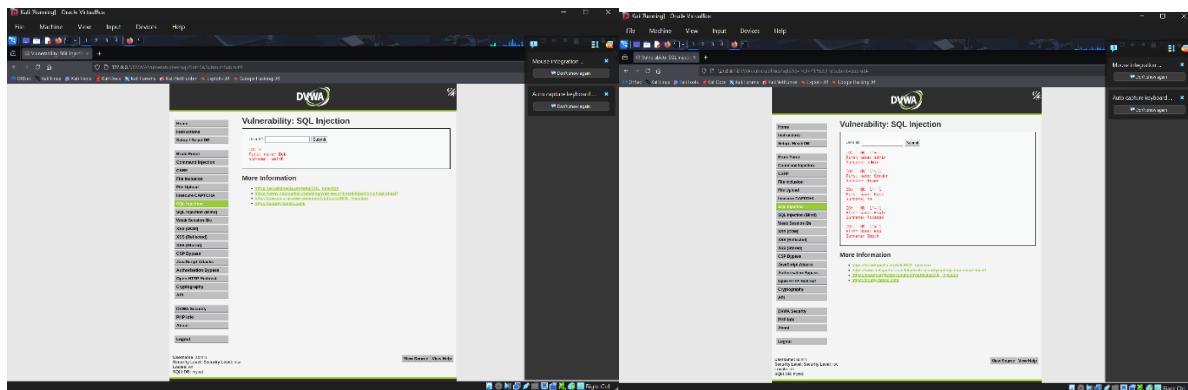
#### - Remediation:

- Implement anti-CSRF tokens
  - Use SameSite cookie attributes
  - Confirm user intent via re-authentication



## 4.SQL Injection (SQLi)

- Module: SQL Injection
- Screenshot Reference: SQL query output and vulnerable code snippet
- Payload Used: ' OR '1'='1
- Steps:
  1. Navigated to SQL Injection module
  2. Entered payload in User ID field
  3. Observed query manipulation and user data leakage
- Impact: Authentication bypass and database exposure
- Remediation:
  - Use parameterized queries (prepared statements)
  - Validate and sanitize input
  - Restrict error messages and query output



SQL Injection Source

vulnerabilities/sqli/source/low.php

```
<?php  
if( $_POST[ 'Submit' ] ) {  
    // Get input  
    $id = $_POST[ 'id' ];  
  
    switch( $DVWA[ 'SQLITE_DB' ] ) {  
        case MYSQL:  
            // MySQL - Standard database  
            $query = "SELECT first_name, last_name FROM users WHERE user_id = '$id';";  
            $result = mysqli_query($GLOBALS["__mysqli_ston"], $query) or die( __($result) . ((is_object($GLOBALS["__mysqli_ston"])) ? mysqli_error($GLOBALS["__mysqli_ston"]) : ($__mysqli_res->error)));  
  
            // Get results  
            while($row = mysqli_fetch_assoc($result)) {  
                $first = $row['first_name'];  
                $last = $row['last_name'];  
  
                // Feedback for end user  
                echo "<p>ID: {$id}<br />First name: {$first}<br />Surname: {$last}</p>";  
            }  
            mysqli_close($GLOBALS["__mysqli_ston"]);  
            break;  
        case SQLITE:  
            global $sqlite_db_connection;  
  
            $sqlite_db_connection = new SQLite3($DVWA[ 'SQLITE_DB' ]);  
            $sqlite_db_connection->enableExceptions(true);  
  
            $query = "SELECT first_name, last_name FROM users WHERE user_id = '$id';";  
            $print_query;  
            try {  
                $results = $sqlite_db_connection->query($query);  
            } catch (Exception $e) {  
                echo 'Caught exception: ' . $e->getMessage();  
            }  
  
            if ($results) {  
                while($row = $results->fetchArray()) {  
                    // Get values  
                    $first = $row['first_name'];  
                    $last = $row['last_name'];  
  
                    // Feedback for end user  
                    echo "<p>ID: {$id}<br />First name: {$first}<br />Surname: {$last}</p>";  
                }  
            }  
    }  
}
```

## **Conclusion:**

The DVWA application, when configured to low security, exposes critical vulnerabilities that are easily exploitable. This lab exercise validates the importance of secure development practices and highlights how attackers can manipulate input fields, session states, and backend queries.

## **Recommendations:**

- Apply input validation and output encoding across all user inputs
- Implement CSRF tokens and secure cookie attributes
- Use prepared statements for database queries
- Sanitize stored data and enforce Content Security Policy
- Conduct regular security assessments and code reviews