🛡️ SECURITY ASSESSMENT REPORT

Client: Internal Security Lab / Practice Project  
Application Tested: Damn Vulnerable Web Application (DVWA)

─────────────────────────────

1. Executive Summary

This report provides a detailed security assessment of the DVWA platform, focusing on vulnerabilities aligned with the OWASP Top 10. The engagement aimed to simulate real-world attacker behaviour and document findings with screenshots, technical impact, and remediation strategies.

─────────────────────────────

1. Scope of Engagement

Target:

* URL: <http://localhost/DVWA>
* Platform: PHP + MySQL
* Authentication: admin / password
* Tools Used: OWASP ZAP, Burp Suite, Firefox, Kali Linux

In-Scope Tests:

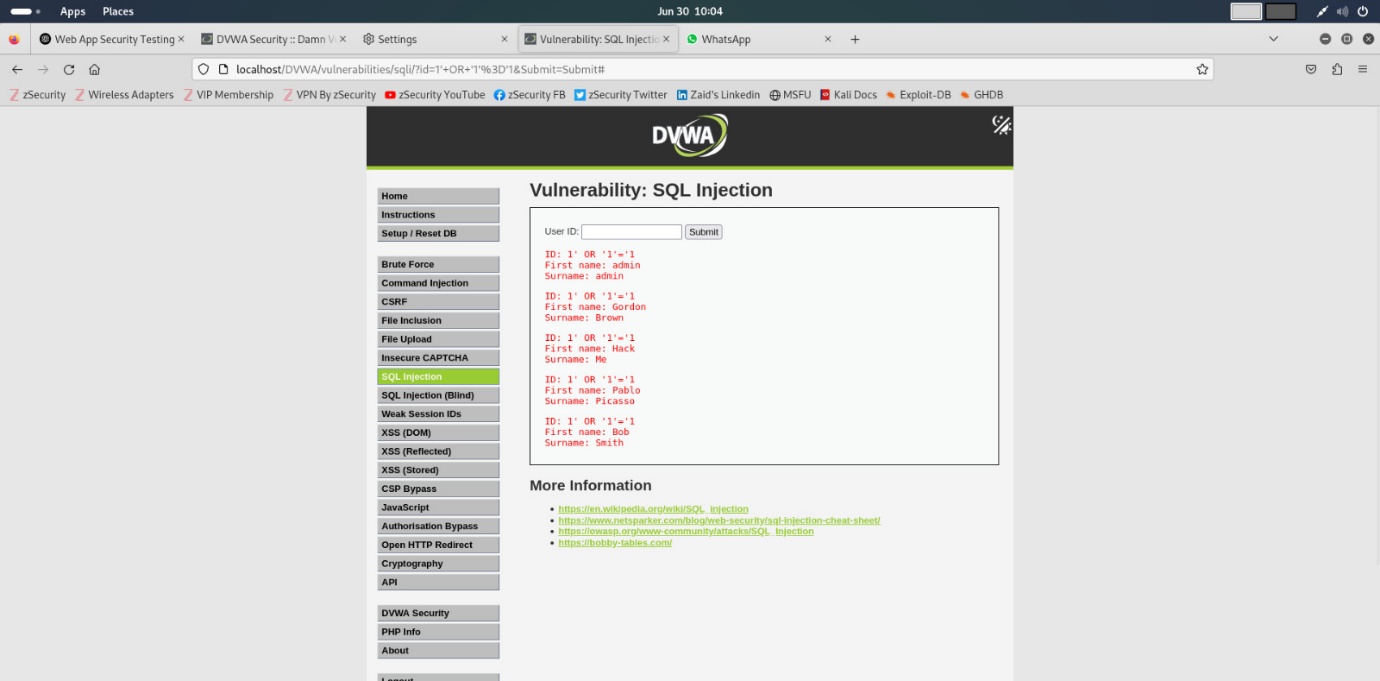
* SQL Injection
* Cross-Site Scripting (XSS)
* Cross-Site Request Forgery (CSRF)

─────────────────────────────

1. Vulnerability Findings

✅ 3.1 SQL Injection

* Affected Page: /vulnerabilities/sqli/
* Method: GET
* Input: id parameter
* Risk: High



Description:  
SQL queries can be manipulated through user input without proper validation. An attacker can retrieve or modify database contents.

Proof of Concept (PoC):  
<http://localhost/DVWA/vulnerabilities/sqli/?id=1>' OR '1'='1&Submit=Submit

Impact:  
Database leakage, unauthorized access

Mitigation:

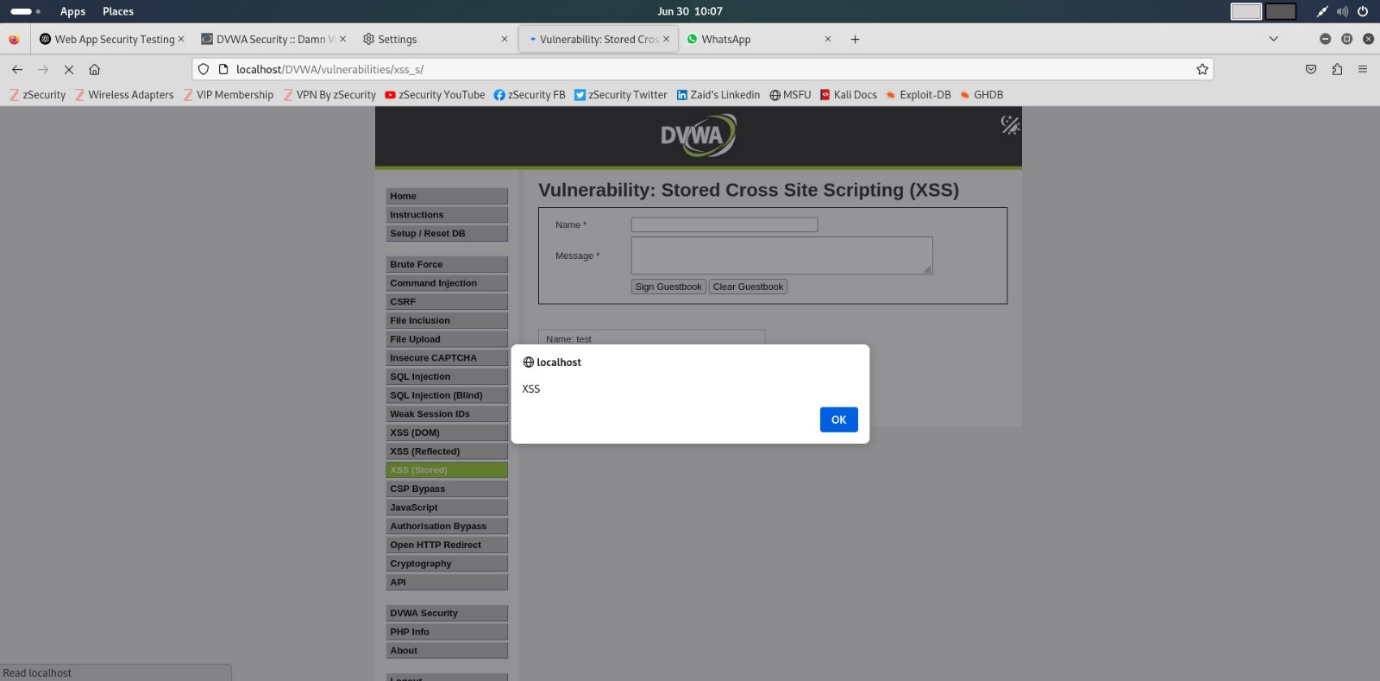
* Use parameterized queries (PDO, prepared statements)
* Server-side input validation

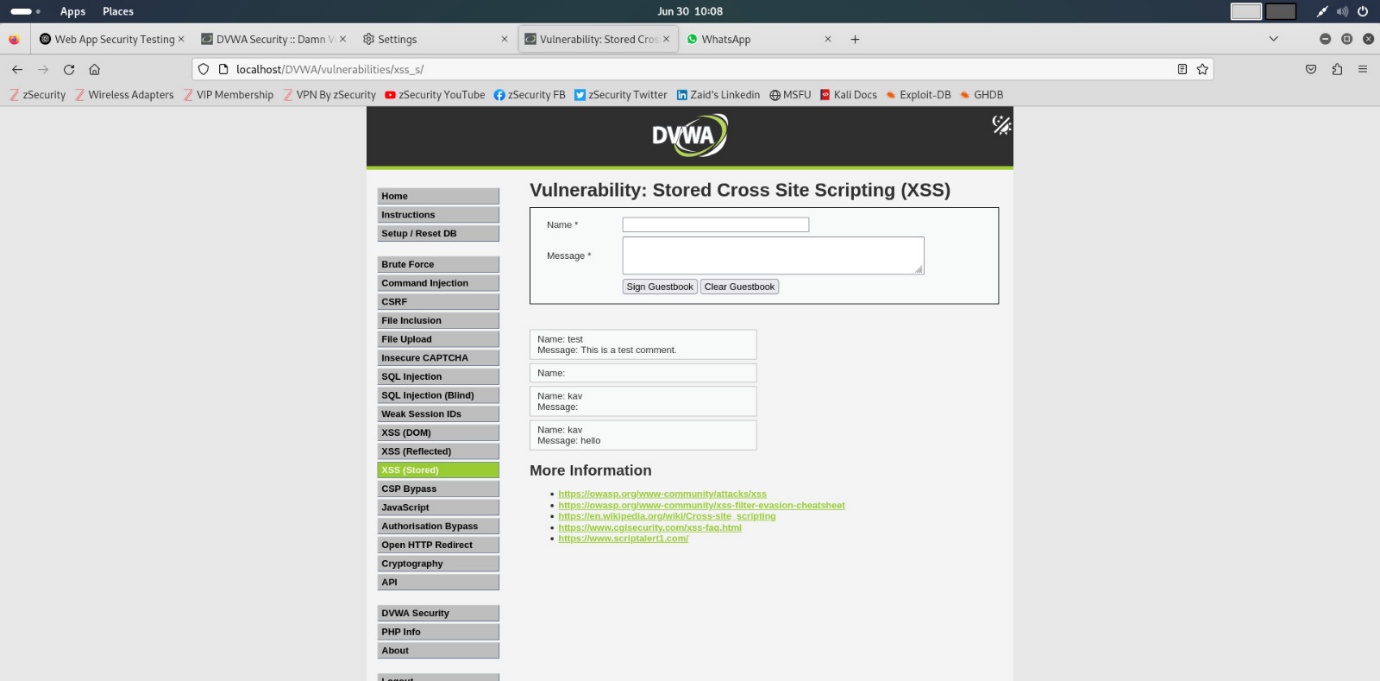
OWASP Mapping: A1 – Injection

─────────────────────────────

✅ 3.2 Stored Cross-Site Scripting (XSS)

* Affected Page: /vulnerabilities/xss\_s/
* Method: POST
* Input: message field
* Risk: Medium





Description:  
Malicious script is stored in the backend database and executed in the browser of users viewing the data.

PoC:

<script>alert('XSS')</script>

Impact:  
Session hijacking, redirecting users to malicious sites

Mitigation:

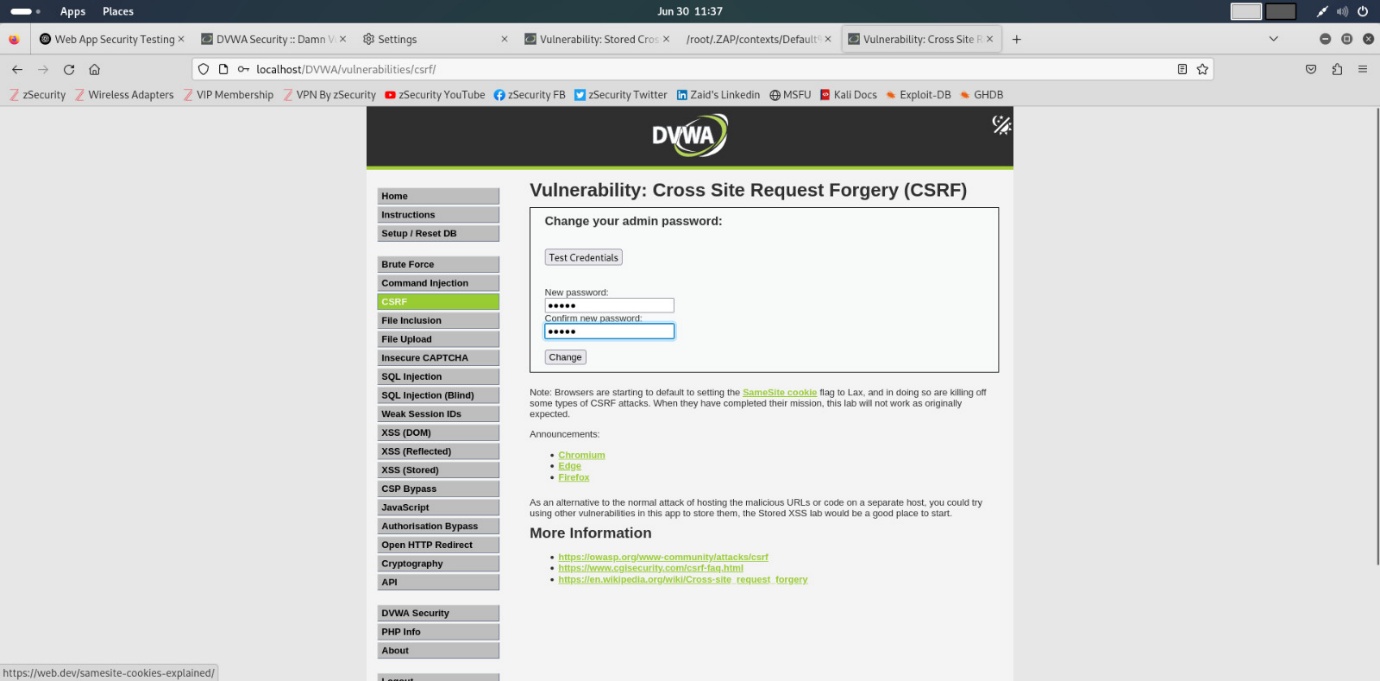
* Encode user input before displaying
* Use CSP headers
* Apply input validation and sanitization

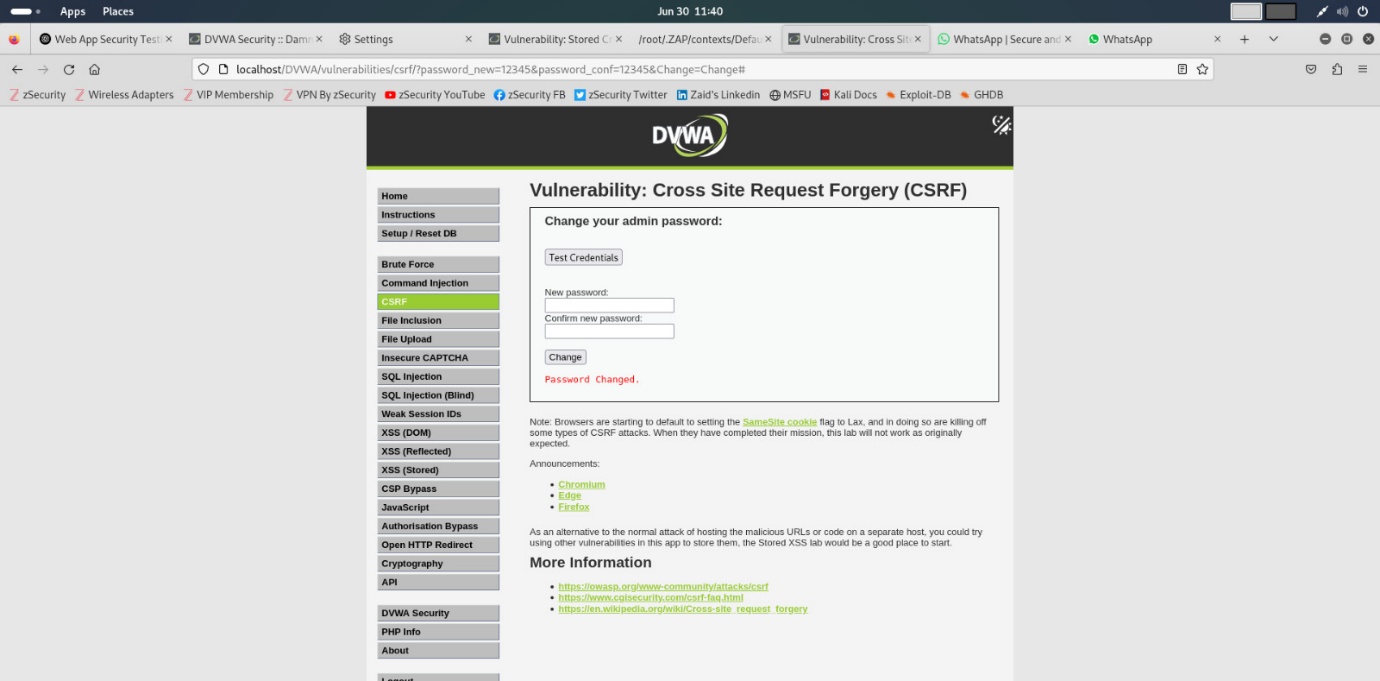
OWASP Mapping: A7 – Cross-Site Scripting (XSS)

─────────────────────────────

✅ 3.3 Cross-Site Request Forgery (CSRF)

* Affected Page: /vulnerabilities/csrf/
* Method: POST
* Risk: Medium





Description:  
CSRF tricks a logged-in user to unknowingly perform an action, like changing a password.

PoC: HTML form auto-submitting change password

Impact:  
Account takeover, user manipulation

Mitigation:

* Use anti-CSRF tokens (per form/session)
* Validate the origin header
* Require re-authentication for sensitive actions

OWASP Mapping: A5 – Broken Access Control (CSRF)

─────────────────────────────

1. OWASP Top 10 Mapping

| **OWASP Category** | **Vulnerability Found** |
| --- | --- |
| A1 – Injection | ✅ SQLi |
| A2 – Broken Auth | ⛔ Not Tested |
| A3 – Sensitive Data Exp | ⛔ Not Tested |
| A4 – XML External Entities | ⛔ Not Tested |
| A5 – Broken Access Control | ✅ CSRF |
| A6 – Security Misconfig | ⛔ Not Tested |
| A7 – XSS | ✅ Stored XSS |
| A8 – Insecure Deser | ⛔ Not Tested |
| A9 – Using Components w/ Vuln | ⛔ Not Tested |
| A10 – Logging/Monitoring | ⛔ Not Tested |

─────────────────────────────

5. Vulnerability Findings (with Risk Ratings & Recommendations)

✅ 5.1 SQL Injection

* Affected Module: /vulnerabilities/sqli/
* Input Field: id (GET parameter)
* Type: Authentication Bypass / Data Extraction
* Payload Used: ' OR '1'='1
* Risk Rating: CRITICAL (CVSS: 9.8)
* OWASP Category: A1 – Injection

🛑 Impact:

* Unauthorized access to sensitive data
* Full database extraction
* Bypassing authentication
* Possible remote code execution in some DBMSs

✅ Mitigation:

* Use parameterized queries (e.g., PDO, PreparedStatement)
* Implement strict server-side input validation
* Use ORM frameworks that prevent query manipulation
* Disable verbose DB error messages

—

✅ 5.2 Stored Cross-Site Scripting (XSS)

* Affected Module: /vulnerabilities/xss\_s/
* Input Field: message (POST parameter)
* Type: Stored XSS (Persistent)
* Payload Used: <script>alert("XSS")</script>
* Risk Rating: HIGH (CVSS: 7.4)
* OWASP Category: A7 – Cross-Site Scripting

🛑 Impact:

* Execution of arbitrary JavaScript in victim's browser
* Session hijacking via document.cookie
* Redirection to malicious sites
* Keylogging or phishing

✅ Mitigation:

* Encode output using HTML entities (<, >, etc.)
* Sanitize user input (e.g., DOMPurify, Python bleach)
* Implement a strict Content Security Policy (CSP)
* Validate input both client and server side

—

✅ 5.3 Cross-Site Request Forgery (CSRF)

* Affected Module: /vulnerabilities/csrf/
* Input: Change password request (POST)
* Risk Rating: MEDIUM (CVSS: 6.5)
* OWASP Category: A5 – Broken Access Control

🛑 Impact:

* Forced state-changing actions on authenticated users
* Unintentional password change
* Loss of session control and account access

✅ Mitigation:

* Use anti-CSRF tokens (synchronizer token pattern or double submit cookies)
* Validate Referer and Origin headers on sensitive actions
* Require re-authentication for critical changes
* Implement CAPTCHA for critical operations

1. Recommendations

* Sanitize and validate all user input (server-side)
* Use security headers: Content-Security-Policy, X-Frame-Options
* Implement CSRF tokens
* Disable detailed error messages in production
* Enable logging and real-time alerting for suspicious activity

─────────────────────────────

📋 OWASP Top 10 Mapping Summary Table

| **OWASP Category** | **Detected** | **Example Module** |
| --- | --- | --- |
| A1 – Injection | ✅ Yes | SQL Injection |
| A2 – Broken Authentication | ⛔ Not Tested |  |
| A3 – Sensitive Data Exposure | ⛔ Not Tested |  |
| A4 – XML External Entities (XXE) | ⛔ Not Tested |  |
| A5 – Broken Access Control | ✅ Yes | CSRF |
| A6 – Security Misconfiguration | ⚠️ Implied | Default creds used |
| A7 – Cross-Site Scripting (XSS) | ✅ Yes | Stored XSS |
| A8 – Insecure Deserialization | ⛔ Not Tested |  |
| A9 – Using Components with Vuln | ⚠️ DVWA itself |  |
| A10 – Insufficient Logging/Monitoring | ⛔ Not Tested |  |

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Conclusion

DVWA demonstrated critical and medium vulnerabilities that reflect real-world risks. These issues must be addressed before hosting any application publicly. Regular security testing and code review are recommended to maintain secure posture.