**🛡️ Web Application Security Assessment Report**

**Target Application:** DVWA (Damn Vulnerable Web Application)  
**Tools Used:** Burp Suite, OWASP ZAP, Firefox, DVWA, Kali Linux

**📄 1. Executive Summary**

This report presents the findings of a manual and automated vulnerability assessment of the Damn Vulnerable Web Application (DVWA). The assessment focused on identifying weaknesses as per OWASP Top 10 standards. Three high-impact vulnerabilities—**SQL Injection**, **Stored Cross-Site Scripting (XSS)**, and **Cross-Site Request Forgery (CSRF)**—were discovered and confirmed through manual exploitation.

**🧰 2. Tools Used**

| **Tool** | **Purpose** |
| --- | --- |
| Burp Suite | Interception and manual testing |
| OWASP ZAP | Automated scanning and crawling |
| Firefox ESR | Testing, proxy setup |
| DVWA | Vulnerable web app for testing |
| Kali Linux | OS with pre-installed security tools |

**Target:**

* **URL:** [**http://localhost/DVWA**](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)**

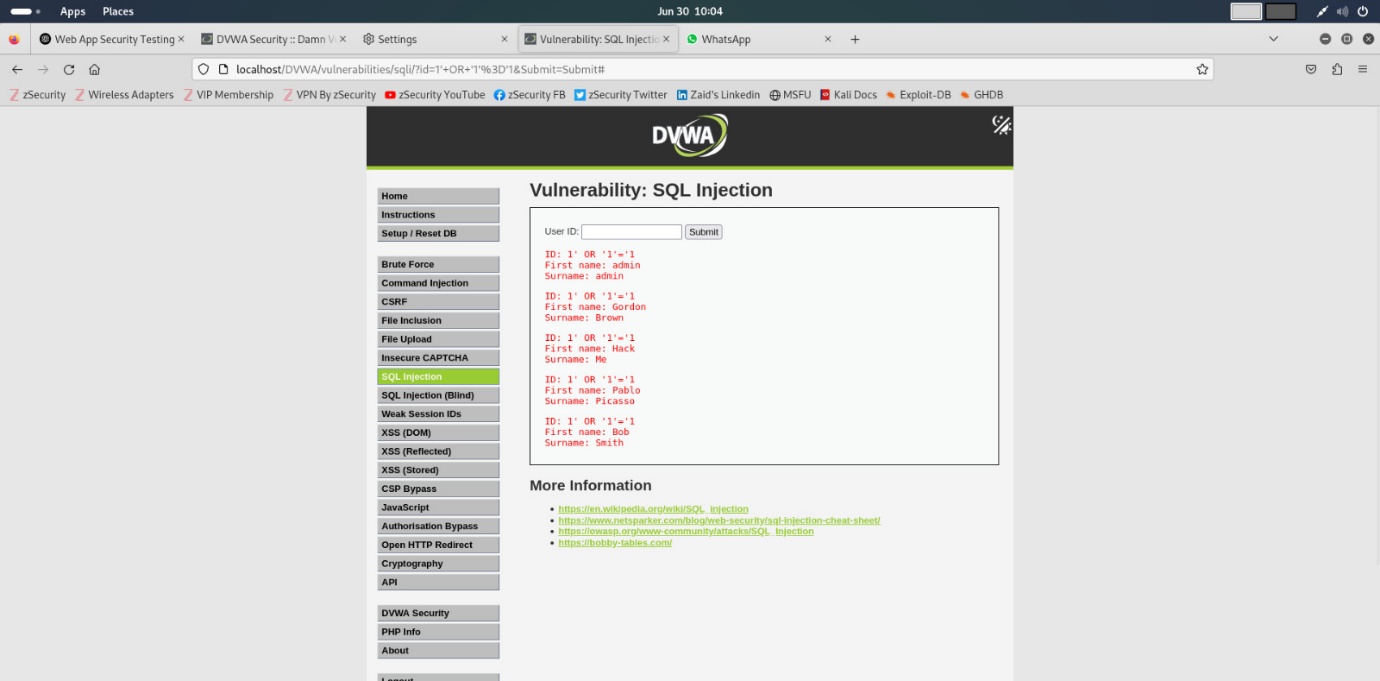
**🕵️ 3. Testing Methodology**

1. Reconnaissance using Burp Suite/ZAP
2. Manual vulnerability testing on DVWA modules
3. Exploitation of:
   * SQL Injection
   * Stored XSS
   * CSRF
4. Mapping results to OWASP Top 10
5. Documentation with screenshots and remediation guidance

**🔍 4. Vulnerabilities Found**

**🔴 4.1 SQL Injection**

* **Location:** vulnerabilities/sqli/
* **Payload:** 1' OR '1'='1
* **Impact:** Full database content exposure
* **Risk Rating:** High

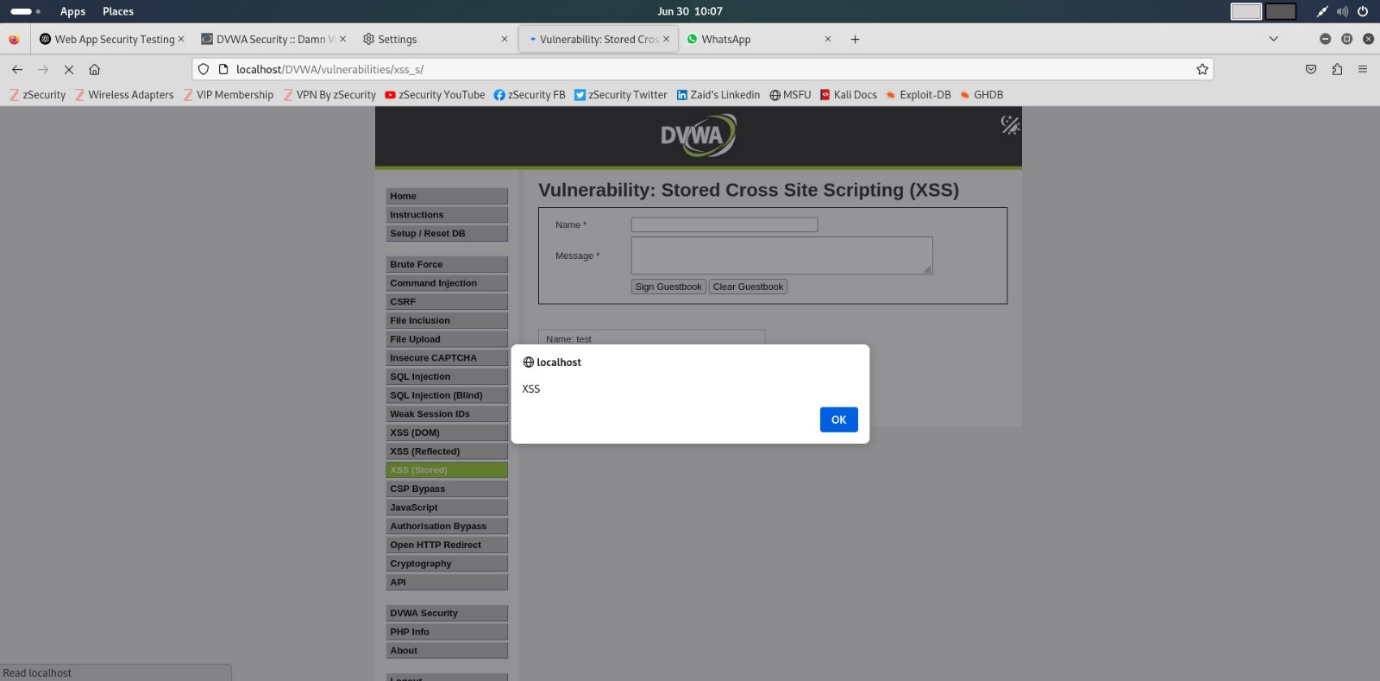


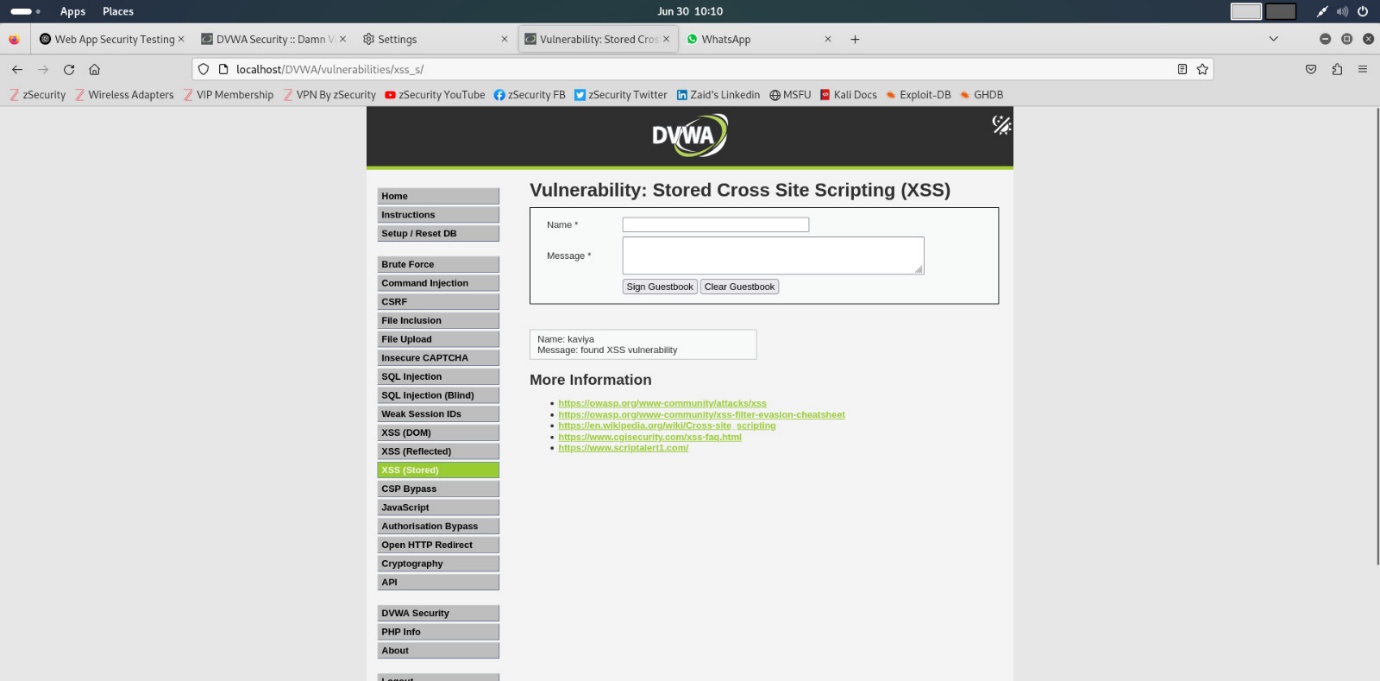
🔧 **Mitigation:**

* Use parameterized queries (e.g., PDO, prepared statements)
* Validate and sanitize user input
* Apply least privilege on database access

**🟠 4.2 Stored Cross-Site Scripting (XSS)**

* **Location:** vulnerabilities/xss\_s/
* **Payload:** <script>alert('XSS');</script>
* **Impact:** Executes attacker JavaScript in victim's browser
* **Risk Rating:** Medium–High



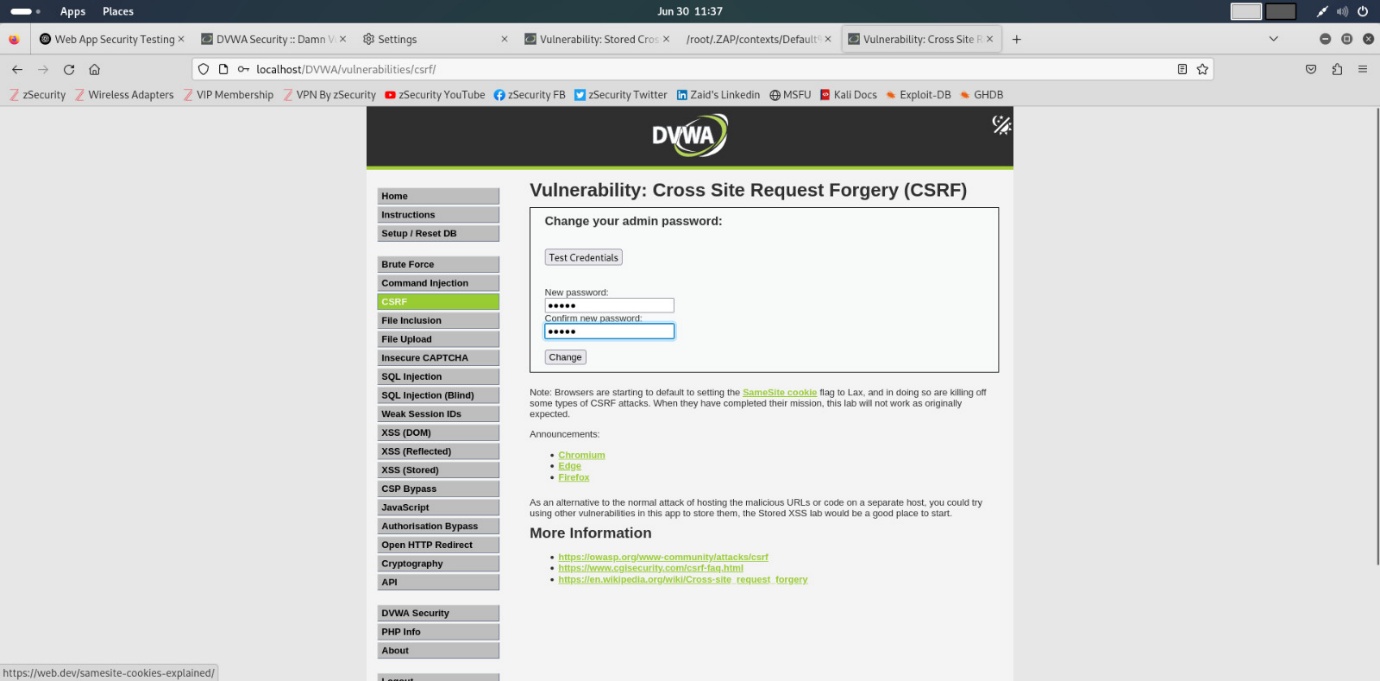


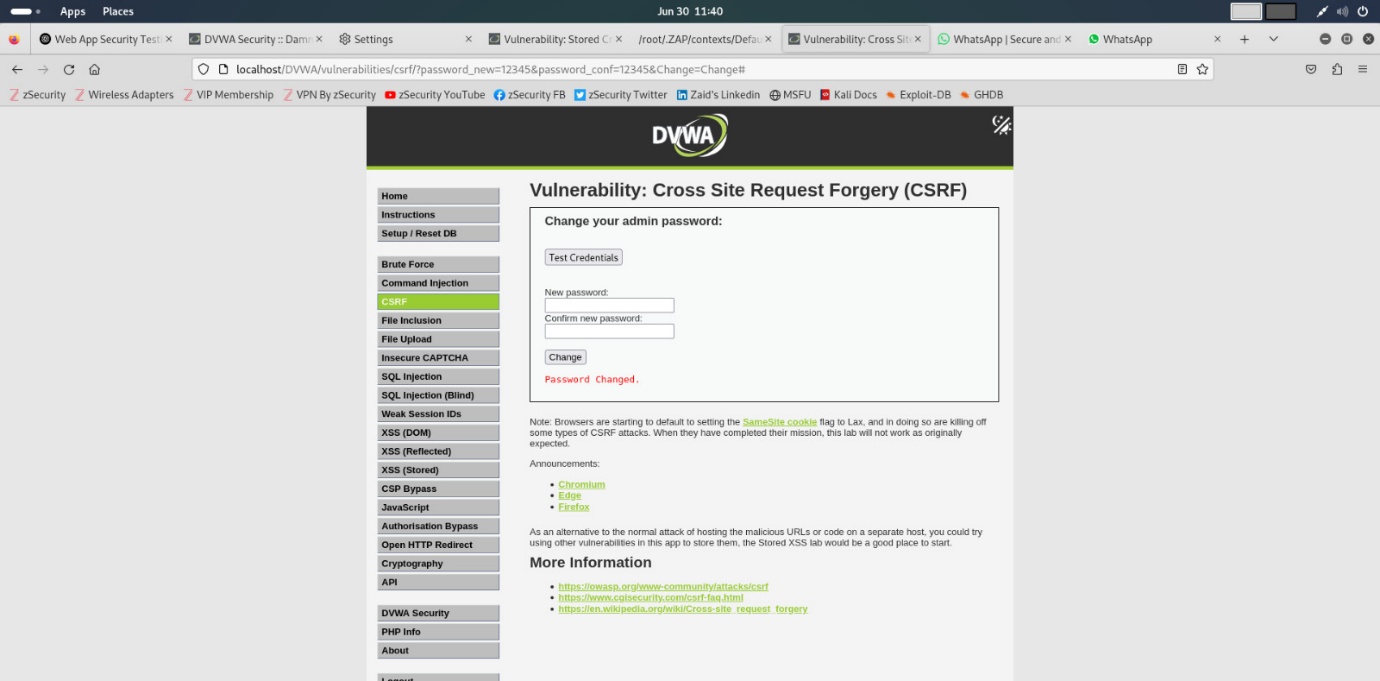
🔧 **Mitigation:**

* Escape user input on output (htmlspecialchars in PHP)
* Implement Content Security Policy (CSP)
* Sanitize inputs via libraries like DOMPurify

**🟡 4.3 Cross-Site Request Forgery (CSRF)**

* **Location:** vulnerabilities/csrf/
* **Payload:** Custom HTML form submitting password change without user action
* **Impact:** Password changed silently
* **Risk Rating:** Medium





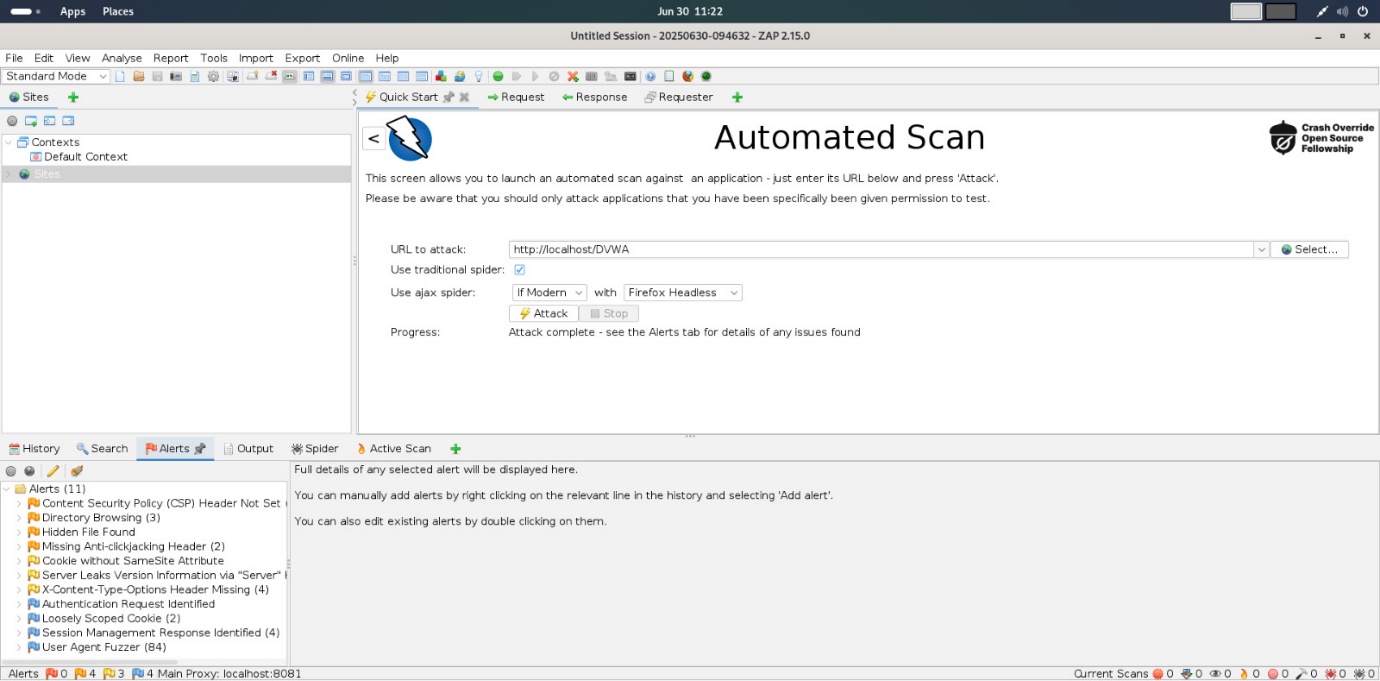
🔧 **Mitigation:**

* Use anti-CSRF tokens in all state-changing forms
* Require user re-authentication for sensitive actions
* Set SameSite attribute on cookies

**📊 5. OWASP Top 10 Compliance Checklist**

| **OWASP Category** | **Status** | **Notes** |
| --- | --- | --- |
| A01:2021 – Broken Access Control | ❌ Not tested explicitly |  |
| A02:2021 – Cryptographic Failures | ❌ Not in scope |  |
| A03:2021 – Injection | ✅ SQLi, XSS confirmed |  |
| A04:2021 – Insecure Design | ❌ Not applicable |  |
| A05:2021 – Security Misconfiguration | ✅ CSRF found; default config insecure |  |
| A06:2021 – Vulnerable and Outdated Components | ❌ DVWA purposely insecure |  |
| A07:2021 – Identification & Auth Failures | ❌ Not tested |  |
| A08:2021 – Software and Data Integrity Failures | ❌ Not in scope |  |
| A09:2021 – Security Logging and Monitoring | ❌ Not tested |  |
| A10:2021 – Server-Side Request Forgery | ❌ Not in scope |  |

**📁 6. Logs and Supporting Evidence**

**🔸 OWASP ZAP Rep**

🔍 **7 . Vulnerability Findings (with Risk Ratings & Recommendations**)

**✅ 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

**✅ 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

**✅ 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

**✅ 8. Summary of Risk Ratings**

| **Vulnerability** | **Risk Level** |
| --- | --- |
| SQL Injection | High |
| Stored XSS | Medium |
| CSRF | Medium |

**🛡 9. Final Recommendations**

1. **Fix critical vulnerabilities immediately** (SQLi, XSS)
2. **Implement secure coding practices** – input validation, output encoding
3. **Use frameworks/libraries with built-in CSRF protection**
4. **Review and monitor all forms and endpoints**
5. **Apply OWASP Secure Coding Guidelines**