# Web Application Vulnerability Assessment Report — DVWA (Local Lab)

Prepared by: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Date: 2025-10-17

Tested Application: Damn Vulnerable Web Application (DVWA)

Environment: Localhost — XAMPP on Windows (Apache 2.4.58, PHP 8.2.12, MySQL)

Scope: Local DVWA instance only (C:\xampp\htdocs\DVWA-master or C:\xampp\htdocs\dvwa)

Tools used: Browser DevTools, Burp Suite Community (optional), sqlmap (optional), OWASP ZAP (optional), XAMPP Control Panel

## Executive summary

A focused assessment of a local DVWA instance was performed to demonstrate common web vulnerabilities and to document reproducible proof-of-concept exploits. The confirmed vulnerabilities (Time-based blind SQL Injection, Reflected XSS, Stored XSS, OS Command Injection, and CSRF password change) are intentionally present for training. Each finding below includes PoC, impact, and remediation guidance. Attach corresponding screenshots and request/response captures in the Evidence section.

## Methodology

Testing followed OWASP guidance and pragmatic manual testing: reconnaissance, environment verification, manual payload testing, PoC capture (screenshots, network), and reporting.

## OWASP Top-10 mapping (covered)

• A03 — Injection: SQL Injection (Time-based)

• A07 — Identification & Authentication Failures / XSS: Reflected & Stored XSS

• A03 — Injection: OS Command Injection

• A05 / A04 — CSRF / Security Misconfiguration: CSRF password change

## Findings

### Finding 1 — Time-Based Blind SQL Injection (High)

Location / URL: http://localhost/dvwa/vulnerabilities/sqli/?id=1 (SQLI GET page)

Payload(s) used (PoC):  
 1 AND SLEEP(5)#

Steps to reproduce (short):  
1. DVWA → Vulnerabilities → SQL Injection (GET).  
2. Set Security = Low.  
3. Enter payload `1 AND SLEEP(5)#` in the ID input box and submit.  
4. Observe ~5s delay and the expected response (e.g., 'User ID exists').

Observation / Proof: The app delayed response by ~5s and returned results, confirming SQL executed on the backend.  
Impact: High — attacker can extract DB information using time-based techniques (data disclosure, auth bypass).  
Remediation: Use parameterized queries/prepared statements, validate input, least-privilege DB user, remove verbose DB error output.

### Finding 2 — Reflected Cross-Site Scripting (XSS) (High/Medium)

Location / URL: http://localhost/dvwa/vulnerabilities/xss\_r/ (Reflected XSS page)

Payload (PoC):  
 <script>alert('XSS')</script>

Steps to reproduce: Submit the payload in the input field on the Reflected XSS page (Security = Low) and observe the alert popup.  
Observation / Proof: The script executed in the browser context (alert shown).  
Impact: Medium–High — attackers can run arbitrary JS to steal cookies/session tokens.  
Remediation: Output-encode/escape user input, apply input validation, set HttpOnly on cookies, and use CSP.

### Finding 3 — Stored Cross-Site Scripting (XSS) (High/Critical)

Location / URL: http://localhost/dvwa/vulnerabilities/xss\_s/ (Stored XSS / Guestbook)

Payload (PoC):  
 <script>alert('Stored XSS')</script>

Steps to reproduce: Add a guestbook entry with the payload and submit. Reload/view the guestbook — the script executes for every viewer.  
Observation / Proof: Script persisted in DB and executed whenever page loaded.  
Impact: Critical — stored XSS affects multiple users and can be used for account compromise or malware delivery.  
Remediation: Sanitize inputs before storage, apply output encoding on rendering, implement CSP and HttpOnly cookies.

### Finding 4 — OS Command Injection (High/Critical)

Location / URL: http://localhost/dvwa/vulnerabilities/exec/ (Command Injection, ping form)

Payloads (PoC):  
 127.0.0.1 && whoami  
 127.0.0.1 && dir

Steps to reproduce: Submit `127.0.0.1 && whoami` on the Command Injection page (Security = Low) and observe the OS command output (e.g., `laptop-...\aakash`).  
Observation / Proof: Execution of `whoami` confirmed, proving arbitrary command execution.  
Impact: Critical — arbitrary OS command execution can lead to full system compromise.  
Remediation: Avoid passing user input to shell commands; whitelist and validate inputs; use safe APIs and least privilege.

### Finding 5 — Cross-Site Request Forgery (CSRF) — Password Change (High)

Location / URL: http://localhost/dvwa/vulnerabilities/csrf/ (CSRF page)

PoC (direct GET):  
 http://localhost/dvwa/vulnerabilities/csrf/?password\_new=attacker123&password\_conf=attacker123&Change=Change

Steps to reproduce: While logged in as admin, open the PoC URL or submit a crafted form from another site. The password is changed without a CSRF token present.  
Observation / Proof: Password changed via GET request without anti-CSRF protections.  
Impact: High — attacker can force actions (password change, transactions) leading to account takeover.  
Remediation: Implement per-request anti-CSRF tokens, set SameSite cookie attributes, validate Origin/Referer headers, and require POST for state-changing actions.

## Evidence (attach images/files)

Please attach the screenshots and exported logs inside an evidence/ folder and name them as follows. Placeholders are provided where images should be inserted in the final document:

**dvwa\_sqli\_timebased\_YYYYMMDD\_HHMM.png :** Address bar with SQLi payload + screenshot showing delay/response

**dvwa\_xss\_reflected\_alert\_YYYYMMDD\_HHMM.png :** Reflected XSS alert screenshot (payload + popup)

**dvwa\_xss\_stored\_alert\_YYYYMMDD\_HHMM.png :** Stored XSS persisted payload + popup screenshot

**dvwa\_cmdinj\_whoami\_YYYYMMDD\_HHMM.png :** Command injection output showing whoami result

**dvwa\_cmdinj\_dir\_YYYYMMDD\_HHMM.png :** Command injection output showing dir listing (optional)

**dvwa\_csrf\_poc\_url\_YYYYMMDD\_HHMM.png :** Address bar with CSRF PoC URL (GET parameters visible)

**dvwa\_csrf\_login\_proof\_YYYYMMDD\_HHMM.png :** Successful login screenshot using admin/attacker123 as proof

**dvwa\_network\_req\_csrf\_YYYYMMDD\_HHMM.png :** DevTools Network request showing GET parameters for CSRF request

**zapro\_report\_YYYYMMDD.html :** OWASP ZAP HTML report (if generated)

**sqlmap\_output\_YYYYMMDD.txt :** sqlmap output (if used)

How to insert images: Open this DOCX in Microsoft Word, place cursor where you want the screenshot, then Insert -> Pictures -> choose the image file from the evidence/ folder. Use captions like 'Figure 1 — dvwa\_sqli\_timebased\_YYYYMMDD\_HHMM.png'.

## Risk summary & recommended priorities

Immediate / High Priority fixes:  
• Parameterize SQL queries and remove unsanitized concatenation.  
• Remove or secure any OS command invocation that uses user data; use whitelist validation.  
• Add CSRF tokens and enforce SameSite & HttpOnly on cookies.  
• Sanitize/encode output to mitigate XSS.  
  
Medium Priority:  
• Harden server configuration, remove default files, disable directory listing.  
• Use least-privilege service accounts, enable logging and monitoring.  
  
Low Priority:  
• Implement CSP headers, keep libraries up-to-date, apply secure headers.

## Conclusion

The DVWA instance deliberately includes multiple severe vulnerabilities. Manual testing confirmed exploitable SQL Injection (time-based), reflected & stored XSS, OS command injection, and CSRF on a state-changing operation. The provided remediations should be applied as development best-practices and followed by a retest in a staging environment.

## Deliverables (what to submit)

• This DOCX/PDF report.  
• Evidence folder (screenshots & tool outputs).  
• OWASP Top 10 checklist.  
• (Optional) Short demo video showing PoC steps.

---  
Note: When you share the evidence images with me, I can embed them into this document in the correct locations and return a completed DOCX/PDF with images in place. Save your images using the filenames listed above for easy embedding.