DVWA Focused Security Assessment — SQL Injection, XSS, Authentication Flaws

Date: 2025-11-08

Assessor: Pawan K C

Target: DVWA instance (lab/local)

Scope: This report covers **only** three vulnerability classes tested on the DVWA instance: **SQL Injection (SQLi)**, **Cross-Site Scripting (XSS)** — both reflected and stored — and **Authentication flaws** (weak credentials, password handling, session management). Replace evidence placeholders with your screenshots, request/response logs, or Burp/ZAP export snippets.

---

1. Executive summary

- **Overview:** The DVWA instance (configured at Low security level unless noted) exhibits exploitable SQL Injection, multiple XSS vectors (reflected and stored), and weak authentication controls consistent with DVWA's training purpose. Each of these vulnerabilities would be critical or high impact in a production environment, enabling data disclosure, session compromise, account takeover, and potential further system compromise.
- **Top remediation priorities:** (1) Fix SQL Injection by using parameterized queries and input validation; (2) Prevent XSS via context-aware output encoding and sanitization; (3) Harden authentication — enforce strong password storage (bcrypt/Argon2), account lockout/rate-limiting, and secure session cookie attributes.

---

2. Rules of engagement & tools

- **Target:** DVWA in an authorized lab environment only.
- **Tools used:** Burp Suite (Proxy/Repeater), OWASP ZAP (passive scan), sqlmap (non-destructive), Browser devtools, curl.
- **Notes:** Tests were non-destructive; only harmless proof-of-concept payloads were executed (e.g., alert() or echo POC).

---

3. Methodology (brief)

1. Enumerate relevant modules/endpoints in DVWA (/vulnerabilities/sqli/, /vulnerabilities/xss_r/, /vulnerabilities/xss_s/, /login.php, etc.).
2. Passive crawl with ZAP while browsing application to collect endpoints.

3. Manual testing with Burp Repeater and targeted payloads.
4. Verified SQLi with sqlmap in discovery mode (no dumping).
5. Documented PoCs and recommended fixes.

---

4. Findings — SQL Injection

4.1 Description

- **Affected endpoint:** GET /dvwa/vulnerabilities/sqli/?id=<value>&Submit=Submit (parameter id).
- **Behavior:** Unsanitized input is concatenated into a SQL query. At DVWA Low level the application returns query results that confirm injection.

4.2 PoC (safe)

- **Manual curl test:**

curl -i "http://127.0.0.1/dvwa/vulnerabilities/sqli/?id=1' OR '1'='1'&Submit=Submit"

- **sqlmap discovery (non-destructive):**

sqlmap -u "http://127.0.0.1/dvwa/vulnerabilities/sqli/?id=1&Submit=Submit" -p id --batch --level=2 --risk=1

4.3 Evidence

- *Insert screenshot of response showing injected result or Burp Repeater request/response here.*

4.4 Impact

- Database disclosure (usernames, password hashes), ability to read/modify data, potential privilege escalation depending on DB permissions.

4.5 Root cause

- Dynamic SQL constructed with string concatenation and no parameterization or proper input validation.

4.6 Recommendations

1. Use parameterized queries / prepared statements for all DB access. Example (PHP PDO):

$stmt = $pdo->prepare('SELECT * FROM products WHERE id = ?');

$stmt->execute([$id]);

2. Validate inputs server-side (whitelist numeric IDs where applicable).
3. Use least-privilege DB accounts and disable privileged functions for web-facing accounts.
4. Add WAF rules to block common SQLi patterns as a temporary mitigation.

4.7 Severity

- **CVSS (estimate):** 9.1 — Critical.

4.8 Retest checks

- Re-run sqlmap discovery; verify parameterized queries block injection and application returns sanitized errors (no DB info leakage).

---

5. Findings — Cross-Site Scripting (XSS)

5.1 Reflected XSS

- **Affected endpoint:** /dvwa/vulnerabilities/xss_r/ (query form reflecting input).
- **PoC payloads:** "><script>alert('XSS')</script> or "><svg/onload=alert(1)>.

Test (curl):

curl -i "http://127.0.0.1/dvwa/vulnerabilities/xss_r/?name=%3Cscript%3Ealert('XSS')%3C/script%3E"

Impact: An attacker can craft a link that executes script in victims' browsers — session theft if cookies are not HttpOnly, UI redress/phishing.

Root cause: Unencoded user input reflected directly into HTML response.

Fixes:

1. Apply context-aware output encoding on all user-supplied data prior to rendering (HTML entity encode, attribute encode, JS encode as appropriate). Use templating engines that auto-escape by default.
2. Implement Content Security Policy (CSP) to limit script execution origins.
3. Set HttpOnly and Secure on session cookies to reduce impact.

5.2 Stored XSS

- **Affected endpoint:** /dvwa/vulnerabilities/xss_s/ (comment/profile storage).

- **PoC:** Post <script>alert('stored XSS')</script> as a comment; observe execution when page viewed.

Impact: Persistent script execution across multiple users, leading to session hijack, mass phishing, and potential account takeover.

Root cause: Lack of server-side sanitization and improper output encoding when rendering stored content.

Fixes:

1. Sanitize incoming rich text with a safe library (e.g., DOMPurify) and whitelist allowed tags/attributes.
2. Encode output for the intended context.
3. Use CSP and consider setting Content-Security-Policy: default-src 'self'; object-src 'none'; as part of defense-in-depth.

Severity: 7.4 — High.

Retest: Attempt same stored payload; verify it is rendered as harmless text or blocked by CSP.

---

6. Findings — Authentication flaws

This section groups authentication issues observed in DVWA lab configuration and general hardening recommendations for production systems.

6.1 Weak/default credentials

- **Observation:** DVWA commonly ships with simple/default credentials in labs (e.g., admin:password).
- **Impact:** Easy account takeover via credential guessing or reuse.
- **Fixes:** Enforce strong passwords, ban default credentials, require password resets on first use for real deployments.

6.2 Password storage

- **Observation:** In training setups passwords or weak hashes may be present for demonstration.
- **Impact:** If weak hashing is used in production (e.g., MD5, unsalted), compromise of password file leads to credential theft.
- **Fixes:** Use modern password hashing algorithms (Argon2id, bcrypt) with per-user salt and appropriate cost factors.

6.3 Session management and cookie flags

- **Observation:** Session cookies in DVWA lab over HTTP lack Secure, HttpOnly, and suitable SameSite attributes.
- **Impact:** Client-side scripts can access cookies (XSS risk), and CSRF is easier without SameSite/CSRF tokens.
- **Fixes:**
    1. Serve application only over HTTPS. Set cookies with Secure; HttpOnly; SameSite=Strict (or Lax where needed).
    2. Implement short session timeouts and inactivity logout.
    3. Regenerate session identifiers after authentication and privilege changes.

6.4 Authentication enumeration & brute-force protection

- **Observation:** Login endpoints may reveal different error messages for unknown user vs wrong password, enabling enumeration.
- **Impact:** User enumeration facilitates targeted attacks and credential stuffing.
- **Fixes:**
    1. Return ambiguous authentication failure messages (e.g., "Invalid credentials").
    2. Implement rate-limiting, account lockout (with progressive delays), and CAPTCHA/MFA for risky attempts.

6.5 Multi-Factor Authentication (MFA)

- **Recommendation:** For production systems, require or offer MFA (TOTP, FIDO2) for privileged accounts.

Severity (overall): Medium to High depending on specific issue — weak/default creds and missing cookie flags can be Medium; lack of MFA or easy account takeover is High.

Retest: Verify password hashes are stored using recommended algorithms, cookie attributes present, session reuse after logout denied, account lockout triggers after multiple failed attempts.

---

7. Combined impact scenarios

- **SQLi → Auth compromise:** Extract user table via SQLi to obtain password hashes; if weak hashing used, offline cracking can yield valid credentials for account takeover.

- **XSS → Session theft:** Reflected or stored XSS can steal session cookies (if HttpOnly not set) enabling account takeover without needing passwords.
- **Upload/LFI combos (note):** Although out of scope for this focused report, SQLi/XSS/auth weaknesses often combine with other vulnerabilities to escalate impact.

---

8. Prioritized remediation actions (practical)

1. **Immediate (within 72 hours):**
    - Disable lab/test instances from public networks. Ensure test DVWA is not accessible externally.
    - Enforce HTTPS and set session cookies Secure; HttpOnly; SameSite=Strict.
    - Apply WAF rules to block simple SQLi payloads while developers remediate code.
2. **Short-term (1–3 weeks):**
    - Replace vulnerable SQL code with parameterized queries/ORM usage.
    - Add output encoding and input sanitization libraries; sanitize stored content.
    - Enforce strong password policy, rotate any default credentials in test environments.
    - Implement CSRF tokens and consistent authentication failure messaging.
3. **Medium-term (3–8 weeks):**
    - Introduce MFA for admin and sensitive users.
    - Add logging/monitoring for anomalous DB queries, failed auth attempts, and XSS alerts.
4. **Long-term:**
    - Integrate SAST/DAST into CI, conduct periodic pentests, and train developers on secure coding practices.

---

9. Retest checklist

- SQLi payloads return no DB data; error messages are generic and do not leak engine info.

- Reflected and stored XSS payloads do not execute; output properly encoded and sanitized.
- Session cookies include Secure; HttpOnly; SameSite and rotate on login; session tokens invalid after logout.
- Login endpoint returns uniform error messages; rate-limiting / lockouts function as expected.

---

10. Appendix — Key commands & payloads

SQLi testing (discovery only)

sqlmap -u "http://127.0.0.1/dvwa/vulnerabilities/sqli/?id=1&Submit=Submit" -p id --batch --level=2 --risk=1

Reflected XSS test (curl)

curl -i "http://127.0.0.1/dvwa/vulnerabilities/xss_r/?name=%3Cscript%3Ealert('XSS')%3C/script%3E"

Stored XSS (post safely via app form)

- Submit <script>alert('stored XSS')</script> as a comment in the stored XSS module (lab only).

Authentication checks

- Observe Set-Cookie headers via Burp Proxy and verify attributes.
- Use Burp Repeater to verify uniform error messages for invalid credentials.
