DVWA Penetration Testing Lab Report

Author: Yuve Bharjana
Date: February 18, 2026
Purpose: Self-directed ethical lab to demonstrate basic web vulnerability assessment and reporting skills for potential client-side pen testing assistance.

Executive Summary
This report documents a hands-on penetration testing exercise on Damn Vulnerable Web Application (DVWA) running locally via Docker at Low security level. Three common OWASP Top 10 vulnerabilities were exploited: SQL Injection, Reflected Cross-Site Scripting (XSS), and Command Injection. Findings include proof-of-concept exploits, potential real-world impacts, and recommended mitigations. The exercise highlights the importance of input validation and output encoding in new web software development.

Scope & Methodology
- Target: DVWA (official Docker image, http://localhost:8080)
- Security Level: Low
- Testing Approach: Manual exploitation using browser tools
- Ethical Note: Fully self-contained lab – no external or production targets

Findings

1. SQL Injection
- **Location**: /vulnerabilities/sqli/
- **OWASP Category**: A03:2021 – Injection
- **Severity**: High
- **Description**: Unsanitized user input in SQL queries allows arbitrary database manipulation.
- **Proof of Concept**:
  1. Enter `' OR 1=1 -- -` → Returns all 5 users from the database.
  2. Enter `' UNION SELECT user, password FROM users -- -` → Extracts usernames and MD5 password hashes.
- **Impact**: In a production application, this could lead to full data exfiltration, credential theft, or database modification/deletion.
- **Evidence**:
  ![SQLi All Users Dump](screenshots/sqli/sqli-all-users.png)
  ![SQLi Hashes Extracted](screenshots/sqli/sqli-union-hashes.png)
- **Remediation**: Use prepared statements / parameterized queries (e.g., PDO in PHP with bindParam). Never concatenate user input into SQL strings.
- **Lesson**: Always validate and sanitize inputs on the backend – this is a classic backend vulnerability.

2. Reflected Cross-Site Scripting (XSS)
- **Location**: /vulnerabilities/xss_r/

- **OWASP Category**: A07:2021 – Cross-Site Scripting (XSS)
- **Severity**: High
- **Description**: Unsanitized input is reflected back in the response, allowing arbitrary JavaScript execution in the victim's browser.
- **Proof of Concept**: Enter `<script>alert('XSS by Yuve')</script>` → Browser executes the alert popup.
- **Impact**: In real apps, attackers could steal session cookies, perform actions as the user, or deliver phishing content.
- **Evidence**:
  ![XSS Payload Input](screenshots/xss/xss-input.png)
  ![XSS Alert Popup](screenshots/xss/xss-alert.png)
- **Remediation**: Encode output with `htmlspecialchars()` (PHP) or equivalent framework escaping (React's JSX auto-escapes, but validate on server too). Implement Content Security Policy (CSP).
- **Lesson**: Output encoding is critical for client-side safety – reflected XSS often slips through frontend-only validation.

3. Command Injection
- **Location**: /vulnerabilities/exec/
- **OWASP Category**: A03:2021 – Injection (OS Command Injection)
- **Severity**: Critical
- **Description**: Unsanitized input passed to system commands allows arbitrary OS command execution.
- **Proof of Concept**:
  1. Enter `127.0.0.1 && whoami` → Reveals server user (e.g., www-data).
  2. Enter `127.0.0.1 && ls /var/www/html` → Lists web root directory.
- **Impact**: Full server compromise possible (webshell, data theft, ransomware).
- **Evidence**:
  ![Command Inj whoami](screenshots/command-injection/whoami-output.png)
  ![Command Inj ls](screenshots/command-injection/ls-output.png)
- **Remediation**: Use safe APIs (e.g., PHP's `escapeshellarg()` or `escapeshellcmd()`). Avoid shell_exec/system entirely if possible; use libraries instead.
- **Lesson**: Never trust user input in system calls – this is a direct backend-to-OS bridge vulnerability.

Recommendations Summary
- Prioritize parameterized queries and prepared statements for all database interactions.
- Implement strict output encoding and Content Security Policy headers.
- Sanitize/validate inputs before passing to OS commands or external processes.
- Conduct regular security testing (static/dynamic analysis) during development of new software.

Conclusion

This lab exercise built practical skills in identifying, exploiting, and documenting web vulnerabilities. Ready to apply these basics to real scans, reporting, or remediation verification on client-side projects.

GitHub Repo: https://github.com/YuvdeepBharjana/DVWA-Penetration-Testing-Project