Vulnerability Assessment & Penetration Testing (VAPT)

Damn Vulnerable Web Application (DVWA)

Author: Amit Mondal

Environment: Kali Purple VM

Date: December 2025

Executive Summary

This report documents a complete Vulnerability Assessment and Penetration Testing (VAPT) conducted on the Damn Vulnerable Web Application (DVWA). The objective of this engagement was to identify, exploit, and document vulnerabilities using industry-standard penetration testing approaches. The assessment revealed multiple critical issues, including SQL Injection, Command Injection, Cross-Site Scripting, File Upload exploitation, and several misconfigurations. These findings demonstrate real-world attack scenarios and highlight the importance of secure coding and system hardening practices.

Scope of Assessment

The assessment was limited to the local DVWA environment running on Kali Purple. The target was installed under Apache2 with a MariaDB backend. Only the DVWA instance was considered within scope, and no external systems were tested. The tools used during this engagement included Burp Suite, Nmap, Nikto, Firefox (configured with proxy), Apache, MariaDB, and PHP.

Methodology

The engagement followed standard OWASP-aligned penetration testing methodology. The reconnaissance phase focused on identifying active services and server information through Nmap scanning. The scanning phase included automated checks using Nmap and Nikto to uncover server weaknesses and insecure configurations. Manual exploitation techniques were then applied to validate critical vulnerabilities, including SQL Injection, Command Injection, XSS, and arbitrary file upload issues. Post-exploitation steps verified the impact of these findings, such as database extraction and system-level command execution. Finally, all results were documented with clear evidence and remediation guidance.

Key Findings Summary

Vulnerability	Severity	Description
SQL Injection	Critical	Enabled extraction of database name, user, version, and hostname. Authentication bypass was also possible.
Command Injection	Critical	Allowed execution of OS-level commands as the webserver user (www-data).
Stored Cross-Site Scripting	High	Persistent JavaScript execution across sessions due to improper output handling.
Reflected Cross-Site Scripting	High	Immediate JavaScript execution through unsanitized request parameters.
Unrestricted File Upload	High	Permitted uploading of malicious PHP files, enabling remote code execution potential.
Server Misconfigurations	Medium	Missing security headers, directory indexing, and exposure of sensitive files.

Detailed Findings

1. SQL Injection (Critical)

The SQL Injection vulnerability was identified in the `/vulnerabilities/sqli/?id=` parameter. Tested payloads included: `1' OR '1'='1 1' UNION SELECT 1,2 -- - 1' UNION SELECT user(),2 -- - 1' UNION SELECT database(),2 -- - 1' UNION SELECT version(),2 -- - 1' UNION SELECT @@hostname,2 -- -`

The application returned sensitive database information and allowed logical bypass of authentication. Remediation requires parameterized SQL queries, strict input validation, and disabling verbose SQL error messages.

2. Command Injection (Critical)

The command injection issue was found in the DVWA command execution module. Tested payloads included: `127.0.0.1; whoami 127.0.0.1; ls -la`

The application executed operating system commands in the context of the Apache user. Mitigation involves removing unsafe PHP functions, validating server-side input strictly, and implementing allowlists.

3. Stored Cross-Site Scripting (High)

The stored XSS vulnerability allowed insertion of persistent JavaScript payloads. Example payload: `<script>alert('Stored XSS')</script>`

This issue enables session hijacking or phishing attacks. Fixing this requires output encoding, sanitization, and enforcing Content Security Policy.

4. Reflected Cross-Site Scripting (High)

The application reflected unsanitized user input back into the page. Example payload: `<script>alert('Hello')</script>`

Mitigation includes sanitizing all dynamic output and enforcing server-side input validation.

5. File Upload Vulnerability (High)

DVWA allowed uploading of arbitrary PHP files, which were stored in an accessible directory. This creates a remote code execution risk. Remediation includes blocking executable uploads, validating MIME types, renaming uploaded files, and disabling PHP execution in the upload directory.

6. Server Misconfigurations (Medium)

Nmap and Nikto scans revealed missing security headers, directory indexing, and sensitive file exposure. Recommended actions include applying HTTP security headers, disabling directory listing, and restricting access to internal folders.

CVSS Scoring Summary

Vulnerability	**Score**	**Severity**
SQL Injection	9.8	Critical
Command Injection	9.8	Critical
File Upload Vulnerability	8.6	High
Stored and Reflected XSS	7.5	High
Server Misconfigurations	5.0	Medium

Recommendations

Applications should enforce strict input validation and adopt secure development practices throughout the stack. Parameterized queries must replace all dynamic SQL statements to prevent SQL injection. Server-side sanitization, output encoding, and Content Security Policy should be implemented to defend against XSS. The upload directory must block executable files and prevent code execution. To harden the environment further, unnecessary PHP functions should be disabled, server headers should be applied, and directory indexing must be turned off.

Conclusion

This VAPT engagement effectively demonstrated real-world vulnerabilities present in insecure web applications. The assessment confirmed exploitation of SQL Injection, Command Injection, File Upload abuse, and Cross-Site Scripting. The findings illustrate the risks associated with insecure coding practices and emphasize the need for defense-in-depth strategies. This project reinforces practical penetration testing skills and aligns with professional security assessment methodologies.

Author

Amit Mondal, Cybersecurity Enthusiast and Ethical Hacker