WEB APPLICATION PENETRATION TEST REPORT

Project Name: DVWA Security Assessment
Client: Capstone Project
Date: January 09, 2026
Author: Pranay Vasanth

Table of Contents

---

1. Executive Summary

1.1 Overview

On January 9, 2026, a comprehensive security assessment was conducted on the **Damn Vulnerable Web Application (DVWA)**. The primary objective was to identify security flaws, demonstrate potential impact, and recommend remediation strategies to secure the application against real-world attacks.

1.2 Summary of Findings

The assessment revealed **critical** security vulnerabilities that could lead to a complete compromise of the web server and the database. The most severe issue identified was **Remote Command Execution**,

allowing unauthorized system access.

Risk Level	Count	Vulnerabilities Identified
Critical	1	Remote Command Injection
High	1	SQL Injection (Union-Based)
Medium	2	Stored XSS, Weak Authentication
Low	1	Information Leakage (Version Headers)

1.3 Risk Assessment

The overall risk posture of the application is **CRITICAL**. Without immediate remediation, the application is susceptible to data theft, defacement, and unauthorized server control.

---

2. Scope & Methodology

2.1 Scope

- **Target:** `http://[192.168.1.9]/dvwa/`
- **Included Components:**
- Authentication Mechanisms
- Input Validation Modules (SQLi, XSS, Command Injection)
- Session Management
- **Excluded:** Denial of Service (DoS), Social Engineering.

2.2 Methodology

This assessment followed the **OWASP (Open Web Application Security Project)** testing guide and **PTES (Penetration Testing Execution Standard)**:

1. **Intelligence Gathering:** Passive and active reconnaissance (Nmap, Nikto).
2. **Threat Modeling:** Identifying potential attack vectors.
3. **Vulnerability Analysis:** Automated scanning and manual verification.
4. **Exploitation:** Controlled exploitation to verify findings.
5. **Reporting:** Documentation of results and fixes.

2.3 Tools Used

- **Kali Linux:** Testing environment.
- **Burp Suite:** Web proxy and request manipulation.
- **Nmap:** Network discovery.
- **Nikto:** Web server vulnerability scanning.
- **Hydra:** Brute-force credential testing.

---

3. Network Architecture Diagram

The following diagram illustrates the testing environment. The Attacker (Kali Linux) communicates with the Web Server (DVWA) through HTTP requests, bypassing basic client-side filters.

```
graph LR
    A[Attacker (Kali Linux)<br>IP: 192.168.1.10] -- HTTP Requests --> B(F
    B -- Forward Traffic --> C[Web Server (Apache/DVWA)<br>IP: 192.168.1.
    C -- SQL Queries --> D[(MySQL Database)]
    C -- System Calls --> E[Operating System<br>(Linux Kernel)]

    style A fill:#ff9999,stroke:#333,stroke-width:2px
    style C fill:#99ccff,stroke:#333,stroke-width:2px
    style D fill:#ffff99,stroke:#333,stroke-width:2px
    style E fill:#e0e0e0,stroke:#333,stroke-width:1px
```

---

4. Technical Findings (Detailed)

4.1 Vulnerability: Remote Command Injection

- **Severity: CRITICAL**
- **CVSS Score:** 9.8
- **Location:** `/vulnerabilities/exec/`

Description:
The application fails to properly sanitize user input in the "Ping" utility. It passes user input directly to a shell command (`ping`). An attacker can append operating system commands using delimiters like `;` or `|` .

Proof of Concept (PoC):
The attacker injected the command `; cat /etc/passwd` to read the system's password file.

- **Payload:** `127.0.0.1; cat /etc/passwd`

Impact:
Full system compromise. An attacker can read sensitive files, install malware, or create a reverse shell to control the server.

4.2 Vulnerability: SQL Injection (Union Based)

- **Severity: HIGH**
- **CVSS Score:** 8.2
- **Location:** `/vulnerabilities/sqli/`

Description:
The "User ID" parameter is not validated before being used in a MySQL query. This allows an attacker to inject arbitrary SQL code to manipulate the database query.

Proof of Concept (PoC):
The attacker used a `UNION SELECT` statement to retrieve the database version and user credentials.

- **Payload:** `1' UNION SELECT user, password FROM users #`

Impact:
Confidentiality breach. Attackers can dump the entire database, including user passwords, emails, and personal information.

4.3 Vulnerability: Stored Cross-Site Scripting (XSS)

- **Severity: MEDIUM**
- **CVSS Score:** 6.1
- **Location:** `/vulnerabilities/xss_s/` (Guestbook)

Description:
The Guestbook stores user comments without HTML entity encoding. Malicious JavaScript entered by an attacker is saved in the database and executed in the browser of anyone who views the page.

Proof of Concept (PoC):

- **Payload:** `<script>alert(document.cookie)</script>`
- **Result:** A pop-up displaying the user's session ID (PHPSESSID) appeared upon loading the page.

Impact:
Session Hijacking. Attackers can steal session cookies, impersonate users (including admins), and redirect users to phishing sites.

---

5. Remediation Strategy

To secure the DVWA application, the following mitigations must be implemented immediately:

5.1 Fix Command Injection

Vulnerable Code:

```
$target = $_REQUEST[ 'ip' ];
$cmd = shell_exec( 'ping  -c 4 ' . $target );
```

Secure Implementation:
Avoid using `shell_exec` if possible. If required, use `escapeshellarg()` or validatation whitelisting.

```
$target = $_REQUEST[ 'ip' ];
// Only allow IP format
if(filter_var($target, FILTER_VALIDATE_IP)) {
    $cmd = shell_exec( 'ping  -c 4 ' . $target );
}
```

5.2 Fix SQL Injection

Vulnerable Code:

```
$id = $_GET['id'];
$query = "SELECT first_name, last_name FROM users WHERE user_id = '$id'";
```

Secure Implementation (Prepared Statements):
Use PDO to parameterize the query. The database treats the input as data, not code.

```
$stmt = $pdo->prepare('SELECT first_name, last_name FROM users WHERE user
$stmt->execute(['id' => $id]);
$user = $stmt->fetch();
```

5.3 Fix Stored XSS

Vulnerable Code:

```
echo "Message: " . $message;
```

Secure Implementation (Output Encoding):
Convert special characters to HTML entities before displaying.

```
echo "Message: " . htmlspecialchars($message, ENT_QUOTES, 'UTF-8');
```

---

6. Conclusion

The DVWA web application in its current state is unfit for production use. The existence of **Remote Command Injection** and **SQL Injection** vulnerabilities poses an unacceptable risk.

By implementing the **Remediation Strategies** outlined in Section 5—specifically switching to Prepared Statements and implementing strict Input Validation—the organization can significantly reduce the attack surface and secure the application against these common threats.