Subtask 5

SQL Injection Assessment Report

Target: DVWA (Damn Vulnerable Web Application)
Hosting Platform: TryHackMe AttackBox
Testing Method: Automated SQL Injection using sqlmap
Security Level: Low

---

1. Objective

The objective of this assessment was to:

- Identify SQL Injection vulnerability in DVWA
- Capture authentication cookies
- Craft a valid request including session cookies
- Use sqlmap to automate exploitation
- Enumerate and retrieve database information
- Document findings and impact

This testing was conducted in a controlled lab environment using TryHackMe infrastructure.

---

2. Lab Environment Setup

Platform Used

- TryHackMe AttackBox (Kali-based environment)
- DVWA hosted on TryHackMe machine
- Browser developer tools for cookie extraction
- sqlmap for automated exploitation

Target Details

- Target IP: 10.48.187.181
- Vulnerable Page:

/vulnerabilities/sqli/

---

3. Step 1 – Obtaining Session Cookies

After logging into DVWA using valid credentials:

- Opened browser Developer Tools
- Navigated to Network tab
- Selected the request to the SQL Injection page
- Extracted cookies from the request headers

Cookies Identified:

- PHPSESSID = (session ID value)
- security = low

The security=low cookie confirms the application is running in low-security mode, which disables input filtering protections.

---

4. Step 2 – Crafting the Target URL

The SQL injection page uses a GET parameter:

id=1

The full vulnerable URL format:

http://10.48.187.181/vulnerabilities/sqli/?id=1&Submit=Submit#

To maintain authenticated access, the request must include:

PHPSESSID=<session_value>; security=low

This ensures sqlmap operates within the authenticated DVWA session.

---

5. Step 3 – Executing SQLMap

sqlmap was used to test and exploit the vulnerable id parameter.

General approach:

- Provided target URL
- Included authentication cookies
- Used enumeration options to retrieve database contents
- Enabled automatic execution mode

The tool successfully:

- Detected SQL Injection vulnerability
- Identified backend DBMS
- Enumerated databases

- Enumerated tables
- Retrieved table contents

---

6. Results of Enumeration

6.1 Database Identified

The backend database management system was identified successfully.

Databases discovered included:

- information_schema
- performance_schema
- dvwa (application database)

6.2 Table Enumeration

Within the DVWA database, tables were enumerated successfully.

Primary table of interest:

users

6.3 Data Retrieval

Using automated dumping functionality, sqlmap retrieved:

- User IDs
- First names
- Last names
- Usernames
- Password hashes

Data was exported to local output files within the AttackBox environment.

---

7. Vulnerability Analysis

Type of Vulnerability:

SQL Injection (Error-based / Automated Enumeration)

Injection Vector:

GET parameter:

id

Root Cause:

- Unsanitized user input
- Direct concatenation of user input into SQL query
- No prepared statements
- No parameterized queries
- No input validation

Example of vulnerable backend logic:

```
SELECT first_name, last_name FROM users WHERE user_id = '$id';
```

Because input is not sanitized, malicious SQL can be appended to modify query logic.

---

8. Risk Assessment

Impact:

High

An attacker can:

- Extract all user credentials
- Access sensitive database information
- Enumerate schema structure
- Potentially escalate to further compromise

Likelihood:

High (Low security mode)

Overall Risk Level:

Critical (in production environments)

---

9. Security Implications

If this vulnerability existed in a real-world production system:

- User credentials could be compromised
- Password hashes could be cracked offline
- Data exfiltration could occur
- Regulatory violations could result

- Full database compromise is possible

---

10. Remediation Recommendations

To prevent SQL Injection:

1. Use Prepared Statements

Use parameterized queries instead of dynamic SQL.

2. Input Validation

Sanitize and validate all user input.

3. Least Privilege Principle

Database users should not have administrative privileges.

4. Error Handling

Disable verbose database error messages.

5. Web Application Firewall (WAF)

Deploy a WAF to detect injection attempts.

---

11. Key Learnings

- SQL Injection can be automated easily using tools like sqlmap.
- Authentication cookies are necessary when exploiting authenticated pages.
- Low security configurations expose full database access.
- Manual testing helps understand query behavior before automation.
- Proper input handling completely prevents this class of vulnerability.

---

12. Conclusion

The DVWA application hosted on TryHackMe was confirmed vulnerable to SQL Injection in low security mode.

By:

1. Extracting session cookies
2. Crafting authenticated request URLs
3. Using sqlmap for automation
4. Enumerating and dumping database contents

The entire database was successfully retrieved.