CYBER SECURITY PROJECT

SQL INJECTION

Team -Members --

Yadidya – 21BCE2302

Jada Avinash – 21BCE0555

Bethu Rohithash – 21BCE3693

Introduction:

SQL Injection (SQLi) is one of the most common and dangerous security vulnerabilities in web applications. It occurs when an attacker is able to manipulate SQL queries by injecting malicious code into an application's input fields (such as a username or password field). This allows the attacker to bypass authentication mechanisms, read sensitive data from the database, modify or delete data, and even execute administrative operations on the database. SQL injection is especially dangerous in systems that directly incorporate user input into SQL queries without proper validation or sanitization.

The purpose of this project is to demonstrate how SQL injection works in a simple login system and highlight the vulnerabilities that arise when secure coding practices are not followed. Understanding SQL injection helps developers recognize its risks and take necessary measures to protect their applications.

Detailed Description -

SQL injection occurs when user input is not properly sanitized before being included in a SQL query. This vulnerability allows an attacker to interfere with the query structure, enabling them to execute unintended commands on the database. The result can be catastrophic, leading to unauthorized access, data breaches, and loss of data integrity.

Here are the different types of SQL injection:

Classic SQL Injection: The attacker inserts or manipulates data directly into SQL queries through user inputs.

Blind SQL Injection: In this case, the attacker cannot see the database's response, but they can still infer information based on the application's behavior (e.g., error messages, delays, etc.).

Error-Based SQL Injection: The attacker causes an error in the SQL query to gather information about the database structure.

Union-Based SQL Injection: The attacker uses the UNION operator to combine the results of the original query with malicious SQL queries.

Time-Based Blind SQL Injection: The attacker sends a query that causes the server to wait for a specific amount of time, allowing them to infer information about the database structure.

The key principle behind SQL injection is that the attacker exploits an application's failure to validate or sanitize user input. This can allow them to manipulate or execute SQL commands that the application was not intended to run.

Example of SQL Injection

Consider a login form where a user enters their username and password. If the application builds the following query without validation:

Sql:

SELECT * FROM users WHERE username = 'admin' AND password = 'password';

An attacker might try entering the following into the username or password field:

Bash:

admin' OR '1'='1

The query would then become:

Sql:
SELECT * FROM users WHERE username = 'admin' AND password = '' OR '1'='1';

Since '1'='1' is always true, the query returns a valid result and allows the attacker to bypass the authentication process.

Impact of SQL Injection

Authentication Bypass: Attackers can log in as any user, including administrators, without needing valid credentials.

Data Exfiltration: Attackers can retrieve sensitive data, such as user credentials or financial information.

Data Modification: Attackers can alter or delete data, disrupting business operations.

Privilege Escalation: Attackers can gain higher privileges on the database, performing administrative operations.

Description of the Tools Used in the Project-

1. SQLmap

SQLmap is an open-source penetration testing tool specifically designed to automate the process of detecting and exploiting SQL injection vulnerabilities in web applications. It supports a wide range of SQL injection techniques, such as error-based, time-based, and blind SQL injection, and can test for various database types (e.g., MySQL, PostgreSQL, SQLite, etc.). SQLmap is widely used by security professionals to identify and exploit SQL injection flaws in a safe and controlled environment.

Key features of SQLmap:

Automated SQL Injection Detection: It scans web applications to identify possible SQL injection points.

Exploitation: SQLmap can automate the process of exploiting detected SQL injection vulnerabilities, including retrieving database schema, tables, and data.

Database Support: Supports multiple database types (e.g., MySQL, Oracle, MSSQL, PostgreSQL, etc.).

Customization: SQLmap allows users to customize and fine-tune the attack parameters (e.g., delay, payload types, or attack vectors).

SQLmap can be used to test the vulnerability of your web application to SQL injection by automating the detection and exploitation of such flaws, ensuring a thorough security analysis.

2. Flask

Flask is a lightweight web framework for Python that allows developers to build web applications quickly and with minimal setup. Flask is widely used for creating small to medium-scale web applications and APIs. It is known for its simplicity and flexibility, making it ideal for both beginners and advanced users.

Key features of Flask:

Minimalistic: Flask provides a minimalistic, easy-to-understand foundation for web development, which is highly extendable.

Routing and Templates: Flask supports URL routing and HTML templating with tools like Jinja2, which allows easy rendering of dynamic web pages.

Development Server: It includes a built-in server for quick development and testing, making it easy to start coding immediately.

Integration with Databases: Flask can be easily integrated with databases (like SQLite, MySQL, PostgreSQL) using ORM libraries such as SQLAlchemy or with native libraries like sqlite3.

Flask was used in your project to build the basic web application, handle user input, and interact with the database, demonstrating the vulnerability of SQL injection.

3. JSQL Injection

JSQL Injection is a tool used for exploiting SQL injection vulnerabilities in web applications. It is a Java-based SQL injection penetration testing tool that automates the process of detecting and exploiting SQL injection flaws. JSQL Injection can be used for a variety of attacks, such as retrieving data from a database or performing SQL-based denial-of-service attacks.

Key features of JSQL Injection:

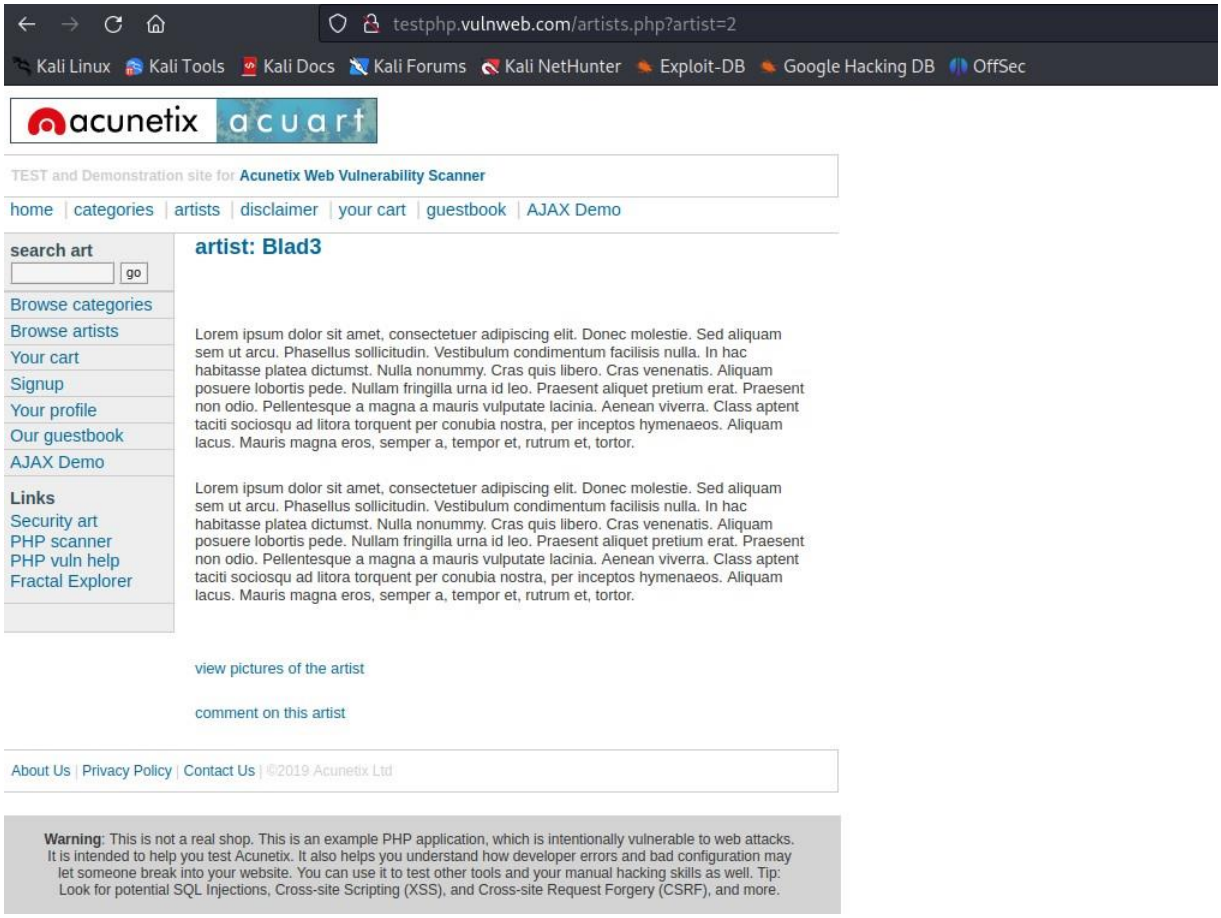
SQL Injection Exploitation: Automates the process of exploiting SQL injection vulnerabilities and can retrieve valuable information from vulnerable web applications.

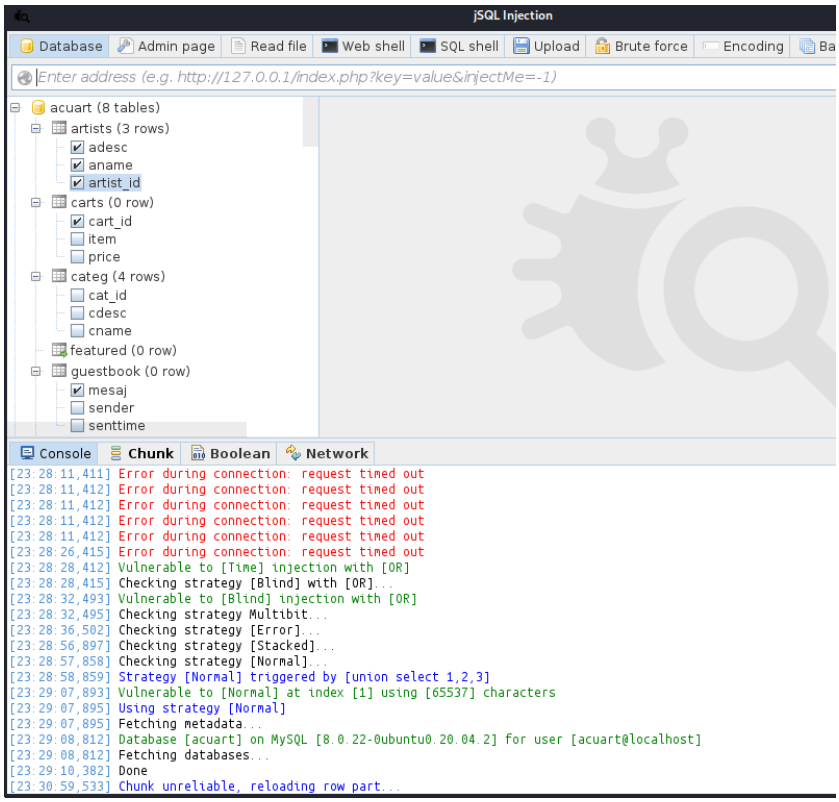
Blind SQL Injection: Supports blind SQL injection techniques, which are useful when error messages are not displayed.

Database Enumeration: JSQL Injection can help identify the structure of a database, including tables and columns, even if the error messages are hidden.

Cross-platform: Being Java-based, JSQL Injection can run on multiple operating systems (Windows, macOS, Linux) without requiring specific configurations.

21BCE2302
YADIDYA

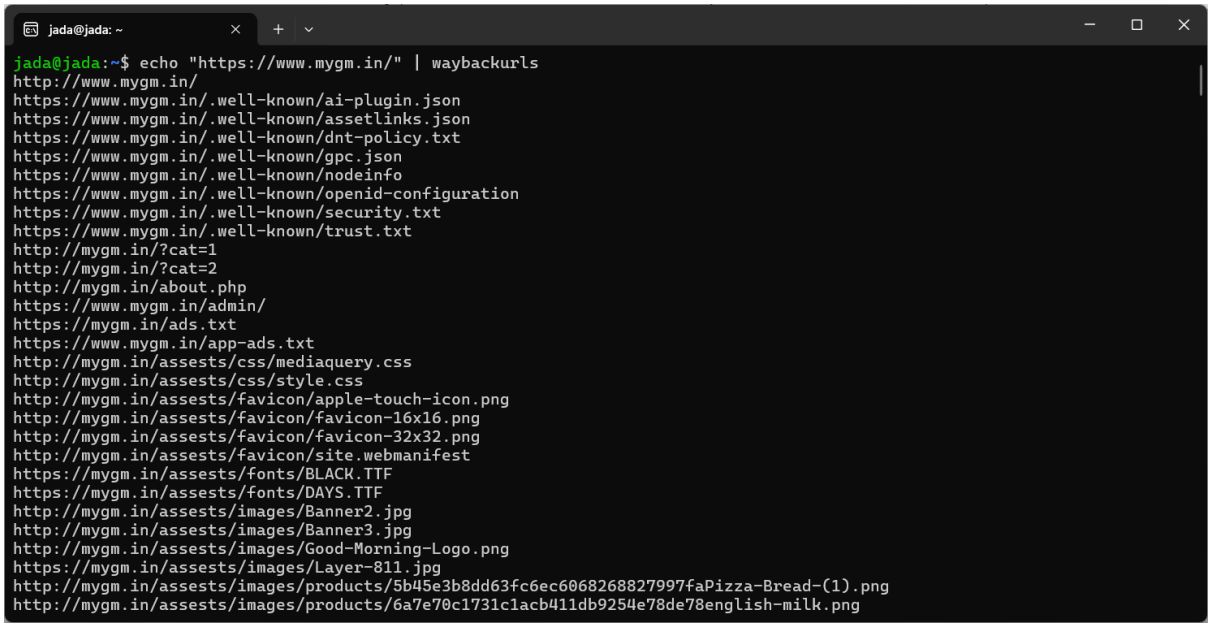



jSQL Injection
Database
Admin page
Read file
Web shell
SQL shell
Upload
Brute force
Encoding
Enter address (e.g. http://127.0.0.1/index.php?key=value&injectMe=-1)
acuart (8 tables)
artists (3 rows)
adesc
aname
artist_id
carts (0 row)
cart_id
item
price
categ (4 rows)
cat_id
cdesc
cname
featured (0 row)
guestbook (0 row)
mesaj
sender
senttime
Console
Chunk
Boolean
Network
[23:28:11,411] Error during connection: request timed out
[23:28:11,412] Error during connection: request timed out
[23:28:11,412] Error during connection: request timed out
[23:28:11,412] Error during connection: request timed out
[23:28:11,412] Error during connection: request timed out
[23:28:26,415] Error during connection: request timed out
[23:28:28,412] Vulnerable to [Time] injection with [OR]
[23:28:28,415] Checking strategy [Blind] with [OR]...
[23:28:32,493] Vulnerable to [Blind] injection with [OR]
[23:28:32,495] Checking strategy Multibit...
[23:28:36,502] Checking strategy [Error]...
[23:28:56,897] Checking strategy [Stacked]...
[23:28:57,858] Checking strategy [Normal]...
[23:28:58,859] Strategy [Normal] triggered by [union select 1,2,3]
[23:29:07,893] Vulnerable to [Normal] at index [1] using [65537] characters
[23:29:07,895] Using strategy [Normal]
[23:29:07,895] Fetching metadata...
[23:29:08,812] Database [acuart] on MySQL [8.0.22-0ubuntu0.20.04.2] for user [acuart@localhost]
[23:29:08,812] Fetching databases...
[23:29:10,382] Done
[23:30:59,533] Chunk unreliable, reloading row part...

21BCE0555
JADA AVINASH

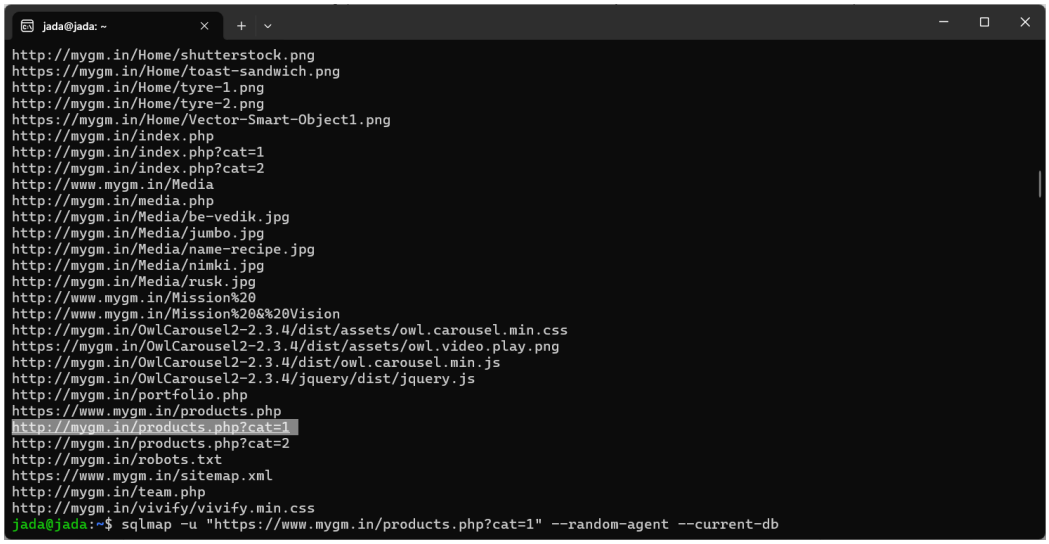
Step 1: URL Enumeration (Waybackurls)
The first screenshot shows the use of the waybackurls command to gather a list of URLs for the target domain (mygm.in). This tool retrieves URLs archived by services like the Wayback Machine. The output shows a variety of URLs, including:

- Paths to administrative pages (e.g., /admin/)
- Asset files (e.g., .css, .js, images)
- Potential API endpoints or PHP scripts (products.php)



Step 2: Identifying Vulnerable Parameters
The next image highlights a **specific URL**, https://mygm.in/products.php?cat=1, which is a dynamic parameter (cat=1). This parameter is selected as a candidate for SQL injection testing using SQLmap. The command:
sqlmap -u "https://mygm.in/products.php?cat=1" --random-agent --current-db

- **-u** specifies the target URL.
- **--random-agent** uses a random user-agent to evade detection.
- **--current-db** aims to identify the current database in use.

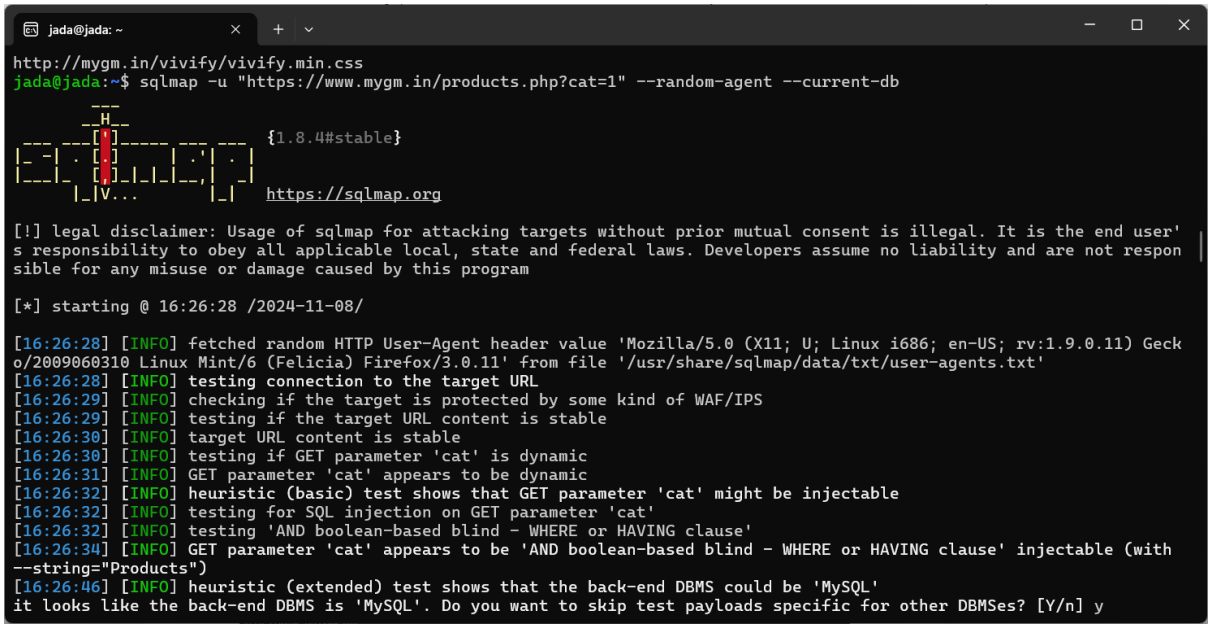


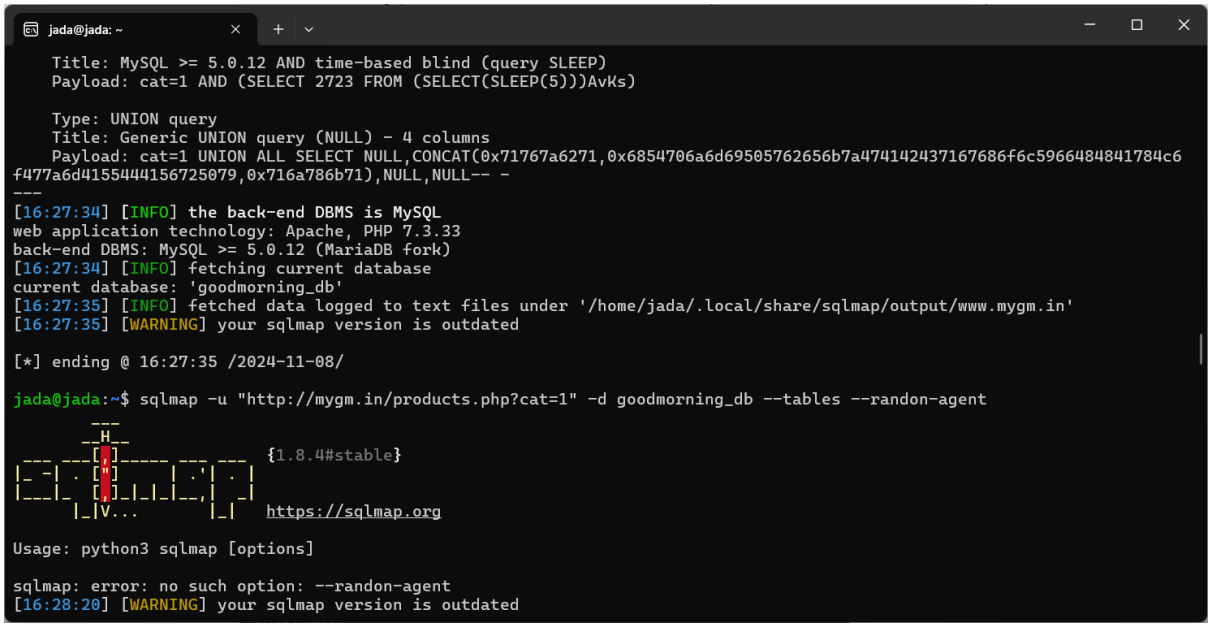
Step 3: SQL Injection Testing

SQLmap begins testing the parameter cat for SQL injection vulnerabilities. The output indicates:

- The parameter is **dynamic and injectable**.
- The backend database is **MySQL**.
- A specific type of injection (boolean-based blind) is possible.

After confirming the vulnerability, SQLmap proceeds to exploit it to retrieve database information.





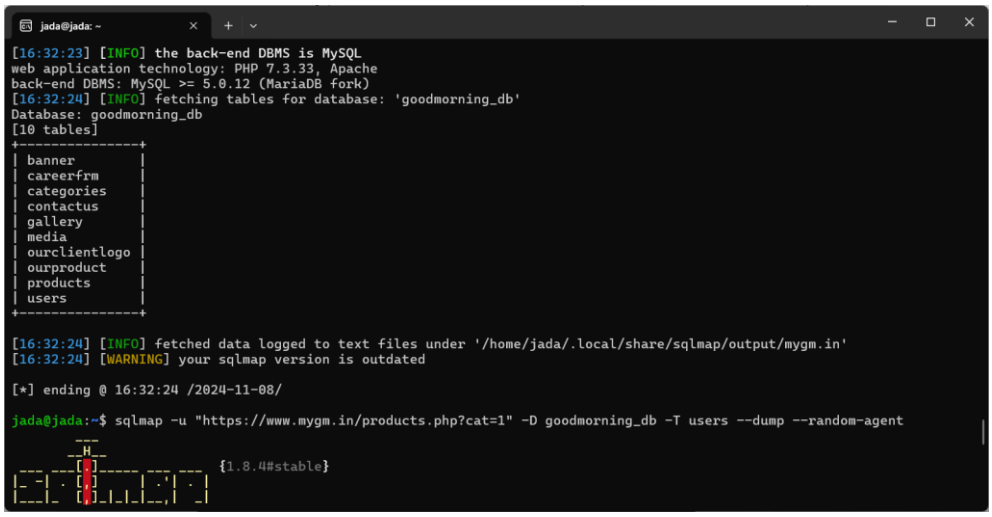
Step 4: Fetching Database Information
SQLmap successfully retrieves the **current database name**, which is goodmorning_db.

Step 5: Enumerating Tables in the Database
The next command:
sqlmap -u "https://mygm.in/products.php?cat=1" -D goodmorning_db --tables --random-agent

- Specifies the database (goodmorning_db) using -D.
- Lists all tables in the database (--tables).

The output reveals several tables, including:

- users (potentially containing sensitive information)
- Other tables like banner, categories, media.



Step 6: Extracting Data from the users Table

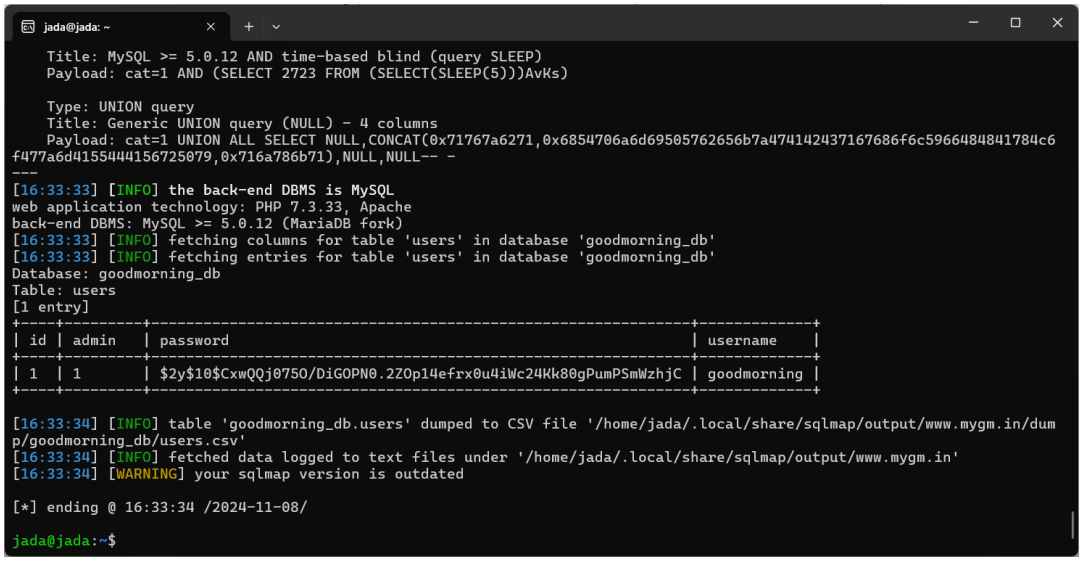
The final command:
sqlmap -u "https://mygm.in/products.php?cat=1" -D goodmorning_db -T users --dump --random-agent

- Specifies the users table with -T users.
- Dumps its contents (--dump).

SQLmap extracts:

- A single user entry (admin).
- The password hash for the admin user.

The dumped data is saved to a CSV file, which can be further analyzed.



BETHU ROHITHASH

21BCE3693

My Work on SQL Injection –

- Login System Setup: The project includes a simple webpage where users can enter a username and password. The backend code, written in Python, checks these credentials against records stored in an SQLite database.
- Demonstrating SQL Injection: The project intentionally constructs SQL queries in a way that leaves them open to injection attacks. This vulnerability allows a user to bypass authentication with an SQL injection payload, such as admin' OR '1'='1. By injecting this statement into the username field, the application returns a successful login without needing a valid password.
- Educational Purpose: This project is a practical demonstration of how SQL injection can compromise login systems, serving as an example of why parameterized queries and secure coding practices are essential in real-world applications.

Breakdown of the Injection -

--When you enter a username like admin' OR '1'='1 in this project, it takes advantage of the way SQL queries are constructed without sanitization. Let's break down how and why this works.

Original SQL Query: In your code, the query is constructed using an f-string, embedding the username and password directly:

Sql:

```
SELECT * FROM users WHERE username = '{username}' AND password = '{password}'
```

When you input admin as the username and password as password, it becomes:

Sql:

```
SELECT * FROM users WHERE username = 'admin' AND password = 'password'
```

This query checks for a user with exactly that username and password.

Injecting SQL with admin' OR '1'='1:

If you enter admin' OR '1'='1 in the username field and anything in the password field, the query will look like this:

Sql:

SELECT * FROM users WHERE username = 'admin' OR '1'='1' AND password = 'any_password'

Here, the username part is manipulated to include an OR condition that always returns true ('1'='1'), so the database will ignore the AND password = 'any_password' part.

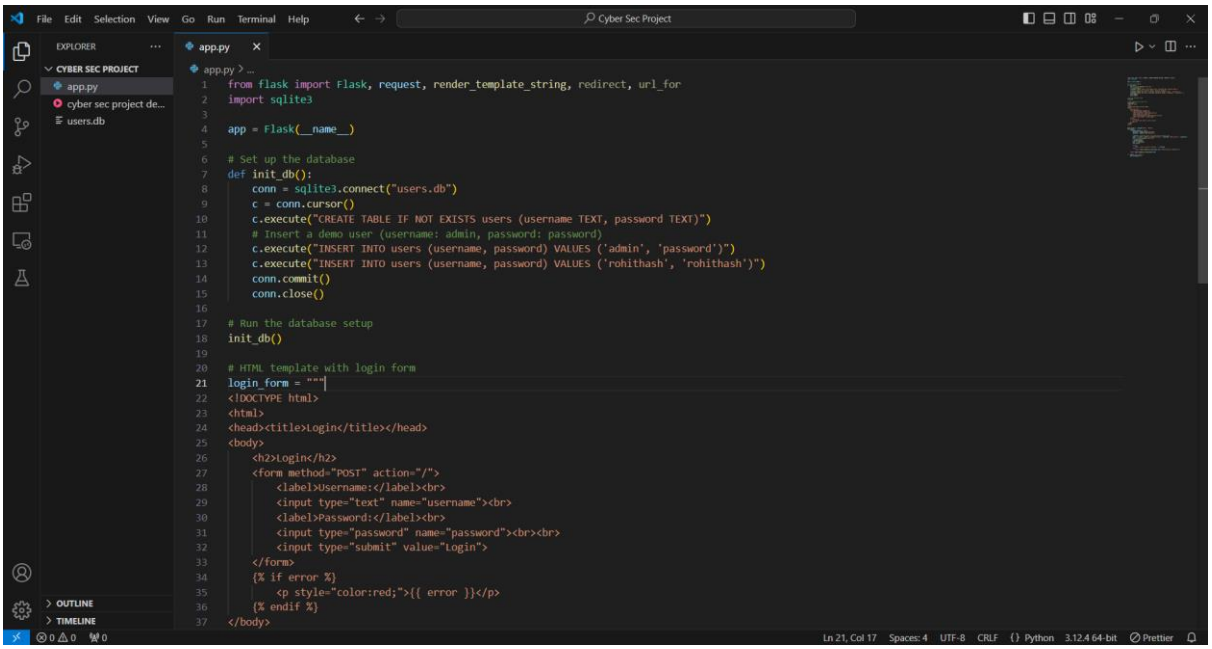
Effect of OR '1'='1':

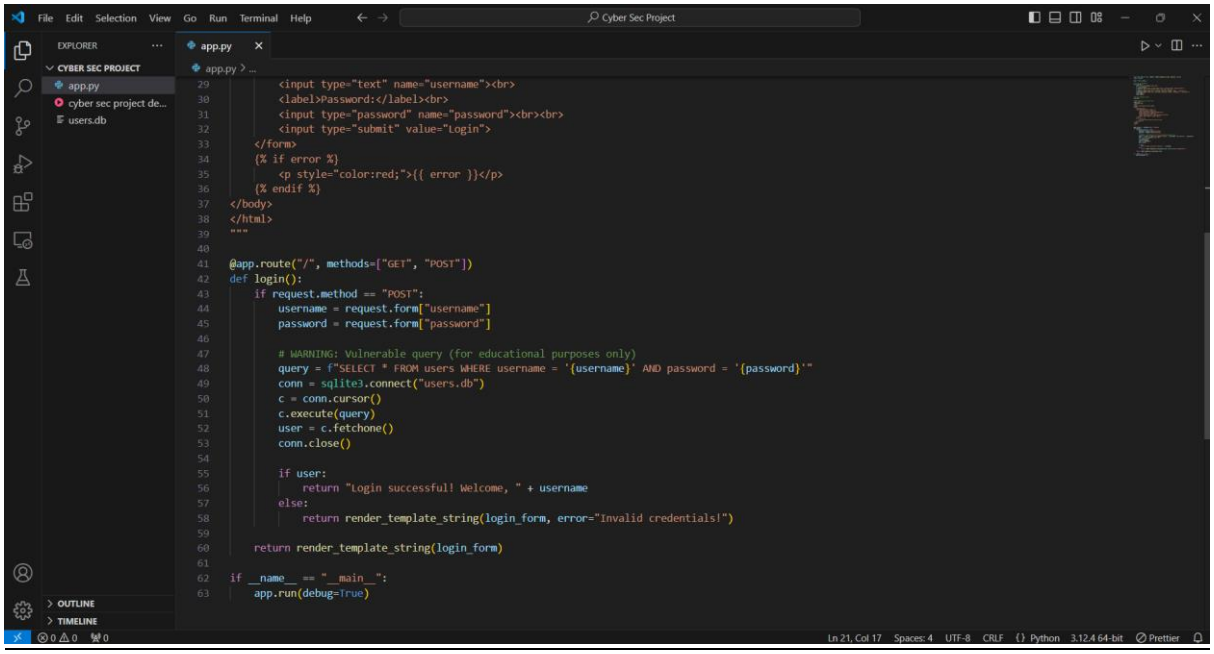
The OR '1'='1' clause turns the entire condition into true regardless of the password value. This trick allows the SQL query to find any row in the users table where the username is admin or where 1=1 (which is always true).

Consequently, it returns a result as if the login is successful, allowing you to bypass the password requirement.

Step-By-Step Demonstration –

- Code and Running Code-





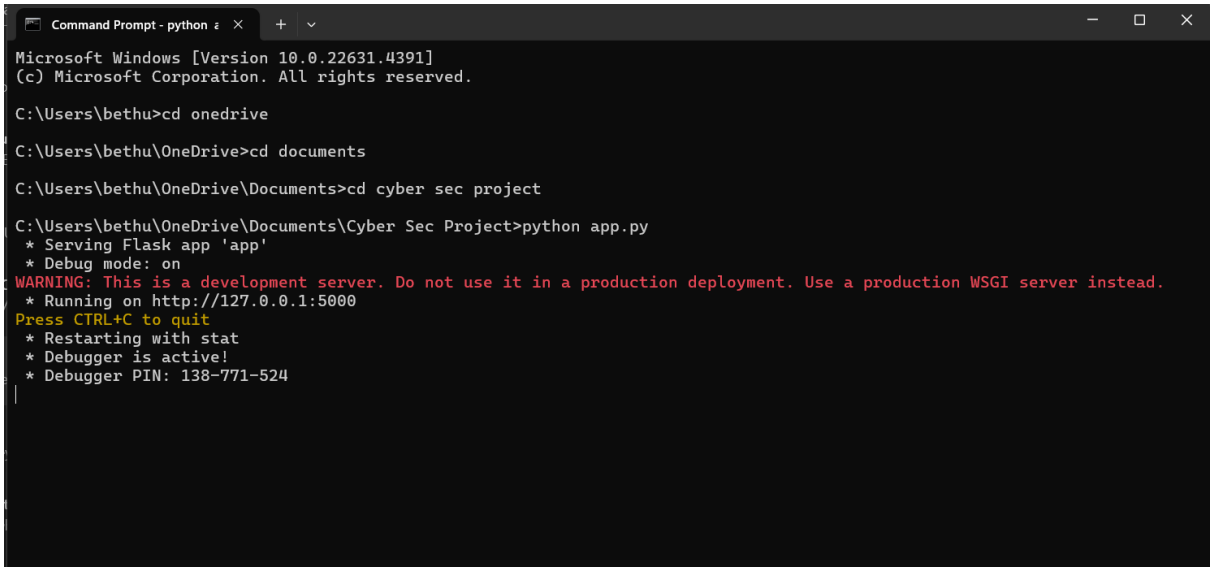
Steps for Running this Code-

1.Download Python from Chrome.

2.Download Flask in cmd by pip install flask.

3.Run this code in cmd python filename.py

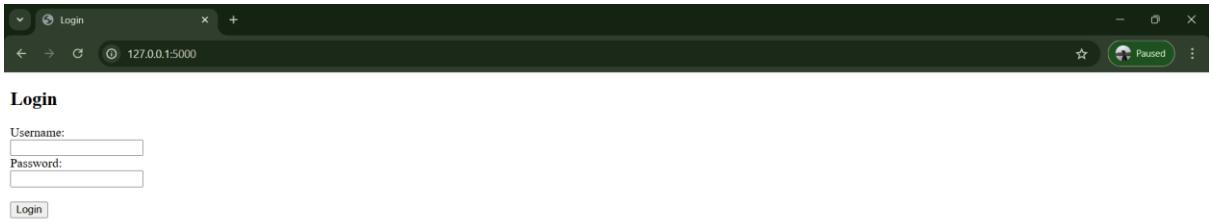
4.The terminal will show a URL (like http://127.0.0.1:5000/) where the login page is hosted.



Step 2 – Went to Login Page

1.Open a web browser and go to the URL provided by Flask (usually http://127.0.0.1:5000/).

2.You'll see a simple login form with fields for Username and Password.



Step 3 - Checking with Different Credentials –

1.Enter a **valid username and password**, such as admin and password, then click **Login**. The system should log in successfully and show "Login successful! Welcome, admin."

2.Enter an **incorrect password** for a valid username, like admin with the password wrongpass, and click **Login**. It should display "Invalid credentials!"

--First we will use Rohithash Rohithash as password and username

--Second we will use invalid credentials and check

--Then we will use sql injection to breach websites.

Login

Username:
rohithash
Password:
••••••••••

Login

Login successful! Welcome, rohithash

Login

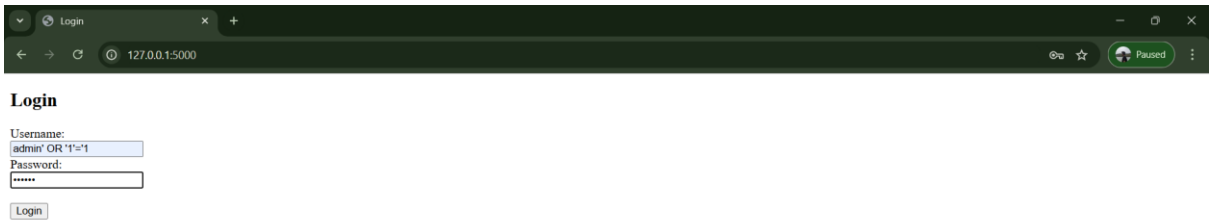
Username:
Tom Holland
Password:
•••••

Login

Step4 – Finally accessing website using SQL Injection-

1.In the **username** field, enter the SQL injection input admin' OR '1'='1 and type anything in the password field (e.g., anything).

2.Click **Login**. Despite the incorrect password, the system should log in successfully because the SQL injection bypasses the password check.



Conclusion-

In conclusion, this project effectively demonstrates how SQL injection vulnerabilities can compromise a web application's authentication mechanism. By constructing a basic login system with an intentionally unsafe query, you showcase how a malicious user can exploit this flaw to gain unauthorized access. This highlights the importance of secure coding practices, such as parameterized queries, input validation, and the use of hashed passwords, which are critical to protecting applications from such attacks.

This project serves as a valuable educational tool for understanding the risks associated with SQL injection and emphasizes the need for developers to adopt robust security measures to safeguard user data and application integrity.

GITHUB LINK - https://github.com/BethuRohithash/Cyber-Security-JSQL-SQLMAP-Implementation

Resources -

https://github.com/sqlmapproject/sqlmap/wiki
https://github.com/ron190/jsql-injection
https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html
https://www.youtube.com/watch?v=QvG6cNc2bA4
SQL Injection: A Practical Guide Book