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.

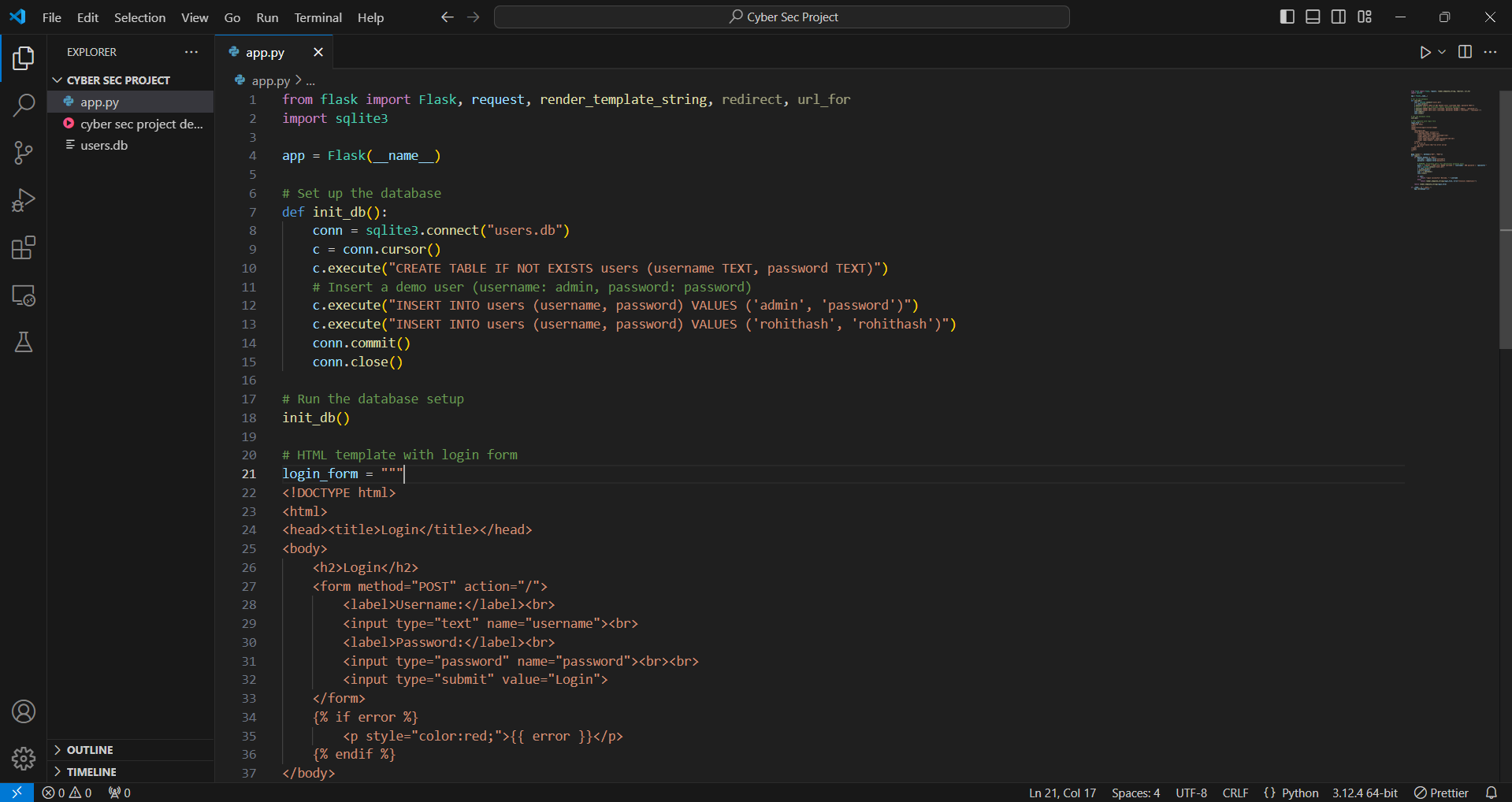
Effect of OR '1'='1':

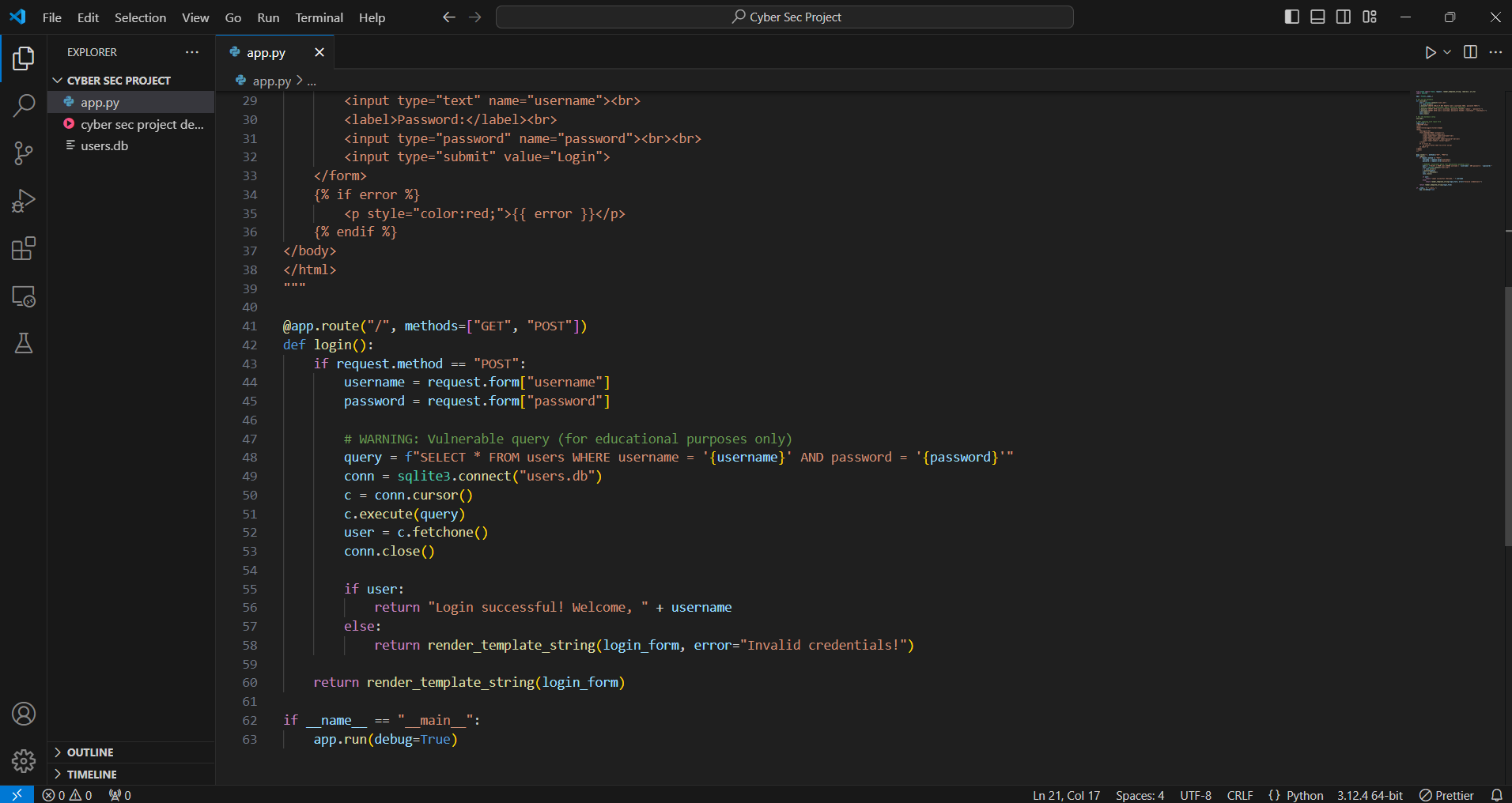
The OR '1'='1' clause turns the entire condition into true regardless of the password value. This trick allows the SQL query to find any row in the users table where the username is admin or where 1=1 (which is always true).

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-





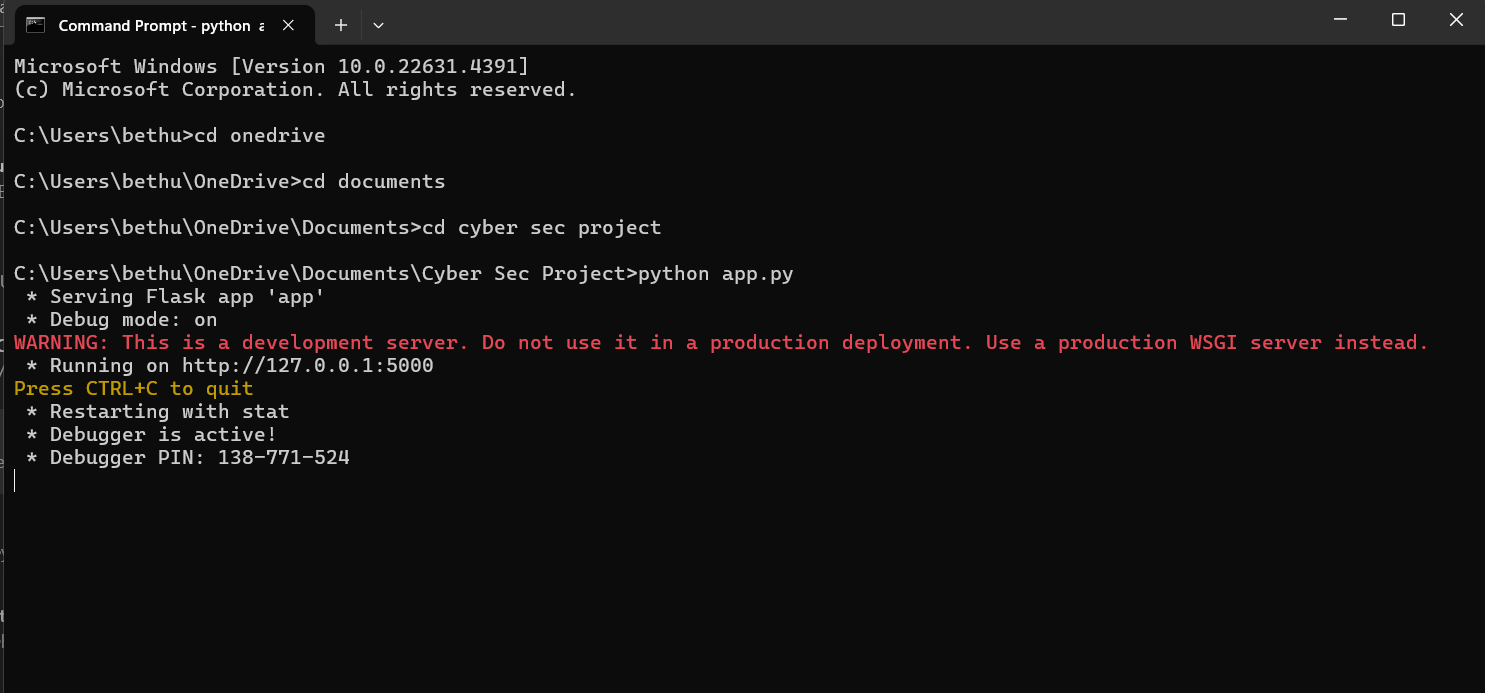
Steps for Running this Code-

1.Download Python from Chrome.

2.Download Flask in cmd by pip install flask.

3.Run this code in cmd python filename.py

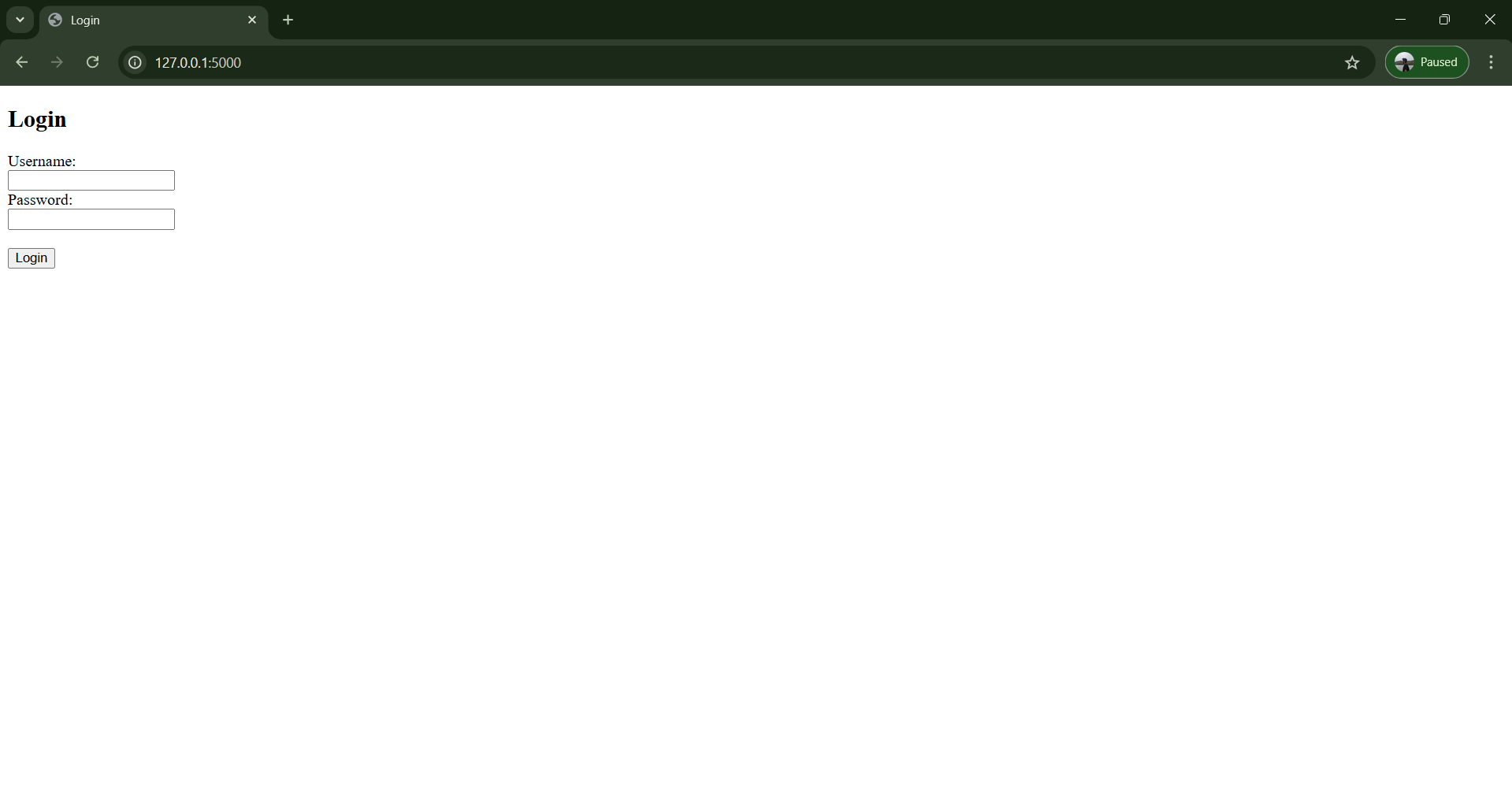
4.The terminal will show a URL (like http://127.0.0.1:5000/) where the login page is hosted.



**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 –**

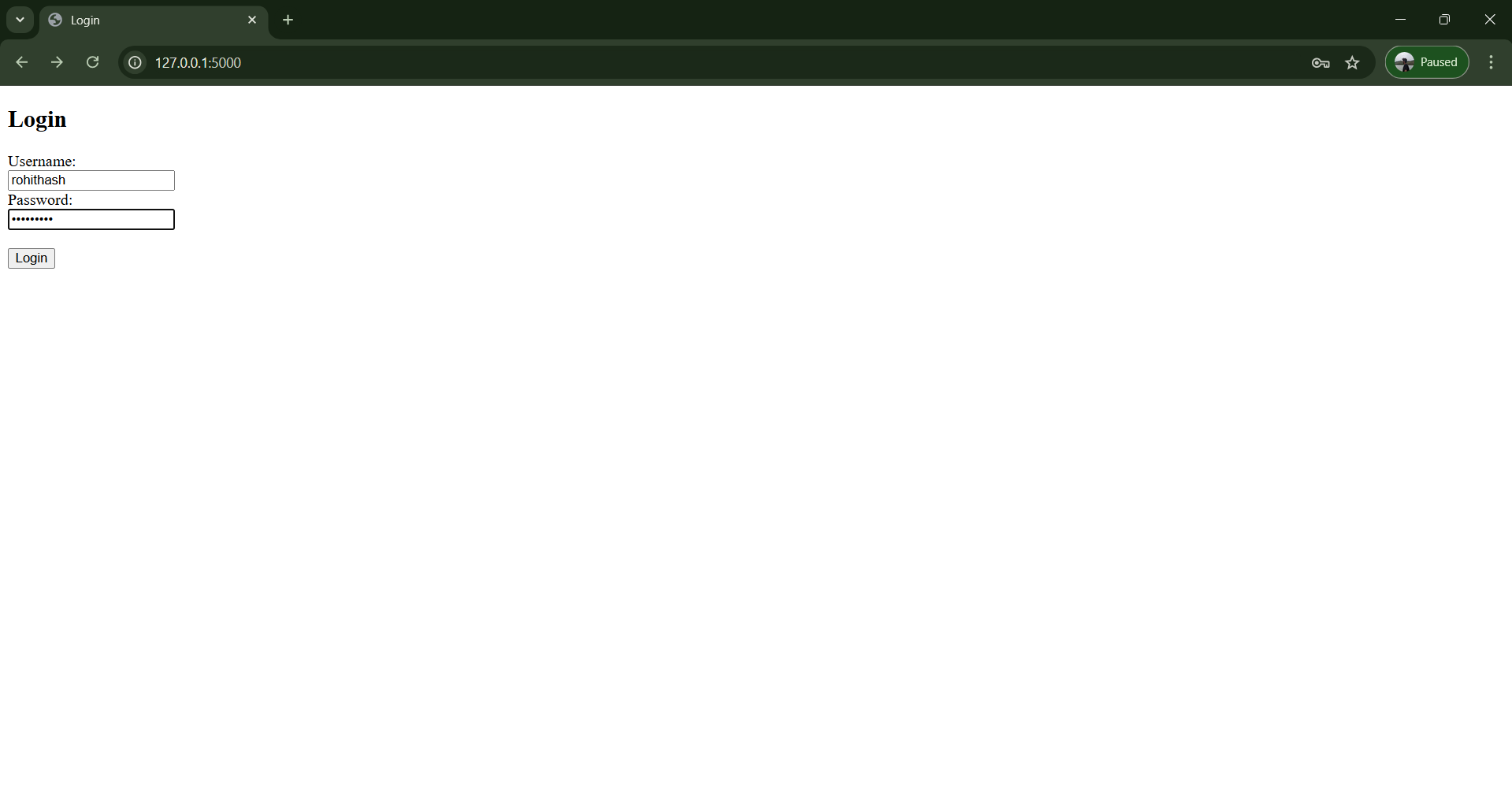
1.Enter a **valid username and password**, such as admin and password, then click **Login**. The system should log in successfully and show "Login successful! Welcome, admin."

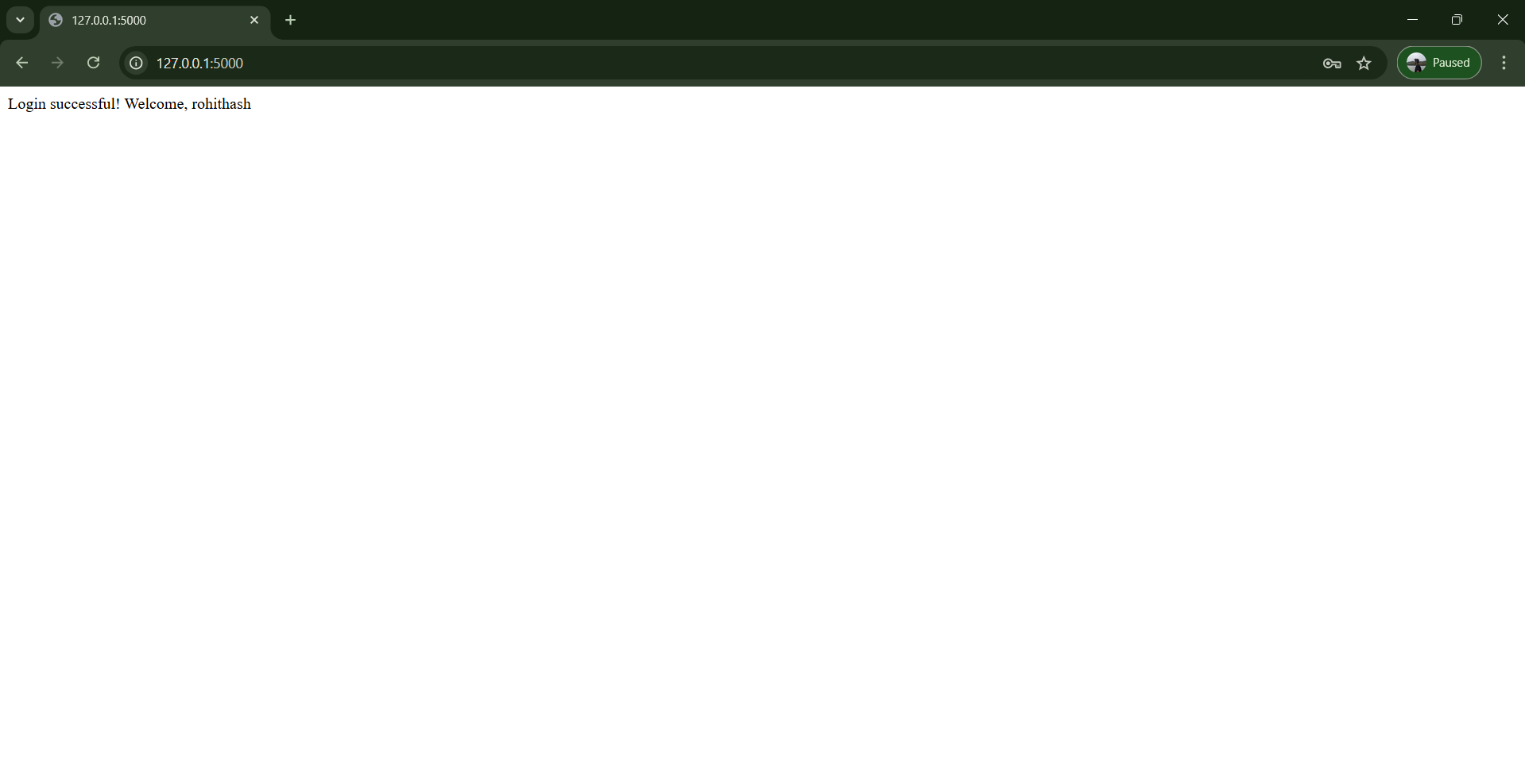
2.Enter an **incorrect password** for a valid username, like admin with the password wrongpass, and click **Login**. It should display "Invalid credentials!"

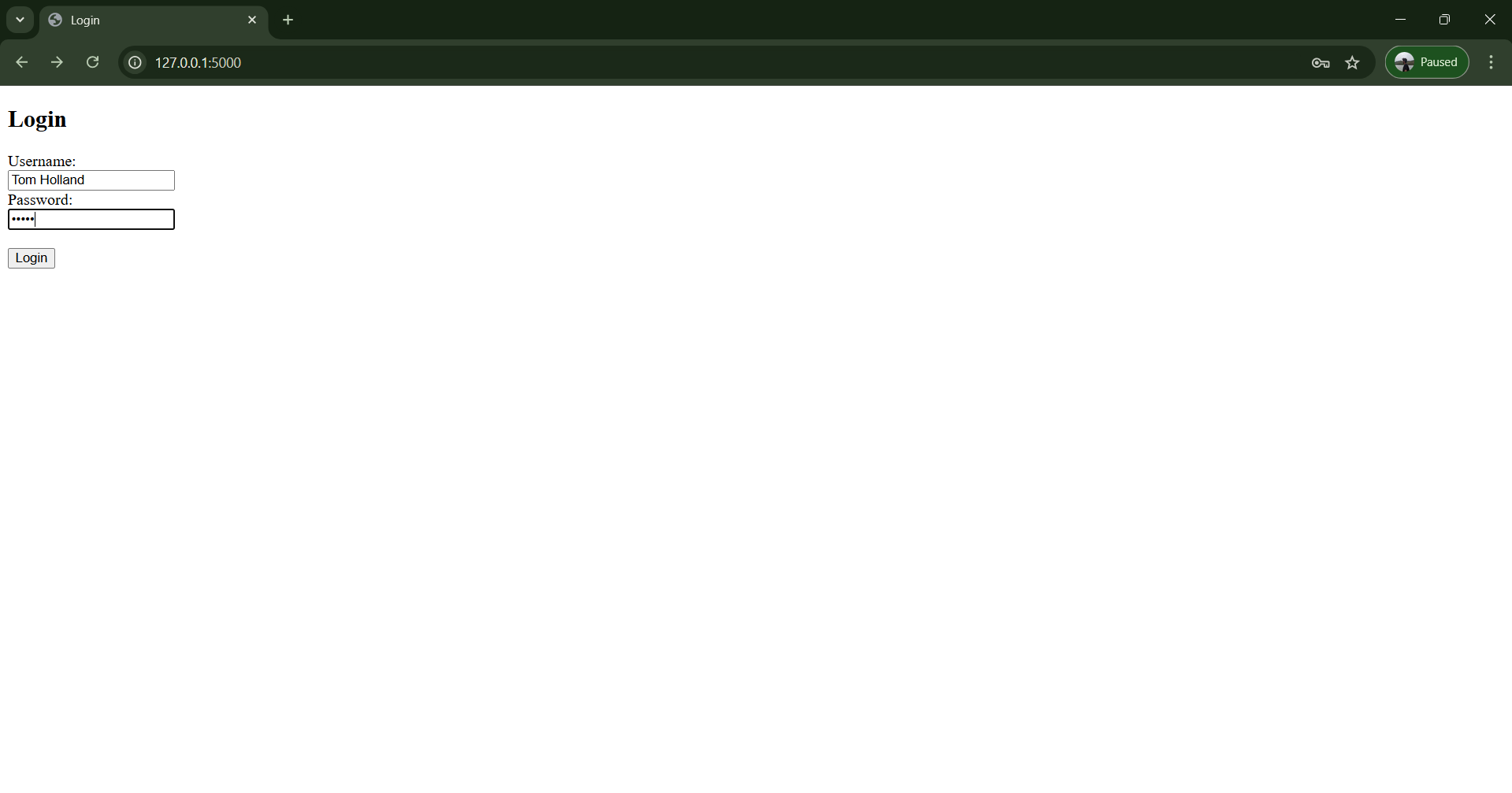
--First we will use Rohithash Rohithash as password and username

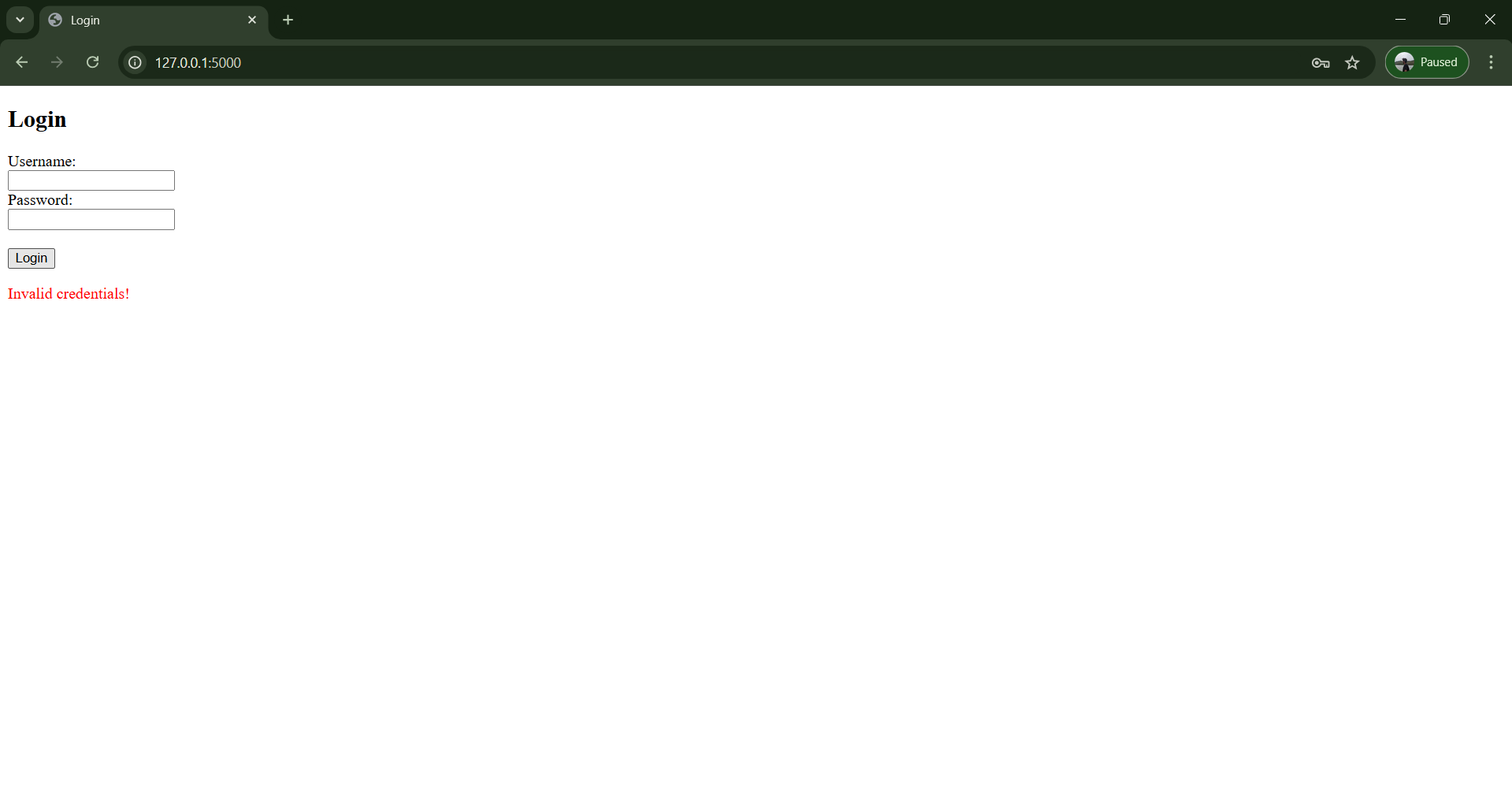
--Second we will use invalid credentials and check

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





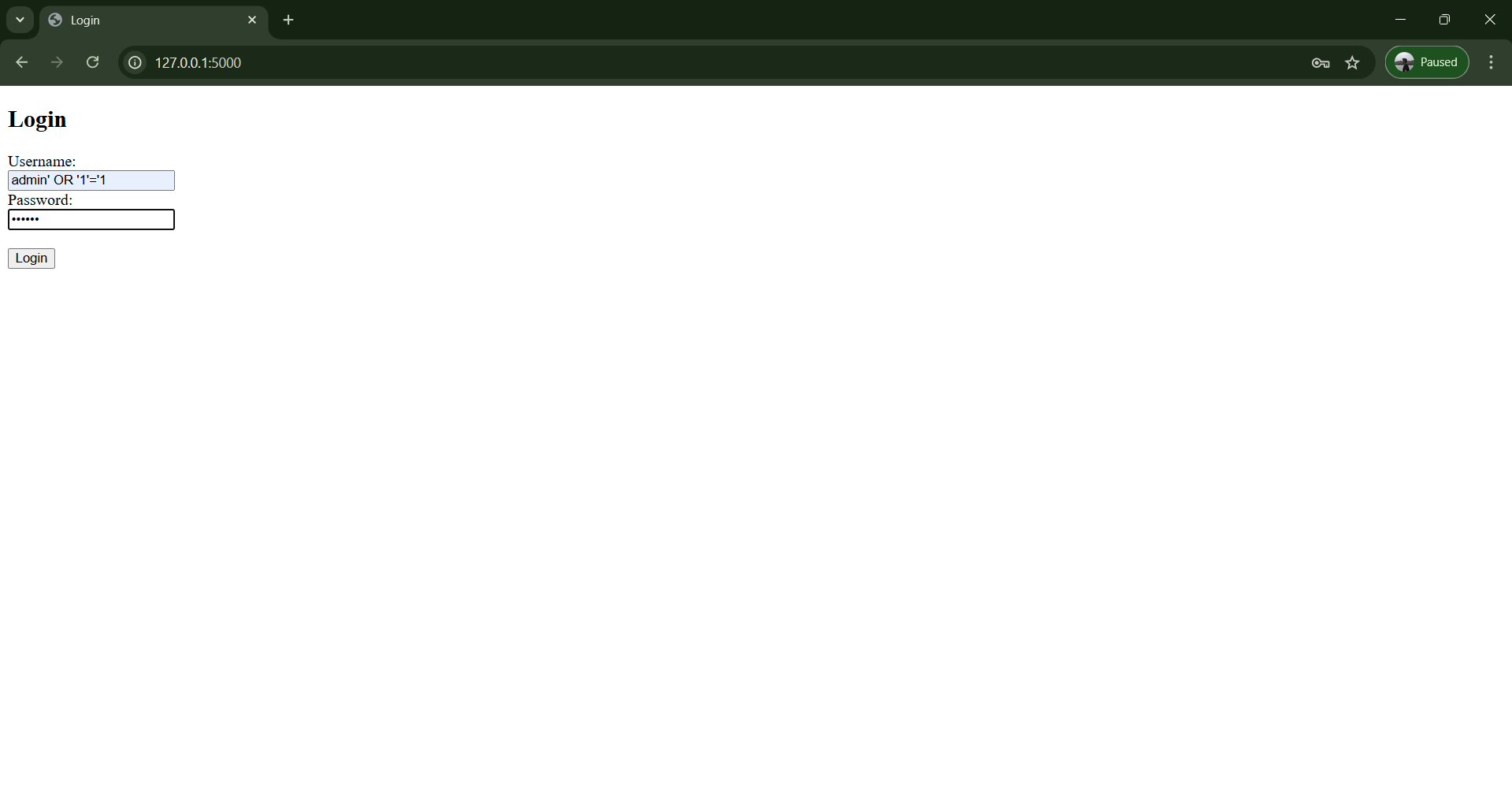


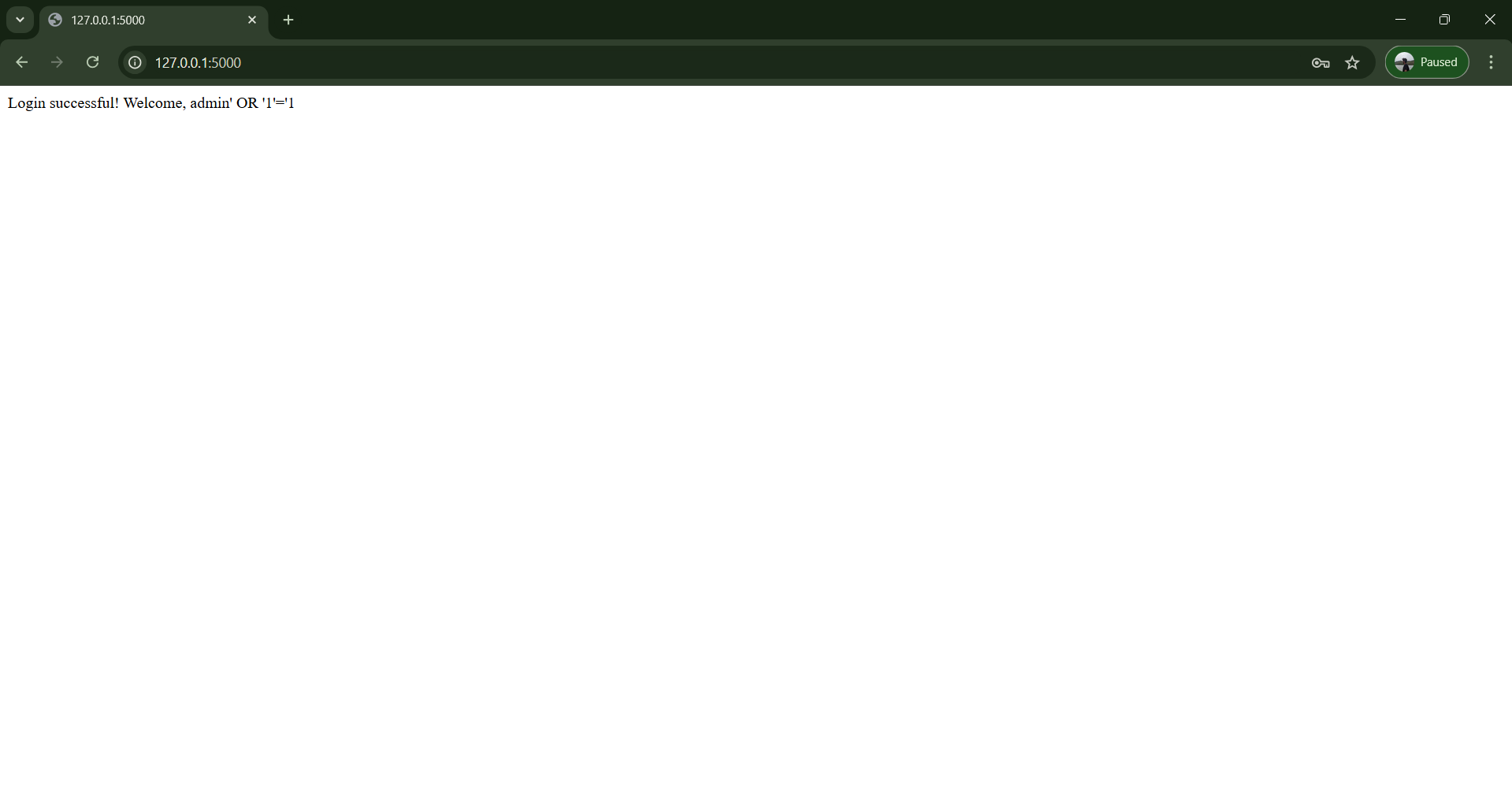


Step4 – Finally accessing website using SQL Injection-

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