SQL Injection Demonstration

1. Introduction

SQL Injection is one of the most common and dangerous web application vulnerabilities. It occurs when user input is improperly sanitized, allowing attackers to manipulate SQL queries and gain unauthorized access to data.

This project demonstrates both:

- A vulnerable login system that can be exploited via SQL injection
- A secure login system using parameterized queries to prevent injection

2. Objective

- Educate developers about SQL injection threats
- Show how easy it is to exploit poorly written queries
- Demonstrate the correct, secure way to query databases
- Help build awareness around safe web coding practices

3. Tools & Technologies Used

• Language: Python 3

Web Framework: Flask

• Database: SQLite

• Frontend: HTML, CSS

• IDE: VS Code / PyCharm / Replit

• Testing: Browser-based with manual SQL injection attempts

4. Steps Involved in Building the Project

Step 1: Environment Setup

- Installed Flask and SQLite
- Created virtual environment (optional but recommended)

Step 2: Database Initialization

- Created a SQLite database with a users table
- Inserted a default test user: admin / admin123

Step 3: Flask Application

- Developed app.py with two login functions:
 - vulnerable_login() using unsafe string interpolation
 - secure login() using parameterized queries

Step 4: Toggle Feature

- Added a global flag SECURE MODE to switch between modes
- Displayed the current mode (vulnerable/secure) on the web UI

Step 5: Web Pages

- index.html: Login form with mode status
- dashboard.html: Success page post-login
- alert.html: Error page for failed login
- style.css: Basic styling

§ 5. Key Features

- Simple and clean UI for demonstration
- Pulnerable login path (for demonstration only)
- Secure login path using safe practices
- Awareness material in the code and README

6. How SQL Injection Works in This Project

Vulnerable Mode (SECURE_MODE = False)

- Input: 'OR '1'='1
- Exploits query like:

SELECT * FROM users WHERE username = "OR '1'='1' AND password = 'anything'

• Bypasses authentication and logs in as the first user in the table.

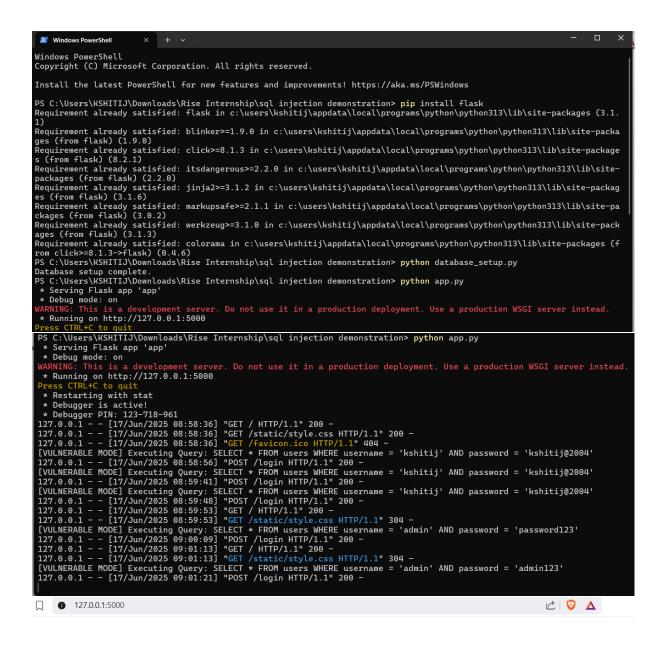
Secure Mode (SECURE_MODE = True)

• Input is safely handled:

cursor.execute("SELECT * FROM users WHERE username = ? AND password = ?", (username, password))

• SQL Injection fails as input is treated as data, not executable code.

7. Screenshots



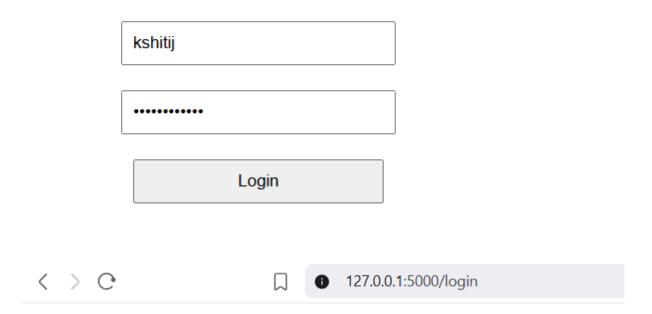
Login Form

VULNERABLE MODE: Unsafe String Interpolation

Username		
Password		_
	Login	

Login Form

VULNERABLE MODE: Unsafe String Interpolation



Login Failed

Incorrect username or password.



Welcome, admin!

This is a protected area.

8. Recommendations for Security

- Always use parameterized queries or ORM
- Validate and sanitize all user input
- Avoid displaying raw SQL/database errors
- Use HTTPS and implement secure headers
- Apply user input whitelisting where applicable

9. Conclusion

This project provides a safe environment to experiment with and understand SQL Injection. By demonstrating both insecure and secure login mechanisms, it builds awareness and helps developers adopt secure practices in their own applications.

It also enhances your skills in:

- Python Flask development
- Web application security
- Secure database interactions
- Real-world attack simulation