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
- 🔓 Vulnerable login path (for demonstration only)
- 🔒 Secure login path using safe practices
- 📖 Awareness material in the code and README
- 🎯 Easy toggle switch via one variable in app.py

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

```
Windows PowerShell
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\KSHITIJ\Downloads\Rise Internship\sql injection demonstration> pip install flask
Requirement already satisfied: flask in c:\users\kshitij\appdata\local\programs\python\python313\lib\site-packages (3.1.
1)
Requirement already satisfied: blinker>=1.9.0 in c:\users\kshitij\appdata\local\programs\python\python313\lib\site-packa
ges (from flask) (1.9.0)
Requirement already satisfied: click>=8.1.3 in c:\users\kshitij\appdata\local\programs\python\python313\lib\site-package
s (from flask) (8.2.1)
Requirement already satisfied: itsdangerous>=2.2.0 in c:\users\kshitij\appdata\local\programs\python\python313\lib\site-
packages (from flask) (2.2.0)
Requirement already satisfied: jinja2>=3.1.2 in c:\users\kshitij\appdata\local\programs\python\python313\lib\site-packag
es (from flask) (3.1.6)
Requirement already satisfied: markupsafe>=2.1.1 in c:\users\kshitij\appdata\local\programs\python\python313\lib\site-pa
ckages (from flask) (3.0.2)
Requirement already satisfied: werkzeug>=3.1.0 in c:\users\kshitij\appdata\local\programs\python\python313\lib\site-pack
ages (from flask) (3.1.3)
Requirement already satisfied: colorama in c:\users\kshitij\appdata\local\programs\python\python313\lib\site-packages (f
rom click>=8.1.3->flask) (0.4.6)
PS C:\Users\KSHITIJ\Downloads\Rise Internship\sql injection demonstration> python database_setup.py
Database setup complete.
PS C:\Users\KSHITIJ\Downloads\Rise Internship\sql injection demonstration> python app.py
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
```

```
PS C:\Users\KSHITIJ\Downloads\Rise Internship\sql injection demonstration> python app.py
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 123-718-961
127.0.0.1 - - [17/Jun/2025 08:58:36] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [17/Jun/2025 08:58:36] "GET /static/style.css HTTP/1.1" 200 -
127.0.0.1 - - [17/Jun/2025 08:58:36] "GET /favicon.ico HTTP/1.1" 404 -
[VULNERABLE MODE] Executing Query: SELECT * FROM users WHERE username = 'kshitij' AND password = 'kshitij@2004'
127.0.0.1 - - [17/Jun/2025 08:58:56] "POST /login HTTP/1.1" 200 -
[VULNERABLE MODE] Executing Query: SELECT * FROM users WHERE username = 'kshitij' AND password = 'kshitij@2004'
127.0.0.1 - - [17/Jun/2025 08:59:41] "POST /login HTTP/1.1" 200 -
[VULNERABLE MODE] Executing Query: SELECT * FROM users WHERE username = 'kshitij' AND password = 'kshitij@2004'
127.0.0.1 - - [17/Jun/2025 08:59:48] "POST /login HTTP/1.1" 200 -
127.0.0.1 - - [17/Jun/2025 08:59:53] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [17/Jun/2025 08:59:53] "GET /static/style.css HTTP/1.1" 304 -
[VULNERABLE MODE] Executing Query: SELECT * FROM users WHERE username = 'admin' AND password = 'password123'
127.0.0.1 - - [17/Jun/2025 09:00:09] "POST /login HTTP/1.1" 200 -
127.0.0.1 - - [17/Jun/2025 09:01:13] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [17/Jun/2025 09:01:13] "GET /static/style.css HTTP/1.1" 304 -
[VULNERABLE MODE] Executing Query: SELECT * FROM users WHERE username = 'admin' AND password = 'admin123'
127.0.0.1 - - [17/Jun/2025 09:01:21] "POST /login HTTP/1.1" 200 -
```

127.0.0.1:5000

Login Form

VULNERABLE MODE: Unsafe String Interpolation

Username

Password

Login

Login Form

VULNERABLE MODE: Unsafe String Interpolation

kshitij

••••••••••••

Login

127.0.0.1:5000/login

Login Failed

Incorrect username or password.

127.0.0.1:5000/login

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