

Name - Harshil Amit Buch

Class: TY CSF-1

Roll Number: 14

Experiment No. 8

Title:

Set up a simple web server and perform a basic SQL injection attack.

Objective:

To understand how SQL injection works by setting up a simple web server and database, and to observe how improper input handling can allow unauthorized access, highlighting the importance of secure coding practices in web applications.

Software/Tools Required:

- Windows/ kali
- VS Code or CMD

Theory:

SQL injection is a web security vulnerability where attackers manipulate input fields to execute unauthorized SQL commands on a database. It occurs when user inputs are directly embedded into queries without validation. Preventing it requires input sanitization, prepared statements, and parameterized queries to protect sensitive data and application integrity.

Installation Commands:

Step 1: Set up your environment

Make sure you have Python 3, SQLite, and Flask installed.

`pip install flask`

Create a new folder for your project, e.g., `sql_injection_demo`.

Step 2: Create a sample database

Inside your folder, create a file `setup_db.py`:

```
import sqlite3
conn = sqlite3.connect('test.db')
c = conn.cursor()
```

```

c.execute("""
    CREATE TABLE users (
        id INTEGER PRIMARY KEY,
        username TEXT,
        password TEXT
    )
""")
# Add sample data
c.execute("INSERT INTO users (username, password) VALUES ('admin',
'admin123')")
c.execute("INSERT INTO users (username, password) VALUES ('user',
'user123')")
conn.commit()
conn.close()
print("Database setup complete!")

```

Run it:

```
python setup_db.py
```

Step 3: Create a simple Flask web app

Create a file app.py:

```

from flask import Flask, request
import sqlite3
app = Flask(__name__)
@app.route('/')
def home():
    return """
    <h1>Login</h1>
    <form method="POST" action="/login">
        Username: <input name="username"><br>
        Password: <input name="password"><br>
        <input type="submit">
    </form>
    """
@app.route('/login', methods=['POST'])

```

```
def login():
    username = request.form['username']
    password = request.form['password']
    # WARNING: vulnerable to SQL injection
    conn = sqlite3.connect('test.db')
    c = conn.cursor()
    query = f"SELECT * FROM users WHERE username='{username}' AND
password='{password}'"
    c.execute(query)
    result = c.fetchone()
    conn.close()
    if result:
        return f"<h2>Welcome {username}!</h2>"
    else:
        return "<h2>Login Failed!</h2>"
if __name__ == "__main__":
    app.run(debug=True)
```

Step 4: Run the web server

`python app.py`

Open browser and go to <http://127.0.0.1:5000/>.

Step 5: Perform a basic SQL injection

In the login page, try these inputs:

Username: ' OR '1'='1

Password: ' OR '1'='1

This will bypass login because the SQL becomes:

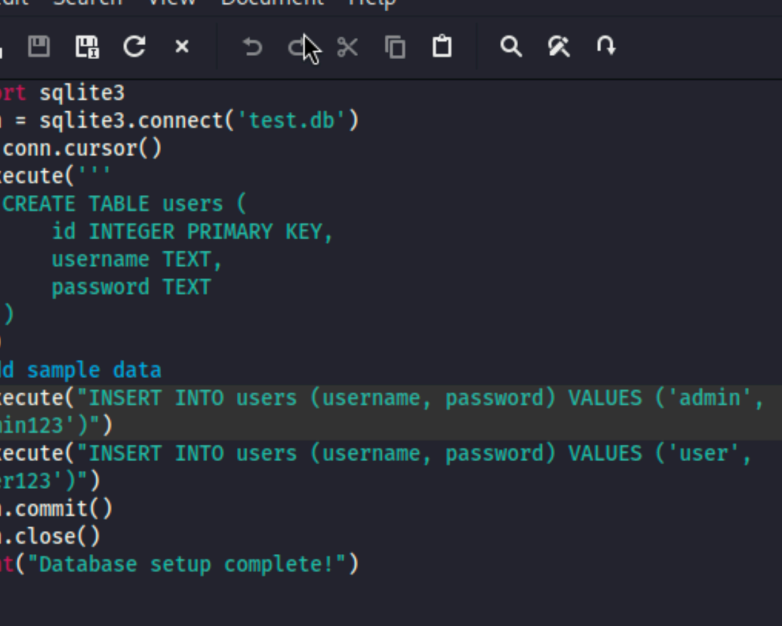
SELECT * FROM users WHERE username=" OR '1'='1' AND password=" OR '1'='1'



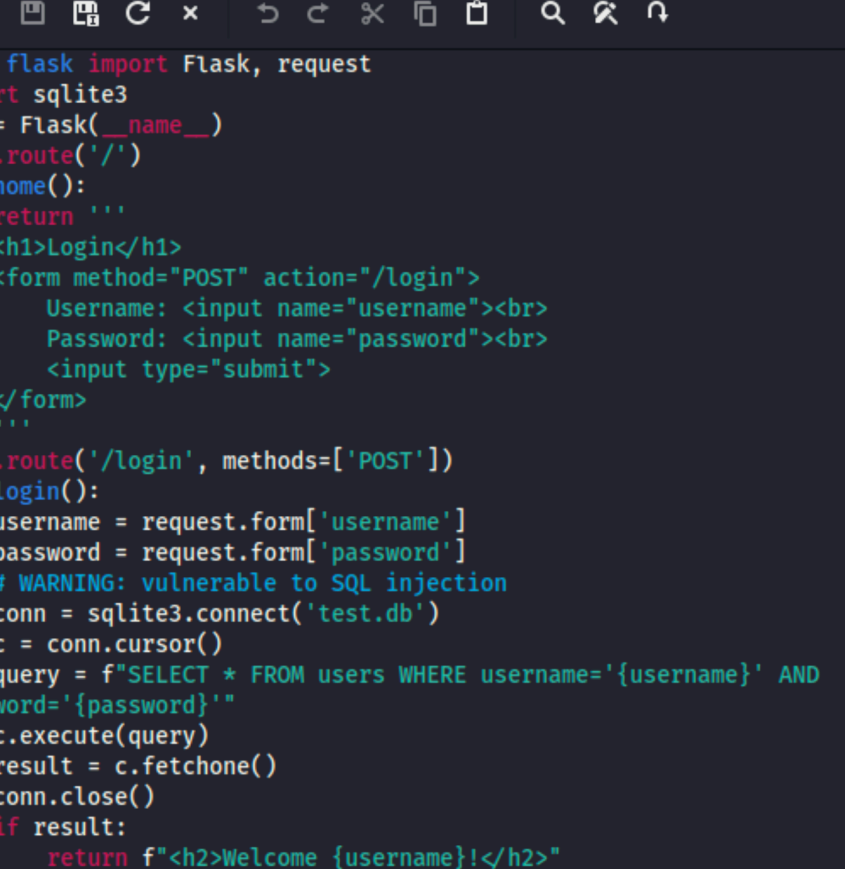
You should see a “Welcome” message even without correct credentials.

Note: Any other normal credentials will display login failed!

Output Screenshots:



```
1 import sqlite3
2 conn = sqlite3.connect('test.db')
3 c = conn.cursor()
4 c.execute('''
5     CREATE TABLE users (
6         id INTEGER PRIMARY KEY,
7         username TEXT,
8         password TEXT
9     )
10 ''')
11 # Add sample data
12 c.execute("INSERT INTO users (username, password) VALUES ('admin',
13 'admin123')")
14 c.execute("INSERT INTO users (username, password) VALUES ('user',
15 'user123')")
16 conn.commit()
17 conn.close()
18 print("Database setup complete!")
```



The screenshot shows a code editor window titled `~/Desktop/sql_injection_demo/app.py - Mousepad`. The editor contains the following Python code:

```

1 from flask import Flask, request
2 import sqlite3
3 app = Flask(__name__)
4 @app.route('/')
5 def home():
6     return '''
7     <h1>Login</h1>
8     <form method="POST" action="/login">
9         Username: <input name="username"><br>
10        Password: <input name="password"><br>
11        <input type="submit">
12    </form>
13    '''
14 @app.route('/login', methods=['POST'])
15 def login():
16     username = request.form['username']
17     password = request.form['password']
18     # WARNING: vulnerable to SQL injection
19     conn = sqlite3.connect('test.db')
20     c = conn.cursor()
21     query = f"SELECT * FROM users WHERE username='{username}' AND
password='{password}'"
22     c.execute(query)
23     result = c.fetchone()
24     conn.close()
25     if result:
26         return f"<h2>Welcome {username}!</h2>"
27     else:
28         return "<h2>Login Failed!</h2>"
29 if __name__ == "__main__":
30     app.run(debug=True)

```

```

(kali@kali)-[~/Desktop/sql_injection_demo]
$ python3 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 watchdog (inotify)
* Debugger is active!
* Debugger PIN: 119-577-206
from flask import Flask, request
import sqlite3
app = Flask(__name__)
@app.route('/')
def home():
    return '''
    <h1>Login</h1>
    <form method="POST" action="/login">
        Username: <input name="username"><br>
        Password: <input name="password"><br>
        <input type="submit">
    </form>
    ...
@app.route('/login', methods=['POST'])
def login():
    username = request.form['username']
    password = request.form['password']
    # WARNING: vulnerable to SQL injection
    conn = sqlite3.connect('test.db')
    c = conn.cursor()
    query = f"SELECT * FROM users WHERE username='{username}' AND password='{password}'"
    c.execute(query)
    result = c.fetchone()
    conn.close()
    if result:
        return f"<h2>Welcome {username}!</h2>"
    else:
        return "<h2>Login Failed!</h2>"
if __name__ == "__main__":
    app.run(debug=True)
127.0.0.1 - - [28/Oct/2025 11:25:58] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [28/Oct/2025 11:25:58] "GET /favicon.ico HTTP/1.1" 404 -
127.0.0.1 - - [28/Oct/2025 11:26:09] "POST /login HTTP/1.1" 200 -

```

127.0.0.1:5000/

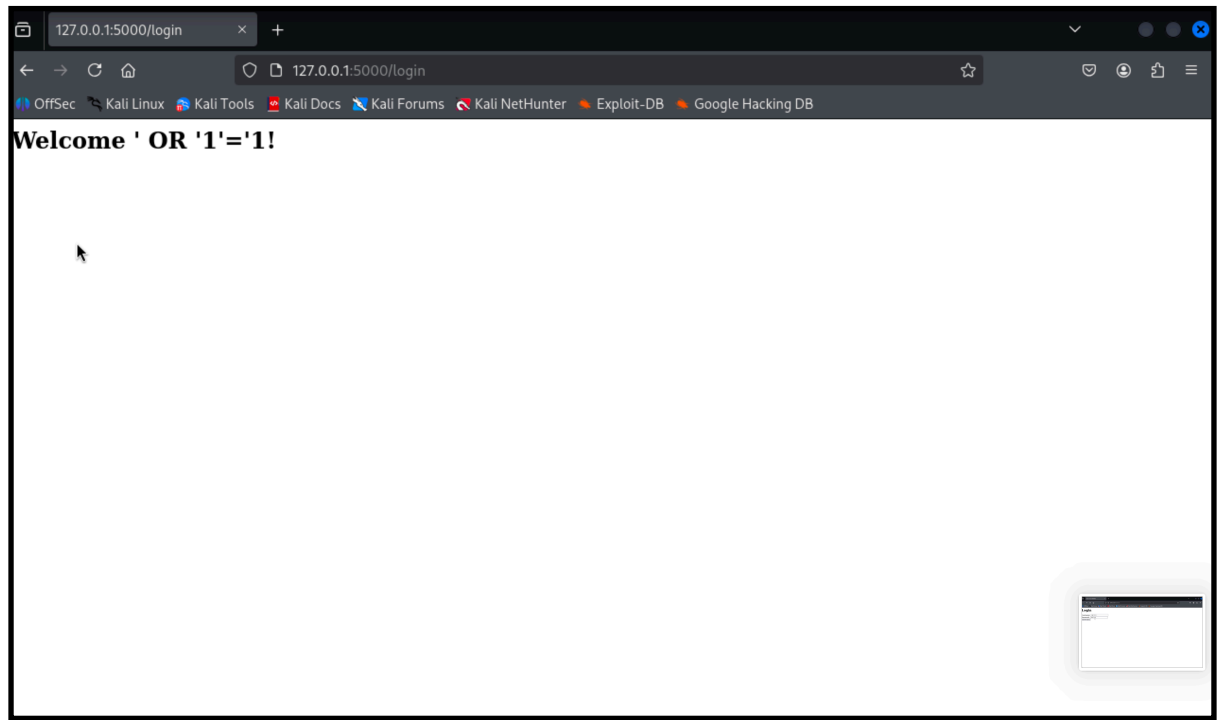
127.0.0.1:5000

OffSec Kali Linux Kali Tools Kali Docs Kali Forums Kali NetHunter Exploit-DB Google Hacking DB

Login

Username:

Password:



Outcome:

Successfully demonstrated a basic SQL injection attack in a controlled environment. Login was bypassed using crafted inputs, showing how malicious users can exploit vulnerable code, emphasizing the risks of directly embedding user inputs into SQL queries.

Conclusion:

SQL injection is a serious security vulnerability that can compromise data. Using parameterized queries, input validation, and secure coding practices prevents such attacks. Hands-on practice reinforces understanding of web security principles and the need to always validate and sanitize user inputs.