SQL INJECTION

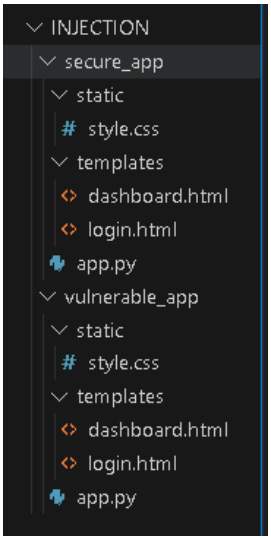
Apa itu SQL INJECTION

SQL injection adalah teknik peretasan yang memanfaatkan celah keamanan dalam aplikasi yang berkomunikasi dengan database. Celah ini memungkinkan peretas untuk memasukkan kode SQL berbahaya melalui input pengguna (seperti form login atau URL) dan memaksa database melakukan sesuatu yang tidak seharusnya, seperti mengakses, mengubah, atau menghapus data.

Misalnya, jika sebuah form login tidak diamankan dengan baik, peretas bisa memasukkan kode yang membuat sistem melewati proses verifikasi login dan memberikan akses tanpa password yang valid. Jadi, SQL injection bisa sangat berbahaya jika aplikasi tidak memiliki perlindungan yang cukup.

Penjelasan terkait web sql injection saya

1. Pertama saya menyiapkan beberapa folder dan file terutama untuk membedakan website secure dan vulnerable



2. Jika sudah kita dapat membuat databasenya terlebih dahulu, disini saya menggunakan docker dengan database mysql
   a. Untuk databasenya disini saya menggunakan container baru untuk membuat nya kalian dapat menggunakan syntax seperti ini **docker run --name (opsional) -e MYSQL_ROOT_PASSWORD=(opsional) -e MYSQL_DATABASE=(opsional) -p 3306:3306 -d mysql:latest**

   b. Jika sudah anda dapat mengeksekusi atau login ke kontainer dan database yang sudah anda buat

```
PS C:\Users\hp> docker exec -it sql_injection-container mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 8
Server version: 9.0.1 MySQL Community Server - GPL

Copyright (c) 2000, 2024, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

c. Jika sudah anda dapat masuk ke database yang sudah anda buat dan membuat tables database agar dapat dihubungkan ke program anda, disini saya akan menampilkan tables dan isi dari database saya

```
mysql> use sql_injection
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> SELECT * FROM user
    -> ;
+----+----------+-------------+
| id | username | password    |
+----+----------+-------------+
|  1 | admin    | password123 |
|  2 | user1    | user1       |
+----+----------+-------------+
2 rows in set (0.01 sec)

mysql>
```

Program web vulnerable yang saya buat

```
from flask import Flask, request, render_template, redirect, url_for, session
import mysql.connector

app = Flask(__name__)
app.secret_key = 'your_secret_key'

def get_db_connection():
    conn = mysql.connector.connect(
        host='localhost',
        port='3309',
        user='root',
        password='secret',
        database='sql_injection'
    )
    return conn

@app.route('/')
def index():
    return render_template('login.html')

# Rentan terhadap SQL Injection
@app.route('/login', methods=['POST'])
def login():
    username = request.form['username']
    password = request.form['password']

    conn = get_db_connection()
    cursor = conn.cursor()

    # Query yang rentan SQL Injection
    query = f"SELECT * FROM user WHERE username = '{username}' AND password =
'{password}'"
    cursor.execute(query)
    user = cursor.fetchone()
    conn.close()

    if user:
        session['username'] = username
        return redirect(url_for('dashboard'))
    else:
        return "Login gagal!"

@app.route('/dashboard')
def dashboard():
    if 'username' in session:
        activities = [
            "Jogging selama 30 menit",
```

```
            "Bersepeda 10 km",
            "Push-up 50 kali",
            "Berenang selama 1 jam"
        ]
        return render_template('dashboard.html', username=session['username'],
activities=activities)
    else:
        return redirect(url_for('index'))

@app.route('/logout')
def logout():
    session.pop('username', None)
    return redirect(url_for('index'))

if __name__ == '__main__':
    app.run(debug=True)
```

Program ini merupakan contoh yang rentan terhadap sql injcetion tepatnya pada kode yang dibawah ini

query = f"SELECT * FROM user WHERE username = '{username}' AND password = '{password}'"

Pengguna bisa memasukkan karakter khusus, seperti **' OR '1'='1**, untuk mengubah struktur query dan mendapatkan akses tanpa login yang sah. Misalnya, jika pengguna memasukkan:

Username: admin' OR '1'='1

Password: (kosong)

Sehingga Query sql yang dihasilkan akan menjadi seperti ini

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

Bagian **OR '1'='1'** selalu benar, sehingga query akan selalu menemukan kecocokan dan mengizinkan login meskipun password salah, sehingga aplikasi menjadi tidak aman, Untuk mengatasi ini, seharusnya digunakan **prepared statements atau parameterized queries** yang memisahkan data dari logika SQL, sehingga input pengguna tidak bisa mengubah struktur query.

Dan ini Program web Secure yang saya buat

```
from flask import Flask, request, render_template, redirect, url_for, session
import mysql.connector

app = Flask(__name__)
app.secret_key = 'your_secret_key'

def get_db_connection():
    conn = mysql.connector.connect(
        host='localhost',
        port='3309',
        user='root',
        password='secret',
        database='sql_injection'
    )
    return conn

@app.route('/')
def index():
    return render_template('login.html')

@app.route('/login', methods=['POST'])
def login():
    username = request.form['username']
    password = request.form['password']

    conn = get_db_connection()
    cursor = conn.cursor()
    cursor.execute("SELECT * FROM user WHERE username = %s AND password = %s",
(username, password))
    user = cursor.fetchone()
    conn.close()

    if user:
        session['username'] = username
        return redirect(url_for('dashboard'))
    else:
        return "Login gagal!"

@app.route('/dashboard')
def dashboard():
    if 'username' in session:
        activities = [
            "Jogging selama 30 menit",
            "Bersepeda 10 km",
            "Push-up 50 kali",
```

```
            "Berenang selama 1 jam"
        ]
        return render_template('dashboard.html', username=session['username'],
activities=activities)
    else:
        return redirect(url_for('index'))

@app.route('/logout')
def logout():
    session.pop('username', None)
    return redirect(url_for('index'))

if __name__ == '__main__':
    app.run(debug=True)
```

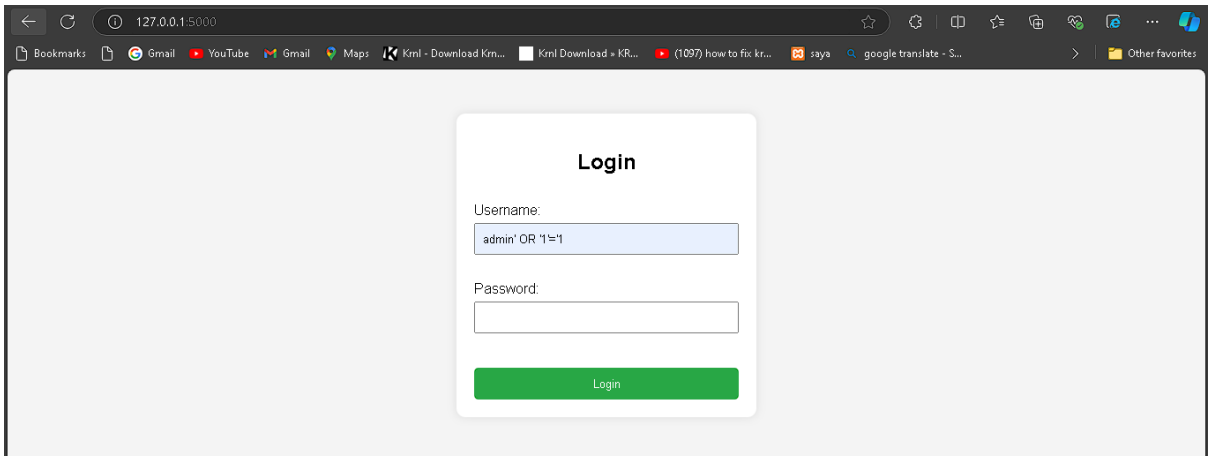
secure dari aplikasi sebelumnya karena telah menggunakan **parameterized queries** untuk mencegah SQL Injection.

cursor.execute("SELECT * FROM user WHERE username = %s AND password = %s", (username, password))

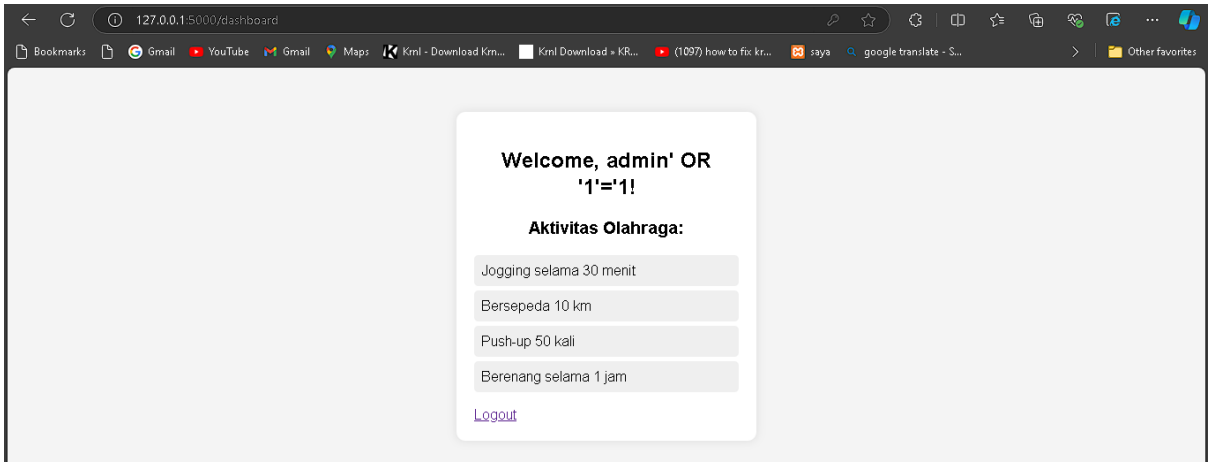
Di sini, parameter **%s** digunakan sebagai placeholder untuk username dan password, sehingga input pengguna akan di-sanitize (dibersihkan) oleh library mysql.connector sebelum digunakan dalam query. Hal ini mencegah input berbahaya dari mengubah struktur query.

Dengan cara ini, jika ada pengguna yang mencoba memasukkan input seperti **' OR '1'='1**, input tersebut hanya akan dianggap sebagai string literal, dan bukan sebagai bagian dari logika query. Ini membuat aplikasi lebih aman dari serangan SQL Injection.

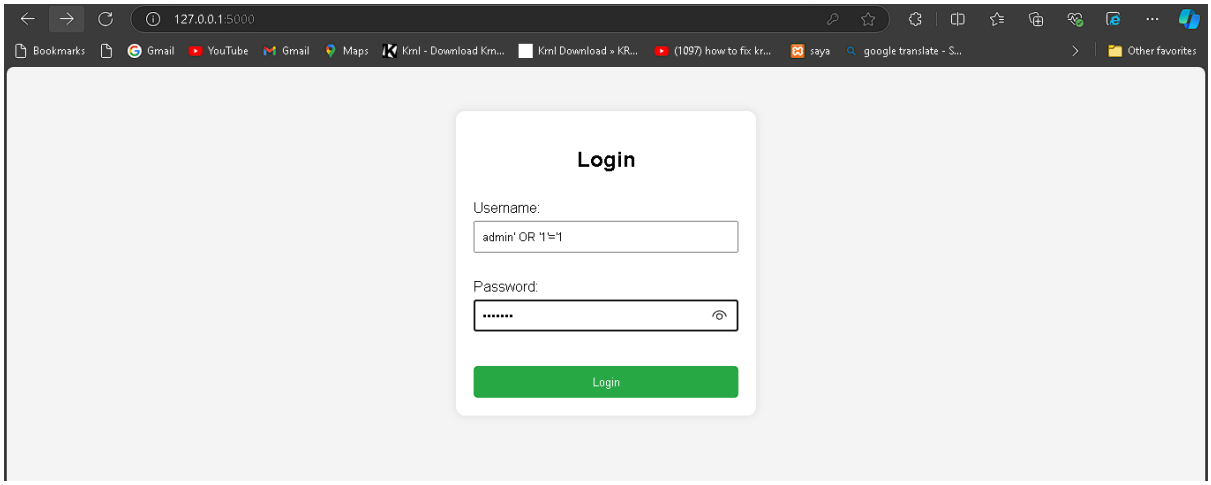
Dan ini bukti pengujian dari web vulnerable saya



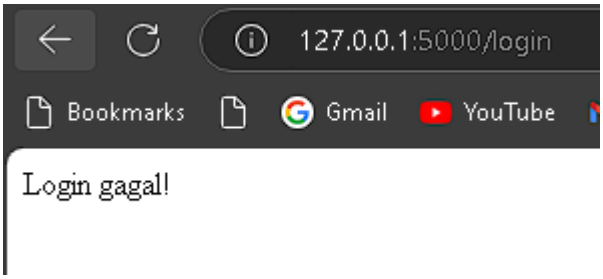
Karena tidak secure dia dapat dimasuki dengan menggunakan **' OR '1'='1** walaupun password yang diberikan asal-asalan



Dan ini bukti pengujian dari web Secure saya



Pada bagian secure baik saat menguji coba dengan menggunakan **' OR '1'='1** baik password benar ataupun salah dia tidak dapat masuk



Sekian dari Laporan saya terimakasih