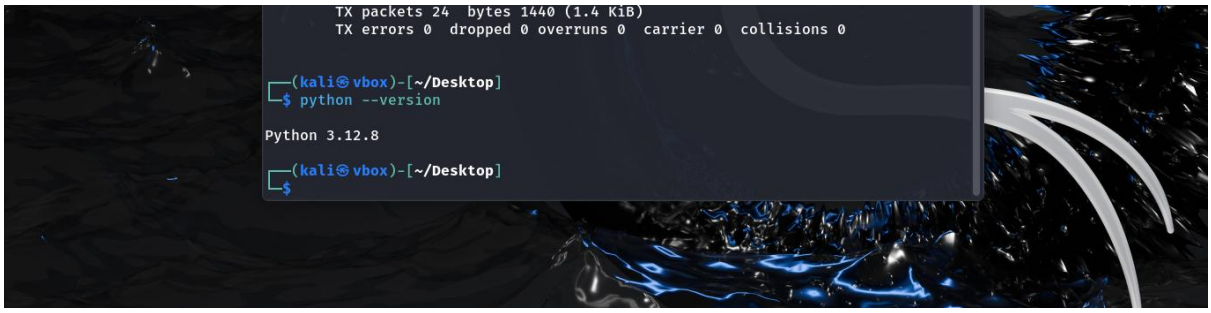
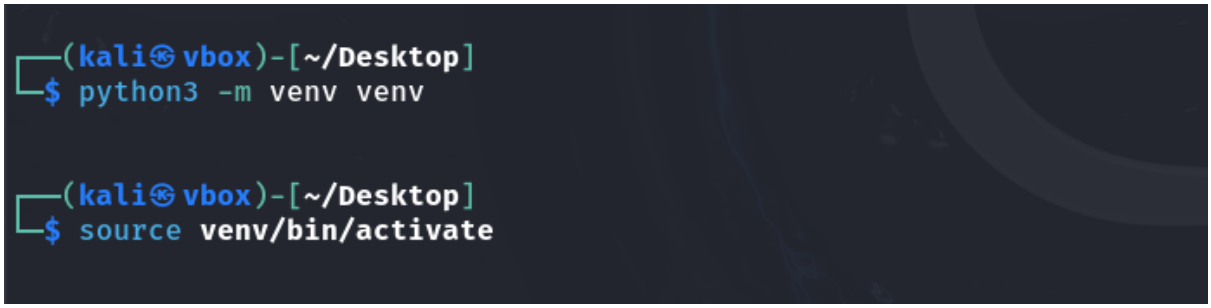
- **Step 1: Set Up the Development Environment**

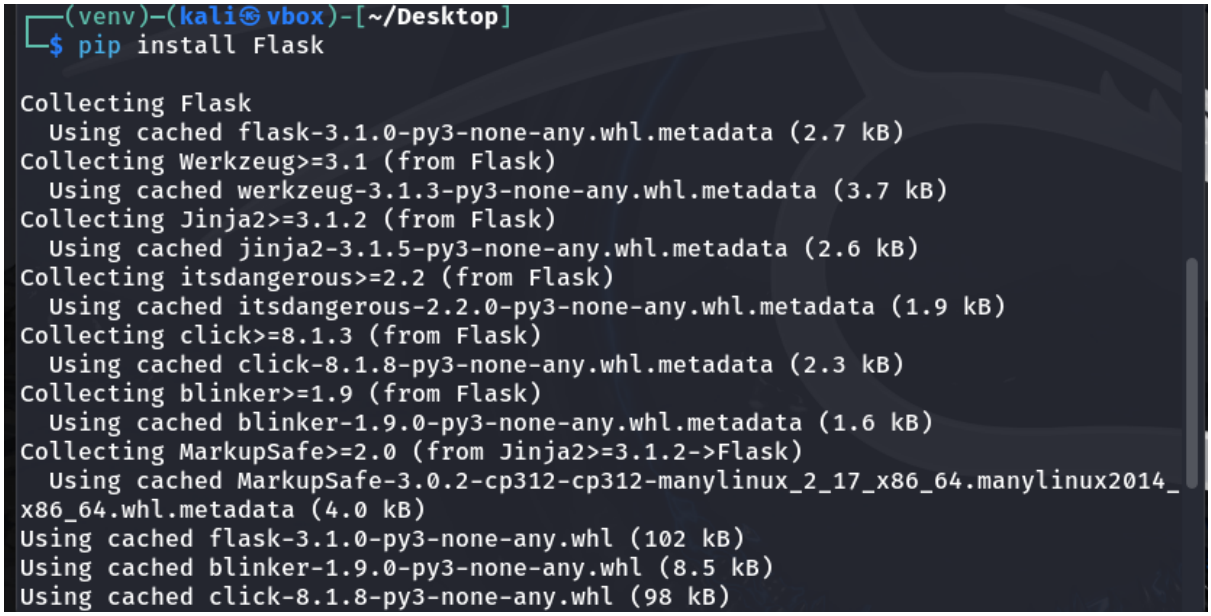


- **Set Up a Virtual Environment**



You should see (venv) before the terminal prompt, which means the environment is activated.

- **Install Flask**



🔹 **Step 2: Download the Sample Application**

- **Create the Application Files**

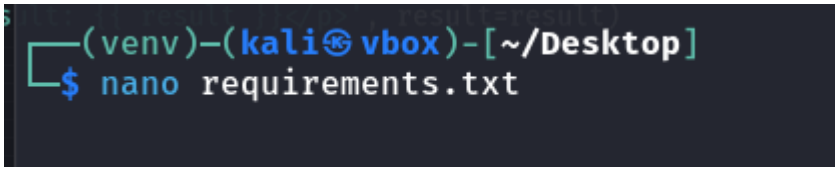
Create a new folder for your project, then inside it, create a file named app.py.

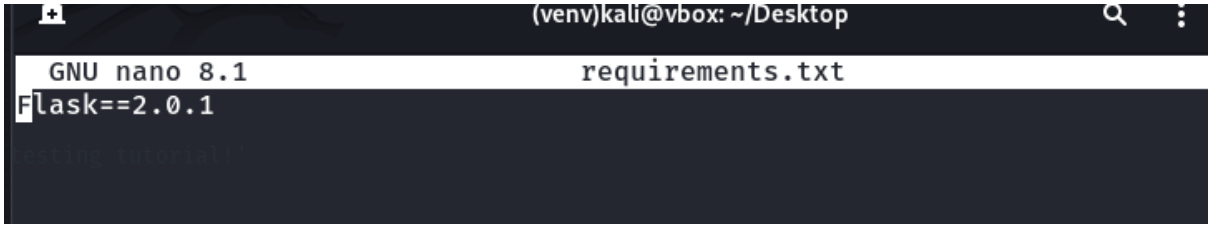
```
from flask import Flask, request, render_template_string

app = Flask(__name__)

@app.route('/')
def home():
    return 'Welcome to the web application security testing tutorial!'

@app.route('/search', methods=['GET', 'POST'])
def search():
    if request.method == 'POST':
        query = request.form['query']
        # 🚨 Vulnerability: SQL Injection 🚨
```

```
    result = execute_query(f"SELECT * FROM users WHERE username = '{query}'")
    return render_template_string('<p>Search result: {{ result }}</p>', result=result)

  return '''
  <form method="post">
  Search: <input type="text" name="query"><br>
  <input type="submit" value="Search">
  </form>
  '''

def execute_query(query):
  # Simulated database query
  users = {'admin': 'password123', 'user1': 'pass1', 'user2': 'pass2'}
  if query in users:
    return f'User: {query}, Password: {users[query]}'
  return 'No results found'

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

- **Create a Requirements File**

Create a new file named **requirements.txt** and add this line:





- **Install Dependencies**

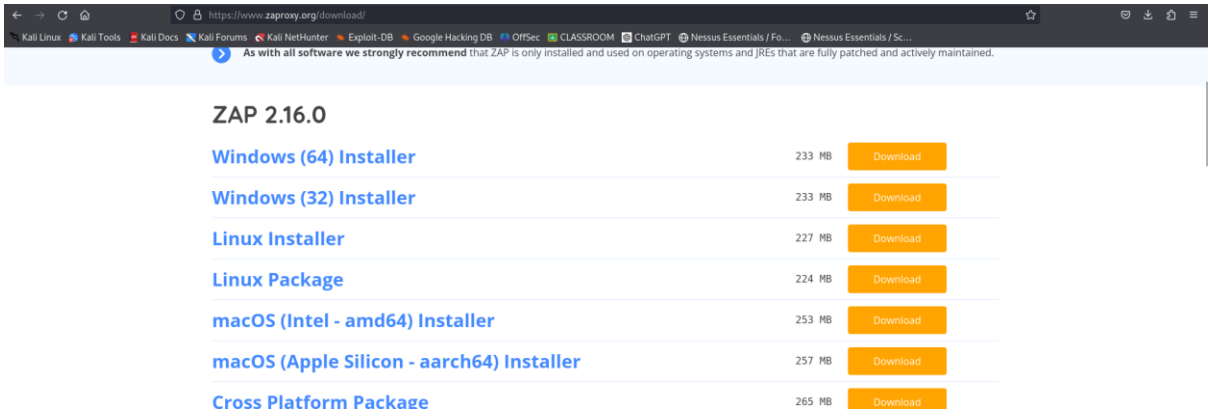
```
┌──(venv)─(kali㉿vbox)-[~/Desktop]
└─$ pip install -r requirements.txt

Collecting Flask==2.0.1 (from -r requirements.txt (line 1))
  Using cached Flask-2.0.1-py3-none-any.whl.metadata (3.8 kB)
Requirement already satisfied: Werkzeug>=2.0 in ./venv/lib/python3.12/site-packa
ges (from Flask==2.0.1->-r requirements.txt (line 1)) (3.1.3)
Requirement already satisfied: Jinja2>=3.0 in ./venv/lib/python3.12/site-package
s (from Flask==2.0.1->-r requirements.txt (line 1)) (3.1.5)
Requirement already satisfied: itsdangerous>=2.0 in ./venv/lib/python3.12/site-p
ackages (from Flask==2.0.1->-r requirements.txt (line 1)) (2.2.0)
Requirement already satisfied: click>=7.1.2 in ./venv/lib/python3.12/site-packag
es (from Flask==2.0.1->-r requirements.txt (line 1)) (8.1.8)
Requirement already satisfied: MarkupSafe>=2.0 in ./venv/lib/python3.12/site-pac
```

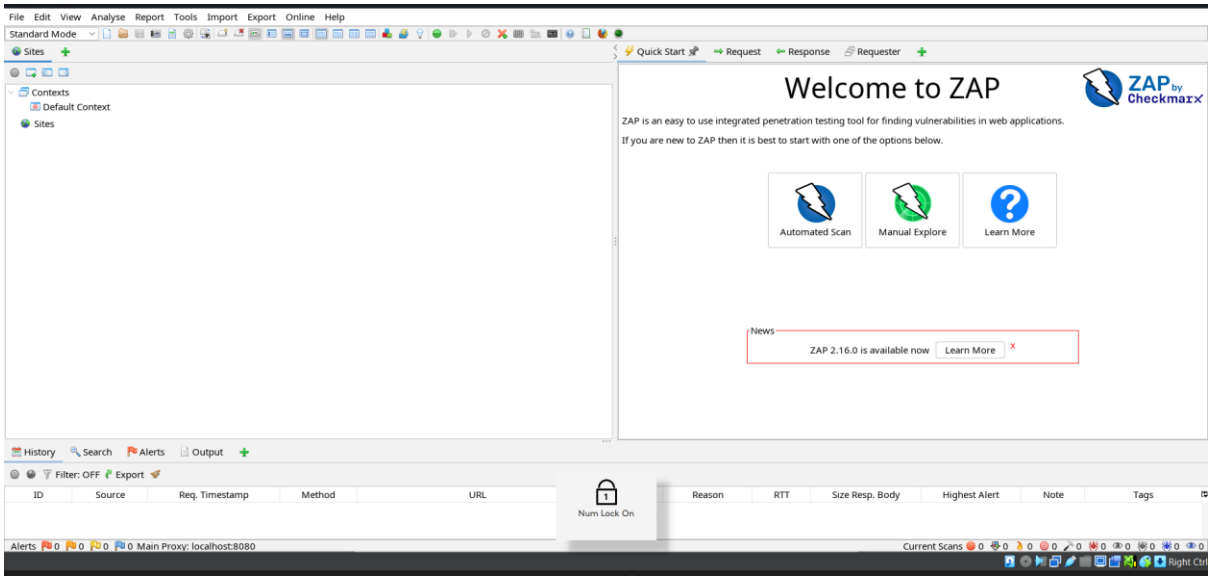
- **Run the Application**

```
┌──(venv)─(kali㉿vbox)-[~/Desktop]
└─$ pip install Werkzeug==2.0.3

Collecting Werkzeug==2.0.3
  Using cached Werkzeug-2.0.3-py3-none-any.whl.metadata (4.5 kB)
Using cached Werkzeug-2.0.3-py3-none-any.whl (289 kB)
Installing collected packages: Werkzeug
  Attempting uninstall: Werkzeug
    Found existing installation: Werkzeug 3.1.3
    Uninstalling Werkzeug-3.1.3:
      Successfully uninstalled Werkzeug-3.1.3
Successfully installed Werkzeug-2.0.3

┌──(venv)─(kali㉿vbox)-[~/Desktop]
└─$ python app.py

 * Serving Flask app 'app' (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployme
nt.
   Use a production WSGI server instead.
 * Debug mode: on
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
 * Restarting with stat
```

🔹 **Step 3: Set Up OWASP ZAP**

- **Download and Install OWASP ZAP**
- 🔗 Download from: https://www.zaproxy.org/download/





- **Step 4: Scan the Application with OWASP ZAP**

  - **Configure OWASP ZAP as a Proxy**

To capture traffic between your browser and the Flask app, you need to set up your browser proxy.

For Firefox:

Open Firefox → Go to Settings → Search for Proxy.

Click "Settings" → Choose "Manual proxy configuration".

Enter:

HTTP Proxy: localhost

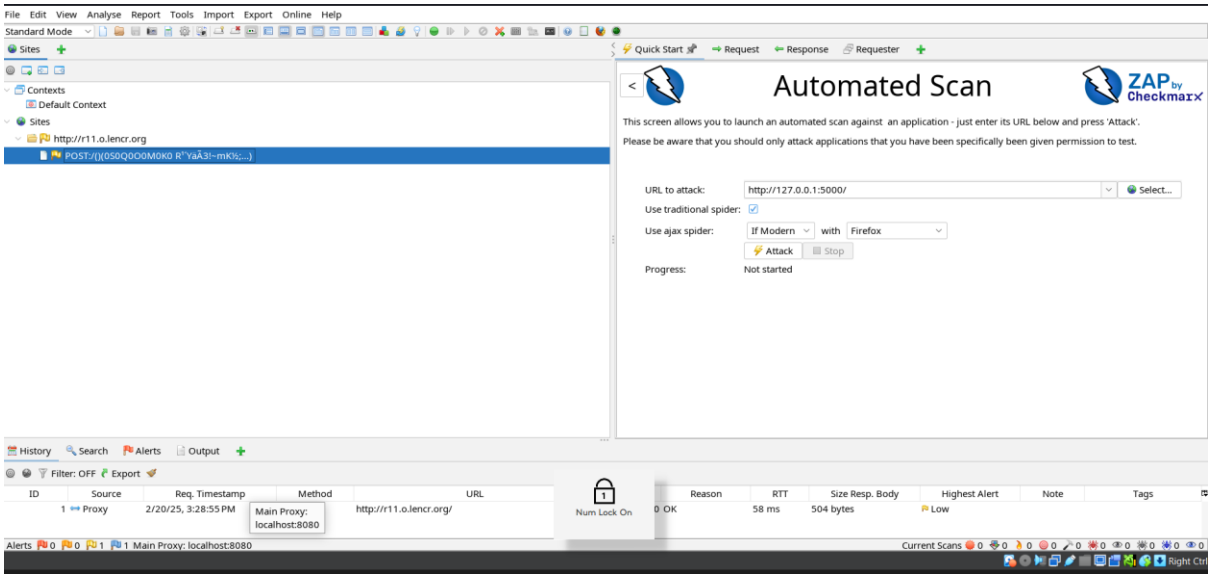
Port: 8080

Click OK

- **Access the Application through ZAP**

1. **Open Firefox and go to http://127.0.0.1:5000/.**

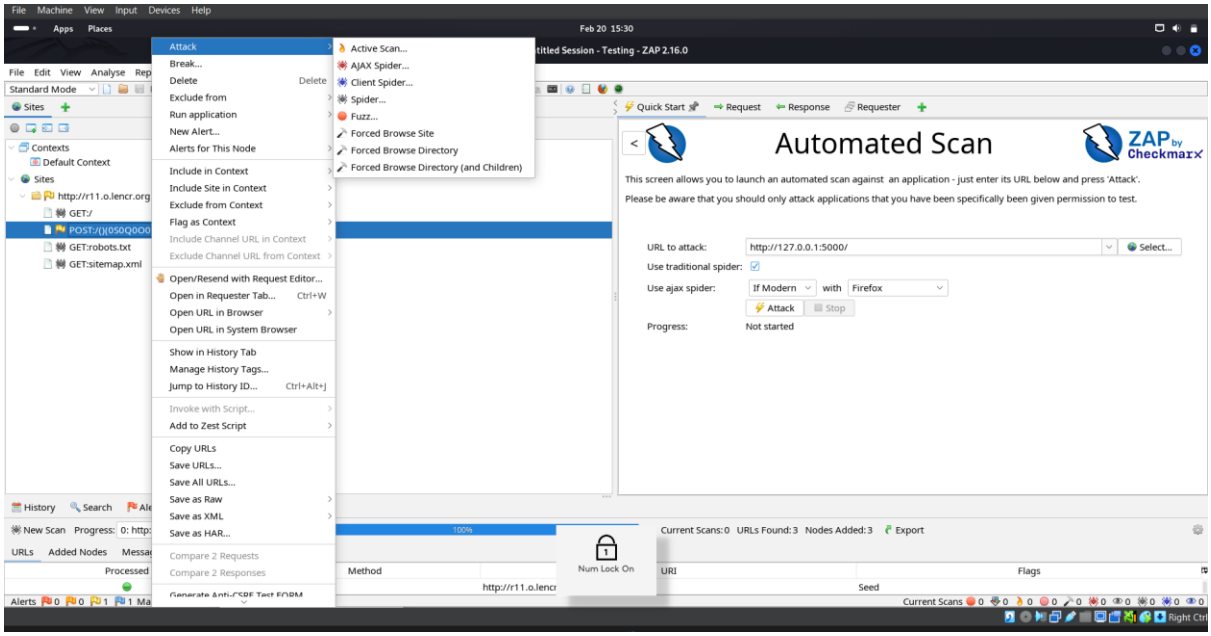
2. **OWASP ZAP will capture the traffic in the "Sites" tab.**



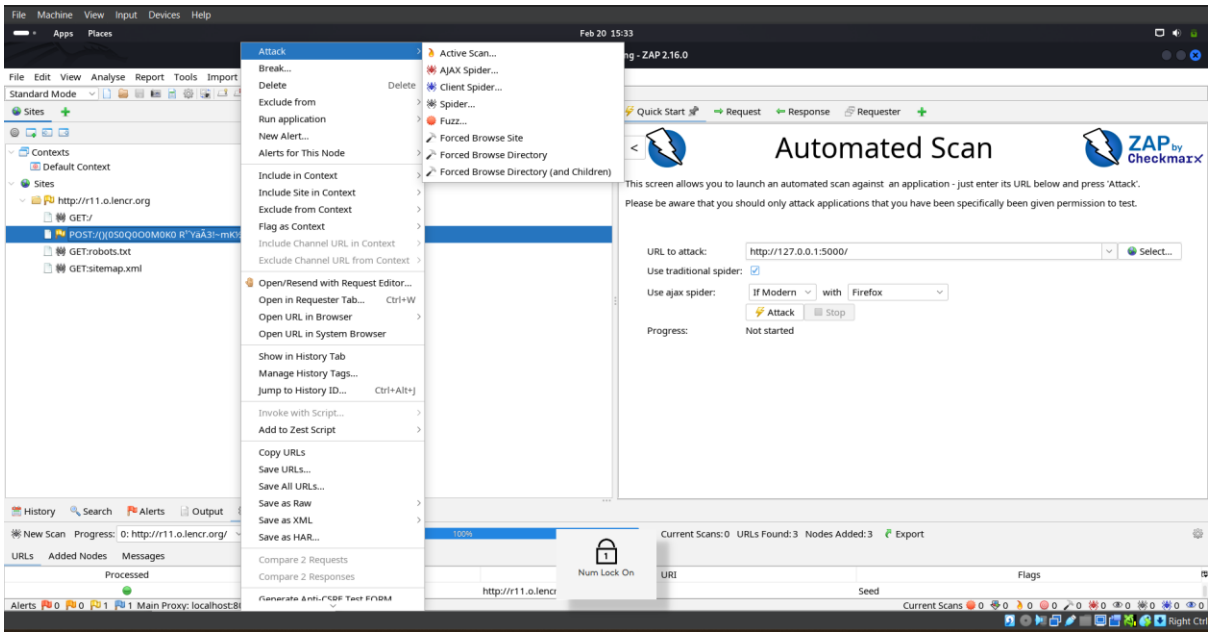
- **Perform a Scan**

1. **In OWASP ZAP, right-click on http://127.0.0.1:5000/ in the Sites tab.**

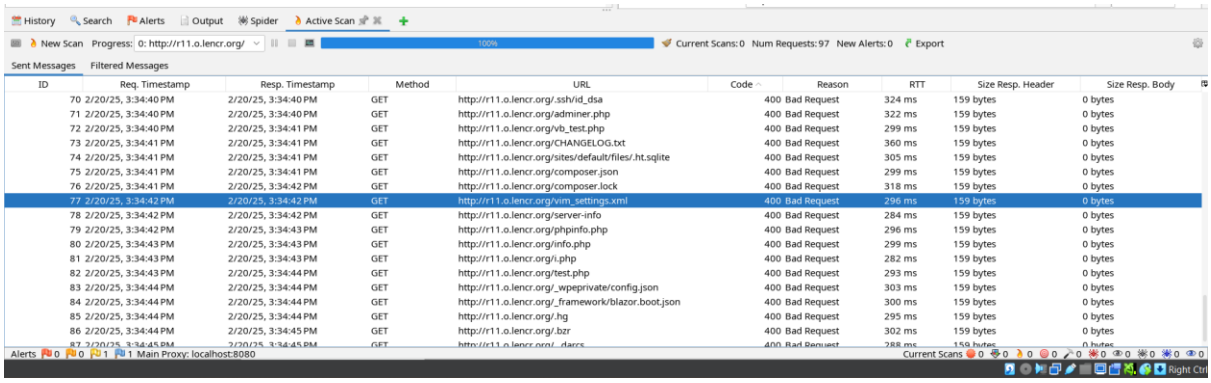
2. **Select "Attack" → "Spider" to crawl the site.**



3. **Once done, right-click again and select "Attack" → "Active Scan".**

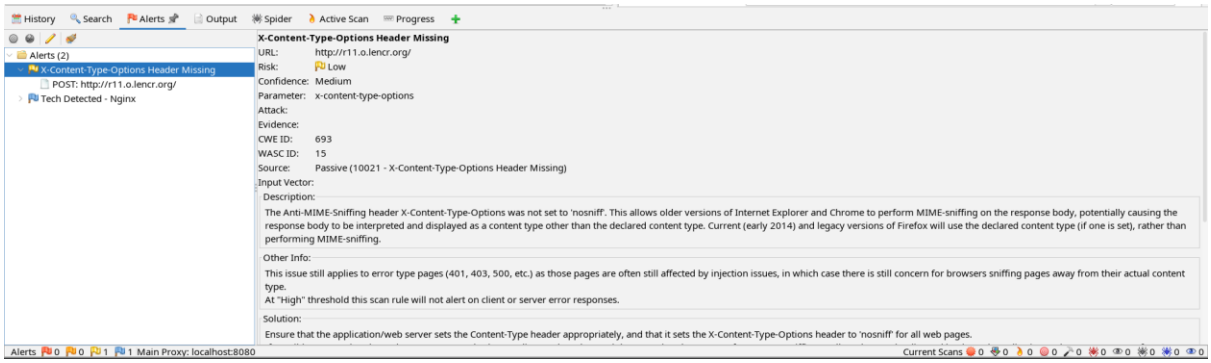


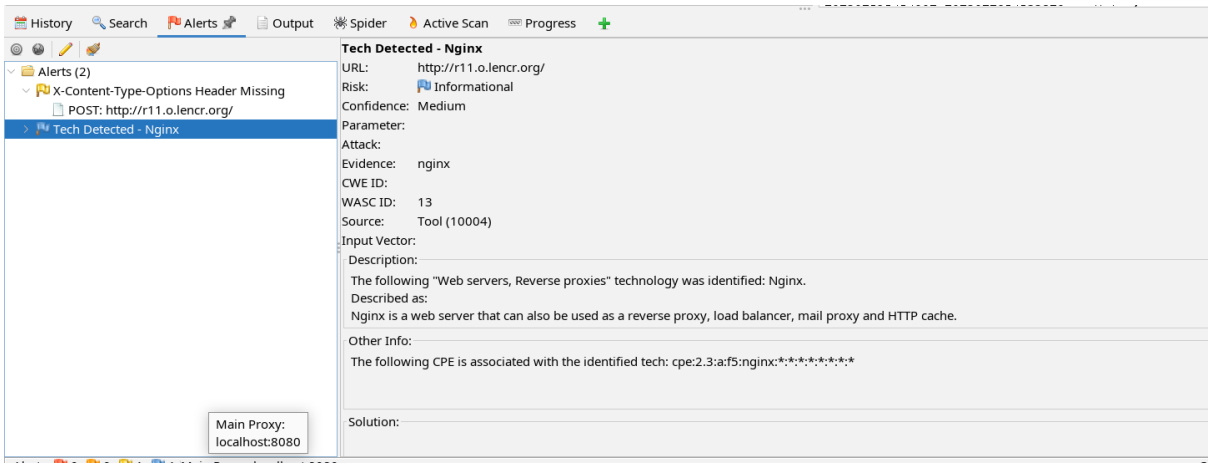
4. **Wait for the scan to complete.**



- **Analyze the Results**

➔ Go to the "Alerts" tab in OWASP ZAP.





From the provided screenshot, there is no direct indication of an SQL vulnerability. The alerts in the image focus on:

1. **X-Content-Type-Options Header Missing** – This allows MIME-sniffing, which could lead to security risks but not SQL Injection.
2. **Server Leaks Version Information** – This reveals server details, making it easier for attackers to find exploits, but it's not an SQL Injection vulnerability.

➔ Look for SQL Injection vulnerabilities.
➔ The vulnerable endpoint is /search, which allows SQL Injection.

- **Step 5: Identify and Fix the Vulnerability**

1) Fix the SQL Injection Vulnerability

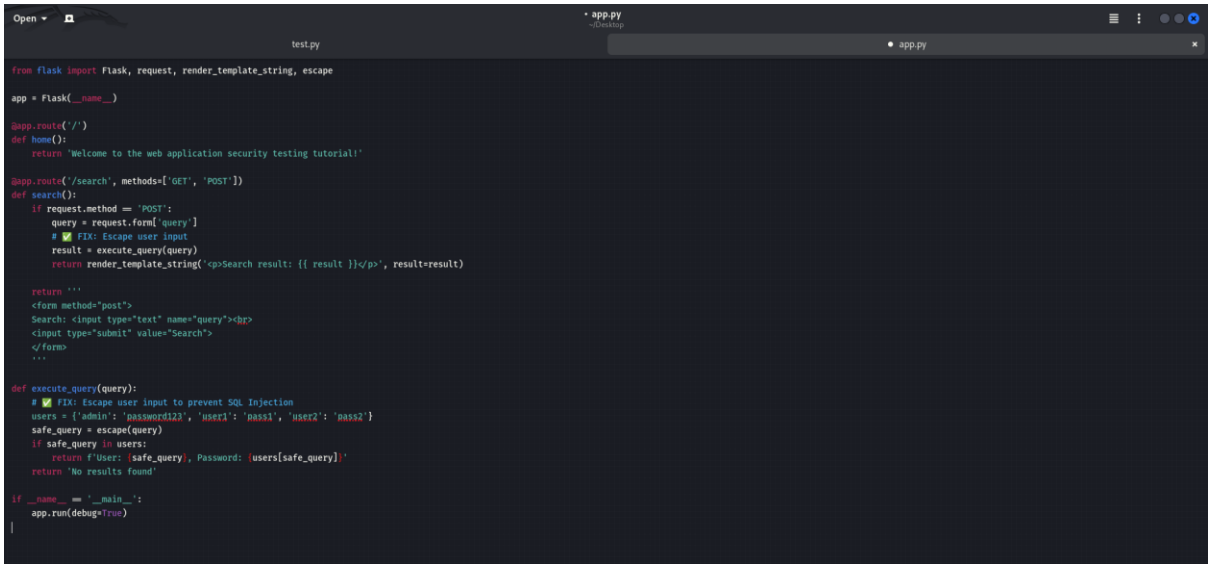
Open app.py and modify the execute_query function:

from flask import Flask, request, render_template_string, escape

app = Flask(__name__)

```
@app.route('/')
def home():
    return 'Welcome to the web application security testing tutorial!'


@app.route('/search', methods=['GET', 'POST'])
def search():
    if request.method == 'POST':
        query = request.form['query']
        # ✅ FIX: Escape user input
        result = execute_query(query)
        return render_template_string('<p>Search result: {{ result }}</p>', result=result)

    return '''
    <form method="post">
    Search: <input type="text" name="query"><br>
    <input type="submit" value="Search">
    </form>
    '''

def execute_query(query):
    # ✅ FIX: Escape user input to prevent SQL Injection
    users = {'admin': 'password123', 'user1': 'pass1', 'user2': 'pass2'}
    safe_query = escape(query)
    if safe_query in users:
        return f'User: {safe_query}, Password: {users[safe_query]}'
    return 'No results found'

if __name__ == '__main__':
```

app.run(debug=True)



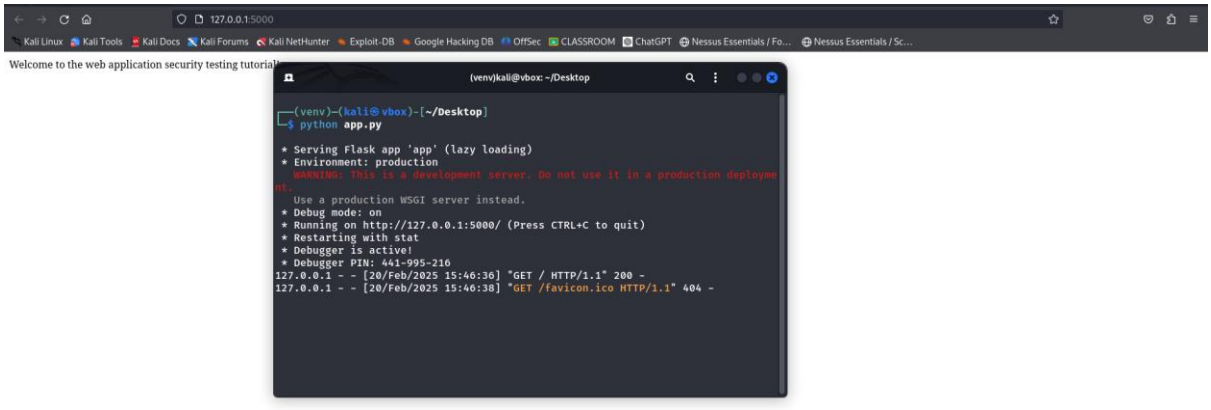
Changes made:

✔ Used escape(query) to prevent SQL injection.

Restart the Application and Re-Test

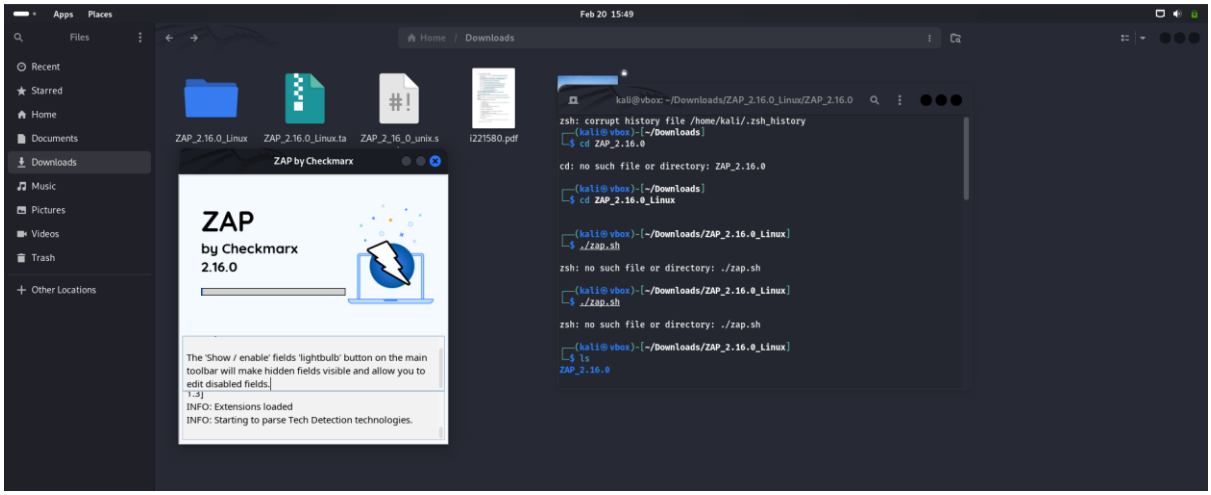
1. Stop the Flask app (CTRL + C).
2. Restart it:

python app.py



3. Perform the OWASP ZAP scan again.

4. The **SQL Injection vulnerability should now be fixed!**



From the updated screenshot, it looks like **SQL Injection** is **not listed** in the **alerts section** — which suggests the vulnerability might be **fixed**!

The alerts showing up are:

- **Server version disclosure**
- **Missing security headers (CSP, Anti-clickjacking, etc.)**

If SQL Injection was still present, ZAP would show it as a **high-risk alert** under something like **"SQL Injection"** or **"Database Error Disclosure"**.

✅ **Next steps to confirm the fix:**

1. **Manual SQL Injection Test:**

- Go to your app's **search bar**.
- Try entering payloads like:

bash

CopyEdit

' OR '1'='1

If the result **does NOT return all users or errors**, your app is safer!

2. **Double-check the code:**

- Make sure you've used **parameterized queries** or **input sanitization** (like Flask's escape() function).