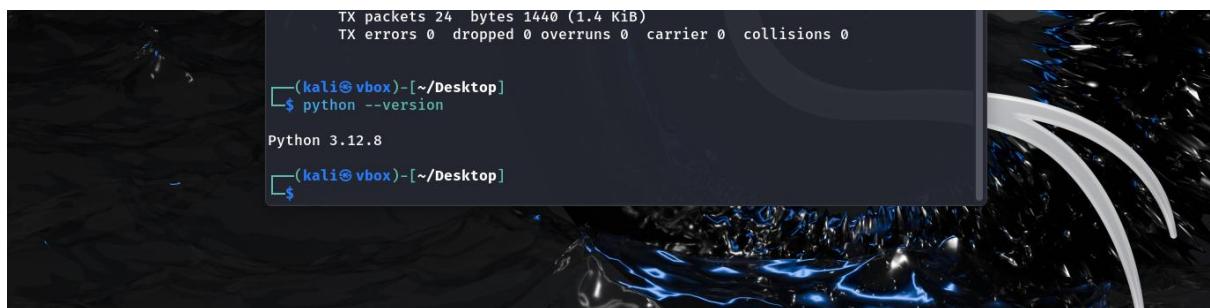


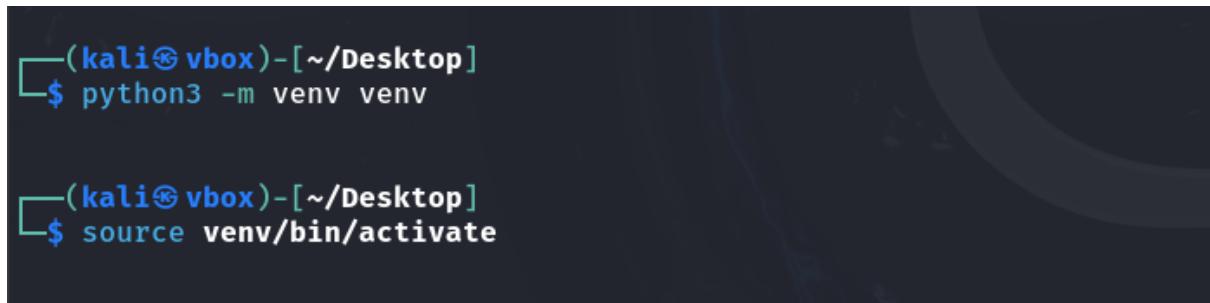
◆ Step 1: Set Up the Development Environment



```
TX packets 24 bytes 1440 (1.4 KIB)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

[(kali㉿vbox)-[~/Desktop]]
$ python --version
Python 3.12.8
[(kali㉿vbox)-[~/Desktop]]
$
```

◆ Set Up a Virtual Environment



```
[(kali㉿vbox)-[~/Desktop]]
$ python3 -m venv venv

[(kali㉿vbox)-[~/Desktop]]
$ source venv/bin/activate
```

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

◆ Install Flask

```
└─(venv)─(kali㉿vbox)─[~/Desktop]
$ pip install Flask

Collecting Flask
  Using cached flask-3.1.0-py3-none-any.whl.metadata (2.7 kB)
Collecting Werkzeug>=3.1 (from Flask)
  Using cached werkzeug-3.1.3-py3-none-any.whl.metadata (3.7 kB)
Collecting Jinja2>=3.1.2 (from Flask)
  Using cached jinja2-3.1.5-py3-none-any.whl.metadata (2.6 kB)
Collecting itsdangerous>=2.2 (from Flask)
  Using cached itsdangerous-2.2.0-py3-none-any.whl.metadata (1.9 kB)
Collecting click>=8.1.3 (from Flask)
  Using cached click-8.1.8-py3-none-any.whl.metadata (2.3 kB)
Collecting blinker>=1.9 (from Flask)
  Using cached blinker-1.9.0-py3-none-any.whl.metadata (1.6 kB)
Collecting MarkupSafe>=2.0 (from Jinja2>=3.1.2->Flask)
  Using cached MarkupSafe-3.0.2-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_
x86_64.whl.metadata (4.0 kB)
Using cached flask-3.1.0-py3-none-any.whl (102 kB)
Using cached blinker-1.9.0-py3-none-any.whl (8.5 kB)
Using cached click-8.1.8-py3-none-any.whl (98 kB)
```

◆ 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'

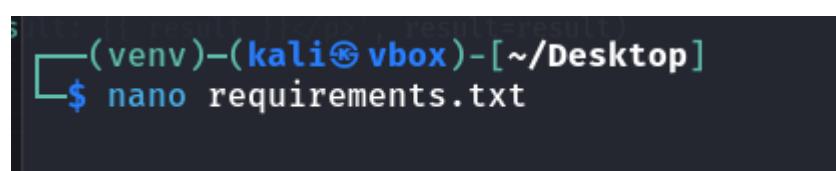
if __name__ == '__main__':

app.run(debug=True)

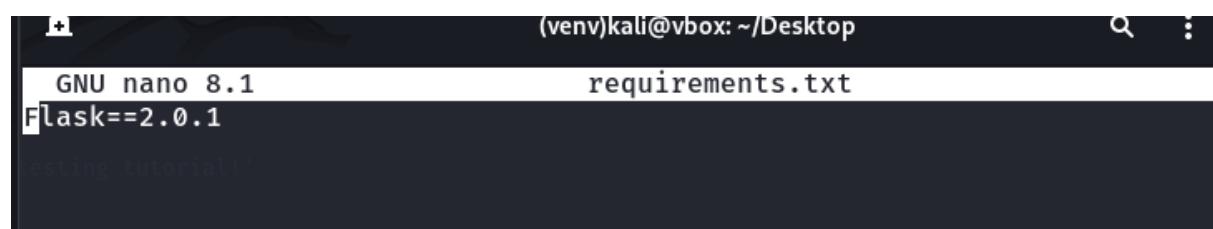
```

- **Create a Requirements File**

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



A terminal window titled '(venv)-(kali㉿vbox)-[~/Desktop]'. The command '\$ nano requirements.txt' is entered and executed.



The terminal shows the contents of the requirements.txt file:

```

+-----+
              (venv)kali@vbox: ~/Desktop
  GNU nano 8.1           requirements.txt
Flask==2.0.1
Testing tutorial!

```

- **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-packages (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-packages (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-packages (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-packages (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 deployment.
  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/>

The image shows two screenshots of the ZAP (Zed Attack Proxy) tool. The top screenshot is a web browser displaying the download page for ZAP 2.16.0, showing various installer and package options with download links. The bottom screenshot shows the graphical user interface of ZAP, featuring a sidebar with site contexts and a main pane titled 'Welcome to ZAP' with options for automated scanning and manual exploration.

◆ 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:

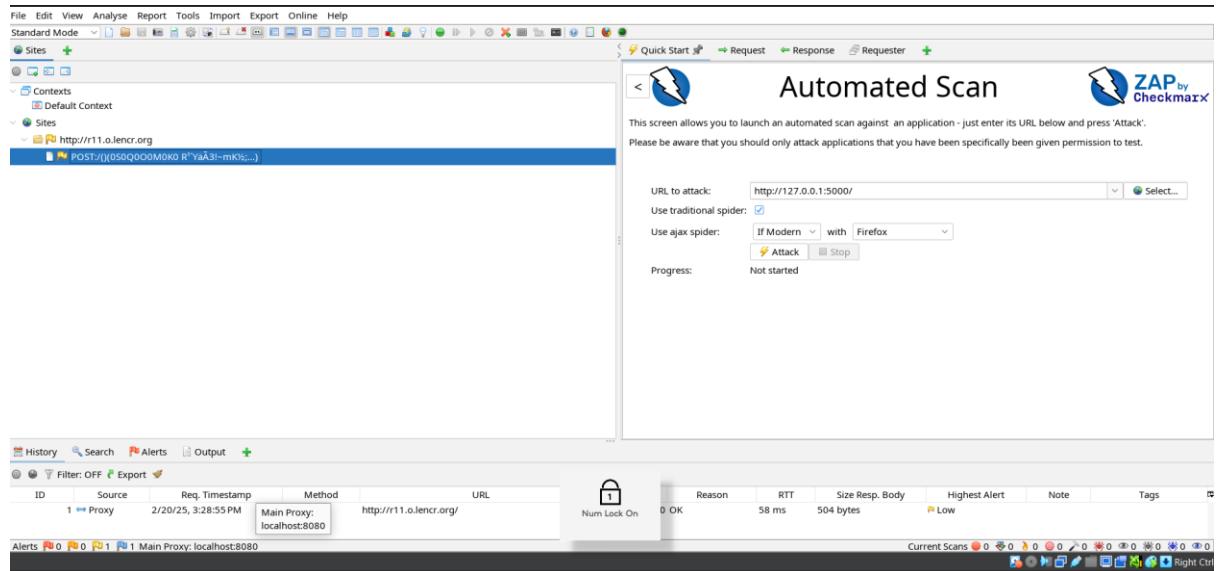
HTTP Proxy: localhost

Port: 8080

Click OK

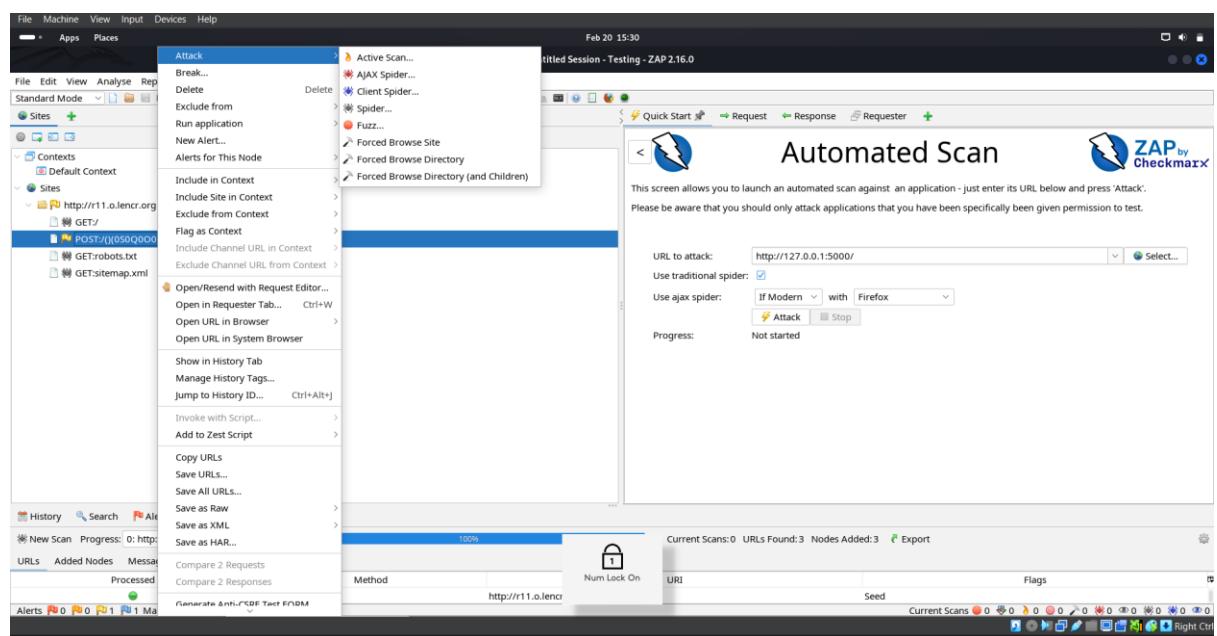
- Access the Application through ZAP

1. Open Firefox and go to <http://127.0.0.1:5000/>.
2. OWASP ZAP will capture the traffic in the "Sites" tab.

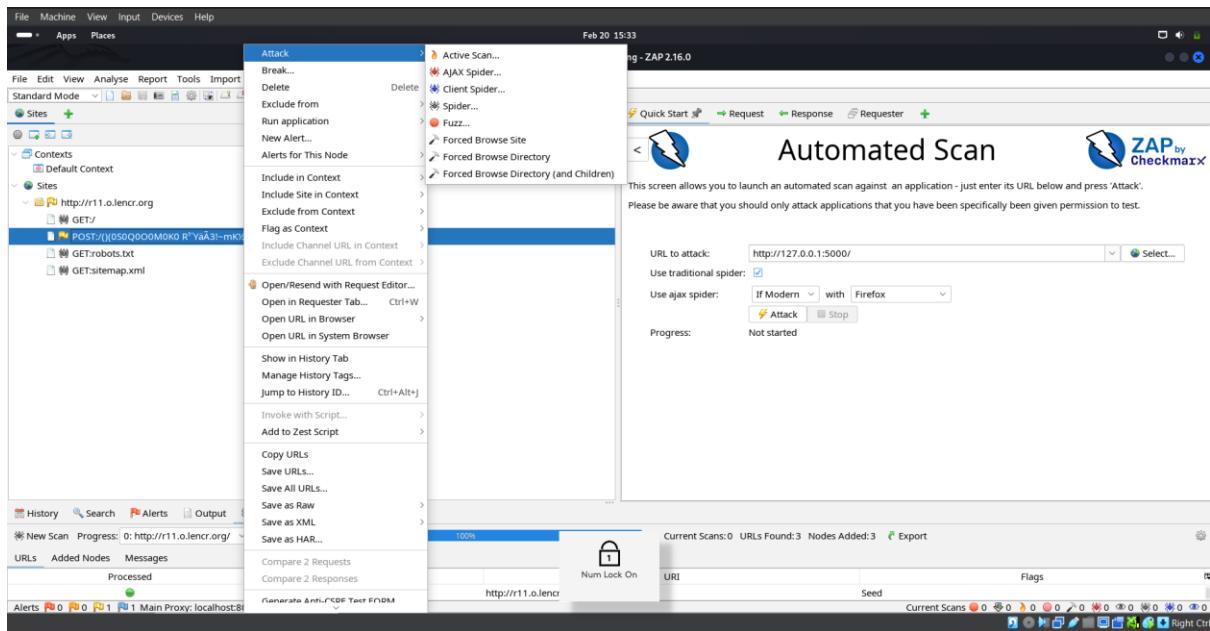


- Perform a Scan

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.

ID	Req. Timestamp	Resp. Timestamp	Method	URL	Code	Reason	RTT	Size Resp. Header	Size Resp. Body
70	2/20/25, 3:34:40 PM	2/20/25, 3:34:40 PM	GET	http://r11.o.lencr.org/.ssh/id_dsa	400	Bad Request	324 ms	159 bytes	0 bytes
71	2/20/25, 3:34:40 PM	2/20/25, 3:34:40 PM	GET	http://r11.o.lencr.org/adminer.php	400	Bad Request	322 ms	159 bytes	0 bytes
72	2/20/25, 3:34:40 PM	2/20/25, 3:34:41 PM	GET	http://r11.o.lencr.org/vb_test.php	400	Bad Request	299 ms	159 bytes	0 bytes
73	2/20/25, 3:34:41 PM	2/20/25, 3:34:41 PM	GET	http://r11.o.lencr.org/CHANGELOG.txt	400	Bad Request	360 ms	159 bytes	0 bytes
74	2/20/25, 3:34:41 PM	2/20/25, 3:34:41 PM	GET	http://r11.o.lencr.org/sites/default/files/ht.sqlite	400	Bad Request	305 ms	159 bytes	0 bytes
75	2/20/25, 3:34:41 PM	2/20/25, 3:34:41 PM	GET	http://r11.o.lencr.org/composer.json	400	Bad Request	299 ms	159 bytes	0 bytes
76	2/20/25, 3:34:41 PM	2/20/25, 3:34:42 PM	GET	http://r11.o.lencr.org/composer.lock	400	Bad Request	318 ms	159 bytes	0 bytes
77	2/20/25, 3:34:47 PM	2/20/25, 3:34:47 PM	GET	http://r11.o.lencr.org/vim_settings.xml	400	Bad Request	296 ms	159 bytes	0 bytes
78	2/20/25, 3:34:42 PM	2/20/25, 3:34:42 PM	GET	http://r11.o.lencr.org/server-info	400	Bad Request	284 ms	159 bytes	0 bytes
79	2/20/25, 3:34:42 PM	2/20/25, 3:34:43 PM	GET	http://r11.o.lencr.org/phpinfo.php	400	Bad Request	296 ms	159 bytes	0 bytes
80	2/20/25, 3:34:43 PM	2/20/25, 3:34:43 PM	GET	http://r11.o.lencr.org/info.php	400	Bad Request	299 ms	159 bytes	0 bytes
81	2/20/25, 3:34:43 PM	2/20/25, 3:34:43 PM	GET	http://r11.o.lencr.org/i.php	400	Bad Request	282 ms	159 bytes	0 bytes
82	2/20/25, 3:34:43 PM	2/20/25, 3:34:44 PM	GET	http://r11.o.lencr.org/test.php	400	Bad Request	293 ms	159 bytes	0 bytes
83	2/20/25, 3:34:44 PM	2/20/25, 3:34:44 PM	GET	http://r11.o.lencr.org/_wepriprivate/config.json	400	Bad Request	303 ms	159 bytes	0 bytes
84	2/20/25, 3:34:44 PM	2/20/25, 3:34:44 PM	GET	http://r11.o.lencr.org/_framework/blazor.boot.json	400	Bad Request	300 ms	159 bytes	0 bytes
85	2/20/25, 3:34:44 PM	2/20/25, 3:34:44 PM	GET	http://r11.o.lencr.org/.hg	400	Bad Request	295 ms	159 bytes	0 bytes
86	2/20/25, 3:34:45 PM	2/20/25, 3:34:45 PM	GET	http://r11.o.lencr.org/.bz	400	Bad Request	302 ms	159 bytes	0 bytes
				http://r11.o.lencr.org/vim_dars...	400	Bad Request	288 ms	159 bytes	0 bytes

• Analyze the Results

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

The screenshot shows a web-based security analysis interface. At the top, there are tabs for History, Search, Alerts, Output, Spider, Active Scan, Progress, and a plus sign for new scans. Below the tabs, there's a sidebar with icons for Home, Scan, New Scan, and Help.

Alerts (2)

- X-Content-Type-Options Header Missing** (Selected)
 - URL: <http://r11.o.lencr.org/>
 - Risk: Low
 - Confidence: Medium
 - Parameter: x-content-type-options
 - Attack:
 - Evidence:
 - CWE ID: 693
 - WASC ID: 15
 - Source: Passive (10021 - X-Content-Type-Options Header Missing)
 - Input Vector:
 - Description: The Anti-MIME-Sniffing header X-Content-Type-Options was not set to 'nosniff'. This allows older versions of Internet Explorer and Chrome to perform MIME-sniffing on the response body, potentially causing the response body to be interpreted and displayed as a content type other than the declared content type. Current (early 2014) and legacy versions of Firefox will use the declared content type (if one is set), rather than performing MIME-sniffing.
 - Other Info: This issue still applies to error type pages (401, 403, 500, etc.) as those pages are often still affected by injection issues, in which case there is still concern for browsers sniffing pages away from their actual content type. At "High" threshold this scan rule will not alert on client or server error responses.
 - Solution: Ensure that the application/web server sets the Content-Type header appropriately, and that it sets the X-Content-Type-Options header to 'nosniff' for all web pages.
- Tech Detected - Nginx**
 - URL: <http://r11.o.lencr.org/>
 - Risk: Informational
 - Confidence: Medium
 - Parameter:
 - Attack:
 - Evidence: nginx
 - CWE ID:
 - WASC ID: 13
 - Source: Tool (10004)
 - Input Vector:
 - Description: The following "Web servers, Reverse proxies" technology was identified: Nginx.
 - Described as: Nginx is a web server that can also be used as a reverse proxy, load balancer, mail proxy and HTTP cache.
 - Other Info: The following CPE is associated with the identified tech: cpe:2.3:a:f5:nginx:.*:.*:.*:.*:*
 - Solution:

Main Proxy: localhost:8080

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!'

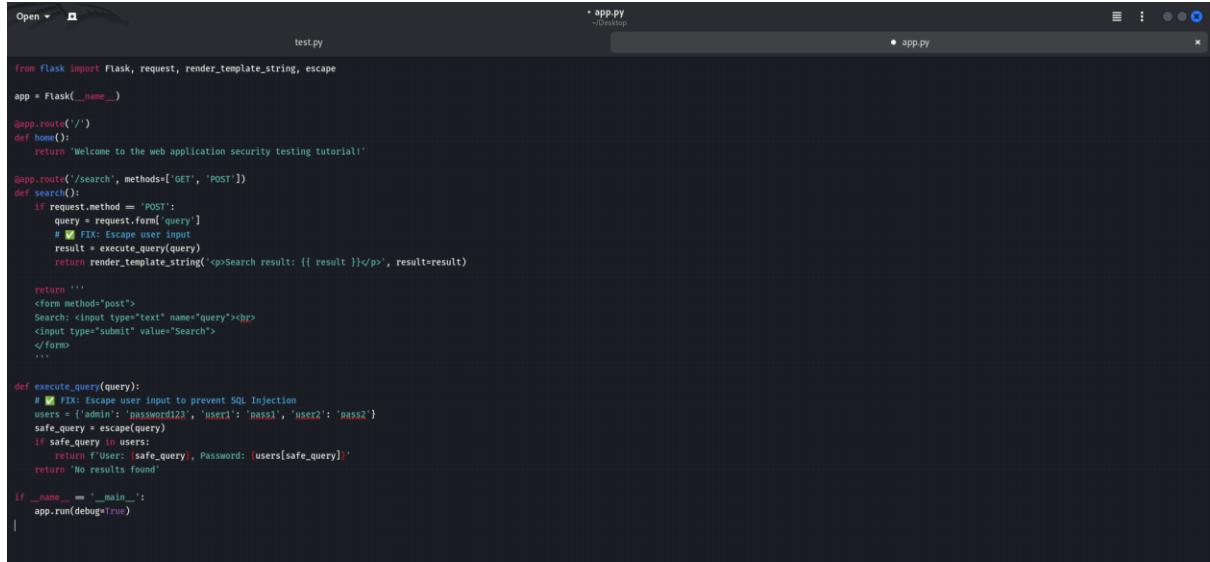
@app.route('/search', methods=['GET', 'POST'])
def search():
    if request.method == 'POST':
        query = request.form['query']
        # ✅ FIX: Escape user input
        result = execute_query(query)
        return render_template_string('<p>Search result: {{ result }}</p>', result=result)

    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)
```



```
Open ▾ testpy app.py ~Desktop app.py

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:

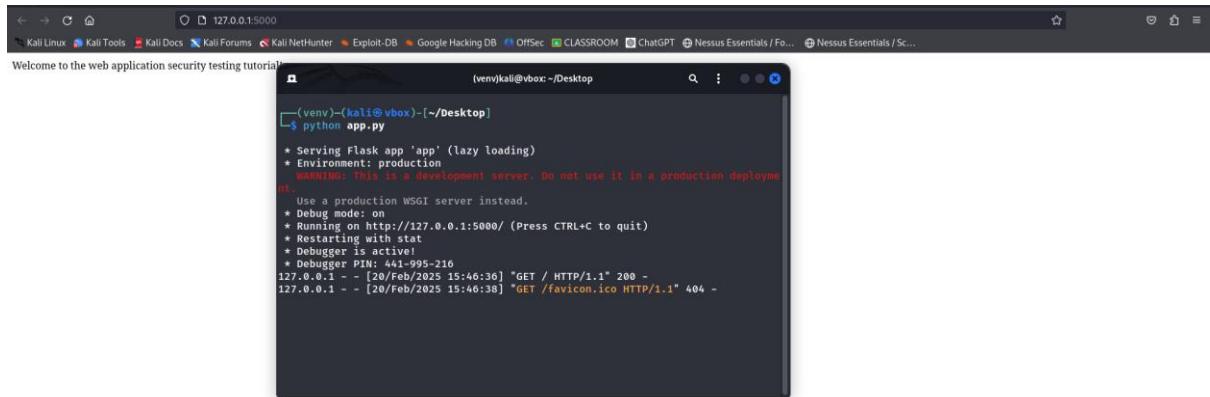
- ✓ Used escape(query) to prevent SQL injection.

Restart the Application and Re-Test

1. Stop the Flask app (CTRL + C).

2. Restart it:

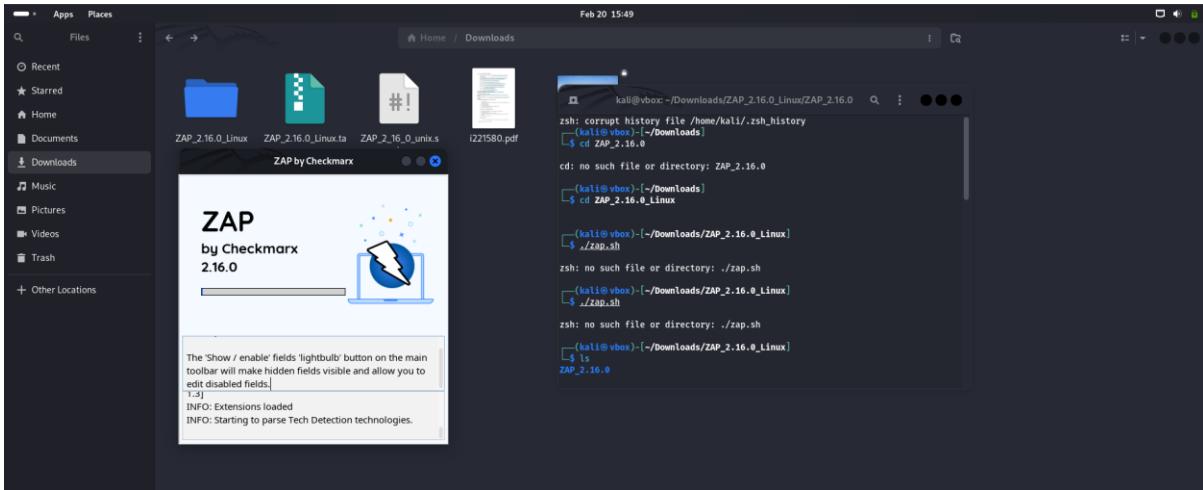
```
python app.py
```



```
← → ⌂ 127.0.0.1:5000
Kali Linux Kali Tools Kali Docs Kali Forums Exploit-DB Google Hacking DB OffSec CLASSROOM ChatGPT Nessus Essentials / Fo... Nessus Essentials / Sc...
Welcome to the web application security testing tutorial!
(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 deployment!
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
 * Debugger is active!
 * Debugger PIN: 441-995-216
127.0.0.1 - - [20/Feb/2025 15:46:36] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [20/Feb/2025 15:46:38] "GET /favicon.ico HTTP/1.1" 404 -
```

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).