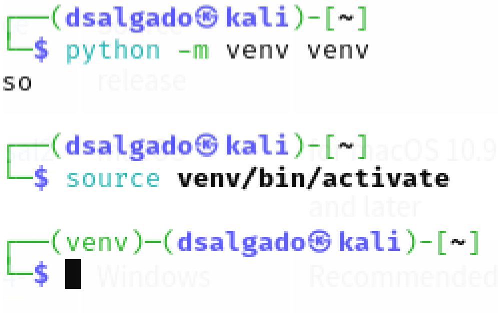
Web Application Security Testing

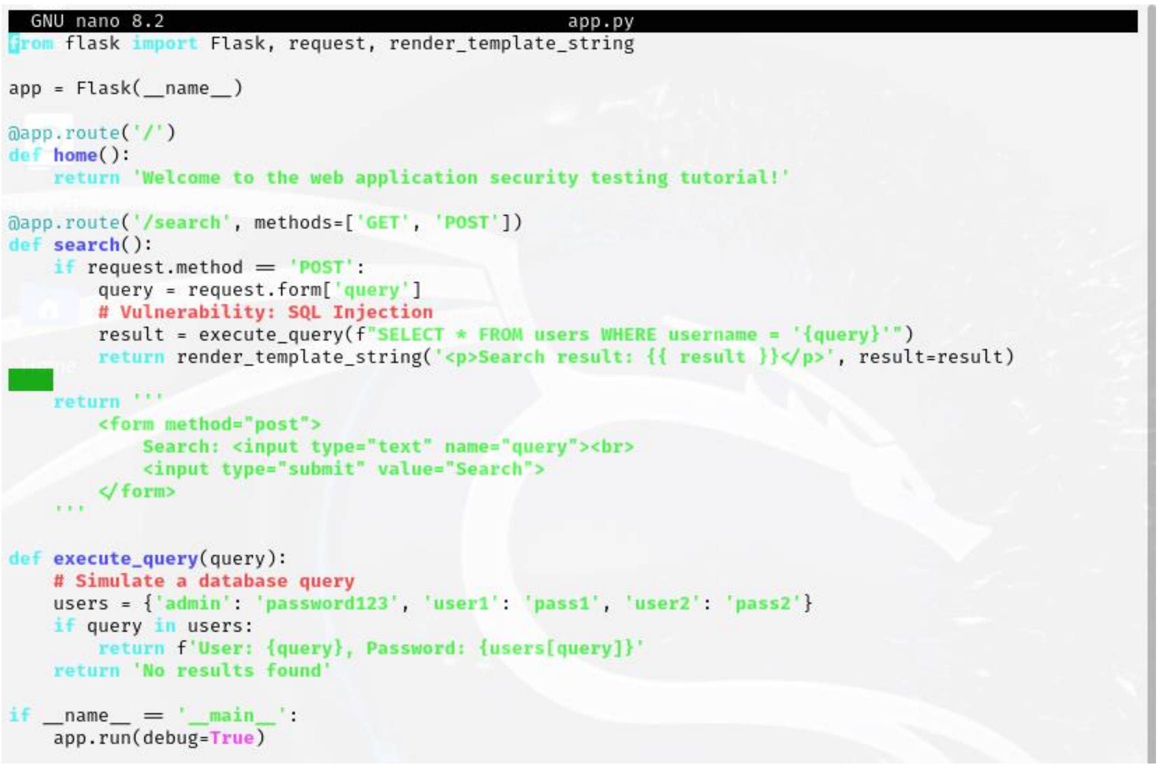
In this project, I will use OWASP ZAP (Zed Attack Proxy) to find and fix vulnerabilities in a web application.

1. I need to create a virtual environment to isolate the project dependencies



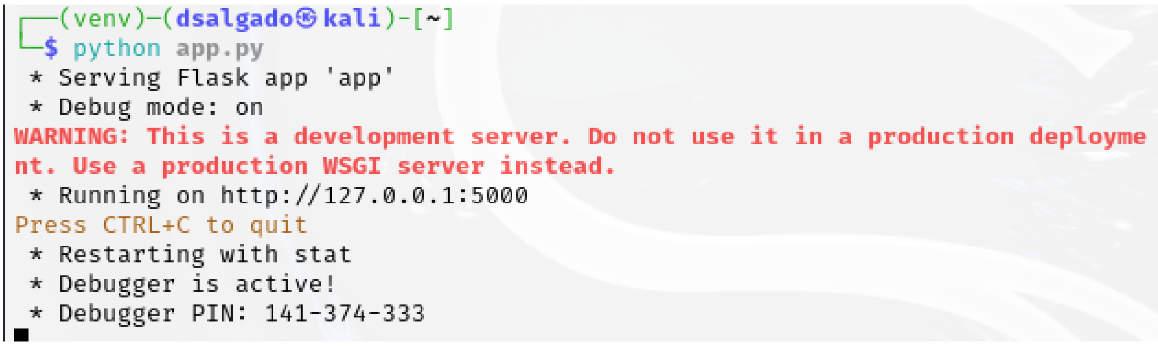
```
┌──(dsalgado㉿kali)-[~]
└─$ python -m venv venv
so

┌──(dsalgado㉿kali)-[~]
└─$ source venv/bin/activate

┌──(venv)─(dsalgado㉿kali)-[~]
└─$ █
```

2. Create application file (app.py) and install dependencies of Flask



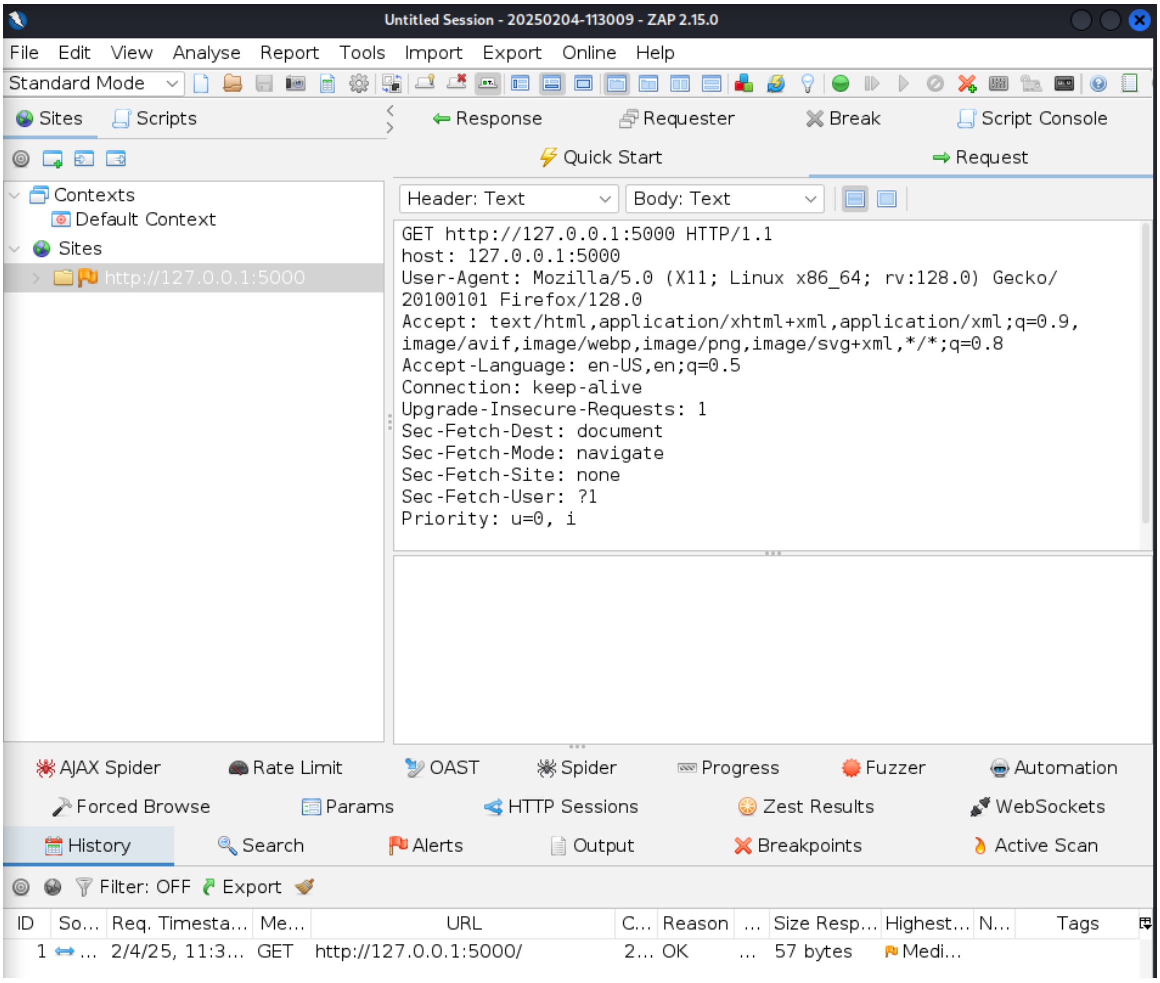
```
  GNU nano 8.2                                       app.py
from flask import Flask, request, render_template_string

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):
    # Simulate a 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)
```

3. Run the app



```
┌──(venv)─(dsalgado㉿kali)-[~]
└─$ python app.py
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployme
nt. 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: 141-374-333
█
```

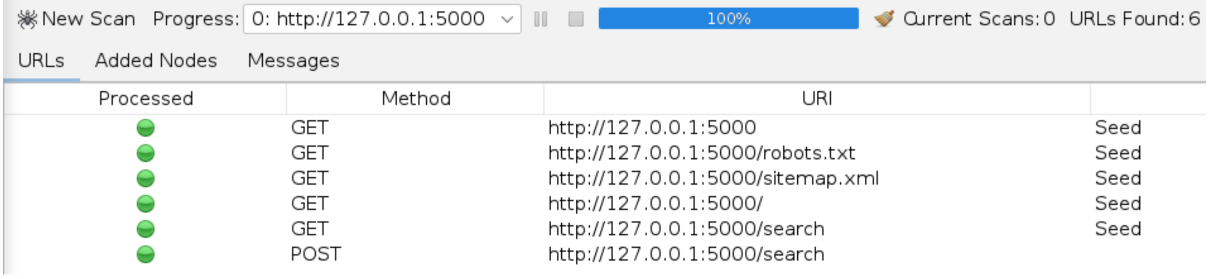
4. Download and Start OWAS ZAP, open my app with http://127.0.0.1:5000. The ZAP will capture the traffic.



```
GET http://127.0.0.1:5000 HTTP/1.1
host: 127.0.0.1:5000
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/
20100101 Firefox/128.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,
image/avif,image/webp,image/png,image/svg+xml,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Connection: keep-alive
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: none
Sec-Fetch-User: ?1
Priority: u=0, i
```

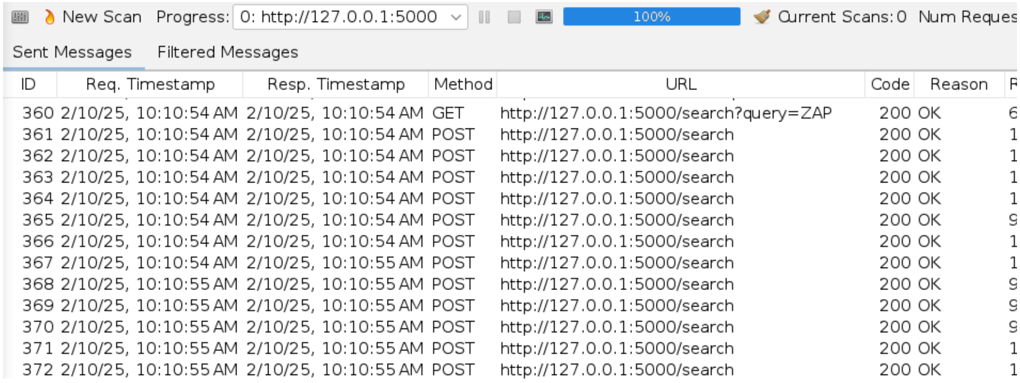
ID	So…	Req. Timesta…	Me…	URL	C…	Reason	…	Size Resp…	Highest…	N…	Tags
1	↔ …	2/4/25, 11:3…	GET	http://127.0.0.1:5000/	2…	OK	…	57 bytes	Medi…		

5. I performed a SPIDER attack on my app:


New Scan Progress: 0: http://127.0.0.1:5000 100% Current Scans: 0 URLs Found: 6

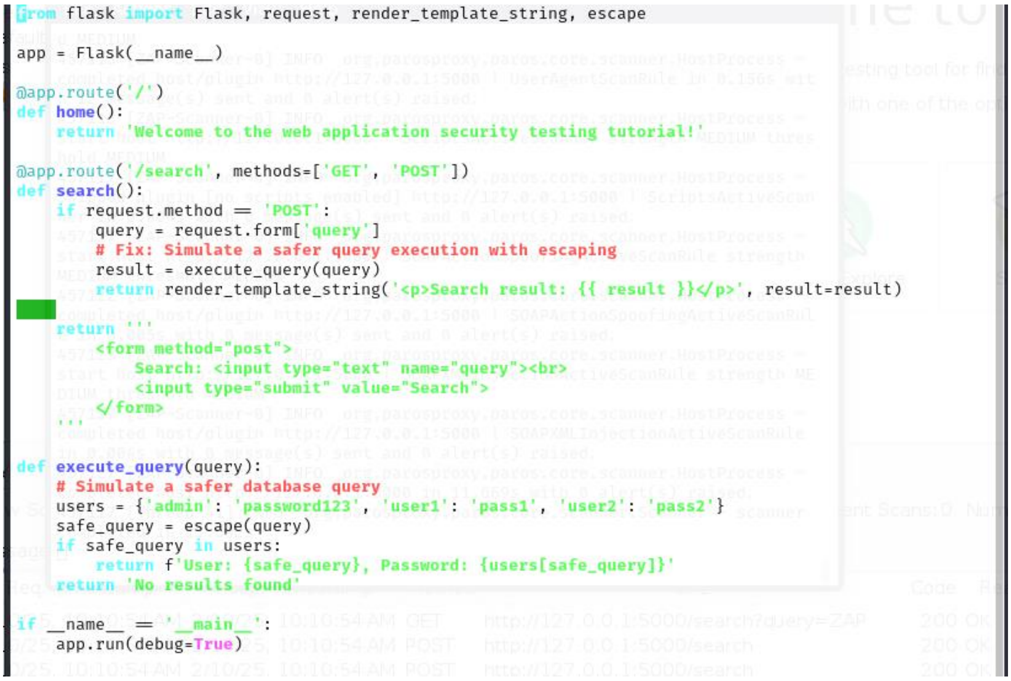
Processed	Method	URI	
●	GET	http://127.0.0.1:5000	Seed
●	GET	http://127.0.0.1:5000/robots.txt	Seed
●	GET	http://127.0.0.1:5000/sitemap.xml	Seed
●	GET	http://127.0.0.1:5000/	Seed
●	GET	http://127.0.0.1:5000/search	Seed
●	POST	http://127.0.0.1:5000/search	

6. Afterwards I made an Active Scan which proves that the site is vulnerable to SQL Injection



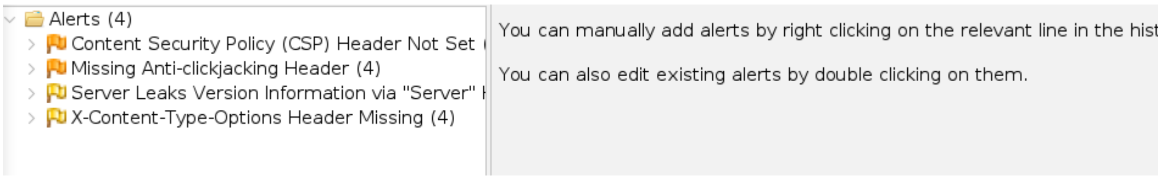
ID	Req. Timestamp	Resp. Timestamp	Method	URL	Code	Reason	R
360	2/10/25, 10:10:54 AM	2/10/25, 10:10:54 AM	GET	http://127.0.0.1:5000/search?query=ZAP	200	OK	6
361	2/10/25, 10:10:54 AM	2/10/25, 10:10:54 AM	POST	http://127.0.0.1:5000/search	200	OK	1
362	2/10/25, 10:10:54 AM	2/10/25, 10:10:54 AM	POST	http://127.0.0.1:5000/search	200	OK	1
363	2/10/25, 10:10:54 AM	2/10/25, 10:10:54 AM	POST	http://127.0.0.1:5000/search	200	OK	1
364	2/10/25, 10:10:54 AM	2/10/25, 10:10:54 AM	POST	http://127.0.0.1:5000/search	200	OK	1
365	2/10/25, 10:10:54 AM	2/10/25, 10:10:54 AM	POST	http://127.0.0.1:5000/search	200	OK	9
366	2/10/25, 10:10:54 AM	2/10/25, 10:10:54 AM	POST	http://127.0.0.1:5000/search	200	OK	1
367	2/10/25, 10:10:54 AM	2/10/25, 10:10:55 AM	POST	http://127.0.0.1:5000/search	200	OK	1
368	2/10/25, 10:10:55 AM	2/10/25, 10:10:55 AM	POST	http://127.0.0.1:5000/search	200	OK	9
369	2/10/25, 10:10:55 AM	2/10/25, 10:10:55 AM	POST	http://127.0.0.1:5000/search	200	OK	9
370	2/10/25, 10:10:55 AM	2/10/25, 10:10:55 AM	POST	http://127.0.0.1:5000/search	200	OK	9
371	2/10/25, 10:10:55 AM	2/10/25, 10:10:55 AM	POST	http://127.0.0.1:5000/search	200	OK	1
372	2/10/25, 10:10:55 AM	2/10/25, 10:10:55 AM	POST	http://127.0.0.1:5000/search	200	OK	1

7. This is the code to fix the vulnerability:



```python
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: Simulate a safer query execution with escaping
        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):
    # Simulate a safer database query
    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)
```

8. With the new code there is not an alert regarding the SQL Injection:



Alerts (4)
- Content Security Policy (CSP) Header Not Set
- Missing Anti-clickjacking Header (4)
- Server Leaks Version Information via "Server" H
- X-Content-Type-Options Header Missing (4)

You can manually add alerts by right clicking on the relevant line in the hist

You can also edit existing alerts by double clicking on them.

Conclusion:

This project demonstrated the process of web application security testing using OWASP ZAP. By setting up a simple Flask application with an intentional SQL injection vulnerability, we explored how automated scanning tools effectively detect security flaws. The results highlighted the importance of identifying and mitigating vulnerabilities through secure coding practices, such as parameterized queries and input validation. This project provided valuable insights into web security testing methodologies and reinforced the need for proactive security measures in web application development.