# Detailed Steps for Building a Web Application Vulnerability Scanner

## Objective:

Develop a Python-based scanner that identifies common web vulnerabilities, including Cross-Site Scripting (**XSS), SQL Injection (SQLi), and Cross-Site Request Forgery (CSRF),** by crawling web pages, injecting attack payloads, analyzing responses, and presenting a Flask web interface for viewing scan results and reports.

## Part 1: Setting Up the Project Directory

### Step 1: Open Terminal and Create a Project Folder

**Command:** mkdir ~/Vulnerability_scanner

cd ~/Vulnerability_scanner

### Step 2: Set Up a Virtual Environment (Optional but Recommended)

**Command:** sudo apt install python3-venv -y

python3 -m venv venv

source venv/bin/activate

## Part 2: Install Required Python Packages

**Command:** pip install requests beautifulsoup4 flask

You can save this in a requirements.txt file:

echo -e "requests\nbeautifulsoup4\nflask" > requirements.txt

pip install -r requirements.txt

## Part 3: Create Project Files

### Step 1: File Structure

```
Vulnerability_scanner/
├── app.py
├── scanner.py
├── templates/
│   └── index.html
├── static/
│   └── style.css (optional)
└── logs/
    └── scan_results.json
```

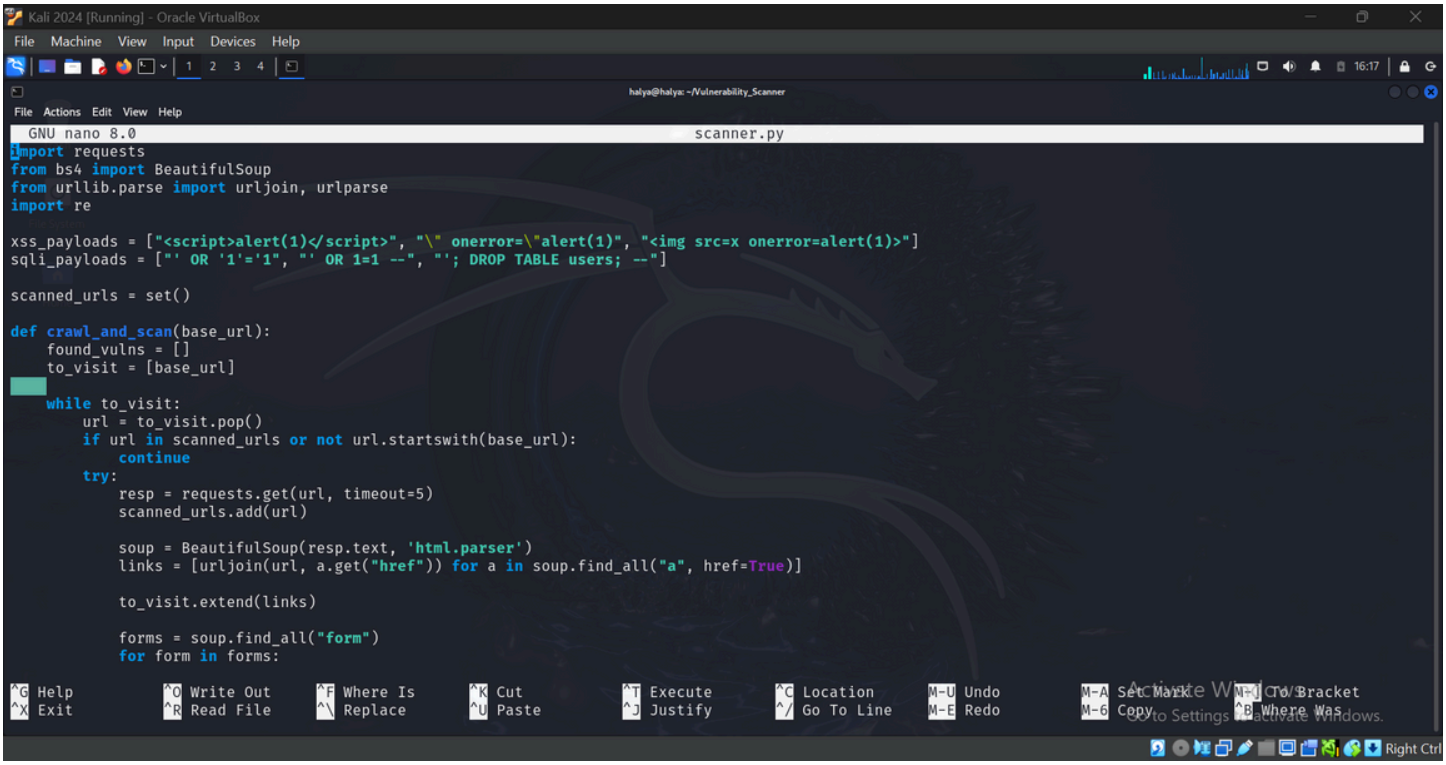
### Step 2: Create scanner.py

**Command:** nano scanner.py

Paste this **sample scanning logic**:

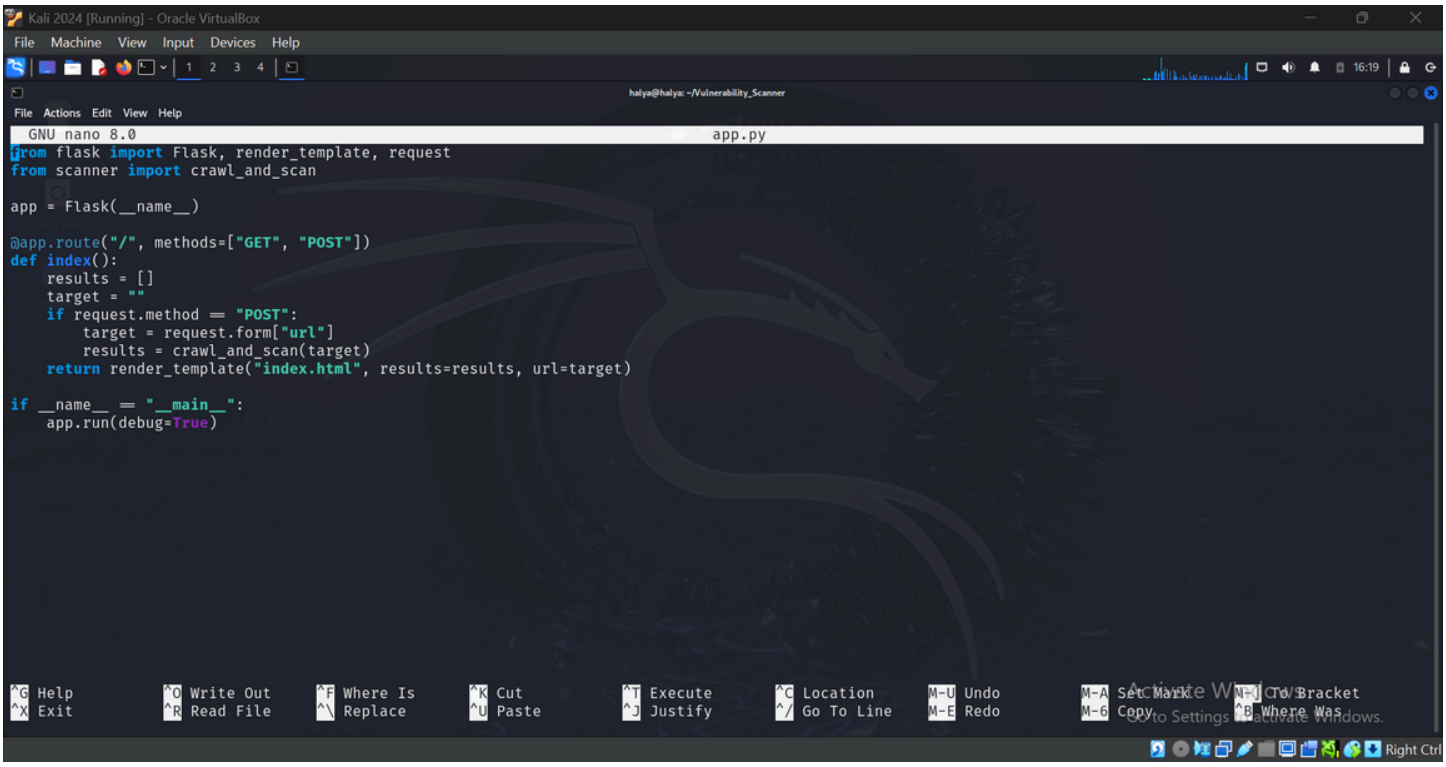
```
import requests
from bs4 import BeautifulSoup
from urllib.parse import urljoin, urlparse
import re

xss_payloads = ["<script>alert(1)</script>", "\" onerror=\"alert(1)", "<img src=x
onerror=alert(1)>"]
sqli_payloads = ["' OR '1'='1", "' OR 1=1 --", "'; DROP TABLE users; --"]

scanned_urls = set()

def crawl_and_scan(base_url):
    found_vulns = []
    to_visit = [base_url]

    while to_visit:
        url = to_visit.pop()
        if url in scanned_urls or not url.startswith(base_url):
            continue
        try:
            resp = requests.get(url, timeout=5)
            scanned_urls.add(url)

            soup = BeautifulSoup(resp.text, 'html.parser')
            links = [urljoin(url, a.get("href")) for a in soup.find_all("a", href=True)]

            to_visit.extend(links)

            forms = soup.find_all("form")
            for form in forms:
                action = urljoin(url, form.get("action"))
                method = form.get("method", "get").lower()
                inputs = form.find_all("input")

                for payload in xss_payloads + sqli_payloads:
                    data = {}
```

```
        for input in inputs:
            name = input.get("name")
            if name:
                data[name] = payload

        if method == "post":
            res = requests.post(action, data=data)
        else:
            res = requests.get(action, params=data)

        if payload in res.text:
            vuln_type = "XSS" if payload in xss_payloads else "SQL Injection"
            found_vulns.append({
                "url": action,
                "payload": payload,
                "type": vuln_type,
                "severity": "High" if vuln_type == "XSS" else "Medium"
            })
    except Exception as e:
        continue
return found_vulns
```




**Step 3: Create app.py (Flask Web Interface)**

**Command:** nano app.py




**Step 4: Create HTML Template**

**Command:** mkdir templates

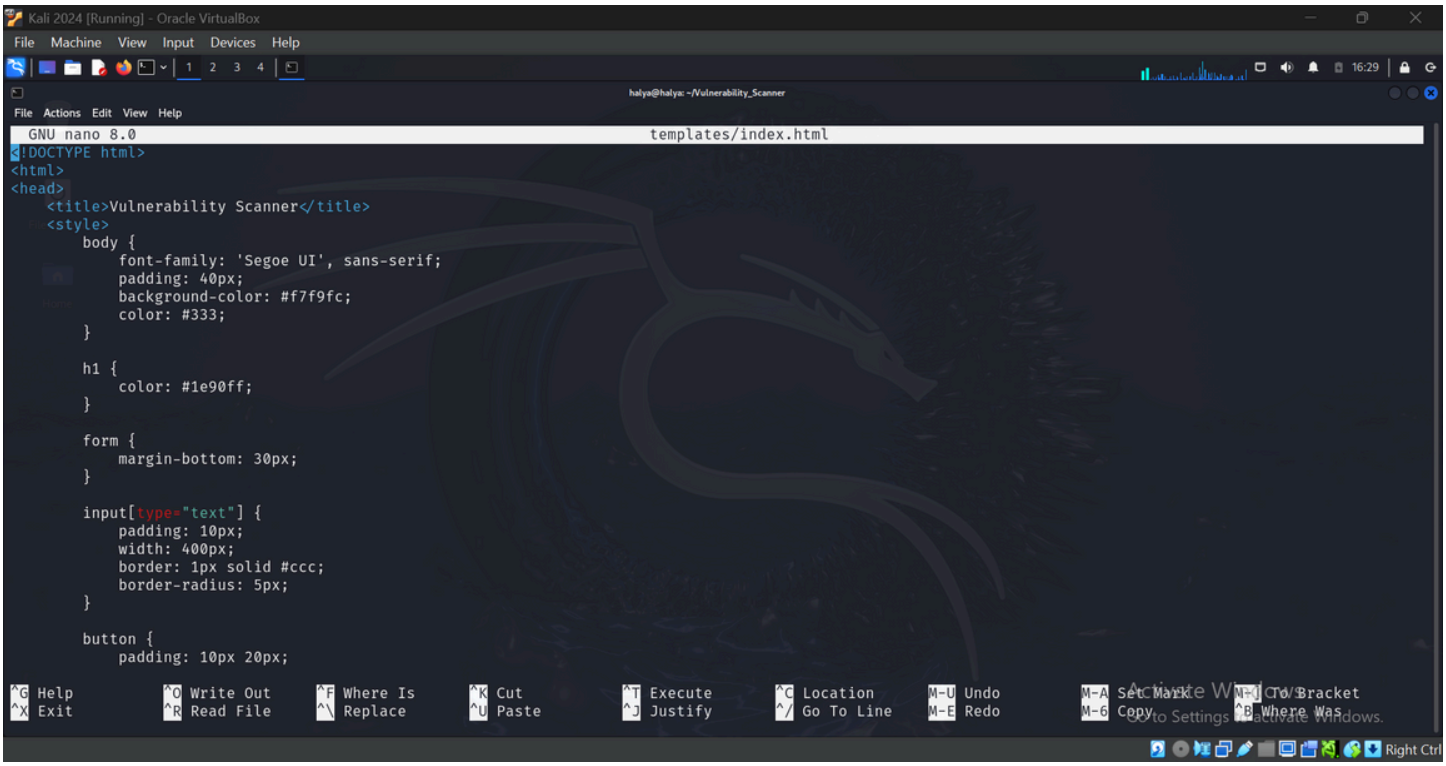
nano templates/index.html

```
<!DOCTYPE html>
<html>
<head>
  <title>Vulnerability Scanner</title>
  <style>
    body {
      font-family: 'Segoe UI', sans-serif;
      padding: 40px;
      background-color: #f7f9fc;
      color: #333;
    }

    h1 {
      color: #1e90ff;
    }

    form {
      margin-bottom: 30px;
    }

    input[type="text"] {
```

```
    padding: 10px;
    width: 400px;
    border: 1px solid #ccc;
    border-radius: 5px;
}

button {
    padding: 10px 20px;
    border: none;
    background-color: #1e90ff;
    color: white;
    border-radius: 5px;
    cursor: pointer;
}

button:hover {
    background-color: #0d6efd;
}

table {
    width: 100%;
    border-collapse: collapse;
    margin-top: 20px;
    background-color: white;
    border-radius: 5px;
    overflow: hidden;
    box-shadow: 0 2px 5px rgba(0, 0, 0, 0.1);
}

th, td {
    padding: 12px;
    text-align: left;
    border-bottom: 1px solid #eee;
}

th {
    background-color: #f2f2f2;
    color: #333;
}
```

```
      .severity-High {
        color: white;
        background-color: #e74c3c; /* Red */
        font-weight: bold;
      }

      .severity-Medium {
        color: black;
        background-color: #f39c12; /* Orange */
        font-weight: bold;
      }

      .severity-Low {
        color: black;
        background-color: #f0e68c; /* Light Yellow */
      }
    </style>
</head>
<body>
    <h1>  Automated Web App Vulnerability Scanner</h1>
    <form method="POST">
      <input type="text" name="url" placeholder="https://example.com" required>
      <button type="submit">Start Scan</button>
    </form>

    {% if results %}
    <h2>Scan Results for <span style="color: #1e90ff;">{{ url }}</span></h2>
    <table>
      <tr>
        <th>Type</th>
        <th>Payload</th>
        <th>Target URL</th>
        <th>Severity</th>
      </tr>
      {% for r in results %}
      <tr>
        <td>{{ r.type }}</td>
        <td><code>{{ r.payload }}</code></td>
        <td><a href="{{ r.url }}" target="_blank">{{ r.url }}</a></td>
        <td class="severity-{{ r.severity }}">{{ r.severity }}</td>
      </tr>
```

```
    </tr>
    {% endfor %}
  </table>
  {% endif %}
</body>
</html>
```



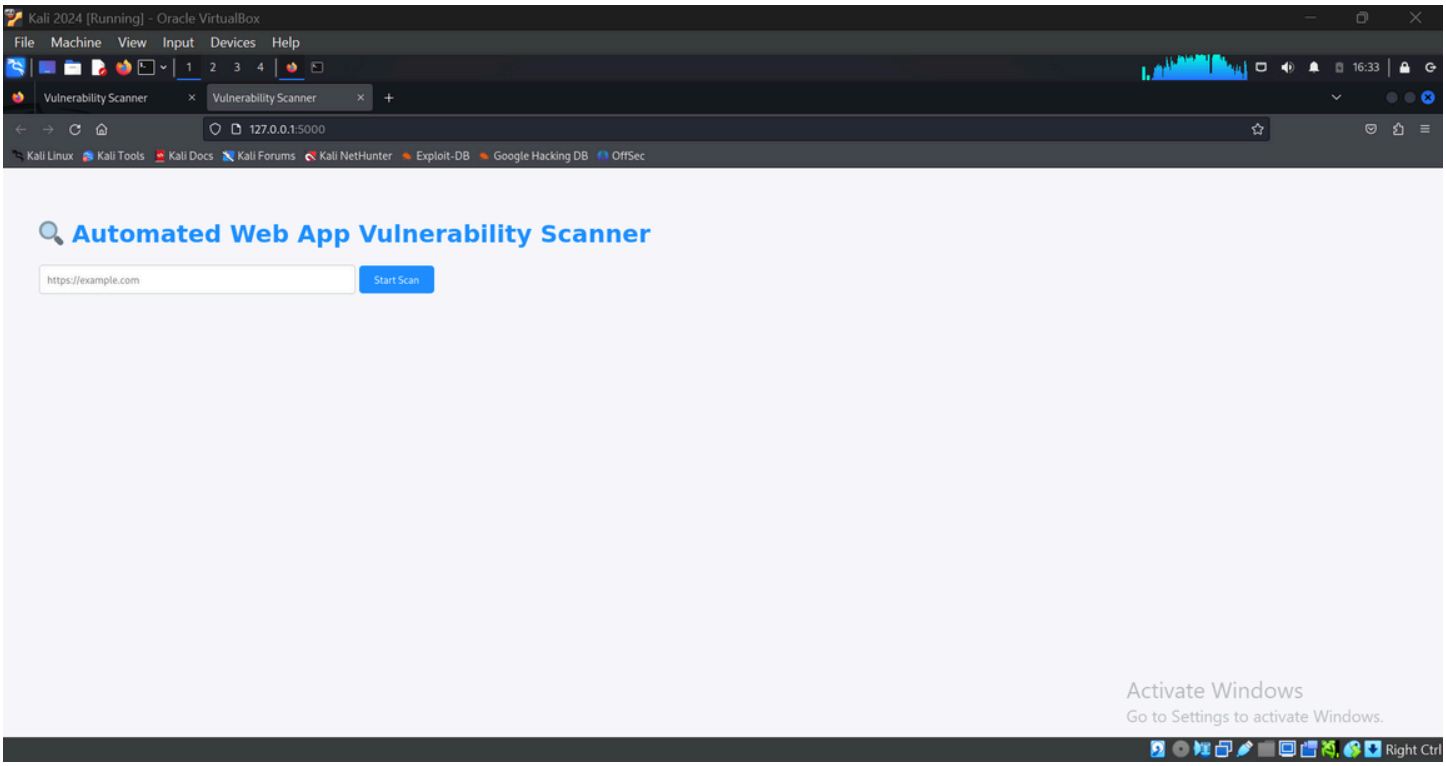

## Part 4: Run the Application

### Run the Flask App

**Command:** python3 app.py

## Open in Browser

Go to: http://127.0.0.1:5000




## Part 5: Save and Organize Scripts on Kali Linux

1. Save the folder Vulnerability_scanner in your home directory: **~/Vulnerability_Scanner**
2. To run anytime:

**Command:** cd ~/Vulnerability_scanner

source venv/bin/activate

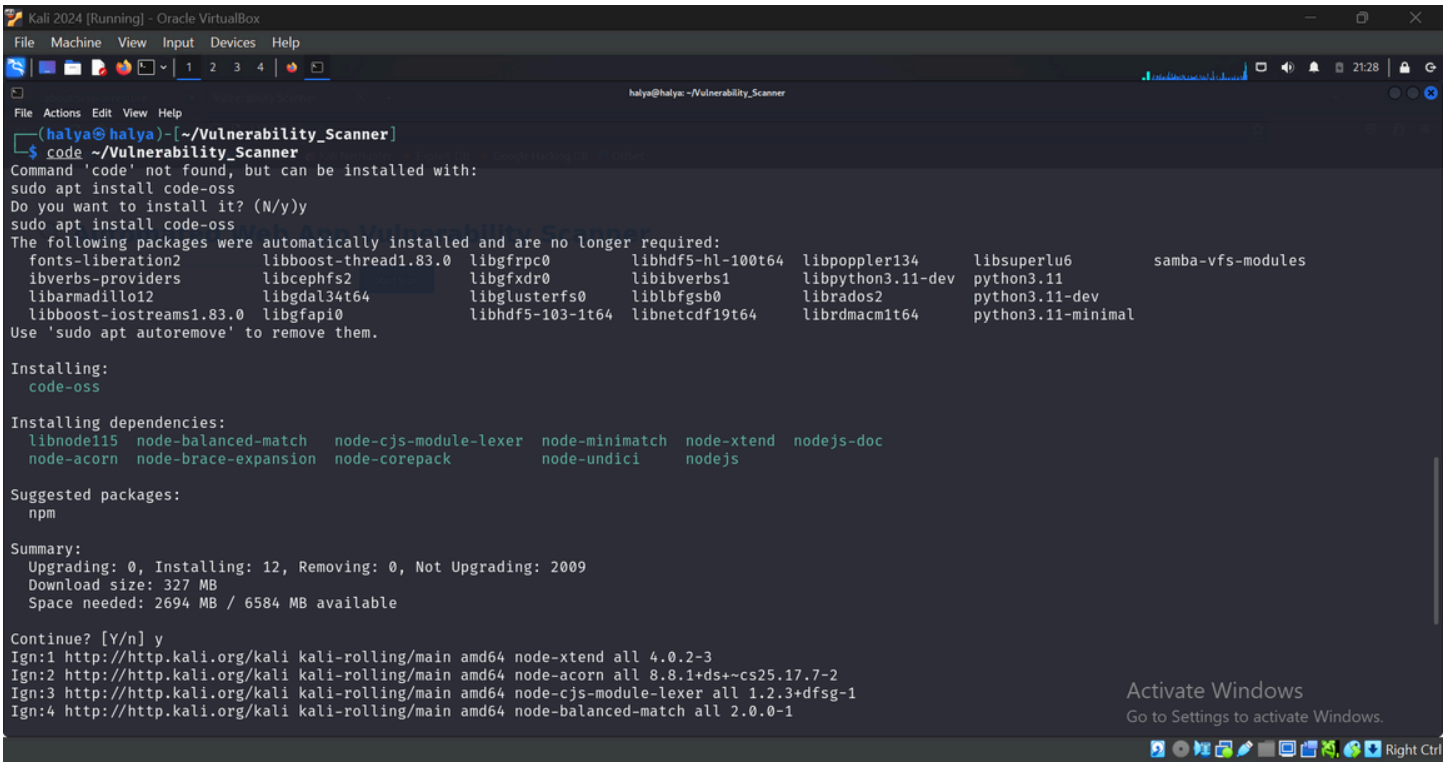
python3 app.py

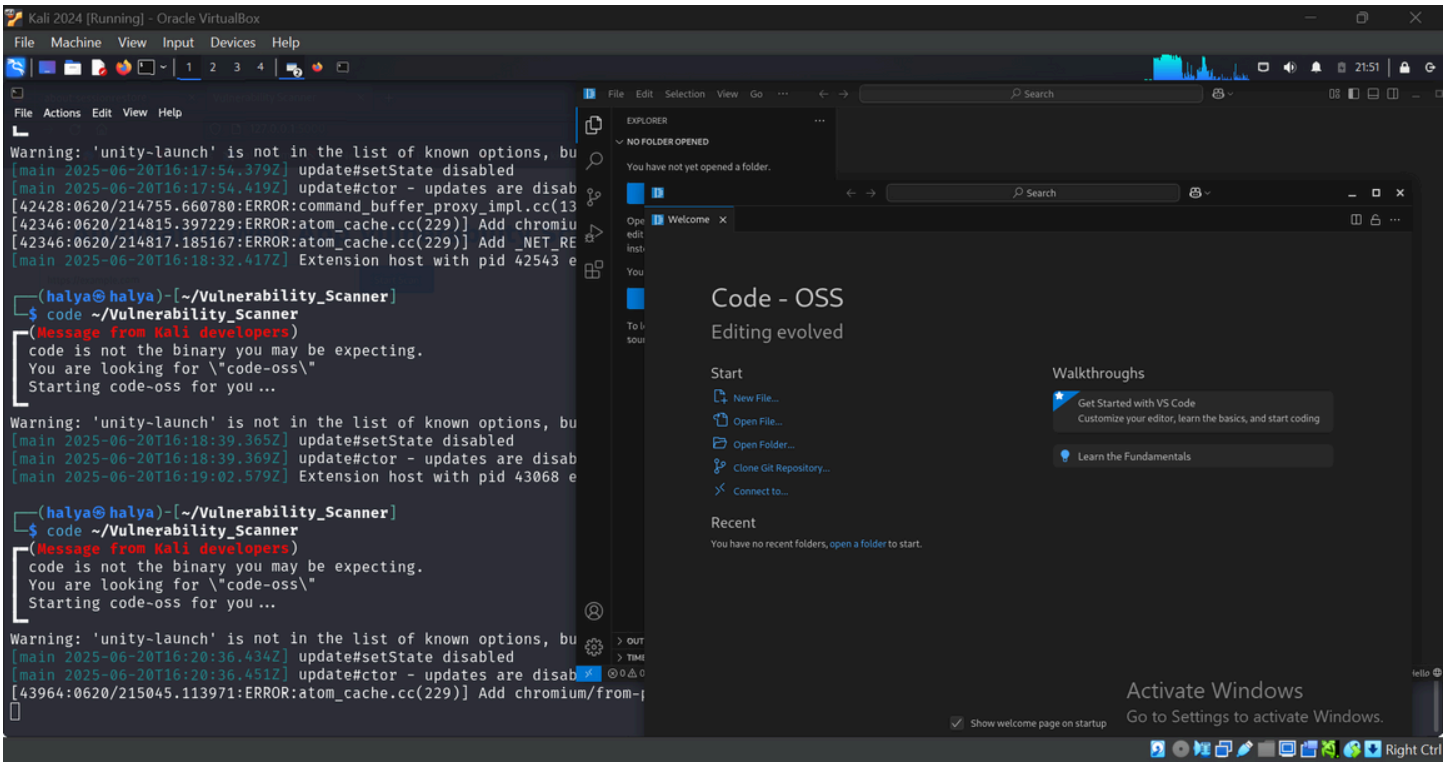
```
┌──(halya㉿halya)-[~]
└─$ cd Vulnerability_Scanner
source venv/bin/activate
python3 app.py

source: no such file or directory: venv/bin/activate
 * 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 stat
 * Debugger is active!
 * Debugger PIN: 265-160-238
127.0.0.1 - - [20/Jun/2025 16:54:47] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [20/Jun/2025 16:54:48] "GET /favicon.ico HTTP/1.1" 404 -
^C
```

**To edit files:**

- Use nano or open VSCode:

**Command:** code ~/Vulnerability_scanner




## Conclusion – Web Application Vulnerability Scanner

The **Web Application Vulnerability Scanner** project effectively demonstrates how to detect common web vulnerabilities such as **Cross-Site Scripting (XSS)** and **SQL Injection (SQLi)** using Python, web crawling, payload injection, and response analysis. By automating the identification of security flaws, this tool aids in reducing the manual effort required in penetration testing.

The addition of a user-friendly **Flask web interface** enhances usability, allowing users to input target URLs, launch scans, and view detailed results, including evidence and severity. The

scanner adheres to security best practices outlined in the **OWASP Top 10**, providing a solid foundation for more advanced security tools.

This project is ideal for learners and security professionals aiming to:

- Understand basic web vulnerabilities.
- Gain hands-on experience with ethical hacking.
- Practice secure coding and testing in a controlled environment.

Future improvements could include:

- CSRF token analysis,
- Authentication handling,
- Multithreading for better performance,
- Enhanced report generation and UI.

By developing and running this scanner on **Kali Linux**, users gain real-world exposure to offensive security techniques while reinforcing responsible ethical practices in cybersecurity.