



WEBSCAN PRO

– AUTOMATED WEB APPLICATION
SECURITY TESTING TOOL

The Problem

- Web applications are highly vulnerable to attacks such as SQL Injection, XSS, and IDOR due to poor validation and access control.

Solution

WebScanPro is an automated security testing tool that scans a given URL, detects vulnerabilities, and generates a structured security report.

Objectives

- Detect common vulnerabilities (OWASP Top 10)
- Automate penetration testing
- Provide mitigation suggestions
- Generate visual security reports

Tools and test platforms

- **Languages:** Python
- **Libraries:** Requests, BeautifulSoup, Streamlit.
- **Testing Platforms:**
 - DVWA
 - OWASP Juice Shop
 - bWAPP
- **Output:** HTML/PDF Security Report

Week-1

Project Initialization & Setup

Problem

Manual security testing is time-consuming and inconsistent.

Approach

Defined project goals

Studied OWASP Top vulnerabilities

Set up vulnerable applications locally

Week-1

Solution Implemented

Installed DVWA / Juice Shop using XAMPP or Docker

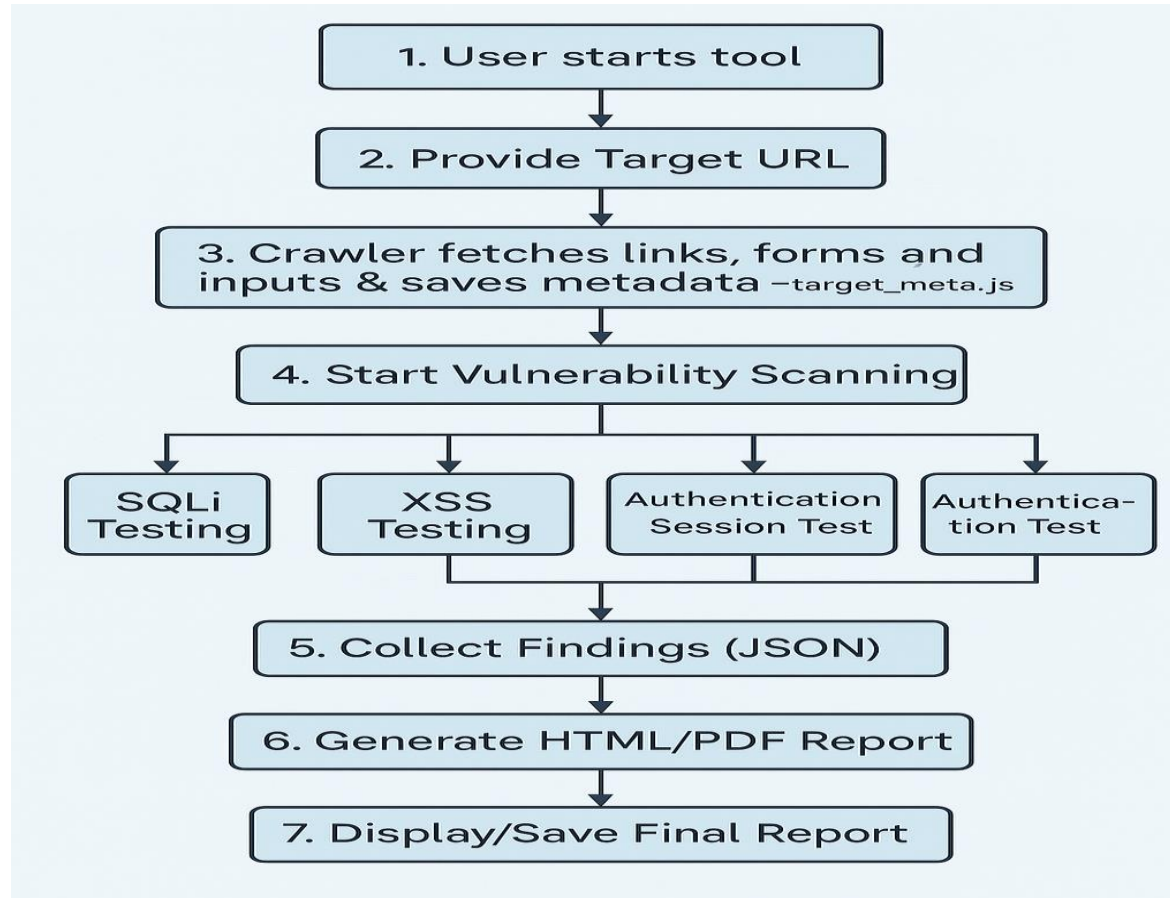
Explored pages, forms, URLs, parameters

Output

Working vulnerable web app environment

Identified attack surfaces

Workflow:



Week-2

Use tools like BeautifulSoup or Selenium to extract interactive elements.

Problem

- Security testing requires knowing all entry points (forms, links, inputs).

Approach

- Web crawling
- HTML parsing

Solution Implemented

- Developed crawler using **BeautifulSoup**

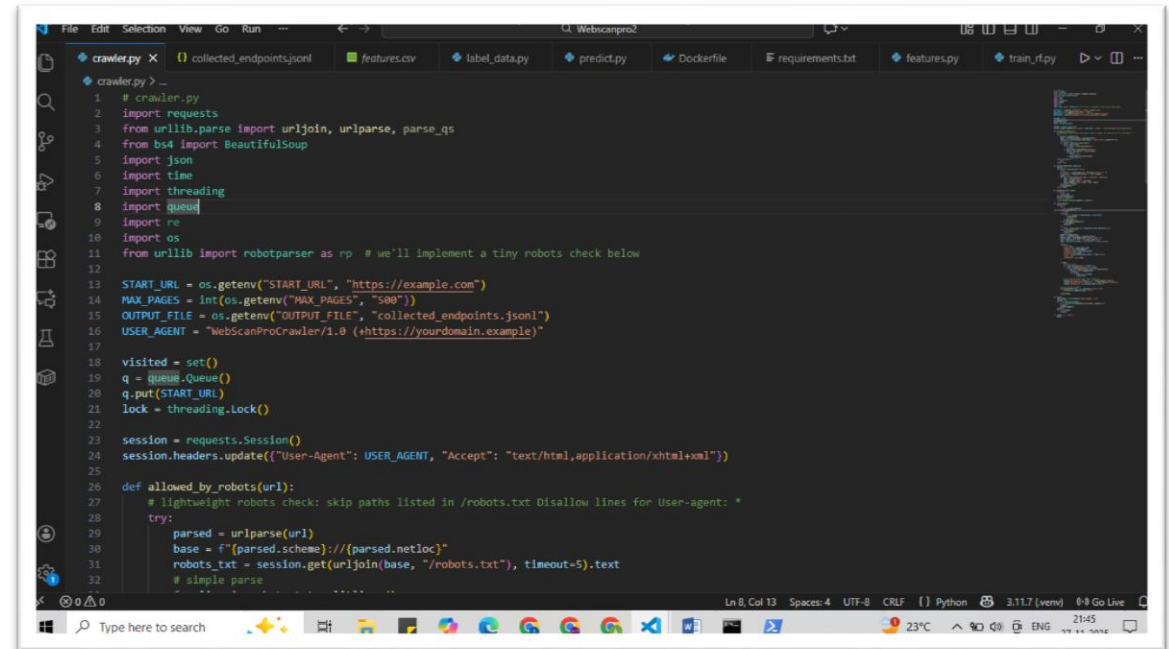
Week-2

Extracted:

- URLs
- Forms
- Input fields
- Parameters

Output

Structured metadata of target website



```
1 # crawler.py
2 import requests
3 from urllib.parse import urljoin, urlparse, parse_qs
4 from bs4 import BeautifulSoup
5 import json
6 import time
7 import threading
8 import queue
9 import re
10 import os
11 from urllib import robotparser as rp # we'll implement a tiny robots check below
12
13 START_URL = os.getenv("START_URL", "https://example.com")
14 MAX_PAGES = int(os.getenv("MAX_PAGES", "500"))
15 OUTPUT_FILE = os.getenv("OUTPUT_FILE", "collected_endpoints.jsonl")
16 USER_AGENT = "WebScanProCrawler/1.0 (+https://yourdomain.example)"
17
18 visited = set()
19 q = queue.Queue()
20 q.put(START_URL)
21 lock = threading.Lock()
22
23 session = requests.Session()
24 session.headers.update({"User-Agent": USER_AGENT, "Accept": "text/html,application/xhtml+xml"})
25
26 def allowed_by_robots(url):
27     # lightweight robots check: skip paths listed in /robots.txt Disallow lines for User-agent: *
28     try:
29         parsed = urlparse(url)
30         base = f"{parsed.scheme}://{parsed.netloc}"
31         robots_txt = session.get(urljoin(base, "/robots.txt"), timeout=5).text
32         # simple parse
```


Week-3

SQL Injection Testing

Problem

Improper input validation leads to SQL Injection attacks.

Approach

- Payload-based testing
- Error response analysis

Solution Implemented

- Injected SQL payloads (' OR 1=1 --)
- Analyzed server responses
- Detected vulnerable endpoints

Output

- SQL Injection findings
- Severity classification
- Fix suggestions (Prepared Statements)

The image shows a Windows 11 desktop environment. The primary focus is the Visual Studio Code (VS Code) editor, which is open to a file named 'features.csv'. The file contains a list of 32 URLs, each preceded by a line number from 1 to 32. The URLs are related to various OWASP (Open Web Application Security Project) resources, including project pages, blog posts, and security tools. The VS Code interface includes a sidebar on the left with icons for Explorer, Search, and Run and Debug. The top of the editor shows the file explorer with several open files, including 'features.csv'. The Windows taskbar at the bottom displays the Start button, a search bar, and several pinned applications: File Explorer, Edge, VS Code, and a terminal. The system tray on the right shows the date and time as 22:10 on 27-11-2025.

Week-4

Cross-Site Scripting (XSS) Testing

Problem

XSS allows attackers to execute malicious scripts in user browsers.

Approach

- JavaScript payload injection

- DOM & response inspection

Solution Implemented

- Injected XSS payloads (`<script>alert()</script>`)

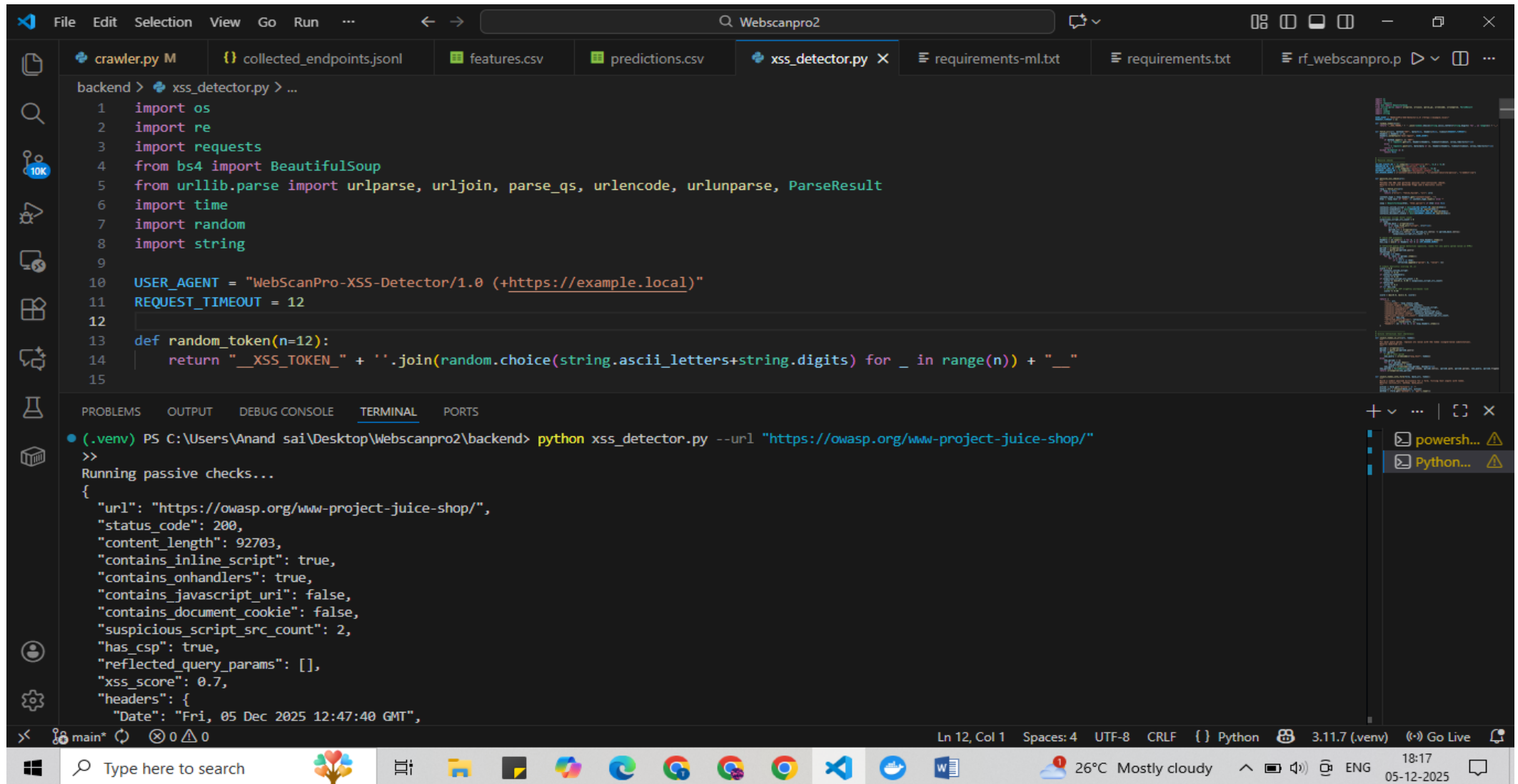
- Detected reflected & stored XSS

Output

- Vulnerable input fields identified

- Mitigation tips (Input sanitization, CSP)

Output:



The image shows a Visual Studio Code editor window with a Python script named `xss_detector.py` open. The script is located in a directory named `backend`. The script imports `os`, `re`, `requests`, `BeautifulSoup` from `bs4`, and `urllib.parse` (aliased as `urlparse`), `urljoin`, `parse_qs`, `urlencode`, `urlunparse`, and `ParseResult`. It also imports `time`, `random`, and `string`. The script defines a `USER_AGENT` string and a `REQUEST_TIMEOUT` of 12. A function `random_token(n=12)` is defined to generate a random token. The terminal output shows the command `python xss_detector.py --url "https://owasp.org/www-project-juice-shop/"` being executed, which results in a JSON output containing various security-related metrics for the specified URL.

```
backend > xss_detector.py > ...
1 import os
2 import re
3 import requests
4 from bs4 import BeautifulSoup
5 from urllib.parse import urlparse, urljoin, parse_qs, urlencode, urlunparse, ParseResult
6 import time
7 import random
8 import string
9
10 USER_AGENT = "WebScanPro-XSS-Detector/1.0 (+https://example.local)"
11 REQUEST_TIMEOUT = 12
12
13 def random_token(n=12):
14     return "__XSS_TOKEN__" + ''.join(random.choice(string.ascii_letters+string.digits) for _ in range(n)) + "__"
15
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
(.venv) PS C:\Users\Anand sai\Desktop\Webscanpro2\backend> python xss_detector.py --url "https://owasp.org/www-project-juice-shop/"
>>
Running passive checks...
{
  "url": "https://owasp.org/www-project-juice-shop/",
  "status_code": 200,
  "content_length": 92703,
  "contains_inline_script": true,
  "contains_onhandlers": true,
  "contains_javascript_uri": false,
  "contains_document_cookie": false,
  "suspicious_script_src_count": 2,
  "has_csp": true,
  "reflected_query_params": [],
  "xss_score": 0.7,
  "headers": {
    "Date": "Fri, 05 Dec 2025 12:47:40 GMT",

```

Ln 12, Col 1 Spaces: 4 UTF-8 CRLF Python 3.11.7 (.venv) Go Live

Type here to search 26°C Mostly cloudy 18:17 05-12-2025

Week-5

Authentication & Session Testing

Problem

Weak authentication leads to account compromise.

Approach

Credential testing

Session analysis

Solution Implemented

Tested weak/default credentials

Checked cookies & session fixation

Simulated brute-force attempts

Secure session recommendations

The screenshot shows a VS Code editor with a Python script named `auth_session_audit.py` open. The script is part of a project named `Webscanpro2`. The script imports various libraries and defines configuration variables. The code is as follows:

```

import argparse
import requests
import time
import csv
import json
import os
import re
import math
import random
from collections import Counter
from urllib.parse import urljoin, urlparse
from http.cookies import SimpleCookie
from datetime import datetime
import pandas as pd
import numpy as np
from sklearn.ensemble import IsolationForest, RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
import joblib
import warnings
warnings.filterwarnings("ignore")

# ----- Configuration -----
OUTPUT_DIR = "audit_output"
os.makedirs(OUTPUT_DIR, exist_ok=True)
LOG_CSV = os.path.join(OUTPUT_DIR, "attempts_log.csv")
RESULT_JSON = os.path.join(OUTPUT_DIR, "results.json")
TOKEN_MODEL = os.path.join(OUTPUT_DIR, "token_rf.pkl")
ANOMALY_MODEL = os.path.join(OUTPUT_DIR, "anomaly_iforest.pkl")

# Small safe lists (keep tiny to avoid heavy testing)
COMMON_WEAK_CREDS = [

```

Week-6

Access Control & IDOR Testing

Problem

- Unauthorized users accessing restricted resources.

Approach

- Parameter manipulation
- Role-based testing

Solution Implemented

- Modified user IDs in URLs
- Tested horizontal & vertical privilege escalation

Output

- IDOR vulnerabilities detected
- Access control fixes (RBAC, ABAC)

Output:

The screenshot shows a Visual Studio Code editor window with a project named 'Webscanpro2'. The Explorer sidebar on the left lists various files, including 'access_control_analyzer.py'. The main editor area displays the code for 'access_control_analyzer.py', which is a Python script for testing credentials and analyzing access control issues. The script includes imports for 'requests', 'json', 'pandas', 'difflib', 'datetime', and 'os'. It defines 'BASE_URL' as 'http://localhost:3000' and 'LOGIN_API' as '/rest/user/login'. The script also sets 'OUTPUT_DIR' to 'access_control_logs' and creates the directory if it doesn't exist. A comment indicates that the following code is for testing credentials in a lab environment.

```
1 # access_control_analyzer.py (AUTO LOGIN VERSION)
2
3 import requests
4 import json
5 import pandas as pd
6 from difflib import SequenceMatcher
7 from datetime import datetime
8 import os
9
10 BASE_URL = "http://localhost:3000"
11 LOGIN_API = "/rest/user/login"
12
13 OUTPUT_DIR = "access_control_logs"
14 os.makedirs(OUTPUT_DIR, exist_ok=True)
15
16 # Test credentials (LAB ONLY)
```

The terminal window at the bottom shows the execution of the script. It starts with a message indicating that login failed due to incorrect credentials. Then, it shows the script logging in as 'Admin' and 'User'. A green checkmark indicates that tokens were obtained successfully. The terminal then displays a table of access control issues found, with columns for 'time', 'endpoint', 'issue', 'severity', and 'issue severity'.

```
OUTPUT DEBUG CONSOLE TERMINAL PORTS
X Login failed. Check credentials.
(.venv) PS C:\Users\Anand sai\Desktop\Webscanpro2> python access_control_analyzer.py
[*] Logging in as Admin...
[*] Logging in as User...
[+] Tokens obtained successfully
Access Control Issues Found:
      time                endpoint  issue severity
0  2025-12-16T17:03:46.171285  /rest/user/whoami  Vertical Privilege Escalation  High
1  2025-12-16T17:03:46.171285  /rest/user/whoami  Horizontal Privilege Escalation  Medium
2  2025-12-16T17:03:48.029349  /rest/admin/application-version  Vertical Privilege Escalation  High
3  2025-12-16T17:03:48.029349  /rest/admin/application-version  Horizontal Privilege Escalation  Medium
(.venv) PS C:\Users\Anand sai\Desktop\Webscanpro2>
```


Week-7

Security Report Generation

Problem

- Raw vulnerability data is hard to understand.

Approach

- Structured reporting
- Severity classification

Solution Implemented

- Generated report with:
- Vulnerability Type
- Affected Endpoint
- Severity (Low / Medium / High)
- Mitigation

Output

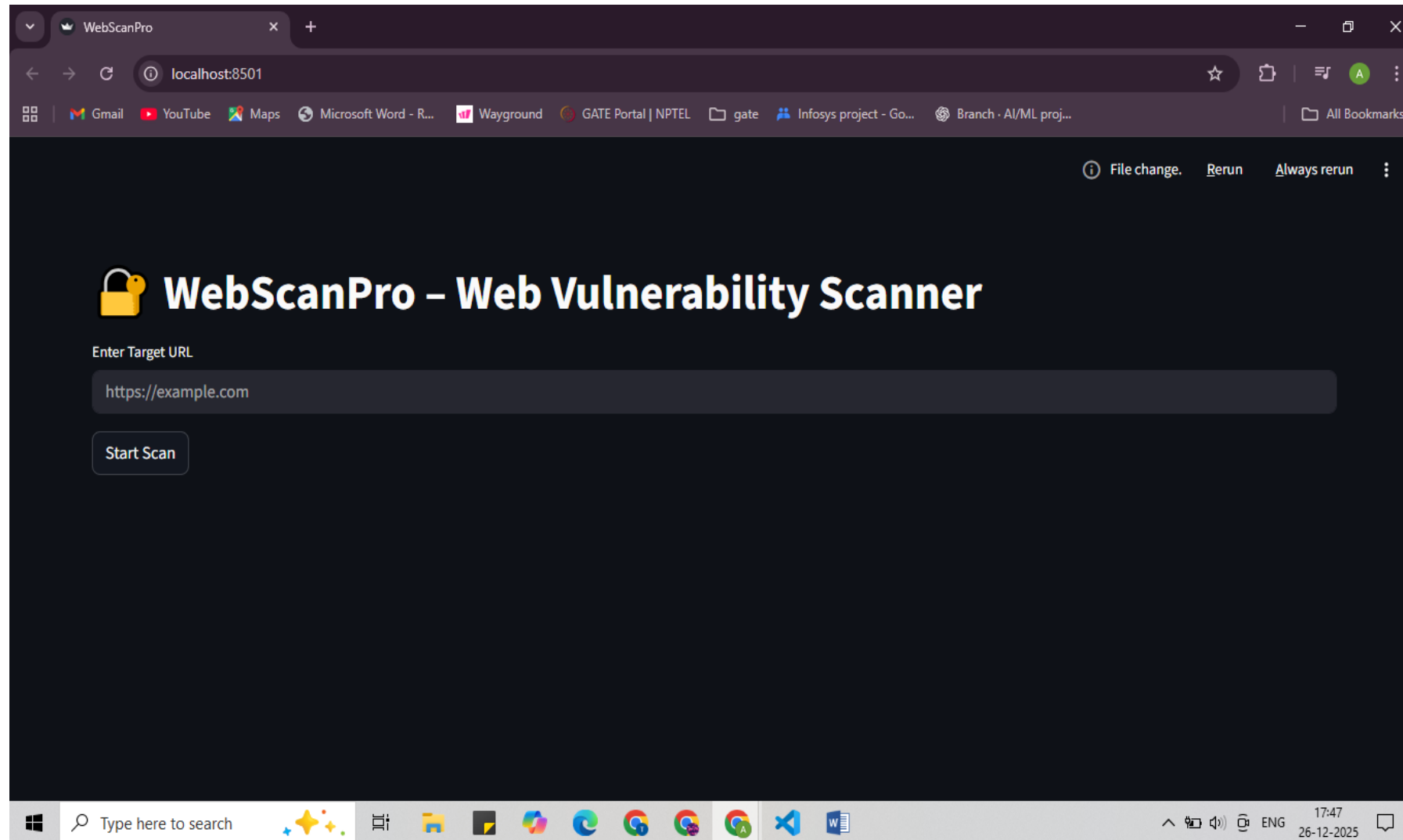
- HTML / PDF security report
- Visual summaries

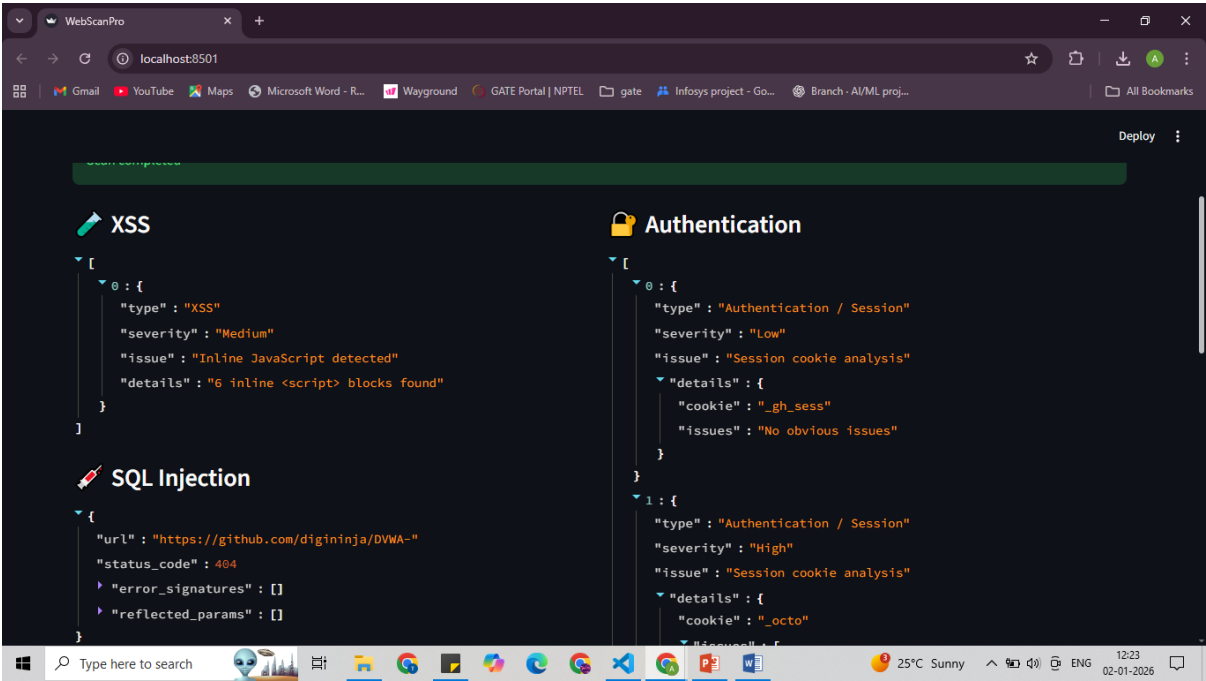
The screenshot displays a Visual Studio Code editor window titled 'Webscanpro2'. The editor contains a Python file named 'app.py' with the following code:

```
backend > app.py > ...
1 import os
2 import sys
3
4 # PATH FIX
5 PROJECT_ROOT = os.path.abspath(os.path.join(os.path.dirname(__file__), ".."))
6 sys.path.append(PROJECT_ROOT)
7
8 import streamlit as st
9 from reports.report_generator import generate_report_html
10 #IMPORT SCANNERS
11 from scanners.xss_detector import passive_xss_check
12 from scanners.sqli_detector import passive_sqli_check
13 from scanners.access_control_analyzer import access_control_check
14 from scanners.auth_session_audit import auth_session_check
15
16 def get_severity(item):
17     if isinstance(item, dict):
18         return item.get("severity", "Low")
19     return "Low"
20
21 #NORMALIZATION
22 def normalize_results(url, xss, sqli, auth, access):
23     findings = []
24
25     for item in xss:
```

Below the code editor, the Streamlit interface is visible, showing the message: "You can now view your Streamlit app in your browser. Network URL: <http://192.168.29.233:8501>". The bottom status bar indicates the file is 'main*', the editor is at 'Ln 21, Col 2', and the environment is 'Python 3.11.7 (venv)' running on 'Port: 3001'.

Final Output:





Generated Report

Vulnerabilites detected

WebScanPro Security Report

Scan Date: 2025-12-26 15:24:42

Summary

- Total vulnerabilities: 9
- High: 1
- Medium: 1
- Low: 7

Detailed Findings

Vulnerability	Affected Endpoint	Severity	Suggested Mitigation
Cross-Site Scripting (XSS)	https://github.com/digininja/DVWA-	Medium	Apply output encoding, input validation, and Content Security Policy (CSP)
SQL Injection	https://github.com/digininja/DVWA-	Low	Use parameterized queries and avoid dynamic SQL
SQL Injection	https://github.com/digininja/DVWA-	Low	Use parameterized queries and avoid dynamic SQL
SQL Injection	https://github.com/digininja/DVWA-	Low	Use parameterized queries and avoid dynamic SQL
SQL Injection	https://github.com/digininja/DVWA-	Low	Use parameterized queries and avoid dynamic SQL
Authentication / Session Management	https://github.com/digininja/DVWA-	Low	Use Secure and HttpOnly cookies, proper session expiry, and strong authentication
Authentication / Session Management	https://github.com/digininja/DVWA-	High	Use Secure and HttpOnly cookies, proper session expiry, and strong authentication
Authentication / Session Management	https://github.com/digininja/DVWA-	Low	Use Secure and HttpOnly cookies, proper session expiry, and strong authentication
Access Control	https://github.com/digininja/DVWA-	Low	Implement role-based access control and server-side authorization checks

THANK YOU