# Web Application Vulnerability Scanner

## 1. Introduction

Web applications are increasingly exposed to various cyber threats due to improper input validation and insecure coding practices. Vulnerabilities like **Cross-Site Scripting (XSS)** and **SQL Injection (SQLi)** are among the most critical issues listed in the **OWASP Top 10**. To combat these threats, penetration testers and developers require simple tools to identify flaws during development or testing phases.

This project aims to develop a **lightweight Web Vulnerability Scanner** using **Python and Flask** to identify potential XSS and SQLi vulnerabilities in form-based web applications.

## 2. Abstract

The Web Application Vulnerability Scanner is an educational cybersecurity tool designed to scan a given website URL for input forms and assess them for potential **XSS** and **SQL Injection** vulnerabilities. The application uses **BeautifulSoup** and **requests** to parse forms and submit payloads. A **Flask-based UI** allows users to input a URL and view results in the browser. Logs are saved with timestamped entries and severity levels.
This scanner is designed to aid beginners and ethical hackers in understanding common input-based vulnerabilities and how to detect them in real-time.

## 3. Tools & Technologies Used

- Python: Core programming language
- Flask: Lightweight web framework
- BeautifulSoup: HTML parsing
- Requests: Sending HTTP payloads and form submissions
- OWASP Top 10: Reference checklist
- HTML/CSS: Frontend for UI

## 4. Steps Involved

**Step 1: User Input via UI**

- The user provides a URL to be scanned on the web interface (http://127.0.0.1:5000).

**Step 2: Form Extraction**

- All forms are extracted from the given URL using BeautifulSoup.
- Input fields (text, submit, etc.) are identified.

**Step 3: Vulnerability Testing**

- For each form:
  - XSS Test: Injects a payload like <script>alert('XSS')</script> into text fields.
  - SQLi Test: Injects payloads like ' OR '1'='1 and ';--.

**Step 4: Response Analysis**

- The application analyzes server responses:
  - If injected payload appears in the response → Potential **XSS**.
  - If error messages like “SQL syntax error” appear → Possible **SQLi**.

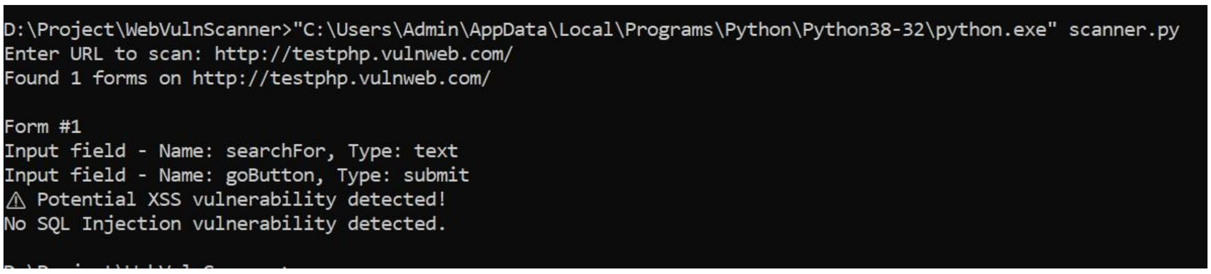
**Step 5: Logging**

- Results are displayed on-screen and logged in a file **scan_log.txt** with:
  - Timestamp
  - URL
  - Vulnerability type

    - Severity
    - Evidence (payload and response)

## 5. Sample Output – Screenshots

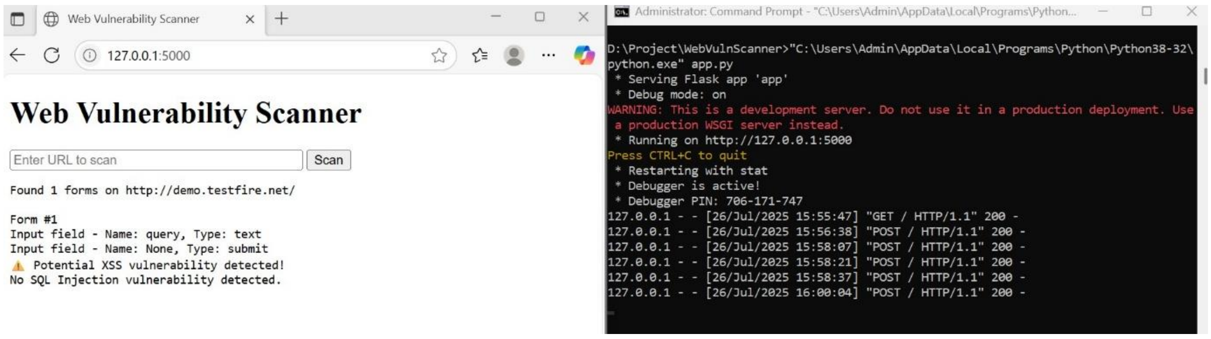
**Figure 1: Vulnerability detection**



```
D:\Project\WebVulnScanner>"C:\Users\Admin\AppData\Local\Programs\Python\Python38-32\python.exe" scanner.py
Enter URL to scan: http://testphp.vulnweb.com/
Found 1 forms on http://testphp.vulnweb.com/

Form #1
Input field - Name: searchFor, Type: text
Input field - Name: goButton, Type: submit
⚠ Potential XSS vulnerability detected!
No SQL Injection vulnerability detected.
```

**Figure 2: Web Vulnerability Scanner – Flask**



Web Vulnerability Scanner
127.0.0.1:5000
Enter URL to scan
Scan
Administrator: Command Prompt - "C:\Users\Admin\AppData\Local\Programs\Python...

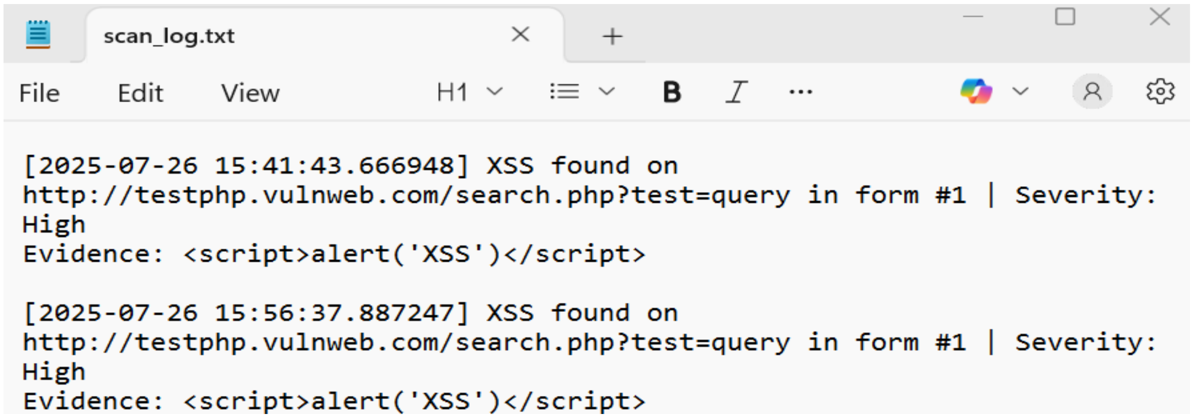
Web Vulnerability Scanner

```
Found 1 forms on http://demo.testfire.net/

Form #1
Input field - Name: query, Type: text
Input field - Name: None, Type: submit
⚠ Potential XSS vulnerability detected!
No SQL Injection vulnerability detected.
```

```
D:\Project\WebVulnScanner>"C:\Users\Admin\AppData\Local\Programs\Python\Python38-32\
python.exe" app.py
 * 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: 706-171-747
127.0.0.1 - - [26/Jul/2025 15:55:47] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [26/Jul/2025 15:56:38] "POST / HTTP/1.1" 200 -
127.0.0.1 - - [26/Jul/2025 15:58:07] "POST / HTTP/1.1" 200 -
127.0.0.1 - - [26/Jul/2025 15:58:21] "POST / HTTP/1.1" 200 -
127.0.0.1 - - [26/Jul/2025 15:58:37] "POST / HTTP/1.1" 200 -
127.0.0.1 - - [26/Jul/2025 16:00:04] "POST / HTTP/1.1" 200 -
```

**Figure 3: log details**



```
[2025-07-26 15:41:43.666948] XSS found on
http://testphp.vulnweb.com/search.php?test=query in form #1 | Severity:
High
Evidence: <script>alert('XSS')</script>

[2025-07-26 15:56:37.887247] XSS found on
http://testphp.vulnweb.com/search.php?test=query in form #1 | Severity:
High
Evidence: <script>alert('XSS')</script>
```

## 7. Conclusion

This project successfully demonstrates a working prototype of a web vulnerability scanner targeting form-based XSS and SQLi flaws. It provides foundational insight into how web application security tools function under the hood. While simple in nature, this project lays the groundwork for further enhancements such as:

- CSRF vulnerability detection
- Advanced payload encoding
- Automated crawling and login simulation
- Exporting results to PDF or CSV