Report: Web Application Vulnerability Scanner

By : Pranav Vijayakumar

Objective

The primary objective of this project is to develop a lightweight, Python-based Web Application Vulnerability Scanner that can identify and report common security vulnerabilities in web applications. The scanner is designed to focus on the OWASP Top 10 vulnerabilities, including **Cross-Site Scripting (XSS)**, **SQL Injection (SQLi)**, and **Cross-Site Request Forgery (CSRF)**. A user-friendly **Flask web interface** provides control over scans, displays results, and generates detailed reports.

Table of Content

Introduction

In today’s digital era, web applications play a vital role in business operations, personal services, and communication. However, with the increasing reliance on web technologies, the surface area for potential cyber-attacks has expanded significantly. Malicious actors constantly exploit vulnerabilities in web applications to gain unauthorized access, steal sensitive data, or disrupt services. As a result, web application security has become a critical concern in software development and deployment.

This project aims to address that concern by developing a **Python-based Web Application Vulnerability Scanner** capable of detecting common security flaws based on the **OWASP Top 10** list, including **Cross-Site Scripting (XSS)**, **SQL Injection (SQLi)**, and **Cross-Site Request Forgery (CSRF)**. The scanner automates the process of crawling websites, injecting known malicious payloads into input fields, and analyzing the responses for signs of vulnerabilities.

The solution integrates a simple and intuitive **Flask-based web interface**, enabling users to initiate scans, monitor progress, and view results through a dashboard. Each identified vulnerability is logged with evidence, affected URL, and severity rating, making it useful for both security researchers and developers to assess and secure their applications proactively.

This tool provides a foundational step toward creating more secure web environments by helping users detect vulnerabilities early in the development or testing phase, thereby reducing the risk of exploitation in production systems.

Tools & Technologies

- **Programming Language:** Python 3.x
- **Libraries:**
  - requests – For sending HTTP requests
  - BeautifulSoup – For parsing and crawling HTML content
  - re (regex) – For pattern-based detection
  - Flask – For creating a web interface
  - logging – For recording scan logs and results
- **Security Reference:** OWASP Top 10 vulnerabilities checklist

Methodology

a. Web Crawler

- The scanner begins by crawling a target URL using requests and BeautifulSoup.
- It collects all internal links and forms from the page recursively.

- Input fields (GET and POST parameters) are extracted for payload injection.

b. Payload Injection

- Predefined payloads for **XSS**, **SQLi**, and **CSRF** are injected into discovered input fields.
- Example payloads:
    - **XSS**: <script>alert(1)</script>
    - **SQLi**: ' OR 1=1 --
- CSRF vulnerabilities are checked by identifying state-changing operations and absence of CSRF tokens.

c. Response Analysis

- The scanner uses regex and pattern matching to analyze responses for signs of vulnerability.
    - XSS detection: Search for reflected payloads in HTML response.
    - SQLi detection: Identify common database error messages.
    - CSRF detection: Look for missing anti-CSRF tokens in forms.

d. Logging & Evidence Collection

- Detected vulnerabilities are logged with:
    - Vulnerability type
    - Affected URL
    - HTTP request and response snippet
    - Severity level (Low/Medium/High)

e. Flask UI

- A web-based interface built using Flask provides:
    - Input field for entering the target URL
    - Start/stop scan control
    - Table view of detected vulnerabilities
    - Option to download a detailed report

System Architecture

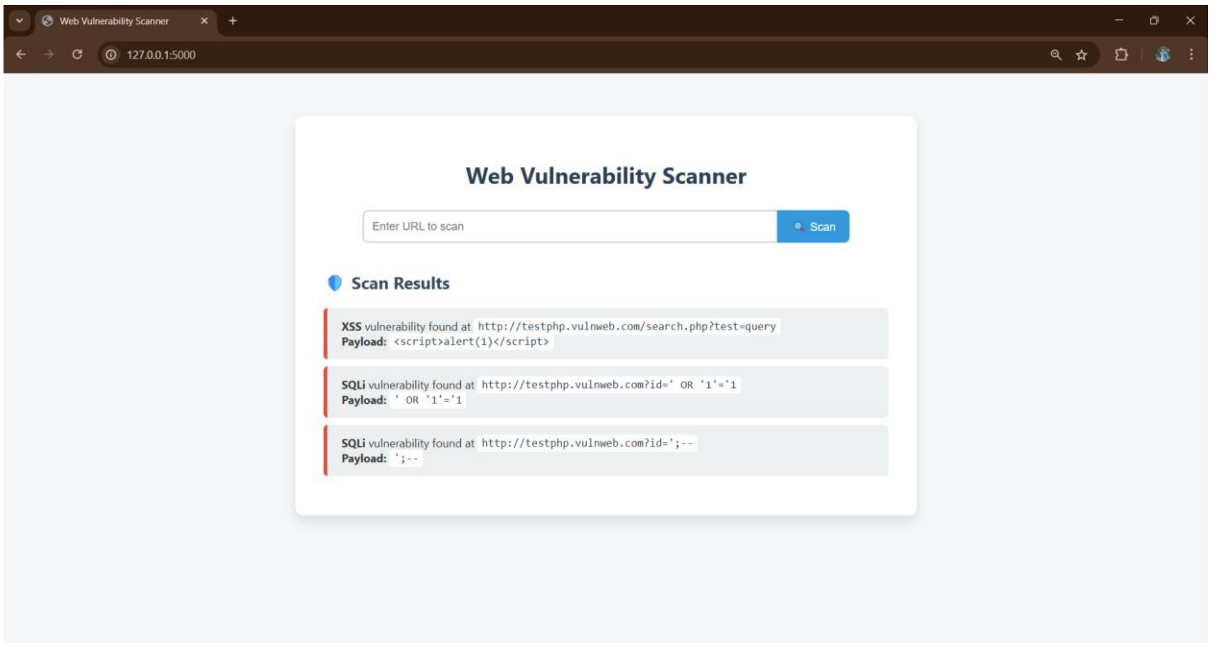
```
+---------------------+      +---------------------+

|     Flask UI       | <---> |   Scanner Engine    |
+---------------------+     +---------------------+
        |                   |
        v                   v
+---------------------+     +---------------------+
|  Requests Module  |      | BeautifulSoup + Re  |
+---------------------+     +---------------------+
        |                   |
        v                   v
+---------------------+     +---------------------+
|  Vulnerability DB  |      |  HTML Analyzer     |
+---------------------+     +---------------------+
```

Features Implemented

Feature	**Status**
Crawl web pages & extract input	✓
Inject XSS, SQLi, and CSRF payloads	✓
Analyze responses using regex	✓
Log vulnerabilities with evidence	✓
Flask Web Interface	✓
Report generation	✓

Web Application interface



Web Vulnerability Scanner

Scan Results

XSS vulnerability found at `http://testphp.vulnweb.com/search.php?test=query`
Payload: `<script>alert(1)</script>`

SQLi vulnerability found at `http://testphp.vulnweb.com?id=' OR '1'='1`
Payload: `' OR '1'='1`

SQLi vulnerability found at `http://testphp.vulnweb.com?id=';--`
Payload: `';--`

Sample Output (JSON-style)

```
{
 "url": "http://testsite.com/login",
 "vulnerabilities": [
  {
   "type": "SQL Injection",
   "parameter": "username",
   "payload": "' OR 1=1 --",
   "evidence": "You have an error in your SQL syntax;",
   "severity": "High"
  },
  {
   "type": "Cross-Site Scripting",
   "parameter": "search",
   "payload": "<script>alert(1)</script>",
   "evidence": "<script>alert(1)</script>",
   "severity": "Medium"
```

```
    }
  ]
}
```

Severity Rating Criteria

Severity	**Criteria**
Low	Potential but not confirmed risk; non-critical data
Medium	Confirmed vulnerability that requires user interaction
High	Critical vulnerability that allows unauthorized access, data leak, or code execution

Conclusion

This Web Application Vulnerability Scanner successfully identifies key web security flaws in real-time by crawling, injecting, and analyzing web pages. It is especially suitable for:

- Ethical hackers and security researchers
- Students learning web security
- Developers testing pre-deployment applications

The modular architecture allows for easy extension, e.g., adding more OWASP vulnerabilities or exporting results in PDF/JSON.

Future Improvements

- Add support for **Command Injection**, **Directory Traversal**, and **File Upload vulnerabilities**.
- Integrate **report export as PDF**.
- Add **authentication support** for scanning protected pages.
- Implement **multi-threading** for faster scanning.
- Visual dashboard using **charts** for vulnerability statistics.