Cyber Security Project Report

Web Application Vulnerability Scanner

Submitted by:
Pusarla Vinay

Date of Submission:
12th September 2025

Internship Duration:
15th Aug – 15th Sept 2025

Institution/Organization:
Cyber Security Intern

About this Project

This project is a **Python-based Web Application Vulnerability Scanner** designed to detect common web application security issues like Cross-Site Scripting (XSS), SQL Injection (SQLi), and Cross-Site Request Forgery (CSRF).
The scanner automatically crawls a target website, analyzes input fields, injects payloads, and reports potential vulnerabilities with severity levels. A **Flask-based web interface** was built to make scanning and reporting more user-friendly.

Index

Introduction:

With the increasing use of web applications in every sector, security has become a major concern. Attackers often exploit vulnerabilities such as SQL Injection, XSS, and CSRF to compromise data and gain unauthorized access.
This project aims to develop a **lightweight and automated scanner** that follows the OWASP Top 10 guidelines to identify and log such vulnerabilities. The tool provides both a **command-line backend** and a **web-based dashboard** for initiating and monitoring scans.

Abstract:

The main objective of this project is to create a **web vulnerability scanner** capable of detecting common attacks with minimal setup. It uses:

- **requests** and **BeautifulSoup** for crawling websites,
- **regex and payload injections** for testing vulnerabilities, and
- **Flask** for creating an easy-to-use interface.

The scanner saves results in **JSON format** for easy analysis and displays findings in a structured table on the web dashboard.

Tools Used:

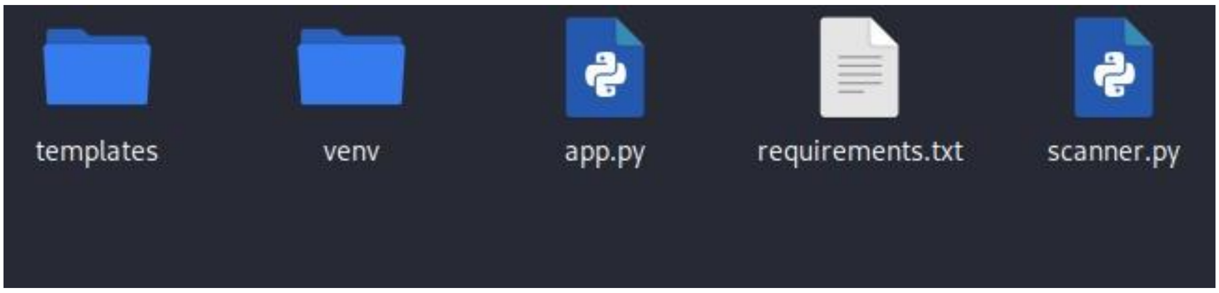
- **Programming Language:** Python
- **Libraries:** requests, BeautifulSoup4, Flask, json, regex
- **Platform:** Kali Linux (can also run on Windows)
- **Security Reference:** OWASP Top 10 checklist

Steps Involved

1. **Project Setup**
    - Created a virtual environment and installed dependencies from requirements.txt.
    - Project structured into files and folders:
        - app.py → Flask web application (UI and API).
        - scanner.py → Core vulnerability scanning logic.
        - templates/ → HTML files (index.html, status.html).
        - scans/ → Folder to store scan reports in JSON format.

2. **Building the Scanner (scanner.py)**
    - Used **requests** and **BeautifulSoup** to crawl links and forms.
    - Implemented test cases for common vulnerabilities (XSS, SQLi, CSRF).
    - Results stored in structured JSON with details like type, payload, and severity.

3. **Web Interface (app.py + templates/)**
    - Developed a Flask web application with:
        - **Index Page (index.html):** Take user input (target URL, scan depth).
        - **Status Page (status.html):** Display scan results in a table.
    - Added REST API (/api/results/<scan_id>) to fetch scan results programmatically.



4. **Running the Scan**
    - Users start a scan from the web interface.
    - Flask calls the run_scan() function from scanner.py.
    - Results saved under /scans with a unique scan ID (e.g., a18fe2cf49224b758743ff32770037cf.json).

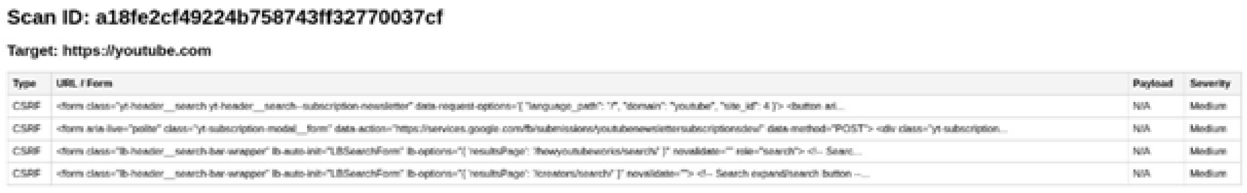
5. **Commands Used**
    - Create and activate a virtual environment:
    - python3 -m venv venv
    - source venv/bin/activate
    - Install required packages:
    - pip install -r requirements.txt
    - Start the Flask application:
    - flask run
    - Access the scanner in the browser:
    - http://127.0.0.1:5000

6.Reporting

- Results displayed in browser with details of vulnerabilities.
- JSON reports can be downloaded from the /scans folder for offline review.

Output :

Target: **https://youtube.com**
Scan ID: **a18fe2cf49224b758743ff32770037cf**



Scan ID: a18fe2cf49224b758743ff32770037cf

Target: https://youtube.com

Type	URL / Form	Payload	Severity
CSRF	<form class="yt-header__search yt-header__search--subscription-newsletter" data-request-options='{ "language_path": "/", "domain": "youtube", "site_id": 4 }'> <button ari...	N/A	Medium
CSRF	<form aria-live="polite" class="yt-subscription-modal__form" data-action="https://services.google.com/fb/submissions/youtubenewslettersubscriptionsdev/" data-method="POST"> <div class="yt-subscription...	N/A	Medium
CSRF	<form class="lb-header__search-bar-wrapper" lb-auto-init="LBSearchForm" lb-options="{ 'resultsPage': '/howyoutubeworks/search/' }" novalidate="" role="search"> <!-- Searc...	N/A	Medium
CSRF	<form class="lb-header__search-bar-wrapper" lb-auto-init="LBSearchForm" lb-options="{ 'resultsPage': '/creators/search/' }" novalidate=""> <!-- Search expand/search button --...	N/A	Medium

Conclusion:

This cybersecurity project demonstrated the importance of systematic Linux auditing and hardening by performing firewall verification, SSH configuration checks, file permission analysis, process monitoring, and rootkit detection. The system was found to be moderately secure, with strengths in areas such as file permissions and disabled root login, but with improvement opportunities in SSH authentication, firewall restrictions, and regular rootkit monitoring.
The project not only validated the effectiveness of Linux security tools and automation scripts but also enhanced practical skills in identifying, interpreting, and mitigating security issues. With a current security score of 80/100, the system is reasonably protected; however, by implementing the recommended improvements, this score can be significantly increased, ensuring stronger resilience against real-world cyber threats.