Web Application Vulnerability Scanner

Abstract

This project aims to create a lightweight and customizable vulnerability scanner for detecting common web application issues, including Cross-Site Scripting (XSS), SQL Injection (SQLi), and Cross-Site Request Forgery (CSRF). The tool is built in Python with a Flask-based web interface, allowing users to enter a target URL, conduct automated scans, and view results with evidence and severity levels. This project supports cybersecurity awareness and offers developers and students a simple way to test web application security in a controlled setup.

Introduction

Web applications are frequent targets for cyber attacks. Vulnerabilities like XSS, SQL Injection, and CSRF are listed in the OWASP Top 10, making them serious risks for any online system. The goal of this project is to build an automated vulnerability scanner that identifies these problems by crawling web pages, injecting payloads, and analyzing responses.

Objectives

1. Create a Python-based web vulnerability scanner.

2. Automate detection of XSS, SQL Injection, and CSRF vulnerabilities.

3. Provide a Flask-based web interface for running scans and viewing results.

4. Generate logs with evidence of vulnerabilities and their severity levels.

Tools & Technologies

• Programming Language: Python 3

• Web Framework: Flask

• Libraries: requests, BeautifulSoup4, lxml, tqdm, python-dotenv

• Frontend: HTML, CSS (basic UI)

• Checklist: OWASP Top 10 vulnerabilities

Methodology

1. Crawl the target URL using requests and BeautifulSoup to gather forms and links.

2. Inject payloads for XSS and SQLi into input fields and monitor responses.

3. Use regex or pattern matching to spot successful payload execution.

4. Log results, including the type of vulnerability, affected URL, and severity.

5. Show results in the Flask web interface for user review.

The developed scanner can successfully detect potential XSS and SQLi vulnerabilities on test websites like http://testphp.vulnweb.com/. The Flask UI makes it easy for users to input URLs, run scans, and see results with severity levels. Logs offer actionable insights for developers to address vulnerabilities.

Conclusion

This project highlights the importance of automated vulnerability scanning in securing web applications. By using Python and Flask, we created a user-friendly tool that can be expanded to cover more vulnerabilities from the OWASP Top 10. Future work includes adding authentication handling, scanning for CSRF, and integrating AI-based triage.