**Project Report: Web Application Vulnerability Scanner**

**Introduction**

In today’s digital world, web applications are a primary target for cyber-attacks such as SQL Injection (SQLi), Cross-Site Scripting (XSS), and Cross-Site Request Forgery (CSRF). Vulnerabilities in poorly secured websites may expose sensitive data, damage brand reputation, or even compromise entire systems.

This project, **Web Application Vulnerability Scanner**, was developed to automatically detect common security flaws in web applications. It crawls the target website, identifies links and forms, injects crafted payloads, and reports potential vulnerabilities. The tool is intended only for educational and authorized penetration testing.

**Abstract**

The scanner is a lightweight, Python-based framework built with **Flask** for the web interface and **Requests + BeautifulSoup4** for crawling and payload injection. It helps security analysts, students, and developers test applications for common vulnerabilities in a safe environment.

The scanner accepts a target URL, crawls for links and forms, attempts injections with pre-defined payloads, and then generates structured vulnerability reports. Users can view results via the Flask interface or download JSON reports.

**Tools Used**

* **Programming Language:** Python 3.13
* **Framework:** Flask (for user interface)
* **Libraries:** Requests, BeautifulSoup4, lxml, dotenv
* **Development Environment:** Visual Studio Code
* **Testing Targets:** OWASP Juice Shop, DVWA (local), TestPHP.vulnweb.com
* **Version Control:** GitHub repository

**Steps Involved in Building the Project**

**1. Environment Setup**

* Created project folder WebVulnScanner/
* Configured Python 3.13 virtual environment
* Installed required dependencies (Flask, requests, beautifulsoup4, etc.)

**2. Project Structure**

WebVulnScanner/

├─ app.py # Flask application

├─ scanner.py # Core vulnerability detection

├─ crawler.py # Website crawler

├─ payloads.py # Payload library

├─ vulnerabilities.py # Data model for results

├─ templates/ # HTML templates

├─ static/ # Stylesheets

├─ reports/ # Auto-generated reports

**3. Implementation**

* **Crawler Module:** Extracts links and forms while ensuring same-origin checks.
* **Scanner Module:** Injects SQLi and XSS payloads in query parameters and forms, checks responses for errors or reflections, and flags issues.
* **Payloads Module:** Stores predefined SQLi and XSS payloads.
* **Flask App:** Provides web UI to enter a URL, run a scan, and view/download reports.
* **Reports:** Results stored in JSON format for auditing and future analysis.

**4. Testing**

* Scanner tested on intentionally vulnerable applications (Juice Shop, DVWA, TestPHP).
* Verified detection of SQL Injection and XSS.
* Validated CSRF heuristics (POST forms without tokens flagged as Low severity).

**Conclusion**

This project successfully demonstrated how automated vulnerability scanners work at a fundamental level. While not as advanced as commercial tools (e.g., Acunetix, Burp Suite), the scanner provides hands-on learning about web application security testing.

The system can be further enhanced by:

* Adding authentication handling (login sessions).
* Expanding payload libraries for more attack types (RCE, LFI).
* Implementing asynchronous scanning with Celery/Redis.
* Adding rate limiting to avoid overloading targets.