Internship Project Report
Web Application Vulnerability Scanner

Introduction

As part of my two-week cybersecurity internship project, I chose to build a *Web Application Scanner.* With the rise in web-based threats, understanding how vulnerabilities such as SQL Injection (SQLi(and Cross-Site Scripting (XSS) work – and how to detect them – is crucial for any aspiring cybersecurity professional. This project simulates a real-world vulnerability assessment tool that scans websites for common weaknesses and provides actionable reports.

Abstract

The Web Vulnerability Scanner is a Python-based tool with a Flask web interface that detects critical vulnerabilities like SQLi and XSS by crawling URLs and analyzing form fields. It uses automated payload injections to identify insecure inputs and reflects results via a user-friendly UI. The goal was to mimic professional scanners like Burp Suite on a small scale while learning the mechanics behind vulnerability detection. The scanner includes real-time status indicators, PDF report generation, and clear logging to simulate a real-world web audit workflow.

Tools Used

- **Python -** Core programming language
- **Flask -** For creating the web interface
- **BeautifulSoup -** HTML parsing and input crawling
- **Requests -** For making HTTP requests
- **xhtml2pdf -** For PDF report generation
- **HTML/CSS/JavaScript -** For frontend styling and functionality

Steps Involved in Building the Project

1. **Project Planning & Folder Structure :**
   Designed a modular folder structure separating frontend, backend, scanning logic, and reports.
2. **Crawler Development :**
   Wrote logic to fetch and parse input fields using BeautifulSoup to discover form attack points.
3. **Payload Injection & Scanner Logic :**
   Implemented XSS and SQLi payload injection functions to simulate attacks on target forms.
4. **Flask Integration :**
   Connected backend scanner functions to a web frontend Flask routes and templates.
5. **Frontend Enhancements :**
   Added CSS styling, a real-time scan timer, a progress bar, and user feedback elements.

6. **Logging & Report Generation :**
   Stored scan logs in text format and provided downloadable vulnerability reports in PDF format with clean formatting.
7. **Testing & Debugging :**
   Used a vulnerable test site (testphp.vulnweb.com) to validate scanner performance and patched edge cases.

Conclusion

This project strengthened my understanding of web vulnerabilities and Python-based automation. I built the complete vulnerability scanning logic independently and received external help for organizing the folder structure and improving syntax where necessary. The project gave me practical insights into how scanners work, how vulnerabilities are exploited, and how automated detection systems can be built. It also improved my frontend and backend integration skills using Flask.