Web Application Vulnerability Scanner

Advanced Security Assessment Platform

BY: Neeraj Vishwakarma | **Date:** November 23, 2025

INTRODUCTION:

This project documents the development of a production-grade **Web Application Vulnerability Scanner**, evolving from a basic requirement into a robust "Bug Hunting Framework." Originally tasked with creating a simple detector for standard vulnerabilities (XSS, SQLi), the project scope was significantly expanded to address modern security defenses. The final tool integrates advanced modules like **Cloudscraper** for WAF evasion and **Selenium** for DOM-based analysis, managed via a modular architecture. This report outlines the transition from a script-based approach to a full-stack security platform capable of identifying high-severity risks in live environments.

TOOLS AND TECHNOLOGIES USED:

- **Core Logic:** Python 3.x, Flask (Web Framework/GUI).
- **Networking & Evasion:** Requests, Cloudscraper (WAF Bypass), Selenium (Headless Chrome).
- **Parsing & Analysis:** BeautifulSoup4 (HTML Parsing), Regex (Pattern Matching).
- **Data & Reporting:** SQLite (Database), FPDF (Automated PDF Reporting).
- **Attack Vectors:** Modular Text-Based Payloads (SQLi, XSS, SSTI, LFI).

PROJECT ARCHITECTURE AND DEVELOPMENT STEPS:

1. Modular Architecture Design

To overcome the limitations of hardcoded scanners, a modular "Orchestrator" pattern was adopted. The system is divided into three distinct layers:

- **Controller (app.py):** Manages the Flask web server and user interface logic.
- **Orchestrator (master_scanner.py):** Coordinates the crawler, detection engine, and reporting modules.
- **Data Layer (payloads/):** This layer is built on a **Text-Based Payload Architecture**, where **each vulnerability (e.g., XSS, SQLi) has its own dedicated .txt file** organized in folders like Injection/, AccessControl/, and Misconfiguration/. The scanner automatically loads all payloads from the relevant file at runtime. This allows security researchers to dynamically add new attack vectors (EX: simply opening `xss.txt` and pasting a new payload) without modifying the core source code. The system automatically adapts to the updated file, making the framework infinitely scalable and adaptable to new threats.

2. Advanced Crawling and Detection Engine

The core engine (crawler.py & injection_engine.py) utilizes a multi-threaded approach to map the target application.

- **Heuristic Tech Detection:** Identifies server technologies (e.g., PHP, Angular) to select relevant payloads.
- **Dynamic Content Rendering:** Integrated Selenium to execute JavaScript, enabling the detection of DOM-based XSS vulnerabilities that standard request libraries miss.

3. WAF Bypass and Session Management

A significant challenge was scanning protected targets. The integration of **Cloudscraper** allows the scanner to bypass Cloudflare challenges. A robust session management system was built to handle authenticated scans, maintaining valid tokens throughout the assessment process.

KEY FEATURES AND RESULTS:

The final deliverable is a comprehensive security platform featuring:

- **Real-Time Dashboard:** A Flask-based UI displaying scan progress, active threads, and live findings.
- **Severity Scoring:** Automatic CVSS v3.1 scoring for identified vulnerabilities.
- **Automated Reporting:** One-click generation of professional PDF reports containing executive summaries, detailed technical findings, and specific mitigation advice.

CONCLUSION:

This project successfully transcended its initial academic scope to become a practical security tool. By solving complex challenges related to WAF evasion and dynamic content handling, the scanner demonstrated its capability to identify critical vulnerabilities (such as SSTI and Reflected XSS) in controlled environments. This experience has provided deep insights into offensive security tool development and modern web defense mechanisms.