# Oculus-Web Application Vulnerability Scanner — Project Report

**Project Name:** Web Application Vulnerability Scanner (“Oculus”)

**Author:** Keerthana N.

**Date:** October 24, 2025

## 1. Abstract

A Python-based web application vulnerability scanner with a simple Flask web interface, a command-line interface (CLI), and a modular scanner backend. The tool performs non-intrusive and configurable checks for common web vulnerabilities (Directory Traversal, Reflected XSS, SQL Injection) and produces structured JSON reports. A lightweight local test server (vuln\_app.py) is included for safe experimentation.

This project is educational. **Do not scan systems without explicit authorization.**

## 2. Objectives

* Build a modular vulnerability scanner in Python that is easy to extend.
* Provide a simple web UI and REST API to start scans and fetch reports.
* Provide a CLI for automated or scripted scans.
* Keep the scanner safe by default (intrusive tests opt-in) and include a local vulnerable test server for learning.

## 3. Tools & Technologies

* **Language:** Python 3.10+ (tested on 3.11/3.13)
* **Web framework:** Flask
* **HTTP client:** requests
* **Parsing:** BeautifulSoup (future crawler use)
* **Dev environment:** VS Code (Windows)
* **Version control:** Git + GitHub

## 4. Project Structure

Oculus/ # project root  
├── app.py # Flask UI + REST API  
├── scan\_worker.py # background scan runner + report writer  
├── scan\_cli.py # CLI runner  
├── vuln\_app.py # local vulnerable test server (safe)  
├── scanner/ # scanner package  
│ ├── \_\_init\_\_.py  
│ ├── attacker.py # injection/testing logic  
│ ├── analyzer.py # detection helpers  
│ └── payloads.py # payload lists  
├── reports/ # generated JSON reports (gitignored)  
├── venv/ # virtual environment (gitignored)  
├── requirements.txt  
└── .gitignore

## 5. Key Components (what & why)

### 5.1 app.py

* Hosts a small HTML UI and JSON REST API endpoints:
  + POST /api/scan — start a scan (returns scan\_id)
  + GET /api/status/<scan\_id> — check status
  + GET /api/report/<scan\_id> — download JSON report
* Spawns scans in background threads referencing scan\_worker.py.

### 5.2 scan\_worker.py

* Creates an Attacker instance, runs run\_scan(), and writes a reports/report\_<scan\_id>.json file.
* Normalizes findings into a JSON-friendly list.

### 5.3 scan\_cli.py

* Simple CLI wrapper to run scans from the terminal and optionally print JSON to stdout.

### 5.4 scanner/attacker.py

* Modular testing functions for directory traversal, reflected XSS, and SQL injection.
* Uses scanner/payloads.py for payload sets and scanner/analyzer.py for detection heuristics.

### 5.5 vuln\_app.py

* Local Flask-based server that simulates vulnerable behavior (safe, does not read actual system files) for learning and testing.

## 6. Installation & Setup (Windows, step-by-step)

1. Clone or copy repository to C:\Users\<you>\Desktop\Oculus.
2. Open a terminal (Command Prompt recommended).
3. Create virtual environment:

* python -m venv venv  
  venv\Scripts\activate

1. Install dependencies:

* pip install -r requirements.txt

1. Run the main app (UI + API):

* python app.py

1. (Optional) In a second terminal, run the local vulnerable server for safe tests:

* venv\Scripts\activate  
  python vuln\_app.py

1. Open browser: http://127.0.0.1:5000 — use the UI to start a scan (target: http://127.0.0.1:5001)

## 7. How to Use

### 7.1 Web UI

* Enter Target URL and click **Start Scan**. A scan\_id is returned. Use the status box to query progress and download JSON report.

### 7.2 REST API (curl examples)

* Start scan (returns scan\_id):
* curl -X POST "http://127.0.0.1:5000/api/scan" -H "Content-Type: application/json" -d '{"target":"http://127.0.0.1:5001"}'
* Get status:
* curl http://127.0.0.1:5000/api/status/<scan\_id>
* Download report:
* curl -O http://127.0.0.1:5000/api/report/<scan\_id>

### 7.3 CLI

* Run a CLI scan and print report:
* python scan\_cli.py http://127.0.0.1:5001 --print

## 8. Example Findings (sample JSON entry)

{  
 "type": "directory\_traversal",  
 "severity": "high",  
 "payload": "../../../../etc/passwd",  
 "url": "http://127.0.0.1:5001/?file=../../../../etc/passwd",  
 "evidence": "Simulated /etc/passwd content found or pattern matched"  
}

## 9. Limitations & Future Work

* **Limited coverage**: Current scanner checks a small set of vulnerabilities (XSS, SQLi, traversal). Expand payloads and add authentication for authenticated scans.
* **No JavaScript execution**: Some XSS/SSRF need a headless browser (Playwright/Selenium) for full coverage.
* **Concurrency & persistence**: Background threads are ephemeral; for scale use a task queue (RQ/Celery) and a database.
* **False positives**: Heuristics may produce false positives; manual verification is required.

Planned enhancements: - Async crawling (aiohttp), authenticated scans, Playwright integration, CVE mapping for detected components, output HTML/PDF reports.

## 10. Ethics & Legal

* This tool is for authorized testing only. Scanning without explicit permission may be illegal. ALWAYS obtain written authorization.
* Intrusive tests (OS command injection, credential stuffing, port scans) are opt-in only.

## 11. How to extend (developer notes)

* Add payloads to scanner/payloads.py and detection logic in scanner/analyzer.py.
* Use scanner.async\_crawler.AsyncCrawler for faster scanning (example available in canvas).
* For authenticated scans, implement scanner/auth.py and pass a requests.Session with cookies to the crawler/attacker.

## 12. Deliverables

* Source code (this repository) — Flask UI, CLI, scanner modules
* Example reports in reports/ (generated during scans)
* Project report (this document)

## 13. Appendix — Useful Commands

* Create venv: python -m venv venv
* Activate venv (Command Prompt): venv\Scripts\activate
* Install deps: pip install -r requirements.txt
* Run app: python app.py
* Run vuln test server: python vuln\_app.py
* Run CLI scan: python scan\_cli.py http://127.0.0.1:5001 --print