Web Application Vulnerability Scanner – Project Report

➢ Introduction

This project focuses on developing a Web Application Vulnerability Scanner capable of detecting common security vulnerabilities such as Cross-Site Scripting (XSS), SQL Injection (SQLi), Cross-Site Request Forgery (CSRF), and insecure configurations.
The project includes a Flask-based web interface, a backend scanning engine, and an automated reporting system to help users analyze the security posture of any target web application.

➢ Abstract

The Web Application Vulnerability Scanner is built using Python, Flask, Requests, BeautifulSoup, and regex-based detection techniques.
The system crawls a target website, identifies input fields, injects crafted payloads, and analyzes the server responses to identify potential vulnerabilities.
The scanner follows the OWASP Top 10 guidelines and generates structured reports containing vulnerability names, severity levels, and proof-of-concept evidence.
This project demonstrates the practical implementation of automated web security testing and showcases the fundamentals of vulnerability assessment.

➢ Tools Used

- Python 3
- Flask Web Framework
- Requests Library
- BeautifulSoup (bs4)
- Regular Expressions (Regex)
- HTML & CSS for UI
- OWASP Top 10 checklist for vulnerability reference
- JSON for storing reports
- Virtual Environment (venv) for dependency isolation

➢ Steps Involved in Building the Project

1. Project Environment Setup

- Created a structured directory with modules for scanner, authentication, payloads, and reports.
- Installed necessary libraries (requests, bs4, Flask, etc.) and activated a virtual environment.

2. Crawling & Input Detection

- Implemented a URL crawler using Requests and BeautifulSoup.

- Identified form fields, GET/POST parameters, and clickable links in the HTML.

3. Payload Injection Module

- Designed custom payloads for:
  - SQL Injection
  - Cross-Site Scripting (XSS)
  - CSRF (basic token validation)
- Injected these payloads into detected fields and recorded server responses.

4. Vulnerability Detection Engine

- Used regex and response-pattern matching to detect anomalies.
- Looked for reflected tags, SQL errors, missing CSRF tokens, etc.
- Classified vulnerabilities based on severity (High/Medium/Low).

5. Flask Web Interface Development

- Created web routes for: Login / Registration, Dashboard, Scan Page, Scan Progress Page, Scan Result Page
- Built HTML templates to make the UI user-friendly and simple.

6. Report Generation

- Stored scan results in JSON format with details such as endpoint, payload used, and confirmation evidence.
- Provided an option for users to download the scan report.

7. Testing & Validation

- Tested using safe, dummy websites and intentionally vulnerable platforms.
- Verified detection accuracy for basic SQLi and XSS.

➢ Conclusion

The Web Application Vulnerability Scanner successfully demonstrates the process of automated web security analysis. It provides a practical understanding of crawling, payload injection, response analysis, and application-level vulnerability detection.
The system can be further enhanced by adding features such as multi-threading, bypass payloads, authenticated scanning, dashboards, and integration with external security tools.
Overall, the project strengthens the understanding of OWASP Top 10 vulnerabilities and practical web security techniques.