Web Application Vulnerability Scanner - Project Report

Introduction

Web applications are frequent targets for cyberattacks due to their exposure and complexity. Detecting vulnerabilities early is crucial for maintaining security and trust. This project delivers a modern, Python-based web application vulnerability scanner that identifies common security flaws such as Cross-Site Scripting (XSS), SQL Injection (SQLi), Cross-Site Request Forgery (CSRF), and missing security headers, all through an intuitive web interface.

Abstract

The scanner automates the process of crawling web applications, injecting test payloads, and analyzing responses to detect vulnerabilities. It leverages Python's requests and BeautifulSoup libraries for crawling and parsing, and Flask for a user-friendly web interface. The tool provides detailed, categorized reports, helping developers and security professionals quickly identify and address security risks. The project emphasizes ethical scanning, ease of use, and extensibility.

Tools Used

- **Python 3.11**: Core programming language
- **Flask**: Web application framework for the UI and API
- **requests**: HTTP client for crawling and payload injection
- **BeautifulSoup4**: HTML parsing and form extraction
- **lxml**: Fast HTML/XML parsing
- **Bootstrap 5 & Font Awesome**: Modern, responsive UI
- **Jinja2**: Templating engine for dynamic HTML
- **python-dotenv**: Environment variable management

Steps Involved in Building the Project

1. **Requirements & Planning**: Defined project scope based on OWASP Top 10 vulnerabilities and user needs.
2. **Environment Setup**: Created a Python virtual environment and installed all dependencies.
3. **Core Scanner Development**:
    - Implemented a crawler to discover forms and input fields using requests and BeautifulSoup.
    - Developed XSS and SQLi detection by injecting payloads and analyzing responses for reflections or error patterns.
    - Added CSRF checks by detecting missing anti-CSRF tokens in forms.
    - Checked for missing security headers in HTTP responses.
4. **Web Interface (Flask)**:
    - Built a modern, responsive UI for scan management and result visualization.
    - Enabled scan history, result viewing, and deletion.
    - Provided REST API endpoints for automation.
5. **Reporting & Logging**:
    - Categorized vulnerabilities by severity (Critical, High, Medium, Low).
    - Logged evidence, payloads, and timestamps for each finding.
    - Stored results in structured JSON files for easy access and review.
6. **Testing & Documentation**:
    - Created unit tests for core functions.
    - Documented setup, usage, and ethical guidelines.

Conclusion

This project delivers a practical, efficient, and user-friendly solution for web application vulnerability assessment. By automating the detection of critical security issues and providing clear, actionable reports, it empowers developers and security teams to proactively secure their applications. The modular design allows for easy extension to cover additional vulnerabilities, making it a valuable tool for ongoing security assurance.