PROJECT REPORT

Introduction

Web application security is a critical aspect of modern software development. Applications are constantly exposed to threats from attackers seeking to exploit vulnerabilities for malicious purposes, such as data theft or service disruption. Common vulnerabilities include SQL Injection (SQLi), which targets databases, and Cross-Site Scripting (XSS), which targets users. Manually testing for these vulnerabilities can be time-consuming and inefficient. This project, the "Web Security Auditor," was developed to automate the process of detecting these common security flaws. The goal was to create a simple, user-friendly tool with a web interface that allows a user to enter a target URL and receive a clear report of potential issues.

Abstract

This project details the creation of the "Web Security Auditor," a Python-based tool designed to scan web applications for common security vulnerabilities. The application is built using the Flask web framework, which provides a user-friendly interface for initiating scans and viewing results. The core scanning engine is composed of modular Python functions that leverage the `requests` library to send crafted HTTP requests with specific payloads to a target URL. The engine analyzes the server's responses to detect evidence of vulnerabilities, including Error-Based SQL Injection, Reflected Cross-Site Scripting, and the absence of critical security headers. Findings are categorized by severity and displayed to the user. For portability and ease of deployment, the entire application was containerized using Docker, demonstrating a modern and consistent development workflow.

Tools Used

- **Programming Language:** Python 3.9
- **Web Framework:** Flask (for the user interface and backend logic)
- **HTTP Library:** Requests (for sending payloads and analyzing web page responses)
- **PDF Generation:** fpdf2 (for creating the downloadable PDF reports)
- **Containerization:** Docker (for packaging the application and its dependencies)
- **Front-End:** HTML & CSS (for structuring and styling the web pages)
- **Code Editor:** Visual Studio Code
- **Terminal:** Windows PowerShell

Steps Involved in Building the Project

The project was developed through a series of logical steps:

1. **Project Setup and Environment:** The initial phase involved creating the project's folder structure, setting up a Python virtual environment to manage dependencies, and installing the core libraries (`Flask`, `requests`, etc.).

2. **UI and Web Server Development:** A basic web server was created using Flask. The front-end was built with HTML to create the input form for the target URL and a template for the results page.
3. **Scanner Module Implementation:** Individual Python functions were developed to test for specific vulnerabilities. Each module (`scan_sqli`, `scan_xss`, `check_headers`) was designed to take a URL, send specific payloads, and return any findings.
4. **Backend Integration:** The Flask UI was connected to the scanner modules. A `/scan` route was created to handle form submissions, call the relevant scanning functions with the user-provided URL, and pass the aggregated results to the results page.
5. **PDF Report Feature:** A "Download Report" feature was implemented using the `fpdf2` library. A new Flask route was added to generate a PDF document summarizing the findings from the most recent scan.
6. **Containerization with Docker:** A `Dockerfile` was written to define the application's environment. This file contains all the instructions to package the Python interpreter, install dependencies from `requirements.txt`, copy the application code, and run the Flask server.
7. **Testing and Debugging:** Throughout the process, the application was tested against known vulnerable sites. Several issues were diagnosed and fixed, including Python import errors within the Docker container, file naming issues, and incorrect `Dockerfile` configurations. The image was rebuilt after each change to ensure the fixes were applied.

Conclusion

This project successfully achieved its objective of creating a functional, containerized web vulnerability scanner. The "Web Security Auditor" can effectively identify basic SQLi, XSS, and missing security header vulnerabilities from a simple web interface. The project demonstrates a practical application of Python for cybersecurity, proficiency with the Flask framework, and an understanding of modern deployment practices using Docker.

Future improvements could include expanding the scanner with more modules (e.g., for command injection or directory traversal), integrating a more advanced web crawler to discover more pages on a target site, and adding a database to store and compare scan results over time.