Web Application Vulnerability Scanner

1. Introduction

With the exponential growth of web applications, ensuring their security has become critical. Web application vulnerability scanners (WAVS) play a vital role in automating the detection of security flaws. These tools scan websites and web-based applications to identify potential vulnerabilities such as SQL Injection, Cross-Site Scripting (XSS), insecure server configurations, outdated libraries, and more.

2. Abstract

This project presents a lightweight, customizable vulnerability scanner that identifies common security issues in web applications. It uses Python's requests and BeautifulSoup libraries to crawl and interact with web pages, sending crafted payloads to test for vulnerabilities like XSS, SQLi, CSRF, and more. The tool includes a Flask-based user interface, where users can initiate scans, view logs, and download detailed reports.

3. Tools and Technologies Used

- Python: Core programming language
- Requests: Sending HTTP requests
- BeautifulSoup: HTML parsing and form extraction
- Urllib/URLparse: URL manipulation and crawling
- Regex: Pattern matching for input validation
- Nmap (optional): Network mapping and port scanning
- OWASP ZAP API (optional): External scanning integration
- SQLite or CSV: Storing scan results

4. Key Features of the Scanner

URL crawling and sitemap generation

Form detection and parameter extraction

Injection testing (e.g., SQLi, XSS payloads)

Response analysis and pattern recognition

Basic report generation in text or HTML

5. Steps Involved in Building the Project

Step 1: Define the Scope

Accept user input for target URL.

Validate and normalize the URL.

Step 2: Web Crawler Module

Use requests to fetch page content.

Parse links using BeautifulSoup.

Recursively crawl internal links to create a sitemap.

Step 3: Form Extractor

Identify forms in HTML.

Extract input fields, method (GET/POST), and action URLs.

Step 4: Vulnerability Test Engine

SQL Injection Testing:

Inject test payloads (' OR '1'='1, --, etc.) into form fields or URL parameters.

Analyze response for SQL error patterns.

XSS Testing:

Inject <script>alert('XSS')</script> and check if it's reflected.

Header Analysis:

Check for missing HTTP security headers (e.g., X-Content-Type-Options, CSP).

Step 5: Response Analysis

Use regular expressions to detect error messages, reflected input, or lack of sanitization.

Log successful injection or header-related issues.

Step 6: Reporting Module

Generate structured reports listing:

Page URL

Type of vulnerability

Payload used

Severity level

Output in plain text, JSON, or HTML.

6. Conclusion

Building a custom web application vulnerability scanner enhances understanding of web security fundamentals and common attack vectors. Although basic, such a tool provides useful insight into a web application's surface-level vulnerabilities. It is recommended to use it alongside established tools like OWASP ZAP, Burp Suite, or Netsparker for more comprehensive coverage.