Web Vulnerability Scanner: Proof-of-Concept Documentation

Abstract

This document details the development of a proof-of-concept web application scanner designed for educational purposes. The tool is implemented using Python and Flask, leveraging key libraries for web interaction and HTML parsing. Its primary function is to demonstrate basic vulnerability detection techniques, specifically identifying potential weaknesses related to **Reflected Cross-Site Scripting (XSS)** and **SQL Injection (SQLi)** by submitting predefined, non-destructive payloads to discovered web forms. This project serves as a foundational exercise in understanding how automated security tests are structured and how web applications respond to malicious inputs.

Introduction

Web application security is a critical field, focusing on preventing unauthorized access, data theft, and denial of service. Vulnerability scanners are automated tools used to identify known weaknesses.

This project simulates the core function of a dynamic application security testing (DAST) tool. It achieves this by:

1. Fetching the HTML content of a target URL.
2. Identifying all input forms.
3. Injecting test payloads into form fields.
4. Analyzing the server's response to determine if the input was processed insecurely.

The scanner's design intentionally uses basic payload-detection methods to emphasize foundational security concepts and the importance of output encoding (XSS prevention) and parameterized queries (SQLi prevention) in web development.

Tools Used

The project relies exclusively on the Python ecosystem, utilizing several powerful libraries:

Tool/Library	Purpose in Project	Security Context
Python 3	Core programming language.	Used for rapid development of security tools.

Flask	Web framework.	Provides the backend structure (/ and /scan routes) to handle user input and display results via a web interface.
Requests	HTTP client library.	Essential for simulating user actions, specifically making GET and POST requests to submit form data containing vulnerability payloads.
BeautifulSoup (bs4)	HTML parser.	Used for inspecting the target webpage's structure, specifically locating <form> and <input> tags to determine attack vectors and submission methods.

Steps Involved in Building the Project

The core functionality resides in the scan_url function, which follows a defined methodology:

1. Web Page Retrieval and Form Discovery

The scan_url(url) function first uses requests.get(url) to fetch the HTML content. BeautifulSoup then parses this content. The parser searches for all <form> elements, extracting two critical pieces of information for each:

- **action**: The URL endpoint where the form data should be submitted.
- **method**: The HTTP method to use (get or post).
- **inputs**: The names of all input fields within the form, which define the data structure for the attack payload.

2. XSS Vulnerability Testing

Cross-Site Scripting (XSS) occurs when an application returns un-sanitized user input that is then executed by the browser. This project tests for the most common type: Reflected XSS.

- **Payloads:** Simple XSS vectors like "<script>alert(1)</script>" are prepared.
- **Injection:** For each form, the scanner submits the test payload in place of one of the form's input values.
- **Detection Logic:** The scanner checks if the exact, raw payload string (e.g., "<script>alert(1)</script>") is directly reflected back in the HTML source code of the response (if payload in test_response.text). The presence of the un-encoded payload suggests a high probability that a malicious script could be executed by a victim's browser.

3. SQL Injection (SQLi) Testing

SQL Injection occurs when an attacker can interfere with the queries an application makes to its database.

- **Payloads:** SQLi test strings like "' OR '1'='1" (a classic bypass attempt) and "1; DROP TABLE users" are used.
- **Injection:** Similar to XSS testing, the SQLi payloads replace a form input value and are submitted to the server.
- **Detection Logic:** The scanner looks for indicators of a database error being exposed to the user. Keywords like 'error', 'syntax', 'exception', or 'mysql_fetch_array()' in the response body are treated as a critical finding. These exposed errors confirm that the server is processing the malicious input and leaking information that an attacker could use for deeper exploitation.

4. Result Aggregation and Presentation

All detected vulnerabilities (type, target URL, payload, and assigned severity) are collected into a list and passed to the Flask application's result template (two.html) for structured display to the user.

Conclusion

This educational project successfully implements a small, proof-of-concept web vulnerability scanner, providing valuable insight into the mechanics of XSS and SQLi vulnerability detection. By seeing how simple payloads can test for security flaws, developers can better understand the importance of **input validation, output encoding, and parameterized queries** in their own code.

Crucial Ethical Reminder: Using automated tools like this on any system you do not explicitly own or have permission to test is considered illegal and unethical hacking. Always practice within a controlled, authorized environment (like a local test server or a bug bounty program) to ensure responsible learning.