Project: SQLi Sentinel - A Web Vulnerability Scanner

Phase 5: Deployment

requirements.txt

Create a file named requirements.txt and add the following lines to it. This file lists all the necessary third-party libraries for the project.
requests
beautifulsoup4

To install these dependencies, a user would run the command: pip install -r requirements.txt

README.md

Create a file named README.md. This is the main documentation for your project.
SQLi Sentinel - A Simple SQL Injection Scanner

SQLi Sentinel is a command-line tool written in Python to detect basic SQL injection vulnerabilities in web applications. It was built as an educational project to demonstrate the fundamentals of web crawling and vulnerability scanning.

Disclaimer: This tool is for educational purposes only. Do not use it on any website or application that you do not own or have explicit, written permission to test. Unauthorized scanning is illegal.

Features

- **Web Crawler:** Discovers pages on the target website up to a specified depth.
- **Vector Identification:** Finds HTML forms and URL parameters to test.
- **Error-Based Detection:** Identifies vulnerabilities by looking for common SQL error messages in server responses.

Installation

1. Clone this repository or download all the `.py` files into a single directory.
2. Install the required Python libraries using pip:
   ```bash
   pip install -r requirements.txt
   ```

Usage

Run the scanner from your terminal. You must provide a starting URL.

```bash
python sqli_sentinel.py --url <target-url> [--depth <number>]
```

Examples

- **Scan a single page:**
  python sqli_sentinel.py --url http://testphp.vulnweb.com/login.php --depth 0

- **Crawl** and scan a website up to a depth of **2:**
  python sqli_sentinel.py --url http://testphp.vulnweb.com --depth 2

Phase 6: Maintenance

The project is complete, but like any software, it can always be improved. This phase is about looking to the future.

Current Limitations

- **Error-Based Only:** The tool currently only detects error-based SQLi. It does not test for the more subtle boolean-based or time-based vulnerabilities we planned for.
- **No JavaScript Rendering:** The crawler cannot find links or forms that are created dynamically by JavaScript.
- **No Authentication:** The scanner cannot log in to websites to test pages that are behind a login wall.

Future Improvements

Here are some great next steps if you want to continue developing this tool:

1. **Implement More Scan Types:** Add the scan_for_boolean_based_sqli() and scan_for_time_based_sqli() methods to the scanner.py module. This is the most valuable next step.
2. **Add Multi-threading:** To speed up the scan, you could use Python's threading module to test multiple URLs or forms at the same time.
3. **Expand Payloads:** Research and add more database-specific payloads to the scanner.py module to detect vulnerabilities in different systems like PostgreSQL, MSSQL, etc.

4. **Improve Reporting:** Instead of just printing to the console, modify