Project: SQLi Sentinel - A Web Vulnerability Scanner

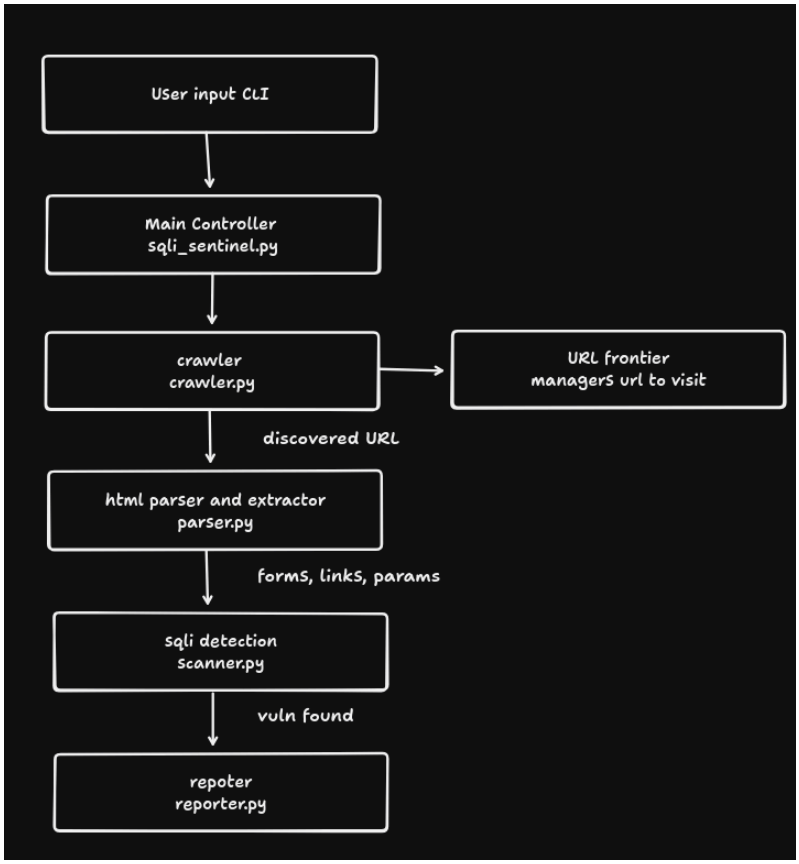
Phase 2: System Design

1. Introduction

This document describes the system architecture for **SQLi Sentinel**. It breaks down the application into logical modules, defines the responsibilities of each module, and illustrates the flow of data between them. This design directly addresses the requirements established in Phase 1.

2. System Architecture Overview

We will use a modular architecture. Each component will be a separate part of the codebase (e.g., its own Python file or a distinct class) with a well-defined purpose. This makes the system easier to build, test, and maintain.
Here is a high-level visual representation of the architecture:



3. Module Descriptions

3.1. Main Controller (sqli_sentinel.py)

- **Responsibility:** This is the entry point of our application.
- **Functionality:**
    - Parses command-line arguments (URL, depth, etc.) using Python's argparse library.
    - Initializes and coordinates all other modules.
    - Manages the overall workflow of the scan.
    - Instantiates the Reporter to handle output.

3.2. Crawler / Spider (crawler.py)

- **Responsibility:** To discover all accessible pages on the target website.
- **Functionality:**
    - Takes a starting URL and a scan depth from the Main Controller.
    - Uses the requests library to fetch the HTML content of a page.
    - Passes the HTML to the Parser to extract new links.
    - Maintains a "URL Frontier" (a queue of URLs to visit) and a set of already visited URLs to prevent re-scanning and getting stuck in loops.
    - For each unique URL found, it will pass it to the Parser for vulnerability vector identification.

3.3. HTML Parser & Vector Extractor (parser.py)

- **Responsibility:** To find all potential points of attack (vectors) on a given HTML page.
- **Functionality:**
    - Uses the BeautifulSoup4 library to parse HTML content.
    - **Extracts Links:** Finds all <a> tags and their href attributes to send back to the Crawler.
    - **Extracts Forms:** Finds all <form> tags. For each form, it identifies the action (where the form submits to), the method (GET or POST), and all of its input fields (<input>, <textarea>, etc.). It will package this information into a structured object.
    - **Extracts URL Parameters:** It will use Python's urllib.parse library to break down URLs and identify parameters and their default values.

3.4. SQLi Detection Engine (scanner.py)

- **Responsibility:** To test each identified vector for SQL injection vulnerabilities. This is the core logic engine.
- **Functionality:**
    - It will receive vector information (like a form object or a URL with parameters) from the Main Controller.
    - It will systematically execute the detection techniques defined in our requirements:
        - **scan_for_error_based():** Submits payloads like ' and checks the response for database error strings.
        - **scan_for_boolean_based():** Submits two payloads (...OR 1=1 and ...OR 1=2), gets both response pages, and compares them for differences.

        - **scan_for_time_based():** Submits a payload with a SLEEP command and measures the response time against a pre-calculated baseline.
    - If a test is positive, it will immediately report the finding to the Reporter and stop testing that specific vector.

3.5. Reporter (reporter.py)

- **Responsibility:** To manage all output to the user.
- **Functionality:**
    - Prints status updates to the console (e.g., [+] Crawling: http://example.com/page).
    - Prints discovered vulnerabilities in a clear, readable format (e.g., [!] SQLi Vulnerability Found!).
    - At the end of the scan, it will collate all findings and print a final summary report.

4. Data Flow

1. User runs python sqli_sentinel.py --url <URL>.
2. **Main Controller** starts the **Crawler** with the URL.
3. **Crawler** requests the URL, gets HTML, and sends it to the **Parser**.
4. **Parser** extracts new links and sends them back to the **Crawler**'s URL frontier.
5. **Parser** also extracts forms and URL parameters (vectors) and sends them to the **Main Controller**.
6. For each vector, the **Main Controller** instructs the **Scanner** to begin testing.
7. **Scanner** runs its various tests on the vector. If a vulnerability is found, it notifies the **Reporter**.
8. **Reporter** prints the finding to the console.
9. The **Crawler** picks the next URL from its frontier and repeats the process until the frontier is empty or the depth limit is reached.
10. Once the scan is complete, the **Reporter** prints the final summary.

5. Key Libraries and Tools

- **requests:** For all HTTP communication (making GET and POST requests).
- **beautifulsoup4:** For robust and easy HTML parsing.
- **argparse:** For creating a professional command-line interface.
- **urllib.parse:** For URL manipulation and parsing.