Project: SQLi Sentinel - A Web Vulnerability Scanner

Phase 3: Implementation

This phase involved translating the system design into functional Python code. Each module was developed independently and then integrated to form the complete application.

Code Modules

1. **reporter.py**: A utility module created to handle all console output. It uses color-coded messages to distinguish between general status updates ([*]), informational messages ([+]), and critical vulnerability alerts ([!]). This keeps the main application logic clean from print statements.
2. **parser.py**: This module uses the BeautifulSoup4 library to parse the HTML content of web pages. Its key functions are:
   - extract_links(): Finds all <a> tags and resolves their href attributes to absolute URLs, ensuring the crawler stays within the target domain.
   - extract_forms(): Finds all <form> tags and gathers their action URL, submission method (GET/POST), and all of their named input fields.
3. **crawler.py**: This module is responsible for navigating the target website. It starts with a single URL and maintains a queue (frontier) of pages to visit. It uses the parser to find new links and keeps a set of visited URLs to avoid redundant work and infinite loops.
4. **scanner.py**: The core of the tool. It contains the logic for testing potential vulnerabilities. In this implementation, it focuses on **Error-Based SQLi**.
   - It takes the forms and URLs discovered by the crawler.
   - It injects a single quote payload (') into each form field and URL parameter.
   - It then inspects the server's response HTML for a list of predefined SQL error strings (e.g., "you have an error in your sql syntax"). If an error is found, it flags the vector as vulnerable.
5. **sqli_sentinel.py**: The main controller and entry point. It uses the argparse library to handle command-line arguments (--url, --depth). It initializes all other modules and orchestrates the overall workflow:
   1. Starts the **Crawler**.
   2. Receives the discovered forms and URLs from the crawler.
   3. Passes each discovered vector to the **Scanner**.
   4. Tells the **Reporter** to print the final summary.

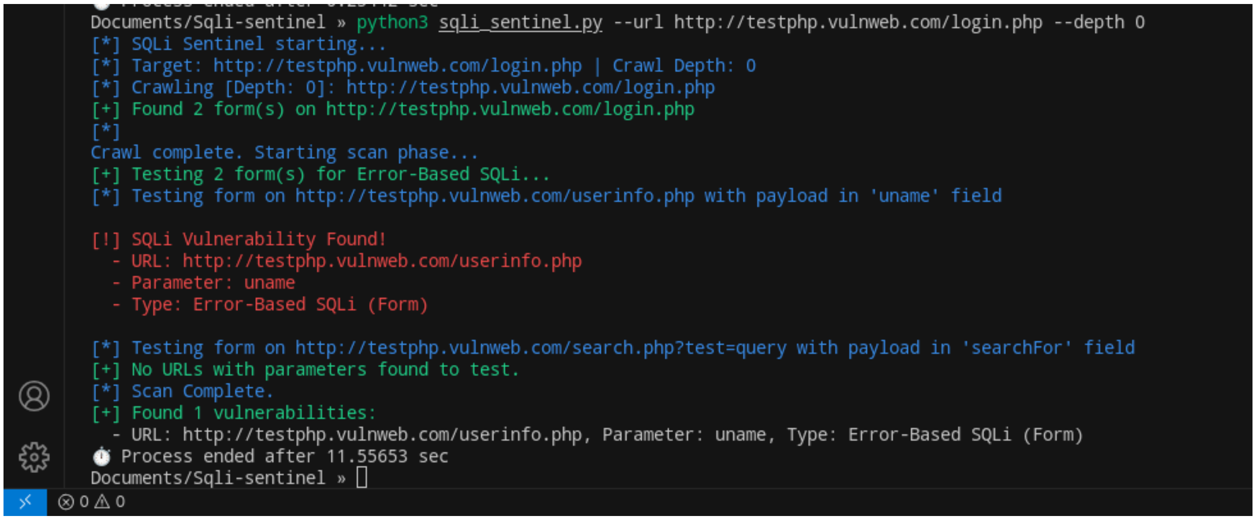
Phase 4: Testing

The purpose of this phase was to verify that the implemented tool functions correctly and can identify a real-world vulnerability.

Test Case & Procedure

- **Objective:** To confirm that the tool can successfully detect an Error-Based SQL injection vulnerability in an HTML form.
- **Test Environment:** A publicly available, intentionally vulnerable web application was used for the test. **Target URL:** http://testphp.vulnweb.com/login.php.
- **Execution:** The tool was executed from the command line with the following command, instructing it to only scan the specified URL and not crawl any further (--depth 0): python3 sqli_sentinel.py --url http://testphp.vulnweb.com/login.php --depth 0

Results



```
Documents/Sqli-sentinel » python3 sqli_sentinel.py --url http://testphp.vulnweb.com/login.php --depth 0
[*] SQLi Sentinel starting...
[*] Target: http://testphp.vulnweb.com/login.php | Crawl Depth: 0
[*] Crawling [Depth: 0]: http://testphp.vulnweb.com/login.php
[+] Found 2 form(s) on http://testphp.vulnweb.com/login.php
[*]
Crawl complete. Starting scan phase...
[+] Testing 2 form(s) for Error-Based SQLi...
[*] Testing form on http://testphp.vulnweb.com/userinfo.php with payload in 'uname' field

[!] SQLi Vulnerability Found!
  - URL: http://testphp.vulnweb.com/userinfo.php
  - Parameter: uname
  - Type: Error-Based SQLi (Form)

[*] Testing form on http://testphp.vulnweb.com/search.php?test=query with payload in 'searchFor' field
[+] No URLs with parameters found to test.
[*] Scan Complete.
[+] Found 1 vulnerabilities:
  - URL: http://testphp.vulnweb.com/userinfo.php, Parameter: uname, Type: Error-Based SQLi (Form)
⏱ Process ended after 11.55653 sec
Documents/Sqli-sentinel » 
```

- **Outcome:** The test was **successful**.
- **Observations:**
    1. The tool correctly identified the login form on the page.
    2. It submitted the form with a single quote payload in the uname parameter.
    3. The server responded with a page containing a MySQL error message.
    4. The scanner successfully detected the error string in the response.
    5. The reporter printed a vulnerability alert to the console, correctly identifying the URL, the vulnerable parameter (uname), and the type of vulnerability (Error-Based SQLi).

This successful test validates that the core functionality of the application works as designed.