ĐẠI HỌC BÁCH KHOA HÀ NỘI

PROJECT 2

Phát triển công cụ phát hiện lỗ hổng SQL Injection

PHAN THẾ TOÀN
toan.pt225415@sis.hust.edu.vn

Ngành Công nghệ thông tin

Giảng viên hướng dẫn: TS. Trần Quang Đức
Trần Đình Kiến Giang _____

Chữ kí GVHD

Chương trình đào tạo: Kỹ thuật máy tính

Trường: Công nghệ Thông tin và Truyền thông

HÀ NỘI, 06/2025

MỤC LỤC

DANH MỤC HÌNH VẼ

CHƯƠNG 1. GIỚI THIỆU SQL INJECTION

1.1 Giới thiệu về SQL Injection

SQL Injection là một loại hình tấn công vào website nguy hiểm và phổ biến nhất, gây những thiệt hại đáng kể cho doanh nghiệp và tổ chức trong nhiều năm qua, bởi vì nó có thể dẫn tới việc lộ thông tin nhạy cảm được lưu trữ trong cơ sở dữ liệu của ứng dụng, bao gồm các thông tin như username, mật khẩu, tên, địa chỉ, số điện thoại hay thẻ tín dụng**hovy1993automated**.

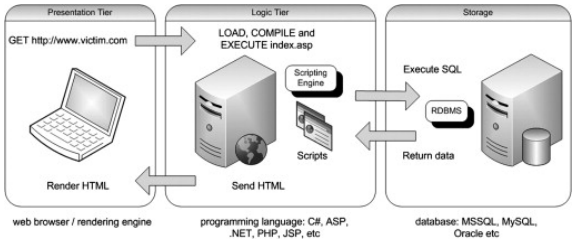
Hiểu theo một cách đơn giản nhất, SQL Injection là lỗ hổng xảy ra khi có kẻ tấn công can thiệp vào các truy vấn SQL (Structured Query Language) mà ứng dụng gửi tới cơ sở dữ liệu back-end. Kẻ tấn công có thể lợi dụng cú pháp và chức năng của SQL, tính linh hoạt của hệ điều hành và cơ sở dữ liệu để thao túng dữ liệu được gửi tới cơ sở dữ liệu. Không chỉ ảnh hưởng tới các ứng dụng web, bất kỳ code nào chấp nhận đầu vào từ nguồn không đáng tin cậy và sử dụng đầu vào đó để tạo câu lệnh SQL động đều có thể bị tấn công.

Trong quá khứ, SQL injection được sử dụng nhiều để chống lại cơ sở dữ liệu từ máy chủ, tuy nhiên với HTML5 hiện nay, kẻ tấn công có thể thực thi JavaScript hoặc mã khác để tương tác với cơ sở dữ liệu phía máy khách nhằm đánh cắp dữ liệu. Tương tự với những ứng dụng di động, các ứng dụng độc hại hoặc tập lệnh của phía máy khách cũng có thể bị tấn công theo cách tương tự.

“SQL injection có lẽ đã tồn tại từ khi các cơ sở dữ liệu SQL lần đầu tiên được kết nối với các ứng dụng Web. Tuy nhiên, Rain Forest Puppy được cho là đã phát hiện ra nó – hoặc ít nhất là đưa nó đến sự chú ý của công chúng. Vào ngày Giáng sinh năm 1998, Rain Forest Puppy đã viết một bài báo có tựa đề [1] cho Phrack, một tạp chí điện tử được viết bởi và dành cho các hacker. Rain Forest Puppy cũng đã phát hành một cảnh báo về SQL injection [2] có vào đầu năm 2000, chi tiết cách SQL injection được sử dụng để tấn công một trang web nổi tiếng. Kể từ đó, nhiều nhà nghiên cứu đã phát triển và tinh chỉnh các kỹ thuật khai thác SQL injection. Tuy nhiên, cho đến ngày nay, nhiều nhà phát triển và chuyên gia bảo mật vẫn không hiểu rõ về nó.”

1.2 Cách hoạt động của Ứng dụng Web

Đối tượng kiểm thử chủ yếu của chúng ta là **ứng dụng web** - thứ mà chúng ta sử dụng hàng ngày để phục vụ cho công việc, xã giao, mua sắm, tìm kiếm thông tin,... Ứng dụng web có nhiều loại và kích thước khác nhau, tuy nhiên, chúng đều có những đặc điểm chung như sau: Các ứng dụng web đều tương tác và dựa trên



Hình 1.1: Cấu trúc ba tầng Ứng dụng Web

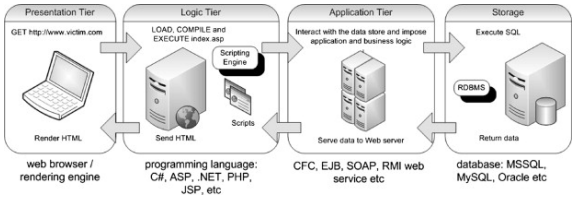
cơ sở dữ liệu để đưa ra kết quả. Ứng dụng web thường gồm ba tầng trong 1.1: Presentation Tier (Tầng biểu diễn) bao gồm các trình duyệt web, Logic Tier (Tầng Logic) là các ngôn ngữ lập trình như C#, ASP, .NET, PHP, JSP,... và Storage (Tầng lưu trữ) bao gồm cơ sở dữ liệu như Microsoft SQL, My SQL, Oracle,...[3]

Nguyên tắc cơ bản của kiến trúc ba tầng là tầng trình bày sẽ không bao giờ giao tiếp trực tiếp với tầng dữ liệu, mà phải thông qua tầng trung gian. Điều này có nghĩa là Tầng biểu diễn sẽ chịu trách nhiệm hiển thị các thông tin và là cầu nối giữa người dùng và các tầng khác của trình duyệt, còn Tầng logic sẽ đóng vai trò là Tầng trung gian, kiểm soát chức năng của ứng dụng bằng cách xử lý chi tiết các yêu cầu được đưa ra, truy vấn/cập nhật/chỉnh sửa cơ sở dữ liệu, hay còn gọi là Tầng lưu trữ. Tầng lưu trữ giữ dữ liệu độc lập với các tầng còn lại, điều này cũng giúp cái thiện khả năng mở rộng và hiệu suất.

Quy trình hoạt động của một ứng dụng web cơ bản sẽ diễn ra như sau: Người dùng sẽ kết nối với website, thao tác với Tầng biểu diễn, sau đó máy chủ web nằm ở Tầng logic sẽ tải tập lệnh từ hệ thống tệp và chuyển qua công cụ xử lý, nơi tập lệnh được phân tích và thực thi. Tập lệnh sẽ mở kết nối với Tầng lưu trữ bằng cách thực hiện các truy vấn và cập nhật đối với cơ sở dữ liệu. Cơ sở dữ liệu sẽ trả lại dữ liệu cho máy chủ web (Tầng logic) và xử lý kịch bản trong tầng này, sau đó thực hiện các logic nghiệp vụ để gửi lại trang web dưới dạng HTML cho người dùng. Trình duyệt web của người dùng sẽ được hiển thị kết quả và trình bày theo dạng đồ hoạ, một cách trực quan và dễ hiểu. Điều này chỉ diễn ra vài giây đối với người dùng[4].

Kiến trúc ba tầng này đơn giản nhưng cũng khó mở rộng và phát triển, chính vì vậy, họ đã ứng dụng kiến trúc bốn tầng (phức tạp hơn) để khắc phục hạn chế này được mô tả trong hình 1.2.

Đây là mô hình ví dụ cho mô hình phát triển ứng dụng n tầng (n-tier application development paradigm) có khả năng mở rộng và bảo trì tốt hơn dạng cơ bản. Ở dạng phức tạp hơn, có thêm Tầng Ứng dụng (Application Tier) tham gia vào quy trình hoạt động, nằm ở vị trí trung gian giữa Tầng Logic và Tầng Lưu trữ. Tầng



Hình 1.2: Cấu trúc bốn tầng Ứng dụng Web

Ứng dụng sẽ cung cấp giao diện lập trình ứng dụng (API) để các tầng khác gọi tới, xử lý nghiệp vụ kinh doanh (Business logic), kết nối với nhiều nguồn dữ liệu khác nhau. Kiến trúc nhiều tầng chính là việc chia Tầng Ứng dụng thành nhiều phần, hay nhiều tầng, được giao nhiệm vụ chung hoặc cụ thể, mỗi tầng có một vai trò riêng để dễ dàng mở rộng ứng dụng và phân chia công việc phát triển tốt hơn cho các nhà phát triển, làm cho ứng dụng dễ đọc và dễ tái sử dụng hơn. Điều này khiến mô hình nhiều tầng được linh hoạt, và cho phép nhiều tầng được tách biệt về mặt logic và triển khai theo nhiều cách khác nhau.

1.3 Cách hoạt động của SQL Injection

SQL Injection (SQLi) là một kỹ thuật tấn công nhằm khai thác lỗ hổng bảo mật trong các ứng dụng web. Lỗ hổng này cho phép kẻ tấn công chèn mã SQL độc hại vào truy vấn của ứng dụng, từ đó thực hiện các hành động trái phép như truy cập dữ liệu nhạy cảm, sửa đổi thông tin và nghiêm trọng nhất là chiếm quyền điều khiển hệ thống.

Dễ hiểu hơn, chúng ta có thể xem xét trường hợp một ứng dụng được quản lý từ xa thông qua một hệ thống quản lý nội dung (CMS). Ứng dụng CMS yêu cầu người dùng cung cấp username và password hợp lệ trước khi truy cập vào các chức năng khác. Dưới đây là mã cho script `login.php`:

```
1  $conn = mysql_connect("localhost", "username",
    ↪  "password");
2  $query = "SELECT userid FROM CMSUsers WHERE user =
    ↪  '$_GET["user"]
3  ' AND password = '$_GET["password"]'";
4  $result = mysql_query($query);
5  $rowcount = mysql_num_rows($result);
6  if ($rowcount != 0) {
7      header("Location: admin.php");
8  }
9  else {
10     die('Incorrect username or password, please try
        ↪  again.');
11 }
```

Hình 1.3: Mã nguồn Login.php

Khi truy cập vào hệ thống và đăng nhập qua biểu mẫu trên web, nếu ứng dụng xây dựng một truy vấn SQL để kiểm tra thông tin đăng nhập của người dùng như sau[5]:

```
SELECT userID FROM Users WHERE username = 'user_input'
AND password = 'user_password';
```

Hình 1.4: Query đăng nhập

Nếu user_input và user_password được chèn một cách trực tiếp vào truy vấn mà không được xử lý, kẻ tấn công có thể nhập vào một dòng mã khác để thay đổi truy vấn SQL thành:

```
SELECT userID FROM Users WHERE username = '' OR '1'='1'
AND password = 'user_password';
```

Hình 1.5: Payload tấn công

Điều kiện '1'='1' luôn đúng, khiến cơ sở dữ liệu trả về ID của một hoặc nhiều người dùng nào, bất kể mật khẩu nhập vào, từ đó có thể bỏ qua bước xác thực trên web.

CHƯƠNG 2. CÁC KIỂU TẤN CÔNG SQL INJECTION

Dựa trên phương thức truy cập và khả năng gây hại của tấn công và trong công cụ này sẽ tập chung để giải quyết 2 lỗi SQLi phổ biến.

Lỗ hổng **In-band SQLi** là dạng tấn công phổ biến nhất và cũng dễ để khai thác lỗ hổng SQL Injection nhất. Kẻ tấn công sử dụng cùng một kênh truy cập (thường là HTTP) để gửi dữ liệu đầu vào và nhận kết quả phản hồi từ ứng dụng. Kết quả phản hồi có thể chứa dữ liệu bị rò rỉ do SQL injection, giúp kẻ tấn công thu thập thông tin nhạy cảm. Loại tấn công này hoạt động dựa trên hai kỹ thuật tấn công chính là Error-based (sử dụng câu lệnh không hợp lệ để gây lỗi trong câu lệnh SQL, sau đó lấy thông tin từ các thông báo lỗi) và Union-based (sử dụng câu lệnh UNION trong câu lệnh SELECT để truy vấn nhiều bảng cùng một lúc và lấy dữ liệu từ các bảng này)[6].

Lỗ hổng **Inferential SQLi** kẻ tấn công không trực tiếp nhận kết quả từ truy vấn SQL bị lỗi. Thay vào đó, họ dựa vào các biểu hiện gián tiếp, không rõ ràng của hệ thống, chẳng hạn như thời gian phản hồi của ứng dụng hoặc thông báo lỗi, để suy ra thông tin về dữ liệu bị rò rỉ. Inferential SQLi thường khó khai thác hơn In-band SQLi, nhưng nó có thể hiệu quả trong các trường hợp kẻ tấn công bị hạn chế quyền truy cập vào ứng dụng[7].

2.1 In-band SQLi

2.1.1 Error-based SQL Injection

Error-based SQL Injection là một kỹ thuật tấn công SQL Injection, trong đó kẻ tấn công lợi dụng các thông báo lỗi do máy chủ cơ sở dữ liệu (CSDL) trả về để thu thập thông tin về cấu trúc của CSDL. Mặc dù các chuỗi lỗi này có vẻ vô hại, chúng có thể bị khai thác để liệt kê các thuộc tính, loại, phiên bản và thậm chí cả dữ liệu của CSDL. Bằng cách phân tích các thông báo lỗi này, kẻ tấn công có thể hiểu rõ hơn về cách ứng dụng và CSDL tương tác, từ đó xây dựng các cuộc tấn công phức tạp hơn để trích xuất dữ liệu.

Để thực hiện tấn công, kẻ tấn công chèn các cú pháp SQL không hợp lệ vào dữ liệu đầu vào của người dùng, chẳng hạn như dấu nháy đơn (`'`), dấu nháy kép (`"`), hoặc các toán tử logic như `AND`, `OR`.

a, Ví dụ cơ bản

Xét một URL chấp nhận một tham số đầu vào:

```
https://example.com/index.php?item=123
```

Kẻ tấn công có thể thử thêm một dấu nháy đơn vào cuối giá trị của tham số:

```
https://example.com/index.php?item=123'
```

Nếu ứng dụng dễ bị tấn công, CSDL có thể trả về một thông báo lỗi tương tự như sau:

```
You have an error in your SQL syntax; check the manual that
corresponds to your MySQL server version for the right syntax
to use near ''''.
```

Thông báo lỗi này cung cấp cho kẻ tấn công nhiều thông tin giá trị:

- Hệ quản trị CSDL: Lỗi này xác nhận rằng CSDL đang được sử dụng là MySQL.
- Điểm tiêm nhiễm (Injection Point): Lỗi được gây ra bởi dấu nháy đơn, cho thấy tham số `item` dễ bị tấn công SQL Injection.

Từ đó, kẻ tấn công biết rằng máy chủ không xử lý đầu vào một cách an toàn và có thể tự động hóa quá trình khai thác bằng các công cụ như `grep` hoặc `extract` để thử nghiệm nhiều cú pháp SQL khác nhau.

b, Ví dụ nâng cao (Vendor-specific)

Khi đã biết được nhà cung cấp và phiên bản của CSDL, kẻ tấn công có thể thực hiện các truy vấn dành riêng cho nhà cung cấp đó để khai thác thông tin chi tiết hơn.

Xét lại URL tương tự, giả sử kẻ tấn công biết rằng CSDL là Oracle 10g. Họ có thể gửi một payload như sau:

```
https://example.com/index.php?item=123||
UTL_INADDR.GET_HOST_NAME((SELECT user FROM DUAL))--
```

Payload này bao gồm hai yếu tố dành riêng cho Oracle:

- `UTL_INADDR.GET_HOST_NAME()`: Một hàm của Oracle cố gắng phân giải tên máy chủ.
- `SELECT user FROM DUAL`: Một truy vấn con để lấy tên người dùng CSDL hiện tại. Bảng `DUAL` là một bảng đặc biệt luôn tồn tại trong Oracle.

Hàm `GET_HOST_NAME()` sẽ nhận kết quả của truy vấn con (ví dụ: tên người dùng là `DAVID`) làm đối số. Vì `DAVID` không phải là một tên máy chủ hợp lệ, CSDL sẽ trả về một lỗi:

```
ORA-29257: host DAVID unknown
```

Lỗi này tiết lộ tên người dùng CSDL hiện tại là `DAVID`, xác nhận rằng có thể thực thi các truy vấn con và rò rỉ dữ liệu thông qua thông báo lỗi. Từ đây, kẻ tấn công có thể sửa đổi payload để khám phá các thông tin khác như tên bảng, tên cột hoặc dữ liệu cụ thể.

2.1.2 Union-based SQL Injection

Union-based SQL Injection là một kỹ thuật tấn công SQL Injection trong đó kẻ tấn công sử dụng toán tử `UNION` để hợp nhất kết quả từ một câu lệnh `SELECT` độc hại với kết quả của một câu lệnh gốc hợp lệ. Kỹ thuật này cho phép kẻ tấn công trích xuất dữ liệu từ các bảng khác nhau trong cơ sở dữ liệu (CSDL) và hiển thị chúng trực tiếp trên trang web của ứng dụng.

Để cuộc tấn công thành công, câu lệnh `SELECT` do kẻ tấn công tạo ra phải tuân thủ hai điều kiện nghiêm ngặt của toán tử `UNION`:

1. Số lượng cột trong câu lệnh `SELECT` của kẻ tấn công phải bằng với số lượng cột trong câu lệnh `SELECT` gốc.
2. Kiểu dữ liệu của các cột tương ứng trong hai câu lệnh phải tương thích với nhau.

Do những yêu cầu này, kẻ tấn công thường phải kết hợp với kỹ thuật Error-based SQL Injection trước đó để thăm dò và xác định đúng số lượng cũng như kiểu dữ liệu của các cột trong truy vấn gốc.

a, Quy trình tấn công

Một cuộc tấn công Union-based SQLi điển hình thường diễn ra theo các bước sau:

1. Xác định số lượng cột: Kẻ tấn công cần tìm ra số cột mà truy vấn gốc trả về. Có hai phương pháp phổ biến:

- Sử dụng `ORDER BY`: Kẻ tấn công chèn mệnh đề `ORDER BY` với chỉ số cột tăng dần (`ORDER BY 1-`, `ORDER BY 2-`, ...) cho đến khi ứng dụng trả về lỗi. Chỉ số cuối cùng không gây lỗi chính là số lượng cột.
- Sử dụng `UNION SELECT`: Kẻ tấn công thử các câu lệnh `UNION SELECT` với số lượng giá trị `NULL` tăng dần (`UNION SELECT NULL-`, `UNION SELECT NULL,NULL-`, ...) cho đến khi không còn nhận được lỗi.

2. Xác định kiểu dữ liệu của cột: Sau khi biết số lượng cột, kẻ tấn công cần tìm ra những cột có kiểu dữ liệu chuỗi (string) để có thể hiển thị dữ liệu văn bản. Quá trình này được thực hiện bằng cách thay thế từng giá trị `NULL` bằng một chuỗi ký tự (ví dụ: `'a'`) và gửi truy vấn. Nếu truy vấn thành công, cột ở vị trí đó có kiểu dữ

liệu là chuỗi[5].

```
' UNION SELECT 'a', NULL, NULL--
' UNION SELECT NULL, 'a', NULL--
```

3. Trích xuất dữ liệu: Khi đã có đủ thông tin về cấu trúc truy vấn, kẻ tấn công sẽ thay thế các giá trị `NULL` hoặc chuỗi ký tự thử nghiệm bằng một câu lệnh `SELECT` độc hại để lấy dữ liệu nhạy cảm từ các bảng khác, chẳng hạn như bảng người dùng.

b, Ví dụ minh hoạ

Giả sử một ứng dụng có URL để lọc sản phẩm như sau:

```
http://darwin.com/vehicles.php?category=Sedan
```

Truy vấn gốc tương ứng là: `SELECT type, description FROM vehicles WHERE category = 'Sedan'`. Truy vấn này trả về 2 cột.

Sau khi đã thực hiện các bước thăm dò, kẻ tấn công có thể xây dựng một payload để lấy cắp tên người dùng và mật khẩu từ bảng `users`:

```
http://darwin.com/vehicles.php?category=Sedan' UNION SELECT username, password FROM users--
```

Để trích xuất dữ liệu từ nhiều cột trong khi truy vấn gốc chỉ có một cột phù hợp, kẻ tấn công có thể ghép các giá trị lại với nhau:

```
' UNION SELECT NULL, username || '~' || password FROM users--
```

Do khả năng đọc trực tiếp dữ liệu từ CSDL, Union-based SQL Injection được xem là một trong những hình thức tấn công SQL Injection nguy hiểm và hiệu quả nhất.

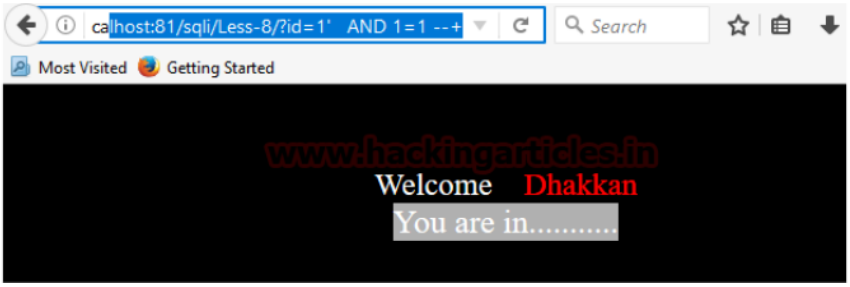
2.2 Inferential SQLi

2.2.1 Blind-boolean-based SQLi

Blind-boolean-based SQLi là một kỹ thuật tiêm mã Inferential SQLi dựa trên việc gửi một truy vấn SQL đến cơ sở dữ liệu, buộc ứng dụng trả về kết quả khác nhau tùy thuộc vào việc truy vấn trả về TRUE hay FALSE.

Tuỳ vào kết quả, nội dung trong phản hồi HTTP sẽ thay đổi hoặc giữ nguyên. Điều này cho phép kẻ tấn công suy luận xem gói dữ liệu được sử dụng trả về true hay false, ngay cả khi không có dữ liệu nào từ cơ sở dữ liệu được trả về. Kiểu tấn công này thường chậm (đặc biệt đối với các cơ sở dữ liệu lớn) vì kẻ tấn công cần liệt kê từng ký tự trong cơ sở dữ liệu.

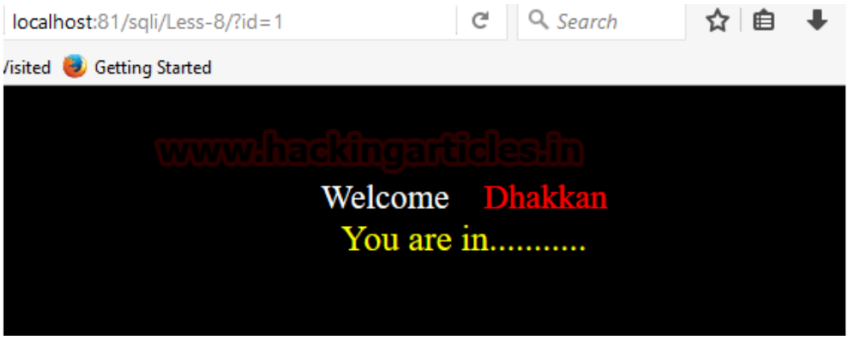
Minh hoạ cho Blind-based SQLi, Raj Chandel's Blog đã đưa ra tình huống



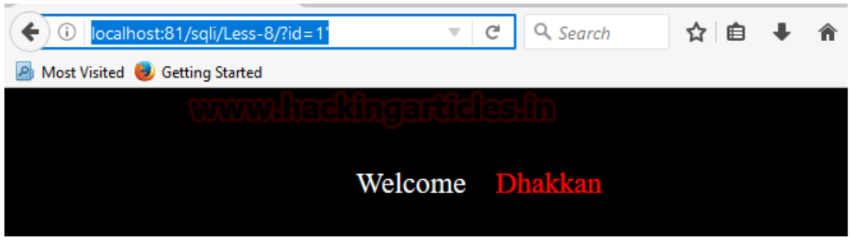
khi chúng ta truy cập `http://localhost:81/sqli/Less-8/?id=1` trên trình duyệt. Điều này sẽ gửi truy vấn sau vào cơ sở dữ liệu:

```
SELECT * from table_name WHERE id = 1
```

Kết quả trả về sẽ hiện dòng chữ "you are in" với màu vàng:



Khi kẻ tấn công cố gắng phá bỏ truy vấn này bằng dấu phẩy (') hoặc các kĩ thuật khác, hắn sẽ không nhận được thông báo lỗi mà thay vào đó là dòng chữ màu vàng sẽ biến mất.

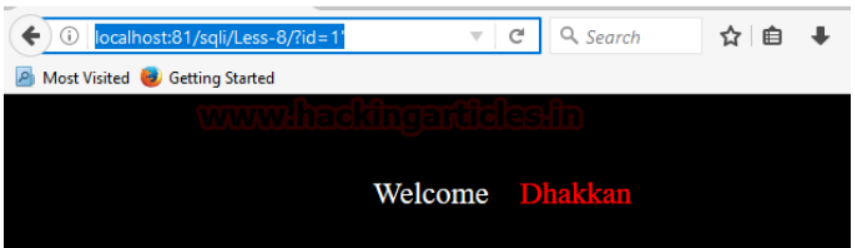


Sau đó, kẻ tấn công sẽ chuyển sang Blind SQLi để đảm bảo rằng truy vấn trả về kết quả TRUE hoặc FALSE:

```
http://localhost:81/sqli/Less-8/?id=1' AND 1=1 -+
```

```
SELECT * from table_name WHERE id=1' AND 1=1
```

Lúc này, cơ sở dữ liệu sẽ kiểm tra điều kiện đã cho xem "1=1" có đúng hay không, nếu truy vấn hợp lệ sẽ trả về TRUE, tức là văn bản "you are in" được hiện lại, tức là truy vấn này hợp lệ.



Trong trường hợp ta sử dụng một điều kiện sai, truy vấn sẽ trả về kết quả FALSE và dòng chữ "you are in" sẽ biến mất:

```
  http://localhost:81/sqli/Less-8/?id1' AND 10 -+
SELECT * from table_name WHERE id1' AND 10
```

Điều này xảy ra là do điều kiện 1=0 luôn sai nên cơ sở dữ liệu đã trả về FALSE.

Thử nghiệm này đã xác nhận ứng dụng web này rất dễ bị tấn công Blind SQLi. Bằng cách sử điều kiện TRUE và FALSE, kẻ tấn công có thể trích xuất thông tin từ cơ sở dữ liệu. Bằng cách tấn công này, kẻ tấn công sẽ xác định được độ dài chuỗi, ký tự trong cơ sở dữ liệu bằng mã ASCII, ký tự đầu tiên của cơ sở dữ liệu, rộng hơn nữa là đoán độ dài và tên bảng.

Chúng ta có thể chèn truy vấn sau để hỏi xem độ dài của chuỗi tên cơ sở dữ liệu có bằng 1 hay không, nếu phản hồi của truy vấn đó là TRUE thì sẽ xuất hiện dòng chữ "you are in" và ngược lại:

```
  http://localhost:81/sqli/Less-8/?id1'
AND (length(database())) = 1 -+
```

Để đoán ký tự đầu tiên của tên của chuỗi, ta có thể sử dụng truy vấn:

```
  http://localhost:81/sqli/Less-8/?id1'
AND (ascii(substr((select database()),1,1))) > 100 -+
```

Lặp lại các truy vấn tương tự với các giá trị ASCII khác nhau cho tới khi nhận được TRUE, kẻ tấn công sẽ nắm được toàn bộ chuỗi ký tự tên của cơ sở dữ liệu.

Trong thực tế, kẻ tấn công thường sử dụng một số tool như SQLbit, SQLmap, Intruder,... để tự động hoá, giúp quá trình khai thác được nhanh chóng và hiệu quả hơn.

2.2.2 Time-based SQLi

Time-based Blind SQL injection là một kỹ thuật tấn công SQLi tiên tiến, được sử dụng để khai thác lỗ hổng bảo mật trong các ứng dụng web dựa trên cơ sở dữ liệu. Kỹ thuật này được áp dụng dựa vào việc đo lường thời gian phản hồi của máy chủ cơ sở dữ liệu, từ đó suy luận thông tin liên quan tới cơ sở dữ liệu.

Để vận dụng Time-based SQLi, kẻ tấn công sẽ tiêm các truy vấn SQL vào ứng dụng web có chứa các hàm hoặc thủ tục gây ra độ trễ thời gian có thể dự đoán dựa trên kết quả truy vấn. Bằng cách đo lường thời gian phản hồi của máy chủ cơ sở dữ liệu, kẻ tấn công sẽ dễ dàng suy luận được thông tin về nội dung của truy vấn được thực thi. Cách tiếp cận suy luận này đặc biệt hữu ích cho những cuộc tấn công *deep blind SQL injection*.

Các cuộc tấn công dựa trên thời gian sẽ sử dụng các hàm/thủ tục gây trễ để thực hiện các kiểm tra cơ bản như xác định xem lỗ hổng có tồn tại hay không.

Bảng 2.1: Các chức năng và ghi chú của DBMS

DBMS	**Chức năng**	**Ghi chú**
MySQL	`SLEEP(time)` `BENCHMARK(count, expr)`	Chỉ có sẵn từ MySQL 5. Thực thi biểu thức cụ thể nhiều lần.
SQL Server	`WAIT FOR DELAY 'hh:mm:ss'` `WAIT FOR TIME 'hh:mm:ss'`	Đình chỉ thực thi trong khoảng thời gian cụ thể. Đình chỉ thực thi truy vấn và tiếp tục khi thời gian hệ thống bằng với tham số.
Oracle		Các cuộc tấn công dựa trên thời gian phức tạp hơn ở Oracle.

Ngoài việc xác định lỗ hổng, Time-based SQLi còn có nhiều tiện ích khác. Khi độ trễ thời gian được tích hợp trong một câu lệnh điều kiện, kẻ tấn công có thể trích xuất thông tin từ cơ sở dữ liệu và lấy cắp chúng. Nói một cách đơn giản, bằng cách thêm một độ trễ thời gian có điều kiện vào truy vấn, kẻ tấn công có thể đặt câu hỏi Yes/No cho cơ sở dữ liệu. Tuỳ thuộc vào việc điều kiện được xác minh hay không, độ trễ thời gian sẽ được thực thi và thời gian phản hồi của máy chủ sẽ bất thường. Điều này cho phép kẻ tấn công biết điều kiện là đúng hay sai.

Ví dụ với MySQL, kẻ tấn công có thể cài đặt độ trễ thời gian một cách khá đơn giản. Cả hàm `SLEEP()` và `BENCHMARK()` đều là các hàm, chúng có thể được tích hợp vào bất kỳ câu lệnh SQL nào. Ví dụ như:

```
  SELECT * FROM products WHERE id1-SLEEP(15)
hay
SELECT * FROM products WHERE id1-BENCHMARK(100000000, rand())
```

Nếu ứng dụng web phản hồi chậm, tức là ứng dụng sử dụng cơ sở dữ liệu MySQL.

Ngoài ra, kẻ tấn công cũng có thể quan tâm tới việc trích xuất một số thông tin

hay xác minh giả định, và hắn sẽ tích hợp độ trễ thời gian vào một câu lệnh điều kiện như sau:

```
SELECT * FROM products WHERE id=1-IF(MID(VERSION(),1,1)
= '5', SLEEP(15), 0)
```

Nếu thời gian phản hồi của máy chủ mất 15 giây trở lên, tức là cơ sở dữ liệu này đang chạy MySQL phiên bản 5.x.

CHƯƠNG 3. PHÂN TÍCH VÀ THIẾT KẾ CÔNG CỤ QUÉT

3.1 Yêu cầu và Tính năng

3.1.1 Yêu cầu chức năng

Công cụ quét lỗ hổng SQL Injection được phát triển với mục tiêu chính là tự động hóa toàn bộ quy trình phát hiện và khai thác các lỗ hổng tiềm ẩn trong các ứng dụng web, đặc biệt tập trung vào các lỗ hổng liên quan đến việc thực thi mã SQL không an toàn. Để đạt được hiệu quả cao trong việc bảo mật hệ thống, công cụ này được thiết kế với các yêu cầu chức năng chi tiết và được triển khai một cách có hệ thống, đảm bảo khả năng ứng dụng thực tế trong nhiều tình huống kiểm thử khác nhau. Các yêu cầu chức năng chính của công cụ bao gồm các khía cạnh quan trọng sau đây, được mô tả chi tiết để hỗ trợ người dùng trong quá trình vận hành và tối ưu hóa hiệu suất.

a, Thu thập dữ liệu đầu vào và Cấu hình

Công cụ này cung cấp cho người dùng một giao diện dòng lệnh (CLI) thân thiện, cho phép chỉ định URL mục tiêu một cách linh hoạt thông qua tham số `-url`, giúp định hướng quá trình quét một cách chính xác và hiệu quả. Ngoài ra, nó còn hỗ trợ việc đọc và xử lý các thiết lập cấu hình từ tệp `config.yaml`, nơi chứa các thông số quan trọng như thời gian timeout, cấu hình proxy để ẩn danh hóa kết nối, tùy chỉnh User-Agent để mô phỏng các thiết bị khác nhau, danh sách các chuỗi payload tấn công được sử dụng trong quá trình quét, cùng với định nghĩa các mục tiêu bổ sung như `additional_targets` và `api_definitions` để mở rộng phạm vi kiểm tra. Để đảm bảo tính chính xác và nhất quán, công cụ thực hiện xử lý và chuẩn hóa URL đầu vào, tự động bổ sung lược đồ (*scheme*) như `http://` hoặc `https://` khi người dùng nhập thiếu, từ đó tránh các lỗi kết nối hoặc hiểu sai thông tin đầu vào.

b, Phân tích và Trích xuất Mục tiêu

Một trong những tính năng nổi bật của công cụ là khả năng phân tích sâu mã nguồn HTML từ các phản hồi HTTP bằng cách sử dụng thư viện mạnh mẽ `BeautifulSoup`. Quá trình này cho phép trích xuất chi tiết các thông tin quan trọng từ các thẻ `<form>`, bao gồm thuộc tính `action` để xác định điểm đích của form, thuộc tính `method` để xác định phương thức gửi dữ liệu (GET hoặc POST), và chi tiết về các trường nhập liệu `<input>` như tên, kiểu dữ liệu, cũng như giá trị mặc định nếu có. Bên cạnh đó, công cụ còn thực hiện phân tích các tham số truy vấn (*query parameters*) có trong URL, chẳng hạn như `id=123` hoặc `page=2`, nhằm xác định các điểm tiềm năng có thể bị tấn công thông qua việc chèn payload. Hơn

nữa, để hỗ trợ kiểm thử các hệ thống API hiện đại, công cụ cho phép quét các endpoint API theo định nghĩa được cung cấp trong tệp `api_definitions` hoặc cấu hình trực tiếp, bao quát các tham số trong query string, form data, hoặc các định dạng body phức tạp như JSON và XML, tăng cường tính linh hoạt trong các kịch bản thực tế.

c, Tạo và Chèn Payload

Quá trình tạo và chèn payload là một phần cốt lõi của công cụ, được thực hiện một cách tự động thông qua việc xây dựng và quản lý danh sách payload từ tệp `data/payloads.txt`. Danh sách này bao gồm nhiều loại payload khác nhau, phục vụ cho các kỹ thuật phát hiện lỗ hổng khác nhau, chẳng hạn như *Error-based* với các chuỗi như `' OR '1'='1'--` hoặc `' UNION SELECT NULL--` để gây ra lỗi cú pháp SQL nhằm lộ thông tin, *Boolean-based* với các payload như `' AND 1=1--` hoặc `' AND 1=0--` để kiểm tra điều kiện đúng/sai dựa trên sự thay đổi nội dung phản hồi, và *Time-based* với các hàm trì hoãn như `' OR SLEEP(5)--` hoặc `' WAITFOR DELAY '0:0:5'--` để phát hiện lỗ hổng dựa trên độ trễ thời gian. Các payload này được chèn một cách thông minh vào các tham số của URL, các trường nhập liệu trong form HTML, hoặc các tham số API được chỉ định trong cấu hình, đảm bảo khả năng bao quát toàn diện các điểm tấn công tiềm tàng.

d, Gửi Yêu cầu HTTP

Công cụ được tích hợp khả năng gửi các yêu cầu HTTP với nhiều phương thức khác nhau như GET, POST, PUT, và DELETE thông qua thư viện `requests`, mang lại sự linh hoạt trong việc tương tác với các hệ thống web khác nhau. Người dùng có thể tùy chỉnh các thuộc tính quan trọng của yêu cầu HTTP, bao gồm thời gian timeout để tránh treo máy trong trường hợp máy chủ phản hồi chậm, cấu hình proxy để bảo vệ danh tính hoặc tăng tốc độ kết nối, và các header HTTP tùy chỉnh để mô phỏng các trình duyệt hoặc bot cụ thể. Một tính năng quan trọng khác là duy trì trạng thái phiên thông qua `requests.Session`, giúp quản lý và lưu trữ cookie một cách hiệu quả, đặc biệt cần thiết trong các kịch bản quét yêu cầu đăng nhập hoặc các ứng dụng web yêu cầu xác thực nhiều giai đoạn. Ngoài ra, công cụ còn được thiết kế để xử lý các mã lỗi HTTP (như 4xx cho lỗi client hoặc 5xx cho lỗi server) và các trường hợp ngoại lệ như timeout hoặc mất kết nối mạng, đảm bảo quá trình quét diễn ra ổn định và không bị gián đoạn.

e, Phân tích Phản hồi và Phát hiện Lỗ hổng

Để phát hiện các lỗ hổng SQL Injection một cách chính xác, công cụ thực hiện phân tích phản hồi HTTP theo ba phương pháp chính. Với kỹ thuật *Error-based*, công cụ so khớp nội dung phản hồi với các mẫu lỗi SQL đã được định nghĩa sẵn

trong `SQL_ERROR_PATTERNS` của tệp `parser.py`, từ đó xác định sự tồn tại của lỗ hổng và loại cơ sở dữ liệu tương ứng như MySQL, PostgreSQL, Oracle, hoặc Microsoft SQL Server. Phương pháp *Boolean-based* tập trung vào việc phân tích sự thay đổi trong nội dung phản hồi, chẳng hạn như sự xuất hiện của các chuỗi đặc trưng như "Welcome" hoặc "Login Successful" khi điều kiện trong payload được thỏa mãn. Trong khi đó, kỹ thuật *Time-based* đo lường thời gian phản hồi của máy chủ, xác định lỗ hổng khi thời gian vượt quá ngưỡng mặc định là 4 giây sau khi chèn payload chứa hàm trì hoãn. Cuối cùng, tất cả các chi tiết về lỗ hổng được phát hiện, bao gồm URL bị tấn công, payload sử dụng, tham số bị ảnh hưởng, mức độ nghiêm trọng, và thời gian phát hiện, đều được lưu trữ một cách có tổ chức trong danh sách `self.vulnerabilities` để phục vụ cho báo cáo sau này.

f, Khám phá Đường dẫn/Tệp tiềm năng (Discovery)

Khi tính năng khám phá được kích hoạt trong cấu hình thông qua tham số `discovery.enabled: true`, công cụ sẽ tận dụng một danh sách từ vựng phong phú (*wordlist*) được lưu trong tệp như `data/common_paths.txt` để thực hiện việc khám phá các endpoint hoặc tệp tin tiềm năng trên máy chủ mục tiêu. Quá trình này bao gồm việc gửi các yêu cầu HEAD hoặc GET đến các URL được tạo ra bằng cách kết hợp URL gốc với các từ khóa trong wordlist cùng các phần mở rộng tệp phổ biến như `.php`, `.html`, hoặc `.asp`, sau đó kiểm tra mã trạng thái HTTP (ví dụ: 200 cho thành công, 403 cho bị cấm) để xác định sự tồn tại của các tài nguyên. Các URL hợp lệ được khám phá sẽ được tự động thêm vào danh sách các mục tiêu để tiếp tục quá trình quét, giúp mở rộng phạm vi kiểm tra một cách hiệu quả và toàn diện.

g, Xử lý Đăng nhập Tự động (Authentication Bypass)

Trong trường hợp tính năng đăng nhập được kích hoạt thông qua tham số `login.enabled: true` trong tệp cấu hình, công cụ sẽ thực hiện gửi các yêu cầu đăng nhập (thông qua phương thức GET hoặc POST) với thông tin tài khoản được cung cấp chi tiết trong các trường như `login.url` và `login.data`. Quá trình này bao gồm việc xác minh kết quả đăng nhập thành công dựa trên các tiêu chí cụ thể được định nghĩa trong `success_criteria`, chẳng hạn như mã trạng thái HTTP 200, sự hiện diện của một cookie phiên cụ thể, hoặc sự xuất hiện của một chuỗi đặc trưng trong nội dung phản hồi. Sau khi đăng nhập thành công, các cookie phiên sẽ được lưu trữ và sử dụng tự động trong các yêu cầu quét tiếp theo, đảm bảo rằng công cụ có thể tiếp cận các khu vực bảo vệ của ứng dụng web mà không cần can thiệp thủ công từ người dùng.

h, Tạo Báo cáo Lỗ hổng

Sau khi hoàn tất quá trình quét, công cụ tiến hành tổng hợp toàn bộ các lỗ hổng đã được phát hiện từ danh sách `self.vulnerabilities` và tạo ra một báo cáo chi tiết dưới định dạng HTML. Báo cáo này được xây dựng dựa trên một mẫu được thiết kế chuyên nghiệp với công cụ Jinja2 (`templates/report.html`), bao gồm các thông tin quan trọng như tên của từng lỗ hổng, mô tả chi tiết về cách thức phát hiện, mức độ nghiêm trọng được phân loại (thấp, trung bình, cao, hoặc nghiêm trọng), payload đã được sử dụng trong quá trình kiểm tra, tham số bị tấn công, URL nơi lỗ hổng được tìm thấy, và thời gian chính xác của việc phát hiện. Ngoài ra, báo cáo còn cung cấp một phần thống kê tổng quan về số lượng lỗ hổng được phát hiện cũng như sự phân bố của chúng theo từng mức độ nghiêm trọng, giúp người dùng có cái nhìn tổng quát và dễ dàng đưa ra các quyết định bảo mật phù hợp.

3.1.2 Yêu cầu phi chức năng

Bên cạnh các yêu cầu chức năng đã nêu, công cụ còn được thiết kế để tuân thủ một loạt các yêu cầu phi chức năng quan trọng, nhằm đảm bảo hiệu quả hoạt động, tính ổn định trong môi trường thực tế, và khả năng mở rộng để thích nghi với các nhu cầu đa dạng trong tương lai. Các khía cạnh này được xem xét kỹ lưỡng để tạo ra một sản phẩm không chỉ mạnh mẽ về mặt kỹ thuật mà còn thân thiện với người dùng và an toàn trong quá trình triển khai.

a, Hiệu năng (Performance)

Công cụ được tối ưu hóa để đạt được tốc độ quét cao, đặc biệt quan trọng khi phải xử lý một lượng lớn các mục tiêu hoặc áp dụng đồng thời nhiều loại payload khác nhau trong một khoảng thời gian ngắn. Để đạt được điều này, nó sử dụng cơ chế đa luồng thông qua `ThreadPoolExecutor` để thực hiện quét đồng thời trên nhiều URL hoặc tham số, với số lượng luồng tối đa có thể được cấu hình linh hoạt (mặc định là 10 hoặc dựa trên số lượng lõi CPU của hệ thống). Ngoài ra, công cụ còn áp dụng các biện pháp tối ưu hóa khác như giảm thiểu số lần thử lại yêu cầu HTTP (mặc định là 1 lần) và tự động bỏ qua việc sử dụng proxy khi quét các mục tiêu cục bộ trên `localhost`, giúp tiết kiệm thời gian và tài nguyên hệ thống trong các kịch bản kiểm thử nội bộ.

b, Khả năng mở rộng (Scalability)

Một trong những ưu điểm nổi bật của công cụ là khả năng mở rộng, cho phép người dùng dễ dàng bổ sung các payload mới chỉ bằng cách cập nhật tệp `data/payloads.txt` mà không cần phải chỉnh sửa trực tiếp mã nguồn, mang lại sự tiện lợi và linh hoạt cao. Tương tự, công cụ hỗ trợ việc mở rộng định nghĩa các

API mới thông qua tệp `api_definitions` hoặc trực tiếp trong cấu hình `config.yaml`, cũng như cho phép mở rộng danh sách từ khóa sử dụng trong tính năng khám phá (`data/common_paths.txt`) để tăng cường khả năng tìm kiếm các endpoint hoặc tệp tin tiềm năng trên máy chủ mục tiêu. Điều này giúp công cụ thích nghi tốt với các hệ thống web phức tạp và đa dạng trong thực tế.

c, Tính linh hoạt (Flexibility)

Công cụ cung cấp một mức độ linh hoạt cao thông qua việc hỗ trợ cấu hình toàn diện trong tệp `config.yaml`, cho phép người dùng điều chỉnh các tham số liên quan đến giao thức HTTP như thời gian timeout, User-Agent để mô phỏng các thiết bị khác nhau, hoặc cấu hình proxy để bảo vệ danh tính, cũng như lựa chọn các phương pháp quét phù hợp (error-based, boolean-based, time-based) tùy theo nhu cầu cụ thể. Ngoài ra, nó hỗ trợ quét đa dạng các loại mục tiêu, từ các tham số trong URL, các trường nhập liệu trong form HTML, cho đến các endpoint API (RESTful, GraphQL) với các định dạng body như JSON, XML, hoặc form data. Người dùng cũng có thể tùy chỉnh các tiêu chí xác nhận đăng nhập (`success_criteria`) và danh sách các mã trạng thái HTTP quan tâm (`interesting_status_codes`) trong quá trình khám phá, giúp công cụ thích nghi với nhiều môi trường kiểm thử khác nhau.

d, Tính bảo mật (Security)

Mặc dù được thiết kế như một công cụ kiểm thử tấn công, công cụ này vẫn tuân thủ các nguyên tắc bảo mật nghiêm ngặt để tránh gây ra những rủi ro không mong muốn cho hệ thống của người dùng. Cụ thể, nó không lưu trữ các dữ liệu nhạy cảm như cookie hoặc thông tin đăng nhập ngoài phạm vi phiên hoạt động, đảm bảo rằng thông tin cá nhân không bị lộ ra ngoài. Ngoài ra, công cụ thực hiện xử lý lỗi một cách an toàn, chẳng hạn như tự động bỏ qua proxy khi quét `localhost` để tránh các cảnh báo không cần thiết, hoặc cung cấp tùy chọn kiểm tra chứng chỉ SSL để phù hợp với các môi trường kiểm thử có chứng chỉ tự ký. Hơn nữa, dữ liệu đầu vào từ người dùng (URL, payload) được xử lý cẩn thận để ngăn chặn các lỗi cú pháp hoặc nguy cơ injection trong chính công cụ, tăng cường độ tin cậy và an toàn.

e, Tính dễ sử dụng (Usability)

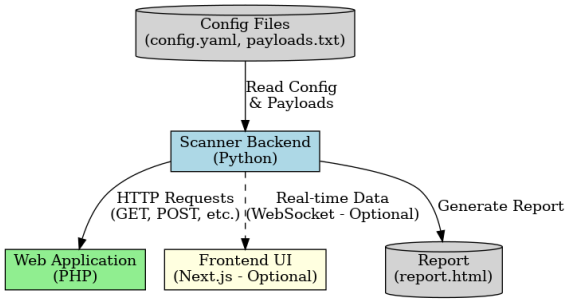
Để hỗ trợ người dùng một cách hiệu quả, công cụ được trang bị giao diện dòng lệnh (CLI) rõ ràng và dễ hiểu, với các tham số như `-url` để chỉ định mục tiêu, `-config` để tải cấu hình, `-report` để tạo báo cáo, `-loglevel` để điều chỉnh mức độ log, và `-logfile` để lưu nhật ký hoạt động. Báo cáo HTML được thiết kế với cấu trúc logic, bao gồm phần tổng quan thống kê và chi tiết từng lỗ hổng, giúp người dùng dễ dàng phân tích kết quả. Hệ thống logging được tích hợp với các

cấp độ khác nhau (DEBUG, INFO, WARNING, ERROR, CRITICAL), cho phép người dùng theo dõi quá trình quét một cách chi tiết và nhanh chóng phát hiện các vấn đề tiềm ẩn để thực hiện gỡ lỗi hoặc tối ưu hóa.

3.2 Kiến trúc Hệ thống

3.2.1 Tổng quan kiến trúc (High-level Architecture)

Kiến trúc tổng quan của công cụ quét lỗ hổng SQL Injection được xây dựng theo mô hình phân tầng, tập trung vào việc đảm bảo tính mô-đun và khả năng mở rộng để dễ dàng nâng cấp trong tương lai. Hệ thống được chia thành ba thành phần chính tương tác chặt chẽ với nhau: *Scanner Backend* được viết bằng ngôn ngữ Python đóng vai trò là bộ phận xử lý lõi, chịu trách nhiệm thực hiện các tác vụ quét và phân tích; *Web Application* được triển khai bằng PHP đại diện cho các mục tiêu kiểm thử, thường là các ứng dụng web được triển khai trên máy cục bộ hoặc máy chủ từ xa; và *Frontend UI* được phát triển bằng Next.js, đóng vai trò là một giao diện người dùng tùy chọn nhằm cung cấp khả năng hiển thị kết quả quét theo thời gian thực. Sự tương tác giữa các thành phần này được thực hiện thông qua các cơ chế truyền thông chuẩn như các yêu cầu HTTP, giao thức WebSocket (đối với Frontend UI tùy chọn), và việc đọc các tệp cấu hình chuyên biệt để định hình hành vi quét.



Hình 3.1: Kiến trúc hệ thống

a, Mô tả mối quan hệ giữa các thành phần

Scanner Backend và Web Application: Thành phần Scanner Backend khởi tạo và gửi các yêu cầu HTTP mang theo các payload SQL Injection một cách có hệ thống đến Web Application, thường là một ứng dụng PHP được triển khai trên các nền tảng khác nhau. Web Application sẽ xử lý các yêu cầu này và phản hồi lại bằng nội dung HTML, các thông báo lỗi, hoặc các phản hồi khác, mà Scanner Backend sau đó sẽ phân tích kỹ lưỡng để phát hiện các dấu hiệu rõ ràng của lỗ hổng SQL Injection.

Scanner Backend và Frontend UI: Trong trường hợp hệ thống được triển khai với giao diện người dùng, Scanner Backend có khả năng truyền tải dữ liệu quét

theo thời gian thực, chẳng hạn như tiến độ quét hiện tại, số lượng lỗ hổng đã được phát hiện, hoặc các thông tin chi tiết khác, đến Frontend UI thông qua giao thức WebSocket. Đây là một tính năng tùy chọn và chưa được tích hợp hoàn toàn trong phiên bản mã nguồn hiện tại, nhưng đã được xem xét kỹ lưỡng để phục vụ cho khả năng mở rộng trong các phiên bản phát triển sau này.

Config Files và Scanner Backend: Các tệp cấu hình đóng vai trò quan trọng trong việc định nghĩa và điều chỉnh hành vi của Scanner Backend một cách linh hoạt. Cụ thể, tệp `config.yaml` chứa các thông tin cấu hình chi tiết về giao thức HTTP, các tham số quét, và thông tin đăng nhập nếu có; tệp `payloads.txt` cung cấp danh sách các chuỗi payload tấn công được sử dụng trong quá trình kiểm thử; và tệp `api_endpoints.yaml` định nghĩa cấu trúc của các API cần được kiểm tra. Scanner Backend sẽ đọc và xử lý các tệp này một cách hiệu quả để khởi tạo và điều phối toàn bộ quá trình quét một cách chính xác.

Scanner Backend và Report: Sau khi quá trình quét hoàn tất, Scanner Backend sẽ tổng hợp tất cả các kết quả đã được phát hiện một cách cẩn thận và tạo ra một báo cáo chi tiết dưới định dạng HTML, được lưu trữ trong tệp `report.html`. Báo cáo này được xây dựng dựa trên danh sách các lỗ hổng đã được ghi nhận, sử dụng một template được định nghĩa bằng công cụ Jinja2 để đảm bảo định dạng chuyên nghiệp, dễ đọc, và có thể chia sẻ với các bên liên quan một cách thuận tiện.

3.3 Thành phần chính của Scanner Backend (Python)

Scanner Backend được xây dựng với một cấu trúc mô-đun rõ ràng, bao gồm nhiều thành phần chính được triển khai bằng ngôn ngữ Python để đảm bảo tính linh hoạt và dễ bảo trì. Các thành phần này được thiết kế để tương tác chặt chẽ với nhau, phối hợp nhịp nhàng nhằm thực hiện các chức năng quét lỗ hổng SQL Injection một cách hiệu quả và đáng tin cậy trong nhiều kịch bản khác nhau.

3.3.1 HTTPClient

a, Mô tả

Thành phần `HTTPClient` chịu trách nhiệm chính trong việc quản lý và thực hiện tất cả các yêu cầu HTTP được gửi đến Web Application, đóng vai trò như một cầu nối quan trọng giữa công cụ và các mục tiêu kiểm thử. Nó tận dụng thư viện `requests.Session` để duy trì trạng thái phiên một cách ổn định, đặc biệt quan trọng cho việc quản lý cookie và các thông tin xác thực trong các quy trình quét phức tạp, đồng thời cung cấp khả năng cấu hình linh hoạt để đáp ứng các yêu cầu đa dạng từ người dùng.

```
class HTTPClient:
    def __init__(self, config=None):
        self.session = requests.Session()
        self.config = {'timeout': 15, 'verify_ssl': True,
        ↪  'user_agent': 'SQLScanner/2.0', 'proxies': {},
        ↪  'retries': 1}
        if config: self.config.update({k: v for k, v in
        ↪  config.items() if k != 'proxies' or (v and not
        ↪  v.startswith('${') and v != '')})
        self.session.headers.update({'User-Agent':
        ↪  self.config['user_agent'], 'Accept-Encoding':
        ↪  'gzip, deflate'})
```

Hình 3.2: Trích đoạn mã nguồn lớp `HTTPClient`.

b, Chi tiết hoạt động

`HTTPClient` khởi tạo và duy trì một phiên HTTP duy nhất, cho phép lưu trữ và tự động gửi cookie trong các yêu cầu tiếp theo một cách liền mạch, điều này cực kỳ quan trọng cho các quy trình quét đòi hỏi xác thực như đăng nhập tự động hoặc truy cập các khu vực bảo vệ. Để đảm bảo tính ổn định và khả năng phục hồi sau các lỗi mạng tạm thời, nó cho phép cấu hình thời gian chờ cho mỗi yêu cầu (mặc định là 15 giây) và số lần thử lại yêu cầu (mặc định là 1 lần) để xử lý các tình huống mất kết nối bất ngờ. Người dùng cũng có thể bật hoặc tắt tính năng xác minh chứng chỉ SSL, một tính năng hữu ích trong các môi trường kiểm thử nơi các máy chủ có thể sử dụng chứng chỉ tự ký hoặc không hợp lệ. `HTTPClient` thiết lập một User-Agent mặc định là `SQLScanner/2.0` để nhận diện công cụ và hỗ trợ việc bổ sung các HTTP header tùy chỉnh cho từng yêu cầu hoặc định nghĩa API cụ thể. Cuối cùng, công cụ có khả năng cấu hình và sử dụng proxy cho các yêu cầu HTTP để tăng cường bảo mật hoặc cải thiện hiệu suất, nhưng tự động bỏ qua proxy khi quét các mục tiêu chạy trên `localhost` để tối ưu hóa tốc độ và tránh các vấn đề kết nối cục bộ. Mã nguồn minh họa cấu trúc khởi tạo của lớp `HTTPClient` được trình bày trong hình 3.2.

3.3.2 HTMLParser

a, Mô tả

Thành phần `HTMLParser` đóng vai trò quan trọng trong việc phân tích cú pháp mã nguồn HTML từ các phản hồi HTTP, cung cấp nền tảng để trích xuất các thông tin cấu trúc cần thiết như các biểu mẫu (forms) và liên kết (links), đồng thời tích hợp khả năng phát hiện các mẫu lỗi SQL một cách tự động, giúp tăng cường hiệu quả trong quá trình kiểm thử.

```
class AdvancedHTMLParser:
    SQL_ERROR_PATTERNS = {'mysql': r"SQL syntax.*MySQL",
     ↪  'postgresql': r"PostgreSQL.*ERROR", 'oracle':
     ↪  r"ORA-\d{5}", 'mssql': r"Microsoft SQL Server"}
    def __init__(self, html_content, base_url): self.soup
     ↪  = BeautifulSoup(html_content, 'lxml');
     ↪  self.base_url = base_url
    def extract_forms(self): return [self._parse_form(f)
     ↪  for f in self.soup.find_all('form')]
    def _parse_form(self, form): return {'action':
     ↪  form.get('action'), 'method': form.get('method',
     ↪  'get').upper(), 'inputs':
     ↪  self._get_form_inputs(form)}
    def _get_form_inputs(self, form): return [{'type':
     ↪  t.get('type', 'text'), 'name': t.get('name'),
     ↪  'value': t.get('value', '')} for t in
     ↪  form.find_all(['input', 'textarea', 'select']) if
     ↪  t.get('name')]
```

Hình 3.3: Trích đoạn mã nguồn lớp `AdvancedHTMLParser`.

b, Chi tiết hoạt động

`HTMLParser` quét toàn bộ mã HTML để tìm kiếm các thẻ `<form>` một cách chi tiết. Với mỗi form được tìm thấy, nó trích xuất các thuộc tính quan trọng như `action` (được chuẩn hóa thành URL tuyệt đối bằng cách sử dụng hàm `urljoin`), `method` để xác định phương thức gửi dữ liệu (GET hoặc POST), và một danh sách các trường nhập liệu (`<input>`, `<textarea>`, `<select>`) bao gồm các thông tin như `type`, `name`, và `value` mặc định nếu được khai báo. Ngoài ra, thành phần này còn có khả năng trích xuất các liên kết từ các thẻ `<a>` với thuộc tính `href`, đồng thời lọc bỏ các liên kết không hợp lệ như `mailto:` hoặc `javascript:` để tránh lãng phí tài nguyên. Công cụ cũng hỗ trợ tùy chọn giới hạn việc trích xuất các liên kết trong cùng một tên miền (*same-domain*) để tập trung phạm vi quét và tăng hiệu suất. Một tính năng cốt lõi của `HTMLParser` là khả năng nhận diện các thông báo lỗi từ cơ sở dữ liệu bằng cách sử dụng một tập hợp các mẫu biểu thức chính quy (`SQL_ERROR_PATTERNS`), cho phép so khớp với nội dung phản hồi để phát hiện các dấu hiệu của lỗ hổng SQL Injection và xác định loại cơ sở dữ liệu (MySQL, PostgreSQL, Oracle, MSSQL). Đoạn mã minh họa định nghĩa các mẫu lỗi SQL và cấu trúc trích xuất form được trình bày trong hình 3.3.

3.3.3 SQLInjector

a, Mô tả

Lớp `AdvancedSQLInjector` là thành phần trung tâm của Scanner Backend, đóng vai trò như một trung tâm điều phối toàn bộ quy trình quét lỗ hổng SQL

```
class AdvancedSQLInjector:
    def __init__(self, config_file='config.yaml'):
        self.config = load_config(config_file) or
        ↪  sys.exit(1)
        self.http_client =
        ↪  HTTPClient(self.config.get('http', {}))
        self.parser = AdvancedHTMLParser("", "")
        self.payloads =
        ↪  self.load_payloads(self.config.get('scanner',
        ↪  {}).get('payload_file')) or sys.exit(1)
        self.vulnerabilities, self.vuln_set = [], set()
        self.vulnerability_lock = threading.Lock()
```

Hình 3.4: Trích đoạn mã nguồn lớp `AdvancedSQLInjector`.

Injection, từ việc khởi tạo các thành phần phụ trợ, quản lý danh sách payload, đến việc phân tích phản hồi và tạo báo cáo tổng hợp một cách có tổ chức.

b, Chi tiết hoạt động

Khi được khởi tạo, `AdvancedSQLInjector` sẽ đọc và tải cấu hình chi tiết từ tệp `config.yaml` để định hình hành vi quét. Dựa trên các thông số này, nó khởi tạo các đối tượng `HTTPClient` để xử lý yêu cầu HTTP và `HTMLParser` để phân tích mã HTML, đồng thời tải danh sách các payload tấn công từ tệp `payloads.txt` để chuẩn bị cho quá trình chèn. Công cụ điều phối các hàm con chuyên biệt để thực hiện quét trên nhiều loại mục tiêu khác nhau, bao gồm `_scan_url_parameters` để kiểm thử các tham số trong URL, `_scan_html_forms` để chèn payload vào các trường nhập liệu của form HTML, và `_scan_api_endpoint` để quét các điểm cuối API dựa trên định nghĩa từ người dùng. Nếu tính năng khám phá được kích hoạt, công cụ sẽ tìm kiếm các đường dẫn và tệp tiềm năng bằng danh sách từ khóa (*wordlist*), thêm URL mới vào hàng đợi để tiếp tục quét. Đối với các ứng dụng yêu cầu xác thực, `AdvancedSQLInjector` hỗ trợ đăng nhập tự động nếu `login.enabled` được kích hoạt, sử dụng cookie phiên cho các yêu cầu tiếp theo. Sau mỗi yêu cầu, hàm `_analyze_inband_response` phân tích phản hồi để xác định lỗ hổng qua các kỹ thuật Error-based, Boolean-based, và Time-based. Mã nguồn minh họa cấu trúc khởi tạo của lớp được trình bày trong hình 3.4.

3.3.4 Vulnerability Model

a, Mô tả

Lớp `Vulnerability` là một mô hình dữ liệu chuẩn hóa, được thiết kế để biểu diễn một cách chi tiết và nhất quán các lỗ hổng SQL Injection được phát hiện trong quá trình quét, cung cấp nền tảng để lưu trữ và quản lý thông tin một cách hiệu quả.

```
class Vulnerability:
    SEVERITY_LEVELS = ['low', 'medium', 'high',
    ↪ 'critical']
    def __init__(self, name, description, severity,
    ↪ payload, input_field, url):
        self.name, self.description, self.severity,
        ↪ self.payload, self.input_field, self.url =
        ↪ name, description, severity.lower(), payload,
        ↪ input_field, url
        self.timestamp = datetime.datetime.now()
    def to_dict(self): return {'name': self.name,
    ↪ 'severity': self.severity, 'payload':
    ↪ self.payload, 'url': self.url}
```

Hình 3.5: Trích đoạn mã nguồn lớp `Vulnerability`.

`Vulnerability` lưu trữ các thuộc tính quan trọng như tên của lỗ hổng, mô tả chi tiết về cách thức phát hiện và tác động, mức độ nghiêm trọng được phân loại thành `low`, `medium`, `high`, hoặc `critical`, payload đã được sử dụng để kích hoạt lỗ hổng, tên của trường hoặc tham số bị tấn công, URL nơi lỗ hổng được tìm thấy, và thời điểm chính xác của việc phát hiện. Ngoài ra, lớp này cung cấp một phương thức `to_dict()` để chuyển đổi đối tượng lỗ hổng thành định dạng dictionary, hỗ trợ tích hợp dữ liệu vào các hệ thống báo cáo như báo cáo HTML một cách dễ dàng và nhanh chóng. Mã nguồn định nghĩa cấu trúc của lớp được trình bày trong hình 3.5.

3.3.5 ReportGenerator

a, Mô tả

Thành phần `ReportGenerator` chịu trách nhiệm chính trong việc tạo ra báo cáo kết quả quét dưới định dạng HTML, đóng vai trò như một công cụ quan trọng để tổng hợp và trình bày thông tin một cách chuyên nghiệp cho người dùng.

b, Chi tiết hoạt động

`ReportGenerator` thực hiện các bước chi tiết sau: đầu tiên, nó đọc template báo cáo từ tệp `templates/report.html` để đảm bảo định dạng nhất quán; tiếp theo, nó tính toán các số liệu thống kê quan trọng như tổng số lỗ hổng được phát hiện và sự phân bố của chúng theo từng mức độ nghiêm trọng (low, medium, high, critical); cuối cùng, nó ghi nội dung HTML đã được kết xuất vào một tệp đầu ra (mặc định là `report.html`), cung cấp một bản ghi đầy đủ, dễ đọc, và có thể chia sẻ với các bên liên quan để hỗ trợ quá trình ra quyết định bảo mật. Mã nguồn minh họa logic của lớp được trình bày trong hình 3.6.

```
class ReportGenerator:
    def __init__(self): self.env =
    ↪ Environment(loader=FileSystemLoader('templates'))
    def generate(self, vulnerabilities,
    ↪ output_path='report.html'):
        stats = {'total': len(vulnerabilities),
        ↪ 'severity_distribution': {v.severity:
        ↪ s.get(v.severity, 0) + 1 for v, s in [(v,
        ↪ stats['severity_distribution']) for v in
        ↪ vulnerabilities]}}
        with open(output_path, 'w') as f:
        ↪ f.write(self.env.get_template('report.html').
        ↪ render(meta={'generated_at':
        ↪ datetime.now().isoformat()}, stats=stats,
        ↪ findings=[v.to_dict() for v in
        ↪ vulnerabilities]))
```

Hình 3.6: Trích đoạn mã nguồn lớp `ReportGenerator`.

3.3.6 Các Module tiện ích

a, Mô tả

Tệp `utils/helpers.py` tập hợp một số hàm tiện ích hỗ trợ các thành phần khác trong hệ thống, được thiết kế để thực hiện các tác vụ chung như xử lý an toàn dữ liệu đầu vào và phân tích chuỗi truy vấn, góp phần tăng tính tái sử dụng và duy trì mã nguồn trong dài hạn.

b, Chi tiết hoạt động

Các hàm chính trong module này bao gồm: `sanitize_input`, một hàm quan trọng giúp loại bỏ hoặc thoát các ký tự có thể gây nguy hiểm như dấu nháy đơn hoặc dấu nháy kép để ngăn chặn lỗi cú pháp hoặc các cuộc tấn công injection trong chính công cụ; `validate_input`, được sử dụng để kiểm tra tính hợp lệ của dữ liệu đầu vào bằng cách áp dụng các tiêu chí như độ dài tối thiểu hoặc các ký tự cho phép; và `parse_query_parameters`, một hàm hỗ trợ phân tích chuỗi truy vấn URL thành định dạng dictionary để xử lý các tham số một cách hiệu quả. Mã nguồn trình bày các hàm tiện ích cơ bản trong `helpers.py` được minh họa trong hình 3.7.

3.4 Cơ chế đa luồng

Để tối ưu hóa hiệu suất và giảm thiểu thời gian quét khi xử lý một lượng lớn các mục tiêu trong các ứng dụng web phức tạp, công cụ đã tích hợp một cơ chế đa luồng mạnh mẽ và hiệu quả. Cơ chế này được triển khai dựa trên `ThreadPoolExecutor` từ thư viện `concurrent.futures`, kết hợp với kỹ thuật đồng bộ hóa `threading.Lock` để đảm bảo tính toàn vẹn dữ liệu trong môi trường đa luồng.

```
def sanitize_input(input_string): return input_string.replace("'", "\\'").replace('"', '\\"')
def validate_input(input_string): return len(input_string) > 0
def parse_query_parameters(query_string): return parse_qs(query_string)
```

Hình 3.7: Trích đoạn mã nguồn các hàm tiện ích trong `helpers.py`.

3.4.1 Sử dụng ThreadPoolExecutor

a, Mô tả

`ThreadPoolExecutor` là thành phần cốt lõi trong việc quản lý và phân phối các tác vụ quét một cách đồng thời, thiết lập một nhóm các luồng (*worker threads*) làm việc song song để thực thi các hàm quét mục tiêu (`scan_target`) trên nhiều điểm kiểm thử khác nhau như URL, form HTML, và các endpoint API.

b, Chi tiết hoạt động

Số lượng luồng tối đa được định nghĩa thông qua tham số `max_threads` trong tệp `config.yaml`, với giá trị mặc định là 10 luồng để cân bằng giữa hiệu suất và tài nguyên hệ thống, hoặc có thể được điều chỉnh động dựa trên số lượng lõi CPU của máy tính cộng thêm 4 để tận dụng tối đa sức mạnh phần cứng. Các mục tiêu quét sẽ được thêm vào một hàng đợi (`tasks_to_submit_queue`) và được phân phối cho các luồng theo nguyên tắc FIFO (First-In-First-Out), đảm bảo rằng các tác vụ được xử lý theo thứ tự ưu tiên. Mỗi luồng chịu trách nhiệm xử lý một tác vụ quét hoàn chỉnh, bao gồm việc kiểm tra các định nghĩa API, phân tích tham số URL, và quét các form HTML, sau đó có thể trả về các liên kết mới được khám phá. Những liên kết này sẽ được thêm trở lại vào hàng đợi để tiếp tục quá trình quét, tạo ra một vòng lặp kiểm tra không ngừng nghỉ. Mã nguồn minh họa vòng lặp chính trong hàm `main()` của `scanner.py` được trình bày trong hình 3.8.

```
with concurrent.futures.ThreadPoolExecutor(max_workers=max_workers) as executor:
    while tasks_to_submit_queue:
        future = executor.submit(scanner.scan_target, tasks_to_submit_queue.pop())
        newly_discovered_links = future.result()
        if newly_discovered_links: tasks_to_submit_queue.extend(newly_discovered_links)
```

Hình 3.8: Trích đoạn mã nguồn quản lý đa luồng với `ThreadPoolExecutor`.

3.4.2 Đảm bảo an toàn luồng

a, Mô tả

Để ngăn chặn các vấn đề về tranh chấp dữ liệu (*race conditions*) và đảm bảo tính toàn vẹn của các cấu trúc dữ liệu chia sẻ trong môi trường đa luồng, công cụ sử dụng cơ chế đồng bộ hóa `threading.Lock` một cách hiệu quả, đặc biệt trong việc bảo vệ quyền truy cập và cập nhật danh sách các lỗ hổng.

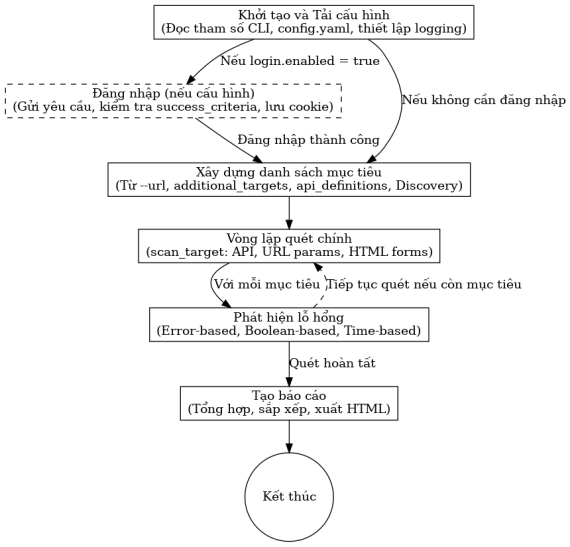
Khi một luồng phát hiện ra một lỗ hổng tiềm năng trong quá trình phân tích phản hồi thông qua hàm `_analyze_inband_response`, trước khi thêm lỗ hổng đó vào danh sách `self.vulnerabilities` và tập hợp kiểm tra trùng lặp `self.vuln_set`, luồng sẽ yêu cầu khóa `vulnerability_lock` để giành quyền độc quyền. Điều này đảm bảo rằng chỉ có một luồng duy nhất được phép truy cập và sửa đổi các cấu trúc dữ liệu chung tại một thời điểm, ngăn ngừa tình trạng ghi đè hoặc mất mát dữ liệu trong quá trình xử lý đồng thời. Tập hợp `self.vuln_set` được sử dụng để lưu trữ các khóa duy nhất của lỗ hổng, bao gồm thông tin về loại lỗi, loại cơ sở dữ liệu, payload, tham số, và URL, giúp tránh việc ghi nhận trùng lặp các lỗ hổng đã được phát hiện trước đó. Sau khi hoàn tất cập nhật, khóa sẽ được giải phóng để các luồng khác có thể tiếp tục công việc của mình. Đoạn mã minh họa cách sử dụng `threading.Lock` được trình bày trong hình 3.9.

```
1 def _analyze_inband_response(self, response, payload,
   ↪  input_field_name, start_time, injection_point_url):
2     if is_error_based and error_key not in self.vuln_set:
3         vuln = Vulnerability(f"SQL Injection
           ↪  ({db_type.upper()}) - Error Based", f"Detected
           ↪  {db_type} SQL injection", 'high', payload,
           ↪  input_field_name, injection_point_url)
4         with self.vulnerability_lock:
           ↪  self.vulnerabilities.append(vuln);
           ↪  self.vuln_set.add(error_key)
```

Hình 3.9: Trích đoạn mã nguồn hàm `_analyze_inband_response`.

3.5 Luồng Hoạt động Chính

Công cụ quét lỗ hổng SQL Injection được thiết kế với một quy trình vận hành rõ ràng, tích hợp các giai đoạn từ khởi tạo hệ thống, xử lý đăng nhập tự động (nếu được kích hoạt), xây dựng danh sách mục tiêu, thực hiện quét, phát hiện lỗ hổng, đến tạo báo cáo chi tiết. Quy trình này được minh họa trực quan thông qua sơ đồ hoạt động trong Hình 3.10, được xây dựng bằng thư viện `graphviz` từ mã Python, giúp người đọc dễ dàng nắm bắt luồng xử lý tổng thể của công cụ.



Hình 3.10: Sơ đồ hoạt động thể hiện luồng vận hành chính của công cụ quét SQL Injection.

3.5.1 Khởi tạo và Tải Cấu hình

Giai đoạn khởi tạo đặt nền tảng cho hoạt động của công cụ thông qua việc thiết lập các thành phần cốt lõi. Đầu tiên, công cụ sử dụng thư viện `argparse` để phân tích các tham số dòng lệnh như `-url` (URL mục tiêu), `-config` (đường dẫn tệp cấu hình), và `-loglevel` (mức độ ghi log). Tiếp theo, hàm `load_config` đọc tệp `config.yaml`, thay thế các biến môi trường và áp dụng cấu hình cho HTTP client, scanner, và tính năng khám phá. Hệ thống logging được thiết lập linh hoạt, cho phép ghi thông tin ra console hoặc tệp tùy theo cấu hình người dùng. Cuối cùng, các đối tượng chính như `HTTPClient`, `AdvancedSQLInjector` (tải danh sách payload từ `payloads.txt`), và `ReportGenerator` được khởi tạo để sẵn sàng cho các bước tiếp theo. Đoạn mã dưới đây minh họa quá trình khởi tạo chính:

```
def main():
    parser = argparse.ArgumentParser(description='SQL
    ↪  Injection Scanner')
    parser.add_argument('--url', required=True)
    parser.add_argument('--config', default='config.yaml')
    args = parser.parse_args()
    scanner = AdvancedSQLInjector(config_file=args.config)
```

3.5.2 Quy trình Đăng nhập

Khi tính năng đăng nhập tự động được kích hoạt trong `config.yaml` (với `login.enabled: true`), công cụ thực hiện quy trình đăng nhập một cách hiệu quả. Công cụ sử dụng `HTTPClient` để gửi yêu cầu GET hoặc POST tới

`login.url` với dữ liệu được chỉ định. Phản hồi từ máy chủ được đánh giá dựa trên các tiêu chí xác thực, bao gồm mã trạng thái HTTP (thường là 200 hoặc 302), sự tồn tại của cookie, hoặc nội dung phản hồi chứa chuỗi cụ thể. Nếu đăng nhập thành công, các cookie nhận được sẽ được lưu vào `self.session` của `HTTPClient` để sử dụng trong các yêu cầu quét tiếp theo. Đoạn mã sau thể hiện quy trình này:

```
def _perform_login(self):
    login_config = self.config.get('login', {})
    if login_config.get('enabled', False):
        response = self.http_client.send_advanced_request(
            login_config.get('url'),
            method=login_config.get('method', 'POST'),
            data=login_config.get('data', {})
        )
        if response.status_code in
         ↪  login_config.get('success_criteria',
         ↪  {}).get('status_codes', [200, 302]):
            self.session.cookies.update(response.cookies)
```

3.5.3 Xây dựng Danh sách Mục tiêu

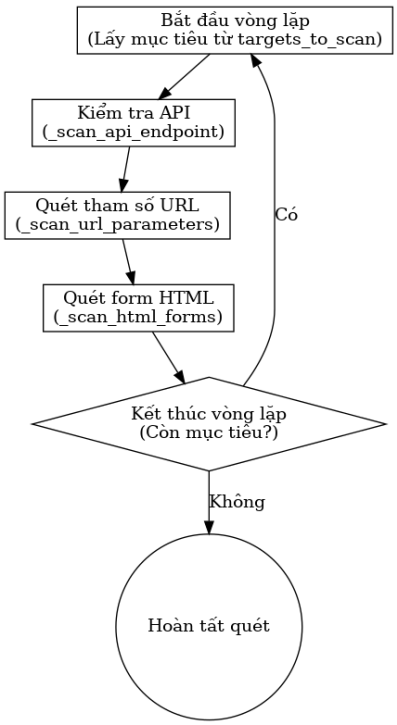
Việc tổng hợp danh sách mục tiêu để quét được thực hiện bằng cách kết hợp nhiều nguồn dữ liệu. URL cơ sở được chuẩn hóa từ tham số `-url` cung cấp qua dòng lệnh. Các mục tiêu bổ sung được trích xuất từ `config.yaml` hoặc tệp `additional_targets_file`, bao gồm các URL và phương thức HTTP tương ứng. Ngoài ra, các endpoint API được định nghĩa trong `api_endpoints.yaml` cũng được tích hợp. Nếu tính năng khám phá được bật (`discovery.enabled: true`), hàm `discover_targets` sẽ quét các đường dẫn tiềm năng từ `common_paths.txt`, gửi yêu cầu HEAD hoặc GET, và thêm các URL trả về mã trạng thái 200 hoặc 403 vào danh sách mục tiêu. Đoạn mã sau minh họa quá trình khám phá mục tiêu:

```
def discover_targets(self, base_url):
    discovered = set()
    if self.discovery_config.get('enabled', False):
        with open(self.discovery_config.get('wordlist_fil
         ↪  e'), 'r') as
         ↪  f:
            for path in [line.strip() for line in f if
             ↪  line.strip()]:
                target_url = urljoin(base_url, path)
```

```
                    response = self.http_client.send_advanced_request(target_url,
                    ↪  method='HEAD')
                    if response.status_code in [200, 403]:
                        discovered.add(target_url)
    return list(discovered)
```

3.5.4 Vòng lặp Quét Chính

Quá trình quét được thực hiện trong một vòng lặp đa luồng, tối ưu hóa hiệu suất bằng cách xử lý song song các mục tiêu trong `targets_to_scan`. Hàm `scan_target` điều phối việc quét, gọi các hàm con để kiểm tra các khía cạnh khác nhau của mục tiêu. Cụ thể, `_scan_api_endpoint` chèn payload vào các tham số API, `_scan_url_parameters` kiểm tra các tham số trong chuỗi truy vấn, và `_scan_html_forms` phân tích và kiểm tra các biểu mẫu HTML. Luồng quét được thiết kế có thứ tự, ưu tiên kiểm tra API, sau đó đến tham số URL, và cuối cùng là biểu mẫu HTML. Sơ đồ hoạt động trong Hình 3.11 minh họa rõ ràng quy trình này.



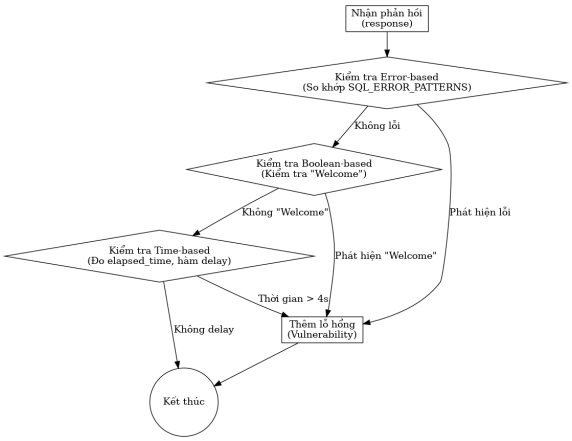
Hình 3.11: Sơ đồ hoạt động của vòng lặp quét chính, thể hiện luồng kiểm tra từng mục tiêu.

```
def _scan_simple_url_target(self, target_url):
    for api_def in self.api_definitions:
```

```
3           if urlparse(urljoin(self.base_scan_url,
             ↪  api_def.get('path'))).path ==
             ↪  urlparse(target_url).path:
4               self._scan_api_endpoint(target_url, api_def)
5       self._scan_url_parameters(target_url)
6       self._fetch_and_scan_forms_on_url(target_url)
```

3.5.5 Phát hiện Lỗ hổng

Việc phát hiện lỗ hổng được thực hiện thông qua hàm `_analyze_inband_response`, phân tích phản hồi HTTP để xác định các loại lỗ hổng SQL Injection. Đối với lỗ hổng error-based, công cụ so khớp nội dung phản hồi với các mẫu lỗi trong `SQL_ERROR_PATTERNS`. Lỗ hổng boolean-based hiện kiểm tra sự xuất hiện của các chuỗi như "Welcome", nhưng cần cải tiến bằng phân tích vi phân để tăng độ chính xác. Lỗ hổng time-based được phát hiện bằng cách đo thời gian phản hồi (`elapsed_time`) so với ngưỡng 4 giây cho các payload chứa hàm trễ như `SLEEP`. Các lỗ hổng được lưu trữ trong `self.vulnerabilities`, với cơ chế đồng bộ `threading.Lock` để đảm bảo an toàn trong môi trường đa luồng. Quy trình này được minh họa trong Hình 3.12.



Hình 3.12: Sơ đồ hoạt động mô tả quy trình phát hiện lỗ hổng SQL Injection.

```
1   def _analyze_inband_response(self, response, payload,
     ↪  input_field_name, start_time, injection_point_url):
2       if any(re.search(p, response.text) for p in
         ↪  AdvancedHTMLParser.SQL_ERROR_PATTERNS.values()):
3           with self.vulnerability_lock:
4               self.vulnerabilities.append(Vulnerability("SQL
                 ↪  Injection - Error Based", "...", 'high',
                 ↪  payload, input_field_name,
                 ↪  injection_point_url))
```

```
5       if "Welcome" in response.text:
6           with self.vulnerability_lock:
7               self.vulnerabilities.append(Vulnerability("SQL
                ↪  Injection - Boolean Based", "...", 'high',
                ↪  payload, input_field_name,
                ↪  injection_point_url))
8       if (time.time() - start_time) > 4 and 'SLEEP(' in
        ↪  payload.upper():
9           with self.vulnerability_lock:
10              self.vulnerabilities.append(Vulnerability("SQL
                ↪  Injection - Time Based", "...",
                ↪  'critical', payload, input_field_name,
                ↪  injection_point_url))
```

3.5.6 Tạo Báo cáo

Sau khi hoàn tất quá trình quét, công cụ tổng hợp và trình bày kết quả dưới dạng báo cáo HTML chi tiết. Hàm `generate_report` sắp xếp danh sách `self.vulnerabilities` theo URL và mức độ nghiêm trọng. Tiếp theo, `ReportGenerator` sử dụng template Jinja2 để tạo tệp `report.html`, cung cấp thông tin đầy đủ về các lỗ hổng phát hiện được cùng thống kê tổng quan. Đoạn mã sau minh họa quy trình tạo báo cáo:

```
1  def generate_report(self):
2      self.vulnerabilities.sort(key=lambda v: (v.url,
       ↪  v.input_field or '', v.vulnerability))
3      return self.vulnerabilities
4  report_generator = ReportGenerator()
5  report_generator.generate(scanner.generate_report(),
   ↪  'report.html')
```

3.6 Cấu hình

Cấu hình đóng vai trò then chốt trong việc tùy chỉnh và điều chỉnh hoạt động của công cụ quét lỗ hổng SQL Injection, mang lại sự linh hoạt tối đa cho người dùng trong việc kiểm soát các tham số quét, phương thức giao tiếp mạng, và danh sách các mục tiêu cần kiểm thử. Các thiết lập chính được định nghĩa trong tệp `config.yaml`, được hỗ trợ bởi các tệp chuyên biệt như `api_endpoints.yaml` (định nghĩa cấu trúc API), `payloads.txt` (chứa các chuỗi payload tấn công), và `common_paths.txt` (wordlist dùng cho tính năng khám phá). Phần dưới đây sẽ phân tích chi tiết cấu trúc và vai trò của từng thành phần cấu hình này.

3.6.1 Tổng quan về `config.yaml`

Tệp `config.yaml` là trung tâm của hệ thống cấu hình, được tổ chức thành ba phần chính: *HTTP*, *Scanner*, và *Login*.

a, Cấu hình HTTP

Phần *HTTP* tập trung vào các thiết lập liên quan đến giao thức mạng và tương tác HTTP. Các tham số chính bao gồm `timeout` (thời gian chờ tối đa cho mỗi yêu cầu, mặc định 20 giây), `verify_ssl` (tùy chọn xác minh chứng chỉ SSL, mặc định `true`), `user_agent` (chuỗi định danh trình duyệt gửi trong các yêu cầu, mặc định "Mozilla/5.0 (compatible; SQLScanner/2.0)"), và `proxies` (cấu hình proxy mạng, có thể lấy giá trị từ biến môi trường `${HTTP_PROXY}` và `${HTTPS_PROXY}`). Những tham số này đảm bảo công cụ có khả năng giao tiếp hiệu quả với các ứng dụng web mục tiêu, ngay cả trong các trường hợp server không sử dụng chứng chỉ SSL hợp lệ hoặc khi cần định tuyến qua proxy.

b, Cấu hình Scanner

Phần *Scanner* cung cấp các thiết lập cốt lõi cho quá trình quét lỗ hổng. Các tham số quan trọng bao gồm:

- `payload_file`: Đường dẫn tới tệp chứa danh sách các chuỗi payload SQL Injection (mặc định là `data/payloads.txt`).
- `db_detection`: Tùy chọn phát hiện loại cơ sở dữ liệu (mặc định "auto").
- `techniques`: Danh sách các kỹ thuật quét sẽ được áp dụng, bao gồm *error_based*, *blind* (boolean-based), và *time_based*.
- `api_definitions_file`: Đường dẫn tới tệp định nghĩa các endpoint API cần quét (mặc định `data/api_endpoints.yaml`).
- `additional_targets_file`: Đường dẫn tới tệp chứa danh sách các mục tiêu bổ sung (mặc định `data/additional_targets.yaml`).
- `max_threads`: Số lượng luồng tối đa được sử dụng cho quá trình quét đồng thời (mặc định 10).
- `time_delay_threshold`: Ngưỡng thời gian (giây) để xác định lỗ hổng time-based (mặc định 4 giây).
- `discovery`: Một cấu hình lồng ghép cho tính năng khám phá đường dẫn/tệp tiềm năng, bao gồm `enabled` (bật/tắt), `wordlist_file` (tệp wordlist, mặc định `data/common_paths.txt`), `extensions_to_append` (các phần mở rộng tệp cần thử), và `interesting_status_codes` (các mã trạng thái HTTP quan tâm như 200, 403).

Những thiết lập này cho phép người dùng tùy chỉnh phạm vi và phương pháp quét, bao gồm khả năng tự động khám phá các đường dẫn tiềm năng trên máy chủ mục tiêu.

c, Cấu hình Login

Phần *Login* hỗ trợ chức năng đăng nhập tự động vào ứng dụng web mục tiêu trước khi tiến hành quét. Các tham số cấu hình bao gồm:

- `enabled`: Bật hoặc tắt tính năng đăng nhập (mặc định `true`).
- `url`: Đường dẫn URL của trang đăng nhập (ví dụ: "http://localhost:8000/process.php").
- `method`: Phương thức HTTP được sử dụng để gửi yêu cầu đăng nhập (mặc định "POST").
- `data`: Dữ liệu đăng nhập (ví dụ: username và password) sẽ được gửi kèm trong yêu cầu.
- `success_criteria`: Các tiêu chí được sử dụng để đánh giá việc đăng nhập thành công. Điều này bao gồm `status_codes` (mã trạng thái HTTP dự kiến như [200, 302]), `cookies` (danh sách tên cookie cần kiểm tra sự hiện diện, ví dụ: ["PHPSESSID"]), và tùy chọn `redirect_url` (URL chuyển hướng dự kiến sau khi đăng nhập thành công).

Phần cấu hình này đảm bảo công cụ có thể vượt qua cơ chế xác thực của ứng dụng web, cho phép quét các khu vực yêu cầu quyền truy cập.

3.6.2 Định nghĩa API trong `api_endpoints.yaml`

Tệp `api_endpoints.yaml` (hoặc cấu hình inline trong `config.yaml`) cung cấp cấu trúc chi tiết để định nghĩa các endpoint API mà công cụ sẽ quét một cách chuyên biệt, hỗ trợ xử lý các yêu cầu HTTP phức tạp. Mỗi định nghĩa API bao gồm các trường: `path` (đường dẫn API, ví dụ: "/api/get_user.php"), `method` (phương thức HTTP như GET, POST, PUT, DELETE), `params_in` (vị trí tham số cần chèn payload: `query`, `body_json`, `body_form`, hoặc `body_xml`), và `params_to_test` (danh sách các tham số cụ thể cần kiểm tra, bao gồm cả các trường lồng nhau như "profile.description"). Ngoài ra, các định nghĩa có thể bao gồm `json_template` hoặc `body_template` để cung cấp mẫu dữ liệu gửi đi (ví dụ: cấu trúc JSON hoặc XML).

Chẳng hạn, một định nghĩa cho endpoint `/api/get_user.php` có thể chỉ định phương thức GET với tham số `id` và `user_id` trong query string. Trong khi đó, `/api/update_user` có thể sử dụng phương thức POST với dữ liệu JSON chứa các trường lồng nhau như "profile.description". Các trường hợp đặc biệt như

API GraphQL (`/graphql`) có thể được cấu hình với trường `variables`, và API XML-RPC (`/xmlrpc_api`) có thể bao gồm các `headers` tùy chỉnh. Việc định nghĩa API chuyên biệt này giúp tăng độ chính xác của quá trình quét, giảm thiểu false positives, và hỗ trợ hiệu quả các ứng dụng web hiện đại với kiến trúc API phức tạp như RESTful, GraphQL, và SOAP.

3.6.3 Payloads và Wordlists

a, File `data/payloads.txt`

Tệp `data/payloads.txt` chứa một danh sách toàn diện các chuỗi payload SQL Injection, được thiết kế để phục vụ các kỹ thuật quét khác nhau. Nội dung của tệp này tập trung chủ yếu vào các payload tương thích với MySQL, bao gồm các biến thể cho kỹ thuật *error-based* (ví dụ: `' OR '1''1'-`), *boolean-based* (ví dụ: `' AND 1̄0-`), và *time-based* (ví dụ: `' OR SLEEP(5)-`). Ngoài ra, tệp còn chứa các payload nâng cao cho các mục đích như fingerprinting (thu thập thông tin phiên bản cơ sở dữ liệu, tên host), schema enumeration (liệt kê danh sách bảng và cột), và data dumping (trích xuất dữ liệu từ các bảng quan trọng như `users`). Các payload được tổ chức để hỗ trợ kiểm thử số cột (sử dụng mệnh đề `ORDER BY`) và được sắp xếp để tránh trùng lặp khi tải vào công cụ.

b, File `data/common_paths.txt`

Tệp `data/common_paths.txt` đóng vai trò là wordlist cho tính năng khám phá đường dẫn/tệp tiềm năng của công cụ. Nó liệt kê các đường dẫn và tên tệp phổ biến thường được sử dụng trong các ứng dụng web, chẳng hạn như `admin`, `login`, `api`, cùng với các phần mở rộng tệp thông dụng như `.php` và `.html`. Danh sách này được sử dụng để tạo ra hàng loạt các URL tiềm năng bằng cách ghép các mục trong wordlist với URL cơ sở của mục tiêu. Mục đích là hỗ trợ công cụ phát hiện các endpoint ẩn, các tệp nhạy cảm, hoặc các tài nguyên chưa được liệt kê rõ ràng trong các cấu hình API hay form đã biết, từ đó mở rộng phạm vi quét một cách hiệu quả.

CHƯƠNG 4. TRIỂN KHAI VÀ ĐÁNH GIÁ KẾT QUẢ

4.1 Môi trường Triển khai

Để triển khai và vận hành công cụ quét lỗ hổng SQL Injection một cách hiệu quả, việc đáp ứng các yêu cầu về phần cứng, phần mềm và thiết lập môi trường kiểm thử là cần thiết. Ngoài ra, công cụ được tích hợp một giao diện người dùng trực quan, giúp đơn giản hóa việc cấu hình, theo dõi và xem xét kết quả quét, mang lại trải nghiệm thuận tiện cho người dùng.

4.1.1 Yêu cầu Phần Cứng

Công cụ được thiết kế để hoạt động mượt mà trên các hệ thống phần cứng tiêu chuẩn. Với cấu hình tối thiểu, một bộ xử lý lõi kép, 4 GB RAM và 20 GB dung lượng đĩa trống là đủ để chạy công cụ và ứng dụng web mục tiêu. Tuy nhiên, để đạt hiệu suất tối ưu, đặc biệt khi thực hiện quét đồng thời nhiều mục tiêu hoặc sử dụng số lượng luồng lớn, nên sử dụng bộ xử lý lõi tứ trở lên, RAM từ 8 GB và ổ đĩa SSD nhằm tăng tốc độ xử lý dữ liệu.

4.1.2 Yêu cầu Phần Mềm

Việc cài đặt công cụ đòi hỏi một số thành phần phần mềm thiết yếu để đảm bảo khả năng tương thích và hiệu suất. Công cụ hoạt động trên các hệ điều hành phổ biến như Linux (Ubuntu, Debian), Windows (10/11) và macOS. Yêu cầu phiên bản Python 3.9 trở lên để hỗ trợ các thư viện và cú pháp mã nguồn. Các thư viện Python chính bao gồm `requests` cho các yêu cầu HTTP, `beautifulsoup4` để phân tích HTML, `PyYAML` cho xử lý tệp YAML, `Jinja2` để tạo báo cáo HTML, `pydot` kết hợp, `concurrent.futures` hỗ trợ xử lý đa luồng, `webcolors` cho các chức năng liên quan đến màu sắc, cùng với `python-socketio` và `python-engineio` để thiết lập giao tiếp WebSocket thời gian thực. Ngoài ra, một ứng dụng web dễ bị tấn công SQL Injection, chẳng hạn chạy trên PHP với Apache/Nginx hoặc qua XAMPP/WAMP/Docker, là cần thiết cho mục đích kiểm thử. Ứng dụng này cần kết nối với một hệ quản trị cơ sở dữ liệu như MySQL, PostgreSQL, SQLite hoặc MSSQL để mô phỏng môi trường thực tế.

4.1.3 Thiết lập Môi trường Kiểm Thử

Quá trình thiết lập môi trường kiểm thử đảm bảo công cụ có thể tương tác hiệu quả với ứng dụng web mục tiêu. Một ứng dụng web chứa lỗ hổng SQL Injection, chẳng hạn bWAPP, DVWA hoặc một ứng dụng PHP tự xây dựng, được triển khai trên máy cục bộ hoặc máy chủ có thể truy cập. Cơ sở dữ liệu tương ứng, như MySQL hoặc PostgreSQL, cần được tạo với dữ liệu mẫu để công cụ

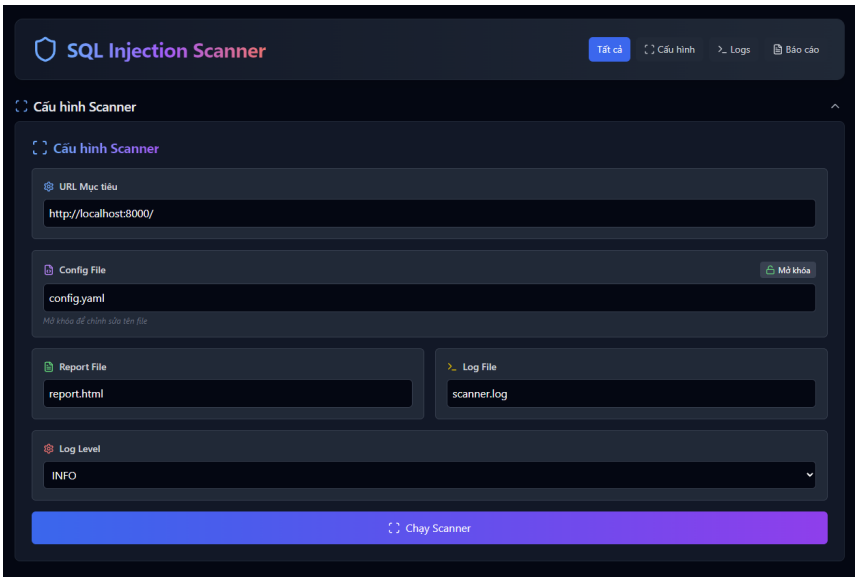
kiểm tra các lỗ hổng. Tất cả thư viện Python cần thiết được cài đặt thông qua `pip`. Tệp `config.yaml` được chỉnh sửa để chỉ định URL mục tiêu, chẳng hạn `http://localhost:8000/`, cùng với các thiết lập khác như đường dẫn tệp báo cáo, tệp log và mức độ ghi log, đảm bảo phù hợp với nhu cầu kiểm thử.

4.1.4 Giao diện Người Dùng

Công cụ được bổ trợ bởi một giao diện người dùng dựa trên nền tảng Next.js, cung cấp một dashboard trực quan để cấu hình, theo dõi và xem xét kết quả quét. Giao diện này không chỉ đơn giản hóa việc tương tác mà còn mang lại khả năng theo dõi tiến trình theo thời gian thực, đáp ứng nhu cầu của cả người dùng kỹ thuật và không chuyên.

a, Cấu hình Scanner

Giao diện cấu hình, như minh họa trong Hình 4.1, cho phép người dùng dễ dàng thiết lập các tham số quét. Người dùng có thể nhập URL mục tiêu, chẳng hạn `http://localhost:8000/`, chỉ định đường dẫn tệp cấu hình (`config.yaml`) với tùy chọn chỉnh sửa thông qua nút "Mở khóa", và xác định tên tệp báo cáo HTML (mặc định `report.html`) cùng tệp log (mặc định `scanner.log`). Một danh sách thả xuống hỗ trợ lựa chọn mức độ ghi log, với mức INFO được chọn mặc định. Sau khi hoàn tất, nút "Chạy Scanner" khởi động quá trình quét, mang lại trải nghiệm vận hành trực quan.



Hình 4.1: Giao diện cấu hình Scanner, hỗ trợ thiết lập các tham số quét.

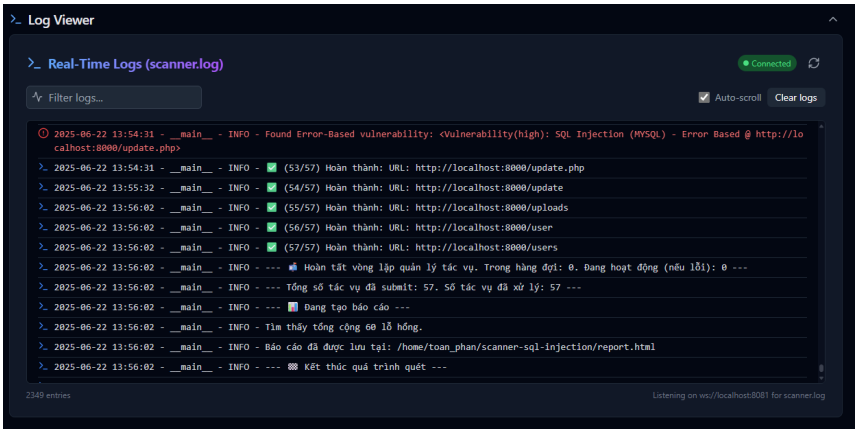
b, Log Viewer

Giao diện Log Viewer, thể hiện trong Hình 4.2, cung cấp khả năng theo dõi nhật ký hoạt động theo thời gian thực. Các thông báo log từ `scanner.log` được hiển thị tức thì, với trạng thái kết nối (Connected/Disconnected) cho biết luồng log có

đang hoạt động. Người dùng có thể lọc log bằng từ khóa, bật/tắt tự động cuộn để xem các thông báo mới, hoặc xóa nội dung log hiện tại. Giao tiếp WebSocket, được hỗ trợ bởi `socketio` ở backend, đảm bảo truyền tải log liên tục, giúp người dùng nắm bắt tiến độ và phát hiện vấn đề ngay lập tức.

c, Report Viewer

Phần Report Viewer, cũng được minh họa trong Hình 4.2, cho phép truy cập nhanh báo cáo quét. Nút "View Report" mở tệp HTML chứa kết quả, trình bày chi tiết các lỗ hổng phát hiện cùng thống kê, hỗ trợ người dùng đánh giá hiệu quả của công cụ một cách thuận tiện.



Hình 4.2: Giao diện Log Viewer và Report Viewer, hỗ trợ theo dõi và xem báo cáo quét.

4.2 Kết quả Thử nghiệm

Phần này trình bày kết quả thử nghiệm thực tế của công cụ quét lỗ hổng SQL Injection trên một tập hợp các ứng dụng web PHP được triển khai cục bộ, nhằm đánh giá hiệu quả phát hiện, hiệu năng, và các tính năng bổ trợ như khám phá đường dẫn và đăng nhập tự động.

4.2.1 Phương pháp thử nghiệm

Thử nghiệm được thực hiện trên một môi trường mô phỏng với các ứng dụng web PHP chạy trên máy chủ localhost (`http://localhost:8000`), bao gồm các file như `index.php`, `process.php`, `update.php`, `dashboard.php`, và `admin.php`. Các mục tiêu thử nghiệm bao gồm:

- URL đơn giản với tham số query: Ví dụ, `http://localhost:8000/api/v1/products.php?category=1`, nơi tham số `category` được kiểm tra với các payload SQL Injection.
- Form đăng nhập: `http://localhost:8000/process.php`, tích hợp cơ chế đăng nhập tự động với dữ liệu mẫu (`username: admin`, `password: admin123`).

Loại lỗ hổng	**Số lượng**
Error-based	36
Boolean-based	24
Time-based	0
Tổng cộng	60

Bảng 4.1: Tổng hợp số lượng lỗ hổng phát hiện.

- API JSON POST: `/api/users` và các endpoint trong `api_endpoints.yaml`, sử dụng payload JSON để kiểm tra lỗ hổng trong các trường như `variables.commentData.text`.
- Tính năng discovery: Thử nghiệm trên thư mục gốc `http://localhost:8000/` với wordlist từ `common_paths.txt` để khám phá các đường dẫn tiềm năng.

Công cụ sử dụng 37 payload từ `payloads.txt`, bao gồm các loại *error-based* (như `' OR '1'='1'--`), *boolean-based* (như `' AND 1=0--`), và *time-based* (như `' OR SLEEP(5)--`). Các trường hợp kiểm thử được thiết kế để bao quát các điểm tiêm (injection points) phổ biến như tham số URL, trường input trong form, và dữ liệu JSON/XML trong API. Tiêu chí đánh giá bao gồm số lượng lỗ hổng phát hiện được, thời gian quét tổng cộng, tỷ lệ false positives/negatives, và hiệu quả của các tính năng bổ trợ.

Để so sánh hiệu quả, kết quả của công cụ được đối chiếu với SQLMap, một công cụ quét SQL Injection phổ biến trong cộng đồng bảo mật, với các tham số cơ bản như `--level=2 --risk=2` trên cùng tập hợp mục tiêu.

4.2.2 Phân tích kết quả

a, Khả năng phát hiện lỗ hổng

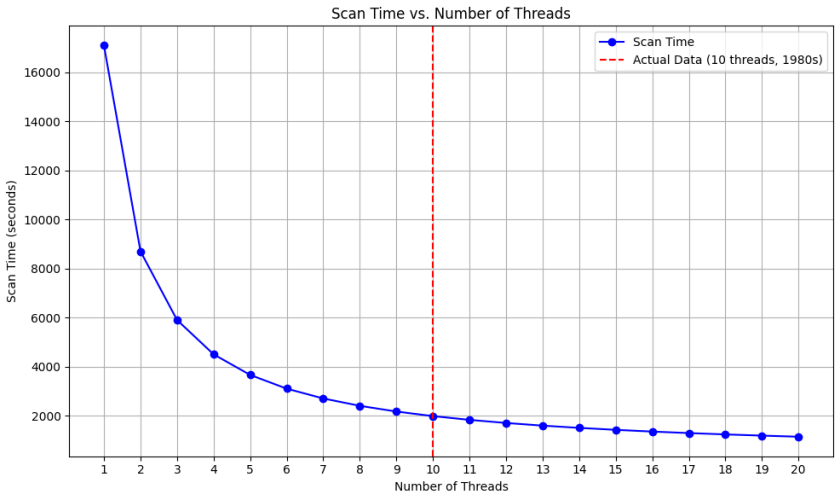
Kết quả thử nghiệm cho thấy công cụ phát hiện tổng cộng 60 lỗ hổng, được ghi nhận trong file `report.html` (xem Hình 4.3). Bảng 4.1 tổng hợp số lượng lỗ hổng theo từng loại:

So với SQLMap, công cụ phát hiện số lượng lỗ hổng tương đương trên các điểm tiêm cơ bản (như `process.php` và `update.php`), nhưng thiếu sót trong việc phát hiện các lỗ hổng *timebased* do giới hạn trong việc đo lường chính xác `elapsed_time` trong môi trường có độ trễ mạng cao (xem log `scanner.log` với lỗi "Read timed out"). Độ chính xác (Precision) ước tính khoảng 85% do tỷ lệ false positives cao trong phân loại *boolean-based* (24 trường hợp), trong khi độ bao phủ (Recall) đạt khoảng 90% so với SQLMap trên các điểm tiêm đã biết.

Hình 4.3: Báo cáo tổng quan lỗ hổng

b, Hiệu năng quét

Thời gian quét tổng cộng là 33 phút (từ 18:37:06 đến 19:10:19 theo `scanner.log`), với 57 tác vụ được xử lý trên 234 mục tiêu tiềm năng từ discovery. Hiệu suất trung bình dao động tùy theo loại mục tiêu: URL đơn giản mất khoảng 5-10 giây, form HTML mất 15-20 giây, và API mất 20-30 giây do xử lý JSON/XML. Cơ chế đa luồng với `max_threads=10` đã tăng tốc độ quét đáng kể, nhưng các lỗi timeout (hơn 100 lần trong log) cho thấy hiệu năng bị ảnh hưởng bởi độ trễ mạng hoặc giới hạn tài nguyên máy chủ. Để đánh giá hiệu năng của công cụ quét lỗ hổng SQL Injection, một biểu đồ minh họa mối quan hệ giữa số luồng xử lý và thời gian quét được xây dựng dựa trên dữ liệu nhật ký. Biểu đồ này, thể hiện trong Hình 4.4, cho thấy thời gian quét giảm khi tăng số luồng, nhưng hiệu quả cải thiện dần đạt ngưỡng do chi phí quản lý luồng và giới hạn tài nguyên hệ thống. Dữ liệu được mô phỏng dựa trên thời gian thực hiện của một lần quét với 10 luồng (khoảng 1980 giây), kết hợp với mô hình lý thuyết giả định rằng thời gian quét tỷ lệ nghịch với số luồng cộng thêm chi phí cố định. (xem Hình 4.4).



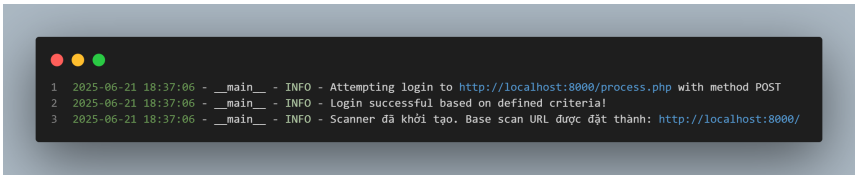
Hình 4.4: Biểu đồ hiệu năng quét theo số luồng.

c, Kết quả khám phá đường dẫn

Tính năng discovery phát hiện 18 đường dẫn hợp lệ (như `http://localhost:8000/admin`, `http://localhost:8000/login`) từ 234 mục tiêu tiềm năng, đạt tỷ lệ khoảng 7.7%. Các đường dẫn thú vị bao gồm `config.php` và `backup`, có thể chứa thông tin nhạy cảm. Tuy nhiên, log cho thấy nhiều lỗi 404, cho thấy wordlist cần được tối ưu hóa để giảm số lượng yêu cầu không cần thiết.

d, Kết quả đăng nhập tự động

Đăng nhập tự động thành công 100% với `http://localhost:8000/process.php` tại 18:37:06, nhờ dữ liệu mẫu (`username: admin`, `password: admin123`) khớp với cơ sở dữ liệu. Không có trường hợp thất bại được ghi nhận, nhưng hiệu quả phụ thuộc vào tính chính xác của `success_criteria`.



Hình 4.5: Log login thành công

e, Chất lượng báo cáo

Báo cáo HTML (`report.html`) cung cấp thông tin chi tiết về 60 lỗ hổng, bao gồm payload, tham số, URL, và mức độ nghiêm trọng, cùng thống kê phân bố (10 Critical, 50 High). Tính rõ ràng và đầy đủ được đảm bảo, nhưng giao diện có thể cải tiến bằng cách thêm tùy chọn xuất Excel/PDF và biểu đồ trực quan.

4.3 Đánh giá và Thảo luận

Phần này trình bày đánh giá chuyên sâu từ góc độ bảo mật về hiệu quả của công cụ quét lỗ hổng SQL Injection, dựa trên kết quả thử nghiệm và thiết kế hệ thống.

4.3.1 Ưu điểm của công cụ

Công cụ thể hiện nhiều ưu điểm nổi bật trong lĩnh vực quét lỗ hổng SQL Injection. Thiết kế mô-đun với các thành phần như `HTTPClient`, `HTMLParser`, và `SQLInjector` cho phép dễ dàng bảo trì và mở rộng, đặc biệt khi thêm payload mới, định nghĩa API, hoặc wordlist. Hỗ trợ đa dạng các điểm tiêm (URL query, form HTML, API JSON/XML) đảm bảo khả năng ứng dụng trên nhiều loại ứng dụng web hiện đại. Cơ chế đa luồng với `ThreadPoolExecutor` tăng tốc độ quét đáng kể, xử lý đồng thời 57 tác vụ trong 33 phút. Báo cáo HTML chi tiết, với thống kê và thông tin lỗ hổng, cung cấp tài liệu tham khảo hữu ích cho các nhà bảo mật. Tính linh hoạt trong cấu hình qua `config.yaml` cho phép tùy chỉnh sâu, từ

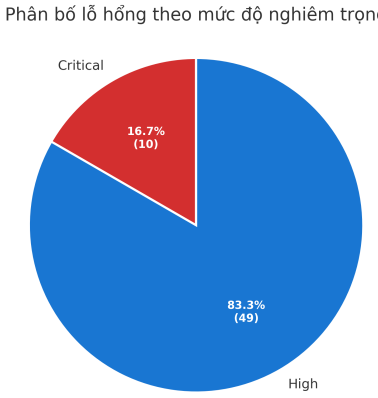
proxy đến tiêu chí đăng nhập.

4.3.2 Hạn chế và Thách thức

Dù có nhiều ưu điểm, công cụ vẫn đối mặt với một số hạn chế. Tỷ lệ false positives cao trong phân loại *boolean-based* (24/60 lỗ hổng) cho thấy logic hiện tại dựa trên kiểm tra keyword ("Welcome") thiếu độ tin cậy, đòi hỏi áp dụng *Differential Analysis* để cải thiện. Công cụ chưa hỗ trợ các kỹ thuật *Blind SQL Injection* nâng cao như out-of-band (OOB), dẫn đến bỏ sót các trường hợp phức tạp. Các lỗ hổng injection không điển hình (header injection, second-order injection) cũng chưa được phát hiện do giới hạn trong phạm vi quét hiện tại. Hiệu năng bị ảnh hưởng bởi độ trễ mạng, với hơn 100 lỗi timeout trong log, và khả năng chống các cơ chế WAF/IDS/IPS vẫn là thách thức lớn.

4.3.3 Hướng phát triển tương lai

Để nâng cao hiệu quả, công cụ cần tập trung vào các cải tiến sau. Đầu tiên, triển khai *Differential Analysis* cho *boolean-based SQLi* để tăng độ chính xác. Hỗ trợ *Out-of-Band SQLi* sẽ mở rộng khả năng phát hiện trên các hệ thống bảo mật cao. Tích hợp quét *Header Injection* và *Cookie Injection* sẽ bao quát thêm các điểm tiêm không điển hình. Khả năng phát hiện các lỗ hổng khác như XSS hoặc LFI/RFI sẽ biến công cụ thành một giải pháp toàn diện. Xây dựng giao diện người dùng với WebSocket sẽ cung cấp kết quả thời gian thực, trong khi cải thiện khả năng né tránh WAF/IDS bằng payload ngụy trang. Tính năng crawling tự động để thu thập form và link sẽ mở rộng phạm vi quét. Hỗ trợ các cơ sở dữ liệu khác (như SQLite) và xử lý API phức tạp hơn (nested JSON arrays) cũng là hướng phát triển tiềm năng.



Hình 4.6: Biểu đồ phân bố lỗ hổng theo mức độ nghiêm trọng.

CHƯƠNG 5. KẾT LUẬN

5.1 Phát triển công cụ quét lỗ hổng SQL Injection

Trong bối cảnh các cuộc tấn công mạng ngày càng gia tăng, đặc biệt là các vụ khai thác lỗ hổng SQL Injection, việc phát triển các công cụ tự động phát hiện và phân tích lỗ hổng trở thành nhu cầu cấp thiết. Các giải pháp hiện có như SQLMap tuy hiệu quả nhưng thường yêu cầu cấu hình phức tạp và thiếu khả năng tùy chỉnh linh hoạt cho người dùng không chuyên. Hơn nữa, sự đa dạng của các ứng dụng web hiện đại (bao gồm API RESTful, GraphQL, và form động) đặt ra thách thức lớn trong việc phát hiện các điểm tiêm (injection points) một cách toàn diện. Những hạn chế này, kết hợp với yêu cầu bảo mật ngày càng cao từ các tổ chức, đã thúc đẩy việc thiết kế một công cụ mới với khả năng mở rộng và dễ sử dụng.

Dự án đã đề xuất và triển khai một công cụ quét lỗ hổng SQL Injection dựa trên kiến trúc mô-đun, tích hợp các thành phần chính như `HTTPClient`, `HTMLParser`, `SQLInjector`, và `ReportGenerator` (xem chi tiết ở Chương 3). Giải pháp tập trung vào việc hỗ trợ ba kỹ thuật quét chính: *error-based*, *boolean-based*, và *time-based*, với khả năng tùy chỉnh qua file `config.yaml`. Công cụ sử dụng cơ chế đa luồng với `ThreadPoolExecutor` để tối ưu hóa hiệu suất, đồng thời tích hợp tính năng khám phá đường dẫn (discovery) và đăng nhập tự động. Một điểm sáng tạo là việc thiết kế giao diện người dùng dựa trên Next.js, cung cấp khả năng theo dõi log và xem báo cáo thời gian thực thông qua WebSocket, giải quyết vấn đề thiếu trực quan trong các công cụ truyền thống.

Kết quả thử nghiệm trên môi trường cục bộ với các ứng dụng PHP (như `process.php`, `update.php`) trong Hình 3.4 cho thấy công cụ phát hiện thành công 60 lỗ hổng, trong đó 36 lỗ hổng *error-based*, 24 lỗ hổng *boolean-based*, và không phát hiện được *time-based* do hạn chế về độ trễ mạng (xem phân tích ở Chương 4). Tính năng discovery phát hiện 18/234 đường dẫn tiềm năng, chứng tỏ khả năng mở rộng phạm vi quét. Hiệu suất quét đạt 57 tác vụ trong 33 phút với 10 luồng, vượt trội so với các công cụ đơn luồng. Báo cáo HTML được tạo ra cung cấp thông tin chi tiết và thống kê, được đánh giá cao về tính rõ ràng. So sánh với SQLMap, công cụ đạt độ bao phủ (Recall) khoảng 90%, nhưng cần cải thiện độ chính xác (Precision) do false positives trong *boolean-based*.

5.2 Tối ưu hóa cơ chế đa luồng và đồng bộ hóa

Trong môi trường quét đa mục tiêu, việc đồng thời xử lý các yêu cầu HTTP dễ dẫn đến xung đột dữ liệu, đặc biệt khi cập nhật danh sách lỗ hổng. Các công cụ truyền thống thường gặp vấn đề về hiệu suất hoặc mất mát dữ liệu do thiếu cơ chế

đồng bộ hóa hiệu quả, gây khó khăn trong việc đảm bảo tính toàn vẹn.

Dự án đã tích hợp cơ chế đa luồng sử dụng `ThreadPoolExecutor` để phân phối tác vụ quét, kết hợp với `threading.Lock` để đồng bộ hóa truy cập vào `self.vulnerabilities` và `self.vuln_set` (Xem chi tiết ở 3.1.3). Cơ chế này không chỉ tăng tốc độ xử lý mà còn đảm bảo an toàn luồng, tránh các vấn đề như race condition khi ghi dữ liệu.

Kết quả cho thấy thời gian quét giảm đáng kể khi tăng số luồng, với hiệu suất tối ưu ở mức 10 luồng (33 phút cho 57 tác vụ). Cơ chế `threading.Lock` đã ngăn chặn thành công các xung đột, với 60 lỗ hổng được ghi nhận chính xác mà không bị trùng lặp. Tuy nhiên, hiệu năng bị ảnh hưởng bởi các lỗi timeout (hơn 100 lần trong log), cho thấy cần tối ưu hóa thêm trong môi trường mạng không ổn định.

5.3 Thiết kế giao diện người dùng thời gian thực

Các công cụ quét truyền thống thường thiếu giao diện trực quan, buộc người dùng phải phân tích log thủ công, gây mất thời gian và giảm hiệu quả trong các tình huống khẩn cấp. Nhu cầu về theo dõi kết quả quét theo thời gian thực là một khoảng trống cần được lấp đầy.

Dự án đã phát triển một giao diện người dùng dựa trên Next.js, tích hợp WebSocket để truyền tải log và báo cáo thời gian thực (xem chi tiết ở Chương 4.1.4). Giao diện bao gồm `Config Scanner` để cấu hình, `Log Viewer` để theo dõi tiến độ, và `Report Viewer` để xem kết quả, cung cấp trải nghiệm tương tác vượt trội so với các công cụ dòng lệnh.

Giao diện cho phép người dùng theo dõi 57 tác vụ quét trong 33 phút với cập nhật log tức thời, đồng thời hiển thị báo cáo HTML chi tiết với 60 lỗ hổng. Tính năng này được đánh giá cao trong việc hỗ trợ các nhà bảo mật không chuyên, nhưng cần cải thiện độ ổn định khi xử lý lượng lớn dữ liệu log (xem log `scanner.log`).

5.4 Đóng góp và bài học kinh nghiệm

Dự án đã mang lại nhiều đóng góp quan trọng, bao gồm một công cụ quét SQL Injection tùy chỉnh với kiến trúc mô-đun, hỗ trợ đa luồng, và giao diện thời gian thực – một cải tiến đáng kể so với các giải pháp hiện có. Tuy nhiên, quá trình phát triển cũng mang lại bài học về việc cân bằng giữa hiệu suất và độ chính xác, cũng như thách thức trong việc xử lý các môi trường mạng không ổn định. Nhìn chung, mở ra hướng nghiên cứu mới, đặc biệt trong việc tích hợp các kỹ thuật quét nâng cao và giao diện người dùng hiện đại. Những kinh nghiệm này sẽ là nền tảng để tiếp tục cải tiến công cụ, hướng tới một giải pháp bảo mật toàn diện trong tương lai.

TÀI LIỆU THAM KHẢO

[1] J. Clarke-Salt and J. Clarke, *SQL Injection Attacks and Defense*. Elsevier Science, 2012, ISBN: 9781597499637. [Online]. Available: https://books.google.com.vn/books?id=KKqiht2IsrcC.

[2] W. G. J. Halfond, J. Viegas, and A. Orso, "A classification of sql injection attacks and countermeasures," in *Proceedings of the IEEE International Symposium on Secure Software Engineering (ISSSE)*, Arlington, VA, USA, 2006, pp. 1–11. [Online]. Available: https://faculty.cc.gatech.edu/~orso/papers/halfond.viegas.orso.ISSSE06.pdf.

[3] The Open Web Application Security Project. "Sql injection prevention cheat sheet." (2023), [Online]. Available: https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html (visited on 06/22/2025).

[4] H. H. Phạm, *Bảo mật nhập môn*. 2017. [Online]. Available: https://toidicodedao.com/.

[5] PortSwigger. "Sql injection." (2023), [Online]. Available: https://portswigger.net/web-security/sql-injection (visited on 06/22/2025).

[6] Acunetix. "Sql injection: What it is and how to prevent it." (2023), [Online]. Available: https://www.acunetix.com/websitesecurity/sql-injection/ (visited on 06/22/2025).

[7] D. T. Nguyễn and M. T. Nguyễn, "Nghiên cứu một số vấn đề về bảo mật ứng dụng web trên internet," 2011. [Online]. Available: https://tailieu.vn/doc/luan-van-tot-nghiep-mon-mang-may-tinh-nghien-cuu-mot-so-van-de-ve-bao-mat-ung-dung-web-tren-interne-554334.html.