ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA KHOA HỌC VÀ KĨ THUẬT MÁY TÍNH

LUẬN VĂN TỐT NGHIỆP

XÂY DỰNG ỨNG DỤNG KIỂM THỬ BẢO MẬT ỨNG DỤNG WEB THÔNG QUA THÔNG ĐIỆP HTTP

Building a HTTP message-based fuzzer for web application security testing

Hội đồng LVTN: Khoa học máy tính

Tập thể hướng dẫn:

TS. Nguyễn An Khương	—	Khoa KH&KT Máy tính, ĐHBK
Kỹ sư Đỗ Đình Huân	—	VNG Corporation
ThS. Nguyễn Cao Đạt	—	Khoa KH&KT Máy tính, ĐHBK
ThS. Nguyễn Thanh Tùng	—	Khoa KH&KT Máy tính, ĐHBK

Giáo viên phản biện:

TS. Lê Thành Sách	—	Khoa KH&KT Máy tính, ĐHBK

Sinh viên thực hiện:

Đặng Nhật Trình	—	1513685

Tp. Hồ Chí Minh, Tháng 9, 2020

Lời cảm ơn

Nhìn lại những thăng trầm trong quá trình thực hiện luận văn tốt nghiệp cũng như quãng đường sinh viên sắp kết thúc của mình, tôi cảm thấy thật may mắn khi đã nhận được rất nhiều sự động viên, giúp đỡ từ gia đình, những người thầy, anh em, bạn bè, về cả vật chất lẫn tinh thần trong suốt quá trình học tập ở Đại học Bách khoa TPHCM.

Đầu tiên, tôi xin gửi lời cảm ơn chân thành nhất từ đáy lòng đến thầy Nguyễn An Khương và anh Đỗ Đình Huân (Senior Red team Security Engineer - VNG Corporation). Ngoài những lời khuyên đầy kinh nghiệm về mặt chuyên môn, học thuật dành cho tôi, hai người còn dạy tôi về thái độ, tinh thần trách nhiệm cần có của một người đàn ông. Anh Huân còn là người đề ra hướng phát triển ứng dụng, luôn theo sát hỗ trợ tôi trong quá trình hiện thực và sửa đổi ứng dụng dựa vào những kinh nghiệm trong nghề bảo mật của mình. Tôi cũng xin gửi lời cảm ơn đến thầy Nguyễn Cao Đạt và thầy Nguyễn Thanh Tùng. Những góp ý trong giai đoạn ban đầu của hai thầy về phương hướng thiết kế một ứng dụng kiểm thử bảo mật là một phần cơ sở để tôi hoàn thành tốt luận văn tốt nghiệp này.

Trong khoảng thời gian bắt đầu nghiên cứu về ngành an toàn thông tin, tôi đã rất may mắn được dìu dắt bởi anh Nguyễn Lê Thành, anh Nguyễn Anh Quỳnh, anh Nguyễn Văn Hòa và bạn Nguyễn Quốc Bảo. Mọi người đã luôn tạo điều kiện cho tôi phát triển bản thân, thông cảm và giúp đỡ tôi rất nhiều trong quá trình nghiên cứu khoa học và làm việc ở VNG Corporation.

Bên cạnh đó, không thể không nhắc tới anh Lê Nhật Quang và bạn Nguyễn Văn Thành, những người đồng nghiệp, người anh em đáng tin cậy nhất của tôi. Họ luôn ở bên cạnh động viên, hỗ trợ tôi rất nhiều mỗi khi tôi gặp khó khăn trong công việc chuyên môn cũng như trong cuộc sống cá nhân.

Ngoài ra, tôi cũng muốn dành lời cảm ơn đến thầy Nguyễn Hứa Phùng, hai em Võ Vĩ Khang (K2017) và Trần Ngọc Tín (K2016), đang sinh hoạt tại câu lạc bộ An toàn thông tin EFIENS, khoa Khoa học và Kỹ thuật Máy tính, đại học Bách Khoa TPHCM. Những góp ý của thầy và hai em rất thiết thực để luận văn này hoàn thiện hơn, về cả hình thức lẫn nội dung.

Sau cùng, tôi muốn bày tỏ lòng biết ơn sâu sắc nhất đến ba mẹ, những người luôn yêu thương tôi vô điều kiện. Ba mẹ sẽ luôn là nguồn động lực to lớn thôi thúc tôi vượt qua

những rào cản của bản thân để đạt được những thành công to lớn hơn trong cuộc đời mình.

Tp. Hồ Chí Minh, Tháng 9/2020.

Tóm tắt luận văn

Thế giới hiện đại ngày càng phụ thuộc nhiều vào Internet, các ứng dụng web cũng ngày càng phổ biến và đa dạng hơn về hình thức lẫn chức năng và ngày càng được sử dụng cho nhiều dịch vụ quan trọng. Tuy nhiên, với bản chất là phần mềm máy tính, các ứng dụng web hoàn toàn có khả năng xuất hiện các lỗ hổng bảo mật, điều đó khiến chúng trở thành những mục tiêu giá trị trong các cuộc tấn công bảo mật. Các lỗ hổng này có thể bắt nguồn từ sự thiếu kinh nghiệm trong quá trình lập trình, hoặc do sự thất bại của hệ thống trong việc phòng thủ khỏi những tác vụ, dữ liệu nguy hiểm từ phía người dùng. Mặc dù trong cộng đồng an toàn thông tin trên thế giới đã xuất hiện nhiều công cụ, dịch vụ và khung thức hỗ trợ kiểm thử bảo mật ứng dụng web nhưng chưa có công cụ nào kết hợp đầy đủ các yếu tố miễn phí, mã nguồn mở, giao diện trực quan, dễ sử dụng, tự động kiểm thử nhanh và ổn định nhiều lỗ hổng bảo mật cơ bản. Trong quá trình thực hiện luận văn này, chúng tôi tiến hành khảo sát về một số phương pháp kiểm thử bảo mật, các thành phần quan trọng cũng như một vài lỗ hổng bảo mật thường gặp ở ứng dụng web, từ đó áp dụng và hiện thực thành công ứng dụng kiểm thử bảo mật **webfuzzer** đáp ứng đầy đủ những yêu cầu trên.

Mục lục

Danh sách bảng

Danh sách hình vẽ

Danh mục viết tắt

API Application Programming Interface. 28, 32, 33, 35, 37–39, 41–44, 49, 50, 58–60, 65, 68, 69

CMS Content Management System. 1, 20

CSS Cascading Style Sheets. 33, 61, 69

CVE Common Vulnerabilities and Exposure. 20, 21

CWE Common Weakness Enumeration. 20

DNS Domain Name Server. 17

DoS Denial of Service. 15, 17, 19

HTML Hypertext Making Language. 7, 33, 58, 65

HTTP Hypertext Transfer Protocol. viii, 3, 5–8, 15, 23, 24, 28, 30, 34–37, 39–41, 44, 46, 47, 49–52, 54, 58, 73, 75, 85, 87

LFI Local File Inclusion. 3, 4, 12, 14, 46–49, 57, 79–83

RCE Remote Code Execution. 12, 16, 19, 21

SASS Syntactically Awesome Style Sheets. 33, 34

SQL Structured Query Language. 15, 16, 34

SQLI Structured Query Language Injection. 3, 4, 12, 14–16, 19, 36, 47, 48, 53, 55, 79–85, 87

UI User Interface. 27, 30, 31, 33, 35, 37, 40, 41, 49, 58–60, 63, 65, 68, 70, 71, 87–89

URL Uniform Resource Locator. 6, 13, 26, 35, 38, 39, 46, 65, 66, 68, 73

VPS Virtual Private Service. 25, 41, 85, 88

WAF Web Application Firewall. 19

XSRF Cross-Site Request Forgery. 87

XSS Cross-Site Scripting. 11, 14, 19, 21

Chương 1

Giới thiệu

1.1 Đặt vấn đề

Thế giới hiện đại ngày càng phụ thuộc nhiều vào Internet, các ứng dụng web cũng ngày càng phổ biến và đa dạng hơn về hình thức lẫn chức năng. Các khung thức, ngôn ngữ lập trình web cũng như các hệ quản trị nội dung (content management system - CMS) ngày càng được cải tiến để thuận tiện hơn trong quá trình phát triển và sử dụng ứng dụng web. Ứng dụng web cũng ngày càng được sử dụng cho nhiều dịch vụ quan trọng như tài chính, mạng xã hội, các cổng tra cứu thông tin,... Tuy nhiên, với bản chất là phần mềm máy tính, các ứng dụng web hoàn toàn có khả năng xuất hiện các lỗ hổng bảo mật, điều đó khiến chúng trở thành những mục tiêu giá trị trong các cuộc tấn công bảo mật. Các lỗ hổng này có thể bắt nguồn từ sự thiếu kinh nghiệm trong quá trình lập trình, hoặc do sự thất bại của hệ thống trong việc phòng thủ khỏi những dữ liệu nguy hiểm từ phía người dùng. Trong lịch sử đã có rất nhiều vụ tấn công đình đám nhằm thay đổi giao diện, đánh cắp thông tin, hay tấn công từ chối dịch vụ trên nhiều cụm ứng dụng web phổ biến trên thế giới. Trong bối cảnh đó, các tổ chức, dự án bảo mật ứng dụng web lớn cũng xuất hiện theo thời gian và ngày càng lớn mạnh.

Có thể thấy, nhu cầu đảm bảo bảo mật cho ứng dụng web là cực kì cấp thiết. Hiện nay trong cộng đồng an toàn thông tin trên thế giới đã xuất hiện nhiều công cụ, dịch vụ hỗ trợ kiểm thử bảo mật ứng dụng web. Tuy nhiên mỗi công cụ, dịch vụ trên đều có một số hạn chế riêng, ví dụ như mã nguồn đóng, chỉ tập trung vào việc phát hiện một lỗ hổng

bảo mật duy nhất, các công cụ phát hiện được nhiều loại lỗ hổng thì người dùng phải trả phí để được sử dụng đầy đủ chức năng. Điểm chung của hầu hết các công cụ, dịch vụ này là người dùng không biết được cách thức hoạt động cụ thể của nó ra sao. Quá trình kiểm thử bằng các ứng dụng đó được diễn ra tự động và không cho phép người dùng có kinh nghiệm trực tiếp lựa chọn cụ thể đối tượng kiểm thử, hay cá nhân hóa để ứng dụng vận hành theo ý mình. Họ chỉ nắm được khái niệm cách làm của công cụ chứ không kiểm soát được công cụ đang làm việc gì cụ thể, đang kiểm thử điểm cuối nào của ứng dụng với trường hợp kiểm thử nào.

Vì những lý do đó, chúng tôi sẽ tiến hành khảo sát về các phương pháp kiểm thử bảo mật, các thành phần quan trọng cũng như một vài lỗ hổng bảo mật thường gặp ở ứng dụng web để hiện thực một ứng dụng kiểm thử bảo mật giải quyết được các vấn đề trên.

1.2 Mục tiêu hiện thực ứng dụng

Xây dựng được một ứng dụng kiểm thử một số lỗ hổng bảo mật ở tầng ứng dụng của ứng dụng web từ phía người dùng, kết hợp được những thế mạnh của những công cụ sẵn có trên thị trường. Cụ thể các tính năng cần có của ứng dụng được liệt kê sau đây.

- Ứng dụng phải thực thi nhanh và ổn định, mã nguồn mở, miễn phí, có khả năng kiểm thử được nhiều hơn một lỗ hổng bảo mật đặc thù.
- Giao diện ứng dụng trong sáng, trực quan, dễ sử dụng, thuận tiện cho người dùng trong việc chọn đối tượng kiểm thử và quan sát tiến trình vận hành của ứng dụng.
- Nhận vào dữ liệu đầu vào là các đối tượng kiểm thử cụ thể theo nhu cầu của người dùng và loại lỗ hổng cần kiểm thử.
- Kiểm thử đối tượng đó bằng một loạt các trường hợp kiểm thử được phân loại theo loại lỗ hổng.
- Đối với từng trường hợp kiểm thử, ứng dụng phải kiểm tra kết quả một cách tự động để phát hiện lỗi hoặc các hành vi khả nghi từ phía ứng dụng web mục tiêu.
- Tổng hợp kết quả kiểm thử của mỗi đối tượng theo từng loại lỗ hổng.

Để thực hiện được các tính năng nêu trên, xuyên suốt quá trình thực hiện đề tài cần đạt được những mục tiêu sau.

- Trước quá trình hiện thực ứng dụng, ta cần tìm hiểu và lực chọn (các) phương pháp kiểm thử bảo mật và công cụ, khung thức hỗ trợ thích hợp.
- Tìm hiểu kĩ đặc điểm và cách thức phát hiện tự động từng loại lỗ hổng bảo mật trong phạm vi hiện thực ứng dụng.
- Xây dựng hoặc phát triển thêm một phương pháp định nghĩa và sửa đổi đối tượng kiểm thử cụ thể theo nhu cầu của người dùng dựa trên những công nghệ có sẵn.
- Thiết kế kiến trúc, cấu hình kiểm thử của ứng dụng một cách hợp lý để tiếp nhận và xử lí tốt các đối tượng đó.
- Các trường hợp kiểm thử được sử dụng trong ứng dụng phải phát hiện được càng nhiều lỗ hổng và phương pháp phòng thủ càng tốt.

1.3 Giới hạn của đề tài

Do giới hạn thời gian và khả năng của bản thân, ứng dụng được hiện thực trong luận văn này sẽ chỉ tập trung phát hiện hai lỗ hổng bảo mật ở tầng ứng dụng cụ thể là Local File Inclusion (LFI) và time-based Structured Query Language Injection (SQLI). Điểm chung của hai lỗ hổng này là đều có thể bị khai thác bằng dữ liệu đầu vào từ phía người dùng và dễ dàng xác nhận kết quả kiểm thử một cách tự động thông qua thông báo phản hồi HTTP. Đồng thời, muốn ứng dụng kiểm thử mục tiêu theo nhu cầu người dùng một cách nhanh chóng thì quá trình chọn mục tiêu kiểm thử buộc phải được thực hiện bằng tay để tránh kiểm thử lan man trên nhiều điểm cuối và tham số không liên quan. Việc tự động hóa quá trình này đòi hỏi tốn rất nhiều công sức và kinh nghiệm nhưng hiệu quả đem lại không cao nên chúng tôi sẽ cân nhắc phát triển trong tương lai.

1.4 Bố cục luận văn

Luận văn này sẽ trải dài trên tám chương, nội dung của từng chương như sau.

Ở **Chương 1**, tôi sẽ tập trung giới thiệu về nội dung, giới hạn của đề tài và mục tiêu hiện

thực ứng dụng. Tiếp đến, tại **Chương 2**, các kiến thức nền tảng về phương pháp kiểm thử ứng dụng web và các thành phần quan trọng về mặt bảo mật của một ứng dụng web sẽ được trình bày.

Tiếp theo đó, tôi sẽ giới thiệu các lớp lỗ hổng bảo mật thường gặp trên ứng dụng web, sau đó đi sâu vào tìm hiểu về lỗ hổng LFI và time-based SQLI ở **Chương 3**. Ngay sau đó, tại **Chương 4**, chúng tôi trình bày xu hướng tấn công các ứng dụng web hiện này, đồng thời đề ra phạm vi và hướng thiết kế ứng dụng.

Những yêu cầu hiện thực và thiết kế chi tiết ứng dụng sẽ được chúng tôi trình bày tại **Chương 5**. Cũng trong chương này, chúng tôi giới thiệu các công nghệ (ngôn ngữ lập trình, công cụ, khung thức) được sử dụng trong quá trình hiện thực ứng dụng. Sau đó, chúng tôi trình bày chi tiết hiện thực và đánh giá kết quả đạt được lần lượt ở **Chương 6** và **Chương 7** của luận văn.

Cuối cùng, tại **Chương 8**, chúng tôi sẽ tổng kết về luận văn tốt nghiệp và trình bày hướng phát triển ứng dụng trong tương lai.

Chương 2

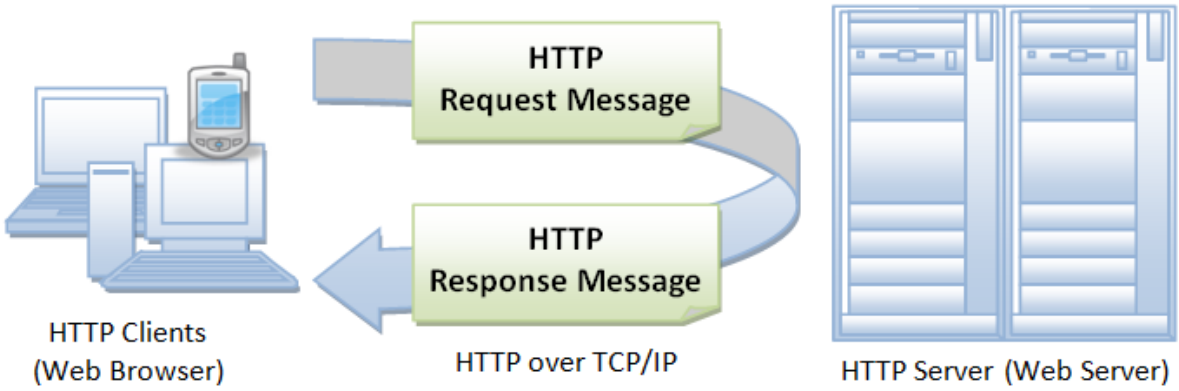
Kiến thức nền tảng

Chương này trình bày các kiến thức nền tằng về ứng dụng web cần thiết để xây dựng webfuzzer, cùng với hai phương pháp kiểm thử bảo mật phần mềm phổ biến, hiệu quả nhất hiện nay, đó là fuzzing và kiểm thử xâm nhập.

2.1 Các thành phần quan trọng về bảo mật của ứng dụng web

2.1.1 Thông điệp HTTP

Giao thức truyền tải siêu văn bản (hypertext transfer protocol - HTTP) là giao thức ứng dụng bất đồng bộ, không trạng thái (stateless) phổ biến nhất dùng để truyền tải dữ liệu qua lại giữa người dùng (client) và máy chủ (server). Có hai loại thông điệp HTTP là thông báo yêu cầu và thông báo phản hồi HTTP, được mô tả như Hình 2.1 dưới đây:

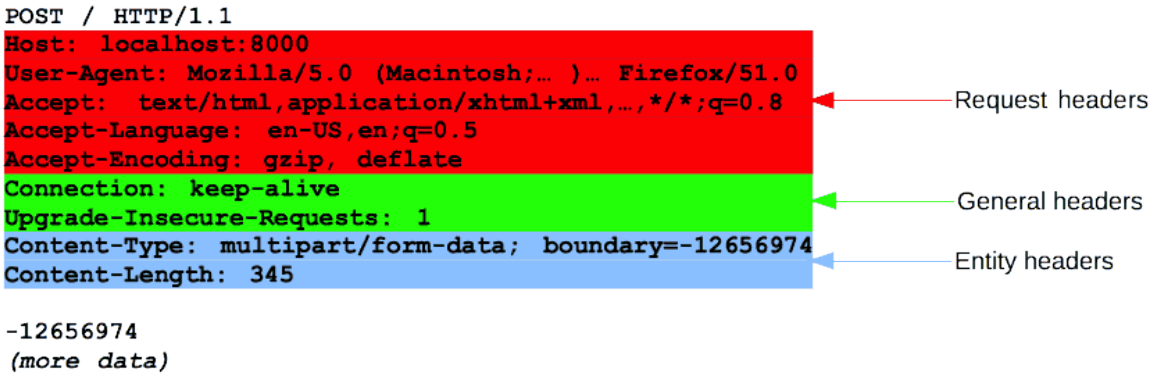


Hình 2.1: Hai loại thông điệp HTTP[1]

Thông báo yêu cầu HTTP (HTTP request) được gửi từ người dùng đến máy chủ để yêu cầu máy chủ thực thi một hành động nào đó. HTTP request gồm các thành phần:

- **Dòng bắt đầu (start line)** ví dụ như `POST / HTTP/1.1`, `GET /1-duck.png HTTP/1.0`, `HEAD /items.html?productId=1337 HTTP/1.1`, `OPTIONS /index.html HTTP/1.0`. Dòng bắt đầu này bao gồm 3 thành phần:
  - **Phương thức HTTP (HTTP method)** dùng để mô tả hành động sẽ được thực thi bởi request đó. Những phương thức thường gặp là `GET`, `POST`, `PUT`, `HEAD`, `OPTIONS`,...
  - **Đối tượng yêu cầu (request target)** thường là một đường dẫn URL hoặc đường dẫn tuyệt đối đến tài nguyên trên máy chủ, nơi mà máy chủ sẽ thực hiện hành động đã yêu cầu.
  - **Phiên bản HTTP** định nghĩa cấu trúc của phần còn lại của request, đồng thời đóng vai trò chỉ thị phiên bản HTTP mà response sẽ dùng.
- **Tiêu đề (headers)** của một request gồm nhiều dòng, mỗi dòng được thiết kế theo cấu trúc quy định: một chuỗi kí tự (phân biệt hoa-thường) và giá trị có cấu trúc tương ứng tùy theo từng chuỗi, phân cách nhau bởi dấu hai chấm. Các dòng tiêu đề này được phần làm 3 loại chính như Hình 2.2 sau:

[1]Nguồn: https://www.ntu.edu.sg/home/ehchua/programming/webprogramming/HTTP_Basics.html



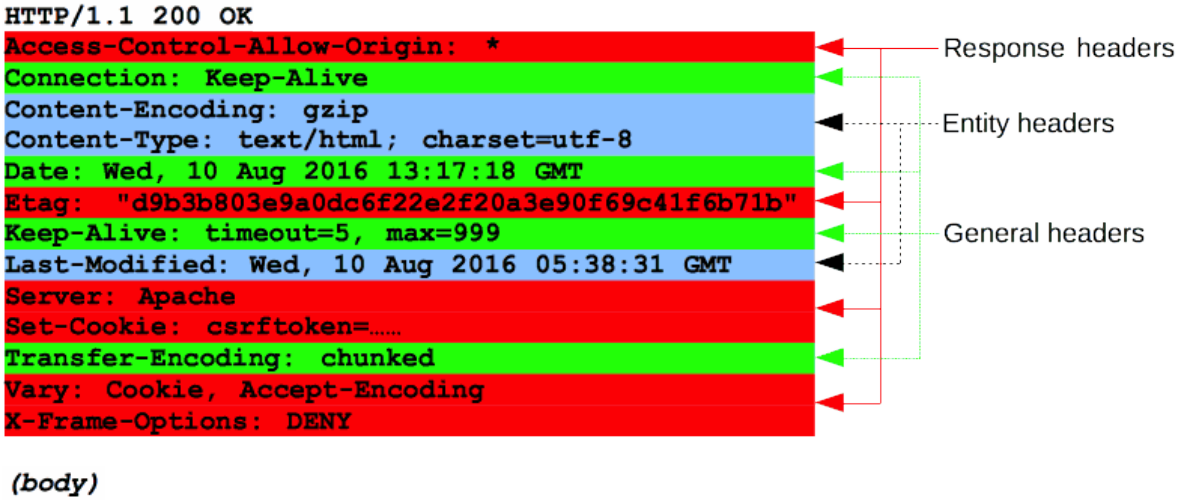
Hình 2.2: Các thành phần trong tiêu đề của một HTTP request[2]

- **Tiêu đề chung (general headers)** gồm những quy tắc/phiên bản được áp dụng lên toàn bộ request.
- **Tiêu đề request (request headers)** làm rõ request bằng việc đặc tả kĩ hơn về bối cảnh yêu cầu cũng như những giới hạn có điều kiện trên request đó.
- **Tiêu đề thực thể (entity headers)** gồm những quy tắc áp dụng lên phần nội dung (body) của request, nếu request không có nội dung thì phần tiêu đề cũng không có những tiêu đề thực thể này.

- **Nội dung (body)** được phân cách với phần tiêu đề bởi một dòng trống. Phần nội dung này có thể có hoặc không, đa phần nó hiện diện trong những request có phương thức `POST` được gửi kèm theo biểu mẫu dữ liệu HTML.

Thông báo phản hồi HTTP (HTTP response) được máy chủ trả về cho người dùng, là câu trả lời cho hành động đã được yêu cầu bởi người dùng thông qua HTTP request. HTTP response gồm các thành phần tương tự như HTTP request. Các dòng tiêu đề trên HTTP response cũng được phân làm 3 loại chính như Hình 2.3 sau.

[2]Nguồn: https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP



Hình 2.3: Các thành phần trong tiêu đề của một HTTP response[3]

2.2 Phương pháp kiểm thử xâm nhập

Kiểm thử xâm nhập (penetration testing) [13] là phương pháp kiểm tra bảo mật trong đó các kĩ sư cố gắng tìm ra, khai thác và đánh giá mức độ nghiêm trọng của các lỗ hổng bảo mật dựa trên những hiểu biết của họ về thiết kế lẫn đặc điểm hiện thực của một hệ thống ứng dụng cụ thể. Quy trình kiểm thử xâm nhập thường được tiến hành bởi đội đỏ (red team) của đội ngũ bảo mật của ứng dụng, hoặc bởi một tổ chức thứ ba chuyên về lĩnh vực này. Phương pháp này đặc biệt hiệu quả đối với các ứng dụng phức tạp gồm nhiều thành phần tổ hợp lại thành một hệ thông hoàn chỉnh như mô hình các ứng dụng web hiện nay. Đội ngũ kiểm thử sẽ xem xét toàn thể ứng dụng từ phía người dùng (client-side) và máy chủ (server-side), cố gắng can thiệp vào hoạt động của các thành phần nằm sâu trong back-end như các tập tin và tiến trình hệ thống, cơ sở dữ liệu, thậm chí cả việc kiểm thử bảo mật vật lý của hệ thống. Bằng cách tiến hành kiểm thử xâm nhập trong môi trường thực tế, đội ngũ bảo mật của hệ thống ứng dụng có thể vá lỗi và phòng chống những lỗ hổng bảo mật trong tương lai trước khi bị tin tặc khai thác.

Theo Weidman và Georgia [14], một cuộc kiểm thử xâm nhập hệ thống ứng dụng sẽ trải qua lần lượt các bước như sau.

1. **Thỏa thuận tiến hành** là giai đoạn đầu tiên trong quá trình kiểm thử xâm nhập. Bên dịch vụ kiểm thử sẽ thống nhất với khách hàng về mục tiêu, phạm vi, cách xử

[3]Nguồn: https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP

lí một số trường hợp đặc thù, thông tin liên hệ cũng như chi phí phát sinh trong quá trình kiểm thử trước khi tiến hành.

2. Tiếp theo là giai đoạn **thu thập thông tin**. Kĩ sư kiểm thử sẽ tận dụng những thông tin, công cụ sẵn có để tìm ra những cách tiềm năng để kết nối vào hệ thống mục tiêu.

3. Sau đó, quá trình kiểm thử bước vào giai đoạn **mô hình hóa nguy cơ bảo mật**. Dựa vào những thông tin thu thập được ở bước trên, kĩ sư kiểm thử sẽ phát triển những chiến lược, phương pháp cụ thể dưới tâm thế tấn công vào hệ thống mục tiêu.

4. Trước khi kĩ sư kiểm thử bắt đầu tấn công vào hệ thống, họ cần thực hiện quá trình **phân tích lỗ hổng**, tính toán khả năng thành công khi thực hiện một chiến lược tấn công nhất định. Một chiến lược tấn công sai lầm có thể làm gián đoạn hoạt động của một vài dịch vụ hoặc cả hệ thống, dẫn tới giảm tỉ lệ khai thác thành công lỗ hổng đã tìm thấy. Quá trình **khai thác** sẽ được tiến hành ngay sau đó.

5. Sau khi **khai thác** thành công, kĩ sư kiểm thử đã xâm nhập được vào một phần của hệ thống mục tiêu. Họ tiếp tục tiến hành giai đoạn **hậu khai thác** để cố gắng leo thang đặc quyền vào các thành phần, cơ sở dữ liệu, trang web quan trọng hơn của hệ thống, khai phá những trường hợp xấu nhất có thể xảy ra sau khi xâm nhập thành công.

6. Cuối cùng, họ sẽ **báo cáo kết quả** tổng quan và chi tiết với khách hàng và có thể đề xuất phương hướng khắc phục những lỗ hổng bảo mật đã tìm thấy trong quá trình kiểm thử.

Như vậy, thông qua quá trình tìm hiểu khái niệm và cách thức tiến hành phương pháp kiểm thử xâm nhập ở trên, chúng tôi nhận thấy phương pháp này khó có thể được dùng để phát triển một công cụ kiểm thử tự động hoặc bán tự động được.

2.3 Phương pháp fuzzing

Fuzzing [5] là một trong những phương pháp phổ biến nhất trong lĩnh vực kiểm thử bảo mật phần mềm nói chung và ứng dụng web nói riêng. Khái niệm fuzzing theo Bekrar,

Bekrar, Groz và Mounier [13] có nghĩa là đưa những dữ liệu đầu vào ngẫu nhiên hoặc không hợp lệ vào phần mềm để gây ra những hành vi bất thường, và xác định các lỗi cũng như khả năng tồn tại lỗ hổng bảo mật trong phần mềm. Mục đích của quan trọng nhất của fuzzing là gây lỗi hoặc làm sập (crash) hệ thống, mỗi một lỗi xuất hiện hoặc một lần hệ thống bị sập trong quá trình kiểm thủ là một dấu hiệu có lỗ hổng bảo mật tiềm ẩn. Sau đó những kĩ sư bảo mật sẽ phân tích các lỗi đó để tìm ra lý do và khai thác sâu hơn hoặc báo cáo lại với nhà cung cấp phần mềm (vendor) để vá lỗi.

Phương pháp fuzzing hữu dụng và phù hợp với việc kiểm thử hộp đen một phần vì tính tự động hóa cao của nó. Những mục tiêu kiểm thử trên phần mềm có thể là mọi "giao diện" có khả năng tương tác và nhận dữ liệu đầu vào từ người dùng. Mục tiêu kiểm thử của một phần mềm thường là những biến môi trường, nội dung và tên của những tập tin đang được sử dụng bởi phần mềm, những trường nhập dữ liệu, giao thức mà phần mềm dùng để giao tiếp với người dùng hoặc các phần mềm khác và các giao diện lập trình ứng dụng (application programming interface - API) được định nghĩa bởi phần mềm đó. Quá trình sinh ra trường hợp kiểm thử (testcase) cho các mục tiêu đó có thể được định nghĩa trước thành một danh sách, hoặc được biến đổi dựa trên một số trường hợp kiểm thử gốc hay mô hình sinh trường hợp nào đó, hoặc được sinh ra hoàn toàn ngẫu nhiên. Đồng thời quá trình phát hiện lỗi, hành vi bất thường hay sập hệ thống trong quá trình kiểm thử cũng dễ dàng quan sát được từ phía công cụ. Hai điều trên cho phép các công cụ fuzzing (gọi tắt là fuzzer) có một lợi thể to lớn là có thể vận hành hoàn toàn tự động với độ chính xác khá cao, từ bước tạo trường hợp kiểm thử, đến việc đưa từng trường hợp vào các mục tiêu kiểm thử và sau cùng là tự động phát hiện hành vi bất thường ở phía phần mềm.

Tính hiệu quả và độ "thông minh" của một fuzzer chủ yếu thể hiện qua cách thức sinh trường hợp kiểm thử, và một phần nhỏ cách thức phát hiện bất thường trên phần mềm. Dựa vào cách thức sinh trường hợp kiểm thử, Harper và các cộng sự [4] đã phân loại các fuzzer thành ba loại như sau.

- **Fuzzer đột biến (mutation fuzzer)** là nhóm các công cụ sinh trường hợp kiểm thử ngẫu nhiên đơn giản nhất. Các trường hợp kiểm thử được sinh ra bằng cách thay đổi ít hoặc nhiều (đột biến) các trường hợp kiểm thử gốc theo một quy luật nào đó, thường là hoàn toàn ngẫu nhiên.

- **Fuzzer tự sinh (generation fuzzer)** còn được gọi là fuzzer hộp trắng. Cách tiếp

cận này dựa vào sự am hiểu vào đối tượng kiểm thử của người phát triển công cụ. Các fuzzer tự sinh này không cần dựa trên các trường hợp kiểm thử mẫu như fuzzer đột biến, mà nó sẽ tự sinh ra trường hợp kiểm thử dựa trên các **mô hình dữ liệu** được định nghĩa trước. Các mô hình này thường được mô tả dưới dạng các tập tin cấu hình có cấu trúc tùy thuộc vào fuzzer sử dụng.

- **Fuzzer tiến hóa (evolutionary fuzzer)** có khả năng chọn ra một tập các trường hợp kiểm thử tốt nhất bằng cách tối ưu hóa mức độ che phủ mã nguồn (code coverage) của mục tiêu kiểm thử theo thời gian. Trong thực tế, công cụ sẽ chọn ra những cách thay đổi (mutate) trường hợp kiểm thử khiến nó chạm tới được những đoạn code sâu hơn trong chương trình hoặc gây lỗi khác với các trường hợp kiểm thử bình thường.

Trên thị trường hiện tại có nhiều fuzzer được thiết kế để sinh trường hợp kiểm thử hoặc thực hiện việc kiểm thử phần mềm nói chung và ứng dụng web nói riêng trên nhiều nền tảng, môi trường khác nhau. Một trong những fuzzer tốt nhất hiện tại là Radamsa[4] được phát triển bởi Aki Helin. Công cụ này có khả năng sinh các trường hợp kiểm thử theo định nghĩa và mục đích của người dùng nên do đó, ta có thể áp dụng những trường hợp kiểm thử sinh được trong bất kì phần mềm nào nói chung và các ứng dụng web nói riêng. Ngoài ra còn có thể kể đến KameleonFuzz [3], một fuzzer tiến hóa được thiết kế để kiểm thử hộp đen lỗ hổng XSS (Cross-Site Scripting) trên các ứng dụng web. Công cụ này không chỉ tự sinh trường hợp kiểm thử tốt hơn theo thời gian mà còn chỉ ra được trường hợp kiểm thử hiện tại đã gần khai thác được lỗ hổng đến mức nào. Phương pháp này fuzzing khá phù hợp để hiện thực một ứng dụng kiểm thử tự động hoặc bán tự động khi đã có sẵn đối tượng và trường hợp kiểm thử.

[4]Nguồn: https://gitlab.com/akihe/radamsa

Chương 3

Tổng quan về một số lớp lỗ hổng bảo mật trên ứng dụng web

Trong phạm vi hiện thực công cụ, tôi chỉ tập trung vào việc tìm hiểu và phát hiện hai lỗ hổng bảo mật ở tầng ứng dụng của ứng dụng web, Local File Inclusion (LFI) và time-based Structured Query Language Injection (SQLI). Đầu tiên, tôi sẽ trình bày về Local File Inclusion (LFI), một trong những lỗ hổng bảo mật phổ biến nhất trên ứng dụng web, luôn góp mặt trong top 10 lỗ hổng nguy hiểm nhất theo OWASP [10] nhiều năm gần đây. Nội dung phần này bao gồm làm rõ khái niệm LFI và phân tích các kĩ thuật khai thác lỗ hổng bảo mật này.

3.1 Local File Inclusion

Lỗ hổng Local File Inclusion (LFI) [8, 12], còn có tên khác là Directory Traversal, là lỗ hổng bảo mật trên ứng dụng web cho phép tin tặc đọc một vài tập tin nhạy cảm trên máy chủ mà ứng dụng web đang chạy trên. Những tập tin này có thể là mã nguồn, hình ảnh, dữ liệu hoặc thậm chí cả những tập tin hệ thống nhạy cảm. Trong một vài trường hợp xử lí phân quyền không tốt ở phía máy chủ, tin tặc còn có thể tạo thêm tập tin mới gây nên sự thay đổi dữ liệu và hành vi của ứng dụng, hoặc hắn có thể thực thi mã (RCE) từ phía máy chủ để chiếm quyền điều khiển hệ thống. Đoạn mã 3.1 dưới đây mô tả một điểm cuối của trang web "`http://vuln-web.com/`" có lỗ hổng LFI.

```
1 /**
```

```
* Get the filename from a GET input
* Example - http://vuln-web.com/?file=filename.php
*/
$file = $_GET['file'];

/**
* Unsafely including the file
* Example - filename.php
*/
include('directory/' . $file);
```

Đoạn mã 3.1: Đoạn mã PHP có lỗ hổng Local File Inclusion

Bằng cách nhận vào tham số `$file` từ đường dẫn URL, máy chủ web trực tiếp trả về nội dung tập tin có tên `filename.php` trong thư mục hiện tại như trong ví dụ trên. Tin tặc có thể lợi dụng sơ hở từ câu lệnh `include('directory/' . $file);` để trực tiếp đọc bất kì tập tin nào bằng cách sử đường dẫn tương đối, kết hợp tiền tố "../" (hoặc "..\" đối với các máy chủ web dùng hệ điều hành Windows Server) để tham chiếu tới thư mục cha của thư mục hiện tại cộng với tên tập tin để đọc nội dung của tập tin đó, ví dụ như "`http://vuln-web.com/?file=../../etc/passwd`". Đoạn mã 3.2 sau đây minh họa một số vector tấn công (payload) mà chúng tôi sẽ sử dụng trong ứng dụng webfuzzer để đọc tập tin "**passwd**" trong đường dẫn **/etc/** trong các máy chủ web Unix.

```
etc%c0%afpasswd
etc%c0%afpasswd%00
/etc/passwd
/etc/passwd%00
..//etc/passwd
../..//etc/passwd
..%2f/etc/passwd
..%2f..%2f/etc/passwd
%2e%2e//etc/passwd
%2e%2e/%2e%2e//etc/passwd
..%252f/etc/passwd
```

```
12 ..%252f..%252f/etc/passwd
13 %252e%252e%252f/etc/passwd
14 %252e%252e%252f%252e%252e%252f/etc/passwd
15 ..\/etc/passwd
16 ..\..\/etc/passwd
17 %252e%252e\/etc/passwd
18 %252e%252e\%252e%252e\/etc/passwd..%5c/etc/passwd
19 ..%5c..%5c/etc/passwd
20 ..%5c..%5c..%5c/etc/passwd
```

Đoạn mã 3.2: Một số payload khai thác lỗ hổng LFI

/etc/passwd là tập tin chứa danh sách tài khoản người dùng trên máy chủ web dùng hệ điều hành Unix, tập tin này bắt đầu bằng chuỗi "`root:x`" và chứa nhiều thông tin quan trọng như tên, địa chỉ email, đường dẫn tuyệt đối đến thư mục home trên đĩa cứng, ID của quyền (hoặc nhóm quyền) của người dùng đó,..., cung cấp cho tin tặc nhiều thông tin quý giá khi bắt đầu xâm nhập vào hệ thống. Do ta không biết độ sâu của thư mục ứng dụng web so với đường dẫn gốc (root directory) của hệ điều hành nên ta phải thử truy vấn đến từng tầng thư mục cha của thư mục hiện hành đến khi chạm tới thư mục gốc sau đó truy cập đến tập tin **/etc/passwd** như ví dụ trong Đoạn mã 3.2.

3.2 Time-based Structured Query Language Injection

Hầu hết các kĩ thuật tấn công ứng dụng web đều tuân theo mô-típ đánh lừa ứng dụng web để thực hiện những hành động nguy hiểm thay tin tặc. Tin tặc không thể lấy được cookies của người dùng một cách chính quy nhưng có thể đánh lừa để trình duyệt web nạn nhân cung cấp cho hắn thông qua lỗ hổng Cross-Site Scripting (XSS). Tin tặc cũng không thể tùy tiện truy xuất bất cứ tập tin nào trên máy chủ web nhưng hắn có thể lừa ứng dụng web làm việc đó thông qua lỗ hổng LFI. Tương tự, tin tặc không thể trực tiếp truy cập và trích xuất cơ sở dữ liệu của ứng dụng web nhưng hắn có thể lừa ứng dụng web làm việc đó thông qua lỗ hổng Structured Query Language Injection (SQLI). Phần này trình bày về khái niệm lỗ hổng SQLI nói chung và time-based SQLI nói riêng.

Lỗ hổng Structured Query Language Injection (SQLI) [6, 12] cho phép tin tặc can thiệp vào các câu truy vấn SQL mà ứng dụng web dùng để tương tác với cơ sở dữ liệu. Thông thường việc này sẽ cho phép tin tặc đọc những dữ liệu mà hắn không có quyền, ví dụ như dữ liệu của những người dùng khác trong ứng dụng hoặc kể cả những dữ liệu mà ứng dụng đó không có quyền truy cập. Trong nhiều trường hợp tin tặc còn có thể sửa đổi hoặc sao chép (dump) những dữ liệu này, dẫn đến rủi ro bị đánh cắp dữ liệu và những hậu quả lâu dài cho nhà cung cấp ứng dụng web. Trong một vài điều kiện đặc biệt hơn, tin tặc còn có thể leo thang một cuộc tấn công SQLI để thỏa hiệp máy chủ ứng dụng web đang vận hành hoặc thực hiện một cuộc tấn công từ chối dịch vụ (DoS - Denial of Service). Đoạn mã 3.3 sau đây[1] là một ví dụ.

```
1  public function doQuery($injection) {
2      $pdo = new SQLite3('database.db', SQLITE3_OPEN_READONLY);
3
4      $query = 'SELECT id,username FROM users WHERE id=' . $injection . '
       LIMIT 1';
5      $getUsers = $pdo->query($query);
6      $users = $getUsers->fetchArray(SQLITE3_ASSOC);
7
8      if ($users) {
9          return $users;
10     }
11
12     return false;
13 }
```

Đoạn mã 3.3: Đoạn mã PHP có lỗ hổng SQL Injection

Câu truy vấn SQL trong ví dụ trên sẽ trả về cho người dùng giá trị của hai trường `id` và `username` của bản ghi đầu tiền trong bảng `users` có trường `id` bằng giá trị của tham số `$injection` được truyền vào. Thông thường giá trị này thường là những chuỗi hoặc những con số có định dạng cụ thể, phục vụ cho việc truy vấn thông tin người dùng. Thông qua giao diện web hoặc HTTP request, tin tặc có thể can thiệp vào ứng dụng web và

[1]Nguồn: http://websec.fr/level01/source.php

chỉnh sửa tùy ý nội dung của tham số `$injection`. Ví dụ trong trường hợp giá trị tham số này là "`1'; DROP TABLE users; --'`", câu truy vấn trở thành

```
SELECT id,username FROM users WHERE id='1'; DROP TABLE users; --'
```

Sau khi truy vấn giá trị `id` và `username` của bản ghi có trường `id = 1` thì câu truy vấn SQL trên sẽ xóa toàn bộ nội dung (drop) bảng users trong cơ sở dữ liệu của ứng dụng. Bằng cách thực hiện nhiều câu truy vấn nối tiếp hoặc lồng nhau (nested queries), đóng mở nháy đơn, sử dụng chú thích (--) phù hợp với mã nguồn của trang web mà tin tặc có thể khai thác lỗ hổng SQLI này và gây ra hậu quả to lớn như thay đổi cấu trúc lược đồ quan hệ như ví dụ trên. Hơn nữa lỗ hổng này còn có thể được tin tặc tận dụng để leo thang đặc quyền (privilege escalation) trên máy chủ ứng dụng web mục tiêu, thực thi được những câu lệnh nhạy cảm trên hệ thống, dẫn đến lỗ hổng RCE.

Lớp lỗ hổng SQLI có nhiều loại được phân ra dựa trên cách thức và mục tiêu tấn công [2], trong phạm vi luận văn này, tôi chỉ tập trung phát hiện lỗ hổng time-based SQLI. Time-based SQLI là kĩ thuật khai thác lỗ hổng SQLI bằng cách sử dụng những câu truy vấn đặc biệt, buộc hệ quản trị cơ sở dữ liệu phải chờ một khoảng thời gian trước khi trả về kết quả truy vấn cho ứng dụng, dựa vào thời gian phản hồi của câu truy vấn, ta có thể nhận định sự tồn tại của lỗ hổng này. Mục đích của việc kiểm thử lỗ hổng này không chỉ có vậy, việc khai thác thành công time-based SQLI trên những điểm cuối của một ứng dụng web còn có nghĩa là điểm cuối này cho phép truy vấn liên tiếp, lồng nhau, hoặc phát hiện được một số phần tử trong danh sách đen cũng như danh sách trắng của bộ lọc dữ liệu đầu vào, những nhận định này là cơ sở để tấn công sâu hơn vào ứng dụng web đó. Việc khai thác bằng kĩ thuật này phụ thuộc các hàm hoãn thời gian của mỗi hệ quản trị cơ sở dữ liệu khác nhau, cụ thể đối với MySQL ta dựa vào hàm `sleep` và `benchmark`, đối với Microsoft SQL Server là hàm `waitfor delay` và `waitfor time`, đối với Postgres SQL là `pg_sleep`,...

Chương 4

Xu hướng tấn công ứng dụng web hiện nay và hướng phát triển ứng dụng webfuzzer

4.1 Xu hướng tấn công ứng dụng web hiện nay

Bảo mật luôn là vấn đề hàng đầu trong nhiều lĩnh vực khác nhau của cuộc sống, đặc biệt trong lĩnh vực phần mềm máy tính/di động và ứng dụng web trong thời đại cách mạng công nghiệp 4.0 hiện nay. Những ứng dụng quan trọng liên quan đến tài chính, thông tin cá nhân của người dùng cũng đang được cung cấp rộng rãi bởi các ngân hàng, ví điện tử, mạng xã hội,... Tất cả điều đó hấp dẫn rất nhiều tin tặc và tổ chức tội phạm mạng, dẫn đến nguy cơ bị tấn công của các phần mềm nói chung và các ứng dụng web nói riêng. Đối với ứng dụng web, ngoài việc khai thác các lỗ hổng bảo mật ở tầng mạng như tấn công từ chối dịch vụ (denial of service - DoS), các phương pháp giả mạo bộ nhớ đệm (DNS/HTTP web server cache poisoning), chiếm tên miền phụ (subdomain takeover),..., tầng ứng dụng (đã được trình bày ở **Chương 3**) thì việc tấn công các ứng dụng web được lưu trữ ảo chung (virtual hosting) và khai thác những lỗ hổng 0-day và n-day mới của các công cụ, khung thức do bên thứ ba cung cấp cho ứng dụng web đang trở thành xu hướng.

Hiện nay, việc khai thác các lỗ hổng bảo mật trên một ứng dụng web còn có thể ảnh hưởng đến nhiều ứng dụng web khác thông qua sự phổ biến của các dịch vụ lưu trữ ảo.

Lưu trữ ảo là một phương pháp cho phép lưu trữ (hosting) nhiều tên miền ứng dụng web trên một máy chủ vật lý. Việc sử dụng hosting ảo giúp tiết kiệm chi phí vận hành, tiết kiệm quỹ địa chỉ IPv4 sẵn có cũng như tạo điều kiện cho các dịch vụ hosting phát triển mạnh mẽ, cho phép họ dễ dàng cung cấp dịch vụ, quản lý việc hosting nhiều ứng dụng web khác nhau trên cùng một máy chủ vật lý, không cần phải đầu tư nhiều về phần cứng lẫn băng thông. Những máy chủ web thường dùng để triển khai hosting ảo là **Nginx**, **Apache**, hoạt động tốt trên các phiên bản của hai hệ điều hành máy chủ phổ biến nhất hiện nay là **Ubuntu Server** và **Windows Server**. Hai máy chủ web này lưu giữ mã nguồn và nội dung của các ứng dụng web trong quá trình vận hành dưới dạng các thư mục con (đối với **Apache**) hoặc các khối máy chủ (Nginx server block - đối với **Nginx**) tương ứng với các ứng dụng web riêng biệt, đồng thời cung cấp nhiều tính năng quản lý hosting, cải thiện hiệu năng, đảm bảo bảo mật cho các ứng dụng web trong cụm hosting ảo. Các tính năng trên thường xuyên được cải tiến, nâng cấp và bản thân **Apache** cũng như **Nginx** có chương trình trả thưởng cho các lỗ hổng bảo mật của họ (bug bounty program) rất hậu hĩnh, chúng ta có thể tạm yên tâm về khía cạnh bảo mật của các máy chủ web này nếu được cấú hình chuẩn bởi nhà cung cấp dịch vụ. Tuy nhiên việc sử dụng hosting ảo cũng tồn tại một số nhược điểm.

- Sự tập trung nhiều ứng dụng web lại một chỗ gây ra rủi ro sụp đổ tại một điểm (single point of failure). Khi có sự cố về phần cứng máy chủ web hoặc một ứng dụng web trong cụm bị tấn công từ chối dịch vụ, khả năng đáp ứng người dùng của những ứng dụng web khác sử dụng chung dịch vụ hosting ảo cũng sẽ bị ảnh hưởng, do sử dụng chung máy chủ vật lý và băng thông.

- Các rủi ro bảo mật cho các ứng dụng web dùng chung hosting chủ yếu đến từ chính các ứng dụng web trong cụm chứ không từ máy chủ web như **Nginx** hay **Apache**, và việc đảm bảo bảo mật cho mỗi ứng dụng web tham gia hosting chung phải do chính chủ sở hữu của từng ứng dụng web trực tiếp quản lý. Tuy dịch vụ hosting ảo có thể hỗ trợ việc này một phần, thông qua một số tính năng/bộ lọc giúp kiểm soát, giới hạn quyền truy cập, bảo mật cookies cũng như sao lưu và phục hồi dữ liệu của **Nginx** và **Apache**, nhưng về mặt lý thuyết, một ứng dụng web trong cụm hosting ảo bị xâm nhập thành công sẽ đồng nghĩa với việc tất cả các ứng dụng web còn lại đều có nguy cơ bị xâm nhập. Điều này có được là do tin tặc hoàn toàn có khả năng

truy xuất nội dung (bên trong các thư mục con, khối máy chủ trong hệ thống) và can thiệp vào các dịch vụ vận hành của các ứng dụng web khác.

Hiện nay, các dịch vụ tường lửa ứng dụng web (web application firewalls - WAF) ngày càng được hoàn thiện, nhiều quy chuẩn (best practices) được đề ra trong cộng đồng lập trình viên và áp dụng rộng rãi trong nhiều ngôn ngữ lập trình và khung thức, công cụ nhằm chống lại việc khai thác những lỗ hổng bảo mật thường gặp của ứng dụng web (như DoS, SQLI, XSS, remote code execution - RCE,...). Một vài ví dụ tiêu biểu có thể kể đến như các dịch vụ tường lửa ModSecurity, CloudFlare, Citrix, AppTrana,..., các thư viện mã hóa và làm sạch dữ liệu đầu vào (input sanitizer) trên các ngôn ngữ lập trình phổ biến như PHP (HTML Purifier[1], PHP Intrusion Detection System, PHP Password Lib, PHPSecLib), NodeJS (awesome-nodejs-security, thư viện Helmet của express framework), ReactJS (dompurify, xss, xss-filters), ESAPI[2], AntiXSS[3],... Trong bối cảnh đề cập ở trên, một trong những hướng tấn công đáng để thử cho những tin tặc là điều tra và khai thác các ứng dụng web có lỗi trong những cụm hosting ảo này. Việc này tuy có thể mất nhiều thời gian công sức hơn tìm và khai thác các lỗ hổng bảo mật trên ứng dụng web mục tiêu của ta nhưng nếu thành công thì hậu quả để lại khó có thể đong đếm được do có rất nhiều trang web của những tổ chức lớn đang sử dụng dịch vụ hosting ảo. Và thông thường, những ứng dụng web quan trọng của một tổ chức nào đó (ví dụ như BKPay hay ứng dụng quản lý điểm số, thông tin cá nhân của sinh viên và cán bộ công nhân viên trong trường,...) thường sẽ được hosting chung để tiện trong công tác quản lý, nâng cấp, bảo mật và bảo trì thường xuyên. Giả sử như một ứng dụng web có lỗ hổng bảo mật và bị khai thác thì các ứng dụng web quan trọng khác cũng chung hosting sẽ ngay lập tức có nguy cơ cực kì cao sẽ bị thỏa hiệp, dẫn đến những mất mát to lớn về tài chính, về sự vận hành của cả tổ chức cũng như lộ thông tin cá nhân của rất nhiều người.

Một hướng tấn công đặc biệt hiệu quả hiện nay là khai thác những lỗ hổng bảo mật 0-day và n-day. Lỗ hổng 0-day là các lỗ hổng bảo mật trên phần mềm, phần cứng hoặc firmware mà nhà cung cấp các sản phẩm này chưa biết để vá lỗi. Thuật ngữ 0-day có thể được hiểu là chính bản thân lỗ hổng hoặc những cuộc tấn công đầu tiên sử dụng lỗ hổng đó. Sau khi một cuộc tấn công 0-day đã được công khai, lỗ hổng được nhà cung

[1]Nguồn: http://htmlpurifier.org/
[2]Nguồn: https://www.owasp.org/index.php/Category:OWASP_Enterprise_Security_API
[3]Nguồn: https://documentation.help/Microsoft-AntiXSS/

cấp thừa nhận và đang tiến hành vá lỗi, các tin tặc có thể viết mã khai thác để tấn công các hệ thống/ứng dụng sử dụng các sản phẩm chưa được vá lỗi đó. Thuật ngữ 1-day hay n-day dùng để chỉ ra thời gian (số ngày) giữa lúc cuộc tấn công diễn ra so với thời điểm nhà cung cấp công bố lỗ hổng hoặc để ám chỉ những lỗ hổng 0-day đã biết nhưng chưa có mã khai thác trên những môi trường khác nhau (ví dụ lỗ hổng A đã được khai thác thành công trên trình duyệt web Firefox trên hệ điều hành Windows nhưng chưa có mã khai thác lỗi A trên Ubuntu hay Android). Trách nhiệm của nhà cung cấp là công bố lỗ hổng ngay sau thời gian quy định (ví dụ như 90 ngày sau khi lỗ hổng được báo cáo đối với Project Google Zero) hoặc ngay khi có bản vá lỗi để cảnh báo các cá nhân và doanh nghiệp đang sử dụng sản phẩm của mình có động thái cập nhật bản vá hoặc tự vệ nếu lỗ hổng vẫn chưa được vá. Đồng thời các nhà cung cấp cùng các nhà phân phối và quản trị viên dưới quyền có trách nhiệm to lớn phải vá lỗ hổng n-day đó càng nhanh càng tốt để giảm thiểu thiệt hại không đáng có cho người dùng. Khái niệm lỗ hổng bảo mật và phơi bày thông tin thường gặp (common vulnerabilities and exposures - CVE) và liệt kê điểm yếu thường gặp của phần mềm (common weakness enumuration - CWE) cũng được hình thành từ quá trình phát hiện - tấn công - vá lỗi liên tục này. CVE là một danh sách các lỗ hổng bảo mật được đặt mã hiệu và định danh của từng lỗ hổng kèm theo ít nhất một tài liệu tham khảo liên quan về lỗ hổng đó đã được công bố công khai. Danh sách này cũng như trang web tra cứu các CVE[4] và CWE[5] được duy trì để phục vụ cho cộng đồng bởi MITRE Corporation. Mục đích của CVE là để hệ thống các lỗ hổng, thuận tiện cho việc chia sẻ thông tin cũng như điều tra về các lỗ hổng có liên quan trong quá khứ của một nhà cung cấp nào đó.

Lỗ hổng bảo mật 0-day có thể xuất hiện ở bất cứ phần mềm, hệ thống, công cụ, khung thức nào, từ phía người dùng lẫn phía máy chủ ứng dụng web. Những lỗ hổng lớn gây nhiều thiệt hại thường nằm ở các trình duyệt web phổ biến như Google Chrome, Firefox, Safari; các khung thức và hệ quản trị nội dung (content management system - CMS) như Telerik, WordPress, Joomla, Wix, Shopify,...; các máy chủ web và dịch vụ lưu trữ, điện toán đám mây như Nginx, Apache, Microsoft Azure, Amazon Web Service,... Bằng việc sử dụng các lỗ hổng n-day hoặc các CVE liên quan đến các sản phẩm trên, tin tặc có thể tổ chức tấn công hàng loạt các cá nhân/doanh nghiệp sử dụng các sản phẩm đó với

[4]Nguồn: https://www.cvedetails.com/
[5]Nguồn: http://cwe.mitre.org/

phiên bản thích hợp, chưa cập nhật bản vá. Khả năng thành công của dạng tấn công này gần như là tuyệt đối trong trường hợp sản phẩm đang được vận hành có cùng phiên bản và môi trường mà sản phẩm đó đang chạy trên, dẫn đến những vụ lộ thông tin cá nhân, thay đổi giao diện (deface), chiếm tên miền phụ,... của một số lượng cực lớn các trang web. Tiêu biểu có thể kể đến một loạt 3 CVE rất nguy hiểm của Telerik Web UI[6] trong năm 2017 cho phép tin tặc tải lên bất kì tệp độc hại nào cũng như tự do thực thi mã trên các ứng dụng web sử dụng các phiên bản Teleril Web UI cũ, hay 10 CVE của Wordpress[7] trong năm 2019 thông qua việc khai thác lỗ hổng XSS và RCE.

Hai hướng tấn công phổ biến đã trình bày ở trên ngày càng tỏ ra hiệu quả và phổ biến hiện nay, một trong những điểm chung của hai cách thức trên là rất khó để có thể tự động hóa bởi công cụ. Không như các phương pháp quét hoặc thử sai khác, việc điều tra hosting ảo cũng như viết mã khai thác các CVE trên nhiều môi trường khác nhau phải được thực hiện bằng tay. Một điểm chung nữa là việc tấn công các ứng dụng web kém bảo mật trong cụm hosting ảo cũng như việc hình thành các CVE phần lớn đều bắt nguồn từ các lỗ hổng bảo mật ở tầng ứng dụng (đã được trình bày ở **Chương 3**) rồi được tổng quát hóa, định danh và phân loại theo dạng lỗ hổng và nhà cung cấp sản phẩm tương ứng. Các lỗ hổng đó là cơ sở cho sự xuất hiện của rất nhiều vấn đề về bảo mật trên ứng dụng web, đồng thời các những phương pháp tự động hóa trong việc kiểm thử các lỗ hổng đó cũng đã có sẵn và ngày càng được hoàn thiện hơn bởi sự đóng góp của cộng đồng. Từ hai điểm chung và những phân tích trên, chúng tôi nhận thấy một trong những hướng hiệu quả và cấp thiết để hiện thực công cụ kiểm thử các ứng dụng web theo đề tài luận văn là tập trung phát hiện các lỗ hổng bảo mật ở tầng ứng dụng với khả năng tự động hóa càng cao càng tốt.

4.2 Phạm vi và hướng phát triển ứng dụng webfuzzer

Trong phạm vi luận văn tốt nghiệp này, tôi sẽ áp dụng phương pháp kiểm thử fuzzing dưới dạng kiểm thử hộp đen để hiện thực công cụ **webfuzzer**. Ngoài khả năng tự động hóa cao, việc áp dụng phương pháp này sẽ đem lại nhiều lợi ích khác.

- Linh hoạt trong việc kiểm thử nhiều ứng dụng web sử dụng các công nghệ, mô hình,

[6]Nguồn: https://www.cvedetails.com/vulnerability-list/vendor_id-14130/Telerik.html
[7]Nguồn: https://www.cvedetails.com/vulnerability-list.php?vendor_id=2337

khung thức phát triển phần mềm khác nhau, cũng như cho phép kiểm thử số lượng lớn ứng dụng web trong một khoảng thời gian ngắn.

- Đơn giản hóa và chuẩn hóa việc kiểm thử các lỗ hổng bảo mật trên những ứng dụng web có thiết kế cao cấp và phức tạp. Những ứng dụng này thường có số lượng dòng code có thể lên đến hàng nghìn thậm chí hàng triệu dòng, việc kiểm thử mã nguồn đối với phương pháp hộp trắng hoặc hộp xám sẽ khó khăn hơn nhiều.

- Dễ dàng đạt được khả năng tự động hóa cao so với việc quét mã nguồn của ứng dụng. Nhìn chung quá trình phát triển công cụ bằng phương pháp này không yêu cầu kĩ năng lập trình cao hoặc những kĩ thuật nhận diện lỗ hổng quá đặc thù.

- Chủ yếu tập trung vào việc kiểm thử chức năng của ứng dụng, không quan tâm đến giao diện, hiệu năng hay trải nghiệm người dùng. Điều này giúp ta có mục tiêu rõ ràng và tiết kiệm công sức cho quá trình thiết kế, cải tiến các trường hợp kiểm thử.

Tương tự như trong lĩnh vực kiểm thử phần mềm, việc áp dụng phương pháp kiểm thử fuzzing dưới dạng kiểm thử hộp đen trong quá trình kiểm thử bảo mật ứng dụng web cũng sẽ có một số hạn chế như sau.

- Do không nắm được mã nguồn của ứng dụng web nên chiến thuật tốt nhất là ta chỉ nên tập trung kiểm tra những vị trí thường phát sinh lỗ hổng nhất, hoặc dò quét và vượt qua (bypass) cách thức phòng thủ của ứng dụng bằng việc thử sai và làm rối (tampering) dữ liệu kiểm thử.

- Số lượng trường hợp kiểm thử kết hợp với các kĩ thuật bypass đã biết là quá lớn, yêu cầu nhiều kinh nghiệm thực tế của người lập trình để chọn lọc và viết ra được một công cụ nhanh và ổn định, tối ưu hóa tài nguyên máy tính và thời gian thực thi. Ta buộc phải đánh đổi thời gian và tài nguyên đó hoặc chấp nhận kiểm thử trên một tập dữ liệu nhỏ hơn chỉ chứa những trường hợp thường gặp nhất.

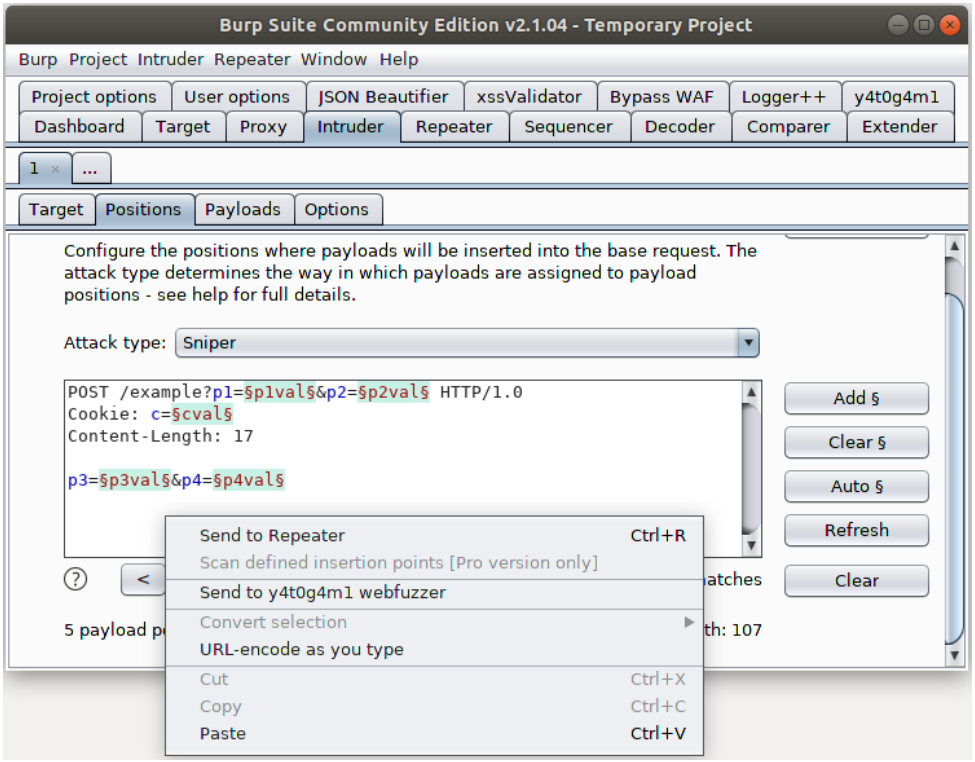
- Lập trình viên cần thường xuyên cập nhật các trường hợp kiểm thử và kĩ thuật tấn công/phòng thủ mới để bổ sung vào công cụ.

Hơn nữa, vấn đề đạo đức nghề nghiệp phải luôn luôn được đặt lên trên hết. Trước và trong quá trình kiểm thử ta phải có thỏa thuận với nhà cung cấp ứng dụng web về mục đích và cách thức tiến hành của mình, hoặc, xác định rõ động cơ của bản thân là để phát

hiện những lỗ hổng nguy hiểm và sẽ báo cáo lại với nhà cung cấp để họ vá lỗi, tăng cường bảo mật và bảo vệ quyền lợi của người dùng ứng dụng đó. Bên cạnh đó ta cũng cần phải lưu ý về vấn đề pháp lý của mỗi vùng lãnh thổ địa lý hoặc quy định của những nhà cung cấp khác nhau, cũng như trang bị một số hiểu biết nhất định để bảo vệ danh tính và sự riêng tư của bản thân khi tiến hành kiểm thử số lượng lớn các ứng dụng web xuyên quốc gia.

Trong quá trình tìm hiểu và so sánh giữa các phương pháp, công cụ kiểm thử bảo mật ứng dụng web hiện tại, chúng tôi nhận thấy sử dụng chức năng `Intruder` của Burp Suite là phương pháp phù hợp nhất để tự động hóa quá trình phát hiện các lỗ hổng bảo mật ở tầng ứng dụng của các ứng dụng web thông qua thông điệp HTTP. `Intruder` là chức năng triển khai tấn công ứng dụng web tự động, mạnh mẽ và dễ cấu hình. Chức năng này có thể được sử dụng thực hiện những cuộc tấn công với phạm vi trải dài từ đơn giản tới phức tạp, từ vét cạn (brute-force) mật khẩu, thư mục ứng dụng web đến phát hiện các lỗ hổng bảo mật web ở tầng ứng dụng bằng phương pháp fuzzing. Bên cạnh đó, chúng tôi cũng sử dụng thêm chức năng `Extender`[8] của Burp Suite để hỗ trợ gửi đối tượng kiểm thử (request mẫu) kèm theo các vị trí cần chèn payload đến backend của webfuzzer. Hình 4.1 sau mô tả giao diện chính của chức năng `Intruder`.

[8] `Extender` là một tính năng của Burp Suite, cung cấp thư viện lập trình và cho phép người dùng tích hợp thêm các phần mở rộng (extension) viết bằng Java hoặc Python vào công cụ này



Hình 4.1: Giao diện chính của chức năng Intruder

Giao diện chính của chức năng này gồm nhiều tab liền kề, mỗi tab được đánh số thứ tự, ứng với một HTTP request mẫu (base request) cần kiểm thử. Mỗi tab này gồm 4 tab phụ được mô tả như sau.

- **Target** chứa thiết lập địa chỉ IP:port của ứng dụng web mục tiêu.
- **Positions** chứa những thiết lập của request mẫu cần kiểm thử, bao gồm các vị trí truyền tham số và kiểu tấn công.
- **Payloads** chứa các thiết lập liên quan đến tập vector tấn công (payload) kiểm thử. Tập payload này có thể được nhập từ tập tin trên máy, hoặc được sinh ra bằng một số phần mở rộng khác của Burp Suite, kèm theo các phương pháp tiền xử lí và encode payload thường dùng.
- **Options** chứa những thiết lập liên quan khác trong quá trình tấn công như thời gian chờ, các chuỗi và biểu thức chính quy để so trùng trên HTTP response trả về,...

Nhìn chung, chức năng này sử dụng cặp kí tự "§" để đánh dấu vị trí thay thế payload vào request mẫu để tạo thành requets kiểm thử, cụ thể việc sử dụng các danh sách payload và

cách thức thay thế payload vào request mẫu như thế nào phụ thuộc vào kiểu tấn công của chức năng `Intruder`, gồm có `Sniper, Battering ram, Pitchfork` và `Cluster bomb`. Kiểu tấn công chúng tôi thường sử dụng nhất trong quá trình fuzzing bằng chức năng này là `Sniper`. Kiểu tấn công này chèn lần lượt từng payload trong danh sách vào từng vị trí thay thế payload trong request mẫu, do đó tổng số request kiểm thử được hình thành bằng tích số lượng payload nhân với số vị trí chèn payload trong request mẫu. Chúng tôi định ra hướng phát triển ứng dụng webfuzzer, hiện thực lại gần như đầy đủ chức năng `Intruder`, sử dụng kiểu tấn công `Sniper` vì những khuyết điểm của ứng dụng Burp Suite như sau.

- Việc kiểm thử một request mẫu với nhiều loại lỗ hổng khá phiền phức, người dùng phải lần lượt sao chép các cấu hình kiểm thử cần thiết vào tab chứa request mẫu. Do đó, nhu cầu định ra một cấu hình kiểm thử mặc định rồi dùng để kiểm thử các request mẫu với nhiều lỗ hổng, cộng với một giao diện trực quan, thân thiện với người dùng hơn, là cần thiết.

- Burp Suite không tự động thực thi yêu cầu kiểm thử tiếp theo sau khi hoàn thành yêu cầu hiện tại. Do đó việc xây dựng một ứng dụng kiểm thử xoay quanh các yêu cầu kiểm thử là cần thiết. Việc này giúp quản lý và phân phối tài nguyên máy tính hợp lý hơn, người dùng có thể kiểm soát một lúc có bao nhiêu yêu cầu kiểm thử đang thực thi, tự động thực thi những yêu cầu kiểm thử đã tạo trong thời gian nhàn rỗi, đồng thời cắt giảm thời gian thực hiện các thao tác lặp đi lặp lại của họ so với sử dụng Burp Suite.

- Ứng dụng có thể dễ dàng được triển khai ứng dụng kiểm thử ở các dịch vụ cung cấp máy chủ ảo hóa (virtual private service - VPS. Việc kiểm thử bằng Burp Suite trên máy tính cá nhân (đặc biệt là tính năng `Intruder`) tiêu tốn khá nhiều tài nguyên máy tính, cản trở công việc, dẫn đến nhu cầu tách riêng chức năng này để triển khai trên một máy chủ độc lập. Việc này cũng giúp tránh được nguy cơ địa chỉ IP của máy tính cá nhân bị block theo vùng địa lý hoặc bởi các chính sách kiểm soát lưu lượng truy cập bởi ứng dụng web mục tiêu do gửi quá nhiều request đến trong một khoảng thời gian ngắn, giúp người kỹ sư kịp thời định ra phương hướng khác để tiếp cận mục tiêu.

- Việc lưu trữ kết quả kiểm thử ở Burp Suite cần phải được làm bằng tay, dẫn đến

nhu cầu cần có một cơ sở dữ liệu để lưu trữ, hệ thống hóa những điểm cuối ứng dụng web có lỗ hổng kèm theo bằng chứng khai thác được lỗi trong quá trình kiểm thử tự động. Việc xây dựng một cơ sở dữ liệu như webfuzzer cũng là tiền đề để mở rộng ứng dụng trên lĩnh vực do thám (reconnaissance) các ứng dụng web mục tiêu trong tương lai.

- Việc xây dựng một ứng dụng web để kiểm thử thay vì sử dụng Burp Suite ở máy tính cá nhân còn giải quyết được nhu cầu làm việc chung và chia sẻ dữ liệu kiểm thử của nhiều người trong cộng đồng, hay hẹp hơn là tổ chức, công ty, thay vì chỉ một cá nhân đơn lẻ. Nhiều người có thể trực tiếp đóng góp vào cơ sở dữ liệu và truy vấn thông tin bằng cách sử dụng ứng dụng webfuzzer thay vì Burp Suite ở máy tính cá nhân.

Chúng tôi hiện thực ứng dụng webfuzzer để khắc phục các khuyết điểm trên của Burp Suite. Cụ thể về phạm vi, chúng tôi chỉ hiện thực chức năng so trùng trên chuỗi, biểu thức chính quy và kiểm tra thời gian phản hồi request của ứng dụng web mục tiêu. Chúng tôi không hiện thực chức năng nâng cao kiểm thử đa luồng (multithread) như Burp Suite vì dễ bị chặn do vượt hạn mức request ở các ứng dụng web mục tiêu. Chúng tôi cũng không hiện thực chức năng encode đối với payload và URL vì payload kiểm thử đã được encode và áp dụng các hình thức làm rối và bypass sẵn.

Chương 5

Yêu cầu hiện thực và thiết kế hệ thống đề xuất của ứng dụng webfuzzer

Chương này trình bày các yêu cầu cần đạt trong quá trình thiết kế - hiện thực và chi tiết kiến trúc của ứng dụng webfuzzer.

5.1 Yêu cầu hiện thực đề xuất

Trong đề mục này, chúng tôi sẽ làm rõ những yêu cầu tổng quan và chức năng của ứng dụng và cơ sở dữ liệu của webfuzzer được chúng tôi đề xuất. Công dụng chính của webfuzzer là để lưu trữ những điểm cuối ứng dụng web khả nghi, kiểm thử theo thiết lập có sẵn và lưu lại lịch sử kiểm thử chi tiết. Trong phạm vi luận văn này, chúng tôi chỉ tập trung vào giao diện người dùng (User Interface - UI), backend và cơ sở dữ liệu của ứng dụng webfuzzer, phần mở rộng Burp Suite đã được cung cấp và có tài liệu sẵn.

5.1.1 Ứng dụng web

Dưới đây là những yêu cầu tổng quan và chức năng của ứng dụng webfuzzer trong quá trình thiết kế và hiện thực backend và UI của ứng dụng webfuzzer.

Về tổng quan

1. Khả năng tiếp cận

   Backend và UI phải dễ dàng được tiếp cận được từ phía phần mở rộng Burp Suite và người dùng. Người dùng có thể tiếp cận mọi chức năng, giao diện lập trình ứng dụng (Application Programming Interface - API) của ứng dụng và trước mắt chưa cần thiết phải có hệ thống phân quyền hay hạn chế việc này.

2. Khả năng mở rộng

   Cấu hình kiểm thử ở phía backend phải linh hoạt, có thể được chỉnh sửa và thêm loại lỗ hổng bảo mật mới có thể được phát hiện theo cách tương tự. UI của ứng dụng nên được viết theo kiểu hướng thành phần (component-based) để có thể dễ dàng mở rộng bằng việc tái sử dụng các thành phần tương tự đã được lập trình trước.

3. Hiệu năng

   Thời gian thực hiện một yêu cầu kiểm thử càng xấp xỉ (hoặc nhanh hơn) kiểm thử trên Burp Suite càng tốt. UI phải thân thiện với người dùng, cung cấp được đầy đủ thông tin cần thiết để người dùng tiến hành kiểm thử và kiểm tra kết quả. Thời gian trả về kết quả khi gọi API không quá 1 giây cho một request trong điều kiện bình thường.

Về chức năng

1. Nhận và hiển thị HTTP request mẫu nhận được từ phần mở rộng Burp Suite

   Backend phải nhận được HTTP request mẫu được gửi từ phần ứng dụng của Burp Suite và lưu vào cơ sở dữ liệu, đồng thời UI phải hiển thị được những request mẫu đó thông qua việc tương tác với API của webfuzzer.

2. Tùy chỉnh được cấu hình kiểm thử ứng với mỗi loại lỗ hổng bảo mật

   Cấu hình kiểm thử đối với từng loại lỗ hổng phải có thể chỉnh sửa được (ít nhất là từ phía backend), đồng thời phải cung cấp API trả về cấu hình kiểm thử để hiển thị trên UI.

3. Tạo và thực thi yêu cầu kiểm thử

   Dựa vào request mẫu và cấu hình kiểm thử hiển thị trên UI, người dùng phải dễ

dàng tạo được yêu cầu kiểm thử trên UI thông qua việc chọn request mẫu và những loại lỗ hổng bảo mật cần kiểm thử. Sau đó người dùng có thể xem lại các thông số đó và tiến hành thực thi yêu cầu tương ứng.

4. Xem kết quả kiểm thử

   Người dùng có thể xem lại kết quả kiểm thử trên UI, bao gồm các thông số, trạng thái, kết quả của mỗi yêu cầu kiểm thử cụ thể, kèm theo danh sách các payload phát hiện được mỗi loại lỗ hổng (nếu có). Đồng thời người dùng có thể dễ dàng lọc riêng những yêu cầu kiểm thử phát hiện được lỗ hổng (dương tính) để xem xét thay vì hiển thị cả những yêu cầu không phát hiện được (âm tính).

5.1.2 Cơ sở dữ liệu

Dưới đây là những yêu cầu tổng quan và chức năng của ứng dụng webfuzzer trong quá trình thiết kế và hiện thực một cơ sở dữ liệu nhanh và bền vững, đáp ứng được lưu cầu lưu trữ và truy xuất dữ liệu từ phía backend.

Về tổng quan

1. Hạn chế trùng lặp dữ liệu

   Cơ sở dữ liệu phải tránh được tối đa việc lưu trữ các bản ghi hoặc thông tin trùng lặp, giữ cho kích thước của cơ sở dữ liệu càng nhỏ càng tốt.

2. Khả năng mở rộng

   Lược đồ quan hệ và dữ liệu lưu trữ trong cơ sở dữ liệu cần có khả năng mở rộng cao, đặc biệt là những thông tin có độ dài không xác định trước được như kết quả kiểm thử, độ dài của request mẫu,...

3. Hiệu năng

   Các câu truy vấn tới cơ sở dữ liệu phải có thời gian phản hồi trong khoảng chấp nhận được trong điều kiện bình thường, sao cho thỏa điều kiện thời gian phải hồi khi gọi API không quá 1 giây cho mỗi lời gọi API ở phía backend.

4. Khả năng tương thích

   Hệ quản trị cơ sở dữ liệu được chọn nên tương thích tốt và có thư viện trên ngôn ngữ lập trình backend của webfuzzer. Việc này giúp các thao tác cấu hình cơ sở dữ

liệu, thiết lập các kết nối (connections) và truy vấn dữ liệu (queries) được hiện thực dễ dàng và hiệu quả ở backend.

Về chức năng. Cơ sở dữ liệu phải lưu trữ được đầy đủ và chính xác các thông tin của request mẫu bao gồm điểm cuối ứng dụng web mục tiêu, phương thức HTTP, cookies (nếu có) và các headers được gửi đến từ phần mở rộng Burp Suite; tương tự đối với các yêu cầu kiểm thử và kết quả kiểm thử tương ứng (nếu có).

5.2 Trường hợp sử dụng

Webfuzzer tập trung chủ yếu vào ba trường hợp sử dụng (usecase) chính sau:

- Trường hợp sử dụng 1: Gửi request mẫu từ phần mở rộng Burp Suite

Bảng 5.1: Trường hợp sử dụng 1

Đối tượng tương tác	Người dùng, phần mở rộng Burp Suite, UI webfuzzer
Ranh giới hệ thống	Máy tính cá nhân của người dùng
Điều kiện trước	Backend và UI của ứng dụng webfuzzer đang hoạt động, phần mềm Burp Suite có cài đặt sẵn phần mở rộng và trình duyệt web có thiết lập proxy sẵn
Luồng thực thi	Người dùng chọn và gửi request mẫu cần kiểm thử đến địa chỉ IP:port của ứng dụng. Webfuzzer sẽ phân giải request đó, lưu vào cơ sở dữ liệu và hiển thị lên giao diện người dùng
Điều kiện sau	Request mẫu được hiển thị trong danh sách trên giao diện webfuzzer sau thao tác gửi request và làm mới từ người dùng

- Trường hợp sử dụng 2: Tạo yêu cầu kiểm thử một điểm cuối ứng dụng web

Bảng 5.2: Trường hợp sử dụng 2

Đối tượng tương tác	Người dùng, giao diện người dùng webfuzzer
Ranh giới hệ thống	Máy tính cá nhân của người dùng
Điều kiện trước	Backend và UI của ứng dụng webfuzzer đang hoạt động
Luồng thực thi	Người dùng chọn request mẫu từ danh sách đang hiển thị, sau đó chọn loại lỗ hổng cần kiểm thử và nhấn nút "**Create fuzz request**"
Điều kiện sau	Yêu cầu kiểm thử tương ứng xuất hiện trong trang **Result** trên giao diện webfuzzer

- Trường hợp sử dụng 3: Xem kết quả kiểm thử theo thời gian

Bảng 5.3: Trường hợp sử dụng 3

Đối tượng tương tác	Người dùng, giao diện người dùng webfuzzer
Ranh giới hệ thống	Máy tính cá nhân của người dùng
Điều kiện trước	Backend và UI của ứng dụng webfuzzer đang hoạt động
Luồng thực thi	Người dùng chọn trang **Result** để xem kết quả kiểm thử hoặc chọn một yêu cầu cụ thể để xem chi tiết
Điều kiện sau	Người dùng xem được tổng quan và chi tiết kết quả của mỗi yêu cầu kiểm thử

Bảng 5.1 thể hiện trường hợp sử dụng đầu tiên, cụ thể là khi người dùng muốn gửi request mẫu cần kiểm thử đến ứng dụng để hiển thị trên giao diện người dùng webfuzzer. Người dùng chọn request mẫu và vị trí tham số theo nhu cầu kiểm thử trên phần mở rộng Burp Suite, sau đó gửi đến địa chỉ IP:port đang triển khai ứng dụng webfuzzer (sẽ được trình bày chi tiết ở **Phụ lục A**). Sau đó nội dung của request mẫu này sẽ được hiển thị trên giao diện để người dùng kiểm tra và tạo yêu cầu kiểm thử theo một số lỗ hổng bảo mật.

Bảng 5.2 thể hiện trường hợp người dùng muốn tạo yêu cầu kiểm thử một request mẫu cụ thể với một số loại lỗ hổng. Người dùng chỉ việc chọn request mẫu trong danh sách và đánh dấu loại lỗ hổng cần kiểm thử và nhấn nút "**Create fuzz request**". Yêu cầu kiểm thử sẽ được đưa vào hàng đợi thực thi, đồng thời trạng thái và kết quả kiểm thử theo thời gian của yêu cầu này sẽ được hiển thị trong trang **Result** trên giao diện ứng dụng.

Trường hợp sử dụng 3 được thể hiện trong Bảng 5.3. Trong trường hợp này, người dùng

có thể xem chung kết quả của một số yêu cầu kiểm thử gần đây nhất hoặc xem chi tiết kết quả của một yêu cầu bất kì theo thời gian thực bao gồm kết quả tổng quát và những payload/biểu thức chính quy phát hiện ra lỗi.

5.3 Công nghệ sử dụng

Đề mục này trình bày tóm tắt những ngôn ngữ lập trình, công nghệ, công cụ được chọn để hiện thực ứng dụng webfuzzer. Những công nghệ này được lựa chọn dựa trên những nghiên cứu nền trong quá trình thực hiện luận văn. Chúng tương thích, hỗ trợ lẫn nhau và thích hợp trong việc thỏa mãn những yêu cầu hiện thực đã đề ra.

Node.js là một nền tảng phát triển ứng dụng độc lập được xây dựng trên Javascript Runtime của trình duyệt Chrome, thường được sử dụng để xây dựng các ứng dụng web một cách nhanh chóng và dễ dàng mở rộng. Phần lõi bên dưới của Node.js được viết hầu hết bằng C++ nên tốc độ xử lý và hiệu năng khá cao. Nhìn chung Node.js là một nền tảng rất thích hợp trong việc nhanh chóng xây dựng những ứng dụng web đa chức năng với hiệu năng cao, thích hợp với các ứng dụng tốt nghiệp này vì những lý do sau.

- Node.js chạy đa nền tảng phía Server, sử dụng kiến trúc hướng sự kiện (event-driven), cơ chế non-blocking I/O nhẹ và hiệu quả.
- Ứng dụng viết bằng Node.js có thể chạy được ở bất kỳ hệ điều hành Mac – Windows – Linux.
- Cộng đồng Node.js rất lớn, hơn 1.300.000 gói thư viện (package) được cung cấp hoàn toàn miễn phí trên trang *https://www.npmjs.com/*, hỗ trợ trải dài trên nhiều lĩnh vực lập trình từ đại trà đến đặc thù.
- Các ứng dụng Node.js đáp ứng tốt thời gian thực và chạy đa nền tảng, đa thiết bị.

Express.js là một khung thức nhỏ, linh hoạt được xây dựng trên nền tảng Node.js. Nhiều khung thức nổi tiếng trên nền Node.js hiện đang sử dụng Express.js như **Sails**, **Feathers**,... Express.js được xây dựng dựa trên rất nhiều gói thư viện bổ trợ và cung cấp thêm nhiều tính năng để hỗ trợ lập trình viên tốt hơn, không làm giảm tốc độ vốn có của Node.js. Trong phạm vi luận văn này, Express.js được sử dụng như một bộ định tuyến (routing), đảm nhiệm việc phản hồi thông tin lại cho người dùng khi họ gọi API

đến những điểm cuối của ứng dụng webfuzzer. Những yêu cầu này thuộc dạng HTTP request bao gồm đường dẫn API, phương thức request (GET, POST, DELETE, PUT,...) và các tham số cần thiết theo định dạng dữ liệu đầu vào của mỗi API.

Swagger là một phần mềm mã nguồn mở, chuyên dùng để thiết kế, xây dựng, sử dụng và làm tài liệu cho các ứng dụng web RESTful. Chúng tôi sử dụng thư viện Swagger UI Express[1] trên khung thức Node.js để làm tài liệu cho các API ở backend và gọi thử các API đó, hỗ trợ quá trình hiện thực UI của ứng dụng.

React.js là một thư viện hỗ trợ lập trình giao diện người dùng phổ biến trên ngôn ngữ Javascript. Một ứng dụng React.js được xây dưng dựa trên các thành phần (component) với khả năng tái sử dụng cao, thay vì dựa vào bản mẫu (template) như các thư viện lập trình UI khác, cho phép nhúng mã HTML trong mã Javascript nhờ vào JSX. JSX có ba điểm nổi bật chính.

- Nhanh hơn. Khung thức React.js thực hiện nhiều biện pháp tối ưu hóa trong khi biên dịch JSX sang mã Javacsript. Điều này cho phép việc thực thi các mã JSX này tốn ít thời gian hơn so với một mã tương đương viết trực tiếp bằng Javascript.

- An toàn hơn. JSX được biên dịch trước khi chạy, giống như các ngôn ngữ tĩnh khác như Java, C++,... Vì thế các lỗi nếu có sẽ được phát hiện ngay trong quá trình biên dịch.

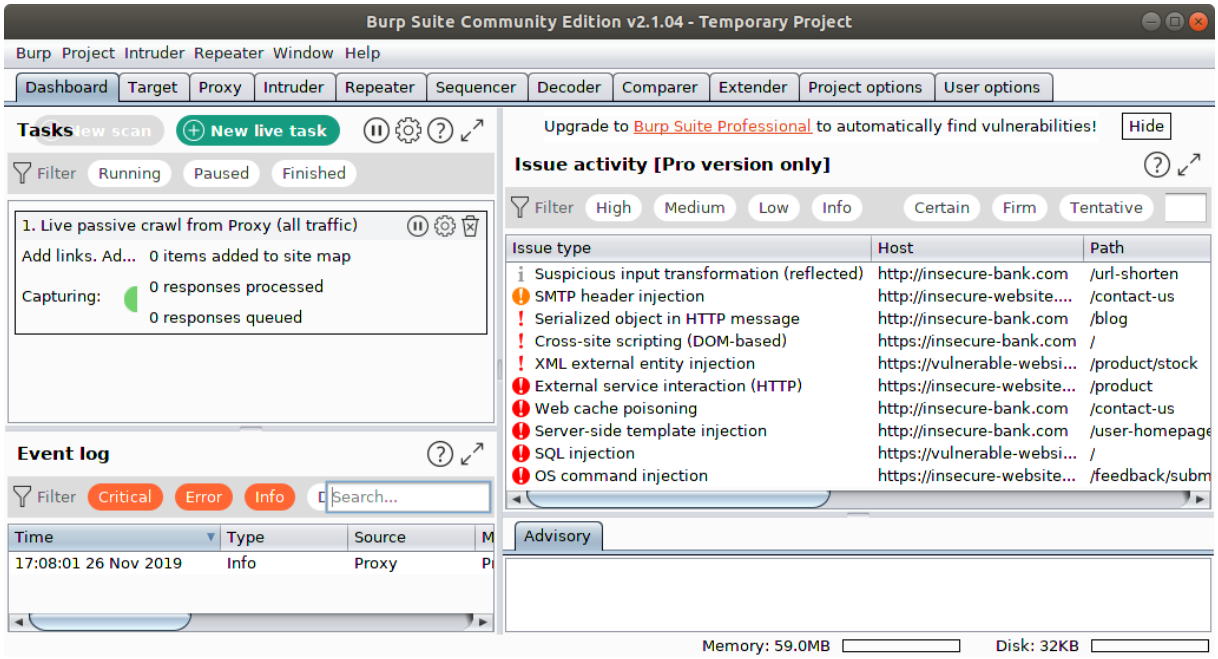
- Dễ sử dụng hơn. JSX được viết dựa trên Javascript, vì vậy rất dễ dàng để các lập trình viên Javascript như chúng tôi có thể hiểu và sử dụng.

SASS (Syntactically Awesome Style Sheets) là một ngôn ngữ kịch bản tiền xử lý (preprocessor scripting language) được biên dịch thành ngôn ngữ CSS (Cascading Style Sheets). CSS là ngôn ngữ định kiểu các phần tử được tạo ra bởi các ngôn ngữ đánh dấu như HTML. Phương thức hoạt động của CSS là nó sẽ dựa vào các vùng chọn, vùng chọn có thể là tên một thẻ HTML, tên một ID, class hay nhiều kiểu khác, sau đó áp dụng các thuộc tính định kiểu lên các vùng chọn đó. SASS giúp quá trình viết mã định kiểu CSS dễ dàng và tiện lợi hơn thông qua cú pháp riêng của nó, sử dụng dấu ngoặc nhọn và thụt lề để biểu thị các khối mã và dấu chấm phẩy để phân tách các quy tắc giữa các thuộc

[1]Nguồn: https://www.npmjs.com/package/swagger-ui-express

tính con trong một khối. SASS được sử dụng để định kiểu và làm đẹp các thành phần trong giao diện người dùng của ứng dụng webfuzzer.

Burp Suite [7] là một khung thức kiểm thử thâm nhập (penetration testing framework) ứng dụng web trên nền Java, được cung cấp và phát triển bởi PortSwigger và đã trở thành bộ công cụ tiêu chuẩn được dùng trong công nghiệp bởi những kĩ sư an toàn thông tin. Công cụ này hỗ trợ nhận diện lỗ hổng bảo mật và xác thực các véc tơ tấn công vào ứng dụng web.



Hình 5.1: Giao diện chính của công cụ Burp Suite

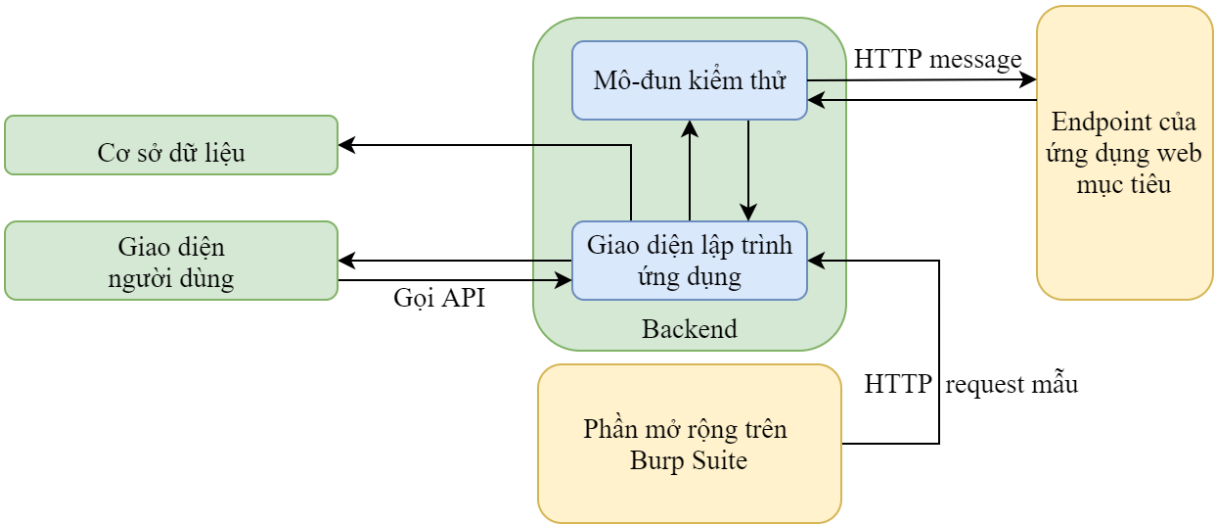
Trong phạm vi luận văn này, Burp Suite vừa đóng vai trò như một proxy trung gian (interception proxy) can thiệp vào quá trình gửi HTTP request đến ứng dụng web mục tiêu, vừa đóng vai trò của một người trung gian (man in the middle - MITM) hỗ trợ chúng tôi phân tích, chỉnh sửa và chuyển tiếp request mẫu đó tới ứng dụng webfuzzer đồng thời cung cấp dữ liệu đối chứng trong quá trình kiểm thử tính đúng đắn và hiệu quả của ứng dụng.

MySQL là một hệ quản trị cơ sở dữ liệu mã nguồn mở, được sử dụng phổ biến trên thế giới vì tốc độ cao, sự ổn định, dễ sử dụng và khả năng hoạt động được trên nhiều hệ điều hành của nó. MySQL là một hệ quản trị cơ sở dữ liệu quan hệ, sử dụng ngôn ngữ truy vấn có cấu trúc (Structured Query Language - SQL). Đặc biệt MySQL có tính sẵn sàng

và nhất quán cao theo mô hình CAP (Consistency - Availability - Partition tolerance), có sẵn thư viện tương tác được viết bằng Node.js và tương thích tốt với các hệ điều hành Windows, Linux, Mac OS X. Dựa vào những yếu tố trên, MySQL rất phù hợp trong việc hiện thực một cơ sở dữ liệu dựa trên những công nghệ và yêu cầu đã đề ra.

5.4 Thiết kế kiến trúc hệ thống đề xuất

Đề mục này mô tả thiết kế chi tiết của kiến trúc hệ thống ứng dụng webfuzzer mà chúng tôi đề xuất, bao gồm ba thành phần backend, UI và cơ sở dữ liệu của ứng dụng, được khái quát như Hình 5.2 sau.



Hình 5.2: Kiến trúc đề xuất của ứng dụng webfuzzer

5.4.1 Backend

Hai thành phần quan trọng nhất của backend là cấu hình - mô-đun kiểm thử và giao diện lập trình ứng dụng. Các cấu hình kiểm thử được xây dựng dựa trên những yếu tố cần thiết để kiểm thử bằng chức năng `Intruder` của Burp Suite. Giao diện lập trình ứng dụng được thiết kế theo kiểu RESTful API, giao tiếp với UI bằng các HTTP request/response. RESTful API là một tiêu chuẩn được áp dụng để thiết kế các API cho ứng dụng web, thông qua việc quy định cách sử dụng các phương thức HTTP (`GET, POST, PUT, DELETE...`) và cách định dạng URL sao cho phù hợp với chức năng của từng API và quản lý tài nguyên chung của ứng dụng một cách có hệ thống. **Cấu hình và mô-đun kiểm thử.** Dựa vào cách thức phát hiện lỗ hổng bằng thời gian phản hồi HTTP request và sự trùng

khớp nội dung của response trả về, đối với mỗi loại lỗ hổng, chúng tôi xây dựng cấu hình kiểm thử bao gồm các thuộc tính sau.

- **label** là tên của lỗ hổng đang kiểm thử, dùng trong việc hiển thị trên giao diện và ghi nhật kí hoạt động.
- **time** là thời gian tối thiểu để một request kiểm thử trả về response, nếu thời gian phản hồi này lớn hơn giá trị của thuộc tính **time** thì được coi là phát hiện thành công lỗ hổng time-based SQLI. Thuộc tính này có thể là số giây cụ thể hoặc một chuỗi rỗng trong trường hợp người dùng không muốn giới hạn thời gian.
- **match** là một mảng các chuỗi để so trùng trên HTTP response, điểm cuối của ứng dụng web mục tiêu sẽ bị coi là có lỗ hổng nếu response trả về có chứa một trong những chuỗi này.
- **payloadFile** là đường dẫn tập tin chứa các payload kiểm thử. Mặc định các payload này được chứa trong thư mục `payloads` trong thư mục gốc của mã nguồn backend và phân loại theo lỗ hổng bảo mật.
- **matchFile** là tập tin chứa các chuỗi để so trùng tương tự như thuộc tính **match** của cấu hình.
- **regex** là biểu thức chính quy dùng để phát hiện các chuỗi đáng ngờ trong HTTP response trả về, tương tự như thuộc tính **match**, điểm cuối của ứng dụng web mục tiêu sẽ bị coi là có lỗ hổng nếu response trả về khớp với các biểu thức chính quy này. Các biểu thức chính quy không chỉ bao gồm được các chuỗi để so trùng mà còn gồm cả các thông báo lỗi từ hệ điều hành, hệ quản trị cơ sở dữ liệu hay của chính ứng dụng web mục tiêu. Điều này gợi ý cho chúng tôi về những tiềm năng khai thác xa hơn, ngoài những lỗ hổng bảo mật ở tầng ứng dụng.

Việc thiết kế mô-đun kiểm thử sẽ được hiện thực dựa trên nguyên tắc hoạt động của chức năng `Intruder` trên Burp Suite, sử dụng loại tấn công `Sniper` như đã đề cập ở **Chương 4**, gồm các mô-đun nhỏ và các bước tuần tự như sau.

1. Mô-đun phân tích request mẫu (`requestParser`) nhận vào HTTP request mẫu và cấu hình kiểm thử. Mô-đun này tiến hành đọc danh sách các payload từ thuộc tính `payloadFile` của cấu hình kiểm thử. Sau đó xác định từng vị trí thay thế payload

riêng lẻ trong request mẫu (ở dạng chuỗi), được đánh dấu bằng cặp kí tự "§" để thay thế lần lượt từng payload vào từng vị trí. Kết quả trả về của mô-đun này là một danh sách các HTTP request mẫu có chứa payload.

2. Mô-đun xây dựng HTTP request kiểm thử (`requestBuilder`) nhận vào danh sách request mẫ có chứa payload ở trên. Mô-đun này lần lượt xây dựng HTTP request kiểm thử đầy đủ dựa trên các thành phần `url, headers, cookies, data, method` của từng request mẫu trong danh sách, sau đó lần lượt gửi các request kiểm thử này đến ứng dụng web mục tiêu.

3. Các mô-đun kiểm thử (`fuzzer`) sẽ kiểm tra thời gian phản hồi và so trùng nội dung của HTTP response trả về dựa trên cấu hình kiểm thử để xác định khả năng có lỗ hổng của điểm cuối ứng dụng web mục tiêu.

4. Trong trường hợp mô-đun kiểm thử xác nhận được lỗ hổng từ quá trình phân tích response trả về như trên, mô-đun lưu trữ kết quả kiểm thử (`updateFuzzingLog`) sẽ được gọi. Mô-đun này đảm nhiệm việc lưu lại (hoặc cập nhật) loại lỗ hổng mà điểm cuối của ứng dụng web mục tiêu mắc phải kèm theo những payload phát hiện được lỗ hổng tương ứng vào cơ sở dữ liệu.

Giao diện lập trình ứng dụng. Giao diện lập trình ứng dụng của webfuzzer sẽ được điều hướng thông qua chức năng định tuyến (routing) được hỗ trợ bởi khung thức Express.js và hiện thực bằng các lớp trình điều khiển (controller class). Các lớp này cung cấp các phương thức (method) để tương tác với cơ sở dữ liệu, trả về thông tin và tiến hành kiểm thử ở phía backend. UI của webfuzzer sẽ tạo HTTP request gọi đến đường dẫn API kèm theo những tham số cần thiết, các phương thức này sẽ xử lý những request đó và trả về response dưới dạng đối tượng JSON. Các lớp trình điều khiển và các phương thức trong đó hoạt động độc lập với nhau, được phân loại theo chức năng như sau.

- Lớp **BurpController** xử lý các yêu cầu liên quan đến HTTP request mẫu, bao gồm nhận request từ phía phần mở rộng Burp Suite và trả về danh sách các request mẫu đã lưu.

- Lớp **TargetController** đảm nhiệm việc tạo yêu cầu kiểm thử, truy vấn cấu hình kiểm thử và thông tin của các yêu cầu kiểm thử. Các phương thức trong lớp này không chỉ trả về một danh sách các yêu cầu kiểm thử được lọc ra theo nội dung tìm

kiểm hoặc phân loại theo kết quả kiểm thử có lỗ hổng hay không mà còn có thể trả về thông tin chi tiết của một yêu cầu kiểm thử đơn lẻ.

- Lớp **FuzzController** cung cấp các phương thức thực thi yêu cầu kiểm thử theo ID hoặc một yêu cầu ngẫu nhiên đã được tạo.

Mỗi phương thức trong các lớp trình điều khiển này thực hiện những công việc tương ứng với chức năng của một API. Cụ thể, Bảng 5.4 dưới đây mô tả kiến trúc giao diện lập trình ứng dụng ở phía backend của webfuzzer, được xây dựng dựa trên tiêu chuẩn RESTful.

Bảng 5.4: Kiến trúc giao diện lập trình ứng dụng của webfuzzer backend

Lớp	Phương thức	Đường dẫn API	Mô tả
Burp Controller	receiveTargetFromBurp	`POST /`	Nhận request mẫu từ phần mở rộng Burp Suite
	getAllEndpoint	`GET /`	Truy vấn danh sách các request mẫu đã nhận
Target Controller	createFuzzRequest	`POST /target`	Tạo yêu cầu kiểm thử
	retrieveRequestInfo	`GET /target`	Truy vấn một yêu cầu kiểm thử theo ID
	getRequestList	`GET /target/list`	Truy vấn danh sách các yêu cầu kiểm thử
	getVulnTypes	`GET /target/configs`	Truy vấn danh sách cấu hình kiểm thử
	searchUrl	`GET /target/search`	Lọc yêu cầu kiểm thử theo URL
Fuzz Controller	executeFuzzRequest	`GET /fuzz`	Thực thi một yêu cầu kiểm thử theo ID
	executeSubmittedRequest	`GET /fuzz/free-request`	Thực thi một yêu cầu kiểm thử đã tạo

5.4.2 Giao diện người dùng

Giao diện của ứng dụng webfuzzer sẽ có hai trang web chính và thanh điều hướng để chuyển đổi qua lại giữa hai trang này. Thiết kế chi tiết kèm theo những API mà các thành phần này sử dụng được mô tả như sau.

1. Trang bảng điều khiển (Dashboard page) để người dùng tạo yêu cầu kiểm thử dựa

trên HTTP request mẫu và cấu hình kiểm thử.

- Khung hiển thị danh sách URL các HTTP request mẫu để chọn. Danh sách này là kết quả trả về của việc gọi API `GET /`, đồng thời có khung hiển thị nội dung của request mẫu đã chọn kèm theo các kí tự "§" đánh dấu vị trí chèn payload như lúc gửi request mẫu từ chức năng `Intruder` của Burp Suite.
- Khung hiển thị và chọn các loại lỗ hổng bảo mật cần kiểm thử. Khung này gồm một danh sách các checkbox kèm theo tên lỗ hổng, người dùng có thể chọn một hay nhiều lỗi tùy theo nhu cầu kiểm thử. Danh sách các lỗ hổng này được trả về thông qua việc gọi API `GET /target/configs`.
- Nút tạo yêu cầu kiểm thử cho phép người dùng tạo yêu cầu kiểm thử từ request mẫu và các lỗ hổng bảo mật đã chọn, gọi đến API `POST /target`.

2. Trang kết quả kiểm thử (Result page) để người dùng thực thi và kiểm tra kết quả kiểm thử. Trang này cho phép người dùng xem thông tin chi tiết của mỗi yêu cầu kiểm thử, đồng thời cho phép (tái) thực thi những yêu cầu đã tạo hoặc kiểm thử chưa hoàn tất.
   - Thành phần chính của trang này là một bảng hiển thị thông tin tổng quan của một danh sách các yêu cầu kiểm thử được trả về từ API `GET /target/list` hoặc `GET /target/search`.
   - Mỗi hàng của bảng tương ứng với một yêu cầu kiểm thử. Cuối hàng có một nút nhấn để xổ xuống thông tin chi tiết của yêu cầu kiểm thử tương ứng, được truy vấn bằng API `GET /target`. Do chi tiết của các yêu cầu kiểm thử này sẽ chứa những trường thông tin tương tự nhau, chúng sẽ được thiết kế như một thành phần (component) để có thể tái sử dụng được nhiều lần trong bảng.
   - Ở cuối phần thông tin chi tiết của một yêu cầu kiểm thử, chúng tôi thêm một nút nhấn gọi đến API `GET /fuzz` để (tái) thực thi yêu cầu kiểm thử đang chọn.

3. Thanh điều hướng (Sidebar) gồm tên ứng dụng webfuzzer, hai mục Dashboard và Result để điều hướng tới hai trang web chính của ứng dụng, kèm theo đường dẫn

đến thư mục mã nguồn ứng dụng trên gitlab[2].

Bên cạnh các thành phần chức năng chính trên, chúng tôi còn cần phải hiện thực thêm một số thành phần của UI để cải thiện trải nghiệm người dùng (user experience - UX) như hệ thống thông báo đẩy (toast notification), cơ chế phân trang (paging) đối với danh sách request mẫu và thông tin yêu cầu kiểm thử, giao diện ứng dụng đẹp mắt, dễ nhìn,...

5.4.3 Cơ sở dữ liệu

Chúng tôi lựa chọn một hệ quản trị cơ sở dữ liệu quan hệ, cụ thể là MySQL vì phù hợp mô hình dữ liệu và nội dung lưu trữ. Nhằm đảm bảo mục đích lưu trữ các HTTP request mẫu, yêu cầu và kết quả kiểm thử, cơ sở dữ liệu của ứng dụng nên tách ra làm ba bảng chứa các đối tượng trên, tạm gọi lần lượt là các bảng `Endpoint, Request, Result`. Đối tượng chính là sẽ là bảng `Request`, chứa request mẫu, cấu hình kiểm thử (nếu có thể), các loại lỗ hổng cần kiểm thử, kết quả kiểm thử và một số thông tin khác. Một request mẫu có thể được kiểm thử nhiều lần tùy theo nhu cầu của người dùng nên ta tách `Endpoint` ra thành một bảng riêng, bảng `Request` lúc này sẽ có một trường tham khảo đến khóa chính của bảng `Endpoint` để tránh tình trạng dư thừa dữ liệu. Tương tự, bảng `Result` chứa kết quả kiểm thử cũng nên được tách riêng và thiết lập quan hệ khóa ngoài với bảng `Request` để dễ dàng thay đổi và mở rộng sau này.

Ngoài ra, có một phần dữ liệu cần lưu trữ thuộc kiểu đối tượng JSON[3] ở phía backend. Tuy MySQL có hỗ trợ lưu trữ trực tiếp kiểu dữ liệu đối tượng JSON nhưng các giá trị có kiểu này nên được chuỗi hóa ở phía backend trước khi gửi đến cơ sở dữ liệu. Lý do là vì các thao tác tìm kiếm và chèn payload trên request mẫu ở mô-đun kiểm thử sẽ rất phức tạp khi triển khai trên kiểu dữ liệu JSON, chúng tôi cần phải duyệt toàn bộ các cặp thuộc tính - giá trị lồng nhau để tìm những vị trí đó. Trong khi nếu sử dụng kiểu dữ liệu chuỗi, việc tìm kiếm và thay thế (bằng chuỗi hoặc biểu thức chính quy) trên chuỗi đều đã được hiện thực sẵn trong ngôn ngữ để lập trình viên sử dụng. Vì vậy một số trường liên quan đến nội dung request mẫu và kết quả kiểm thử trong các bảng sẽ có kiểu chuỗi để đồng nhất trong quá trình lưu trữ và xử lý.

[2]Nguồn: https://gitlab.com/y4t0g4m1/webfuzzer-nodejs

[3]JSON (JavaScript Object Notation) là một kiểu dữ liệu mở thường sử dụng trong Node.js và Javascript, các đối tượng này được cấu thành từ nhiều cặp thuộc tính - giá trị lồng nhau. Định nghĩa chi tiết của kiểu dữ liệu JSON có thể được tham khảo tại `https://www.json.org/json-en.html`

Chương 6

Hiện thực ứng dụng webfuzzer

Chương này mô tả chi tiết hiện thực của kiến trúc hệ thống ứng dụng webfuzzer, bao gồm ba thành phần backend, UI và cơ sở dữ liệu của ứng dụng dựa trên thiết kế đã đề ra ở **Chương 5**. Mã nguồn của ứng dụng webfuzzer được tải lên ở `https://gitlab.com/y4t0g4m1/webfuzzer-nodejs`. Thông tin triển khai ứng dụng trên VPS được đề cập ở **Phụ lục A** tại thời điểm hoàn thành luận văn.

6.1 Backend

Đề mục này trình bày chi tiết hiện thực backend của ứng dụng webfuzzer, bao gồm giao diện lập trình ứng dụng, định dạng dữ liệu của phần mở rộng Burp Suite, các biến môi trường, cấu hình và mô-đun kiểm thử.

6.1.1 Giao diện lập trình ứng dụng

Ứng dụng webfuzzer sử dụng khung thức Express.js để hiện thực một máy chủ web ở backend, dễ dàng định tuyến các API ở backend và lắng nghe các HTTP request gọi đến các API này. Đề mục này mô tả chi tiết hiện thực của các API thuộc lớp trình điều khiển `target`, hai lớp trình điều khiển còn lại được hiện thực tương tự. Trước tiên backend nạp đường dẫn gốc của các lớp API, nạp các lớp trình điều khiển vào tương ứng, sau đó khởi động một máy chủ lắng nghe ở một cổng xác định bằng các phương thức `app.use`, `app.listen` như trong Đoạn mã 6.1 sau.

```
app.use('/', burpRoute(express.Router(), app, burpMethods));
app.use('/target', targetRoute(express.Router(), app, targetMethods));
app.use('/fuzz', fuzzRoute(express.Router(), app, fuzzMethods));
app.use('/recon', reconRoute(express.Router(), app, reconMethods));
app.use('/api-docs', swaggerUi.serve, swaggerUi.setup(swaggerDocument));
app.listen(port, '0.0.0.0', () => {
    console.log(`webfuzzer is listening on 0.0.0.0:${port}`);
});
```

Đoạn mã 6.1: Thiết lập đường dẫn gốc của các API và khởi động backend

Chúng tôi sử dụng thư viện Swagger UI Express để xây dựng tài liệu cho giao diện lập trình ứng dụng của webfuzzer. Thư viện này hỗ trợ vẽ ra một trang web chứa đầy đủ thông tin của các API từ tập tin `swagger.json` trong đường dẫn gốc của thư mục mã nguồn backend, đồng thời cho phép người dùng gọi API đến địa chỉ máy chủ được thiết lập sẵn trong thuộc tính `host` của tập tin trên. Trang web trên nằm ở đường dẫn `/api-docs` trên máy chủ backend webfuzzer.

Lấy ví dụ các API thuộc lớp `target`, chúng tôi định nghĩa đường dẫn của các API này bằng cách thêm chúng vào đối tượng `router` sẵn có của khung thức Express.js. Chúng tôi định nghĩa mỗi đường dẫn API sẽ làm gì khi được gọi đến như Đoạn mã 6.2 sau.

```
let targetRoute = (router, expressApp, targetMethods) => {
    router.post('/', targetMethods.createFuzzRequest);
    router.get('/', targetMethods.retrieveRequestInfo);
    router.get('/list', targetMethods.getRequestList);
    router.get('/configs', targetMethods.getVulnTypes);
    router.get('/search', targetMethods.searchUrl);
    return router;
}
const _targetRoute = targetRoute;
export { _targetRoute as targetRoute };
```

Đoạn mã 6.2: Định nghĩa đường dẫn cho các API thuộc lớp target

Ví dụ, câu lệnh `router.post('/', targetMethods.createFuzzRequest);` có nghĩa là khi người dùng gọi đến đường dẫn `/target` với phương thức `POST` thì phương thức `targetMethods.createFuzzRequest` của lớp `targetController` sẽ được khởi chạy. Các phương thức trong mỗi lớp trình điều khiển là nơi thực hiện chức năng của các API ở backend, chủ yếu là đọc/ghi dữ liệu từ cơ sở dữ liệu thông qua các câu truy vấn và thực hiện kiểm thử. Lớp `targetController` gồm các phương thức đã được thiết kế ở **Chương 5** như Hình 6.1 sau.

```
4   export default class targetController {
5       constructor(injectedTargetService, injectedResponseHandler) {
6           targetService = injectedTargetService;
7           response = injectedResponseHandler;
8       }
9   >   async createFuzzRequest(req, res) {…
34      }
35  >   async retrieveRequestInfo(req, res) {…
45      }
46  >   async getRequestList(req, res) {…
55      }
56  >   async getVulnTypes(req, res) {…
66      }
67  >   async searchUrl(req, res) {…
77      }
78  }
```

Hình 6.1: Các phương thức của lớp targetController

Backend của ứng dụng webfuzzer sẽ làm việc rất nhiều với cơ sở dữ liệu MySQL. Chúng tôi hiện thực các hàm kết nối cơ sở dữ liệu, truy vấn và mở/đóng kết nối sau quá trình thực hiện câu truy vấn, định dạng dữ liệu trả về từ database cũng như các phương thức liên quan đến giao dịch (transaction) trong tập tin `services/db.js` trong thư mục gốc mã nguồn backend để tiện quản lý và sửa đổi một cách nhất quán. Những API liên quan đến việc truy vấn danh sách request mẫu và thông tin yêu cầu kiểm thử đều được hỗ trợ thêm các thuộc tính `limit, offset` và `total` để truy vấn một khoảng dữ liệu xác định, phục vụ quá trình phân trang các dữ liệu này ở giao diện người dùng như Đoạn mã 6.3 sau.

```
1 let endpointList = await mySqlConnection.query(`SELECT * FROM Endpoint
    ORDER BY Id DESC LIMIT ? OFFSET ?;`, [limit ? parseInt(limit) : 10,
    offset ? parseInt(offset) : 0]);
```

```
2 if (endpointList.error) return reject(endpointList.error);
3 endpointList.results = endpointList.results.map(ele => {
4     return { ...ele, BaseRequest:
    JSON.parse(escapeTarget(ele.BaseRequest)) };
5 })
6 let total = await mySqlConnection.query(`SELECT Count(Endpoint.Id) as
    total FROM Endpoint;`);
7 return resolve({ endpointList: endpointList.results, total:
    total.results[0].total });
```

Đoạn mã 6.3: API truy vấn danh sách request mẫu có hỗ trợ phân trang

Định dạng kết quả trả về của cái API được quy định bởi Đoạn mã 6.4 sau. Nếu có lỗi trong quá trình gọi API thì HTTP response phản hồi có mã lỗi 400 cùng với thông tin về lỗi trong trường `error`, ngược lại trong trường hợp gọi API thành công thì response phản hồi có mã lỗi 200 cùng với dữ liêu cần truy vấn trong trường `results`.

```
1 const response = (res, result, error) => {
2   res.status(error ? (error.httpCode || 400) : (result ? result.httpCode
    || 200 : 404)).send({ 'error': error, 'results': result });
3 }
```

Đoạn mã 6.4: Định dạng kết quả API trả về ở backend

6.1.2 Phần mở rộng Burp Suite

Phần mở rộng Burp Suite cho phép gửi HTTP request mẫu từ mô-đun `Intruder` của Burp Suite đến công cụ **webfuzzer** theo định dạng JSON như Đoạn mã 6.5 dưới đây, quá trình chọn và gửi request mẫu sẽ được trình bày ở **Phụ lục A** của luận văn.

```
1 {
2   "url": "http://testphp.vulnweb.com:80/showimage.php
    ?file=\xa7./pictures/1.jpg\xa7",
3   "cookies": "",
4   "headers": {
```

```
5       "User-Agent": "Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:74.0)
    Gecko/20100101 Firefox/74.0",
6       "Accept": "text/html,application/xhtml+xml,application/xml;
    q=0.9,image/webp,*/*;q=0.8",
7       "Accept-Language": "en-US,en;q=0.5",
8       "Accept-Encoding": "gzip, deflate",
9       "DNT": "1",
10      "Connection": "close",
11      "Upgrade-Insecure-Requests": "1"
12    },
13    "method": "get"
14  }
```

Đoạn mã 6.5: Cấu trúc của một HTTP request mẫu được gửi từ phần mở rộng Burp Suite

Chuỗi "`\xa7`" trong trường `data.searchFor` biểu thị kí tự "§" trong request mẫu. Các kí tự này thường nằm ở trường `data` hoặc `url` trong đoạn dữ liệu JSON để đánh dấu những vị trí chèn payload để ứng dụng web xử lí.

6.1.3 Thiết lập các biến môi trường

Các biến môi trường (environment variables) được thiết lập để điều chỉnh hành vi của backend ứng dụng webfuzzer một cách linh hoạt mà không cần phải sửa đổi mã nguồn chương trình. Các biến này thường quy định luồng xử lý và các giá trị khởi tạo để vận hành chương trình, giúp việc điều chỉnh hành vi của chương trình dễ dàng hơn. Trong backend của ứng dụng webfuzzer, giá trị mặc định của chúng được chứa trong tập tin `globalConfig.js` trong thư mục gốc của mã nguồn. Chúng tôi đặt các biến môi trường trong tập tin `.js` rồi xuất (export) ra để backend sử dụng, tận dụng các kiểu dữ liệu có sẵn của Node.js như đối tượng JSON, biểu thức chính quy, mảng,... thay vì chỉ là chuỗi và số như các biến môi trường thông thường. Người dùng có thể sửa đổi theo nhu cầu và khởi động lại backend để áp dụng các giá trị mới. Bảng 6.1 dưới đây liệt kê và mô tả chức năng của các biến môi trường.

Bảng 6.1: Danh sách các biến môi trường ở backend ứng dụng webfuzzer

Tên biến môi trường	Kiểu dữ liệu	Mô tả
SERVICE_PORT	Integer	Cổng khởi chạy backend của ứng dụng
mysqlConfig	Object	Các thiết lập để kết nối đến dịch vụ MySQL, bao gồm host:port, tên CSDL và thông tin đăng nhập
defaultFuzzConfig	Object	Cấu hình kiểm thử mặc định
defaultVulnTypes	Array	Các lỗ hổng mặc định được chọn khi tự động tạo yêu cầu kiểm thử
fuzzStrategy	String	Chiến thuật kiểm thử mặc định
autoCreateFuzzRequest	Boolean	Tự động tạo yêu cầu kiểm thử sau khi nhận HTTP request mẫu từ Burp Suite với các lỗ hổng mặc định
autoFuzz	Boolean	Tự động thực thi yêu cầu kiểm thử vừa tạo nếu không có yêu cầu nào khác đang được thực thi
autoExecuteQueuedRequest	Boolean	Tự động thực thi một yêu cầu kiểm thử đã tạo ngay sau khi thực thi xong yêu cầu hiện tại
encodeUrl	Boolean	Encode URL của HTTP request kiểm thử gửi đi
verbose	Boolean	Hiển thị thêm chi tiết hoạt động của backend trên terminal
displayResultOnTerminal	Boolean	Hiển thị kết quả kiểm thử trên terminal ở backend

6.1.4 Cấu hình kiểm thử

Điểm chung của những lỗ hổng bảo mật đã giới thiệu ở **Chương 3** là đều có thể bị khai thác bằng dữ liệu đầu vào từ phía người dùng. Vì thế, ta có thể phát hiện các lỗ hổng này bằng cách thử gửi nhiều dạng payload kiểm thử theo các tham số của HTTP request và phân tích phản hồi của máy chủ trong response trả về, việc làm này đặc biệt phù hợp với phương pháp fuzzing. Cụ thể giới hạn và phương pháp phát hiện của ứng dụng **webfuzzer** đối với từng lớp lỗ hổng như sau.

- LFI: ứng dụng tập trung phát hiển lỗ hổng này trên các máy chủ web dùng hệ điều hành nhân Unix. Cụ thể ứng dụng sẽ dùng những payload như Đoạn mã 3.2 để

cố gắng đọc nội dung tập tin `/etc/passwd`. Ứng dụng sẽ kiểm tra HTTP response trả về xem có xuất hiện chuỗi "**root:x**" hoặc khớp với các biểu thức chính quy hay không để kết luận.

- Time-based SQLI: ứng dụng phát hiện tốt trên các ứng dụng web sử dụng các hệ quản trị cơ sở dữ liệu MySQL, Microsoft SQL Server và Postgres SQL bằng cách sử dụng các payload có chứa các hàm hoãn thời gian (đã đề cập ở **Phần 3.2**) của cả ba hệ quản trị cơ sở dữ liệu này, sau đó căn cứ vào thời gian phản hồi của HTTP response để kết luận.

Các payload, biểu thức chính quy, kĩ thuật tấn công, làm rối sử dụng trong ứng dụng được tổng hợp từ kinh nghiệm của chúng tôi và tổng hợp từ các kho lưu trữ (repository) mã nguồn trên GitHub [9, 1], từ nhiều blog chia sẻ cũng như lời giải các thử thách cướp cờ (capture the flag - CTFs) của đàn anh đi trước trong ngành, từ các tweet được các kĩ sư bảo mật chia sẻ trên mạng xã hội Twitter,... Sau đó các payload này sẽ được chúng tôi chọn lọc và sửa đổi cho phù hợp với phương pháp kiểm thử của ứng dụng như sau.

- Đối với các payload LFI, chúng tôi thu thập tám payload mẫu với số lượng các chuỗi "../" tăng dần, nhằm tăng độ sâu của đường dẫn để tiếp cận được tập tin `/etc/passwd` của máy chủ. Từ 8 payload này, chúng tôi áp dụng các kĩ thuật làm rối riêng biệt để sinh ra các payload khác. Các kĩ thuật này bao gồm encode các chuỗi kí tự "../", "..\" và "..\/" theo các tập kí tự (charset) ít phổ biến như "`..%c0%af`", "`..%252f`", "`%%32%65`", "`..%%32%66`", "`..%25c1%259c`",... để vượt qua một số kĩ thuật lọc và làm sạch dữ liệu ở phía server-side của máy chủ ứng dụng web mục tiêu.

- Đối với các payload time-based SQLI, chúng tôi sử dụng các payload chứa hai loại hàm chính để hoãn thời gian thực hiện request. Cụ thể là hàm `sleep` (đối với MySQL, tương tự là `pg_sleep` đối với PostgreSQL và `waitfor delay`, `waitfor time` đối với Microsoft SQL Server) và hàm `BENCHMARK(10000000,MD5('Y4T0G4M11337'))` để kéo dài thời gian thực hiện request đối với các ứng dụng web mắc lỗi này. Bên cạnh đó, với mỗi payload gốc, chúng tôi áp dụng thêm nhiều cách thức đóng ngoặc, chú thích để tránh gặp lỗi cú pháp khi thực thi đoạn mã SQL ở phía máy chủ, ví dụ một số payload trong Đoạn mã 6.6 sau.

```
') OR 1337 LIKE BENCHMARK(10000,MD5('Y4T0G4M11337'))--+-
OR ELT(1337 LIKE 1337,SLEEP(10))--+-
' ,(SELECT (CASE WHEN (1337 LIKE 1337) THEN (SELECT 1337
   FROM PG_SLEEP(10)) ELSE 1/(SELECT 0) END)) AND '
' ,(SELECT (CASE WHEN (1337 LIKE 1337) THEN SLEEP(10) ELSE
   1337 END))--+-
' RLIKE SLEEP(10)--+-
' (SELECT * FROM (SELECT(SLEEP(10)))Y4T0G4M11337)--+-
' WAITFOR DELAY '0:0:10'--+-
1)) or pg_sleep(*index*)--
```

Đoạn mã 6.6: Ví dụ một số payload kiểm thử lỗ hổng time-based SQLI

Cụ thể cấu hình được áp dụng trong ứng dụng và biểu thức chính quy áp dụng trong trường hợp phát hiện lỗi LFI được thể hiện ở Đoạn mã 6.7 dưới đây.

```
1 let lfiRegex1 = /Warning: include\(|Warning: unlink\(|for inclusion
     \(include_path=|fread\(|Failed opening required|Warning:
     file_get_contents\(|Fatal error: require_once\(|Warning:
     file_exists\(/g;
2 let lfiRegex2 =
     /java\.io\.FileNotFoundException|java\.lang\.Exception|java\.lang\
     .IllegalArgumentException|java\.net\.MalformedURLException/g;
3 let defaultFuzzConfig = {
4     "1": { "label": "Local File Inclusion", "time": "", "match":
     ["root:x", "localhost"], "payloadFile": "lfi/lfi-full.txt",
     "matchFile": "", "regex": [lfiRegex1, lfiRegex2] },
5     "2": { "label": "Time-based SQL Injection", "time": 10, "match": "",
     "payloadFile": "sqli/sqli-time-based.txt", "matchFile": "", "regex":
     [sqliRegex] },
6     "3": { "label": "Remote Code Execution", "time": 10, "match": "",
     "payloadFile": "rce.txt", "matchFile": ["root:x", "371337",
     "99cebf4d56e5168a1915dd72213db9eb"], "regex": "" }
```

```
7 }
```

Đoạn mã 6.7: Cấu hình kiểm thử mặc định

Các cấu hình kiểm thử được quản lý bằng ID dạng số, được định nghĩa và cung cấp từ phía backend thông qua API để thống nhất giữa ba thành phần của hệ thống trong quá trình thực thi và lưu trữ. Các cấu hình này thường được chúng tôi sử dụng trong quá trình tìm lỗ hổng bảo mật trên các ứng dụng web bằng Burp Suite, chúng tôi sử dụng lại những cấu hình này cho việc kiểm thử ở ứng dụng webfuzzer để tiện so sánh và đánh giá kết quả hiện thực ứng dụng. Vì gặp khó khăn trong việc truyền biểu thức chính quy trong Node.js qua lại giữa UI và cơ sở dữ liệu (bao gồm việc chuỗi hóa để lưu vào cơ sở dữ liệu và tái cấu trúc biểu thức chính quy để kiểm thử trên backend) nên hiện tại các cấu hình trên đang được thiết lập dưới dạng biến môi trường. Các tập tin payload được phân loại và lưu trữ trong thư mục `payloads` ở thư mục gốc mã nguồn backend, mỗi payload nằm trên từng dòng riêng biệt, thuận lợi cho việc đọc payload từ phía mô-đun kiểm thử. Ứng dụng chưa hỗ trợ việc sửa đổi cấu hình kiểm thử từ phía UI.

6.1.5 Mô-đun kiểm thử

Đề mục này trình bày chi tiết quá trình hiện thực mô-đun kiểm thử qua 4 bước đã được thiết kế ở **Chương 5**. Mô-đun này mô phỏng lại một phần chức năng `Intruder` của Burp Suite với chiến thuật kiểm thử mặc định `Sniper`. Dữ liệu đầu vào của toàn bộ mô-đun kiểm thử gồm có cấu hình kiểm thử mặc định, các tập tin payload, thông tin của yêu cầu kiểm thử (bao gồm HTTP request mẫu và các loại lỗ hổng cần kiểm thử). Đầu ra của mô-đun này là danh sách các lỗ hổng mà điểm cuối của ứng dụng web mục tiêu mắc phải kèm theo các payload phát hiện được lỗ hổng (nếu có). Một yêu cầu kiểm thử sẽ có ba trạng thái ứng với quá trình tạo yêu cầu (`Submitted`), đang tiến hành kiểm thử (`Processing`) và đã hoàn tất kiểm thử (`Completed`). Dữ liệu đầu vào trước tiên được đưa đến mô-đun phân tích request mẫu. Để mô tả quá trình hiện thực của mô-đun kiểm thử, chúng tôi mô phỏng việc kiểm thử request mẫu trong Đoạn mã 6.5 với lỗ hổng LFI.

Mô-đun phân tích request mẫu (`requestParser`) đầu tiên sẽ tìm những vị trí cần chèn payload trong chuỗi request mẫu, được đánh dấu bằng một hoặc nhiều cặp "`\xa7`". Chiến thuật `Sniper` tương ứng với việc thay lần lượt từng payload vào mỗi vị trí một

lần. Đoạn mã 6.8 dưới đây tách request mẫu ra thành một danh sách các khuôn mẫu (pattern) để chèn payload vào, sao cho mỗi khuôn mẫu chỉ chứa một cặp "\xa7".

```
let patternRegex = /\\xa7.*?\\xa7/g;
const getPatternFromRequest = (req) => {
  try {
    let retList = [], match, currReq = req;
    let count = req.match(patternRegex).length;
    if (count === 1) retList.push(req, '');
    else {
      retList.push(req);
      while (match = patternRegex.exec(currReq)) {
        currReq = currReq.replace(match[0], match[0].replace(/\\xa7/g,
    ''));
        retList.push(currReq);
      }
    }
    return retList.splice(0, retList.length - 1);
  } catch (ex) {
    console.log("============> components => request => requestParser =>
    getPatternFromRequest => exception: ", ex);
    return [];
  }
}
```

Đoạn mã 6.8: Trích xuất khuôn mẫu từ request mẫu

Sau khi có các khuôn mẫu này, mô-đun phân tích tiếp tục đọc tập tin chứa payload tương ứng với loại lỗ hổng cần kiểm thử. Đường dẫn đến tập tin payload được chứa trong thuộc tính `payloadFile` của cấu hình, tương ứng với loại lỗ hổng cần kiểm thử. Đoạn mã 6.9 sau sẽ đọc tập tin ở đường dẫn trong tham số `file` để lấy ra danh sách payload này. Đối với những đoạn mã cần dùng đi dùng lại nhiều lần ở nhiều mô-đun khác nhau trong mã nguồn, ví dụ như các thao tác trên tập tin, escape/unescape các chuỗi và payload, định dạng HTTP response API trả về, các mô-đun kiểm thử (fuzzers),... chúng tôi sẽ phân loại

trong thư mục `components` ở thư mục gốc mã nguồn backend để tiện cho việc sửa đổi, quản lý và tái sử dụng.

```
const fs = require('fs');
const path = require('path');
const newline = /\r\n|\r|\n/g;
const readPayloadFile = (file) => {
  let filePath = path.join(__dirname, '../../payloads/' + file);
  return fs.readFileSync(filePath, 'utf8').toString().split(newline);
}
```

Đoạn mã 6.9: Dọc danh sách payload từ tập tin

Sau khi có danh sách khuôn mẫu và payload, ta lần lượt thay từng payload vào mỗi khuôn mẫu, đẩy vào một danh sách các request mẫu đã chứa payload, chuyển đến mô-đun xây dựng request kiểm thử. Mỗi phần tử của một danh sách này gồm thành phần chính là một chuỗi dạng đối tượng JSON chứa các thành phần của request mẫu kèm theo payload và một vài thông tin phục vụ quá trình kiểm thử. Danh sách này sau đó được gửi đến mô-đun xây dựng HTTP request kiểm thử để dựng thành một request kiểm thử hoàn chỉnh và gửi đến ứng dụng web mục tiêu. Đoạn mã 6.10 dưới đây là ví dụ về một phần tử trong danh sách gồm các trường được mô tả như sau.

```
{
  normalReq: '0',
  vulnId: '1',
  payload: '..//etc/passwd',
  req: '{"url":"http://testphp.vulnweb.com:80/showimage.php?
   file=..//etc/passwd",
   "cookies":"","headers":{"User-Agent":"Mozilla/5.0 (X11; Ubuntu; Linux
   x86_64; rv:74.0)Gecko/20100101
   Firefox/74.0","Accept":"text/html,application/xhtml+xml,
   application/xml;q=0.9,image/webp,*/*;q=0.8",
   "Accept-Language":"en-US,en;q=0.5","Accept-Encoding":"gzip,deflate",
   "DNT":"1","Connection":"close",
   "Upgrade-Insecure-Requests":"1"},"method":"get"}',
```

```
6   timeout: 0
7 }
```

Đoạn mã 6.10: Ví dụ về một phần từ của danh sách request mẫu chứa payload

- **normalReq** chỉ ra vị trí chính xác của cặp "\xa7" bị thay thế bởi payload trong request mẫu.
- **vulnId** chứa ID của loại lỗ hổng cần kiểm thử.
- **payload** chứa payload kiểm thử được dùng trong request này.
- **req** chứa request mẫu có chứa payload ở dạng chuỗi.
- **timeout** chứa thời gian tối đa đợi ứng dụng web mục tiêu trả về response đối với request này.

Mô-đun xây dựng HTTP request kiểm thử (`requestBuilder`) sử dụng thư viện Request[1] của Node.js hỗ trợ dựng HTTP request kiểm thử, gửi đến ứng dụng web mục tiêu và nhận response tương ứng trả về. Mô-đun này đầu tiên sẽ phân tích (parse) chuỗi `req` trong request mẫu thành đối tượng JSON, cung cấp những tham số cần thiết để dựng request. Đoạn mã 6.11 sau là một ví dụ về các tham số này.

```
1 {
2   method: 'get',
3   url: 'http://testphp.vulnweb.com:80/showimage.php?file=..//etc/passwd',
4   headers: {
5     'User-Agent': 'Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:74.0)
    Gecko/20100101 Firefox/74.0',
6     Accept:
    'text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,
    */*;q=0.8','Accept-Language': 'en-US,en;q=0.5',
7     'Accept-Encoding': 'gzip, deflate',
8     DNT: '1',
9     Connection: 'close',
```

[1]Nguồn: https://www.npmjs.com/package/request

```
10      'Upgrade-Insecure-Requests': '1'
11    },
12    form: undefined,
13    gzip: true,
14    resolveWithFullResponse: true,
15    time: true,
16    simple: false,
17    followAllRedirects: true
18  }
```

Đoạn mã 6.11: Ví dụ về tham số xây dựng request kiểm thử

Các tham số này một phần được kế thừa từ request mẫu, bao gồm `url`, `method`, `headers`, `cookies`, `data` (form - nội dung kèm theo những request POST). Ngoài ra còn có một số tham số quan trọng khác như sau.

- **resolveWithFullResponse** nhằm trả về đầy đủ các thuộc tính của đối tượng response trả về, phục vụ cho việc kiểm thử.
- **gzip** chỉ thị tự động thêm kiểu nén dữ liệu thông dụng `gzip` vào trường `Accept-Encoding` của headers nếu chưa có.
- **time** chỉ thị ghi lại thời gian phản hồi của máy chủ (elapsed time) kể từ thời điểm gửi request kiểm thử, phục vụ cho việc kiểm tra lỗ hổng time-based SQLI.
- **timeout** là thời gian chờ response phản hồi tối đa, đơn vị tính là mili giây, nếu thời gian phản hồi của máy chủ lâu hơn tham số này thì mã lỗi `ESOCKETTIMEDOUT` sẽ được đẩy ra. Tham số này kế thừa từ thuộc tính `time` của cấu hình kiểm thử lỗ hổng tương ứng. Trong trường hợp kiểm thử lỗ hổng time-based SQLI, thuộc tính này giúp tiết kiệm thời gian kiểm thử rất nhiều, tương tự như thuộc tính timeout trong thiết lập của Burp Suite. Các payload khai thác lỗi này thường kéo thời gian thực thi request ở phía máy chủ web mục tiêu lên đến 20-60 giây, trong khi chỉ cần khoảng 10 giây như cấu hình kiểm thử là ta đã đủ cơ cở đã kết luận điểm cuối này mắc lỗi.

Sau khi dựng và gửi request kiểm thử có kèm payload đến ứng dụng web mục tiêu, chúng

tôi lọc bớt một vài thuộc tính của response trả về, đồng thời thêm một vài thuộc tính phục vụ cho việc xác nhận lỗi của mô-đun kiểm thử (`fuzzers`). Đoạn mã 6.12 là một ví dụ response trả về, bao gồm những thuộc tính như mô tả dưới đây.

```
{
  payload: '..//etc/passwd',
  vulnId: '1',
  body: '\nWarning: fopen(): Unable to access ..//etc/passwd in
   /hj/var/www/showimage.php on line 7\n\nWarning:
   fopen(..//etc/passwd): failed to open stream: No such file or
   directory in /hj/var/www/showimage.php on line 7\n\nWarning:
   fpassthru() expects parameter 1 to be resource, boolean given in
   /hj/var/www/showimage.php on line 13\n',
  statusCode: 200,
  elapsedTime: 353,
  contentLength: 323,
  headers: {
    server: 'nginx/1.4.1',
    date: 'Sat, 02 May 1970 19:09:17 GMT',
    'content-type': 'image/jpeg',
    'transfer-encoding': 'chunked',
    connection: 'close',
    'x-powered-by': 'PHP/5.3.10-1~lucid+2uwsgi2'
  }
}
```

Đoạn mã 6.12: Ví dụ về response phản hồi từ ứng dụng web mục tiêu

- **payload** là payload đi kèm theo request kiểm thử tương ứng.
- **vulnId** là ID của loại lỗ hổng đang kiểm thử.
- **statusCode** là mã trạng thái HTTP của response trả về.
- **elapsedTime** là thời gian trả về response từ khi gửi request kiểm thử. Nếu có mã lỗi

`ESOCKETTIMEDOUT` được đẩy ra trong quá trình chờ response thì thuộc tính này sẽ mang giá trị của tham số `timeout` khi xây dựng request.

- **contentLength** là độ dài của phần thân (body) của response.
- **headers** chứa các headers của response.

Các mô-đun kiểm thử (`fuzzers`) sẽ đảm nhiệm việc so trùng các chuỗi và biểu thức chính quy trên trường `body` của response trả về, cộng với việc kiểm tra thời gian phản hồi ở trường `elapsedTime` của response đối với trường hợp kiểm thử lỗ hổng time-based SQLI. Đối với công đoạn so trùng, chúng tôi kiểm tra các thuộc tính `match, matchFile, regex` trong cấu hình kiểm thử. Các thuộc tính này chứa các chuỗi và biểu thức chính quy cần so, nếu có thì tiến hành tìm kiếm các chuỗi khớp trong body của response, tương tự với thuộc tính `time` trong trường hợp cần so sánh thời gian thực thi request trong quá trình kiểm thử lỗ hổng time-based SQLI. Đoạn mã 6.13 hiện thực phần so khớp và kiểm tra thời gian phản hồi dựa trên cấu hình kiểm thử.

```
let retList = [], timebasedResult = false;
let grepObj = resp.body;
if (config['match'] != '') {
  for (var i in config['match']) {
    if (grepObj.indexOf(config['match'][i]) >= 0)
      retList.push(config['match'][i]);
  }
}
if (config['matchFile'] != '') {
  let matchList = [], tmpList;
  for (var idx in config['matchFile']) {
    tmpList = readPayloadFile(config['matchFile'][idx]);
    matchList = matchList.concat(tmpList);
  }
  for (var i in matchList)
    if (grepObj.indexOf(matchList[i]) >= 0)
      retList.push(matchList[i]);
}
```

```
19 if (config['regex'] != '') {
20   for (var i in config['regex']) {
21     let matchList = grepObj.match(config['regex'][i]);
22     if (matchList && matchList.length > 0) {
23       retList.push(matchList.toString());
24     }
25   }
26 }
27 if (config['time'] != '')
28   timebasedResult = resp.elapsedTime >= config['time'] * 1000;
29
30 return {
31   matchResult: retList.length === 0,
32   timebasedResult
33   matchList: retList
34 };
```

Đoạn mã 6.13: Hiện thực các mô-đun kiểm thử

Kết quả kiểm thử ứng với mỗi payload sẽ gồm trường `matchResult`, `timebasedResult` lần lượt ứng với kết quả so trùng và kiểm tra thời gian thực thi request. Những chuỗi khớp được đẩy vào mảng `matchList` như bằng chứng có lỗ hổng để lưu trữ vào cơ sở dữ liệu. Song song với quá trình kiểm thử, backend của ứng dụng cũng xuất ra kết quả kiểm thử trên terminal theo từng dòng, mỗi dòng ứng với một payload như Hình 6.2 sau.

```
[fuzzController] executeFuzzRequest... requestId: 10
[fuzzService] executeFuzzRequest... fuzzing request id: 10
Start fuzzing for Local File Inclusion...
  Status         Code   Length  Time    Payload
 [failed]        200    323     644     ..//etc/passwd
 [passed]        200    845     332     ../..//etc/passwd       root:x
 [failed]        200    413     332     ../../..//etc/passwd
Done fuzzing for Local File Inclusion!
 Done fuzzing request ID  10  Result ID is  8
```

Hình 6.2: Kết quả kiểm thử trên terminal backend

Những payload phát hiện được lỗi (có xuất hiện các chuỗi khớp trong body hoặc thời gian

thực thi request thỏa cấu hình kiểm thử) sẽ được đổi màu để làm nổi bật trên terminal như Hình 6.2. Trong trường hợp trên, payload "`../..//etc/passwd`" khiến máy chủ web mục tiêu trả về response có body chứa chuỗi "`root:x`", dấu hiệu khai thác thành công lỗ hổng LFI. Mỗi dòng kết quả này bao gồm các thành phần sau.

- **Kết quả kiểm thử** được đặt trong cặp ngoặc vuông, có thể là `[passed]` trong trường hợp payload khai thác thành công lỗ hổng hoặc `[failed]` trong trường hợp payload không khai thác thành công.
- **Mã trạng thái HTTP** của response trả về, thường là `200` trong trường hợp request được gửi thành công hoặc các mã lỗi `401`, `404`, `500`,... khi có sự cố trong quá trình xử lí request ở ứng dụng web mục tiêu.
- **Độ dài nội dung** của response trả về, **thời gian xử lí** của request tính theo giây, **payload** tương ứng được gửi theo request kèm theo những chuỗi khớp với cấu hình kiểm thử (nếu có).

Mô-đun lưu trữ kết quả kiểm thử (`updateFuzzingLog`) sẽ lưu trữ những payload kèm theo loại lỗ hổng phát hiện được dựa trên kết quả kiểm thử trên. Những kết quả này được lưu lại dưới dạng chuỗi vào cơ sở dữ liệu theo cấu trúc như Đoạn mã 6.14 sau.

```
{
  "1": [
    {
      payload: '../..//etc/passwd',
      payloadIdx: '0',
      matchList: ['root:x'],
      timebased: false
    },
    ...
  ],
  ...
}
```

Đoạn mã 6.14: Cấu trúc kết quả kiểm thử

Kết quả kiểm thử sẽ được phân loại theo ID của mỗi loại lỗ hổng. Đối với mỗi payload phát hiện được lỗ hổng này, chúng tôi lưu trữ đầy đủ kết quả so trùng, thời gian thực thi request và vị trí chèn payload trên HTTP request mẫu, phục vụ việc trả kết quả về hiển thị trên UI và hiện thực chức năng tái hiện request/response kiểm thử trong tương lai.

6.2 Giao diện người dùng

Giao diện người dùng của ứng dụng webfuzzer được xây dựng dựa trên bản mẫu (template) React Reduction [11]. Bản mẫu này được viết trên React.js với sự hỗ trợ của thư viện reactstrap[2], hiện thực sẵn một số thành phần như thanh điều hướng, các nút bấm, ô lựa chọn, bảng biểu, kiểu chữ,... cũng như giao diện mặc định đẹp mắt, dễ dàng chỉnh sửa theo nhu cầu lập trình. Công việc của chúng tôi là hiện thực và thiết lập luồng hoạt động của ba thành phần chính của UI, đảm nhiệm được việc gọi các API backend cung cấp và xử lí kết quả trả về, hiển thị được những thông tin cần thiết đồng thời đáp ứng nhu cầu kiểm thử của người dùng như yêu cầu và thiết kế UI đã đề ra ở **Chương 5**.

React.js là một khung thức lập trình giao diện người dùng hướng thành phần (component-based), mỗi trang web và các thành phần nhỏ hơn trong trang đa phần đều kế thừa các class có sẵn của React.js và được hiện thực dưới dạng các component. Mỗi component có tập các trạng thái riêng (state) và tập những thuộc tính được truyền vào (props). Bản thân một component trong React.js sẽ chỉ thay đổi được state của nó bằng phương thức `setState` chứ không thay đổi được giá trị của những props mà nó được nhận. Mỗi khi hàm `setState` thực hiện xong, phương thức `render` của component đó sẽ được khởi chạy, tải lại nội dung hiển thị của component theo những giá trị state mới mà không làm mới lại cả trang web. Phương thức `render` của các component trả về giá trị là các JSX, gồm những đoạn mã HTML hoặc các component lồng với JavaScript đặc trưng của khung thức này, tương ứng với nội dung hiển thị của component đó ở giao diện người dùng.

Trước tiên, trong quá trình hiện thực UI của ứng dụng, chúng tôi sử dụng thư viện Axios[3] để tương tác với các API ở backend ứng dụng webfuzzer. Thư viện này hỗ trợ gửi và nhận các thông điệp HTTP một cách đơn giản thông qua việc cung cấp các tham số cần thiết như đường dẫn API, phương thức HTTP kèm theo body và các tham số nếu có. Những

[2]Nguồn: https://reactstrap.github.io/
[3]Nguồn: https://www.npmjs.com/package/axios

lời gọi API này nằm trong các đoạn mã xử lí sự kiện của JSX khi người dùng tương tác với giao diện và khi tải nội dung trang. Đoạn mã 6.15 sau hiện thực việc gọi API và lấy ra kết quả trả về trong mã nguồn UI.

```
import axios from 'axios';
const BASE_URL = require('./globalConfig').BASE_URL;
const callApi = async (endpoint, method = 'get', body) => {
  try {
      let { data } = await axios({
        method: method,
        url: `${BASE_URL}${endpoint}`,
        data: body
      });
      return data;
  } catch (err) {
      console.log(err);
      return null;
  }
}
```

Đoạn mã 6.15: Gọi API từ mã nguồn UI

Để điều hướng hai trang web chính của UI, chúng tôi sử dụng các component `Switch`, `Route` có sẵn của React.js trong Đoạn mã 6.16 sau.

```
<Switch>
  <MainLayout breakpoint={this.props.breakpoint}>
    <React.Suspense fallback={<PageSpinner />}>
      <Route exact path="/" component={DashboardPage} />
      <Route exact path="/result" component={ResultPage} />
    </React.Suspense>
  </MainLayout>
  <Redirect to="/" />
</Switch>
```

Đoạn mã 6.16: Điều hướng các trang web trên UI của webfuzzer

Giả sử UI của ứng dụng webfuzzer được triển khai ở cổng 3000 của máy chủ có địa chỉ IP `61.28.235.183`. Khi đó, địa chỉ web `http://61.28.235.183:3000/` sẽ dẫn đến component trang bảng điều khiển và đường dẫn `http://61.28.235.183:3000/result` dẫn đến component trang kết quả kiểm thử của ứng dụng.

6.2.1 Trang bảng điều khiển

Đầu tiên khi tải trang bảng điều khiển xuống, người dùng cần có sẵn danh sách các request mẫu và loại lỗ hổng cần kiểm thử để chọn tạo yêu cầu kiểm thử. Để làm được điều này, ta gọi các API `GET /` và `GET /target/configs` trong phương thức `componentDidMount` của component, phương thức này luôn chạy đầu tiên mỗi khi component chứa nó được gọi đến. Đoạn mã 6.17 sau là phương thức `componentDidMount` của component trang bảng điều khiển, lấy các danh sách cần thiết từ API và lưu vào state của component này.

```
1  componentDidMount = async () => {
2    let apiList = [this.getListEndpoints(), this.getVulneTypes()]
3    let [res1, res2] = await Promise.all(apiList);
4  }
5  getListEndpoints = async (limit = 5, offset = 0) => {
6    ...
7    let endpointList = await callApi(endpoints.getAllEndpoinst(limit,
       offset), 'get', null);
8    if (endpointList && endpointList.results) {
9      let list = endpointList.results.endpointList;
10     this.setState({
11       endpointsList: list,
12       totalRecord: endpointList.results.total
13     });
14   }
15   ...
16 }
17 getVulneTypes = async () => {
18   ...
19   let { results } = await callApi(endpoints.getListVulnes, 'get', null);
```

```
20    let arr = [];
21    if (results) {
22      Object.keys(results).forEach((key, idx) => {
23        let vulne = {
24          id: String(key),
25          label: results[key].label,
26          check: false,
27          type: key === '6' ? 'common' : 'auto'
28        }
29        arr.push(vulne);
30      })
31    }
32    this.setState({ vulneTypes: [...arr] });
33    ...
34  }
```

Đoạn mã 6.17: Hiện thực các thành phần của trang bảng điều khiển

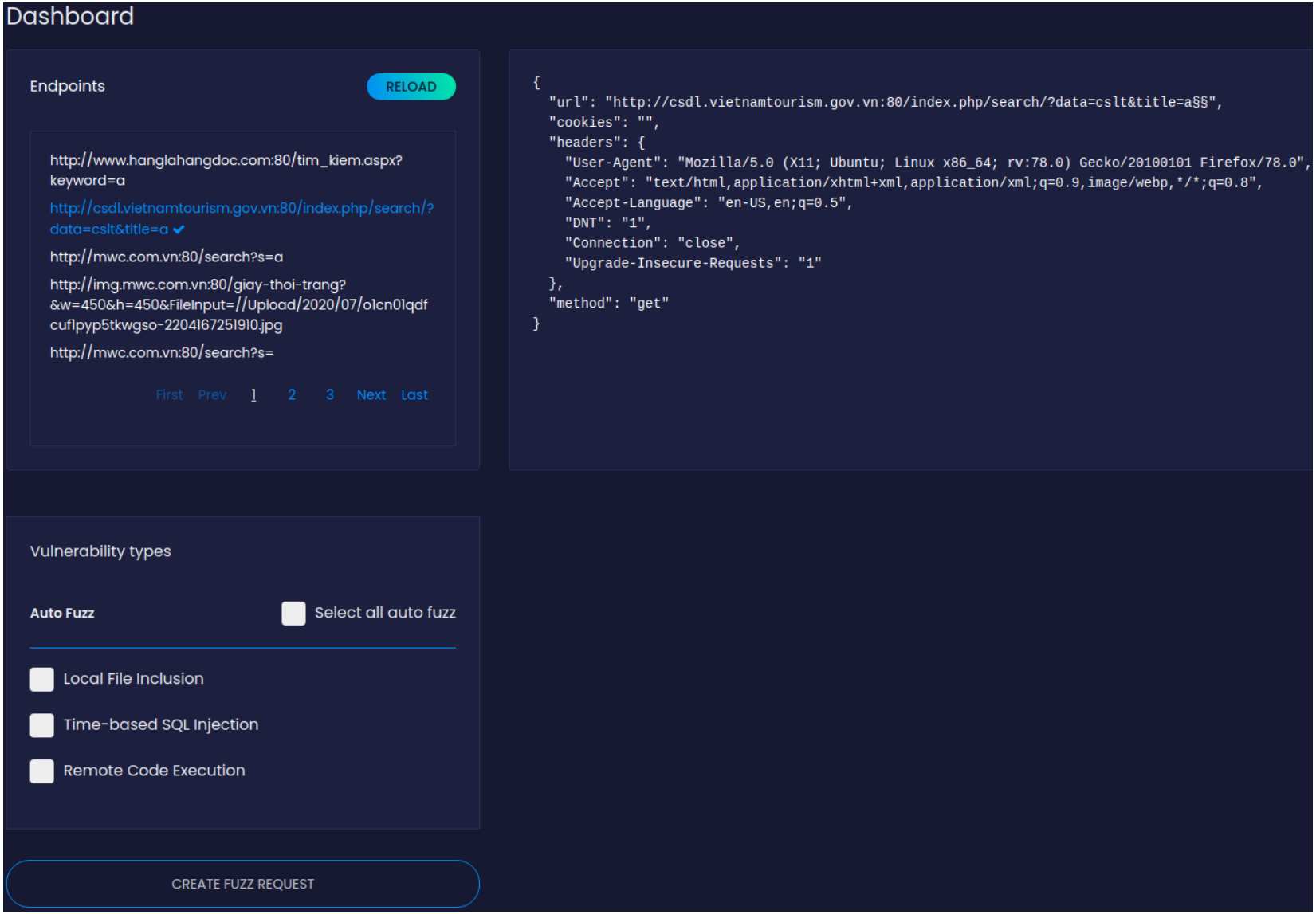
Theo thiết kế giao diện đã đề ra ở **Chương 5**, trang này cần ba khung để hiển thị danh sách request mẫu, chi tiết request mẫu, khung chọn lỗ hổng cần kiểm thử và một nút nhấn để tạo yêu cầu kiểm thử. Phương thức `render` của component này căn chỉnh các khung vuông vức bằng các thành phần (component) `Row, Column` trong Layout của thư viện reactstrap cùng với CSS như Đoạn mã 6.18 sau.

```
1  <Page
2    className="DashboardPage"
3    title="Dashboard"
4  >
5    <Row>
6      <Col lg="4">
7        <EndpointsComponent
8          endpointsList={this.state.endpointsList}
9          selectEndpoint={this.selectEndpoint}
10         endpointSelected={this.state.endpointSelected}
```

```
11          getListEndpoints={this.getListEndpoints}
12          loading={this.state.loading}
13          totalRecord={this.state.totalRecord}
14        />
15      </Col>
16      <Col lg="8">
17        <BaseRequestComponent
      selectedEndpoint={this.state.endpointSelected} />
18      </Col>
19    </Row>
20    <Row>
21      <Col lg="4">
22        <ListVulnes
23          listVulnes={this.state.vulneTypes}
24          selectVulne={this.selectVulne}
25          checkedAllAutoFuzz={this.state.checkedAllAutoFuzz}
26          toogleAllAutoFuzz={this.toogleAllAutoFuzz}
27        />
28        <div className="btn-submit">
29          {this.state.loadingSubmit ?
30            <AwaitingComponent /> :
31            <button
32              disabled={!this.state.endpointSelected.Id}
33              onClick={() => this.submitFuzzRequest()}>Create fuzz request
34          </button>}
35        </div>
36      </Col>
37      <Col lg="8"></Col>
38    </Row>
39  </Page>
```

Đoạn mã 6.18: Phương thức `render` của trang bảng điều khiển

Khung chứa danh sách request mẫu tương ứng với component `EndpointsComponent` và khung hiển thị nội dung request mẫu tương đương ứng component `BaseRequestComponent`. Component `EndpointsComponent` sẽ lưu lại ID của request mẫu đang được chọn vào state `endpointSelected`, đồng thời component `BaseRequestComponent` sẽ hiển thị nội dung của request mẫu có ID đó ở bên cạnh. Danh sách các request mẫu sẽ được phân thành nhiều trang để đảm bảo bố cục trang web, mỗi trang gồm 5 request mẫu, việc hiện thực cơ chế phân trang này sẽ được để cập ở **Đề mục 6.2.3**. Mỗi khung trên được đặt trong một component `Col` riêng và cả hai nằm trong cùng một component `Row` trong mã nguồn để nằm ngang hàng nhau trên UI như Hình 6.3 sau.



Hình 6.3: Giao diện trang bảng điều khiển ở UI ứng dụng webfuzzer

Tương tự, khung chọn loại lỗ hổng kiểm thử tương ứng với component `ListVulnes` và nút tạo yêu cầu kiểm thử ứng với thẻ `<div className="btn-submit">` trong mã nguồn. Các thuộc tính `lg="4"` và `lg="8"` của component `Col` mô tả chiều rộng của các khung trên UI. Hàm xử lí sự kiện `onClick` nhấn nút này là hàm `this.submitFuzzRequest()`, có chức năng gọi đến API `POST /target` để tạo yêu cầu kiểm thử dựa trên request mẫu và những loại lỗ hổng đã chọn.

6.2.2 Trang kết quả kiểm thử

Tương tự như trang bảng điều khiển, phương thức `componentDidMount` của trang sẽ gọi API

`GET /target/list` để lấy danh sách các yêu cầu kiểm thử cần hiển thị. Bố cục chính của trang là một bảng gồm nhiều hàng, mỗi hàng bao gồm thông tin cơ bản của một yêu cầu kiểm thử và một nút xổ xuống thông tin chi tiết của yêu cầu đó, và một khung tìm kiếm yêu cầu kiểm thử theo URL của request mẫu. Chúng tôi sử dụng thẻ `table` cùng với các thẻ con `thead, tr, th` để hiện thực một bảng với các hàng và cột tương tự ngôn ngữ HTML. Ngoài ra trang này còn cung cấp một khung tìm kiếm yêu cầu kiểm thử theo URL của request mẫu bằng API `GET /target/search`. Người dùng cũng có thể lọc ra những yêu cầu kiểm thử có lỗ hổng từ cột `Vulnerable`. Danh sách các yêu cầu kiểm thử được chứa trong thuộc tính state `requestList` nên khi người dùng tìm kiếm hoặc lọc làm thay đổi danh sách này, component của trang sẽ được hiển thị lại. Danh sách các yêu cầu kiểm thử này cũng sẽ được phân trang.

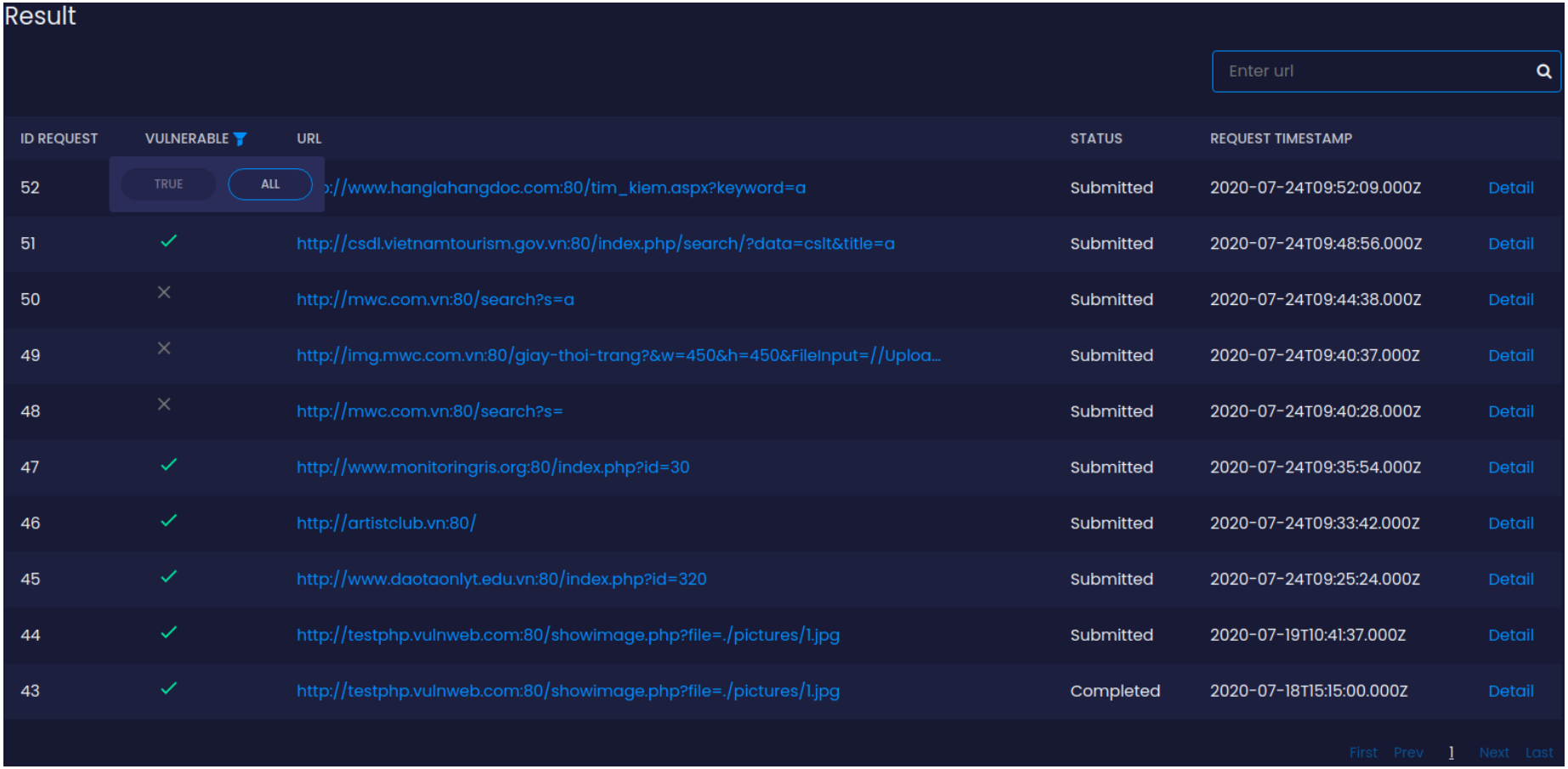
Khung tìm kiếm yêu cầu kiểm thử theo URL của request mẫu được đặt trên góc phải của bảng yêu cầu kiểm thử. Khung này gồm một thẻ `input` để người dùng nhập vào một phần của URL cần tìm kiếm và một nút bấm bên cạnh để gửi yêu cầu tìm kiếm. Thẻ `input` sẽ liên tục gọi hàm xử lí sự kiện `onChangeInput` mỗi khi có thay đổi từ người dùng nhập vào. Sau đó khi người dùng nhấn nút tìm kiếm, hàm xử lí sự kiện `onSearchSubmit` sẽ gọi API `GET /target/search` để trả về danh sách các yêu cầu kiểm thử thỏa điều kiện tìm kiếm.

```
1 <div>
2   <form onSubmit={(e) => this.onSearchSubmit(e)}>
3     <input placeholder="Enter url" onChange={(event) =>
```

```
    this.onChangeInput(event)} />
4     <button type="submit">
5       <i className="fa fa-search" aria-hidden="true"></i>
6     </button>
7   </form>
8 </div>
```

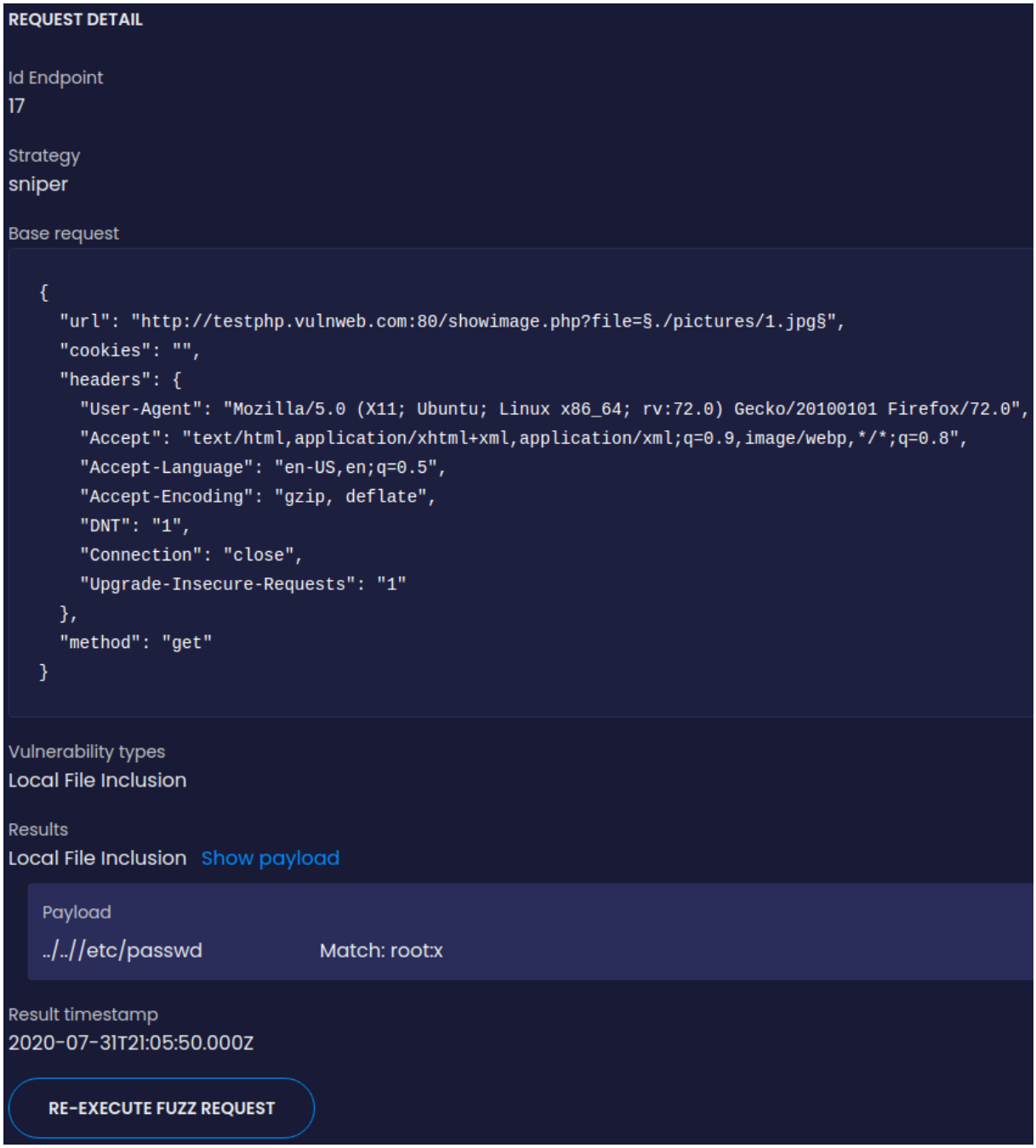
Đoạn mã 6.19: Hiện thực khung tìm kiếm yêu cầu kiểm thử theo URL

Giao diện chính của trang kết quả kiểm thử gồm hai thành phần trên được mô tả như Hình 6.4 sau.



Hình 6.4: Giao diện trang kết quả kiểm thử ở UI ứng dụng webfuzzer

Các thông tin cơ bản của một yêu cầu kiểm thử bao gồm ID, đường dẫn URL, thời điểm tạo, trạng thái và kết quả là có lỗ hổng hay không và các payload phát hiện được nếu có. Các thông tin này được truy vấn từ API `GET /target` theo ID của yêu cầu kiểm thử và hiển thị trên UI như Hình 6.5 sau.



Hình 6.5: Thông tin chi tiết của một yêu cầu kiểm thử

Chúng tôi hiện thực khung thông tin chi tiết của một yêu cầu kiểm thử trong component `RequestDetailComponent`. Mỗi thông tin hiển thị trong component này được đặt trong

một thẻ `div` lần lượt từ trên xuống. Khung nội dung của request mẫu được chúng tôi tái sử dụng định dạng CSS từ khung nội dung request mẫu ở trang bảng điều khiển. Khi người dùng nhấn nút "`Show payload`" bên cạnh loại lổ hổng ở khung kết quả kiểm thử, những payload phát hiện được lỗ hổng đó sẽ hiện ra. Chúng tôi chỉ lọc ra payload, những chuỗi so trùng và kết quả kiểm tra thời gian thực thi request kiểm thử (nếu có) trong kết quả kiểm thử tương ứng để hiển thị. Cuối cùng là nút (tái) thực thi một yêu cầu kiểm thử, nút này sẽ gọi đến API `GET /fuzz` thông qua hàm xử lí sự kiện `executeFuzzRequest` khi người dùng nhấn nút. Các thành phần trên của component `RequestDetailComponent` được hiện thực thông qua Đoạn mã 6.20 sau.

```
1  <div className="detail-item-title">Base request</div>
2  <div className="detail-item-content"><pre
       className="base-request">{JSON.stringify(JSON.parse(baseRequest),
       null, 2).replace(/\\\\xa7/gi, '§')}</pre></div>
3  ...
4  <div className="result-content-item-title">Payload</div>
5    {element.value ? element.value.map((item, idx) => (
6      <div key={idx} className="result-content-item">
7        <div className="result-content-item-content">{item.payload}
8          <span style={{ marginLeft: '3rem' }} >
9            {item.timebased ? 'Timebased: true' : null}
10         </span>
11         <span style={{ marginLeft: '3rem' }} >
12           {item.matchList.length > 0 ? 'Match: ' +
       item.matchList.toString() : null}
13         </span>
14       </div>
15     </div>
16   )) : ''}
17 </div>
18 ...
19 <div className="detail-item">
20   <button disabled={this.checkDisableButton(requestId)} onClick={async
```

```
    () => this.executeFuzzRequest()}>{status == 'Completed' || status ==
    'Processing' ? 're-execute' : 'Execute'} fuzz request</button>
</div>
```

Đoạn mã 6.20: Hiện thực một số thành phần của component RequestDetailComponent

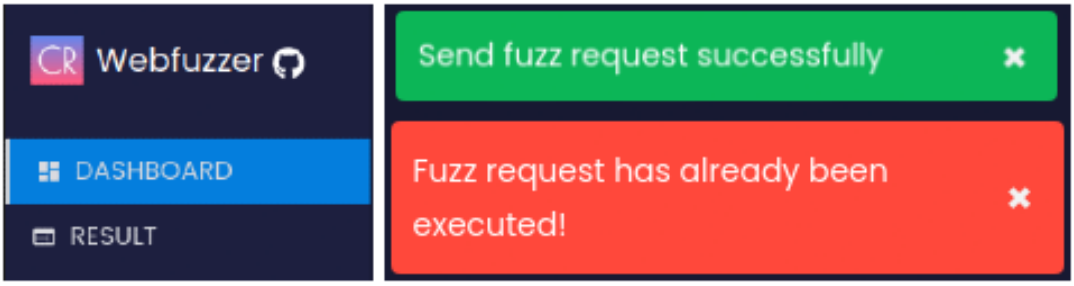
6.2.3 Thanh điều hướng và các thành phần khác

Thanh điều hướng của UI được hiện thực bằng các component `Nav`, `Navbar`, `NavItem`, `NavLink` của thư viện reactstrap. Thuộc tính `vertical` của component `Nav` sẽ cố định thanh điều hướng dọc theo lề trái của trang web. Hai trang bảng điều khiển, kết quả kiểm thử được đưa vào thành hai thẻ trên thanh điều hướng dưới dạng các component `NavLink` như Đoạn mã 6.21 sau. Kết quả hiện thực thanh điều hướng và thông báo đẩy được thể hiện trong Hình 6.6.

```
const navItems = [
  { to: '/', name: 'dashboard', exact: true, Icon: MdDashboard },
  { to: '/result', name: 'result', exact: false, Icon: MdWeb },
];
...
<Nav vertical>
  {navItems.map(({ to, name, exact, Icon }, index) => (
    <NavItem key={index} className={bem.e('nav-item')}>
      <NavLink
        id={`navItem-${name}-${index}`}
        className="text-uppercase"
        tag={NavLink}
        to={to}
        activeClassName="active"
        exact={exact}
      >
        <Icon className={bem.e('nav-item-icon')} />
        <span className="">{name}</span>
      </NavLink>
```

```
20      </NavItem>
21    ))}
22  </Nav>
```

Đoạn mã 6.21: Các thuộc tính props truyền vào component Nav



Hình 6.6: Thanh điều hướng và ví dụ thông báo đẩy trên giao diện người dùng

Các thông báo đẩy trên UI của webfuzzer có hai kiểu, thành công (success) hoặc thất bại (error) như Hình 6.6 trên. Định dạng của các kiểu này được thiết lập trong biến `NOTIFICATION_SYSTEM_STYLE` ở tập tin `src/utils/constants.js` trong thư mục gốc mã nguồn UI. Chúng tôi sử dụng component `NotificationSystem` của thư viện `react-notification-system` để hiện thực các thông báo đẩy này. Các props chính mà component này nhận bao gồm nội dung thông báo (`message`), kiểu thông báo (`level`) và định dạng kiểu (`style`) như Đoạn mã 6.22 sau.

```
1  this.notificationSystem.addNotification({
2    message: 'Send fuzz request successfully',
3    level: 'success',
4    position: 'tc'
5  });
6  ...
7  <NotificationSystem
8    dismissible={false}
9    ref={notificationSystem =>
10     (this.notificationSystem = notificationSystem)
11   }
12   style={NOTIFICATION_SYSTEM_STYLE}
```

```
13 />
```

Đoạn mã 6.22: Các thuộc tính props truyền vào component NotificationSystem

Để phân trang các danh sách request mẫu và thông tin yêu cầu kiểm thử, chúng tôi sử dụng component `Pagination` có sẵn trong thư viện `react-js-pagination`. Component này nhận vào các props như danh sách phần tử cần hiển thị, tổng số lượng phần tử và số lượng phần tử ở mỗi trang, kèm theo các đoạn text hiển thị ở thao tác chuyển trang như trong Đoạn mã 6.23 sau.

```
1  <Pagination
2    activePage={this.state.activePage}
3    itemsCountPerPage={this.state.limit}
4    totalItemsCount={this.state.totalRecord}
5    pageRangeDisplayed={5}
6    onChange={this.handlePageChange.bind(this)}
7    prevPageText={'Prev'}
8    nextPageText={'Next'}
9    firstPageText={'First'}
10   lastPageText={'Last'}
11 />
```

Đoạn mã 6.23: Các thuộc tính props truyền vào component Pagination

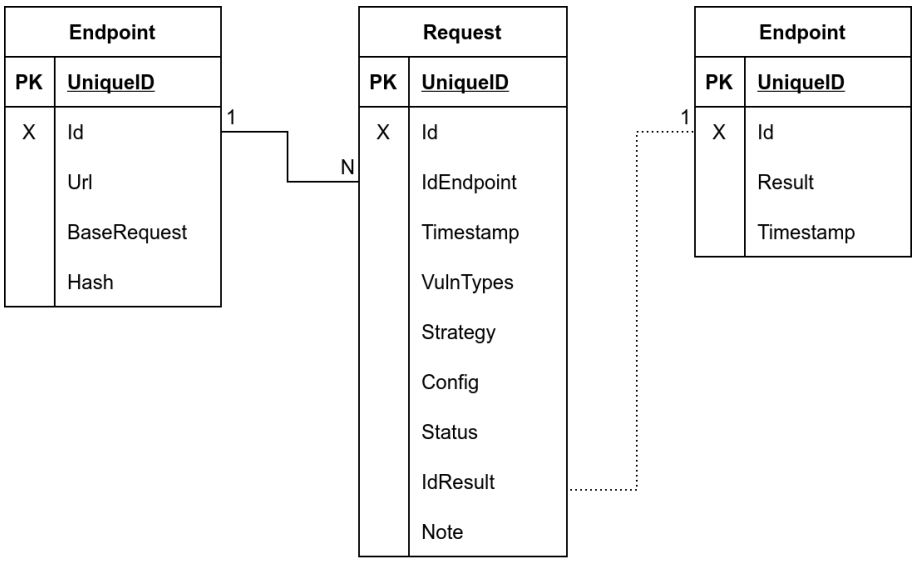
6.3 Cơ sở dữ liệu

Dựa vào thiết kế đã đề ra ở **Chương 5**, chúng tôi hiện thực cấu trúc cơ sở dữ liệu của ứng dụng webfuzzer gồm ba bảng chính. Bảng 6.2 dưới đây mô tả tên và chức năng của từng bảng trong lược đồ.

Bảng 6.2: Các bảng trong lược đồ quan hệ

Tên	Mô tả
Endpoint	Bảng Endpoint lưu lại các request mẫu mục tiêu được gửi đến máy chủ từ phần mở rộng Burp Suite
Request	Bảng Request lưu những yêu cầu kiểm thử một request mẫu trong bảng Endpoint của người dùng
Result	Bảng Result lưu kết quả kiểm thử chi tiết ứng với mỗi yêu cầu kiểm thử trong bảng Request

Hình 6.7 dưới đây mô tả quan hệ giữa các bảng trong lược đồ.



Hình 6.7: Sơ đồ mối quan hệ giữa các bảng trong lược đồ

Bảng `Endpoint` lưu trữ HTTP request mẫu được gửi từ phần mở rộng Burp Suite đến backend thông qua API `POST /`. Request mẫu đó chứa dưới dạng chuỗi dữ liệu JSON trong trường `BaseRequest`. Trường `Url` chứa riêng URL của điểm cuối ứng dụng web mục tiêu để thuận lợi trong việc lọc ra những yêu cầu kiểm thử của cùng một trang web thông qua API `POST /target/search`. Trường `Hash` chứa giá trị băm của chuỗi `BaseRequest`, đảm bảo không lưu hai request mẫu y hệt nhau gây dư thừa dữ liệu. Bảng `Request` chứa thông tin của các yêu cầu kiểm thử, trong đó trường `IdEndpoint` là khoá ngoại tham chiếu tới khoá chính `Id` của bảng `Endpoint` và trường `IdResult` là khoá ngoại tham chiếu tới khoá chính `Id` của bảng `Result` chứa kết quả kiểm thử (trong trường hợp có lỗ hổng). Mỗi bản ghi trong bảng `Request` chứa HTTP request mẫu gửi đến các điểm cuối của ứng dụng web mục tiêu, tập các lỗ hổng cần kiểm thử, trạng thái, và kết quả kiểm thử tương

ứng, bao gồm danh sách lỗ hổng của điểm cuối đó và các payload phát hiện được. Trường `BaseRequest` của bảng `Endpoint`, trường `VulnTypes` của bảng `Request` và `Result` của bảng `Result` là các trường có kiểu chuỗi, chứa các giá trị kiểu đối tượng JSON đã được chuỗi hóa như đã đề cập trong phần thiết kế kiến trúc cơ sở dữ liệu ở trên.

Chương 7

Kiểm định và đánh giá kết quả

Chương này chúng tôi trình bày kết quả sử dụng ứng dụng webfuzzer để kiểm thử một số ứng dụng web và so sánh trực tiếp với quá trình kiểm thử bằng chức năng `Intruder` của công cụ Burp Suite để đánh giá kết quả đạt được thông qua quá trình thiết kế và hiện thực ứng dụng.

7.1 Thiết lập kiểm thử ở Burp Suite

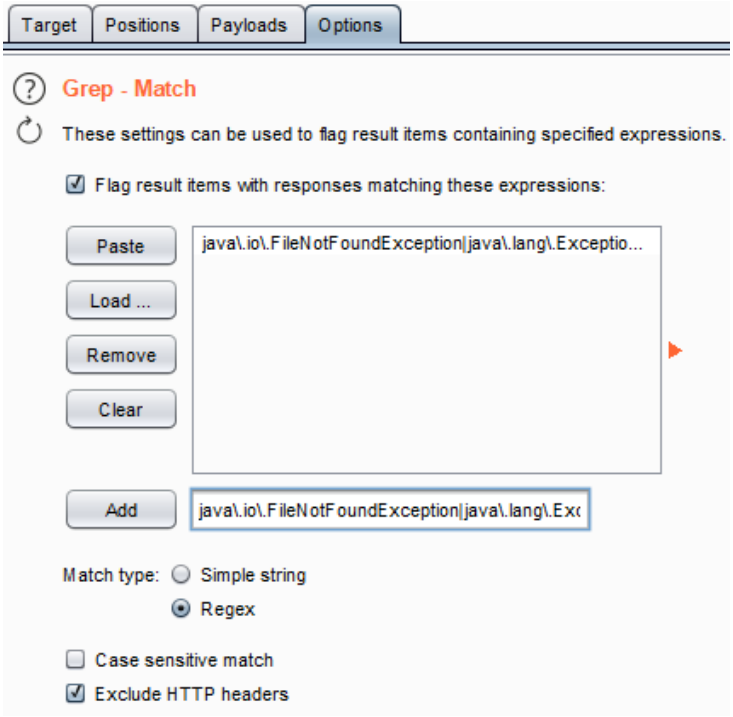
Ở ứng dụng Burp Suite, chúng tôi cần thiết lập thời gian timeout của các HTTP request kiểm thử, danh sách payload và các chuỗi, biểu thức chính quy so trùng trên response. Các thông số trên lần lượt tương ứng với các trường `time`, `payloadFile` và `match, matchFile, regex` trong cấu hình kiểm thử. Chức năng `Intruder` không cho phép điều chỉnh timeout riêng của các request trong chức năng này, chúng tôi phải thiết lập thời gian timeout toàn cục cho một dự án (project) của Burp Suite dưới dạng đối tượng JSON như Đoạn mã 7.1 sau, đơn vị tính là mili giây. Đây cũng là một sự tiện lợi khi sử dụng webfuzzer vì người dùng có thể dễ dàng điều chỉnh thông số này ở cấu hình kiểm thử tùy theo từng loại lỗ hổng.

```
1 "timeouts":{
2     "domain_name_resolution_timeout":300000,
3     "failed_domain_name_resolution_timeout":60000,
4     "normal_timeout":10000,
5     "open_ended_response_timeout":10000
```

```
6 },
```

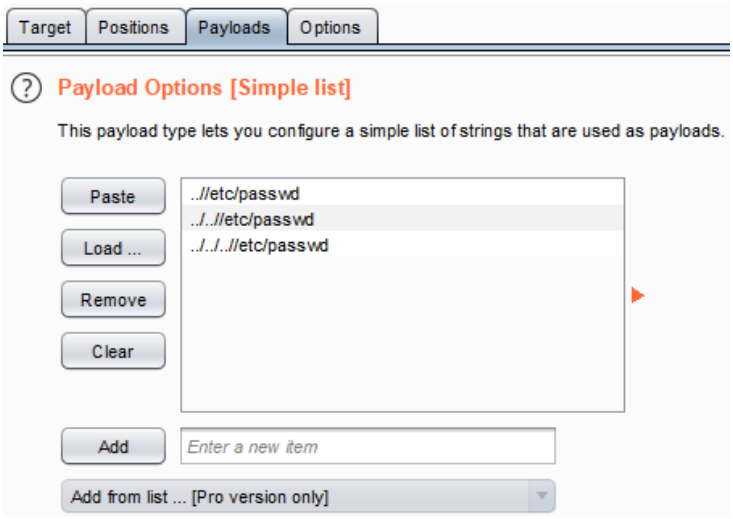
Đoạn mã 7.1: Thiết lập thời gian timeout của request kiểm thử ở Burp Suite

Các chuỗi và biểu thức chính quy so trùng có thể được thêm vào bằng tay ở mục `Grep - Match` trong tab `Options` như Hình 7.1 sau.



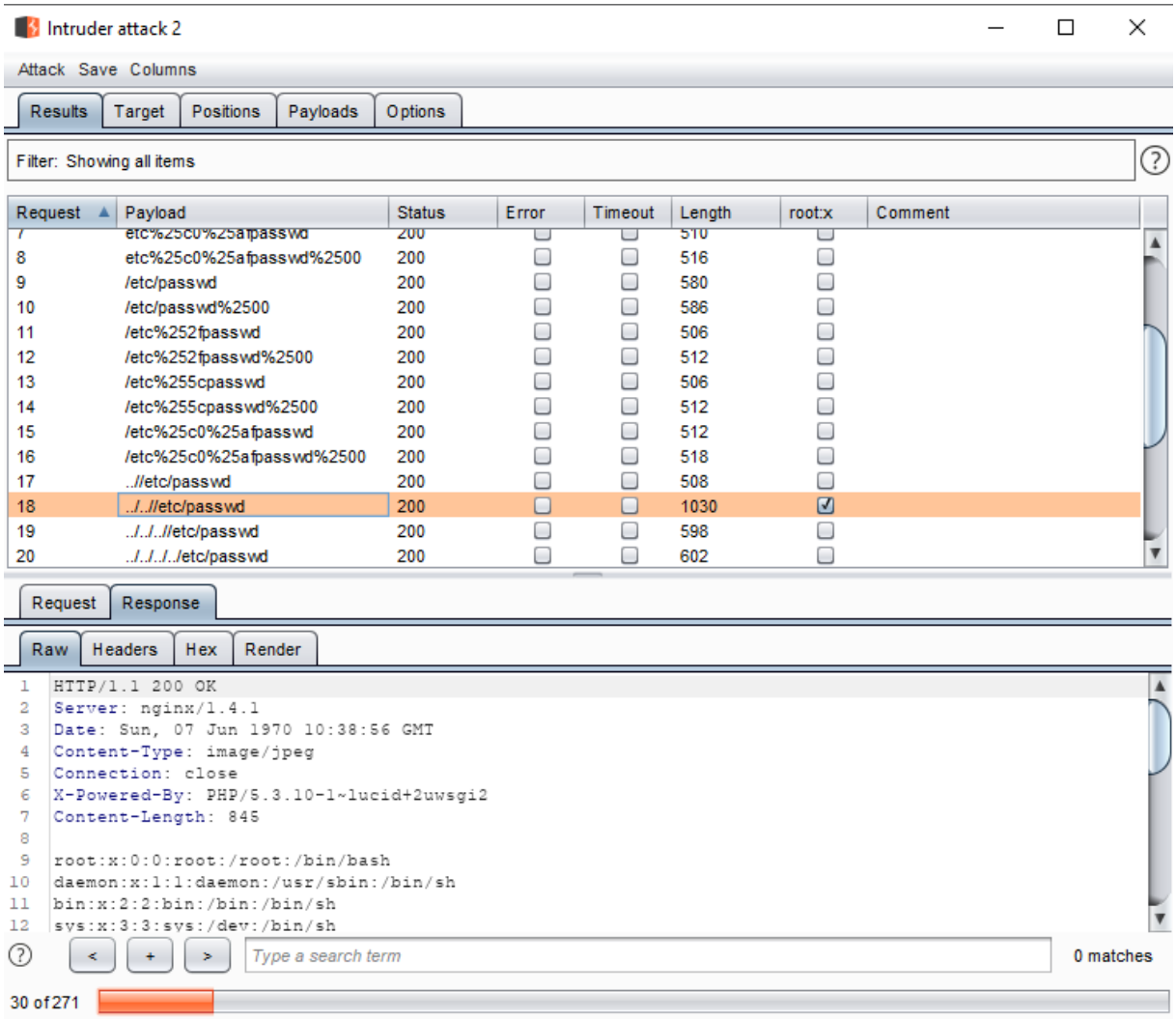
Hình 7.1: Thêm thiết lập so trùng vào cấu hình kiểm thử ở Burp Suite

Tương tự, chức năng danh sách payload có thể được thêm vào tuần tự hoặc trích xuất từ tập tin ở mục `Payload Options [Simple list]` trong tab `Payloads` như Hình 7.2 sau. Việc lưu trữ và sao chép các thiết lập kiểm thử (payload, các chuỗi so trùng) giữa các đối tượng kiểm thử trong chức năng `Intruder` khá phiền phức so với webfuzzer. Người dùng webfuzzer chỉ việc chọn những cấu hình kiểm thử thường dùng đã được định nghĩa sẵn ở backend, đây cũng là một ưu điểm của ứng dụng so với Burp Suite.



Hình 7.2: Thêm payload vào cấu hình kiểm thử ở Burp Suite

Trong quá trình kiểm thử bằng Burp Suite, chúng tôi phải tự nhận diện lỗ hổng thủ công thông qua giao diện kiểm thử của chức năng `Intruder` như Hình 7.3 sau.



Hình 7.3: Giao diện kiểm thử bằng chức năng `Intruder` của Burp Suite

Từng request kiểm thử chứa payload tương ứng sẽ được hiển thị thành một dòng trên giao diện, lần lượt thể hiện các thông tin của request đó, backend của webfuzzer cũng xuất ra kết quả kiểm thử tương tự trên terminal. Các chuỗi và biểu thức chính quy so trùng được hiển thị dưới dạng một cột, cột này sẽ được đánh dấu khi nội dung response tương ứng có chứa chuỗi đó như Hình 7.3. Khung bên dưới hiển thị nội dung chi tiết của response ứng với request kiểm thử đang chọn ở trên. Tương tự khi request kiểm thử vượt quá thời gian timeout đã thiết lập thì cột `Timeout` trên giao diện sẽ được đánh dấu. Sau khi quá trình kiểm thử kết thúc, người dùng phải tự hệ thống lại xem payload nào phát hiện được lỗ hổng.

7.2 Tập kiểm thử và kết quả đánh giá công cụ

Để quá trình đánh giá kết quả được khách quan, chúng tôi sử dụng tập kiểm thử gồm có các trang web có và không có các lỗ hổng LFI và time-based SQLI. Chúng tôi đồng thời kiểm thử các trang web này bằng webfuzzer và Burp Suite, so sánh kết quả kiểm thử, các payload phát hiện lỗ hổng nếu có, và thời gian kiểm thử mỗi mục tiêu của hai phần mềm này. Máy tính thực hiện kiểm thử trên cả hai phần mềm sử dụng CPU Intel Core i5-9400F và 8GB DDR4 RAM, trên hệ điều hành Ubuntu 20.04 LTS Focal. Ở chức năng `Intruder` của Burp Suite chúng tôi thiết lập số lượng luồng (threads) là 1, số lần thử gửi lại request khi thất bại là 0, và không áp dụng các phương pháp encode nào khác trên payload vì trong webfuzzer chúng tôi không hiện thực các chức năng này. Trong quá trình trình bày kết quả đánh giá, chúng tôi rút gọn trường cookies và headers của các đối tượng kiểm thử (request mẫu) trong trường hợp không cần thiết. Chúng tôi ghi lại kết quả đánh giá dưới dạng Bảng 7.1 sau, trong đó `N` là số payload phát hiện lỗi (trong số 271 payload LFI và 476 payload time-based SQLI) và `T` là thời gian kiểm thử của ứng dụng tính bằng mili giây.

Bảng 7.1: Mẫu bảng so sánh kết quả kiểm thử ứng dụng webfuzzer so với Burp Suite

Loại lỗ hổng	**Webfuzzer**		**Burp Suite**	
	Số payload	Thời gian	Số payload	Thời gian
LFI	N	T	N	T
Time-based SQLI	N	T	N	T

Nội dung request mẫu và kết quả kiểm thử các request này trên hai ứng dụng lần lượt được trình bày bằng các Đoạn mã và các Bảng dưới đây. Danh sách đầy đủ các payload phát hiện được lỗ hổng ứng với mỗi đối tượng kiểm thử có thể được tham khảo tại `http://61.28.235.183:3000/result`.

```
1 {
2     "url":
    "http://testphp.vulnweb.com:80/showimage.php?file=§./pictures/1.jpg§",
3     "cookies": "",
4     "headers": { ... },
```

```
    "method": "get"
}
```

Đoạn mã 7.2: Request mẫu 1 có lỗ hổng LFI

Bảng 7.2: Kết quả kiểm thử request mẫu 1

Loại lỗ hổng	Webfuzzer		Burp Suite	
	Số payload	Thời gian	Số payload	Thời gian
LFI	1	89998	1	88400

Request mẫu 1 được dùng để truy vấn một tập tin ảnh trên máy chủ web `http://testphp.vulnweb.com/` chứa nhiều lỗ hổng bảo mật, chuyên dùng cho mục đích thử nghiệm, đánh giá hiệu quả của các công cụ kiểm thử. Webfuzzer phát hiện lỗ hổng LFI bằng một payload giống với Burp Suite với thời gian chỉ chậm hơn một giây.

```
{
    "url": "http://testphp.vulnweb.com:80/search.php?test=query",
    "cookies": "",
    "headers": { ... },
    "data": {
        "searchFor": "a§§",
        "goButton": "go"
    },
    "method": "post"
}
```

Đoạn mã 7.3: Request mẫu 2 có lỗ hổng time-based SQLI

Bảng 7.3: Kết quả kiểm thử request mẫu 2

Loại lỗ hổng	Webfuzzer		Burp Suite	
	Số payload	Thời gian	Số payload	Thời gian
Time-based SQLI	14	298195	14	299600

Request mẫu 2 kiểm thử tính năng tìm kiếm của trang web trên. Bằng cách chèn payload kiểm thử vào nội dung tìm kiếm, webfuzzer phát hiện được lỗ hổng time-based SQLI

thông qua 14 payload giống Burp Suite, với thời gian thậm chí nhỉnh hơn.

```
{
    "url": "http://www.daotaonlyt.edu.vn:80/img/§§",
    "cookies": { ... },
    "headers": { ... },
    "method": "get"
}
```

Đoạn mã 7.4: Request mẫu 3

Bảng 7.4: Kết quả kiểm thử request mẫu 3

Loại lỗ hổng	Webfuzzer		Burp Suite	
	Số payload	Thời gian	Số payload	Thời gian
LFI	0	2925	0	3500

Đối với request mẫu 3, chúng tôi cố gắng khai thác lỗ hổng LFI trên trang web `http://www.daotaonlyt.edu.vn` ở một yêu cầu truy vấn hình ảnh tương tự request mẫu 1. Cả hai ứng dụng đều không phát hiện được lỗ hổng nhưng thời gian thực thi của webfuzzer nhanh hơn khá nhiều so với Burp Suite.

```
{
    "url": "http://artistclub.vn:80/",
    "cookies": { ... },
    "headers": { ... },
    "data": {
        "email_nhantin": "aaaa@gmail.com§§"
    },
    "method": "post"
}
```

Đoạn mã 7.5: Request mẫu 4 có lỗ hổng time-based SQLI

Bảng 7.5: Kết quả kiểm thử request mẫu 4

Loại lỗ hổng	Webfuzzer		Burp Suite	
	Số payload	Thời gian	Số payload	Thời gian
Time-based SQLI	7	92952	7	89700

Đối với request mẫu 4, chúng tôi cố gắng khai thác lỗ hổng time-based SQLI ở khung điền email đăng kí nhận thông báo trên trang web `http://www.daotaonlyt.edu.vn`. Hai ứng dụng phát hiện được lỗ hổng bằng các payload giống hệt nhau, nhưng thời gian thực thi của webfuzzer chậm hơn Burp Suite khoảng 3 giây.

```
{
    "url": "http://artistclub.vn:80/upload/sanpham/
        §b814bf98-5aad-4146-ade5-6a41f3b631490060_323x360.jpeg§",
    "cookies": { ... },
    "headers": { ... },
    "method": "get"
}
```

Đoạn mã 7.6: Request mẫu 5

Bảng 7.6: Kết quả kiểm thử request mẫu 5

Loại lỗ hổng	Webfuzzer		Burp Suite	
	Số payload	Thời gian	Số payload	Thời gian
LFI	0	4086	0	4900

```
{
    "url": "http://mwc.com.vn:80/search?s=a§§",
    "cookies": { ... },
    "headers": { ... },
    "method": "get"
}
```

Đoạn mã 7.7: Request mẫu 6

Bảng 7.7: Kết quả kiểm thử request mẫu 6

Loại lỗ hổng	Webfuzzer		Burp Suite	
	Số payload	Thời gian	Số payload	Thời gian
Time-based SQLI	0	52760	0	54300

Tương tự như request mẫu 3 và 5, cả hai ứng dụng đều không phát hiện thấy lỗ hổng LFI và time-based SQLI trên các trang web mục tiêu ở request mẫu 6. Nhìn chung cả ba trường hợp không phát hiện được lỗ hổng thì thời gian thực thi của webfuzzer luôn nhanh hơn Burp Suite.

```
{
    "url": "http://www.monitoringris.org:80/index.php?id=30§§",
    "cookies": "",
    "headers": { ... },
    "method": "get"
}
```

Đoạn mã 7.8: Request mẫu 7 có lỗ hổng time-based SQLI

Bảng 7.8: Kết quả kiểm thử request mẫu 7

Loại lỗ hổng	Webfuzzer		Burp Suite	
	Số payload	Thời gian	Số payload	Thời gian
Time-based SQLI	4	434913	4	430200

```
{
    "url": "http://www.daotaonlyt.edu.vn:80/index.php?id=320§§",
    "cookies": { ... },
    "headers": { ... },
    "method": "get"
}
```

Đoạn mã 7.9: Request mẫu 8 có lỗ hổng time-based SQLI

Bảng 7.9: Kết quả kiểm thử request mẫu 8

Loại lỗ hổng	Webfuzzer		Burp Suite	
	Số payload	Thời gian	Số payload	Thời gian
Time-based SQLI	4	66493	4	64900

```
{
    "url": "http://csdl.vietnamtourism.gov.vn:80/index.php/
        search/?data=cslt&title=a§§",
    "cookies": "",
    "headers": { ... },
    "method": "get"
}
```

Đoạn mã 7.10: Request mẫu 9 có lỗ hổng time-based SQLI

Bảng 7.10: Kết quả kiểm thử request mẫu 9

Loại lỗ hổng	Webfuzzer		Burp Suite	
	Số payload	Thời gian	Số payload	Thời gian
Time-based SQLI	86	1572156	86	1481300

```
{
    "url": "http://www.hanglahangdoc.com:80/tim_kiem.aspx?keyword=a§§",
    "cookies": { ... },
    "headers": { ... },
    "method": "get"
}
```

Đoạn mã 7.11: Request mẫu 10 có lỗ hổng time-based SQLI

Bảng 7.11: Kết quả kiểm thử request mẫu 10

Loại lỗ hổng	Webfuzzer		Burp Suite	
	Số payload	Thời gian	Số payload	Thời gian
Time-based SQLI	6	219560	6	212800

Đối với các request mẫu 7, 8, 9 và 10 còn lại, webfuzzer đều phát hiện được lỗ hổng time-based SQLI bằng những payload giống với Burp Suite. Xét về thời gian kiểm thử, webfuzzer đều chậm hơn Burp Suite một vài giây.

Qua quá trình kiểm thử các request mẫu trên, chúng tôi nhận thấy các payload phát hiện lỗi ở cả hai ứng dụng đều giống hệt nhau, đây là thành công lớn nhất trong quá trình hiện thực webfuzzer, ứng dụng này đã mô phỏng thành công một phần chức năng `Intruder` của Burp Suite như hướng phát triển công cụ đã đề ra ở **Chương 4**. Thời gian kiểm thử của webfuzzer cũng xấp xỉ Burp Suite, chỉ chậm hơn khoảng 2-10 giây tùy vào mỗi request mẫu, hơn nữa webfuzzer có phần nhanh hơn Burp Suite ở các request mẫu mà cả hai ứng dụng đều không phát hiện được lỗ hổng. Vì vậy ngoài lý do không thể tránh khỏi về kết nối, đường truyền giữa các lần kiểm thử, chúng tôi nhận định khoảng thời gian chênh lệch trong việc kiểm thử các request mẫu có lỗ hổng phần lớn đến từ việc lưu trữ kết quả kiểm thử vào cơ sở dữ liệu, do đó chênh lệch thời gian vài giây là một khoản đánh đổi chấp nhận được. Thời gian phản hồi của các HTTP request gửi đến backend webfuzzer được triển khai ở máy chủ nội bộ (`http://localhost:13336/`) hoặc ở VPS (`http://61.28.235.183:13336/`) chỉ dao động trong khoảng 20-200 mili giây.

Xét về trải nghiệm người dùng, ứng dụng webfuzzer trực quan và tiện lợi hơn Burp Suite rất nhiều. Thay vì việc thiết lập cấu hình kiểm thử và nhận diện lỗ hổng thủ công ở Burp Suite như đã mô tả ở **Đề mục 7.1**, webfuzzer đã tích hợp sẵn khả năng nhận diện lỗ hổng trong mô-đun kiểm thử, đồng thời tổng hợp sẵn những cấu hình kiểm thử tiện dụng, dễ dàng chỉnh sửa. Người dùng chỉ việc chọn request mẫu và loại lỗ hổng, kết quả kiểm thử sẽ được ứng dụng webfuzzer cung cấp chi tiết, request kiểm thử chứa payload nào bị timeout, hay trả về response có chứa chuỗi so trùng nào, phát hiện được lỗ hổng gì. Tổng kết từ các nhận định trên, chúng tôi đánh giá ứng dụng webfuzzer đã được hiện thực hoàn chỉnh ở mức chấp nhận được, thỏa mãn yêu cầu thiết kế hệ thống.

Chương 8

Tổng kết

8.1 Các kết quả đạt được

Trong quá trình thực hiện đề tài "*Xây dựng ứng dụng kiểm thử bảo mật ứng dụng web thông qua thông điệp HTTP*", tôi đã tìm hiểu được một vài phương pháp kiểm thử bảo mật phần mềm nói chung và ứng dụng web nói riêng, tìm hiểu về các thành phần quan trọng về khía cạnh bảo mật của ứng dụng web cùng với xu hướng tấn công các ứng dụng web hiện nay. Hơn nữa tôi cũng tìm hiểu được một số lớp lỗ hổng bảo mật phổ biến trên ứng dụng web. Từ những hiểu biết đó, chúng tôi xây dựng thành công một phương pháp mới để hiện thực **webfuzzer**, một ứng dụng kiểm thử bảo mật ứng dụng web nhanh và ổn định, cho phép người dùng tập trung vào chính xác vào request mẫu mà họ muốn kiểm thử. Thông qua các tiêu chí đánh giá và kết quả kiểm thử với một số ứng dụng web mục tiêu, chúng tôi nhận định đã hiện thực khá tốt ứng dụng **webfuzzer**, thỏa mãn những mục tiêu đã đề ra ban đầu.

8.2 Hướng nghiên cứu và phát triển trong tương lai

Trong thời gian sắp tới, tôi có thể thực hiện những công việc sau đây để hoàn thiện hơn ứng dụng về cả chức năng lẫn hiệu năng.

- Xây dựng cơ chế xác thực người dùng (authentication) trên ứng dụng webfuzzer lẫn phần mở rộng Burp Suite.

- Phát triển cấu hình kiểm thử và các mô-đun liên quan để phát hiện các lỗ hổng khác trên ứng dụng web như các dạng khác của SQLI, remote file inclusion, XSRF,...

- Cập nhật thêm payload, các kĩ thuật bypass đã và sẽ xuất hiện trong tương lai vào ứng dụng. Những kĩ thuật này bao gồm: các phương pháp mới để phát hiện những lỗ hổng hiện tại, phát hiện những lỗ hổng có thể phát sinh từ lỗ hổng hiện tại đó; các kĩ thuật làm rối cộng với payload đặc thù mới để xuyên qua các kĩ thuật chống bypass, các bộ lọc dữ liệu đầu vào của tường lửa ứng dụng web,...

- Hiện thực mô-đun phát hiện HTTP request bị chặn khi gửi đến ứng dụng web mục tiêu.

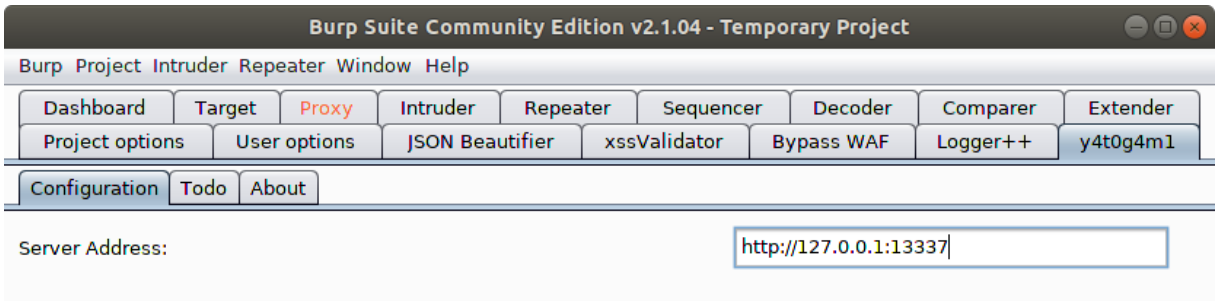
- Hiện thực trang điều chỉnh biến môi trường backend trên UI của ứng dụng, cho phép người dùng thay đổi các giá trị này mà không cần can thiệp vào mã nguồn backend.

- Cho phép người dùng xem HTTP response trả về tương tự như giao diện kiểm thử của chức năng `Intruder`.

Phụ lục A

Hướng dẫn sử dụng phần mở rộng Burp Suite

Trong trường hợp lần đầu sử dụng ứng dụng, ta cần cài đặt phần mở rộng[1] `y4t0g4m1.jar` trong đường dẫn gốc của mã nguồn backend vào công cụ Burp Suite. Giao diện chính của phần mở rộng trên Burp Suite được mô tả như Hình A.1 dưới đây, chỉ đơn giản gồm một textbox để thiết lập địa chỉ máy chủ ứng dụng nhận request mẫu. Mặc định giá trị của trường này là `http://61.28.235.183:13336`, là địa chỉ backend ứng dụng được triển khai trên VPS với cổng mặc định là 13336. UI của webfuzzer hiện đang được triển khai trên đường dẫn `http://61.28.235.183:3000` tại thời điểm hoàn thành luận văn.



Hình A.1: Giao diện chính của phần mở rộng trên Burp Suite

Sau khi khởi chạy UI của ứng dụng webfuzzer và cài đặt phần mở rộng trên Burp Suite, ta tiến hành thiết lập proxy trên Burp Suite và trình duyệt web để bắt request mẫu[2]. Sau

[1]Tham khảo tại https://portswigger.net/support/how-to-install-an-extension-in-burp-suite
[2]Tham khảo tại https://matthewsetter.com/introduction-to-burp-suite/

đó ta chuyển request mong muốn đến tab `Intruder` để tạo và gửi request mẫu đến máy chủ webfuzzer. Hình 4.1 mô tả quá trình tạo request mẫu bằng cách thêm hoặc bỏ các kí tự "§" bao quanh giá trị của tham số và chọn "`Send to y4t0g4m1 webfuzzer`". Request mẫu này sau đó sẽ hiển thị trên khung danh sách request mẫu ở trang bảng điều khiển của UI ứng dụng webfuzzer để người dùng tiến hành chọn lỗ hổng và kiểm thử.

Tài liệu tham khảo

[1] 0verpwn. *0verpwn/Fuzzing: Fuzzing Payloads to Assist in Web Application Testing.* 2019. https://github.com/0verpwn/Fuzzing (visited on 11/25/2019).

[2] Khaleel Ahmad. "Classification of SQL Injection Attacks". In: *VSRD Technical & Non-Technical Journal* 1 (May 2010), pp. 236–242.

[3] Fabien Duchene et al. "KameleonFuzz: Evolutionary Fuzzing for Black-Box XSS Detection". In: Mar. 2014. DOI: 10.1145/2557547.2557550.

[4] Allen Harper et al. *Gray hat hacking the ethical hackers handbook.* McGraw-Hill Osborne Media, 2011.

[5] George Klees et al. "Evaluating fuzz testing". In: *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security.* ACM. 2018, pp. 2123–2138.

[6] Xiaowei Li and Yuan Xue. "A survey on web application security". In: *Nashville, TN USA* (2011).

[7] PortSwigger Ltd. *Burp Suite - Cybersecurity Software from PortSwigger.* 2019. https://portswigger.net/burp (visited on 11/26/2019).

[8] PortSwigger Ltd. *What is directory traversal, and how to prevent it?* 2019. https://portswigger.net/web-security/file-path-traversal (visited on 11/26/2019).

[9] Daniel Miessler. *SecLists/Fuzzing at master · danielmiessler/SecLists.* 2019. https://github.com/danielmiessler/SecLists/tree/master/Fuzzing (visited on 11/25/2019).

[10] OWASP. *Top 10-2017 Top 10 - OWASP.* 2017. https://www.owasp.org/index.php/Top_10-2017_Top_10 (visited on 11/26/2019).

[11] *reduction-admin/react-reduction: React Reduction - Free Admin Template Built with React and Bootstrap4.* https://github.com/reduction-admin/react-reduction (visited on 06/23/2020).

[12] Bryan Sullivan and Vincent Liu. *Web application security, a beginner's guide*. McGraw-Hill Education Group, 2011.

[13] Mikko Vimpari. "An Evaluation of Free Fuzzing Tools". In: *Master's Thesis, University of Oulu* (2015).

[14] Georgia Weidman. *Penetration testing: a hands-on introduction to hacking*. No Starch Press, 2014.