ĐẠI HỌC QUỐC GIA HÀ NỘI
ĐẠI HỌC CÔNG NGHỆ

Khóa luận tốt nghiệp

Phát triển một công cụ hỗ trợ phát hiện và khai thác tự động các lỗ hổng Web

Nguyễn Đình Biển

Hà Nội, 2022

ĐẠI HỌC QUỐC GIA HÀ NỘI
ĐẠI HỌC CÔNG NGHỆ

Khóa luận tốt nghiệp

Phát triển một công cụ hỗ trợ phát hiện và khai thác tự động các lỗ hổng Web

Nguyễn Đình Biển

Giảng viên hướng dẫn: TS. Nguyễn Đại Thọ

Hà Nội, 2022

Tóm tắt

Zed Attack Proxy (ZAP) là một công cụ mã nguồn mở giúp rà soát lỗ hổng tự động theo tập luật được phát triển liên tục bởi cộng đồng. Tuy nhiên, ZAP lại thiếu đi các công cụ xác minh và khai thác các lỗ hổng. Công cụ Nextgen Automation Framework (NAF) được phát triển trong khuôn khổ của khóa luận này sẽ giúp bổ sung cho ZAP về việc xác minh và khai thác lỗ hổng phía máy chủ Web tự động.

Các công cụ khai thác sẵn có như Invicti, Acunetix,... cho phép xác minh một cách tự động các lỗ hổng Web nghiêm trọng như SQL Injection, Command Injection, Remote File Inclusion, Server Side Template Injection,... là thực sự tồn tại giúp công tác kiểm thử an ninh được thực hiện một cách hiệu quả hơn, giảm thiểu các thao tác thủ công phức tạp tiêu tốn nhiều thời gian.

Công cụ NAF của chúng tôi tích hợp vào ZAP các bộ công cụ mã nguồn mở phổ biến như SQLMap, Commix, Tplmap,... biến ZAP trở thanh một công cụ vừa có chức năng dò quét vừa có chức năng xác minh tự động các lỗ hổng an ninh Web và cung cấp thêm các công cụ khai thác. Do đều là các công cụ mã nguồn mở được bảo trì liên tục nên có khả năng khai thác được miền rất rộng các lỗ hổng với kỹ thuật được cập nhật. Do còn phụ thuộc vào các tập luật được phát triển bởi cộng đồng và chưa có khả năng phát hiện được một số lỗ hổng khi rà quét nhưng NAF lại có khả năng xác minh và khai thác một số lỗ hổng nhiều hơn so với các công cụ thương mại như Invicti điển hình là lỗ hổng Server Side Template Injection.

Lời cảm ơn

Lời đầu tiên cho phép em được gửi lời cảm ơn tới Khoa Công Nghệ Thông Tin - Trường Đại học Công Nghệ ĐHQG Hà Nội đã tạo điều kiện thuận lợi cho em được học tập, nghiên cứu và thực hiện đề tài tốt nghiệp này.

Em cũng xin được bày tỏ lòng biết ơn sâu sắc tới thầy Nguyễn Đại Thọ đã tận tình hướng dẫn, đóng góp những ý kiến giúp em hoàn thành khóa luận tốt nghiệp một cách tốt nhất.

Em cũng vô cùng biết ơn các thầy cô trong trường tận tình giảng dạy, truyền thụ cho em những kiến thức và kỹ năng quan trọng làm hành trang vững chắc trên con đường học vấn của bản thân.

Chúc mọi người luôn luôn mạnh khoẻ và gặt hái được nhiều thành công trong cuộc sống.

Lời cam đoan

Em xin cam đoan rằng khóa luận tốt nghiệp này không sao chép từ bất kỳ ai, tổ chức nào khác. Toàn bộ nội dung được trình bày trong tài liệu đều là cá nhân em qua quá trình học tập, tìm hiểu và nghiên cứu mà hoàn thiện. Mọi tài liệu tham khảo đều được ghi chép lại và trích dẫn hợp pháp. Nếu lời cam đoan là sai sự thật thì em xin chịu mọi trách nhiệm và hình thức kỷ luật theo quy định từ phía nhà trường.

Mục lục

Danh sách hình vẽ

Danh sách bảng

Chương 1

Mở đầu

Đối với bất kỳ sản phẩm phần mềm nào, việc đảm bảo an toàn an ninh, khắc phục và vá các lỗ hổng cũng cần phải được thực hiện bởi nhà phát triển, cung cấp sản phẩm. Các phần mềm hệ thống, như các hệ điều hành, các chương trình điều khiển thiết bị, các chương trình tiện ích,... thông thường là những sản phẩm đại chúng, được phát triển bởi những doanh nghiệp lớn, chuyên nghiệp, có uy tín nên thời gian thời gian vá lỗ hổng nhanh và kịp thời trước khi các cuộc tấn công xảy ra. Do đó các hệ thống sẽ an toàn trước các lỗ hổng khi được cập nhật liên tục.

Trái lại, các ứng dụng Web thường chỉ được bảo trị bởi nhà phát triển Web - là các đơn vị nhỏ và vừa không có hoặc ít có khả năng phát hiện và vá được các lỗ hổng trong thời gian ngắn. Phần lớn các lỗ hổng đó liên quan đến việc không chú trọng đến việc đảm bảo an ninh trong thời gian phát triển ứng dụng. Một số báo cáo năm 2019 chỉ ra rằng khoảng 82% lỗ hổng nằm trong mã nguồn và hầu hết các trang web đều có lỗ hổng mức nghiêm trọng.

Các cuộc kiểm thử xâm nhập là một phần không thể thiếu để nhận được cảnh báo an ninh trong các sản phẩm. Với một cuộc kiểm thử xâm nhập, một quy trình đặc trưng có thể có các bước sau đây

Thu thập thông tin thu thập các thông tin về mục tiêu như thông tin về hệ thống, ứng dụng.

Khảo sát từ các thông tin bên trên, tiếp tục khảo sát về hạ tầng, các dịch vụ đang sử dụng từ các nguồn trên Internet.

Khám phá và rà quét với mỗi trang Web cụ thể sẽ thực hiện khám phá và rà quét các lỗ hổng để thu thập được lượng lớn điểm khả nghi là lỗ hổng nhất.

Chỉ thị các lỗ hổng xác minh lại và chỉ ra lỗ hổng bảo mật có tồn tại hay không và đánh giá tác động của nó.

Khai thác và hậu khai thác đối với một số cuộc tấn công sẽ có yêu cầu khai thác và hậu khai thác lỗ hổng để chỉ ra ảnh hưởng trong trường hợp bị tấn công.

Duyệt và báo cáo tổng hợp lỗ hổng, đảm bảo chất lượng của báo cáo và gửi lại báo cáo cho đơn vị.

Trong đó, các bước như thu thập thông tin, khảo sát, khám phá và rà quét hiện nay đã được tự động hóa phần lớn bằng các công cụ tự động. Tuy nhiên, để các lỗ hổng được đánh giá là có tồn tại hay không thì cần được xác minh lại bằng phương pháp thủ công, làm tốn lượng lớn thời gian. Tuy nhiên, với một số lỗ hổng nghiêm trọng có thể chỉ ra ngay được tính đúng đắn của lỗ hổng và tác động của nó tới hệ thống như SQL Injection, Command Injection,...

Do đó, nếu có thể tích hợp các công cụ này vào trong quá trình kiểm thử xâm nhập sẽ giúp giảm được thời gian xác minh các lỗ hổng này, đưa ra được tác động của nó.

Nắm được điều này, trên thị trường đã có các công cụ tự động sử dụng cơ chế cung cấp bằng chứng khai thác (Proof-of-Exploit) để đưa ra các báo cáo có độ tin cậy cao trong thời gian ngắn, điển hình là Invicti (Netsparker). Với công cụ thương mại Invicti cho phép xác minh tự động các lỗ hổng và cho phép sinh ra bằng chứng khai thác các lỗ hổng điển hình như sau

- SQL Injection
- Boolean SQL Injection
- Blind SQL Injection
- Remote File Inclusion (RFI)
- Command Injection
- Blind Command Injection
- XML External Entity (XXE) Injection
- Remote Code Evaluation
- Local File Inclusion (LFI)
- Server-side Template Injection

- Remote Code Execution
- Injection via Local File Inclusion

Open Web Application Security Project (OWASP) là một tổ chức phi lợi nhuận quốc tế chuyên cung cấp thông tin về bảo mật ứng dụng web. OWASP thường đưa ra các khuyến cáo cho từng loại lỗ hổng về độ nguy hiểm và cách phòng, chống trong thực tế. OWASP Top 10 là báo cáo đánh giá các loại lỗ hổng theo danh mục, được xếp hạng theo độ quan trọng nhất được cập nhật thường xuyên bởi một nhóm chuyên gia đến từ nhiều quốc gia, tổ chức trên toàn cầu. OWASP Top 10 thường được sử dụng làm tiêu chuẩn cho việc đánh giá an toàn thông tin ở các tổ chức tổ chức.

Năm 2021, OWASP tiếp tục được cập nhật thêm so với lần cuối cùng là năm 2017. Báo cáo được liệt kê với 10 danh mục sau

A01 Broken Access Control

A02 Cryptographic Failures

A03 Injection

A04 Insecure Design

A05 Security Misconfiguration

A06 Vulnerable and Outdated Components

A07 Identification and Authentication Failures

A08 Software and Data Integrity Failures

A09 Security Logging and Monitoring Failures

A10 Server Side Request Forgery (SSRF)

Với việc hợp nhất các lỗ hổng, thay đổi danh mục, xếp hạng có nhiều sự thay đổi như việc đưa danh mục Broken Access Control và Cryptographic Failures lên đầu thay cho Injection cho thấy việc các chuyên gia nhận thấy sự quan trọng của việc bảo mật thông tin so với các bảng xếp hạng trước đây. Tuy nhiên, các danh mục Injection vẫn luôn đứng hạng đầu do sự phổ biến của việc không kiểm soát đầu vào của người dùng dẫn đến trình duyệt hoặc máy chủ có thể thực thi câu lệnh có hành vi bất thường.

Các lỗ hổng Injection ở phía máy chủ dễ dàng xác minh và đánh giá tự động ảnh hưởng tới ứng dụng Web bằng cách đưa ra các bằng chứng về việc khai thác các lỗ hổng đó trên máy chủ. Tuy nhiên với các lỗ hổng Injection ở

phía máy khách thường dễ được rà quét tự động bằng cách cung cấp đường dẫn nhưng lại rất khó đánh giá được tự động tác động do cần có một kịch bản cụ thể và thông tin nghiệp vụ cụ thể để xác minh.

OWASP Zed Attack Proxy (ZAP) một công cụ bảo mật mã nguồn mở được phát triển và duy trì bởi cộng đồng với OWASP là tổ chức bảo hộ. ZAP là một công cụ phổ biến được sử dụng trong quá trình kiểm thử xâm nhập các ứng dụng Web do khả năng làm việc tự động với quét lỗ hổng thụ động và quét lỗ hổng chủ động với tính tự động cao.

Với việc bổ sung các công cụ tự động xác minh, khai thác như SQLMap, Commix, Tplmap,... vào trong ZAP, sẽ khiến quá trình kiểm thử sử dụng ZAP giảm được thời gian xác minh lỗ hổng và cũng có thể đưa ra được bằng chứng về việc khai thác như công cụ Invicti. Các thử nghiệm cho thấy, công cụ NAF được xây dựng trong phạm vi khóa luận này, tuy rằng việc phát hiện lỗ hổng thông qua công đoạn rà quét chưa thực sự tốt như Invicti khi mà vẫn tồn tại các lỗ hổng mà NAF không thể phát hiện do phụ thuộc vào các luật của cộng đồng phát triển. Nhưng NAF đã xác minh được số lượng các loại lỗ hổng gần với các loại lỗ hổng mà Invicti có thể xác minh và có thể cung cấp chứng khai thác tương đương các loại lỗ hổng, tốt hơn với các loại lỗ hổng như Server-side Template Injection.

Chương 2

Kiểm thử an ninh ứng dụng Web

2.1 Lỗ hổng ứng dụng Web

Hiện nay, có rất nhiều kĩ thuật tấn công vào ứng dụng Web bao gồm cả phía máy chủ và phía máy khách. Dưới đây mà một số lỗ hổng được trình bày trong phạm vi được khai thác trong khóa luận.

2.1.1 SQL Injection

Với các ứng dụng Web hiện nay, việc lưu trữ dữ liệu thường được lưu tại cơ sở dữ liệu. Các cơ sở dữ liệu đến ngày nay vẫn là một cơ sở dữ liệu phổ biến do sự tin cậy và tốc độ của chúng. Để truy vấn được đến các dữ liệu này, các máy chủ Web cần sử dụng các truy vấn SQL để lấy dữ liệu đến người dùng. Tuy nhiên, sự không kiểm soát các đầu vào của người dùng có thể khiến các câu lệnh truy vấn SQL có thể bị thay đổi để thực thi các câu lệnh bất thường làm lộ thông tin Cơ sở dữ liệu hoặc tệ hơn nữa làm cho cơ sở dữ liệu mất quyền kiểm soát.

Ví dụ với ngôn ngữ PHP dưới đây

Listing 2.1: SQL Injection

```
// The article parameter is assigned to $article
// variable without any sanitization or validation
$articleid = $_GET['article'];
// The $articleid parameter is passed as part of
// the query
$query = "SELECT␣*␣FROM␣articles␣WHERE␣articleid␣=␣
   $articleid";
```

Khi người dùng tùy chỉnh tham số 'article' thành các câu lệnh dạng như '1 or 1=1– -' sẽ khiến vô tình dữ liệu của toàn bộ bảng 'article' được đưa ra ngoài.

Không chỉ tại câu lệnh truy vấn đó, các câu lệnh có thể sử dụng để lấy toàn bộ cơ sở dữ liệu thậm chí là đặt cửa hậu để tấn công về sau.

Hiện nay các kĩ thuật tấn công rất đang dạng như boolean-based blind, time-based blind, error-based, UNION query-based, stacked queries and Out-of-band.

2.1.2 OS Command Injection

OS Commnad Injection là lỗ hổng cho phép thực thi các câu lệnh của hệ điều hành trái phép. Nguyên do cũng đến từ việc cho phép người dùng nhập trực tiếp hoặc xử lý không hết hoàn toàn các câu lệnh thực thi trên máy chủ Web. Các lỗ hổng này thường dẫn đến thực thi lệnh trái phép trên máy chủ Web. Ví dụ với ngôn ngữ PHP dưới đây về một chương trình giúp phân giải tên miền thông qua câu lệnh 'host' của Linux

Listing 2.2: OS Command Injection

```
echo shell_exec("host␣{$_POST["site"]}");
```

Với các tham số của 'site' như 'google.com; cat /etc/pasword' sẽ khiển chương trình thực thi câu lệnh 'host google.com; cat /etc/passwd' ở hệ điều hành làm lộ thông tin máy chủ.

2.1.3 Remote Code Evaluation

Remote Code Evaluation là lỗ hổng cho phép thực thi các đoạn mã trái phép trên trình phân tích cú pháp của ngôn ngữ - thường xảy ra với các ngôn ngữ thông dịch. Các đầu vào khi không được kiểm soát hoàn toàn có thể khiển các đoạn mã được thực thi trái phép. Các lỗ hổng dạng này cũng thường dẫn đến thực thi lệnh trái phép trên máy chủ Web.

Một ví dụ về việc thực thi 'eval()' trong PHP dẫn đến lỗ hổng

Listing 2.3: Remote Code Evaluation

```
eval ("echo␣".$_REQUEST["parameter"].";");
```

Với tham số 'parameter' là '1;phpinfo();' khiến câu truy vấn này đưa ra thông tin máy chủ Web của thông tin PHP do hàm 'phpinfo()' được thực thi.

2.1.4 Server Side Template Injection

Tương tự với Remote Code Evaluation, Server Side Template Injection cũng cho phép thực thi các đoạn mã trên trình phân tích của Template Engine. Template Engine là các phần thực hiện việc phân tích cú pháp, thực thi các đoạn mã để sinh ra được nhiều đầu ra khác nhau thông qua một bản mẫu chỉ cần thay đổi tham số đầu vào. Lỗ hổng này cũng thường dẫn đến việc thực thi lệnh trái phép trên máy chủ Web.

Ví dụ đoạn mã sau

Listing 2.4: Server Side Template Injection

```
$output = $twig->render($_GET['custom_email'],
   array("first_name" => $user.first_name) );
```

Ba lỗ hổng OS Command Injection, Remote Code Evaluation và Server Side Template Injection có vector khai thác và mức độ tác động tương tự nhau nên trong một số trường hợp được gọi chung với cái tên Code Injection.

2.1.5 Local File Inclusion và Directory Traversal

Local File Inclusion là lỗ hổng làm cho máy chủ làm lộ hoặc thực thi các nội dung tập tin trên máy chủ đó. Trong các trường hợp như PHP, khi được phép tải lên tùy ý tập tin lên thì lỗ hổng này có thể dẫn đến thực thi lệnh trái phép trên máy chủ Web.

Ví dụ như đoạn mã sau:

Listing 2.5: Local File Inclusion

```
$file = $_GET['file'];
include($file);
```

Với ví dụ trên, khi người dùng được phép tải lên tập tin tùy ý sẽ khiến cho có thể thực thi bất kì đoạn mã nào do PHP cho phép xử lý động các đoạn mã khi được gọi câu lệnh 'include'. Ví dụ như tham số 'file' là '/var/www/upload/evil.php' Với 'evil.php' là tập tin có chứ các đoạn mã thực thi lệnh trái phép.

Kể cả khi không được tải lên tùy ý, lỗ hổng Local File Inclusion cũng thường được kết hợp với Directory Traversal, cho phép tùy ý đọc tập tin tại vị trí không được phép để làm lộ các thông tin nhạy cảm của hệ thống.

Ví dụ với tham số được chèn '../../etc/passwd' sẽ khiến thông tin về tập tin '/etc/passwd' được trả về cho người dùng.

2.1.6 Remote File Inclusion

Tương tự như Local File Inclusion, Remote File Inclusion cho phép máy chủ làm lộ hoặc thực thi các tập tin nhưng với nội dung từ xa. Thường thì các dẫn đến thực thi lệnh trái phép trên máy chủ Web.

Tiếp tục với ví dụ của lỗ hổng Local File Inclusion, khi tùy chọn 'php allow$_{u}rl_{f}open'\emptyset cbtschophpPHPtivccnidungtrongURL.NuktncngØavothams`file'l`http://attacker.example.com/evil.php'thktncngcththcthilnhtrnmychPHP.

2.2 Các công cụ rà quét lỗ hổng

2.2.1 Zed Attack Proxy (ZAP)

OWASP Zed Attack Proxy (thường được gọi là ZAP hay ZAProxy) là công cụ mã nguồn mở, công cụ cho phép các các kỹ sư an toàn thông tin tìm kiếm được lỗ hổng một cách tự động/bán tự động/thủ công thông qua cơ chế "middle-in-the-middle-proxy" giúp người dùng đứng giữa ứng dụng Web để thực hiện kiểm thử qua các plugin trong được xây dựng trong cộng đồng.

Điểm mạnh của ZAP là ứng dụng được viết bằng Java nên có thể triển khai trên nhiều nền tảng và có cộng đồng phát triển thường xuyên cập nhập các tri thức liên quan đến lỗ hổng hiện hành.

2.2.1.1 ZAP Core

ZAP Core là thành phần lõi cung cấp toàn bộ API giao tiếp với bên ngoài cho các plugin (bao gồm các luật, Addon) trong ZAP. Các plugin sẽ không trực tiếp giao tiếp với môi trường bên ngoài mà sẽ sử dụng các API này để gửi và nhận yêu cầu, đọc ghi dữ liệu, ghi cơ sở dữ liệu... Thành phần chính ZAP Core được phát triển từ dự án Paros Proxy - là một Interceptor Proxy thường được sử dụng để để thử các ca kiểm thử xâm nhập.

Các thành phần đáng kể nhất của ZAP Core phải kể đến là: Proxy, HttpSender, Session, Site Map, Scanner, Context và các Addon Core. Proxy chính là interceptor được kế thừa từ Paros, cho phép nhặn gửi-nhận các yêu cầu, phản hồi tới và đi qua Proxy. Thông thường, các trình duyệt sẽ sử dụng thông qua proxy này để thực hiện kiểm thử. Http Sender sẽ quản lý việc gửi và nhận của các thông báo HTTP. Session và Site Map sẽ quản lý lượng đầu ra và đầu vào các thông báo HTTP, để xây dựng thành một cây thông tin cung cấp các thành phần khác. Scanner cung cấp một khung để các luật thực hiện xác định lỗ hổng trên các thông báo HTTP. Context cung cấp ngữ cảnh cho việc rà quét, xác định lỗ hộng theo người dùng.

Ngoài ra, một số Addon sẽ được phát triển như một phần của ZAP Core thay vì là các phần mở rộng thông thường và cũng được các Addon khác sử dụng như một phần của ZAP Core.

2.2.1.2 ZAP Addons

Điểm đặc biệt của ZAP là cơ chế mở rộng bằng ZAP Addons cho phép cộng đồng xây dựng được các phần mở rộng riêng cho mình trên ZAP mà không cần sửa các phần liên quan đến ZAP Core, đảm bảo sự độc lập tránh sự mâu thuẫn giữa các ZAP Addon hoặc ZAP Addon với chính ZAP, đồng thời cũng đảm bảo được các phần được bảo trị độc lập liên tục.

2.2.1.3 Addon Spider và Addon Spider Ajax

Spider là một trong các Addons chính, được phát triển để thu thập và phát hiện được các thành phần đơn trong Website như các URL, hành động,...được gọi là các Site Node. Site Node cung cấp sẽ thông tin cho người dùng về thông tin Website đồng thời cũng cung cấp các thông tin về Website cho các Addon khác thông qua các API của ZAP.

Spider sẽ thu thập dựa trên cây DOM của Website, do đó Spider này sẽ chỉ nhận được các thông tin từ web tĩnh. Để lấy được các thông tin trong việc chạy JavaScript thì cần có một Addon khác là Ajax Spider. Ajax spider sử dụng Selenium như một trình web để duyệt web ảo và thực hiện thu thập các thông tin về Site Node.

2.2.1.4 Addon Passive Scan và Active Scan

Là các Addons chính, được sử dụng để chạy các kiểm thử dạng được lập trình sẵn thành các plugin đặc biệt gọi là luật(Rule). Các luật sẽ dựa theo thông tin của hệ thống cung cấp để chạy các ca kiểm thử và phân tích kết quả dựa trên các hành vi của Website để đưa ra các cảnh báo (Alert) cho người dùng.

Passive Scan sẽ chỉ sử dụng các phản hồi của proxy để lấy tông tin, thường các lỗ hổng liên quan đến lộ thông tin sẽ được phát hiện bằng trình quét này.

Active Scan sẽ chủ động gửi các truy cấn để phát hiện ra lỗ hổng dựa vào các hành vi của Web trả lại. Nhược điểm của trình quét này sẽ gửi số lượng lớn và các truy vấn tới mọi Site Node của Website có thế dẫn đến các hành vi không mong muốn đối với Website mục tiêu.

Các luật liên tục được cộng đồng duy trì và phát triển theo các lỗ hổng đã xuất hiện và hiện hành.

2.2.2 Invicti

Invicti hay trước đây là NetSparker là một công cụ rà quét lỗ hổng tự động được xây dựng bằng cách cung cấp các bằng chứng về việc khác thác lỗ hổng luôn trong quá trình rà quét. Điều này khiến cho các lỗ hổng thu thập được trong quá trình rà quét có độ tin cậy cao hơn rất nhiều so với các công cụ rà quét tự động khác. Ngoài ra, các thành phần khác như tổ chức quét tự động, thiết lập cầu hình quét, sinh báo cáo và cung cấp công cụ khai thác/hậu khai cũng giúp cho Invicti là một công cụ được sử dụng nhiều trong các doanh nghiệp.

2.2.3 Các công cụ khác

Trên thị trường, hiện nay còn có rất nhiều công cụ được sử dụng trong việc rà quét lỗ hổng như Acunetix và BurpSuite.

Acunetix là cũng là một công cụ rà quét tự động được sử dụng phổ biến trong doanh nghiệp. Điểm nổi trội của Acunetix là sử dụng công nghệ AcuSensor cho phép kiểm thử bằng cách thực thi ngay tại máy chủ Web mã phía máy chủ để cung cấp thêm thông tin cho trình phân tích. Công nghệ này được giới thiệu rằng có thể đưa ra các lỗ hổng có tính xác thực đến gần 100% các lỗ hổng như SQL Injection, Code Injeciton, File Inclusion...

BurpSuite là một trình quét sử dụng dựa trên cơ chế "middle-in-the-middle-proxy" tương tự ZAP nhưng sử dụng nhiều các thành phần được phát triển bởi các nhà nghiên cứu của BurpSuite với kì vọng đem lại tất cả các tính năng thông qua chúng. BurpSuite thường được sử dụng với các nhà nghiên cứu và săn tiền thưởng.

2.3 Các công cụ khai thác lỗ hổng

2.3.1 Công cụ SQL Map

SQLMap là Một công cụ mã nguồn mở được sử dụng để tự động quá trình xác định, khai thác và hậu khai thác các lỗ hổng về SQL Injection.

SQLMap được cộng đồng phát triển với một engine có khả năng phát hiện các lỗ hổng ở một miền lỗ hổng cực kỳ rộng và cho rất nhiều hệ thống khác nhau. Đồng thời nó cũng có các công cụ khai thác dựa trên các lỗ hổng được phát hiện trên dump cơ sở dữ liệu hiệu quả.

Hiện nay SQLMap, hỗ trợ gần như các cơ sở dữ liệu phổ biến như MySQL, Oracle, PostgreSQL, Microsoft SQL Server, Microsoft Access, IBM DB2, SQLite, Firebird, Sybase, SAP MaxDB, Informix, MariaDB, MemSQL,

TiDB, CockroachDB, HSQLDB, H2,... với rất nhiều kỹ thuật như boolean-based blind, time-based blind, error-based, UNION query-based, stacked queries and out-of-band. Ngoài ra, SQLMap giúp liệt kê bằng các thông tin của cơ sở dữ liệu nếu users, password hashes, privileges, roles, databases, tables and columns.

2.3.2 Công cụ Commix

Commix (viết tắt của command injection exploiter) là một công cụ mã nguồn mở được sử dụng để tự động xác định, khai thác và hậu khai thác các lỗ hổng về Command Injection. Một số lỗ hổng liên quan đến Remote Code Evaluation cũng có thể được phát hiện và khai thác bằng công cụ này.

Commix cũng được cộng đồng phát triển để có khả năng khai thác một miền rộng các lỗ hổng trên nhiều hệ điều hành khác nhau và có thể thực hiện hậu khai thác ở đây.

2.3.3 Công cụ Tplmap

Tplmap là một công cụ mã nguồn mở được sử dụng để xác minh và khai thác các lỗ hổng liên quan đến Code Injection and Server-Side Template Injection.

Hơn 15 ngôn ngữ, template engine, có thể khai thác với nhiều kỹ thuật khác nhau kể các các lỗ hổng dạng mù (blind).

2.3.4 Công cụ Nuclei

Một công cụ mã nguồn mở được sử dụng để rà quét lỗ hổng theo mẫu rà quét (Template). Đối với các lỗ hổng phức tạp cần dạng các mắt xích của nhiều lỗ hổng để khai thác thì các công cụ rò quét thông thường không thể phát hiện được đăc biệt các dạng CVE. Nuclei được phát triển để có thể mô hình được tất cả các lỗ hổng kể cả các lỗ hổng phức tạp thành qua mẫu rà quét.

Số lượng mẫu rà quét được các nhà nghiên cứu bảo mật và cộng đồng cập nhật liên tục khiến đây là công cụ rất hiệu quả để phát hiện các lỗ hổng trong các sản phẩm, thư viện phổ biến hiện nay (tính đến tháng 05/2020 khoảng 3200 mẫu).

Chương 3

Phát triển NAF - Một công cụ rà quét và xác minh tự động các lỗ hổng Web

3.1 Mô hình tổng quan

Dù rằng ZAP đã cung cấp một mô hình tương đối hoàn chỉnh để thực hiện được việc kiểm thử an toàn thông tin, nhưng phần lớn các ZAP Addon lại rất phụ thuộc vào việc được cung cấp thông tin về Website để mô hình thành cây Site Node từ cơ chế “middle-in-the-middle-proxy" tức là người kiểm thử an ninh cần sử dụng trực tiếp trên trình duyệt Web một thời lượng lớn dù rằng các ZAP Addon có thể thay thế một phần việc này.

Một vấn đề khác là do các ZAP Addon được thiết kế để chạy được độc lập nên sự tương tác giữa các ZAP Addon chưa cao dù rằng chỉ với một số Addon trong ZAP Core đã có thể hoàn thành lượng lớn công việc đánh giá lỗ hổng.

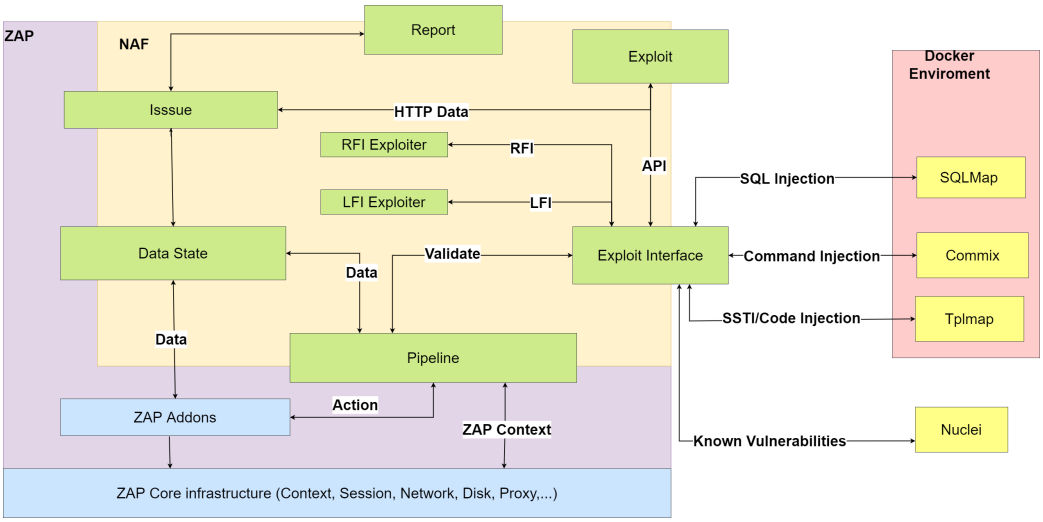
ZAP cũng chỉ xây dựng được cơ chế cảnh báo từ việc thực hiện đánh giá lỗ hổng thông qua các luật, điều này là không đủ để sinh ra được báo cáo hoàn chỉnh mục đích đưa ra được Proof-of-Concept cho lỗ hổng, cũng như cách xử lý cho nhà phát triển.

Ngoài ra, các ZAP Addon cũng không thể cung cấp được các công cụ quét lỗ hổng chuyên sâu hoặc lỗ hổng mức hệ thống và thiếu vắng đi các công cụ khai thác tự động để đưa ra được Proof-of-Exploit cho lỗ hổng.

Do đó, công cụ AVA được sinh ra để thống nhất việc tương tác giữa các Addon để nâng mức tự động lên mức độ chấp nhận được và bổ sung thêm các chi tiết, tính năng thiếu đã nêu ở trên để phù hợp đây là một công cụ tự động xác minh và khai thác tự động.

3.1.1 Các thành phần

Các thành phần sẽ được mô hình hóa thành các thành phần. Mỗi thành phần sẽ phụ trách một Logic riêng theo từng phần của công cụ. Các thành phần này phụ thuộc nhau dạng cây như hình 3.1 và được tổ chức mã nguồn như hình 3.2.



Hình 3.1: Biểu đồ phụ thuộc của các thành phần

Đối với ngôn ngữ triển khai, em lựa chọn Kotlin là một ngôn ngữ JVM có thể tương tác được với Java.

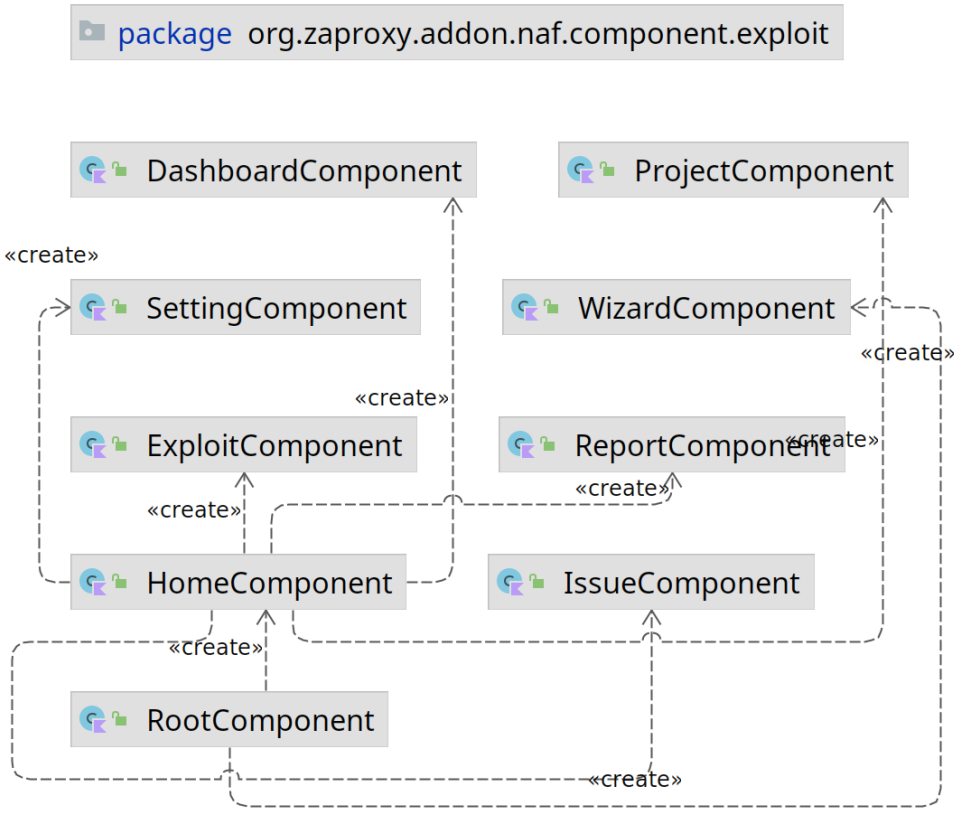
Về triển khai giao diện người dùng và tổ chức mã nguồn, em lựa chọn Jetpack Compose và Decompose làm các thư viện chính và tổ chức thành phần thay cho Swing - một thư viện khá cũ và khó tổ chức khi đưa cả logic và giao diện người vào chung một mã nguồn.

3.2 Chức năng kiểm thử theo đường ống

Như đã nêu vấn đề ở trên, ZAP sử dụng các Addon như một phần nên việc thích hợp các công cụ xác minh tự động khá khó. Do đó, em xây dựng một đường ống (Pipeline) để tương tác được với ZAP Core, sử dụng được với các ZAP Addons khác, đồng thời có thể thêm các chi tiết khác vào quá trình kiểm thử tự động.

Mỗi đoạn ống sẽ thuộc về một pha xác định của kiểm thử, pha có độ ưu tiên cao được thực hiện trước để cung cấp thông tin cho các pha sau.

Các pha bao gồm: Khởi tạo, Kiểm thử mờ, Thu thập dữ liệu, Rà quét lỗ hổng.



Hình 3.2: Tổ chức component trong mã nguồn

Các kiến trúc này xoay quanh các khai niệm sau:

- Ngữ cảnh quét (Context) là thành phần chứa thông tin của các cuộc tìm kiếm lỗ hổng tự động, các đoạn của đường ống sẽ sử dụng thông tin này để thực hiện các hoạt động của mình một cách động lập trên toàn đường ống. Ngữ cảnh quét ánh xạ tới các thành phần ZAP Core.
- Đường ống (Pipeline) là quá trình liên tục chứa các đoạn, tại đây diễn ra việc tìm lỗ hổng tự động và xác minh tự động.
- Các đoạn của đường ống (Stage) là các đơn vị chạy các việc để thực hiện quá trình kiểm thử tự động.

3.2.1 Khởi tạo

Đối với các đoạn của đường ống là đoạn khởi tạo ngữ cảnh thì cần có tương tác trực tiếp với cả ZAP Core và cả đường ống.

Các tương tác này được xử lý như sau

- Thiết lập ngữ cảnh riêng cho việc kiểm thử: Mỗi lần kiểm thử tự động nên có một ngữ cảnh riêng ứng với đó là một ngữ cảnh của ZAP riêng chứa các thông tin về thông tin cho việc sử dụng Proxy, Crawler, Scanner,...

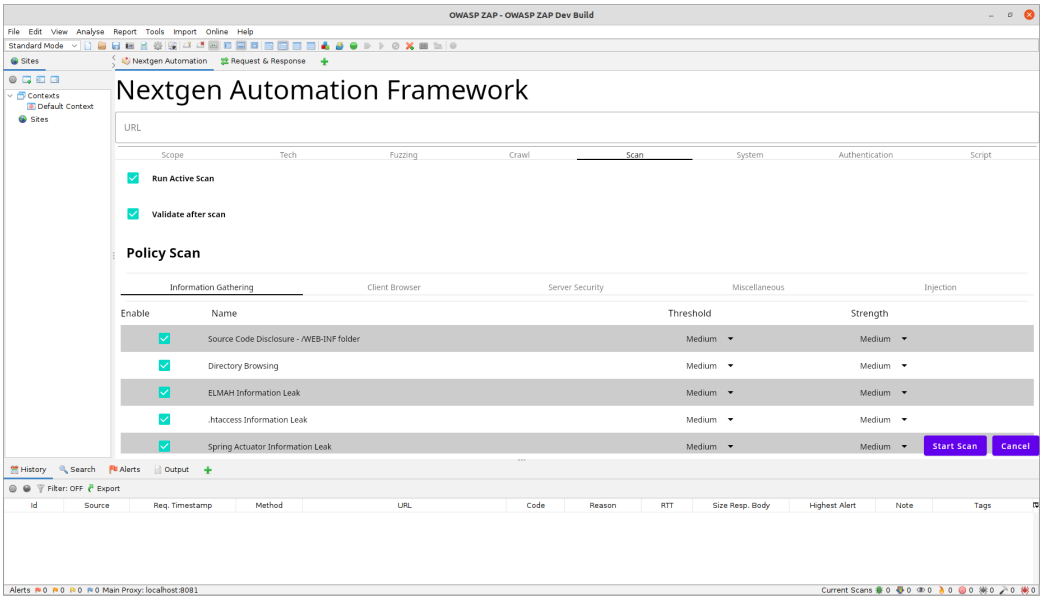
- Thiết lập mục tiêu (Target) cho kiểm thử: Thực hiện truy cập vào truy vấn tới URL mục tiêu, cung cấp cho hệ thống thêm thông tin về Site Node gốc, đánh dấu điểm bắt đầu của Sitemap.

- Thông tin về các ca kiểm thử được sử dụng. Thông tin được đánh dấu dưới dạng Chính sách (Policy) 3.3

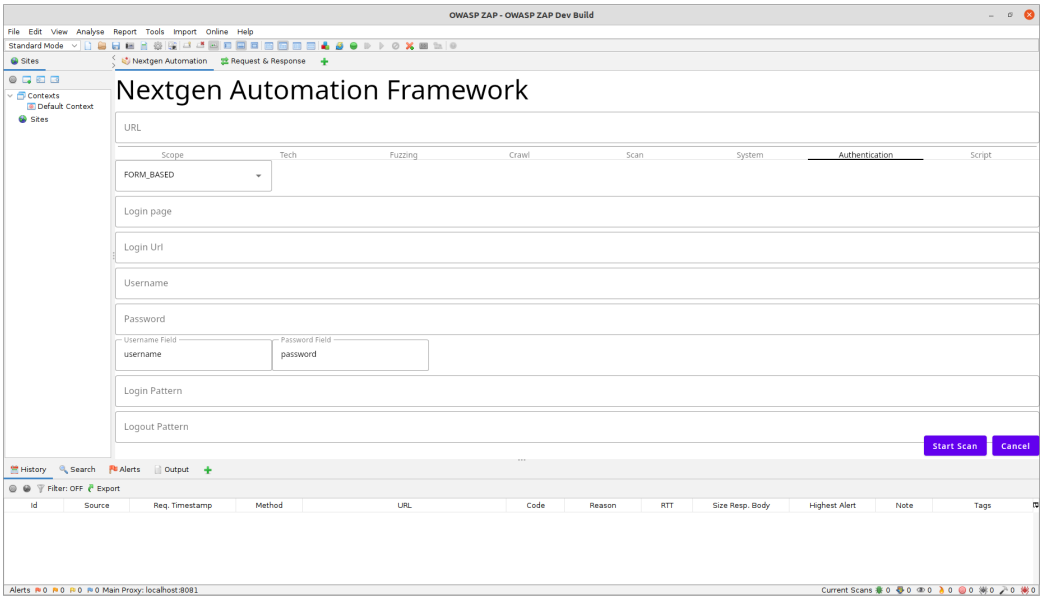
  - Về công nghệ sử dụng: Được cung cấp qua tập công nghệ sử dụng (Tech Set) trong ZAP Context, Scanner sẽ sử dụng thông tin này để lựa chọn phù hợp để kiểm thử, đối với trường hợp thông thường tập công nghệ được đánh dấu toàn bộ sẽ khiến lượng ca kiểm thử trở lên rất lớn.
  - Về ngưỡng cảnh báo (Threshold): Các ca kiểm thử đều có ngưỡng cảnh báo riêng dựa trên hành vi của hệ thống đối với ca kiểm thử, mức độ càng thấp tăng lượng âm tính giả càng lớn ngược lại làm tăng mức dương tính giả.
  - Mức độ cảnh báo (Strength): Thiết lập để phù hợp với các mục tiêu kiểm thử, mức độ này sẽ được trả về hệ thống thông qua cơ chế Thông báo (Alert)

- Thông tin về xác thực: Cần được thực hiện một lần đăng nhập để lấy thông tin để cung cấp đầy đủ thông tin cho ZAP Context 3.4. ZAP Context sẽ "hướng dẫn" Crawler, Scanner thực hiện đăng nhập khi bị đăng xuất. Các thông tin cần được cung cấp bao gồm:

  - Trang đăng nhập
  - Thông tin về yêu cầu đăng nhập
  - Thông tin ủy nhiệm (Credential)
  - Biểu hiện khi đã đăng nhập/đăng xuất.



Hình 3.3: Điều chỉnh các các chính sách



Hình 3.4: Đưa ra thông tin xác thực

Với các thông tin trên nên được thực hiện tự động và ẩn đi phần logic phức tạp trên, người dùng chỉ nên được tương tác thông qua một trình Wizard.

Các thông tin này được em triển khai bằng cách giữ lại nhưng một trạng thái có thể lưu lại, chỉ được thực hiện khi bắt đầu quá trình kiểm thử tự

động. Khi kiểm thử tự động sẽ dựa vào các thông tin này để sinh ra các thiết lập để đưa vào ZAP Core một cách tự động.

3.2.2 Kiểm thừ mờ

Một việc cần thiết là để ZAP có thể hoạt động hiệu quả cung cấp thêm thông tin về cấu trúc của Website cho ZAP. Một trong những phương pháp phổ biến cho việc này là vét cạn các điểm cuối thường có của một Website theo từ điển (thường được sử dụng với tên Fuzzing). Kỹ thuật này hiệu quả với việc kiểm thử hộp đen và hộp xám khi không có hoặc có rất ít các thông tin về hệ thống. Ngoài ra, nó cũng giúp phát hiện ra có lỗ hổng do việc thiết lập cấu hình không an toàn.

Cộng đồng phát triển ZAP đã phát triển một Addons là Forced Browsing trên cơ sở sử dụng thư viện DirBuster có hiệu quả khá cao trong việc kiểm thử mờ cũng như cung cấp một vài danh sách các từ phổ biến cho việc kiểm thử mờ. Đoạn của đường ống này sẽ sử dụng Addon trên và cung cấp danh sách các từ (người dùng lựa chọn) để thực hiện việc kiểm thử này. Việc này cũng được thiết đặt tự động trong trình Wizard thay vì phải thiết lập thủ công 3.5.

Đây là một trong đoạn đầu tiên của đường ống, nên chưa có nhiều thông tin về ngữ cảnh được cung cấp, nên chỉ cần dựa trên danh sách các từ được người dùng lựa chọn và các tùy chọn để thực hiện kiểm thử mờ và cố gắng thu thập lại lượng endpoint phù hợp để sinh ra lượng Site Node lớn nhất để có thông tin về Website.

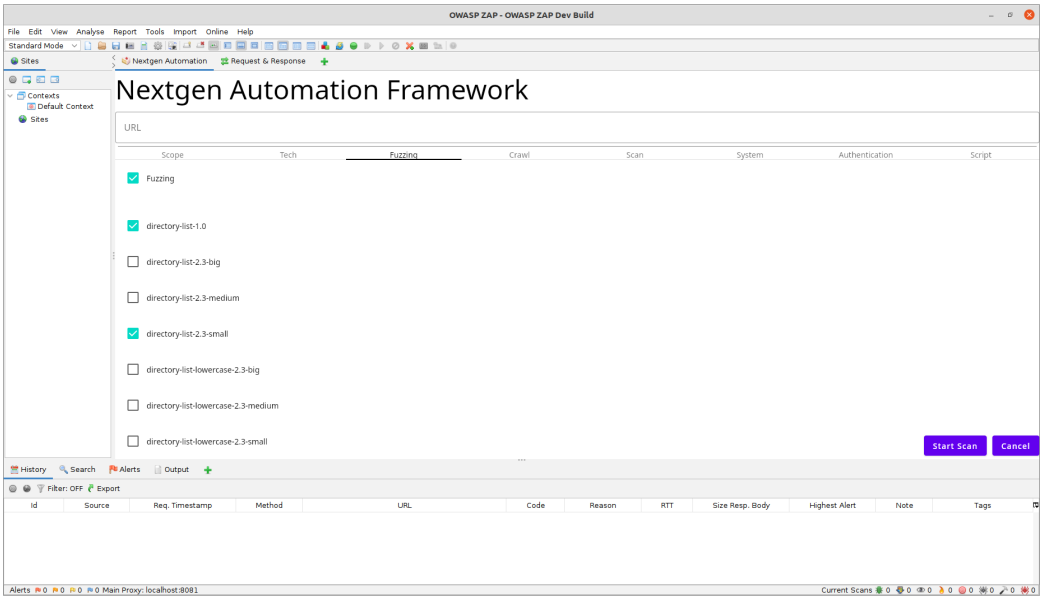
3.2.3 Thu thập dữ liệu

Một cách nữa để cung cấp thông tin về cấu trúc của Website là thu thập (Crawl) dữ liệu trên toàn bộ Website để lấy thông tin.

Đội ngũ phát triển ZAP cũng đã phát triển hai Addon là Spider và Ajax Spider. Spider được sử dụng để thu thập dữ liệu, thông tin lấy từ cây DOM của Website trong các phản hồi của các thông báo HTTP.

Đối với một số Website sẽ sử dụng thêm Ajax để lấy thông tin từ máy chủ, các trình thu thập dữ liệu tĩnh như Spider sẽ không phát hiện được các yêu cầu này. Đội ngũ phát triển ZAP cũng đã phát triển thêm Ajax Spider, công cụ xây dựng dựa trên Selenium để thực hiện việc thu thập dữ liệu trực tiếp bằng Selenium đấy được thông tin về các yêu cầu thực hiện thông qua Ajax bằng cách thực thi Javascript như một trình duyệt thực.

Hai Addon này cần sử dụng trực tiếp các thông tin từ ngữ cảnh của ZAP Core. Để tự động được các Addon này cần đồng thời xử lý thông tin ở ZAP



Hình 3.5: Tùy chọn danh sách các từ để kiểm thử mờ

Context trước khi gửi thông tin cho Addon và thực thiện gọi các Addon thực hiện theo trình tự của của các pha.

Các đoạn của đường ống này có mục tiêu là tối ưu hóa việc thu thập dữ liệu từ DOM và các yêu cầu Ajax. Để chúng hoạt động tốt nhất cần cung cấp ZAP Context chứa các thông tin về phạm vi và thông tin đăng nhập để quá trình thu thập dữ liệu không bị hủy bỏ do vấn đề xác thực.

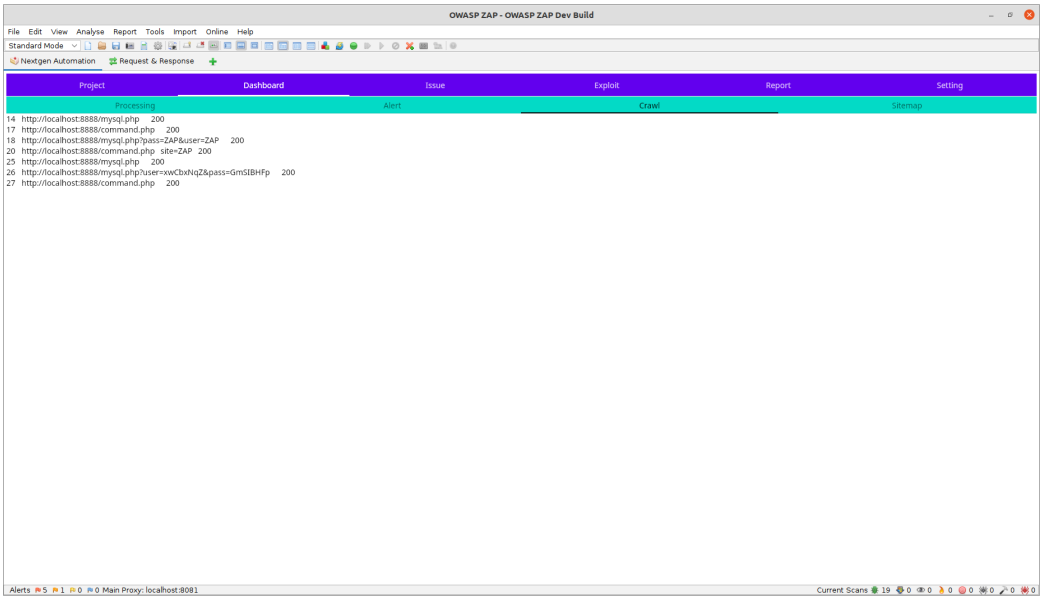
3.2.4 Rà quét lỗ hổng

Đây là pha chính trong việc kiểm thử an ninh, tại đây các luật (Rule) của cộng đồng đóng góp sẽ được sử dụng để kiểm tra các hành vi của hệ thống để suy diễn ra các lỗ hổng của hệ thống. Cộng đồng phát triển ZAP đã phát triển hai Addons là Passive Scan và Active Scan. Tuy nhiên, Addon Passive Scan lại chỉ phù hợp khi thực hiện kiểm thử thủ công hoặc bán tự động do chỉ rà quét các thông tin bằng cách rà quét thụ động. Addon Active Scan là công cụ hiệu quả cho việc chạy các luật được đưa ra của chính sách trong ngữ cảnh của ZAP. Chính công cụ này sẽ thực hiện việc thử các luật, loại bỏ bớt các luật dựa trên các tập công nghệ được đưa vào, lựa chọn các ca kiểm thử phù hợp với ngưỡng quét và đưa ra cảnh báo dựa trên mức độ cảnh bảo. Ngoài ra, cần thích hợp thêm công cụ để phát hiện được các lỗ hổng phức tạp hơn.

Đoạn của đường ống hoạt động này dựa trên chính sách đã được cung cấp

để đưa ra cảnh báo. Quá trình này cũng cần được cung cấp ZAP Context để có thể thực hiện ca kiểm thử mà không hủy bỏ do vấn đề xác thực. Đồng thời cũng nên đưa vào các vùng nguy hiểm trong phạm để tránh làm lỗ hệ thống (ví dụ các thiết lập tài khoản, hệ thống).

Việc cài đặt bước này cần đem được các thông tin từ pha trước kết hợp với thông tin người dùng cung cấp để thực hiện. Các Addons tương đối độc lập nên thông tin đầu ra của từng Addons là khác nhau và không ổn định, các thông tin đầu ra được thu thập, chuẩn hóa và đưa lại vào trong AVA để trực quan hóa thông tin 3.6.



Hình 3.6: Kết quả crawl

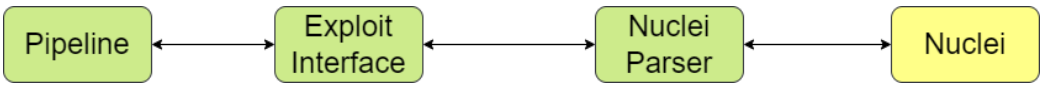
3.2.5 Rà quét lỗ hổng của thành phần bên thứ ba

Đối với các thành phần thứ ba như thư viện, khung phát triển hay các máy chủ Web, thì việc sử dụng các thành phần cũ có thể khiến ứng dụng Web có thể bị khai thác dù rằng mã nguồn đã an toàn. Việc phát hiện các lỗ hổng này cũng không thế phát hiện qua các công cụ rò quét thông thường. Nuclei là một công cụ rà quét lỗ hổng cho các trường hợp phức tạp và thường được các kĩ sư an toàn thông tin sử dụng cho trường hợp này.

Để thích hợp được Nuclei vào công cụ, cần xây dựng một API giao tiếp với Nuclei. Do được xây dựng từ ngôn ngữ Go nên Nuclei là một tập tin khởi chạy nhị phân và cần bắt lấy đầu và chuẩn và đầu ra chuẩn để lấy được dữ

liệu. Dữ liệu đầu vào cũng cần được chuẩn hóa từ dạng JSON vào trong hệ thống, đưa thành các cảnh báo đã được xác minh.

Mô hình xử lý có thể biểu diễn như 3.7:



Hình 3.7: Biểu đồ tương tác giữa Nuclei và Đường ống

Trong đó, Nuclei Parser sẽ làm đồng thời cả hai nhiệm vụ lắng nghe các thông tin đầu ra của Nuclei và xử lý thông tin đó gửi trả lại cho đường ống để cập nhật các thông tin trên thông qua API.

3.2.6 Thích hợp công cụ xác minh tự động

Các công cụ xác minh tự động làm nhiệm vụ xác minh lại các cảnh bảo, nếu được xác minh, một Vấn đề (Issue) mới được tạo ra và được thêm vào cơ sở dữ liệu.

Để xác định được cảnh báo nào thuộc về lỗ hổng nào cần sử dụng đến giá trị định danh của Common Weakness Enumeration (CWE) để xác định tương đối lỗ hổng nào thuộc về loại nào.

Cần xem quét cách mà các luật được sử dụng để đưa ra được cách xác minh phù hợp với từ loại lỗ hổng. Trong phạm vi của công cụ, các lỗ hổng đã được xem xét thông qua mã nguồn của các luật được sử dụng trong rà quét chủ động bao gồm:

SQL Injection Luật thiết lập ZAP chủ động gửi các trọng tải mang gây lỗi cho các truy vấn SQL như ', \, ;, (,), NULL, ... với các trọng tải khác phù hợp với các kĩ thuật như Error-based, Boolean-Based, Union-Based,... Và sẽ lấy các thông báo phản hồi bất thường như các lỗi, thông tin trả về như một Proof-Of-Concept để thông báo tới người dùng.

Command Injection Luật cũng đưa vào các dấu ngắt lệnh như ', ;, | ... để cố gắng chạy thêm các lệnh phù hợp với từ hệ điều hành. Luật cũng sử dụng các hành vi của máy chủ Web để xem xét đưa ra Proof-Of-Concept phù hợp với các kĩ thuật tấn công như Feedback-based hay Blind Time-based.

Code Injection Luật chỉ tập trung vào việc xác minh Remote Code Evaluation. bằng cách chèn thêm các trọng tải chứa các đoạn mã thực thi đưa vào phản hồi các đoạn đánh dấu đặc biệt và kiểm tra xem phản hồi có cả đoạn mã này hay không.

Path Traversal Luật đưa gửi các trọng tải liên quan đến việc truy cập các tập tin, đặc biệt của hệ điều hành như /etc/passwd, hay system.ini, khi các thông tin của tập tin được trả về thì có thể xác định ngay xem có bị lỗ hổng hay không.

Remote File Include Luật đưa vào một đường dẫn điển hình tới chứa trọng tải tới một trang Web và cố xác định xem trang Web đó có được thêm vào hay không. Tuy nhiên luật này sẽ có trường hợp dương tính giả với lỗ hổng Server-side Request Forgery do không xác định xem việc thêm vào có được thực thi hay không.

Qua các triển khai của luật trên, các lỗ hổng cần được xác minh cần được phân phối cho các công cụ và được triển khai như sau:

SQL Injection SQLMap

Command Injection/Code Injection Commix

Code Injection Tplmap

Remote File Inclusion RFIExploiter

Local File Inclusion/Path Traversal LFIExploiter

3.2.6.1 SQLMap

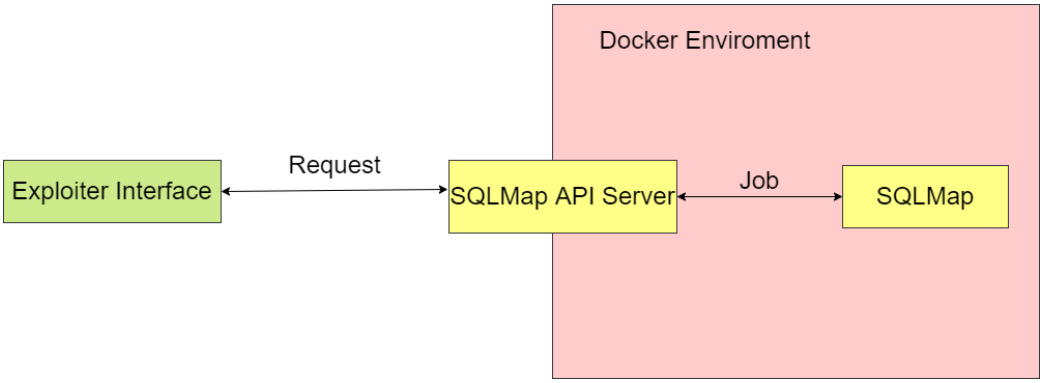
Công cụ SQLMap, đi kèm với một API Server nên việc giao tiếp có thể thiết lập qua công API Server này. API Server sử dụng là REST-API sử dụng JSON là format để truyền tải thông tin. Tuy nhiên dữ liệu trả về không là có định dạng chuẩn do SQLMap được xây dựng bằng Python là một ngôn ngữ kiểu động. Do đó, cần xây dựng thêm một số Parser riêng để có thế đưa dữ liệu trả về đúng kiểu 3.8.

Lúc này các nhiệm vụ khai thác được mô hình hóa thành các Job, API Server sẽ xử lý các Job trong Job Pool và trả về dữ liệu theo các yêu cầu của công cụ gửi lên. Các yêu cầu có thể là xác minh lỗ hổng, lấy các thông tin của cơ sở dữ liệu.

Đối với pha xác minh, SQLMap chỉ cần xác minh là lỗ hổng có tồn tại hay không dựa trên thông tin cảnh báo đưa ra.

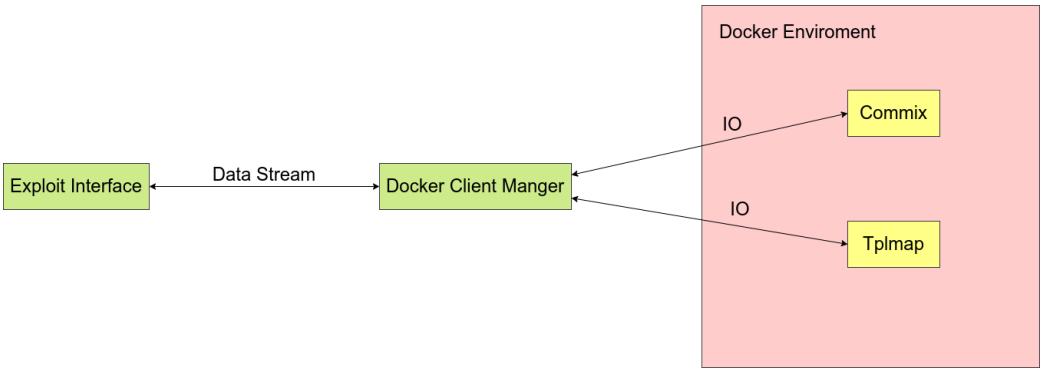
3.2.6.2 Commix và Tplmpa

Khác với SQLMap, Commix và Tplmap không có một API Server, các tương tác cần được đưa vào và ra thông qua đầu vào chuẩn (STDIN) và đầu ra chuẩn(STDOUT) 3.9.



Hình 3.8: Biểu đồ tương tác giữa SQLMap và công cụ.

Đối với Docker Engine cũng cần có API giao tiếp gián tiếp để có thể thực thi và lấy được đầu vào chuẩn và đầu ra chuẩn của Container.



Hình 3.9: Biểu đồ tương tác công cụ với Commix và Tplmap.

Đối với việc xác minh, Commix và Tplmap sẽ cố gắng thực thi một lệnh và kiểm tra lệnh có được thực thi hay không bằng kiểm tra đầu ra chuẩn để xác minh.

3.2.6.3 Remote File Inclusion

Lỗ hổng này là đặc trưng của PHP nên sẽ chỉ được kiểm tra với PHP. Đưa đường dẫn tới một pha đầu vào chứa một đoạn mã thực thi của PHP, nếu được thực thi thì được xác minh là đúng. Với lỗ hổng này không có các dự án mã nguồn mở để khai thác nên em cần tự triển khai một trình khai thác riêng.

3.2.6.4 Local File Inclusion/Path Traversal

Lỗ hổng này sẽ không cần xác minh lại do độ, nhưng vẫn được xây dựng để phát triển công cụ khai thác. Lỗ hổng này cũng không có dự án mã nguồn mở để khai thác nên em cũng tự triển khai một trình khai thác riêng.

3.3 Chức năng hiển thị

Dù là các ZAP Addons cung cấp khá đầy đủ các thông tin về thông tin của chính Addons đó cho người dùng nhưng chính sự độc lập của các Addons lại khiến thông tin của toàn bộ ứng dụng ZAP nói chung lại bị phân tán thành tại các vị trí khác nhau. Đồng thời Addon lại chia sẻ rất ít thông tin cho các Addon khác.

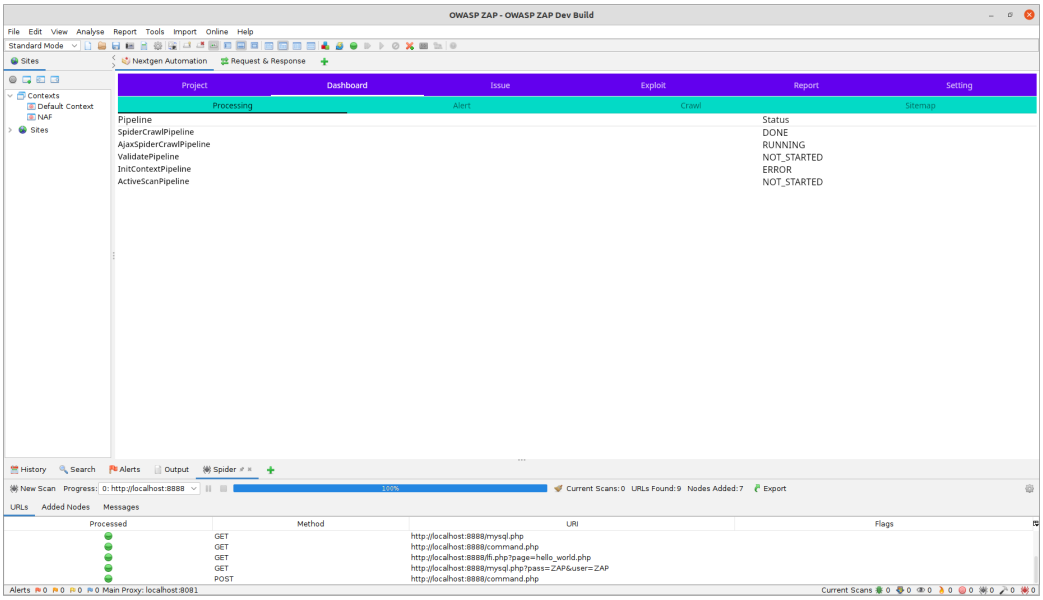
Điều này làm nảy sinh một nhiệm vụ cần được xây dựng là tập hóa các thông tin về quá trình kiểm thử, thông tin thu thập được thành một trang có sức trực quan hóa cao để hiệu quả trong việc theo dõi.

Kiến trúc Event Bus của ZAP Core và các thông tin được cung cấp thêm trực tiếp các Addon khác và các thành phần được thêm vào. Cần thu thập thông tin lượng vừa đủ và chuẩn hóa các thông tin thu thập được thành một miền cố định để có thể trực quan hóa. Như đã nói ở trên, Addons rất ít chia sẻ thông tin ra ngoài nên lượng thông tin em cần thực hiện qua một số kĩ thuật khác với luồng hoạt động thông thường của ZAP để có thể tổng hợp được nhiều và đầy đủ hơn lượng thông tin 3.10.

3.4 Chức năng quản lý sự kiện và sinh báo cáo

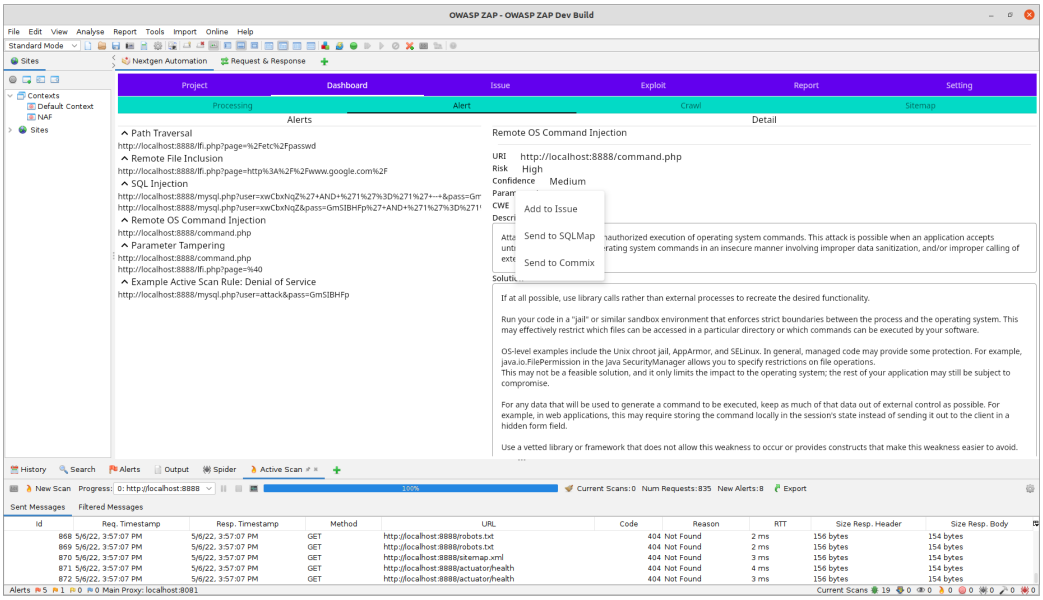
Theo luồng được cộng đồng ZAP định ra, thì cảnh báo (Alert) là đơn vị được sử dụng để quản lý các lỗ hổng được sinh ra từ ZAP, tuy nhiên điều này hơi không phù hợp do hai lý do. Một là lượng cảnh báo mà ZAP sinh ra rất lớn và được sinh ra liên tục dẫn đến cần được xác thực lại liên tục. Hai là mục tiêu cuối cùng là cần sinh ra được báo cáo bao gồm thông tin chi tiết, cách tái hiện và gợi ý về cách xử lý cho phía phát triển xử lý. Điều này dẫn để cần chuẩn hóa lại các miền để phù hợp với mục đích cuối cùng là giúp nhà phát triển và vấn hành xử lý được các lỗ hổng tồn tại.

Từ đó em đã mô hình hóa lại thành sự kiện (Issue) để xử lý hai vấn đề ở trên: chỉ sử dụng cảnh báo nhưng một nguồn và định thông tin tiêu chuẩn để báo cáo xử lý và cho phép xuất ra một bản báo cáo dễ hình hơn để cung cấp đủ thông tin về lỗ hổng 3.11.



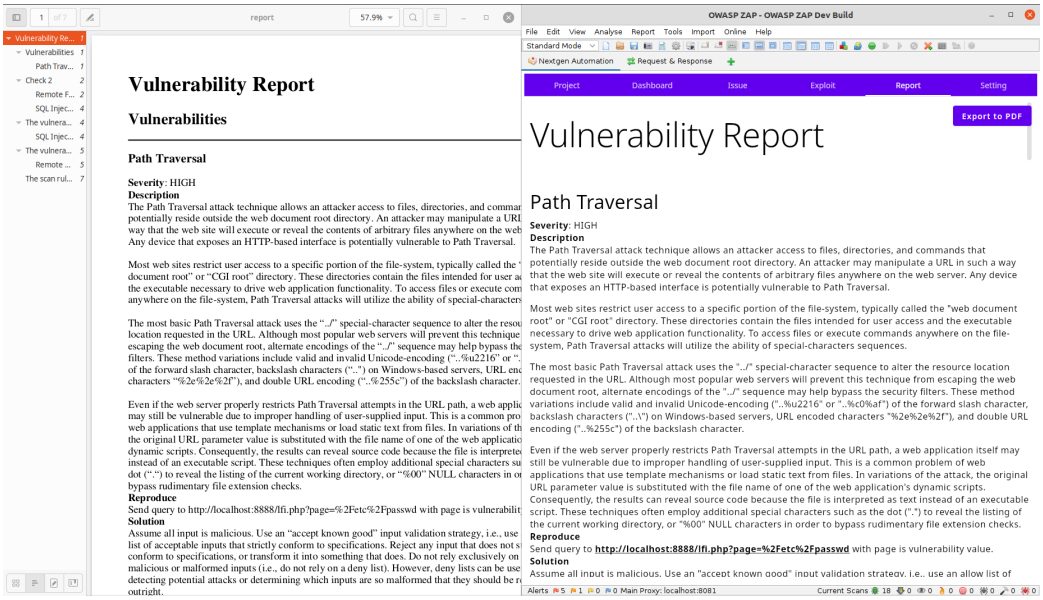
Hình 3.10: Thanh trạng thái pipeline đang chạy

Các cảnh báo có thể gửi tới các tab khai thác.



Hình 3.11: Quản lý alert chung của các công cụ thích hợp

Báo cáo sinh ra cho phép sử dụng Issue không phải trạng thái UNKNOWN để tạo ra một File PDF chứa báo cáo với format định trước 3.12.



Hình 3.12: Sinh báo cáo cho các lỗ hổng.

3.5 Chức năng khai thác và hậu khai thác

Ngoài việc xác minh tự động trong đường ống, các công cụ thích hợp còn có thể sử dụng như một công cụ khai thác tự động trong ZAP. Việc thích hợp sử dụng chung các thành phần chính với công cụ xác minh tự động.

Để thuận tiện cho việc khai thác thì có thể tạo ra nhiều nhãn (Tab) khai thác khác nhau, mỗi nhãn này là các yêu cầu khai thác khác nhau. Tuy nhiên, đặc điểm chung là các nhãn này sẽ không bao giờ được lưu lại khi hết phiên để tránh lộ thông tin do các vấn đề về an toàn.

3.5.1 SQLMap

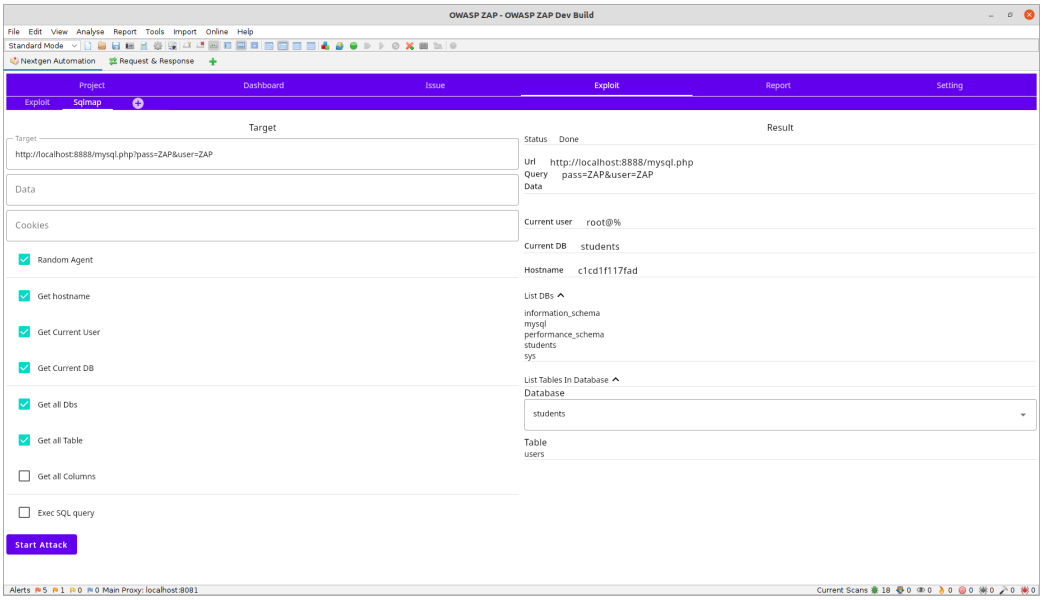
Với lỗ hổng này, người dùng có thể khai thác từ một cách tự động bằng cách trích xuất các thông tin từ lỗ hổng 3.13.

Dữ liệu đầu vào có thể gửi trực tiếp từ các cảnh báo tới, các dữ liệu như URL, Cookie, Data sẽ được gửi cùng sang để việc khai thác trở nên ngắn gọn hơn hoặc tự chỉnh sửa để dễ dàng xác nhận hơn.

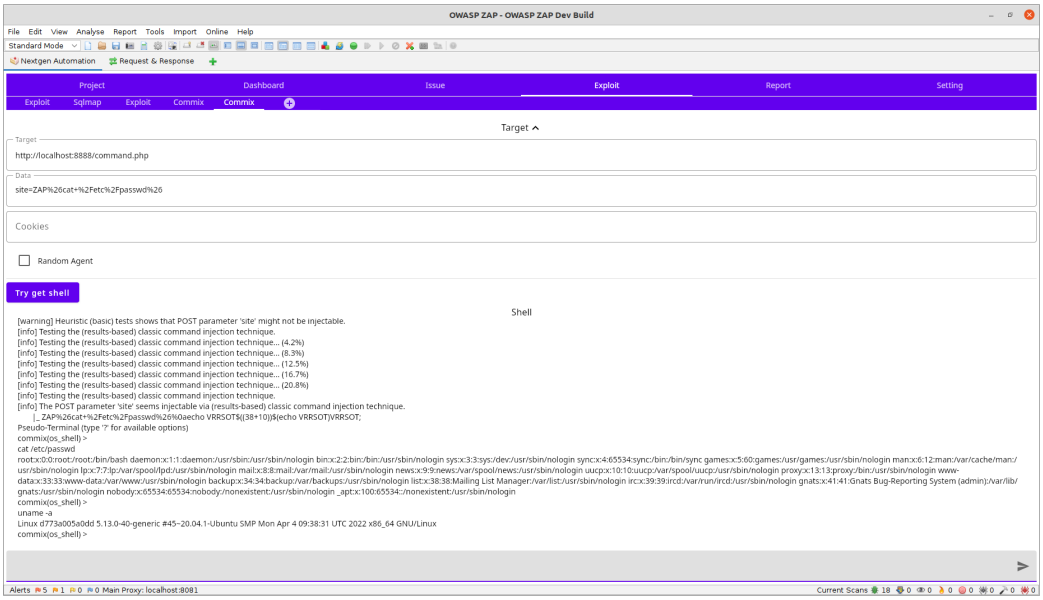
3.5.1.1 Commix

Dữ liệu đầu vào cũng có thể gửi từ cảnh báo sang tương tự SQLMap.

Các câu lệnh sẽ được gửi tới Commix và nhận về output tương tự một trình pseudo-shell 3.14.



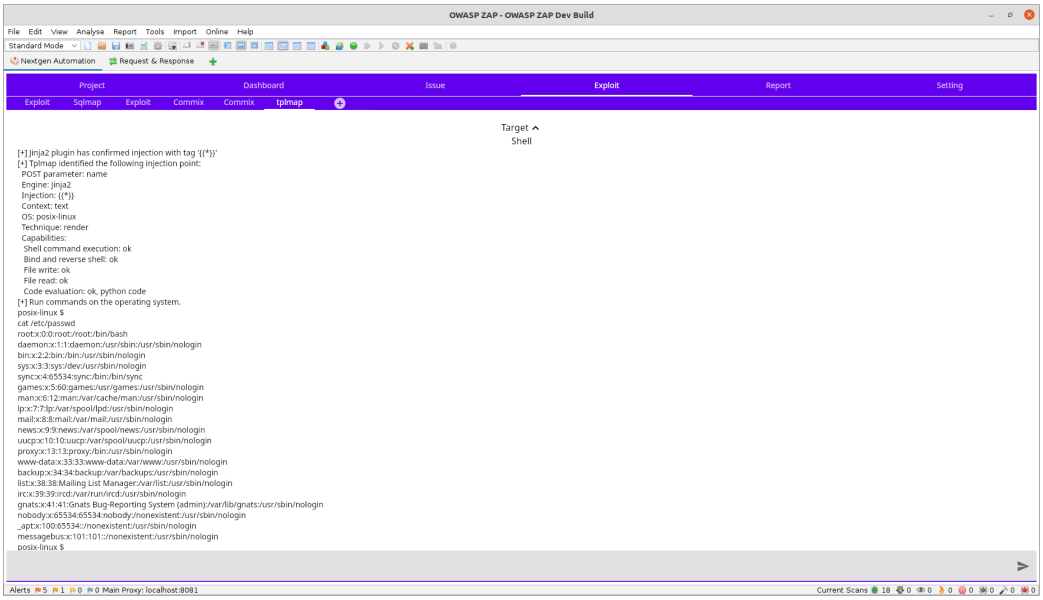
Hình 3.13: Khai thác lỗ hổng SQL Injection thông qua SQLMap



Hình 3.14: Khai thác lỗ hổng OS Command Injection bằng Commix

3.5.2 Tplmap

Tương tự SQLMap và Commix, Tplmap câu lệnh khởi tạo có thể được gửi sang từ một cảnh báo. Các câu lệnh trong Tplmap cũng được cài đặt để nhận một đầu vào và một đầu ra nhưng một trình pseudo-shell 3.15.



Hình 3.15: Khai thác lỗ hổng SSTI bằng Tplmap

3.5.3 Remote File Inclusion

Việc khai thác Remote File Inclusion PHP được em triển khai bằng cách thông qua một tập tin thực thi tự triển khai được công khai trên Internet. Trong đây sẽ có một số tham số cho phép mình thực thi lệnh, các đầu ra của lệnh sẽ được gửi về bằng cách phân tích phản hồi của yêu cầu 3.16.

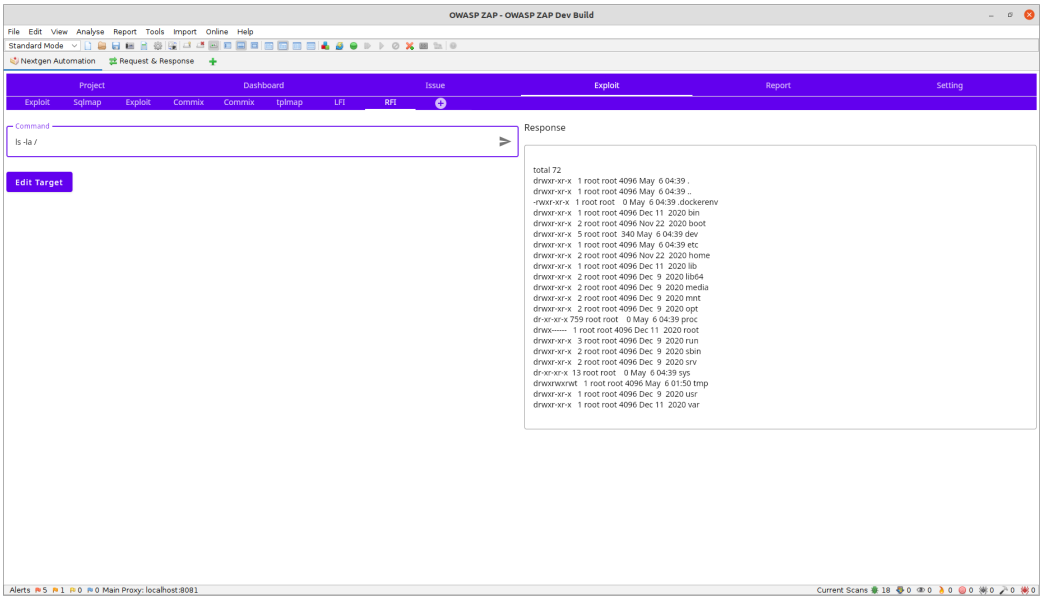
3.5.4 Local File Inclusion/Path Traversal

Trình khai thác lỗ hổng Local File Inclusion/Path Traversal được em triển khai bằng cách cho phép người dùng lựa chọn một số đường dẫn và thay đổi các đường dẫn để lấy dữ liệu về, người dùng có thể xem dữ liệu trả về thông qua việc quan sát phản hồi 3.17.

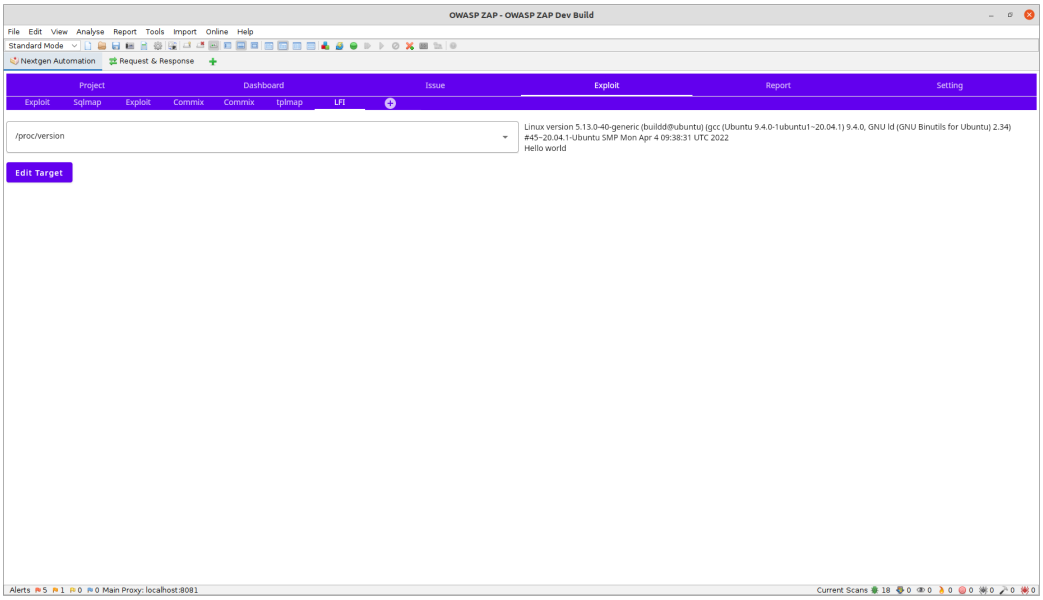
3.6 Đóng gói các thành phần

Ngoài việc cần có các Interface để giao tiếp với từng công cụ mã nguồn mở khác nhau thì cũng cần đóng gói các công cụ mở rộng vào thành một Addons hoàn chỉnh. Để giảm thiểu các cài đặt thêm để chạy được các công cụ mở rộng thì em lựa chọn Docker để đóng gói.

Điều này có lợi ích sẽ giúp các công cụ được đóng gói toàn diện, chạy được đa nền tảng tương tự ZAP. Docker cũng là một nền tảng ảo hóa được



Hình 3.16: Khai thác lỗ hổng RFI bằng RFI Exploiter



Hình 3.17: Khai thác LFI/Path Traversal bằng công cụ LFI Exploiter

sử dụng ngày càng phổ biến nên cho phép công cụ dễ dàng thích nghi hơn.

Sự lựa chọn tối ưu để đóng gói các thành phần mà vẫn đem lại hiệu xuất tốt cho các công cụ mở rộng là sử dụng Docker.

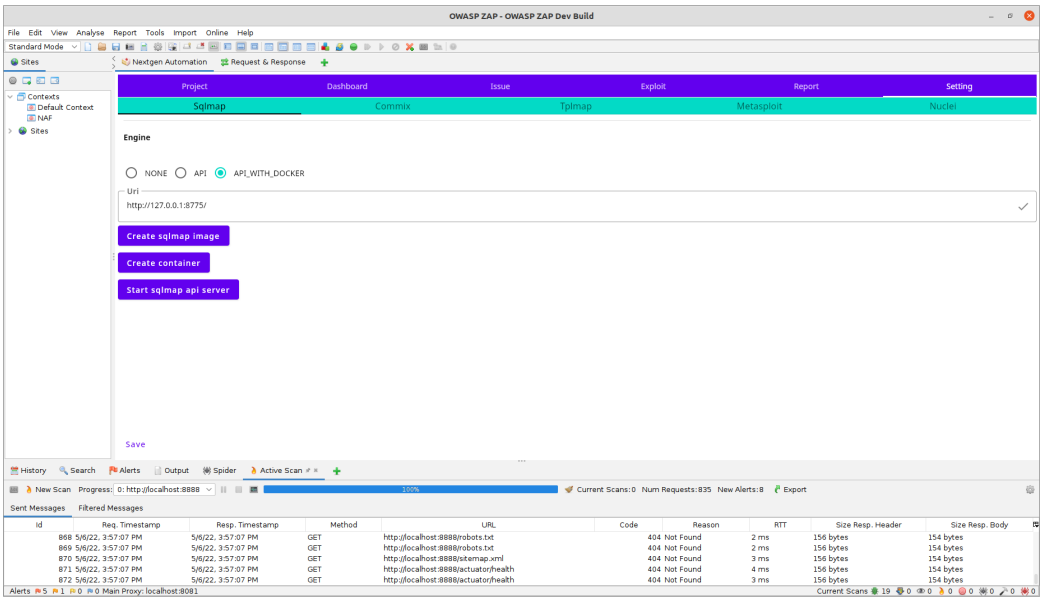
Docker có cung cấp Interface để giao tiếp với Docker Daemon thông qua

REST-API. Java và Kotlin không được hỗ trợ chính thức các thư viện để giao tiếp nhưng cộng đồng có các thư viện để giao tiếp tuy nhiên đã cũ và có rất ít tài liệu hướng dẫn nên cần bọc lại cung cấp một API hoàn chỉnh hơn cho các thành phần sử dụng **??**.

Các API này bao gồm

- Việc xây dựng các Docker Image chứa các công cụ ở ngoài.
- Tạo Container để phù hợp với các tác vụ.
- Giao tiếp với các Container.
- Xóa bỏ các Container không sử dụng.

Để sử dụng các Container hiệu quả, các các Container nên được đặt có các Network Interface cùng với hệ điều hành để tránh việc kết nối bị từ chối do Container sử dụng một mạng ảo riêng.



Hình 3.18: Quản lý Docker của từng thành phần

Chương 4

Thử nghiệm và đánh giá

Để đánh giá được lỗ hổng, em tự xây dựng một bài kiểm tra cho công cụ kiểm thử tự động ở trên. Để tập trung vào lỗ hổng đã xây dựng, em chỉ xây dựng các lỗ hổng cho công cụ mình có thể khai thác. Các lỗ hổng khác sẽ nằm ngoài phạm vi đánh giá của bài thử nghiệm này.

4.1 Môi trường thử nghiệm

4.1.1 Môi trường thời gian chạy

Ngôn ngữ PHP 7.2 và Python.

Cơ sở dữ liệu MySql 5.7.38

Môi trường triển khai Docker.

4.1.2 Các chức năng và lỗ hổng

4.1.2.1 Tìm kiếm người dùng

Mô tả Chức năng cho phép tìm kiếm người dùng khi nhập đúng thông tin gồm username và password.

Loại lỗ hổng SQL Injection

4.1.2.2 Xem thông tin tên miền

Mô tả Chức năng cho sử dụng câu lệnh 'host' của hệ thống để đưa ra thông tin về ip, tên miền của website được nhập vào.

Loại lỗ hổng Command Injection.

4.1.2.3 Biểu diễn trang

Mô tả trang để biểu diễn một đoạn từ file khác được bằng cách include file vào nhưng cho phép người dùng điều chính tên trang.

Loại lỗ hổng Path Traversal, Local File Inclusion, Remote File Inclusion.

4.1.2.4 Xin chào

Mô tả Một trang cho phép nhập tên người dùng để in ra sử dụng một template engine.

Loại lỗ hổng Server Side Template Injection.

4.2 Đánh giá

Kết quả việc tự động kiểm thử và sử dụng tính năng tự xác minh lỗi hổng được phát triển 4.1.

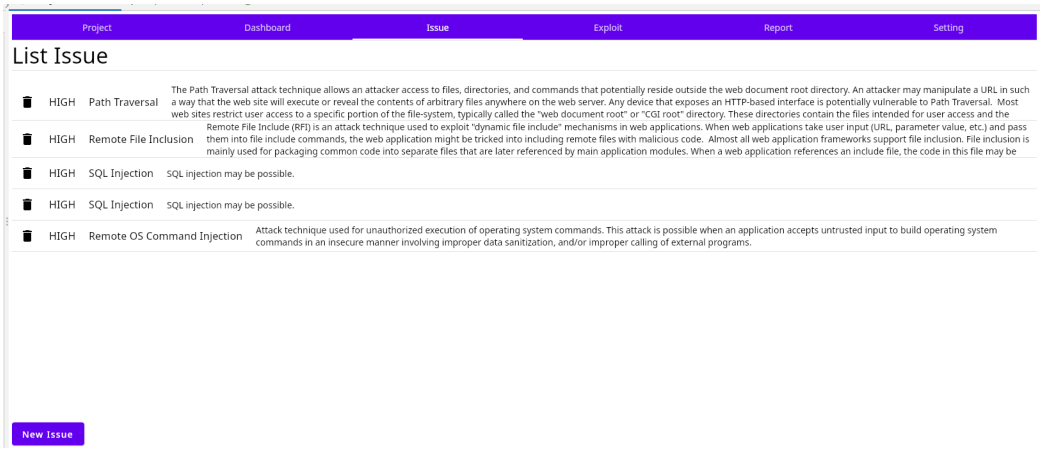
Lỗ hổng	**Số lượng**	**Đã xác minh?**	**Khai thác?**
Server Side Template Injection	0		Có
OS Command Injection	1	Có	Có
SQL Injection	2	Có	Có
Remote File Inclusion	1	Có	Có
Local File Inclusion	1	Có	Có

Bảng 4.1: Kết quả đánh thử nghiệm của NAF trên môi trường tự phát triển

Đồng thời đánh giá với Invicti (Nesparker) 5.8.1 là một công cụ thương mại, em được kết quả như sau.

Lỗ hổng	**Số lượng**	**Đã xác minh?**	**Khai thác?**
Server Side Template Injection	1	Không	Không
OS Command Injection	1	Có	Có
SQL Injection	2	Có	Có
Remote File Inclusion	1	Có	Có
Local File Inclusion	1	Có	Có

Bảng 4.2: Kết quả đánh thử nghiệm của Invicti trên môi trường tự phát triển



Hình 4.1: Kết quả đánh thử nghiệm của NAF trên môi trường tự phát triển

Nhận xét: Mặc dù NAF không quét đủ được các lỗ hổng so với Invicti nhưng đã khai thác tốt hơn với lỗ hổng Server Side Template Injection.

Chương 5

Kết luận

Qua quá trình cài đặt và đánh giá, công cụ Nextgen automation framework đã hoàn thành được mục tiêu tự động hóa quá trình khai thác và hậu khai thác những lỗ hổng đã đề ra. Bằng việc áp dụng công cụ vào quá trình kiểm thử xâm nhập, người kiểm thử có thể tiết kiệm được phần lớn thời gian và giảm khả năng lỗ hổng bị bỏ sót.

Trong tương lai, công cụ này sẽ được mở rộng để hỗ trợ nhiều loại lỗ hổng hơn và thực hiện khai thác trong thời gian ngắn hơn. Khi sản phẩm được hoàn thiện hơn, sản phẩm sẽ được công khai trên ZAP Market.