

ĐẠI HỌC QUỐC GIA HỒ CHÍ MINH  
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN  
KHOA MẠNG MÁY TÍNH VÀ TRUYỀN THÔNG

TRẦN VỸ KHANG  
NGUYỄN ĐẶNG NGUYỄN KHANG

ĐỒ ÁN CHUYÊN NGÀNH  
NGHIÊN CỨU GIẢI PHÁP TỰ ĐỘNG HÓA KIỂM  
THỬ THÂM NHẬP SỬ DỤNG MÔ HÌNH NGÔN  
NGỮ LỚN

RESEARCH ON AUTOMATED PENETRATION TESTING  
SOLUTIONS USING LARGE LANGUAGE MODELS

KỸ SƯ NGÀNH AN TOÀN THÔNG TIN

TP. Hồ Chí Minh, ....., 2025

ĐẠI HỌC QUỐC GIA HỒ CHÍ MINH  
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN  
KHOA MẠNG MÁY TÍNH VÀ TRUYỀN THÔNG

TRẦN VỸ KHANG - 22520628  
NGUYỄN ĐẶNG NGUYỄN KHANG - 22520617

ĐỒ ÁN CHUYÊN NGÀNH  
NGHIÊN CỨU GIẢI PHÁP TỰ ĐỘNG HÓA KIỂM  
THỬ THÂM NHẬP SỬ DỤNG MÔ HÌNH NGÔN  
NGỮ LỚN  
**RESEARCH ON AUTOMATED PENETRATION TESTING  
SOLUTIONS USING LARGE LANGUAGE MODELS**

KỸ SƯ NGÀNH AN TOÀN THÔNG TIN

GIẢNG VIÊN HƯỚNG DẪN:  
ThS. Phan Thế Duy

TP.Hồ Chí Minh - ....., 2025

## LỜI CẢM ƠN

Trong quá trình nghiên cứu và hoàn thành khóa luận, nhóm đã nhận được sự định hướng, hỗ trợ và những ý kiến đóng góp quý báu từ các giáo viên hướng dẫn và giáo viên bộ môn. Nhóm xin gửi lời cảm ơn sâu sắc đến thầy Phan Thế Duy đã tận tình chỉ bảo và hỗ trợ trong suốt quá trình nghiên cứu.

Ngoài ra, nhóm xin gửi lời cảm ơn đến gia đình và bạn bè đã luôn động viên và đóng góp ý kiến quý báu trong suốt quá trình làm khóa luận.

Nhóm cũng chân thành cảm ơn các thầy cô tại trường Đại học Công nghệ Thông tin - ĐHQG TP.HCM, đặc biệt là các thầy cô trong khoa Mạng máy tính và Truyền thông, cũng như các thầy cô thuộc bộ môn An toàn Thông tin đã giúp đỡ nhóm trong quá trình thực hiện nghiên cứu.

**Nguyễn Đặng Nguyên Khang**

**Trần Võ Khang**

## MỤC LỤC

LỜI CẢM ƠN . . . . .	i
MỤC LỤC . . . . .	ii
DANH MỤC CÁC KÝ HIỆU, CÁC CHỮ VIẾT TẮT . . . . .	iv
DANH MỤC CÁC HÌNH VẼ . . . . .	v
DANH MỤC CÁC BẢNG BIỂU . . . . .	vi
MỞ ĐẦU . . . . .	1
<b>CHƯƠNG 1. TỔNG QUAN</b>	<b>2</b>
1.1 Giới thiệu vấn đề . . . . .	2
1.2 Giới thiệu những nghiên cứu liên quan . . . . .	4
1.2.1 Mô hình kiểm thử tự động . . . . .	4
1.2.2 Hệ thống đa tác nhân . . . . .	4
1.3 Tính ứng dụng . . . . .	4
1.4 Những thách thức . . . . .	4
1.5 Mục tiêu, đối tượng, và phạm vi nghiên cứu . . . . .	5
1.5.1 Mục tiêu nghiên cứu . . . . .	5
1.5.2 Đối tượng nghiên cứu . . . . .	5
1.5.3 Phạm vi nghiên cứu . . . . .	5
1.5.4 Cấu trúc đề án chuyên ngành . . . . .	5
<b>CHƯƠNG 2. CƠ SỞ LÝ THUYẾT</b>	<b>7</b>
2.1 Quy trình kiểm thử Web . . . . .	7
2.1.1 Thu thập thông tin . . . . .	7
2.1.2 Quét và liệt kê . . . . .	8
2.1.3 Phân tích và khai thác lỗ hổng . . . . .	9
2.1.4 Báo cáo và đề xuất khắc phục . . . . .	9

2.2	Top 10 OWASP 2021 . . . . .	9
2.2.1	Broken Access Control . . . . .	10
2.2.2	Authentication and Identification Failure . . . . .	10
2.2.3	Injection . . . . .	11
2.2.4	Server-Side Request Forgery (SSRF) . . . . .	12
2.3	Hệ thống đa tác nhân . . . . .	12
2.3.1	Lợi ích của hệ thống đa tác nhân trong kiểm thử web . . .	13
2.3.2	Thách thức của hệ thống đa tác nhân . . . . .	14
<b>CHƯƠNG 3. MÔ HÌNH KIỂM THỬ TỰ ĐỘNG</b>		<b>15</b>
3.1	Giới thiệu về mô hình kiểm thử tự động . . . . .	15
3.2	Mô hình . . . . .	16
3.2.1	Planner . . . . .	16
3.2.2	Generator . . . . .	18
3.2.3	Summarizer . . . . .	19
3.2.4	Hệ thống đa tác nhân . . . . .	19
<b>CHƯƠNG 4. THÍ NGHIỆM VÀ ĐÁNH GIÁ</b>		<b>23</b>
4.1	Thiết lập môi trường thực nghiệm . . . . .	23
4.1.1	Tập dữ liệu . . . . .	25
4.1.2	Quy trình kiểm thử với hệ thống đa tác nhân vào VulnBot	26
4.1.3	Tổng hợp kiểm thử 7 Challenge trên PyGoat và DVWA . .	31
<b>CHƯƠNG 5. KẾT LUẬN</b>		<b>34</b>
5.1	Kết luận . . . . .	34
5.2	Hướng phát triển . . . . .	35
<b>TÀI LIỆU THAM KHẢO</b>		<b>37</b>

## DANH MỤC CÁC KÝ HIỆU, CÁC CHỮ VIẾT TẮT

LLM	Large Language Model (Mô hình ngôn ngữ lớn)
SSRF	Server-Side Request Forgery
XSS	Cross-Site Scripting
SQLi	SQL Injection
JWT	JSON Web Token
OWASP	Open Web Application Security Project
MFA	Multi-Factor Authentication (Xác thực đa yếu tố)
PoC	Proof of Concept
CVSS	Common Vulnerability Scoring System

## DANH MỤC CÁC HÌNH VẼ

Hình 3.1	Mô hình Kiểm thử tự động đã được tinh chỉnh . . . . .	16
Hình 3.2	Penetration Testing Task Graph . . . . .	17
Hình 3.3	Merge Plan Algorithm . . . . .	18
Hình 4.1	PyGoat in Docker . . . . .	24
Hình 4.2	Ví dụ về một mẫu trong dataset . . . . .	26
Hình 4.3	Minh hoạt Input và Plan . . . . .	27
Hình 4.4	Log suy luận và tạo lời gọi tới Agent . . . . .	28
Hình 4.5	Tác nhân điền payload SQLi ' OR '1'='1 vào trường nhập liệu . . . . .	29
Hình 4.6	Kết quả phản hồi sau khi khai thác SQL Injection thành công	30
Hình 4.7	Log terminal: kết luận SQL Injection thực hiện thành công .	31
Hình 4.8	Bar chart so sánh hiệu quả giữa các mô hình kiểm thử . . .	33

## DANH MỤC CÁC BẢNG BIỂU

Bảng 4.1	Bảng kết quả tỉ lệ né tránh của trình tạo đột biến so với 3 mô hình khi không sử dụng trình phát hiện. . . . .	32
----------	---	----



## TÓM TẮT ĐỒ ÁN

### *Tính cấp thiết của đề tài nghiên cứu:*

Trong những năm gần đây, sự gia tăng nhanh chóng về số lượng và mức độ tinh vi của các cuộc tấn công mạng nhắm vào ứng dụng web đã khiến kiểm tra thâm nhập cho ứng dụng web trở thành một nhu cầu cấp thiết trong lĩnh vực an ninh mạng. Các ứng dụng web, với vai trò trung tâm trong kinh doanh, thương mại điện tử và các dịch vụ trực tuyến, thường là mục tiêu của các lỗ hổng như SQL injection, XSS hay cấu hình sai máy chủ, gây ra nguy cơ rò rỉ dữ liệu, mất uy tín và thiệt hại nghiêm trọng. Tuy nhiên, các phương pháp kiểm thử web truyền thống phụ thuộc nhiều vào chuyên gia, tốn thời gian (thường kéo dài vài ngày cho giai đoạn thăm dò) và khó đáp ứng trước các mối đe dọa ngày càng phức tạp. Với sự phát triển của thị trường an ninh mạng và nhu cầu về các giải pháp tự động hóa hiệu quả, các mô hình penetration testing tự động được tích hợp với mô hình ngôn ngữ lớn như PentestGPT, VulnBot được ra đời để giải quyết vấn đề này.

Sau khi nghiên cứu hướng sử dụng mô hình ngôn ngữ lớn để gia tăng hiệu quả kiểm thử, chúng tôi nhận thấy đây là một hướng có nhiều tiềm năng. Nhóm cũng nhận thấy được hầu hết các mô hình tự động kiểm thử hiện tại vẫn còn hạn chế ở việc thực hiện các hành động kiểm thử. Vì vậy trong nghiên cứu này chúng tôi đề xuất một giải pháp, kết hợp mô hình tự động kiểm thử xâm nhập với hệ thống đa tác nhân với mục đích mở rộng các hành động của mô hình và gia tăng hiệu quả kiểm thử tự động.

## CHƯƠNG 1. TỔNG QUAN

Chương này giới thiệu về vấn đề và các nghiên cứu liên quan. Đồng thời, trong chương này chúng tôi cũng trình bày phạm vi và cấu trúc của Khóa luận.

### 1.1. Giới thiệu vấn đề

Trong bối cảnh các cuộc tấn công mạng ngày càng gia tăng về số lượng và mức độ tinh vi, ứng dụng web đã trở thành mục tiêu chính của các tin tặc do vai trò trung tâm trong kinh doanh, thương mại điện tử và các dịch vụ trực tuyến [1]. Các lỗ hổng bảo mật phổ biến như SQL injection, Cross-Site Scripting (XSS) hay cấu hình sai máy chủ không chỉ gây ra nguy cơ rò rỉ dữ liệu nhạy cảm mà còn dẫn đến mất uy tín và thiệt hại tài chính nghiêm trọng cho các tổ chức. Do đó, kiểm tra thâm nhập (penetration testing) đã trở thành một phương pháp quan trọng để phát hiện và khắc phục các lỗ hổng bảo mật, đảm bảo an toàn cho các ứng dụng web.

Tuy nhiên, các phương pháp kiểm thử thâm nhập truyền thống hiện nay vẫn tồn tại nhiều hạn chế. Những phương pháp này thường phụ thuộc nhiều vào chuyên gia an ninh mạng, đòi hỏi thời gian dài (thường kéo dài vài ngày cho giai đoạn thăm dò) và khó đáp ứng được yêu cầu kiểm tra nhanh chóng trước các mối đe dọa ngày càng phức tạp. Để giải quyết vấn đề này, các giải pháp tự động hóa như mô hình kiểm thử thâm nhập tự động tích hợp với mô hình ngôn ngữ lớn (Large Language Models - LLM) như PentestGPT [2] và VulnBot [5] đã được phát triển. Những mô hình này tận dụng khả năng xử lý ngôn ngữ tự nhiên và phân tích dữ liệu của LLM để tự động hóa các bước trong quy trình kiểm thử, từ đó giảm thiểu sự phụ thuộc vào chuyên gia và tiết kiệm thời gian.

Mặc dù các mô hình tự động này mang lại nhiều lợi ích, nghiên cứu của

chúng tôi chỉ ra rằng chúng vẫn còn hạn chế trong việc thực hiện đầy đủ các hành động kiểm thử cần thiết. Cụ thể, các mô hình hiện tại thường thiếu khả năng mô phỏng các kịch bản tấn công phức tạp hoặc khám phá các lỗ hổng đòi hỏi sự phân tích sâu và tương tác đa chiều. Ví dụ, việc phát hiện các lỗ hổng liên quan đến logic ứng dụng hoặc các cấu hình phức tạp thường yêu cầu sự phối hợp của nhiều hành động kiểm thử, điều mà các mô hình tự động hiện nay chưa thực hiện hiệu quả.

Để khắc phục những hạn chế này, nhóm nghiên cứu nhận thấy rằng việc tích hợp mô hình kiểm thử thâm nhập tự động với hệ thống đa tác nhân (multi-agent system) là một hướng đi đầy tiềm năng. Hệ thống đa tác nhân cho phép phân chia các nhiệm vụ kiểm thử thành nhiều tác nhân độc lập, mỗi tác nhân đảm nhận một vai trò cụ thể như thăm dò, phân tích hoặc khai thác lỗ hổng. Sự phối hợp giữa các tác nhân này có thể mô phỏng các kịch bản tấn công phức tạp hơn, từ đó mở rộng khả năng thực hiện các hành động kiểm thử và nâng cao hiệu quả của quá trình kiểm tra tự động.

Trong nghiên cứu này, chúng tôi đề xuất một giải pháp kết hợp mô hình kiểm thử thâm nhập tự động với hệ thống đa tác nhân[4], nhằm cải thiện khả năng thực hiện các hành động kiểm thử và tăng cường độ chính xác trong việc phát hiện lỗ hổng. Giải pháp này không chỉ giúp tự động hóa quy trình kiểm thử mà còn đáp ứng tốt hơn nhu cầu bảo mật trong bối cảnh các mối đe dọa tấn công ngày càng tinh vi. Để đánh giá hiệu quả của giải pháp, chúng tôi sẽ tiến hành thử nghiệm trên các ứng dụng web thực tế và so sánh với các mô hình kiểm thử tự động hiện có, từ đó xác định mức độ cải thiện về hiệu suất và khả năng phát hiện lỗ hổng.

## 1.2. Giới thiệu những nghiên cứu liên quan

### 1.2.1. Mô hình kiểm thử tự động

Dựa trên mô hình kiểm thử tự động VulnBot. Mô hình này với khả năng tự động hóa cao, dùng mô hình ngôn ngữ lớn để thực hiện nhiệm vụ kiểm thử dựa trên quy trình làm việc của một nhóm kiểm thử chuyên nghiệp.

### 1.2.2. Hệ thống đa tác nhân

Hệ thống đa tác nhân hoạt động bằng cách để nhiều tác nhân AI phối hợp cùng nhau nhằm đạt được các mục tiêu chung. Mỗi tác nhân có mức độ tự chủ nhất định, sở hữu khả năng chuyên biệt và góc nhìn cục bộ về toàn bộ hệ thống. Đặc biệt, những hệ thống này được thiết kế để xử lý các bài toán phức tạp có nhiều ràng buộc phụ thuộc lẫn nhau.

## 1.3. Tính ứng dụng

Hệ thống kết hợp VulnBot và mô hình đa tác nhân nâng cao hiệu quả kiểm thử bảo mật thực tế. Các tác nhân thực hiện kiểm thử hoàn toàn tự động và dựa trên quy trình kiểm thử thực tế giúp tăng tốc độ và giảm thiểu chi phí so với kiểm thử thủ công.

## 1.4. Những thách thức

Do giới hạn tài nguyên nên nhóm chỉ có thể fine-tuning và thực hiện kiểm thử chủ yếu với mô hình Llama3.2-3B của Ollama Framework, một mô hình khá nhỏ, khả năng suy luận khá kém và tính ổn định thấp. Do lỗ hổng bảo mật Web khá đa dạng nên nhóm chúng tôi chỉ tập trung vào một số lỗ hổng điển hình trong nhóm Top 10 Web Application Security Risks (Top 10 OWASP 2021) [9]. Ngoài ra, mặc dù các hoạt động tương tác web của mô hình kiểm thử đã được

mở rộng nhưng vẫn còn hạn chế ở khả năng làm giả các trường session, cookie hoặc jwt trong request. Do yêu cầu xử lý giải mã phức tạp và đa dạng.

## **1.5. Mục tiêu, đối tượng, và phạm vi nghiên cứu**

### ***1.5.1. Mục tiêu nghiên cứu***

Ứng dụng mô hình ngôn ngữ lớn và hệ thống đa tác nhân để thực hiện kiểm thử tự động cho website, cung cấp khả năng tương tác đa dạng giữa hệ thống với website được kiểm thử.

### ***1.5.2. Đối tượng nghiên cứu***

*Đối tượng nghiên cứu:*

- Website.
- Lỗ hổng bảo mật Web.
- Mô hình kiểm thử tự động dựa trên mô hình ngôn ngữ lớn.
- Hệ thống đa tác nhân

### ***1.5.3. Phạm vi nghiên cứu***

Tập trung vào kiểm thử Web với các lỗ hổng thuộc Top 10 OWASP 2021: Broken Access Control, Authentication And Identification Failure, Injection and SSRF.

### ***1.5.4. Cấu trúc đề án chuyên ngành***

Chúng tôi xin trình bày nội dung của Đề án theo cấu trúc như sau:

- Chương 1: Giới thiệu tổng quan về đề tài của Khóa luận và những nghiên cứu liên quan.

- Chương 2: Trình bày cơ sở lý thuyết và kiến thức nền tảng liên quan đến đề tài.
- Chương 3: Trình bày mô hình kiểm thử tự động.
- Chương 4: Trình bày thực nghiệm và đánh giá.
- Chương 5: Kết luận và hướng phát triển của đề tài.

## CHƯƠNG 2. CƠ SỞ LÝ THUYẾT

Chương này trình bày cơ sở lý thuyết của nghiên cứu: Bao gồm quy trình kiểm thử Web, Top 10 Web Application Security Risks (Top 10 OWASP 2021) và hệ thống đa tác nhân.

### 2.1. Quy trình kiểm thử Web

Quy trình kiểm thử web là một chuỗi các bước có hệ thống nhằm phát hiện các lỗ hổng bảo mật trong ứng dụng web, đảm bảo rằng hệ thống được kiểm tra toàn diện và các rủi ro bảo mật được xác định một cách chính xác [14]. Trong bối cảnh nghiên cứu này, quy trình kiểm thử web được thiết kế để tích hợp với mô hình kiểm thử thâm nhập tự động và hệ thống đa tác nhân, tập trung vào một số lỗ hổng thuộc Top 10 OWASP 2021 [9] (Broken Access Control, Authentication and Identification Failure, Injection, và SSRF). Quy trình này bao gồm các giai đoạn chính sau:

#### *2.1.1. Thu thập thông tin*

Giai đoạn thu thập thông tin là bước đầu tiên trong quy trình kiểm thử web, nhằm thu thập dữ liệu về mục tiêu để hiểu rõ cấu trúc, công nghệ và các điểm yếu tiềm năng của ứng dụng web. Các hoạt động chính trong giai đoạn này bao gồm:

- **Phân tích tên miền:** Sử dụng các công cụ như whois để thu thập thông tin WHOIS về tên miền của mục tiêu, bao gồm thông tin về chủ sở hữu, thời gian đăng ký, và máy chủ tên miền (DNS). Điều này giúp xác định phạm vi của mục tiêu và các tài sản liên quan.

- **Liệt kê DNS:** Công cụ dnsx được sử dụng để liệt kê các bản ghi DNS, bao gồm các bản ghi A, CNAME, MX, v.v. Điều này giúp khám phá các tên miền phụ (subdomains) và các máy chủ liên quan đến ứng dụng web.
- **Nhận diện công nghệ:** công cụ WhatWeb được sử dụng để phân tích các công nghệ được sử dụng trên website, như hệ quản trị nội dung (CMS), framework, hoặc các thư viện JavaScript. Thông tin này rất quan trọng để xác định các lỗ hổng tiềm năng liên quan đến các công nghệ cụ thể.
- **Kiểm tra cấu hình SSL/TLS:** Sử dụng sslscan để đánh giá cấu hình SSL/TLS của website, phát hiện các giao thức yếu, chứng chỉ không hợp lệ, hoặc các cấu hình sai có thể bị khai thác.

### 2.1.2. Quét và liệt kê

Sau khi thu thập thông tin, giai đoạn quét và liệt kê được thực hiện để xác định các điểm truy cập và các lỗ hổng tiềm năng trên ứng dụng web. Các hoạt động bao gồm:

- **Quét lỗ hổng tự động:** Công cụ nuclei được sử dụng để quét các lỗ hổng bảo mật phổ biến dựa trên các mẫu (templates) được định nghĩa sẵn. Công cụ này tập trung vào việc phát hiện các lỗ hổng như Injection, SSRF, hoặc cấu hình sai, thuộc danh sách Top 10 OWASP.
- **Liệt kê thư mục và tệp:** Công cụ dirsearch được sử dụng để khám phá các thư mục và tệp ẩn trên máy chủ web. Điều này giúp xác định các điểm cuối (endpoints) không được bảo vệ hoặc các tệp cấu hình có thể bị lộ.
- **Thu thập thông tin request HTTP:** Dùng script Python để phân tích phản hồi HTTP.



### ***2.1.3. Phân tích và khai thác lỗ hổng***

Giai đoạn này tập trung vào việc phân tích các lỗ hổng đã phát hiện và thực hiện các kịch bản tấn công để kiểm tra tính nghiêm trọng của chúng. Các tác nhân trong hệ thống đa tác nhân được phân công vai trò cụ thể:

- **Tác nhân phân tích:** Một tác nhân được giao nhiệm vụ gọi tool và phân tích dữ liệu từ các công cụ như `nuclei` và `dirsearch` để xác định các lỗ hổng tiềm năng.
- **Các tác nhân khai thác:** các tác nhân này sử dụng các kỹ thuật tấn công phù hợp với mỗi loại lỗ hổng để thực hiện khai thác lỗ hổng.

### ***2.1.4. Báo cáo và đề xuất khắc phục***

Sau khi hoàn tất quá trình kiểm thử, các kết quả được mô hình ngôn ngữ lớn tổng hợp và phân tích để tạo ra báo cáo chi tiết. Báo cáo này bao gồm:

- Danh sách các lỗ hổng được phát hiện, mức độ nghiêm trọng (dựa trên thang điểm CVSS hoặc tiêu chuẩn OWASP).
- Các bước tái hiện lỗ hổng (Proof of Concept - PoC) để minh họa cách một lỗ hổng có thể bị khai thác.
- Đề xuất khắc phục cụ thể cho từng lỗ hổng, ví dụ: vá lỗi SQL Injection bằng cách sử dụng câu lệnh chuẩn bị (prepared statements) hoặc sửa cấu hình sai trên máy chủ.

## **2.2. Top 10 OWASP 2021**

Danh sách Top 10 OWASP 2021 [9] là một tài liệu tiêu chuẩn do Tổ chức Bảo mật Ứng dụng Web Mở (OWASP) công bố, liệt kê các rủi ro bảo mật ứng dụng web phổ biến và nghiêm trọng nhất. Trong nghiên cứu này, chúng

tôi tập trung vào bốn loại lỗ hổng chính thuộc danh sách này: Broken Access Control, Authentication and Identification Failure, Injection, và Server-Side Request Forgery (SSRF). Dưới đây là mô tả tổng quan và phương pháp kiểm thử cho từng loại lỗ hổng:

### ***2.2.1. Broken Access Control***

Broken Access Control [9] xảy ra khi hệ thống không hạn chế đúng cách quyền truy cập của người dùng vào các tài nguyên hoặc chức năng. Điều này cho phép người dùng trái phép truy cập dữ liệu hoặc thực hiện các hành động vượt quá quyền hạn của họ.

#### **Phương pháp kiểm thử:**

- **Kiểm tra phân quyền:** Thử truy cập các tài nguyên hoặc chức năng không được phép bằng cách thay đổi các tham số URL, ID người dùng, hoặc các trường ẩn trong biểu mẫu.
- **Kiểm tra vượt quyền ngang (Horizontal Privilege Escalation):** Thử truy cập dữ liệu của người dùng khác bằng cách thay đổi các định danh (ví dụ: `user_id`) trong yêu cầu HTTP.
- **Kiểm tra vượt quyền dọc (Vertical Privilege Escalation):** Thử truy cập các subdomain bằng tài khoản người dùng thông thường.
- **Phân tích logic ứng dụng:** Kiểm tra các lỗ hổng trong luồng logic, như bỏ qua bước xác thực hoặc các kiểm tra quyền không đầy đủ.

### ***2.2.2. Authentication and Identification Failure***

Authentication and Identification Failure [9] liên quan đến các vấn đề trong cơ chế xác thực và quản lý phiên, cho phép kẻ tấn công giả mạo danh tính người dùng, đánh cắp phiên, hoặc bỏ qua các kiểm tra xác thực.

#### **Phương pháp kiểm thử:**

- **Kiểm tra độ mạnh mật khẩu:** Thử các cuộc tấn công brute force hoặc đoán mật khẩu với các giá trị phổ biến để kiểm tra chính sách mật khẩu yếu.
- **Kiểm tra quản lý phiên:** Phân tích các trường session, cookie, hoặc JWT để kiểm tra khả năng giả mạo, tái sử dụng phiên, hoặc hết hạn phiên không đúng cách.
- **Kiểm tra cơ chế khôi phục tài khoản:** Thử các kịch bản tấn công vào chức năng quên mật khẩu hoặc đặt lại mật khẩu, như sử dụng thông tin để đoán hoặc bỏ qua bước xác minh.
- **Kiểm tra xác thực đa yếu tố (MFA):** Đánh giá xem MFA có được triển khai đúng cách hay có thể bị bypass không.

### ***2.2.3. Injection***

Injection [9] xảy ra khi dữ liệu không đáng tin cậy được gửi đến hệ thống và được thực thi hoặc xử lý mà không qua kiểm tra hoặc làm sạch. Các loại injection phổ biến bao gồm SQL Injection, Command Injection, và Cross-Site Scripting (XSS).

#### **Phương pháp kiểm thử:**

- **Kiểm tra SQL Injection:** Thử chèn các ký tự đặc biệt (như ', -, hoặc ;) vào các trường đầu vào để kiểm tra khả năng thực thi câu lệnh SQL trái phép.
- **Kiểm tra XSS:** Thử chèn các đoạn mã JavaScript vào các trường đầu vào (như biểu mẫu, tham số URL) để kiểm tra khả năng thực thi mã phía client.
- **Phân tích phản hồi:** Quan sát các phản hồi từ máy chủ để phát hiện các thông báo lỗi hoặc hành vi bất thường, từ đó xác định khả năng khai thác.

### 2.2.4. *Server-Side Request Forgery (SSRF)*

SSRF [9] xảy ra khi kẻ tấn công có thể khiến máy chủ web gửi yêu cầu đến các tài nguyên nội bộ hoặc bên ngoài không mong muốn, dẫn đến việc truy cập trái phép hoặc rò rỉ dữ liệu.

#### **Phương pháp kiểm thử:**

- **Kiểm tra các tham số URL:** Thử thay đổi các tham số URL hoặc đầu vào của người dùng để khiến máy chủ gửi yêu cầu đến các địa chỉ nội bộ hoặc bên ngoài không được phép để kiểm tra khả năng truy cập tài nguyên nội bộ.

## 2.3. Hệ thống đa tác nhân

Hệ thống đa tác nhân (multi-agent system) [4] là một mô hình trong đó nhiều tác nhân AI độc lập phối hợp với nhau để đạt được các mục tiêu chung. Mỗi tác nhân có mức độ tự chủ nhất định, sở hữu khả năng chuyên biệt và góc nhìn cục bộ về toàn bộ hệ thống. Trong bối cảnh kiểm thử thâm nhập web, hệ thống đa tác nhân được thiết kế để phân chia các nhiệm vụ kiểm thử phức tạp thành các hoạt động riêng lẻ, từ đó tăng cường hiệu quả và khả năng mô phỏng các kịch bản tấn công tinh vi. Các đặc điểm chính của hệ thống đa tác nhân bao gồm:

- **Tính tự chủ:** Mỗi tác nhân trong hệ thống có khả năng hoạt động độc lập, đưa ra quyết định dựa trên dữ liệu đầu vào và vai trò được giao. Ví dụ, một tác nhân có thể chuyên về thu thập thông tin, một tác nhân khác tập trung vào khai thác lỗ hổng Broken Access Control, trong khi một tác nhân khác sẽ khai thác một loại lỗ hổng khác.
- **Phối hợp và giao tiếp:** Các tác nhân giao tiếp với nhau thông qua các giao thức thời gian thực, chia sẻ thông tin về kết quả kiểm thử, trạng thái hệ thống, và các phát hiện quan trọng. Điều này cho phép hệ thống mô

phỏng các kịch bản tấn công phức tạp, chẳng hạn như kết hợp các lỗ hổng để thực hiện truy cập trái phép.

- **Chuyên biệt hóa vai trò:** Mỗi tác nhân được gán một vai trò cụ thể, chẳng hạn như:
  - **Tác nhân thu thập thông tin và quét lỗ hổng:** Sử dụng các công cụ như `whois_tool`, `dnsx_tool`, hoặc `whatweb_tool` để thu thập dữ liệu về mục tiêu, bao gồm thông tin tên miền, công nghệ sử dụng, và cấu hình máy chủ. Sử dụng `nuclei_tool` hoặc `dirsearch_tool` để xác định các lỗ hổng tiềm năng, như Injection hoặc SSRF.
  - **Tác nhân tương tác và khai thác lỗ hổng:** Thực hiện các kịch bản tương tác và thử nghiệm tấn công nhiều bước để kiểm tra tính nghiêm trọng của các lỗ hổng, ví dụ: thử nghiệm các payload SQL Injection hoặc giả mạo yêu cầu HTTP để kiểm tra Broken Access Control.
  - **Tác nhân tấn công vét cạn:** tiến hành vét cạn password dựa trên wordlist và dữ liệu sinh thêm từ mô hình ngôn ngữ lớn để tìm kiếm password phù hợp với username.
  - **Tác nhân SSRF:** tìm kiếm lỗ hổng SSRF trên mục tiêu bằng cách sử dụng mô hình ngôn ngữ lớn để suy luận những parameter tiềm năng có thể bị lỗi SSRF và tiếp tục dùng mô hình ngôn ngữ lớn để tạo ra những URL độc hại có thể khai thác lỗ hổng SSRF.

### ***2.3.1. Lợi ích của hệ thống đa tác nhân trong kiểm thử web***

Hệ thống đa tác nhân mang lại nhiều lợi ích trong việc tự động hóa và tối ưu hóa quy trình kiểm thử web:

- **Tăng hiệu quả:** Bằng cách phân chia nhiệm vụ cho các tác nhân chuyên biệt, hệ thống giảm thiểu thời gian kiểm thử so với các phương pháp thủ công.

- **Mô phỏng kịch bản tấn công phức tạp:** Sự phối hợp giữa các tác nhân cho phép mô phỏng các cuộc tấn công đa tầng, chẳng hạn như kết hợp SSRF với các lỗ hổng logic ứng dụng để truy cập tài nguyên nội bộ.
- **Giảm sự phụ thuộc vào chuyên gia:** Tự động hóa các bước kiểm thử giúp giảm thiểu nhu cầu về chuyên gia an ninh mạng, từ đó tiết kiệm chi phí và thời gian.

### *2.3.2. Thách thức của hệ thống đa tác nhân*

Mặc dù có nhiều lợi ích, hệ thống đa tác nhân vẫn đối mặt với một số thách thức:

- **Hạn chế về tài nguyên tính toán:** Hệ thống đa tác nhân cần tương tác rất nhiều bước với mô hình ngôn ngữ lớn từ đó có thể tốn nhiều tài nguyên tính toán.
- **Phối hợp phức tạp:** Việc đảm bảo giao tiếp và phối hợp hiệu quả giữa các tác nhân đòi hỏi thiết kế hệ thống cẩn thận để tránh xung đột hoặc lặp lại công việc.

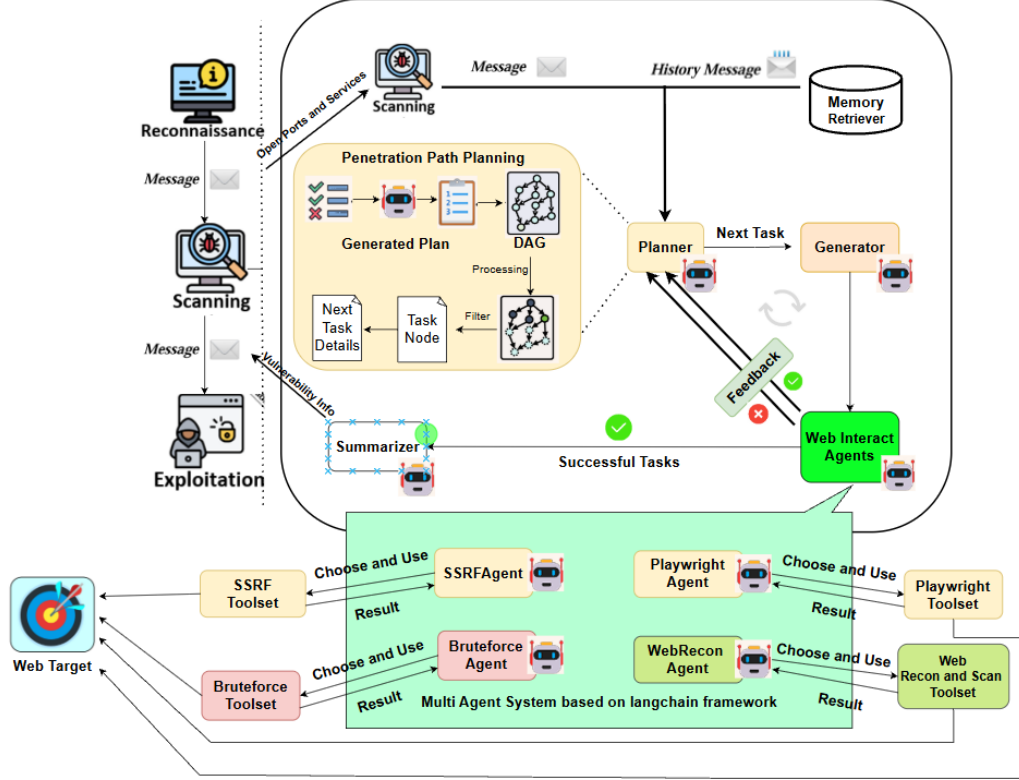
## CHƯƠNG 3. MÔ HÌNH KIỂM THỬ TỰ ĐỘNG

Ở chương này chúng tôi sẽ trình bày về mô hình kiểm thử tự động kết hợp với hệ thống đa tác nhân giúp mở rộng tương tác của mô hình kiểm thử với website được kiểm thử.

### 3.1. Giới thiệu về mô hình kiểm thử tự động

Trong bối cảnh các cuộc tấn công mạng ngày càng phức tạp và tinh vi, việc tự động hóa quy trình kiểm thử xâm nhập đã trở thành một hướng nghiên cứu cấp thiết nhằm giảm tải công sức thủ công và tăng hiệu quả phát hiện lỗ hổng bảo mật [1]. Tuy nhiên, nhiều hệ thống hiện nay, kể cả các mô hình áp dụng trí tuệ nhân tạo và mô hình ngôn ngữ lớn (LLM), việc giới hạn tương tác trong hệ thống kiểm thử tự động vẫn rất lớn. Ví dụ như mô hình mặc định của VulnBot mặc dù có tích hợp hệ thống Multi Agent nhưng chỉ phân chia thành 3 loại tác nhân Reconnaissance, Scanning và Exploitation và các tác nhân chỉ sử dụng những CLI tools mặc định như nmap, whatweb, metasploit,... [5]. Vì vậy cho nên cách mà VulnBot có thể tương tác với môi trường Web là vô cùng hạn chế. Trong đề tài này, chúng tôi đề xuất một mô hình kiểm thử mới dựa trên VulnBot và mở rộng cách VulnBot tương tác với Web.

### 3.2. Mô hình



Hình 3.1: Mô hình Kiểm thử tự động đã được tinh chỉnh

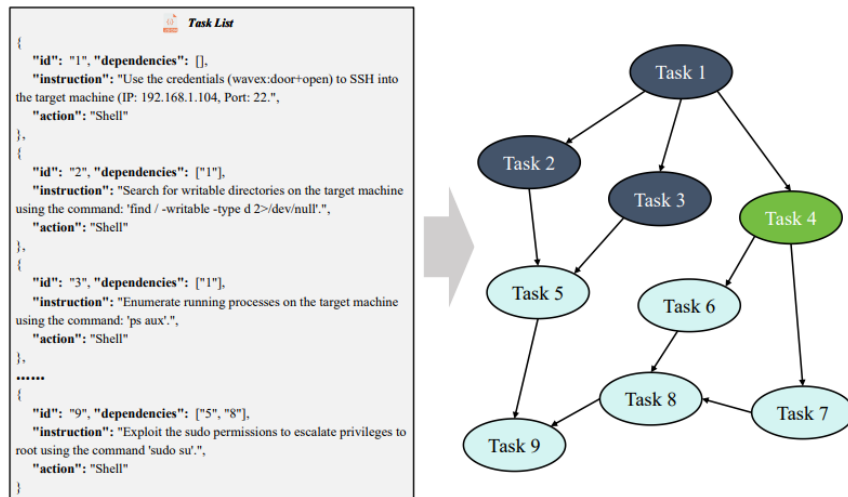
#### 3.2.1. Planner

Planner tạo ra một kế hoạch kiểm tra thâm nhập được cấu trúc theo định dạng JSON, tùy chỉnh dựa trên yêu cầu của người dùng và đặc điểm của hệ thống mục tiêu. Kế hoạch này được phân rã thành các danh sách nhiệm vụ có cấu trúc, trong đó mỗi nhiệm vụ được xác định bởi các định danh duy nhất, phụ thuộc, hướng dẫn chi tiết và loại hành động. Mục tiêu chính là xây dựng một *Penetration Testing Task Graph* (PTG) [5], biểu diễn chuỗi các nhiệm vụ theo thứ tự logic để thực thi. Kế hoạch được cập nhật động dựa trên kết quả thực thi nhiệm vụ, tích hợp phản hồi từ cả nhiệm vụ thành công và thất bại. Quá trình này được điều khiển bởi hai cơ chế chính:

- **Task-driven Mechanism:** Sắp xếp các nhiệm vụ vào một đồ thị có hướng



không vòng (directed acyclic graph), đảm bảo thực thi theo trình tự hợp lý và quản lý các phụ thuộc giữa các nhiệm vụ [5].



**Hình 3.2:** Penetration Testing Task Graph

- **Check and Reflection Mechanism:** Liên tục cải thiện và điều chỉnh kế hoạch thông qua phản hồi lặp lại từ kết quả thực thi nhiệm vụ, cho phép thích ứng với các thay đổi hoặc phát hiện mới trong quá trình kiểm tra [5].

---

**Algorithm 1** Merge Plan Algorithm
 

---

```

1: Input:
2:   newTasks (List of new tasks)
3:   oldTasks (List of old tasks)
4: Output:
5:   mergedTasks (List of merged tasks)
6: completedTasks  $\leftarrow$  GETCOMPLETEDTASKS(oldTasks)
7: mergedTasks  $\leftarrow$  []
   Step 1: Add completed tasks not in the new task list
8: for all task  $\in$  completedTasks do
9:   if EXISTSIN(task, newTasks) = false then
10:    mergedTasks  $\leftarrow$  mergedTasks  $\cup$  {task}
11:   end if
12: end for
   Step 2: Process new tasks and merge with completed tasks
13: for all newTask  $\in$  newTasks do
14:   task  $\leftarrow$  GETTASK(newTask, completedTasks)
15:   if task  $\neq$  null then
16:     UPDATESEQUENCE(task)
17:     UPDATEDDEPENDENCIES(task)
18:   else
19:     task  $\leftarrow$  CREATENEWTASK(newTask)
20:   end if
21:   mergedTasks  $\leftarrow$  mergedTasks  $\cup$  {task}
22: end for
23: return mergedTasks

```

---

*Hình 3.3: Merge Plan Algorithm*

### 3.2.2. Generator

Generator có nhiệm vụ tạo lời gọi tương tác tới bộ 4 Agent dựa trên nội dung Next Task, bao gồm:

- WebRecon Agent
- SSRF Agent
- Playwright Agent

- **Bruteforce Agent**

### ***3.2.3. Summarizer***

Summarizer có nhiệm vụ tóm tắt kết quả trả về từ bộ 4 Agent:

- **WebRecon Agent**
- **SSRF Agent**
- **Playwright Agent**
- **Bruteforce Agent**

Sau đó Summarizer sẽ gửi bản tóm tắt cho **Exploitation Agent** để phân tích.

### ***3.2.4. Hệ thống đa tác nhân***

Hệ thống được tinh chỉnh để gia tăng khả năng tương tác và tập trung vào các lỗ hổng bảo mật Web, bao gồm: SSRF, Injection, Broken Access Control và Authentication and Identification Failure. Các thành phần bao gồm:

- Reconnaissance Agent.
- Scanning Agent.
- Exploitation Agent.
- WebRecon Agent.
- SSRF Agent.
- Playwright Agent.
- Bruteforce Agent.

**Nhiệm vụ của từng Agent:**

- **Reconnaissance Agent:** Trong mô hình của nhóm chúng tôi, Reconnaissance Agent đóng vai trò như một tác nhân chính, phối hợp với WebReconAgent để thực hiện các tác vụ trình sát web, thu thập thông tin chi tiết về mục tiêu nhằm hỗ trợ phân tích và lập kế hoạch cho các hoạt động kiểm tra bảo mật.
- **Scanning Agent:** Chịu trách nhiệm thực hiện các tác vụ quét chủ động, tương tác với mục tiêu để phát hiện các lỗ hổng bảo mật, dịch vụ đang chạy, hoặc các điểm yếu tiềm tàng, cung cấp dữ liệu quan trọng cho việc đánh giá và bảo vệ hệ thống.
- **Exploitation Agent:** Chịu trách nhiệm thực hiện các cuộc tấn công hoặc khai thác các lỗ hổng bảo mật đã được xác định từ các giai đoạn Reconnaissance và Scanning, nhằm kiểm tra tính bảo mật của hệ thống mục tiêu hoặc thực hiện các hành động cụ thể dựa trên kế hoạch đã được lập.
- **WebRecon Agent:** Chọn tool trong WebRecon Toolset để thực hiện reconnaissance hoặc scanning tùy theo mô tả của Reconnaissance Agent hoặc Scanning Agent. Sau đó phân tích và tóm tắt kết quả thực hiện để gửi về Agent đã gọi như Reconnaissance Agent hoặc Scanning Agent.
- **SSRF Agent:** Chọn tool trong bộ SSRF toolset để thực hiện finding và exploit lỗ hổng SSRF. Sau đó phân tích và tóm tắt kết quả thực hiện và gửi báo cáo về Exploitation Agent.
- **Playwright Agent:** Chọn tool trong bộ Playwright toolset để thực hiện các tác vụ được mô tả bởi Exploitation Agent sau đó phân tích kết quả từ các tool trả về và gửi báo cáo cho Exploitation Agent.
- **Bruteforce Agent:** Chọn tool trong bộ Bruteforce toolset để thực hiện brute-force password của người dùng dựa theo mô tả từ Exploitation Agent, sau đó phân tích kết quả để tìm password phù hợp và gửi về cho Exploitation Agent.

### Chi tiết về các bộ tool được trang bị cho các agent:

- **WebRecon toolset:** Một bộ công cụ bao gồm `whois`, `dnsx`, `whatweb`, `ssllscan`, `nuclei`, `dirsearch` và `http_fetch`, được thiết kế để thực hiện reconnaissance và scanning toàn diện trên hệ thống mục tiêu, hỗ trợ phân tích bảo mật và đánh giá cấu hình web.
- **SSRF toolset:** Một bộ tool bao gồm 2 tool là SSRF finding và SSRF exploit. Trong đó SSRFfinding sử dụng LLM để dự đoán những tham số có thể tiêm và trả về cho SSRFAgent. Còn SSRF exploit có nhiệm vụ tạo payload để tiêm vào các parameter đó và sinh request để thực hiện tấn công SSRF.
- **Playwright toolset:** Một bộ công cụ sử dụng Playwright [6] để tự động hóa trình duyệt và kiểm tra bảo mật web, gồm các công cụ `GoToWebsite`, `ClickElement`, `WriteIntoElement`, `FindElement`, `InjectURLPayloads`, `GetNowRequest`, `ModifyRequest`. Trong đó:
  - `GoToWebsite`: Điều hướng trình duyệt đến một URL được cung cấp, tải nội dung trang và lưu trữ HTML để phân tích.
  - `ClickElement`: Nhấn vào một phần tử trên trang web dựa trên mô tả tự nhiên (ví dụ: "nút đăng nhập"), sử dụng LLM để phân tích DOM và tạo XPath phù hợp.
  - `WriteIntoElement`: Nhập văn bản vào một trường biểu mẫu dựa trên mô tả tự nhiên (ví dụ: "ô nhập tên người dùng"), sử dụng LLM để phân tích DOM và xác định XPath.
  - `FindElement`: Tìm và trả về XPath của một phần tử trên trang dựa trên mô tả tự nhiên, sử dụng LLM để phân tích DOM.
  - `InjectURLPayloads`: Chèn các payload SQLi hoặc XSS vào các tham số truy vấn của URL, gửi yêu cầu và trả về trạng thái cùng đoạn nội dung phản hồi.

- **GetNowRequest**: Ghi lại chi tiết yêu cầu HTTP vừa gửi (phương thức, URL, tiêu đề, dữ liệu).
- **ModifyRequest**: Sửa đổi yêu cầu HTTP từ **GetNowRequest** để khai thác các lỗ hổng như Broken Access Control hoặc Authentication Failure, gửi yêu cầu đã chỉnh sửa và trả về phản hồi.
- **Bruteforce toolset**: Chỉ bao gồm một tool duy nhất dùng để vét mật khẩu cho các user được chỉ định. Sử dụng LLM để sinh password tiềm năng và kết hợp với dữ liệu từ bộ wordlist rockyou.

## CHƯƠNG 4. THỰC NGHIỆM VÀ ĐÁNH GIÁ

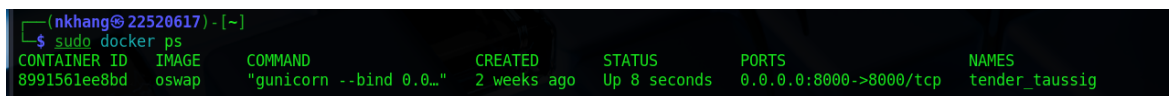
Ở chương này chúng tôi tiến thiết lập môi trường kiểm thử, tiến hành kiểm thử lần lượt qua 3 model là ( **Model gốc LLM:Llama3.2**; **Model tinh chỉnh LLM:Llama3.2-fine-tuned**; **Model tinh chỉnh LLM:Gpt-4o-mini**) đưa ra các tiêu chí đánh giá về mức độ hoàn thành nhiệm vụ được giao và

### 4.1. Thiết lập môi trường thực nghiệm

Nhóm chúng tôi tiến hành kiểm thử trên môi trường máy ảo VMware, trong đó Kali Linux ( với cấu hình RAM là 6GB ) đóng vai trò như một máy server chịu trách nhiệm chứa và khởi tạo môi trường để mô hình có thể thực nghiệm thông qua các nền tảng chuyên về các lỗ hổng TOP 10 OWASP điển hình thông dụng hiện nay như PyGoat [10] và DVWA [16].

Cụ thể đối với từng môi trường thực nghiệm , Pygoat được cài đặt và triển khai trực tiếp trên máy Kali Linux thông qua Docker, đảm bảo môi trường kiểm thử được cô lập, dễ quản lý và có thể khởi động lại nhanh chóng khi cần thiết. Nhóm đã thực hiện các bước sau :

1. Kiểm tra các container đang tồn tại trên Docker bằng *sudo docker ps* (như hình 4.1 )
2. Tiến hành khởi động bằng lệnh *sudo docker start tender\_taussig* ( với *tender\_taussig* là CONTAINER ID hoặc NAMES )
3. Truy cập *http://IP\_kali:8000* để vào được trang chủ PyGoat từ Client để tiến hành kiểm thử tự động .



```

(nkhang@22520617) - [~]
$ sudo docker ps

```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
8991561ee8bd	oswap	"gunicorn --bind 0.0..."	2 weeks ago	Up 8 seconds	0.0.0.0:8000->8000/tcp	tender_taussig

**Hình 4.1:** *PyGoat in Docker*

Bên cạnh PyGoat, nhóm chúng tôi triển khai thêm **môi trường kiểm thử thứ hai là DVWA** (Damn Vulnerable Web Application) – một nền tảng mô phỏng nhiều lỗ hổng bảo mật phổ biến trong danh sách OWASP Top 10. DVWA giúp kiểm thử khả năng phát hiện và khai thác các lỗ hổng web như SQL Injection, Command Injection, XSS, CSRF, File Upload, Brute Force, v.v. Không giống như PyGoat – được triển khai bằng Docker – môi trường DVWA được cài đặt theo phương thức thủ công để giúp nhóm hiểu rõ hơn về quy trình cấu hình hệ thống thực tế. Các bước triển khai cụ thể như sau:

1. Cài đặt Apache, PHP và MariaDB để tạo môi trường web cơ bản cho DVWA hoạt động:

```
sudo apt install apache2 php mariadb-server php-mysql php-gd libapache2-mod-php
```

2. Tải mã nguồn DVWA từ GitHub về thư mục web:

```
cd /var/www/html
```

```
sudo git clone https://github.com/digininja/DVWA.git
```

3. Cấu hình kết nối CSDL trong file config.inc.php, đặt user/password và đảm bảo quyền truy cập đến database dvwa

4. Tạo database và cấp quyền cho user 'dvwa' trong MariaDB:

```
CREATE DATABASE dvwa;
```

```
GRANT ALL PRIVILEGES ON dvwa.* TO 'dvwa'@'localhost' IDENTIFIED BY 'dvwa';
```

```
FLUSH PRIVILEGES;
```

5. Khởi động dịch vụ và truy cập DVWA qua trình duyệt tại:

```
http://localhost/DVWA
```



## 6. Cấu hình thêm để phù hợp kiểm thử :

Bỏ cơ chế đăng nhập thủ công (tự động đăng nhập user admin) bằng cách chỉnh sửa file login.php để hỗ trợ kiểm thử nhanh.

Thiết lập mức bảo mật mặc định là **Low** bằng cách thêm dòng `$_DVWA['default_security_level'] = 'low'` vào file cấu hình.

Bật các module cần thiết như `mod_rewrite`, `allow_url_include` trong `php.ini` để kích hoạt đầy đủ tính năng cho DVWA.

### *4.1.1. Tập dữ liệu*

**Pentesting Eval:** Bộ dataset preemware/pentesting-eval [12] gồm 241 câu hỏi trắc nghiệm, mỗi câu có các lựa chọn, đáp án đúng và phần giải thích chi tiết. Mục tiêu của bộ dữ liệu là giúp đánh giá và cải thiện khả năng Penetration Testing cho mô hình ngôn ngữ (LLM), đặc biệt trong các ngữ cảnh như phân tích khai thác lỗ hổng và chọn công cụ/hành vi phù hợp trong quá trình pentest.

**JSON Planning Format** 160 writeups với format JSON mong muốn được generate tự động bằng ChatGPT, giúp LLM có thể tạo ra dạng json đúng với cấu trúc (hình 4.2)

```

json
[
  {
    "id": "1",
    "dependent_task_ids": [],
    "instruction": "Enumerate user accounts via SMB service using enum4linux",
    "action": "enumerate"
  },
  {
    "id": "2",
    "dependent_task_ids": ["1"],
    "instruction": "Attempt to exploit SMB service to escalate privileges",
    "action": "exploit"
  }
]

```

*Hình 4.2: Ví dụ về một mẫu trong dataset*

**Pentest Writeup Dataset** Ngoài ra, nhóm cũng tổng hợp bộ Pentest Writeup Dataset gồm 497 tài liệu writeup (.pdf và .md) từ các nguồn cộng đồng:

- **The-Viper-One/Pentest-Everything** [15]
- **a-rey/writeups** [13]
- **juliocesarfort/public-pentesting-reports** [3]
- **udaypali/CTF-Writeups** [11]

Tập dữ liệu này phục vụ việc huấn luyện hoặc fine-tune mô hình LLM trong các ngữ cảnh thực tế liên quan đến phân tích tấn công, ghi nhận quá trình khai thác, và học hỏi từ các case study trong cộng đồng pentest.

#### ***4.1.2. Quy trình kiểm thử với hệ thống đa tác nhân vào VulnBot***

Trong hệ thống kiểm thử tự động VulnBot tích hợp mô hình đa tác nhân (Multi-Agent System), mỗi tác nhân đảm nhận một vai trò cụ thể trong quy trình kiểm thử. Một trong những tác nhân chính là **PlaywrightAgent**, được thiết kế để tương tác giống như một người dùng thực tế. Trong phần này, chúng tôi trình bày cụ thể cách hệ thống lập kế hoạch (planning) và thực thi (execution)

một nhiệm vụ kiểm thử lỗ hổng SQL Injection trong ứng dụng DVWA thông qua agent này.

*http://192.168.222.76/DVWA/vulnerabilities/sqli*

Hệ thống bắt đầu bằng cách yêu cầu người dùng mô tả nhiệm vụ kiểm thử.

**Ví dụ về input đầu vào :**

*Exploit the SQL Injection vulnerability in the DVWA lab at http://192.168.222.76/DVWA*

*Focus only on this endpoint.*

Dựa vào mô tả, hệ thống khởi tạo một kế hoạch kiểm thử tự động với nội dung cụ thể như sau:

```
Do you want to continue from a previous session? (y/n) n
Please describe the penetration testing task.
> Exploit the SQL Injection vulnerability in the DVWA lab at http://192.168.222.76/DVWA
Focus only on this endpoint.
Plan Initialized.
2025-07-10 17:07:53.924 | INFO | actions.planner:plan:27 - plan:
[
  {
    "id": "1",
    "dependent_task_ids": [],
    "instruction": "Use WebReconAgent to perform a full scan of the target at http://192.168.222.76/DVWA to identify all open ports and services.",
    "action": "scan"
  },
  {
    "id": "2",
    "dependent_task_ids": ["1"],
    "instruction": "Use PlaywrightAgent to go to http://192.168.222.76/DVWA, find the input field for SQL injection, inject a SQL payload to test for vulnerabilities, then submit the form and observe the response.",
    "action": "interact"
  }
]
```

**Hình 4.3:** Minh hoạt Input và Plan

**Giải thích các thành phần trong kế hoạch:**

- "id": ID của tác vụ, ở đây là "1".
- "dependent\_task\_ids": Không có tác vụ phụ thuộc nào, có thể thực thi độc lập.
- "instruction": Mô tả chi tiết yêu cầu của người dùng, được viết lại dưới dạng hành động có thể thực hiện.
- "action": Kiểu hành động — ở đây là "interact", nghĩa là tác nhân sẽ tương tác trực tiếp với giao diện web.

**Các bước hệ thống thực hiện:** Dưới sự điều phối của PlaywrightAgent, hệ thống sẽ lần lượt thực hiện các hành động sau:

1. Truy cập vào URL mục tiêu.
2. Xác định trường nhập liệu trên trang chứa lỗ hổng.
3. Sinh ra payload SQL Injection đơn giản, ví dụ: ' OR '1'='1'.
4. Tự động điền payload vào input đã xác định.
5. Submit form chứa payload.
6. Quan sát phản hồi và xác định xem khai thác có thành công hay không.

**Minh họa từ log thực tế:** Dưới đây là một phần log ghi nhận lại quá trình thực thi kế hoạch:

```
[DEBUG]next_task: **Task Description**
The purpose of this task is to exploit a SQL Injection vulnerability on the specified target page of DVWA (Damn Vulnerable Web Application). This is a vulnerability testing task aimed at demonstrating the SQLi attack vector. The correct agent to use for this task is the PlaywrightAgent.

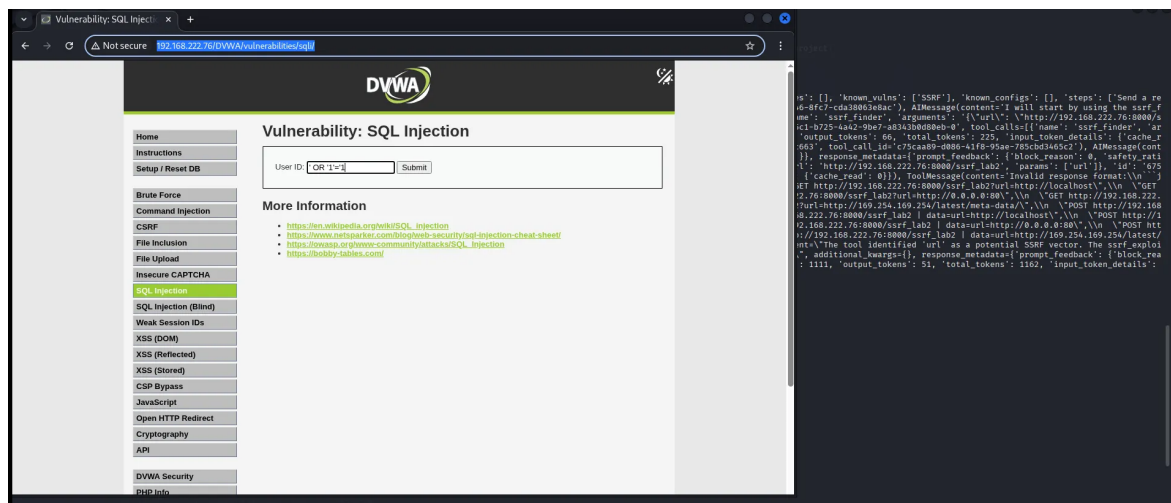
**Agent Execution Block**
<execute>
{
  "agent_name": "PlaywrightAgent",
  "target": "http://192.168.222.76/DVWA/vulnerabilities/sql",
  "plan": {
    "goal": "Identify the input field for SQL injection, inject a normal SQLi payload, and submit the form.",
    "known_services": [],
    "known_vulns": ["SQL Injection"],
    "known_configs": [],
    "steps": [
      "Navigate to the SQL Injection vulnerability page.",
      "Locate the SQL input field on the page.",
      "Generate a normal SQL injection payload (e.g., ' OR '1'='1').",
      "Fill the input field with the SQL injection payload.",
      "Submit the form.",
      "Observe the response for successful exploitation."
    ]
  }
}
</execute>
```

*Hình 4.4: Log suy luận và tạo lời gọi tới Agent*

**Kết quả tương tác với giao diện DVWA:** Sau khi PlaywrightAgent thực hiện toàn bộ chuỗi hành động đã được lên kế hoạch, hệ thống tiến hành khai thác trên trang web mục tiêu như sau:

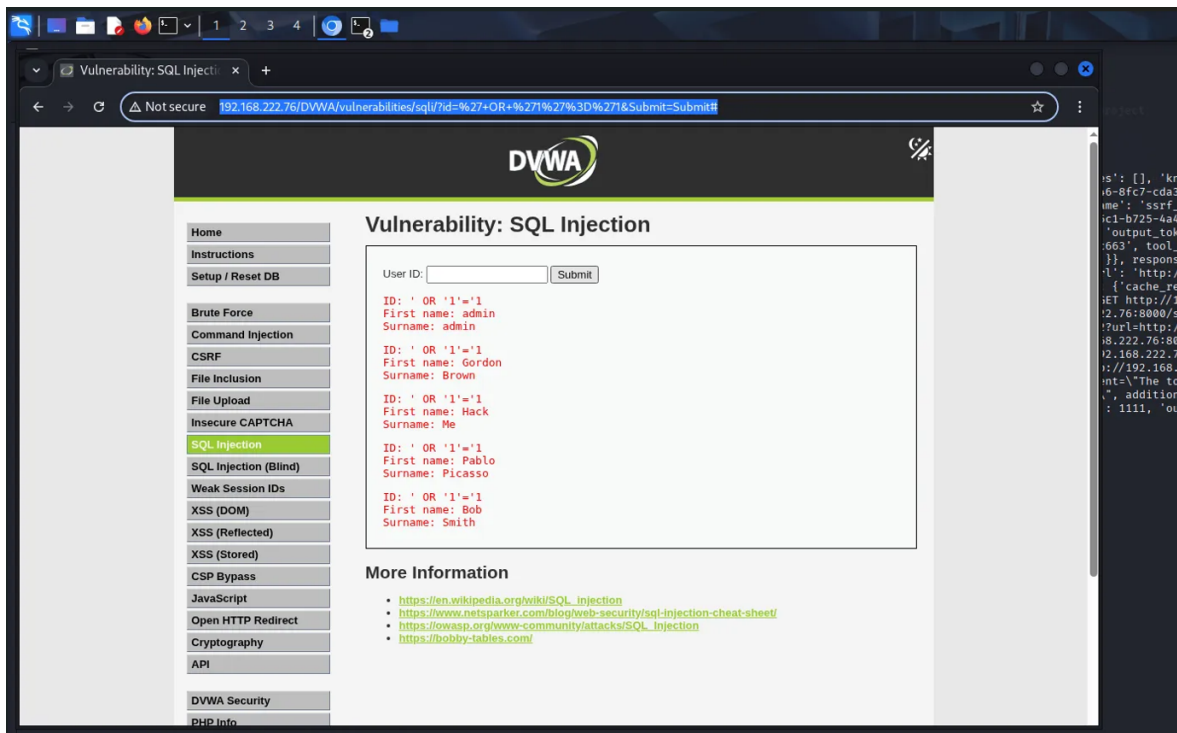
1. Tác nhân truy cập vào đường dẫn /vulnerabilities/sql.

2. Tự động điền payload SQL Injection ' OR '1'='1 vào trường User ID.
3. Submit biểu mẫu và quan sát phản hồi.



**Hình 4.5:** Tác nhân điền payload SQLi ' OR '1'='1 vào trường nhập liệu

Sau khi submit, trang web trả về danh sách người dùng có trong cơ sở dữ liệu cho thấy việc khai thác SQL Injection thành công. Dữ liệu trả về bao gồm các bản ghi chứa các thông tin như hình bên dưới .



**Hình 4.6:** Kết quả phản hồi sau khi khai thác SQL Injection thành công

Phản hồi này xác nhận rằng hệ thống đã thực hiện đầy đủ chuỗi hành động: từ nhận mô tả mục tiêu, lập kế hoạch hành động, thực thi với PlaywrightAgent, đến phân tích kết quả trả về. Tác nhân đã có thể tương tác đúng giao diện người dùng, tiêm payload hợp lệ và xử lý phản hồi để xác định tính khả thi của lỗ hổng.

**Ghi nhận quá trình thực thi của tác nhân trong terminal:** Bên cạnh việc hiển thị kết quả trên giao diện người dùng, hệ thống cũng ghi lại log tương tác chi tiết trong terminal. Các thông tin này phản ánh quá trình thực thi của tác nhân như sau:

- Tác nhân xác định đúng trường đầu vào qua XPath: `//input[@name='id']`.
- Ghi payload SQLi: `' OR '1'='1` vào trường id.
- Tác nhân nhấn nút Submit.

- Phân tích phản hồi: Hệ thống xác nhận có bảng dữ liệu chứa các trường ID, First name, Surname xuất hiện trong nội dung trả về.
- Kết luận: “The SQL injection was successful”.

```

{
  "description": "User ID:"},
  "response_metadata": {
    "prompt_feedback": {
      "block_reason": 0,
      "safety_ratings": []
    },
    "finish_reason": "STOP",
    "safety_ratings": []
  },
  "id": "run--3fe257ea-e7ef-4e4a-9544-f4a1ce133fbc-0",
  "tool_calls": [
    {
      "name": "FindElement",
      "args": {
        "description": "User ID:",
        "id": "dab51929-a345-4905-80a2-90569b9c6598",
        "type": "tool_call"
      }
    }
  ],
  "usage_metadata": {
    "input_tokens": 443,
    "output_tokens": 6,
    "total_tokens": 449,
    "input_token_details": {
      "cache_read": 0
    }
  },
  "tool_message": {
    "content": "//p[contains(text(), 'User ID:')]//input[@name='id']",
    "name": "FindElement",
    "id": "0bac7ba2-dacb-4e82-b35b-953ff8434d5d",
    "tool_call_id": "dab51929-a345-4905-80a2-90569b9c6598"
  },
  "ai_message": {
    "content": "",
    "additional_kwargs": {
      "function_call": {
        "name": "WriteIntoElement",
        "arguments": {
          "text": "\\ OR '1'='1",
          "field": "id"
        }
      }
    },
    "response_metadata": {
      "prompt_feedback": {
        "block_reason": 0,
        "safety_ratings": []
      },
      "finish_reason": "STOP",
      "safety_ratings": []
    },
    "id": "run--10a065d9-75c1-4bf6-bc90-cb4e9a3ff6fc-0",
    "tool_calls": [
      {
        "name": "WriteIntoElement",
        "args": {
          "text": "\\ OR '1'='1",
          "field": "id"
        }
      }
    ],
    "usage_metadata": {
      "input_tokens": 470,
      "output_tokens": 13,
      "total_tokens": 483,
      "input_token_details": {
        "cache_read": 0
      }
    },
    "tool_message": {
      "content": "Wrote '\\ OR '1'='1' into 'id' using XPath: //input[@name='id']",
      "name": "WriteIntoElement",
      "id": "adf4d194-10ec-4f1e-a764-167aa87aa06a",
      "tool_call_id": "000c280c-1d6c-4056-871d-7db7c8dc2a31"
    },
    "ai_message": {
      "content": "",
      "additional_kwargs": {
        "function_call": {
          "name": "ClickElement",
          "arguments": {
            "description": "Submit"
          }
        }
      },
      "response_metadata": {
        "prompt_feedback": {
          "block_reason": 0,
          "safety_ratings": []
        },
        "finish_reason": "STOP",
        "safety_ratings": []
      },
      "id": "run--4c3ac102-82ae-4d61-b3fc-a4ad934da27f-0",
      "tool_calls": [
        {
          "name": "ClickElement",
          "args": {
            "description": "Submit"
          }
        }
      ],
      "usage_metadata": {
        "input_tokens": 511,
        "output_tokens": 4,
        "total_tokens": 515,
        "input_token_details": {
          "cache_read": 0
        }
      },
      "tool_message": {
        "content": "Clicked on element matching: Submit",
        "name": "ClickElement",
        "id": "db6c91bc-b4c2-41ce-9048-0604cf4d064c",
        "tool_call_id": "d4644c94-2541-4c77-b85c-8d7f25b4c6f1"
      },
      "ai_message": {
        "content": "The SQL injection was successful.",
        "additional_kwargs": {},
        "response_metadata": {
          "prompt_feedback": {
            "block_reason": 0,
            "safety_ratings": []
          },
          "finish_reason": "STOP",
          "safety_ratings": []
        },
        "id": "run--bc602f18-e4c9-4737-9601-ee0647a3ec23-0",
        "usage_metadata": {
          "input_tokens": 599,
          "output_tokens": 7,
          "total_tokens": 606,
          "input_token_details": {
            "cache_read": 0
          }
        }
      }
    }
  ]
}

```

**Hình 4.7:** Log terminal: kết luận SQL Injection thực hiện thành công

Như vậy có thể thấy rõ rằng hệ thống kiểm thử đa tác nhân đã thực thi hoàn chỉnh và chính xác toàn bộ quy trình khai thác lỗ hổng SQL Injection được mô tả trong kế hoạch ban đầu. Điều này cho thấy khả năng phối hợp hiệu quả giữa tác nhân điều hướng giao diện (PlaywrightAgent), mô hình LLM phân tích hành động, và công cụ ghi nhận đánh giá đầu ra.

#### 4.1.3. Tổng hợp kiểm thử 7 Challenge trên PyGoat và DVWA

Bên cạnh thử nghiệm SQL Injection đã được trình bày chi tiết, nhóm chúng tôi đã triển khai kiểm thử tự động trên tổng cộng 7 Challenge thực tế, bao gồm 3 thử thách thuộc nền tảng PyGoat và 4 thử thách thuộc nền tảng DVWA. Cấu trúc kiểm thử, phương pháp lập kế hoạch và thực thi được thực hiện thống nhất dựa trên mô hình đa tác nhân và LLM tương tự như phần đã mô tả.

#### Danh sách Challenge đã kiểm thử:

- **PyGoat:**

- Broken Access Control – Lab 1
- Broken Access Control – Lab 2
- Server-Side Request Forgery (SSRF) – Lab 2

- **DVWA:**

- SQL Injection
- XSS – DOM-Based
- XSS – Reflected
- Brute Force

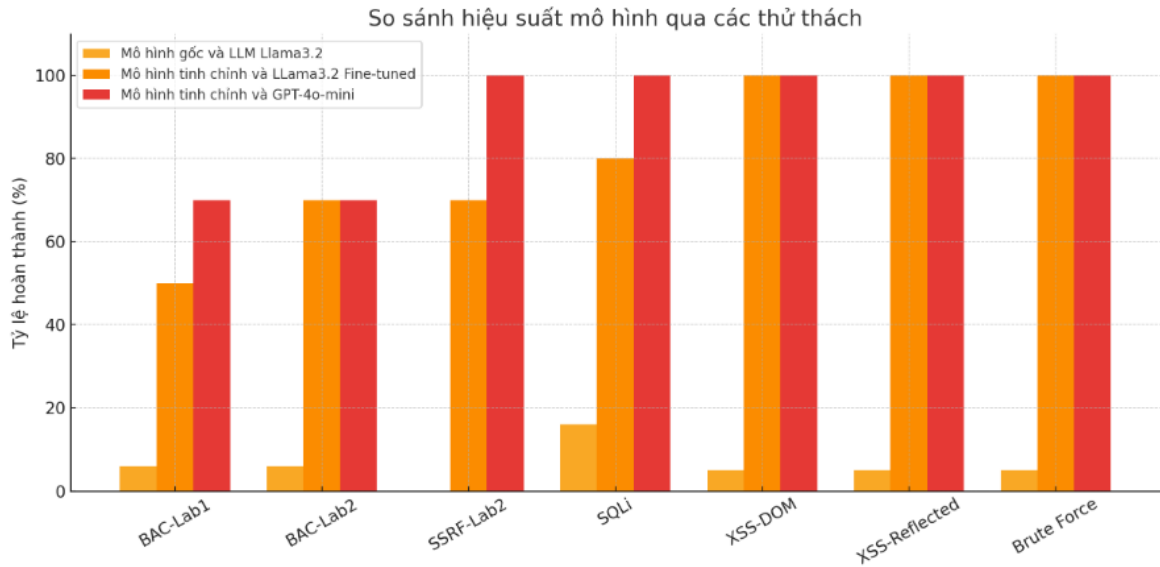
**So sánh hiệu quả giữa các mô hình ngôn ngữ:** Để đánh giá tác động của việc tinh chỉnh mô hình ngôn ngữ lớn (LLM) trong kiểm thử tự động, nhóm đã tiến hành thực nghiệm với 3 mô hình:

1. Llama3.2 (gốc, chưa tinh chỉnh)
2. Llama3.2-fine-tuned (được tinh chỉnh trên tập dữ liệu kiểm thử web)
3. GPT-4o-mini (phiên bản nhẹ, hỗ trợ tương tác tốt)

gray!20	Challenge	Model gốc & Llama3.2-3B	Model tinh chỉnh & Llama3.2-3B-fine-tuned	Model tinh chỉnh & Gpt-4o-mini
<b>3*PYGOAT</b>	BAC - Lab 1	6%	50%	70%
	BAC - Lab 2	6%	70%	70%
	SSRF - Lab 2	0%	70%	100%
<b>4*DVWA</b>	SQL Injection	16%	80%	100%
	XSS - DOM-Based	5%	100%	100%
	XSS - Reflected	5%	100%	100%
	Brute Force	5%	100%	100%

**Bảng 4.1:** Bảng kết quả tỉ lệ né tránh của trình tạo đột biến so với 3 mô hình khi không sử dụng trình phát hiện.





**Hình 4.8:** Bar chart so sánh hiệu quả giữa các mô hình kiểm thử

#### Nhận xét về độ bám task:

- Hiệu quả của mô hình tăng dần theo mức độ tinh chỉnh.

Llama3.2 (model gốc) cho kết quả rất thấp, hiệu suất từ 0%–16% (16% là cao nhất ở SQLi)  $\Rightarrow$  gần như không đủ khả năng giải các bài OWASP, challenge cao nhất là SQLi chỉ lên được đúng plan; gần như tất cả đều dính illegal, không hiểu plan mặc dù đã được chỉ dẫn rất rõ task.

- Llama3.2 fine-tuned cải thiện mạnh mẽ: hầu hết các bài đều đạt  $\geq 70\%$ , đặc biệt là các challenge XSS đạt 100%, cho thấy việc fine-tune đã giúp mô hình hiểu tốt hơn ngữ cảnh bảo mật web.
- Gpt-4o-mini đạt 100% ở đa số các challenge (trừ 2 challenge Broken Access Control đạt 70%), khẳng định ưu thế vượt trội về reasoning, generalization và năng lực hiểu task.

## CHƯƠNG 5. KẾT LUẬN

Ở chương này, chúng tôi tổng kết những kết quả đã đạt được trong quá trình xây dựng mô hình kiểm thử tự động, đồng thời chỉ ra một số hạn chế còn tồn tại và định hướng phát triển hệ thống trong tương lai.

### 5.1. Kết luận

Trong bối cảnh các cuộc tấn công mạng ngày càng tinh vi, việc xây dựng một hệ thống kiểm thử tự động nhằm giảm thiểu thao tác thủ công là cần thiết và có tính ứng dụng cao. Qua quá trình thiết kế và thử nghiệm, nhóm chúng tôi đã phát triển một mô hình kiểm thử tự động dựa trên hệ thống đa tác nhân, với khả năng phối hợp giữa các Agent để phát hiện và khai thác một số loại lỗ hổng web phổ biến.

Một số kết quả đạt được:

- Xây dựng mô hình kiểm thử tự động với kiến trúc gồm các thành phần chính: **Planner**, **Generator**, **Summarizer** và hệ thống đa tác nhân hỗ trợ cải thiện tương tác Web.
- Thiết kế và triển khai các Agent chuyên biệt như WebRecon Agent, SSRF Agent, Playwright Agent và Bruteforce Agent, hỗ trợ khai thác các lỗ hổng như SSRF, XSS, SQLi, Broken Access Control.
- Tích hợp Playwright và LLM để tăng cường khả năng tương tác với giao diện web thông qua mô tả tự nhiên.
- Xây dựng sơ đồ đồ thị nhiệm vụ kiểm thử (Pentest Task Graph) giúp hệ thống có thể lập kế hoạch và điều chỉnh linh hoạt trong quá trình kiểm thử.

Tuy nhiên, hệ thống vẫn còn tồn tại một số hạn chế:

- Một số Agent như **Bruteforce Agent** chưa mang lại hiệu quả thực tế cao trong nhiều tình huống.
- Cơ chế chỉnh sửa yêu cầu HTTP chưa thật sự linh hoạt và hiệu quả.
- Việc nhận diện và xử lý kết quả từ các lỗ hổng dạng mù (ví dụ: Blind SQL Injection) còn hạn chế, ảnh hưởng đến khả năng tự động hóa toàn diện.

## 5.2. Hướng phát triển

Để cải thiện hệ thống kiểm thử tự động trong tương lai, chúng tôi đề xuất một số hướng phát triển như sau:

- **Tinh chỉnh và tối ưu hóa các Agent hiện tại:** Đặc biệt tập trung vào việc nâng cao hiệu quả Agent và đồng thời cải thiện khả năng phân tích kết quả của các lỗ hổng dạng mù (Blind SQLi).
- **Tích hợp xử lý các lỗ hổng liên quan đến Crypto:** Mở rộng phạm vi kiểm thử đến các lỗ hổng liên quan đến mã hóa như padding oracle, JWT token manipulation, hash length extension, v.v.
- **Phát triển thêm các Agent mới:** Nhằm hỗ trợ kiểm thử các dạng lỗ hổng phức tạp hơn như Web Cache Poisoning, SSRF chaining, OAuth mis-configuration, SSTI, và IDOR.
- **Phát triển giao diện người dùng trực quan (UI):** Thiết kế một dashboard giúp theo dõi, kiểm soát, và tùy chỉnh quá trình kiểm thử, hỗ trợ người dùng không chuyên kỹ thuật cũng có thể sử dụng mô hình hiệu quả.
- **Tăng tính tự học của hệ thống:** Áp dụng Reinforcement Learning [7] để hệ thống có thể học và điều chỉnh hành vi kiểm thử dựa trên kết quả từ các phiên chạy trước.

- **Hỗ trợ đa nền tảng và phân tán:** Cho phép triển khai các Agent trên nhiều máy chủ nhằm kiểm thử song song [8] và tiết kiệm thời gian.

Với những định hướng này, chúng tôi tin rằng hệ thống kiểm thử tự động có thể trở thành một công cụ hữu ích, không chỉ hỗ trợ các chuyên gia an toàn thông tin mà còn phù hợp với các hệ thống kiểm thử bảo mật tự động trong môi trường doanh nghiệp.

## TÀI LIỆU THAM KHẢO

### Tiếng Anh:

- [1] Esra Abdullatif Altulaihan, Abrar Alismail, and Mounir Frikha (2023), “A Survey on Web Application Penetration Testing”, *Electronics*, 12 (5), p. 1229, DOI: 10.3390/electronics12051229, URL: <https://www.mdpi.com/2079-9292/12/5/1229>.
- [2] Gelei Deng et al., “PentestGPT: Evaluating and Harnessing Large Language Models for Automated Penetration Testing”, in: *Proceedings of the 33rd USENIX Security Symposium*, Philadelphia, PA, USA, 2024, URL: <https://www.usenix.org/conference/usenixsecurity24/presentation/deng>.
- [3] Julio Cesar Fort, *Public Pentesting Reports*, <https://github.com/juliocesarfot/public-pentesting-reports>, Accessed: July 2025, 2025.
- [4] Shanshan Han et al., *LLM Multi-Agent Systems: Challenges and Open Problems*, 2024, URL: <https://arxiv.org/abs/2402.03578>.
- [5] He Kong et al., *VulnBot: Autonomous Penetration Testing for a Multi-Agent Collaborative Framework*, 2025, arXiv: 2501.13411 [cs.SE], URL: <https://arxiv.org/abs/2501.13411>.
- [6] Microsoft, *Playwright Documentation*, <https://playwright.dev/docs/intro>, Accessed: July 2025, 2025.
- [7] T. Nguyen et al., “PenGym: Pentesting Training Framework for Reinforcement Learning Agents”, in: *Proceedings of the 10th International*

*Conference on Information Systems Security and Privacy (ICISSP)*, vol. 1, 2024, pp. 498–509.

- [8] R.K. Nidhi, M. Pradish, and M. Suneetha (2024), “Cyber Security Analysis of a Power Distribution System Using Vulnerability Assessment and Penetration Testing Tools”, *Power Research - A Journal of CPRI*, pp. 17–25, DOI: 10.33686/pwj.v20i1.1163.
- [9] OWASP Foundation, *OWASP Top 10 – 2021: The Ten Most Critical Web Application Security Risks*, <https://owasp.org/www-project-top-ten/>, Accessed: July 2025, 2021.
- [10] OWASP Foundation, *OWASP PyGoat: Vulnerable Python Web Application for Security Testing*, <https://owasp.org/www-project-pygoat/>, Accessed: July 2025, 2025.
- [11] Uday Pali, *CTF Writeups*, <https://github.com/udaypali/CTF-Writeups>, Accessed: July 2025, 2025.
- [12] Preemware, *pentesting-eval Dataset*, <https://huggingface.co/datasets/preemware/pentesting-eval>, Accessed: July 2025, 2024.
- [13] A Rey, *a-rey/writeups*, Accessed: 2025-07-10, 2020, URL: <https://github.com/a-rey/writeups>.
- [14] Strahinja Stankovic, *Web Application Penetration Testing: Steps, Methods, & Tools*, <https://purplesec.us/learn/web-application-penetration-testing/>, Reviewed by Joshua Selvidge, Published March 31, 2024, Mar. 2024.
- [15] The-Viper-One, *Pentest Everything*, <https://github.com/The-Viper-One/Pentest-Everything>, Accessed: July 2025, 2024.
- [16] Robin Wood and DVWA contributors, *Damn Vulnerable Web Application (DVWA)*, <https://github.com/digininja/DVWA>, Accessed: July 2025, 2025.