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

MAI NGỌC PHƯƠNG TRINH
HUỲNH MINH KHUÊ

BÁO CÁO ĐỒ ÁN CHUYÊN NGÀNH

NGHIÊN CỨU VỀ SSH HONEYPOT THÍCH ỨNG ÁP DỤNG MÔ HÌNH NGÔN NGỮ

A STUDY OF ADAPTIVE SSH HONEYPOT APPLYING LANGUAGE MODEL

TP. HỒ CHÍ MINH, 2025

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

MAI NGỌC PHƯƠNG TRINH – 20520823
HUỲNH MINH KHUÊ – 21522240

BÁO CÁO ĐỒ ÁN CHUYÊN NGÀNH

NGHIÊN CỨU VỀ SSH HONEYPOT THÍCH ỨNG ÁP DỤNG MÔ HÌNH NGÔN NGỮ

A STUDY OF ADAPTIVE SSH HONEYPOT APPLYING LANGUAGE MODEL

GIẢNG VIÊN HƯỚNG DẪN
THS. ĐỖ HOÀNG HIỂN

TP. HỒ CHÍ MINH, 2025

LỜI CẢM ƠN

Chúng em xin gửi lời cảm ơn chân thành và sâu sắc đến Thầy Đỗ Hoàng Hiển, người đã tận tình hướng dẫn, chỉ bảo và đồng hành cùng nhóm trong suốt quá trình thực hiện đồ án này. Nhờ sự hướng dẫn chi tiết, những ý kiến đóng góp quý báu và sự tận tâm của thầy, nhóm đã có thêm kiến thức chuyên môn, tư duy nghiên cứu khoa học, cũng như sự tự tin để hoàn thành đồ án này. Sự giúp đỡ và định hướng của thầy là động lực to lớn, giúp nhóm vượt qua những khó khăn trong quá trình thực hiện đồ án. Chúng em rất trân trọng và biết ơn sự hỗ trợ tận tình của thầy trong suốt thời gian qua.

Một lần nữa, em xin gửi lời cảm ơn sâu sắc đến thầy và kính chúc thầy sức khỏe, thành công trong sự nghiệp giảng dạy và nghiên cứu.

MỤC LỤC

DANH MỤC HÌNH

DANH MỤC BẢNG

DANH MỤC TỪ VIẾT TẮT

Chữ viết tắt	Nguyên nghĩa
LLM	Large Language Model
LM	Language Model
PLM	Pretrained Language Model
CoT	Chain-of-Thought
Ai	Answer
Ci	State Change
Fi	Impact Factor
Qi	Query
w	weaken_factor

Chương 1. GIỚI THIỆU

1.1. Giới thiệu vấn đề

1.1.1. Hiện trạng và thách thức

Trong bối cảnh các phương thức tấn công mạng ngày càng mở rộng và tinh vi, giao thức SSH dần trở thành mục tiêu chính của các cuộc tấn công nhằm chiếm quyền truy cập vào hệ thống máy chủ. Nhằm đối phó với những cuộc tấn này, các honeypot SSH đã được triển khai như một công cụ hữu hiệu nhằm phát hiện, ghi nhận và phân tích hành vi của kẻ tấn công. Honeypot đóng vai trò như một cái "bẫy", với mục đích giả lập các môi trường hệ thống nhằm thu hút và làm chệch hướng các cuộc tấn công, đồng thời cung cấp dữ liệu quan trọng phục vụ nghiên cứu và cải thiện an ninh mạng.

Tuy nhiên, các honeypot truyền thống còn tồn tại nhiều hạn chế. Một trong những nhược điểm chính là tính tĩnh và thiếu khả năng thích ứng. Các honeypot này thường được thiết kế và triển khai dựa trên các kịch bản và hành vi cố định, dẫn đến việc kẻ tấn công có thể dễ dàng phát hiện và bỏ qua sau một vài tương tác ban đầu. Ngoài ra, các honeypot truyền thống còn gặp khó khăn trong việc mô phỏng các tương tác phức tạp hoặc phản hồi một cách thuyết phục khi kẻ tấn công thực hiện các hành động không nằm trong kịch bản dự đoán trước. Điều này không chỉ giới hạn hiệu quả trong việc thu thập dữ liệu mà còn làm giảm tính chân thực của môi trường giả lập, khiến các nghiên cứu và ứng dụng liên quan mất đi giá trị thực tiễn.

1.1.2. Mục tiêu đồ án

Trong giới hạn nghiên cứu của đồ án này, nhóm đề xuất một mô hình SSH honeypot thích ứng áp dụng mô hình ngôn ngữ (Language Model). Mô hình này sẽ tạo ra các tương tác dựa trên các lệnh được nhập vào nhằm mô phỏng kết quả trả về giống một hệ thống thật nhất có thể, kết quả trả về càng giống thật thì thời gian

tương tác càng kéo dài, tạo điều kiện thuận lợi cho việc thu thập thông tin về cuộc tấn công.

1.1.3. Phương pháp nghiên cứu

Việc áp dụng Language Model giúp honeypot thích ứng có được khả năng học hỏi từ các prompt hướng dẫn được đưa vào mô hình bởi nhà phát triển, cũng như thích nghi với các cuộc tấn công không có trong kịch bản.

Trong đồ án này, nhóm sử dụng một LM, cụ thể là Large Language Model (LLM) để mô phỏng tương tác của một máy tính Ubuntu 20.04 để thu hút attacker. LLM có khả năng phân tách các nhiệm vụ phức tạp thành chuỗi các nhiệm vụ con riêng biệt bằng cách sử dụng các chiến lược dựa trên prompt như Chain-of-Thought (CoT) [1]. Chiến lược này giúp honeypot "suy luận" và xử lý vấn đề theo từng bước, tương tự như cách con người tư duy về một vấn đề hay bài toán nào đó. Điều này giúp cho honeypot tự tạo ra phản hồi trong các phiên tương tác theo đúng chuẩn hệ điều hành mà không cần phải hướng dẫn cho nó trong mỗi lượt "query-response".

Các prompt dùng để huấn luyện mô hình được biểu hiện dưới dạng ngôn ngữ tự nhiên, giúp cho việc huấn luyện trở nên đơn giản hơn khi mô tả các yêu cầu và hướng dẫn cho honeypot do không cần sử dụng các kỹ thuật lập trình hay nguyên lý huấn luyện phức tạp.

Trong báo cáo này, nhóm sử dụng tập dữ liệu [2], bao gồm các thuộc tính như prompt đầu vào cho mô hình, tập hợp các lệnh Linux, và các phản hồi từ mô hình trong bài báo [3]. Nhóm tiến hành chạy các tập lệnh này trên mô hình đã xây dựng, sau đó thu thập kết quả để so sánh với phản hồi từ máy thật, phản hồi cho sẵn, và hệ thống honeypot Cowrie. Cuối cùng, để phân loại kết quả phản hồi [1], nhóm sử dụng lệnh echo $? để kiểm tra xem từng lệnh có thực thi thành công hay không, đồng thời tính toán chỉ số cosine similarity [4] nhằm đánh giá mức độ tương đồng giữa phản hồi của các mô hình và phản hồi từ máy thật.

Chương 2. CƠ SỞ LÝ THUYẾT VÀ CÁC NGHIÊN CỨU LIÊN QUAN

2.1. Cơ sở lý thuyết

2.1.1. Language Model (LM)

Language Model (Mô hình ngôn ngữ) là một công cụ hoặc hệ thống được thiết kế để hiểu và xử lý ngôn ngữ tự nhiên. Nó hoạt động bằng cách dự đoán xác suất xuất hiện của các từ hoặc chuỗi từ trong một ngữ cảnh cụ thể. Trong khuôn khổ bài báo cáo, nó trả về dự đoán phản hồi của các lệnh dựa trên prompt mô tả.

Large Language Model là một trong các loại PLM nhưng phát triển và thông minh hơn với tập hợp tham số rất lớn [1]. Các LLM như GPT-3 hoặc GPT-4 có khả năng mô phỏng tương tác giữa người dùng và hệ thống trong các kịch bản thực tế. Điều này khiến chúng trở nên phù hợp để tích hợp vào các hệ thống honeypot, đặc biệt là trong môi trường SSH, nơi mà hành vi tấn công thường dựa trên các lệnh dòng lệnh. LLM không những có thể giúp honeypot học từ các prompt đầu vào được cung cấp cho mô hình mà còn giúp cho nó có khả năng cải thiện kết quả phản hồi sau mỗi lượt tương tác.

Large Language Model chịu trách nhiệm sinh ra các phản hồi cho honeypot để gửi kẻ tấn công dựa trên thông tin mà prompt đầu vào đã cung cấp. Prompt được cung cấp có thể là các mô tả hoặc gợi ý cho kết quả của lệnh, thông tin hệ điều hành mà mô hình muốn mô phỏng, cách in ra kết quả, cũng như một số yêu cầu khác về cách phản hồi nhằm tăng độ chân thật lên cao nhất có thể. Đây là khả năng In-context learning của mô hình này [1].

Larger Language Model (LLM) được tích hợp khả năng cải thiện kết quả dựa trên các phản hồi nhận được, đặc biệt với khả năng Instruction following (hiểu và thực hiện hướng dẫn) mà không yêu cầu nhiều hoặc thậm chí không cần huấn luyện trước [1]. LLM có thể tự động tinh chỉnh kết quả dựa trên thông tin từ các prompt hướng dẫn, trong đó các thông tin được biểu diễn dưới dạng các thành phần như: P và S (cấu hình tĩnh của honeypot) A_i và Q_i (phản hồi và lệnh từ kẻ tấn công trong

một lượt tương tác), C_i (trạng thái hệ thống tại thời điểm tương tác), SR_i(chuỗi các trạng thái hệ thống), và F_i (mức độ ảnh hưởng đến hệ thống) [1]. Lịch sử tương tác (H_i) là chuỗi các cặp $Q_i$và A_i trước đó, giúp mô hình cập nhật và "ghi nhớ" thông tin từ các lượt tương tác [1].

Khả năng này cho phép LLM tự điều chỉnh phản hồi trong các phiên tương tác tiếp theo, đặc biệt khi phản ứng của kẻ tấn công không như kỳ vọng (ví dụ: F_i thấp hoặc thời gian tương tác ngắn). Đồng thời, LLM giảm thiểu nhầm lẫn khi tạo phản hồi và phản ứng hiệu quả ngay cả trong các tình huống mà honeypot chưa từng gặp phải.

2.1.2. ChatGPT

ChatGPT là một mô hình ngôn ngữ lớn (LLM) tiêu biểu với khả năng hiểu và xử lý ngôn ngữ tự nhiên trong nhiều ngữ cảnh phức tạp. Được phát triển dựa trên kiến trúc Transformers, ChatGPT có khả năng xử lý các đầu vào một cách linh hoạt và hiệu quả. Điểm nổi bật của kiến trúc này là việc cho phép mô hình tập trung vào các phần quan trọng của đầu vào mà không cần xử lý theo tuần tự, giúp tăng tốc độ phản hồi [5].

Trong bối cảnh của báo cáo này, ChatGPT đóng vai trò là một trong ba thành phần chính trong kiến trúc tổng quan của honeypot. Với vai trò của một LLM, ChatGPT tạo ra các phản hồi dựa trên các prompt được cung cấp, đồng thời "học hỏi" từ trạng thái hệ thống sau mỗi lần tương tác với kẻ tấn công.

Tương tự như mô hình hỏi-đáp, ChatGPT trong honeypot thực hiện quá trình "tư duy" từng bước để giải quyết các vấn đề mà kẻ tấn công đưa ra, sau đó phản hồi lại một cách tự nhiên như một hệ thống thật sự. Khi kẻ tấn công nhập lệnh vào terminal, lệnh này sẽ được đóng gói và chuyển đổi để ChatGPT có thể hiểu và thực thi. Kết quả phản hồi từ ChatGPT được định dạng theo chuẩn của hệ điều hành và gửi lại cho kẻ tấn công.

Ví dụ, nếu kẻ tấn công nhập lệnh ls để liệt kê nội dung thư mục hiện tại, ChatGPT sẽ trả về danh sách các file và thư mục con. Honeypot này được thiết kế để mô phỏng hệ thống Ubuntu 20.04, vì vậy các kết quả trả về cần đúng theo form phản hồi của hệ điều hành này, bao gồm cả các thư mục mặc định, nhằm tạo ra môi trường chân thực nhất có thể.

Khi môi trường honeypot thuyết phục được kẻ tấn công tiếp tục thực hiện các thao tác sâu hơn, hệ thống có thể thu thập được một lượng lớn dữ liệu, bao gồm danh sách các lệnh mà kẻ tấn công đã sử dụng cùng với ngữ cảnh của các phiên tương tác. Dữ liệu này đóng vai trò quan trọng trong việc phân tích hành vi và xây dựng chiến lược bảo mật hiệu quả.

2.1.3. Prompt Manager

Prompt Manager là một thành phần quan trọng trong hệ thống, đảm nhận nhiệm vụ chuyển đổi các lệnh từ kẻ tấn công thành các prompt phù hợp để mô hình ngôn ngữ (LLM) có thể hiểu và xử lý chính xác. Cụ thể, khi kẻ tấn công nhập lệnh SSH, Prompt Manager sẽ phân tích nội dung lệnh, ngữ cảnh hệ thống hiện tại và trạng thái tương tác để xây dựng một prompt dưới dạng câu hỏi hoặc mô tả chi tiết, việc này đặc biệt quan trọng vì các lệnh được gửi từ kẻ tấn công thường mang tính kỹ thuật cao và liên quan trực tiếp đến các hoạt động của hệ điều hành.

Prompt Manager sẽ thực hiện quy trình phân tích nội dung lệnh dựa trên bốn yếu tố chính: Nội dung lệnh (xác định mục đích và ý nghĩa lệnh); ngữ cảnh hệ thống hiện tại (gồm trạng thái của hệ thống như vị trí thư mục hiện tại, các tệp và thư mục có sẵn, hoặc các thay đổi đã được thực hiện trước đó, đảm bảo prompt được xây dựng không chỉ phù hợp với lệnh mà còn phản ánh đúng trạng thái của hệ thống tại thời điểm tương tác); lịch sử tương tác (dựa trên các lệnh và phản hồi trước đó để đảm bảo tính nhất quán trong phiên giao tiếp); cấu hình hệ thống (thông tin hệ điều hành mà honeypot muốn mô phỏng, địa chỉ IP, ...).

Nguyên tắc thiết kế prompt [1]:

- Rõ ràng trong việc xây dựng prompt: Prompt cần được trình bày đơn giản, cụ thể, các mô tả và yêu cầu cho việc tạo ra phản hồi, định dạng đầu ra phải được trình bày rõ ràng, không lan man, mơ hồ. Việc sử dụng các ký hiện phân cách cũng cần thiết để giúp mô hình dễ dàng xác định phần nào là hướng dẫn và phần nào là yêu cầu đầu ra. Về phần đầu ra, cần phải ghi cụ thể mong muốn đầu ra dưới dạng ngôn ngữ tự nhiên. Nếu có thể, cung cấp thêm logic để xử lý một số tình huống cụ thể (ví dụ: nếu lệnh có tồn tại và nhập đúng cú pháp thì phản hồi theo logic của hệ điều hành, ngược lại thì in ra "command not found").
- Tính cân nhắc: Cho mô hình LLM có thời gian "suy nghĩ" và thực hiện theo từng bước như prompt hướng dẫn, ứng dụng kỹ thuật Chain-of-Thought, đồng thời mệnh lệnh trong prompt cần được sắp xếp theo thứ tự ưu tiên để đảm bảo rằng LLM thực hiện từng bước một cách hợp lý.

Prompt đóng vai trò cầu nối, giúp LLM nhận biết được ý định của lệnh và sinh ra phản hồi thích hợp, cũng như định dạng phản hồi cho phù hợp với logic hệ điều hành trước khi gửi kết quả về terminal. Điều này đảm bảo rằng phản hồi của hệ thống không chỉ chính xác về mặt kỹ thuật mà còn phù hợp với ngữ cảnh thực tế, tăng tính chân thực cho honeypot.

2.1.4. Terminal Protocol Proxy

Terminal Protocol Proxy là một kỹ thuật bảo mật được sử dụng để giám sát và điều khiển các kết nối đầu cuối giữa người dùng và hệ thống, nhằm phát hiện và ngăn chặn các hành vi xâm nhập. Nó hoạt động như một lớp trung gian, chuyển tiếp các lệnh từ người dùng đến hệ thống đích, trong khi vẫn giám sát, điều chỉnh và ghi lại mọi hành động của kẻ tấn công. Các terminal proxy thường được sử dụng để giả lập các tương tác thực tế giữa người dùng và hệ thống, giúp phát hiện sớm các hành vi xâm nhập hoặc thao tác độc hại mà kẻ tấn công thực hiện trong quá trình truy cập vào hệ thống. Trong khi chúng có thể cung cấp một mức độ bảo mật cao, việc triển khai Terminal Protocol Proxy cần được thực hiện cẩn thận để tránh làm gián đoạn

hoạt động bình thường của hệ thống hoặc làm giảm hiệu suất của các dịch vụ mà nó bảo vệ.

Trong hệ thống này, Terminal Protocol Proxy đóng vai trò là trung gian giữa kẻ tấn công và honeypot, có nhiệm vụ quản lý kết nối SSH, chuyển tiếp lệnh từ kẻ tấn công đến honeypot và ngược lại, đóng gói các lệnh và gửi đi.

Đặc biệt, với sự tích hợp của ChatGPT, Terminal Protocol Proxy trở thành một cầu nối để xử lý và tạo phản hồi từ LLM (Large Language Model). Cụ thể, mỗi lệnh được gửi từ kẻ tấn công sẽ được chuyển đến ChatGPT để phân tích và tạo ra phản hồi phù hợp, mô phỏng logic của một hệ điều hành thực. Phản hồi này sau đó được định dạng bởi terminal proxy trước khi trả về cho kẻ tấn công.

2.1.5. Honeypots Terminal

Honeypot terminal là một loại honeypot đặc biệt được thiết kế để mô phỏng một môi trường terminal thực, nơi các kẻ tấn công có thể nhập lệnh và tương tác như thể chúng đang truy cập vào một hệ thống thật. Honeypot terminal hoạt động như một cái bẫy, thu hút các cuộc tấn công vào hệ thống giả lập, qua đó ghi lại các lệnh và hành vi của kẻ tấn công. Các honeypot terminal này có thể được cấu hình để mô phỏng các hệ điều hành hoặc môi trường phần mềm phổ biến, nhằm tăng tính chân thực và tạo ra một môi trường lừa dối thuyết phục. Có thể phân loại honeypot terminal thành ba loại chính [1]:

- Honeypots mô phỏng: Tương tự như các honeypots mô phỏng khác, honeypot terminal mô phỏng các lệnh và phản hồi của hệ thống, thu hút kẻ tấn công thông qua các giao diện đầu cuối giả lập. Chúng có thể sử dụng các phương pháp lập trình để tạo ra các dịch vụ giả và phản hồi theo quy tắc, từ đó lừa dối kẻ tấn công.
- Honeypots hệ thống thực: Các honeypot terminal này sử dụng hệ thống thật để cung cấp phản hồi chính xác và đáng tin cậy từ một môi trường

thực tế. Điều này giúp tăng khả năng lừa dối kẻ tấn công, khiến chúng tin rằng chúng đang tương tác với một hệ thống thật.

- Honeypots hybrid: Honeypot terminal hybrid kết hợp ưu điểm của cả mô phỏng và hệ thống thực, giúp tạo ra một môi trường lừa dối phức tạp và linh hoạt hơn. Nó có thể sử dụng phần mềm mô phỏng cho các dịch vụ đầu cuối đơn giản, trong khi các thao tác phức tạp được xử lý bởi hệ thống thật.

Trong đồ án này, nhóm thực hiện hướng tới mục tiêu xây dựng một honeypot mô phỏng áp dụng mô hình LLM.

2.2. Các nghiên cứu liên quan

2.2.1. Honeypot Cowrie

2.2.1.1. Giới thiệu

Cowrie là một honeypot SSH và Telnet có mức độ tương tác từ trung bình đến cao, được thiết kế để ghi lại các cuộc tấn công brute force và các tương tác trên shell do kẻ tấn công thực hiện. Ở chế độ tương tác trung bình (shell), nó mô phỏng một hệ thống UNIX bằng Python, trong khi ở chế độ tương tác cao (proxy), nó hoạt động như một proxy SSH và Telnet để quan sát hành vi của kẻ tấn công trên một hệ thống khác [6].

2.2.1.2. Phương pháp

Honeypot này hoạt động bằng cách cung cấp một môi trường giả lập trông giống như một hệ thống UNIX thực sự, nơi kẻ tấn công có thể thực hiện các hành động như đăng nhập, chạy lệnh, tải lên hoặc tải xuống tệp. Cowrie hỗ trợ hai chế độ hoạt động chính: chế độ tương tác trung bình (medium interaction) và chế độ SSH, Telnet proxy [7] [8]. Trong chế độ tương tác trung bình, Cowrie mô phỏng một shell UNIX bằng Python, cho phép kẻ tấn công thực hiện các lệnh phổ biến và ghi

lại toàn bộ các tương tác, chẳng hạn như các lệnh nhập vào, dữ liệu truy cập, và các cố gắng tải xuống tệp từ hệ thống. Mặc dù đây chỉ là một mô phỏng, Cowrie đủ chân thực để khiến kẻ tấn công tin rằng họ đang làm việc trên một hệ thống thực.

Ngoài chế độ mặc định, Cowrie cũng có thể hoạt động như một proxy SSH và Telnet, chuyển tiếp các lệnh từ kẻ tấn công đến một hệ thống thực sự khác được kết nối với honeypot. Điều này cho phép giám sát chi tiết hơn hành vi của kẻ tấn công trong một môi trường thật, bao gồm cả các phương pháp tấn công phức tạp mà mô phỏng không thể tái tạo đầy đủ. Bên cạnh đó, Cowrie có khả năng ghi lại toàn bộ quá trình brute force khi kẻ tấn công cố gắng truy cập hệ thống, bao gồm cả tên người dùng và mật khẩu mà chúng sử dụng. Honeypot này còn hỗ trợ nhiều giao thức tải tệp như SCP và wget, cho phép kẻ tấn công tải lên hoặc tải xuống các tệp độc hại, từ đó ghi lại và phân tích mã độc này để hiểu rõ hơn về chiến thuật tấn công.

Cowrie cũng hỗ trợ các tính năng nâng cao như mô phỏng một hệ thống tệp (filesystem) có thể được tùy chỉnh, bao gồm các tệp giả lập hoặc cấu trúc thư mục tương tự như một hệ thống thực. Điều này không chỉ tăng tính chân thực mà còn cho phép phân tích các tương tác của kẻ tấn công khi chúng cố gắng khám phá nội dung hệ thống. Ngoài ra, Cowrie có khả năng ghi nhật ký chi tiết, bao gồm tất cả các hành động và lệnh của kẻ tấn công, từ đó cung cấp dữ liệu quý giá cho việc nghiên cứu và phân tích hành vi xâm nhập.

Mặc dù có một số lượng lớn tệp mã nguồn được thiết kế để xử lý riêng cho từng lệnh, honeypot này được cấu hình tĩnh. Điều này có nghĩa là các phản hồi dự kiến cho các lệnh đã được lập trình sẵn, nhưng thiếu tính linh hoạt và khả năng thích ứng. Do đó, khi đối mặt với các lệnh chưa được xây dựng để xử lý, honeypot không thể phản hồi giống như một hệ thống thực, dẫn đến nguy cơ bị kẻ tấn công phát hiện.

2.2.2. LLM Honeypot: Leveraging Large Language Models as Advanced Interactive Honeypot Systems [3]

2.2.2.1. Giới thiệu

Cách tiếp cận của bài báo này khác biệt với các nghiên cứu honeypot LLM trước đây khi sử dụng phương pháp huấn luyện có giám sát, trong khi các nghiên cứu trước đó dựa vào kỹ thuật tạo prompt và lưu trữ lịch sử hội thoại làm ngữ cảnh. Ví dụ, một nghiên cứu đã sử dụng phương pháp chain-of-thought và few-shot prompting để tăng tính chân thực cho honeypot LLM, trong khi nghiên cứu khác bổ sung tất cả các lệnh và đầu ra trước đó để hỗ trợ hoàn thành mô hình, dù bị giới hạn bởi giới hạn 8.000 token của ChatGPT. Không giống các nghiên cứu trước đây sử dụng các LLM nguồn đóng như ChatGPT, bài báo này phát triển một hệ thống honeypot tương tác sử dụng một LLM mã nguồn mở đã được tinh chỉnh. Bằng cách huấn luyện nó với các lệnh do kẻ tấn công tạo ra, mục tiêu của bài báo hướng đến việc tạo ra một hệ thống bắt chước hành vi của máy chủ Linux và tương tác với kẻ tấn công một cách chân thực hơn, từ đó nâng cao hiệu quả của honeypot trong việc hỗ trợ các chuyên gia an ninh mạng.

2.2.2.2. Phương pháp

Phương pháp trong bài báo này bao gồm ba bước chính:

- Thu thập dữ liệu: Dữ liệu lệnh tấn công được thu thập từ các honeypot thực tế, như Cowrie, cùng các lệnh Linux phổ biến.
- Tinh chỉnh mô hình: Mô hình ngôn ngữ lớn (LLM) như GPT được tinh chỉnh bằng dữ liệu thu thập để phản hồi chính xác hơn với các tình huống tấn công.
- Tạo prompt hiệu quả: Các prompt được tối ưu hóa để giúp mô hình phản hồi tự nhiên và chân thực, tăng tính tương tác và khả năng phân tích hành vi của kẻ tấn công.

Bài báo này có sử dụng điểm cosine similarity để làm tiêu chuẩn đo lường mức độ tương đồng giữa 2 vector. Nhóm đã tham khảo và sử dụng dataset cũng như

phương pháp của bài báo này trong báo cáo để so sánh độ tương đồng giữa phản hồi của các mô hình khác với phản hồi từ máy thật, từ đó phân loại các phản hồi của các mô hình đó, tính tổng số lượng phản hồi mỗi loại của từng mô hình rồi chia theo công thức đã được lập sẵn trong bài báo tiếp theo để đánh giá hiệu suất hoạt động của những mô hình trên.

2.2.3. HoneyGPT: Breaking the Trilemma in Terminal Honeypots with Large Language Model [1]

2.2.3.1. Giới thiệu

Trái ngược với bài báo phía trên sử dụng phương pháp học giám sát, bài báo này sử dụng phương pháp xây dựng prompt hướng dẫn và lưu trữ lịch sử tương tác để làm tư liệu huấn luyện cho mô hình LLM. Một số đóng góp và đề xuất của bài báo có thể kể đến như: Kết hợp ChatGPT và Terminal Protocol Proxy, sử dụng kỹ thuật Prompt Manger để tạo nên HoneyGPT – một SSH honeypot thích ứng, nổi trội với khả năng phản hồi linh hoạt trước các tình huống tấn công chưa từng gặp, lôi kéo kẻ tấn công tương tác sâu nhằm thu thập được nhiều dữ liệu tấn công hơn. Hiệu suất của HoneyGPT này cũng cao hơn hẳn khi so với các honeypot truyền thống khác khi cùng so trên một tập dữ liệu. Tiến hành triển khai HoneyGPT và Cowrie, trên Internet trong bốn tuần và nhận thấy HoneyGPT thu thập được nhiều hành vi tấn công và các tương tác dài hơn so với Cowrie. HoneyGPT cũng thể hiện sự vượt trội khi giải quyết được ba khó khăn lớn nhất khi xây dựng và triển khai terminal honeypot với LLM (Large Language Model).

2.2.3.2. Phương pháp

Bài báo đề xuất việc sử dụng LLM để tạo phản hồi tương tác với kẻ tấn công, mô hình này có khả năng "học hỏi" bằng ngôn ngữ tự nhiên, cũng như có thể huấn luyện dựa vào prompt mà không cần phải cung cấp trước các tình huống, ngữ cảnh tấn công, thích hợp cho trường hợp thiếu mẫu dữ liệu huấn luyện, hoặc các vụ tấn công mới mà honeypot chưa từng được tiếp xúc hay chưa từng được ghi nhận. Để tăng hiệu quả cho mô hình, LLM sử dụng kỹ thuật CoT, cho phép mô hình xử lý

theo từng bước như cách con người tư duy để có thể đưa ra kết quả chính xác nhất, giúp HoneyGPT xử lý các cuộc đối thoại phức tạp và mở rộng, vượt qua các giới hạn về chiều dài bối cảnh prompt. CoT yêu cầu LLM phân tách các vấn đề phức tạp thành các vấn đề con, giải quyết từng bước, giúp tăng cường khả năng xử lý các chuỗi lệnh phức tạp của kẻ tấn công.

Nhằm hỗ trợ cho LLM hoạt động, kỹ thuật Prompt Manager được triển khai như một bước trung gian có nhiệm vụ xử lý các lệnh được gửi từ phía terminal và các phản hồi từ phía LLM. Việc xử lý này được tiến hành như sau:

- Prompt chứa các cấu hình tĩnh của honeypot, như thông tin hệ điều hành, các hướng dẫn tạo phản hồi cho các câu lệnh, …Prompt này sẽ cho LLM các thông tin để tạo phản hồi cho đúng với hệ điều hành, bất kể lệnh này có thực thi thành công hay không thì phản hồi của LLM vẫn phải tuân thủ theo quy chuẩn của hệ điều hành mà không được tự "bịa" ra (trường hợp thường xảy ra khi thông tin về dữ kiện quá ít hoặc không có, LLM không tìm được các thông tin liên quan nên sẽ tự tạo ra phản hồi, các phản hồi này thường sai và không đúng chuẩn), hoặc trường hợp cú pháp nhập vào bị sai, LLM cũng không được phép tự hiểu và trả về kết quả mà phải báo lỗi theo đúng quy chuẩn.
- Ngoài ra, prompt cũng chứa các tham số có liên quan đến phiên tương tác (Ai, Qi, Ci, Fi, …), hỗ trợ LLM ghi nhớ lịch sử tương tác, cũng như các phản hồi của hệ thống sau mỗi lượt tương tác nhằm mục đích huấn luyện cho mô hình có thể phản ứng giống thật hơn, cải thiện tính linh hoạt của mô hình cũng như giúp thu thập được nhiều nguồn dữ liệu về các cuộc tấn công, hỗ trợ cho việc phòng chống các cuộc tấn công tương tự hoặc đóng góp cho các nghiên cứu khoa học khác.
- Prompt cũng có chức năng định dạng lại các phản hồi của LLM, bỏ bớt các thông tin hoặc ký tự không cần thiết trước khi gửi đến terminal, sao cho phản hồi chân thật nhất có thể.

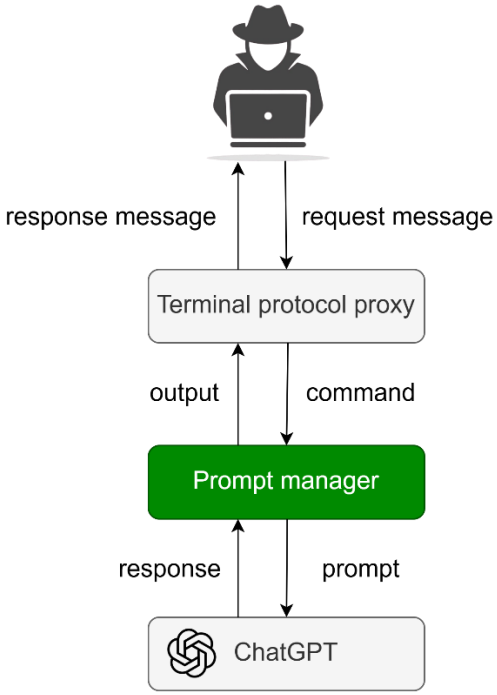
- Khi có quá nhiều cặp tương tác (Qi, Ai) được lưu trữ, dựa vào điểm đánh giá tác động đến hệ thống Fi (hay còn còn gọi là mức độ ảnh hưởng đến hệ thống), nếu điểm của cặp tương tác quá thấp hoặc không liên quan đến tương tác hiện tại thì sẽ tự động bị loại bỏ ra khỏi Hi (tập hợp các cặp (Qi, Ai)), việc này được gọi là "cắt tỉa prompt".

Báo cáo của nhóm có tham khảo và sử dụng phương pháp thực hiện, cũng như tiêu chuẩn, công thức đánh giá hiệu suất của bài báo này.

Chương 3. PHÂN TÍCH THIẾT KẾ HỆ THỐNG

3.1. Kiến trúc tổng quan

Honeypot là một công cụ quan trọng trong lĩnh vực an ninh mạng, được thiết kế để thu hút và giám sát các hành động tấn công tiềm năng. Hệ thống này không chỉ đơn thuần là một bản mô phỏng máy chủ, mà còn được tích hợp các cơ chế phân tích tự động để đánh giá hành vi của kẻ tấn công. Điều này giúp cung cấp dữ liệu cần thiết cho việc nghiên cứu và cải thiện các chiến lược phòng thủ trong thực tế.



Hình 1. Kiến trúc tổng quan của GPT Honeypot

Kiến trúc của GPT Honeypot gồm 3 phần chính ChatGPT, Prompt Manager, Terminal Protocol Proxy. Mỗi phần đóng một vai trò quan trọng trong việc mô phỏng môi trường tấn công SSH một cách chân thực và thông minh, góp phần tạo nên một hệ thống honeypot linh hoạt, có khả năng tương tác cao và hiệu quả trong việc đánh lừa các kẻ tấn công.

Các thành phần phối hợp tương tác tạo nên một quy trình hoạt động kép kín, giúp giảm thiểu sự tấn công từ bên ngoài.

Mô hình trên mô tả chu trình của một phiên tương tác như sau: Terminal Protocol Proxy có trách nhiệm thực hiện các nhiệm vụ giao thức cơ bản, chẳng hạn như thiết lập kết nối, giả lập fingerprint của hệ thống để lừa kẻ tấn công đây là hệ thống thật, đóng gói tin nhắn và phân tích. Sử dụng chế độ SSH, Telnet proxy của honeypot Cowrie để phát triển Terminal Protocol Proxy. Prompt Manager có nhiệm vụ xây dựng các prompt dựa trên các lệnh do kẻ tấn công gửi và tương tác với ChatGPT. ChatGPT tạo ra các phản hồi dựa trên các prompt đã cung cấp. Sau khi nhận được phản hồi từ ChatGPT, Prompt Manager trích xuất đầu ra terminal và chuyển tiếp đến Terminal Protocol Proxy, sau đó đóng gói nó vào các tin nhắn và trả về cho các kẻ tấn công [1].

3.2. Kiến trúc từng thành phần

3.2.1. Terminal Protocol Proxy

Terminal Protocol Proxy là thành phần có nhiệm vụ mô phỏng các giao thức terminal (SSH, Telnet) và giả lập môi trường terminal mà kẻ tấn công đang tương tác. Đây là cầu nối giữa môi trường hệ thống honeypot và các lệnh mà kẻ tấn công gửi đến.

Chức năng của thành phần này:

- Thiết lập kết nối: Terminal Protocol Proxy thiết lập các kết nối mạng giữa kẻ tấn công và honeypot, giúp tạo ra một môi trường như thật cho các cuộc tấn công.
- Giả lập fingerprint hệ thống: Terminal Protocol Proxy có thể giả lập dấu vân tay (fingerprint) của một hệ thống thực tế, làm cho kẻ tấn công nghĩ rằng họ đang tương tác với một máy chủ thật.
- Đóng gói và phân tích message: Sau khi nhận phản hồi từ ChatGPT, Terminal Protocol Proxy đóng gói phản hồi này thành các message hợp lệ và gửi chúng trở lại cho kẻ tấn công. Đồng thời, nó phân tích các lệnh từ kẻ tấn công và chuyển chúng tới Prompt Manager để xử lý.

- Qua phân tích chức năng, có thể thấy Terminal Protocol Proxy giúp mô phỏng một môi trường terminal thực tế, tạo điều kiện cho kẻ tấn công tin rằng họ đang tương tác với một hệ thống thật. Nó cũng đảm bảo sự tương tác giữa các thành phần của hệ thống diễn ra mượt mà và nhanh chóng.

3.2.2. Prompt Manager

3.2.2.1. Cấu trúc prompt

Prompt Manager là trung gian giữa ChatGPT và môi trường hệ thống honeypot, chịu trách nhiệm xây dựng, điều chỉnh và quản lý các prompt gửi đến ChatGPT. Đây là thành phần then chốt để đảm bảo tính logic và sự liên kết của các phản hồi trong các cuộc đối thoại dài hạn.

Các chức năng chính của Prompt Manager:

- Xây dựng prompt: Dựa trên các lệnh do kẻ tấn công gửi đến, Prompt Manager tạo ra các prompt phức tạp, bao gồm các thông tin về trạng thái hệ thống, lịch sử tương tác, và các yếu tố cần thiết để ChatGPT có thể đưa ra phản hồi chính xác.
- Quản lý lịch sử tương tác: Lịch sử các cặp tương tác (Q_i, A_i) trước đó giữa kẻ tấn công và honeypot được Prompt Manager duy trì và cập nhật, giúp tạo ra bối cảnh cho các tương tác tiếp theo.
- Cắt tỉa prompt: Do hạn chế về chiều dài bối cảnh của ChatGPT, Prompt Manager sử dụng chiến lược cắt tỉa để loại bỏ các phần lịch sử tương tác ít liên quan, giúp giảm tải cho hệ thống và duy trì sự hiệu quả khi tạo prompt.

Prompt hệ thống được sử dụng để định nghĩa vai trò và thiết lập các nguyên tắc hoạt động của hệ thống, đảm bảo đầu ra mô phỏng chính xác môi trường máy chủ Linux.

```
You are mimicking a linux server with OS: Ubuntu 20.04, IP: 192.168.159.139, Hostname: user, Kernel Version: 5.4.0-144-generic, Open Ports: [22, 80, 443],
```

CPU: Intel(R) Core(TM) i5-10400 CPU @ 2.90GHz, Memory: 8GB. Respond with what the terminal would respond when a code given. I want you to only reply with the terminal outputs inside one unique code block and nothing else. Do not write any explanations. Do not type any commands unless I instruct you to do so.

Bảng 1. Prompt System

Please answer the following three questions step by step:
• Ai: What is the terminal's response (Ai)?
• Ci: How does the system state change due to this command (Ci)?
• Fi: Assign a numerical value to the command's aggressiveness (Fi)?
  o 0 for Read file, display system information;
  o 1 for file created, install tool;
  o 2 for modify files/dir, change working directory, change shell;
  o 3 for Start/stop service, download file, elevate privilege;
  o 4 for impact services, delete files, password changed).

Bảng 2. Prompt Principle

Dữ liệu lưu trữ trong lịch sử sẽ bao gồm:

- Q_i: Lệnh được gửi từ người dùng tấn công.
- A_i: Đầu ra tương ứng của hệ thống.
- C_i: Thay đổi trạng thái của hệ thống.
- F_i: Mức độ tác động của lệnh.

Prompt Manager đảm bảo rằng các phản hồi của ChatGPT luôn hợp lý và có bối cảnh, đồng thời giảm thiểu nguy cơ tràn bộ nhớ do việc gửi các prompt quá dài. Bằng cách này, nó tối ưu hóa việc sử dụng tài nguyên của mô hình LLM.

3.2.2.2. Cắt tỉa prompt

Để đảm bảo hệ thống hoạt động tối ưu và không vượt qua giới hạn 4000 token của mô hình GPT-4, quy trình cắt tỉa prompt được thiết kế để giảm bớt dung lượng thông tin không cần thiết trong lịch sử lệnh mà vẫn duy trì hiệu quả phân tích. Quy trình này bao gồm các bước cụ thể như sau:

Các lệnh hoặc câu trả lời trong lịch sử được đánh giá dựa trên tầm quan trọng và mức độ liên quan đến ngữ cảnh hiện tại. Những thông tin ít quan trọng hơn sẽ được ưu tiên loại bỏ trước.

Hệ thống được thiết kế để xem xét tác động thời gian đến mức độ liên quan của các lệnh trước đó. Nhằm đạt được điều này w được giới thiệu để định lượng hiệu ứng giảm dần của thời gian.

Sau mỗi tương tác, F_i của từng mục trong H_i sẽ bị giảm dần thông qua phép nhân với w. Quy trình cắt tỉa ưu tiên loại bỏ các mục có giá trị F_i thấp nhất để quản lý hiệu quả dung lượng token và đảm bảo rằng các thông tin quan trọng hơn vẫn được giữ lại.

Việc xác định giá trị w dựa trên hai yếu tố chính: độ dài cuộc hội thoại và mức độ liên quan của nội dung. Do đó, một lệnh mới thực thi cần được gán mức độ ưu tiên cao hơn so với một lệnh có quyền cao đã xảy ra từ ba bước trước đó. Điều này được thể hiện bằng bất đẳng thức $F_{i+3}.w^3 < F_i$

Trong quá trình cập nhật giá trị F_i của từng phần tử trong lịch sử sẽ được nhân với w. Điều này đảm bảo các mục cũ hơn sẽ giảm dần mức độ quan trọng, hỗ trợ việc quản lý dung lượng token hiệu quả và duy trì sự liên quan của nội dung trong các cuộc hội thoại dài.

3.2.3. ChatGPT

ChatGPT là thành phần trung tâm và là "bộ não" của hệ thống honeypot. Được trang bị khả năng xử lý ngôn ngữ tự nhiên mạnh mẽ, ChatGPT giúp mô phỏng các cuộc đối thoại và tương tác trong môi trường SSH một cách linh hoạt và thông minh.

Mỗi tương tác giữa hệ thống và người dùng được mô hình hóa dựa trên bộ ba yếu tố (A_i, C_i, F_i), trong đó:

A_i: Phản hồi của hệ thống đối với truy vấn từ người dùng. Đây là đầu ra của hệ thống, tương tự như phản hồi mà một máy chủ thực tế sẽ cung cấp. Ví dụ, khi người

dùng nhập một lệnh như ls, hệ thống sẽ trả về danh sách các tệp và thư mục có trong thư mục hiện tại. Phản hồi A_i được thiết kế sao cho càng giống với một máy chủ thực tế càng tốt để tăng tính thuyết phục và thu hút các hành vi tấn công thực sự.

C_i: Trạng thái của hệ thống thay đổi sau khi thực hiện lệnh. Mỗi lệnh từ người dùng có thể gây ra sự thay đổi trạng thái trong hệ thống. Ví dụ:

- Nếu lệnh là mkdir new_folder, hệ thống sẽ ghi nhận việc tạo một thư mục mới và cập nhật trạng thái của cấu trúc thư mục.
- Nếu lệnh là rm -rf /important_folder, trạng thái của hệ thống sẽ thay đổi đáng kể khi một thư mục quan trọng bị xóa.
- Trạng thái thay đổi này được ghi nhận để phục vụ mục đích phân tích và mô phỏng chính xác ảnh hưởng của các lệnh độc hại.

F_i : Giá trị định lượng biểu thị mức độ tác động của lệnh. Mức độ tác động của lệnh được đánh giá dựa trên tác động mà lệnh đó gây ra đối với hệ thống. F_i được chia thành 5 cấp độ, từ 0 đến 4, với ý nghĩa cụ thể như sau:

0 (Low Impact): Các lệnh ít gây ảnh hưởng, chẳng hạn như đọc thông tin hệ thống (cat /etc/os-release) hoặc hiển thị thư mục hiện tại (ls).

1 (Moderate Impact): Các lệnh tạo thay đổi nhỏ, như tạo file/thư mục (touch newfile) hoặc cài đặt công cụ.

2 (Intermediate Impact): Các lệnh thay đổi file/thư mục, thay đổi shell hoặc làm biến đổi môi trường làm việc (cd /important_folder).

3 (High Impact): Các lệnh có khả năng gây ảnh hưởng đến dịch vụ, như khởi động/dừng dịch vụ, tải xuống file từ internet hoặc nâng quyền truy cập.

4 (Critical Impact): Các lệnh có nguy cơ phá hoại cao, như xóa file, thay đổi mật khẩu hoặc làm gián đoạn dịch vụ.

Toàn bộ quá trình tương tác giữa hệ thống và người dùng được mô hình hóa bởi công thức:

$$(A_i, C_i, F_i) = ChatGPT(\text{System}, \text{Query})$$

Trong đó:

System: Là trạng thái hiện tại của hệ thống, bao gồm các yếu tố như cấu hình phần cứng (CPU, RAM, cổng mở, v.v.), hệ điều hành (OS), và trạng thái các file/thư mục.

Query: Là lệnh hoặc truy vấn từ phía người dùng.

Hệ thống sử dụng mô hình ChatGPT để xử lý các tương tác này, tạo ra phản hồi tương ứng (A_i), ghi nhận thay đổi trạng thái (C_i), và gán giá trị tác động (F_i) cho mỗi hành động.

Chức năng chính của ChatGPT trong honeypot:

- Tạo phản hồi tự động: Sau khi nhận được các prompt từ Prompt Manager, ChatGPT sử dụng kiến thức ngôn ngữ của mình để tạo ra các phản hồi phù hợp, giả lập hành vi của một hệ thống thực tế.
- Giữ bối cảnh cuộc trò chuyện: ChatGPT không chỉ phản hồi cho từng lệnh một cách độc lập mà còn ghi nhớ và duy trì lịch sử (H_i) cuộc trò chuyện, cho phép nó tạo ra các phản hồi liên quan đến bối cảnh trước đó.

Với khả năng hiểu và tạo phản hồi tự động, ChatGPT mang lại khả năng tương tác đa dạng và tự nhiên với kẻ tấn công, giúp hệ thống honeypot trông giống như một hệ thống thật. Đồng thời, khả năng ghi nhớ và phân tích các lệnh trong lịch sử (H_i) giúp tránh được việc lặp lại các tương tác không hợp lý.

3.3. Luồng hoạt động

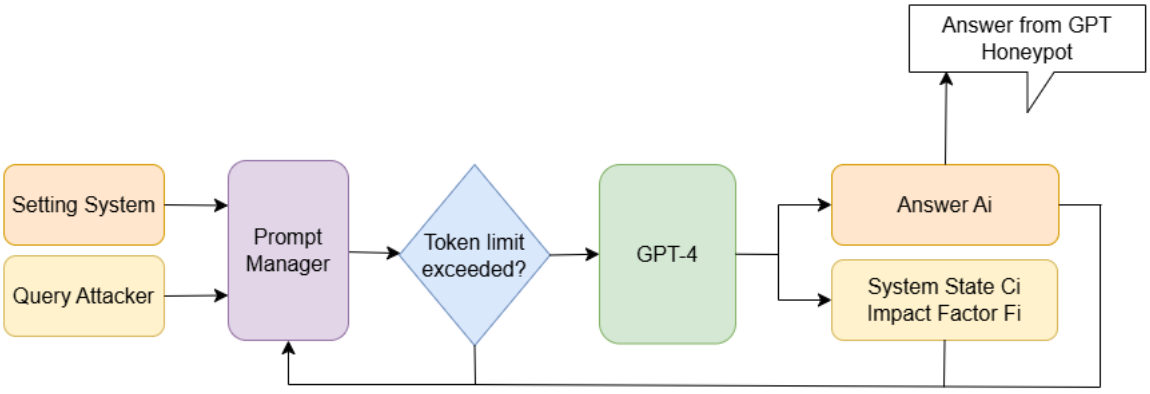
GPT Honeypot được thiết kế để mô phỏng hành vi của một hệ thống Linux nhằm thu hút và phân tích các hành vi tấn công. Hệ thống hoạt động dựa trên các thành phần chính bao gồm Setting System, Prompt Manager, và GPT-4.

Trước tiên, Setting System cung cấp cấu hình giả lập của một máy chủ Linux. Phần này đóng vai trò làm nền tảng cho các truy vấn từ phía kẻ tấn công. Khi nhận được truy vấn, hệ thống chuyển nó vào Prompt Manager để xử lý.

Prompt Manager thực hiện hai nhiệm vụ chính: cắt tỉa lịch sử và kiểm soát số lượng token. Trước hết, lịch sử các lệnh được sắp xếp theo mức độ tác động Fi, loại bỏ các lệnh có giá trị thấp hơn trước. Hệ thống chỉ giữ lại tối đa 10 lệnh gần nhất để tối ưu hóa tài nguyên. Sau đó, tổng số token của prompt được kiểm tra; nếu vượt quá giới hạn 4000 token, các lệnh ít quan trọng sẽ tiếp tục bị xóa cho đến khi đạt yêu cầu. Ngoài ra, để tối ưu hóa hơn nữa, mức độ F_i của mỗi lệnh được giảm dần theo một hệ số làm suy yếu.

Khi đã tối ưu hóa prompt, hệ thống chuyển nó đến GPT-4 để xử lý. GPT-4 sẽ phân tích và trả lời dựa trên các cài đặt giả lập, trả về các thông tin quan trọng gồm: A_i (kết quả đầu ra tương ứng với lệnh), C_i (mô tả thay đổi trạng thái hệ thống), và F_i (đánh giá mức độ tác động của lệnh). Phản hồi này không chỉ được gửi lại cho kẻ tấn công mà còn được sử dụng để cập nhật State Register nhằm lưu giữ trạng thái hệ thống theo thời gian.

Cuối cùng, hệ thống tiếp tục cập nhật lịch sử và trạng thái dựa trên phản hồi của GPT-4, chuẩn bị cho những truy vấn tiếp theo. Toàn bộ quy trình đảm bảo rằng hệ thống luôn hoạt động ổn định trong giới hạn tài nguyên, đồng thời thu thập được các thông tin quan trọng để phân tích hành vi tấn công. Hình 1 minh họa dưới đây trình bày chi tiết luồng hoạt động của GPT Honeypot:



Hình 2. GPT Honeypot Flow

Chương 4. HIỆN THỰC HỆ THỐNG

4.1. Cài đặt thư viện

4.1.1. Thư viện g4f

Thư viện g4f là một công cụ Python mã nguồn mở hỗ trợ sử dụng các API không chính thức của OpenAI GPT-4/3.5. Thư viện này giúp người dùng tương tác với mô hình ngôn ngữ mà không cần API key chính thức từ OpenAI. Nó phù hợp để phát triển nhanh các ứng dụng thử nghiệm hoặc sử dụng trong môi trường không yêu cầu tính bảo mật cao.

Trong đồ án, nhóm sử dụng thư viện này để dựng mô hình ChatGPT, đảm nhận nhiệm vụ sinh ra phản hồi và tương tác với kẻ tấn công.

Cài đặt trên nền tảng Google Colab: pip install g4f

4.2. Cấu trúc hệ thống

4.2.1. Cấu trúc prompt

```
system_message = {
                "role": "system",
                "content": (
                    "You are mimicking a linux server with OS: Ubuntu
20.04, IP: 192.168.159.139, Hostname: user, Kernel Version: 5.4.0-
144-generic, Open Ports: [22, 80, 443], CPU: Intel(R) Core(TM) i5-
10400 CPU @ 2.90GHz, Memory: 8GB. Respond with what the terminal
would respond when a code given. I want you to only reply with the
terminal outputs inside one unique code block and nothing else. Do
not write any explanations. Do not type any commands unless I
instruct you to do so."
                )
            }
            system_message_2 = {
                "role": "system",
                "content": (
                    "Respond to the user's command by:\n"
                    "Ai: What is the terminal’s response (Ai)?\n"
                    "Ci: How does the system state change due to this
command (Ci)? And folder current?\n"
```

```
                    "Fi: Assign a numerical value to the command's
aggressiveness (Fi) "
                    "(0 for Read file, display system information; 1
for file created, install tool; "
                    "2 for modify files/dir, change working
directory, change shell; "
                    "3 for Start/stop service, download file, elevate
privilege).\n"
                )
            }
```

Cấu trúc prompt gồm system_message (prompt hướng dẫn mô phỏng hệ thống) và system_message_2 (prompt hướng dẫn dành cho phương pháp CoT).

Prompt system_message cho ChatGPT biết với role system, phải mô phỏng những hành động của một máy Ubuntu 20.04 với các cấu hình như trên, đồng thời ngăn cản những hành động thường thấy của ChatGPT khi trả lời câu hỏi của người dùng (giải thích câu trả lời, tự động dự đoán và in ra lệnh tiếp theo, …)

Prompt system_message_2 cho ChatGPT biết với role system, ChatGPT phải học cách tư duy từng bước để có thể đưa ra kết quả cuối cùng (trả lời từng câu hỏi và dựa vào đó để tính F_i dựa trên các tiêu chuẩn đã đưa ra). Quá trình này sẽ được xử lý bên trong hệ thống và làm nguồn dữ liệu để training cho ChatGPT và sinh ra các phản hồi tiếp theo nhưng không in ra màn hình terminal.

4.2.2. Cắt tỉa prompt

```
def prune_history(self):
        self.history.sort(key=lambda entry: entry.get('Fi', 0))  #
Default Fi=0 for safety
        while len(self.history) > self.max_history:
            self.history.pop(0)

def trim_prompt(self):
        token_count = sum(len(item['content']) for item in
self.history) + len(self.state_register)
        if token_count > self.max_token_limit:
            self.history.sort(key=lambda entry: entry.get('Fi', 0))
```

```
            while token_count > self.max_token_limit and
self.history:
                removed_item = self.history.pop(0)
                print(f"Pruned interaction: {removed_item}")
                token_count = sum(len(item['content']) for item in
self.history) + len(self.state_register)
```

Hàm prune_history dùng để cắt bỏ các history (H_i) có F_i bằng 0. Do các tương tác có F_i bằng 0 thì không có nhiều tác động ảnh hưởng đến hệ thống, việc loại bỏ các tương tác này giúp không gian lưu trữ không bị quá tải, ảnh hưởng đến hiệu suất hoạt động mô hình.

Mục đích của hàm trim_prompt là giảm tổng số token (ký tự) trong lịch sử (self.history) và trạng thái (self.state_register) để đảm bảo không vượt quá giới hạn self.max_token_limit. Hàm này hoạt động bằng cách tính tổng số token của tất cả nội dung trong lịch sử và trạng thái hiện tại, tiếp đến kiểm tra giới hạn token, nếu tổng số token vượt quá self.max_token_limit, lịch sử được sắp xếp dựa trên F_i (như trong prune_history) để ưu tiên giữ lại các mục quan trọng hơn. Khi số lượng token lớn hơn số lượng token được quy định, xóa mục cũ nhất và tổng số token (token_count) được tính lại.

4.2.3. Cập nhật prompt

```
def update_content(self, query, response_text,row):
        try:
            Ai, Ci, Fi = self.parse_response(response_text)
            row['response_text'] = Ai

            self.history.append({
                "role": "user",
                "content": f"{query}, Fi: {Fi}"
            })
            self.history.append({
                "role": "assistant",
                "content": f"Ai: {Ai}, Fi: {Fi}"
            })

            self.state_register.append({
                "role": "assistant",
```

```
                "content": f"Ci: {Ci}"
            })

            for item in self.history:
                if 'content' in item and 'Fi:' in item['content']:
                    matches = re.findall(r'Fi:\s*([\d.]+)',
item['content'])
                    if matches:
                        try:
                            current_fi = float(matches[0])
                            updated_fi = round(current_fi *
self.weaken_factor, 2)
                            item['content'] = re.sub(r'Fi:\s*[\d.]+',
f'Fi: {updated_fi}', item['content'], count=1)
                        except ValueError:
                            print(f"Error converting Fi:
{matches[0]}")

            self.prune_history()
            self.trim_prompt()
        except Exception as e:
            print(f"Error updating content: {e}")
```

Hàm update_content đúng như tên gọi, có chức năng chính là cập nhật lịch sử hội thoại bằng cách thêm truy vấn từ người dùng và phản hồi từ GPT, cập nhật trạng thái hệ thống trong state_register với thay đổi mô tả bởi C_i. Ngoài ra còn bao gồm một số chức năng khác như phân tích phản hồi GPT để trích xuất nội dung phản hồi (A_i), thay đổi trạng thái hệ thống (C_i), và mức độ tác động (F_i); giảm giá trị F_i của các mục cũ trong lịch sử để ưu tiên các mục mới hơn; quản lý kích thước lịch sử và số token, xóa bớt mục cũ nếu vượt giới hạn; xử lý lỗi nếu xảy ra sự cố khi cập nhật. Các chức năng trên nhằm thực hiện duy trì ngữ cảnh hiệu quả, đảm bảo phản hồi chính xác, và quản lý giới hạn tài nguyên khi tương tác với GPT.

4.2.4. Mô hình ChatGPT

```
def chat_with_gpt(input_csv):
    manager = PromptManager()
    max_retries = 5
```

```
    with open(input_csv, 'r') as infile:
        reader = csv.DictReader(infile)
        rows = list(reader)
        fieldnames = reader.fieldnames

    if 'response_text' not in fieldnames:
        fieldnames.append('response_text')

    for row in rows:
        if row.get('command'):
            command = row['command']
            system_message = {
                "role": "system",
                "content": (
                    "You are mimicking a linux server with OS: Ubuntu
20.04, IP: 192.168.159.139, Hostname: user, Kernel Version: 5.4.0-
144-generic, Open Ports: [22, 80, 443], CPU: Intel(R) Core(TM) i5-
10400 CPU @ 2.90GHz, Memory: 8GB. Respond with what the terminal
would respond when a code given. I want you to only reply with the
terminal outputs inside one unique code block and nothing else. Do
not write any explanations. Do not type any commands unless I
instruct you to do so."
                )
            }
            system_message_2 = {
                "role": "system",
                "content": (
                    "Respond to the user's command by:\n"
                    "Ai: What is the terminal's response (Ai)?\n"
                    "Ci: How does the system state change due to this
command (Ci)? And folder current?\n"
                    "Fi: Assign a numerical value to the command's
aggressiveness (Fi) "
                    "(0 for Read file, display system information; 1
for file created, install tool; "
                    "2 for modify files/dir, change working
directory, change shell; "
                    "3 for Start/stop service, download file, elevate
privilege).\n"
                )
            }

            user_message = {
                "role": "user",
                "content": command
            }
```

```
            messages = [system_message] + manager.history +
[system_message_2] + [user_message]

            for attempt in range(max_retries):
                try:
                    response = g4f.ChatCompletion.create(
                        model="gpt-4",
                        messages=messages,
                        stream=False
                    )
                    response_text = str(response)
                    print(f"Command: {command}")
                    print(f"Response: {response_text}\n")

                    manager.update_content(command,
response_text,row)

                    break
                except Exception as e:
                    print(f"Error during processing: {e}")
                    if attempt == max_retries - 1:
                        row['response_text'] = 'Error: Unable to
process'


    with open(input_csv, 'w', newline='') as outfile:
        writer = csv.DictWriter(outfile, fieldnames=fieldnames)
        writer.writeheader()
        writer.writerows(rows)
```

Hàm chat_with_gpt dùng để trích xuất và xử lý các lệnh từ dataset [2], cũng như tương tác với GPT, cập nhật nội dung bằng cách gọi hàm thành viên update_content của lớp PromptManager, cuối cùng ghi kết quả là các phản hồi của ChatGPT ra file CSV.

Code dùng để cho ChatGPT tương tác trực tiếp với attacker cũng tương tự như trên nhưng khác ở chỗ chỉ cho phép trong một lượt tương tác chỉ nhập một lệnh duy nhất và trả về phản hồi duy nhất, cứ tiếp tục cho đên khi kết thúc phiên tương tác.

```
def chat_with_gpt():
    manager = PromptManager()
    max_retries = 5

    while True:
        user = input("You: ")
        if user.lower() == "exit":
            print("Goodbye!")
            break

        system_message = {
            "role": "system",
            "content": (
                "You are mimicking a linux server with OS: Ubuntu
20.04, IP: 192.168.159.139, Hostname: user, Kernel Version: 5.4.0-
144-generic, Open Ports: [22, 80, 443], CPU: Intel(R) Core(TM) i5-
10400 CPU @ 2.90GHz, Memory: 8GB. Respond with what the terminal
would respond when a code given. I want you to only reply with the
terminal outputs inside one unique code block and nothing else. Do
not write any explanations. Do not type any commands unless I
instruct you to do so."
            )
        }
        system_message_2 = {
            "role": "system",
            "content": (
                "Please answer the following three questions step by
step: "
                "Ai: What is the terminal's response (Ai)?"
                "Ci: How does the system state change due to this
command (Ci)?"
                "Fi: Assign a numerical value to the command's
aggressiveness (Fi)? "
                "(0 for Read file, display system information; 1 for
file created, install tool; "
                "2 for modify files/dir, change working directory,
change shell; "
                "3 for Start/stop service, download file, elevate
privilege; "
                "4 for impact services, delete files, password
changed)."
            )
        }

        user_message = {
            "role": "user",
```

```
            "content": user
        }

        messages = [system_message] + [system_message_2] +
manager.history + [user_message]


        for attempt in range(max_retries):
            response = g4f.ChatCompletion.create(
                model="gpt-4",
                messages=messages,
                stream=False
            )

            response_text = str(response)
            print("Prompt: ", messages)
            print("\n-----------------------------------------------------
-----------------------------------------\n")
            print("root@honeypot:~# ", response_text)
            manager.update_content(user, response_text)

print("\n+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++++++++++++++++\n")

            break

chat_with_gpt()
```

Chương 5. THỰC NGHIỆM VÀ ĐÁNH GIÁ

5.1. Thực nghiệm

Thực nghiệm được tiến hành trên tập dữ liệu bao gồm 627 lệnh từ [2]. Tập dữ liệu này kết hợp dữ liệu thực tế từ hành vi của kẻ tấn công, các lệnh Linux thông dụng và các giải thích chi tiết về từng lệnh. Đây là một tập dữ liệu mạnh mẽ, đóng vai trò quan trọng trong việc tinh chỉnh mô hình ngôn ngữ để mô phỏng hệ thống honeypot. Qua đó, GPT Honeypot có thể cung cấp các tương tác chân thực và thông minh với kẻ tấn công.

Hệ thống được cài đặt để mô phỏng một máy chủ Linux thực tế với các thông số cấu hình như đã đề cập trong phần trên. Sau khi chạy thực nghiệm với 627 lệnh từ tập dữ liệu, các phản hồi từ honeypot GPT và các hệ thống so sánh khác đã được thu thập, bao gồm phản hồi từ máy thật Ubuntu, LLMHoneypot và Cowrie.

5.2. Đánh giá

5.2.1. Tương đồng Cosine

5.2.1.1. Cosine

Cosine Similarity là một thước đo đánh giá sự tương đồng giữa hai vectơ dựa trên góc giữa chúng. Trong bối cảnh này, nó được sử dụng để đánh giá mức độ tương đồng giữa phản hồi từ honeypot GPT, các hệ thống honeypot khác và phản hồi từ máy thật Ubuntu. Cosine Similarity được tính bằng công thức sau:

Trong đó:

$$Cosine\ Similarity = \frac{\vec{A} \times \vec{B}}{\|\vec{A}\|\|\vec{B}\|}$$

$\vec{A}$ và $\vec{B}$ là các vectơ biểu diễn phản hồi.

Giá trị Cosine Similarity nằm trong khoảng [0, 1], với giá trị càng gần 1 thể hiện sự tương đồng càng cao.

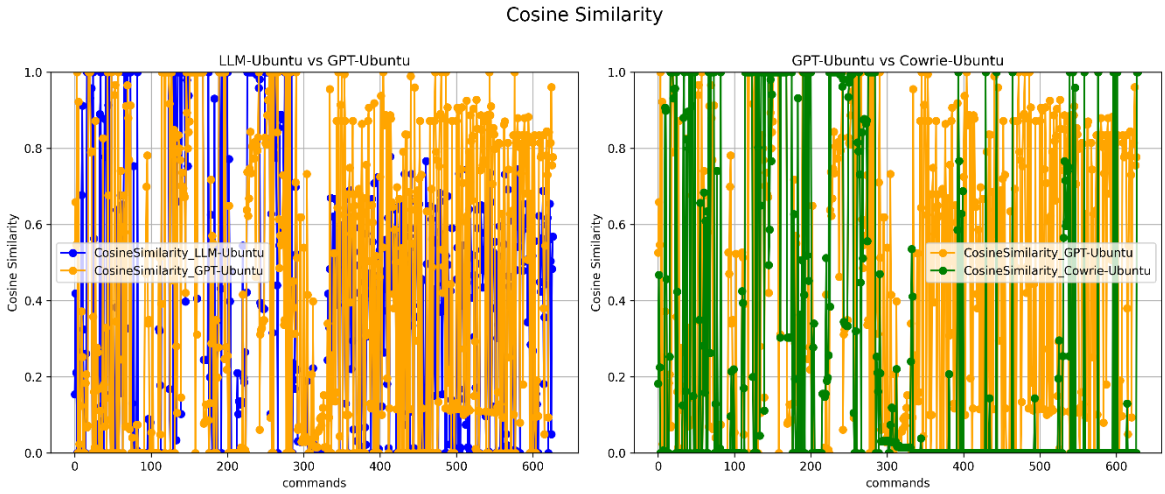
5.2.1.2. Kết quả

Hình 3 trình bày sự tương đồng Cosine của các phản hồi từ máy thật Ubuntu với các hệ thống honeypot khác, bao gồm GPT Honeypot, LLMHoneypot và Cowrie.

CosineSimilarity_LLM-Ubuntu: Phản hồi từ LLMHoneypot so với máy thật.

CosineSimilarity_GPT-Ubuntu: Phản hồi từ GPT Honeypot so với máy thật.

CosineSimilarity_Cowrie-Ubuntu: Phản hồi từ Cowrie so với máy thật.



Hình 3. Cosine Similarity

GPT Honeypot thể hiện mức độ tương đồng cao với máy thật Ubuntu so với Cowrie và LLMHoneypot, đặc biệt ở các lệnh có độ phức tạp cao.

LLMHoneypot có sự tương đồng ổn định nhưng không cao như GPT Honeypot.

Cowrie, mặc dù cung cấp phản hồi hợp lệ, nhưng không đạt được mức độ tương đồng cao, đặc biệt ở các lệnh phức tạp hoặc ít phổ biến.

Điều này cho thấy GPT Honeypot có khả năng tái tạo phản hồi sát với môi trường thực tế, mang lại tiềm năng lớn trong việc phân tích hành vi tấn công.

5.2.2. So sánh cơ bản

5.2.2.1. Các chỉ số

Để đánh giá hiệu quả của các hệ thống honeypot khác nhau, phân chia các phản hồi thành bốn nhóm dựa trên sự tuân thủ logic của hệ điều hành và kết quả của các cuộc tấn công. Cụ thể, các nhóm được định nghĩa như sau:

- SALC (Successful Attack with Logic Compliance): Các cuộc tấn công thành công và tuân thủ logic của hệ điều hành.
- SALNLC (Successful Attack without Logic Compliance): Các cuộc tấn công thành công nhưng không tuân thủ logic của hệ điều hành.
- FALC (Failed Attack with Logic Compliance): Các cuộc tấn công thất bại nhưng tuân thủ logic của hệ điều hành.
- FALNLC (Failed Attack without Logic Compliance): Các cuộc tấn công thất bại và không tuân thủ logic của hệ điều hành.

Dựa trên bốn nhóm này, tính toán ba chỉ số quan trọng để so sánh hiệu suất của GPT Honeypot, LLMHoneypot và Cowrie:

Độ chính xác (Accuracy): Đo lường khả năng của hệ thống trong việc tạo ra các phản hồi nhất quán với logic của một hệ điều hành thực. Công thức tính:

$$Accuracy = \frac{SALC}{SALC + SALNLC}$$

Giá trị độ chính xác cao hơn thể hiện khả năng mô phỏng chính xác hơn hành vi của hệ điều hành, giảm nghi ngờ từ kẻ tấn công và nâng cao khả năng thu thập thông tin.

Khả năng hấp dẫn (Temptation): Đánh giá hiệu quả của hệ thống trong việc thu hút sự tham gia của kẻ tấn công. Công thức tính:

$$Temptation = \frac{SALC}{SALC + FALC}$$

Điểm số hấp dẫn cao hơn thể hiện hệ thống có khả năng thu hút kẻ tấn công tốt hơn, từ đó tăng cơ hội thu thập thông tin.

Tuân thủ logic hệ điều hành (OS Logic Compliance): Đánh giá khả năng của hệ thống trong việc tạo ra các phản hồi phù hợp với logic của một hệ điều hành thực. Công thức tính:

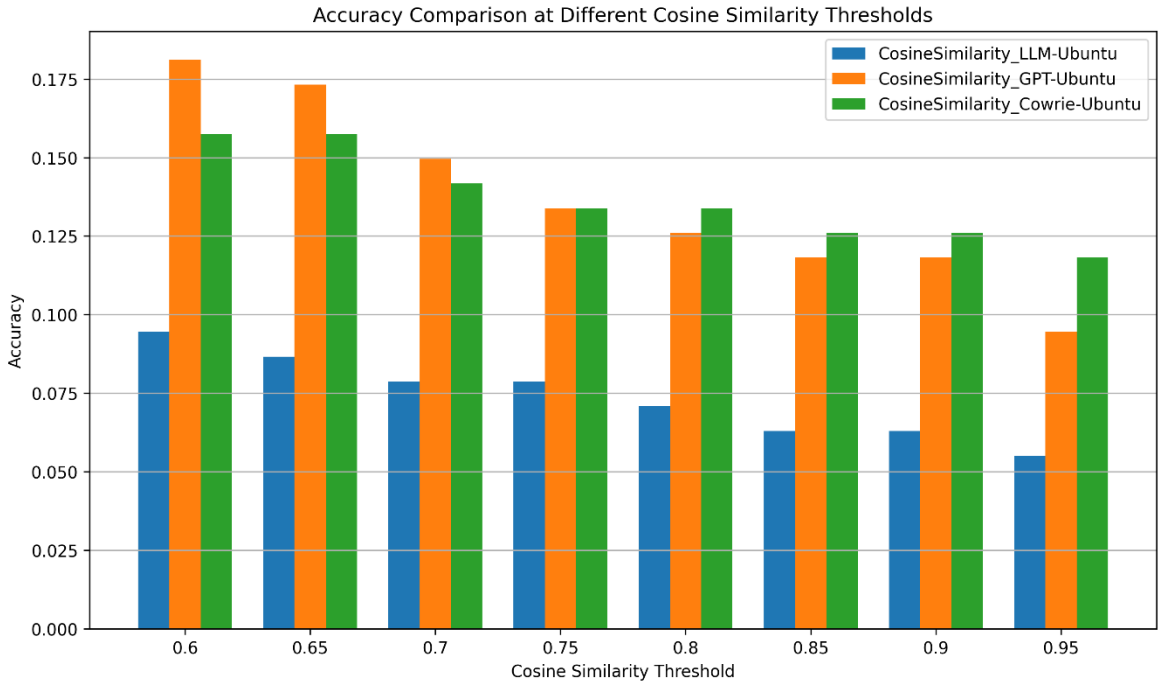
$$OS\ Logic\ Compliance = \frac{SALC\ +\ FALC}{Total\ Attacks}$$

Giá trị tuân thủ logic cao hơn đảm bảo tính chân thực của hệ thống mô phỏng, giúp giảm khả năng bị kẻ tấn công phát hiện.

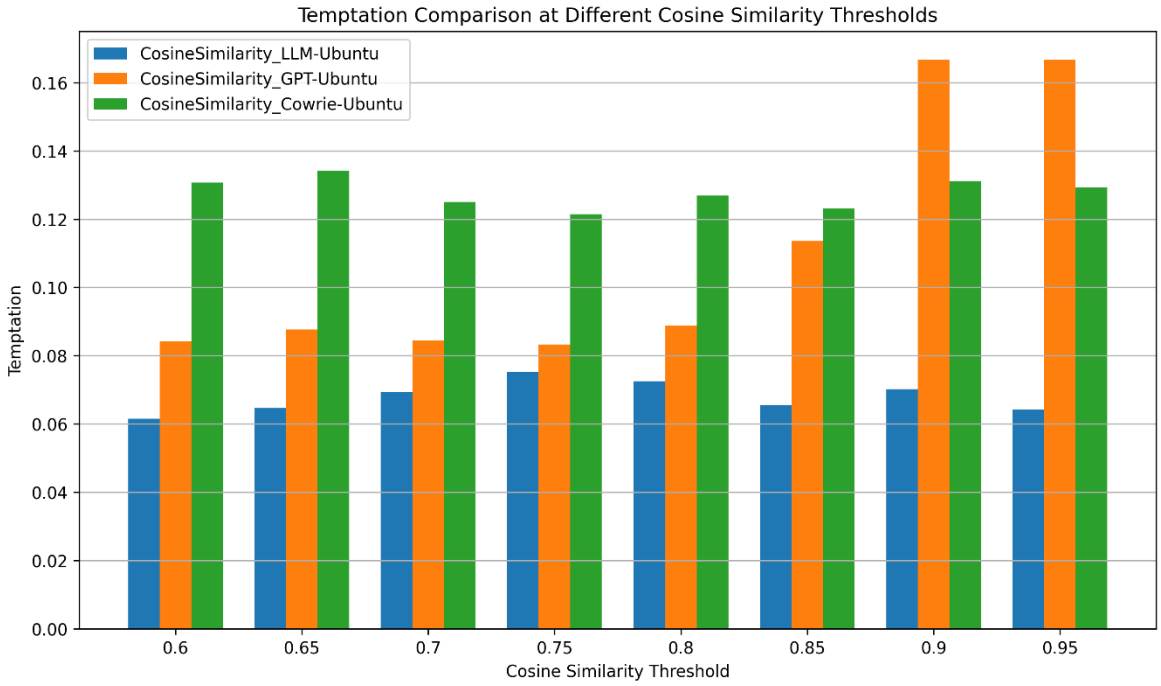
5.2.2.2. Kết quả

Biểu đồ cột trong Hình 4, Hình 5, Hình 6, minh họa hiệu suất của ba hệ thống honeypot trên ba chỉ số này ở các ngưỡng cosine khác nhau:

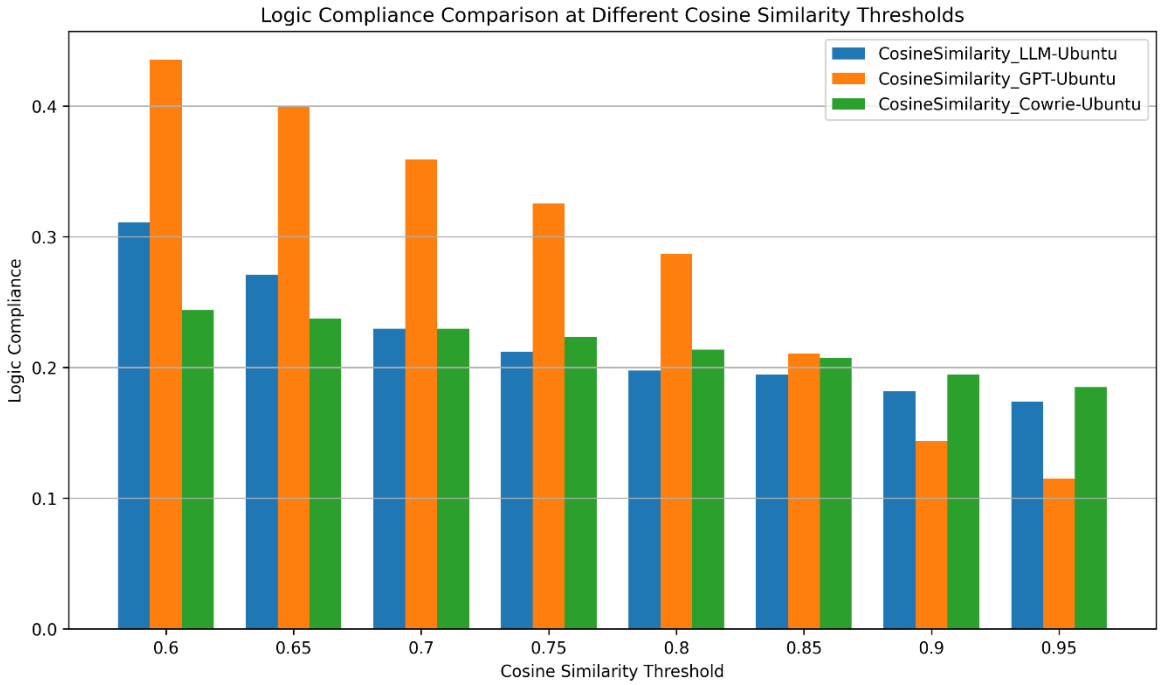
- Độ chính xác (Accuracy): GPT Honeypot và Cowrie đạt điểm số cao hơn so với LLMHoneypot. Điều này cho thấy cả hai hệ thống đều vượt trội trong việc tạo ra các phản hồi tuân thủ logic của hệ điều hành trong các cuộc tấn công thành công.
- Khả năng hấp dẫn (Temptation): GPT Honeypot đạt điểm số cao hơn LLMHoneypot nhưng thấp hơn Cowrie. Điều này cho thấy GPT Honeypot có hiệu quả trong việc thu hút sự tham gia của kẻ tấn công, mặc dù chưa bằng Cowrie.
- Tuân thủ logic hệ điều hành (OS Logic Compliance): GPT Honeypot đạt điểm số cao nhất, vượt qua Cowrie và LLMHoneypot. Điều này khẳng định khả năng vượt trội của GPT Honeypot trong việc tạo ra các phản hồi tuân thủ logic của hệ điều hành, yếu tố quan trọng để giảm nghi ngờ từ kẻ tấn công.



Hình 4. So sánh hiệu suất của ba mô hình ở các ngưỡng cosine



Hình 5. So sánh độ thu hút của ba mô hình ở các ngưỡng cosine



Hình 6. So sánh tuân thủ logic hệ điều hành của ba mô hình ở các ngưỡng cosine

Tóm lại, GPT Honeypot vượt trội hơn LLMHoneypot trên cả ba chỉ số và có hiệu suất tương đương với Cowrie. Kết quả này chứng minh tiềm năng của GPT Honeypot trong việc tạo ra một môi trường mô phỏng chân thực và hấp dẫn, đồng thời duy trì tính tuân thủ logic chặt chẽ với hành vi của hệ điều hành.

5.2.3. So sánh ở các ngưỡng cosine

Để đánh giá độ chính xác (accuracy) của GPT Honeypot, chúng tôi tiến hành so sánh các phản hồi của nó với phản hồi trên một máy thật chạy Ubuntu. Việc đánh giá dựa trên ngưỡng tương đồng cosine giữa các phản hồi, cụ thể là các ngưỡng $\geq 0.95; \geq 0.9; \geq 0.85; \geq 0.8; \geq 0.75 \geq 0.7; \geq 0.65; \geq 0.6$

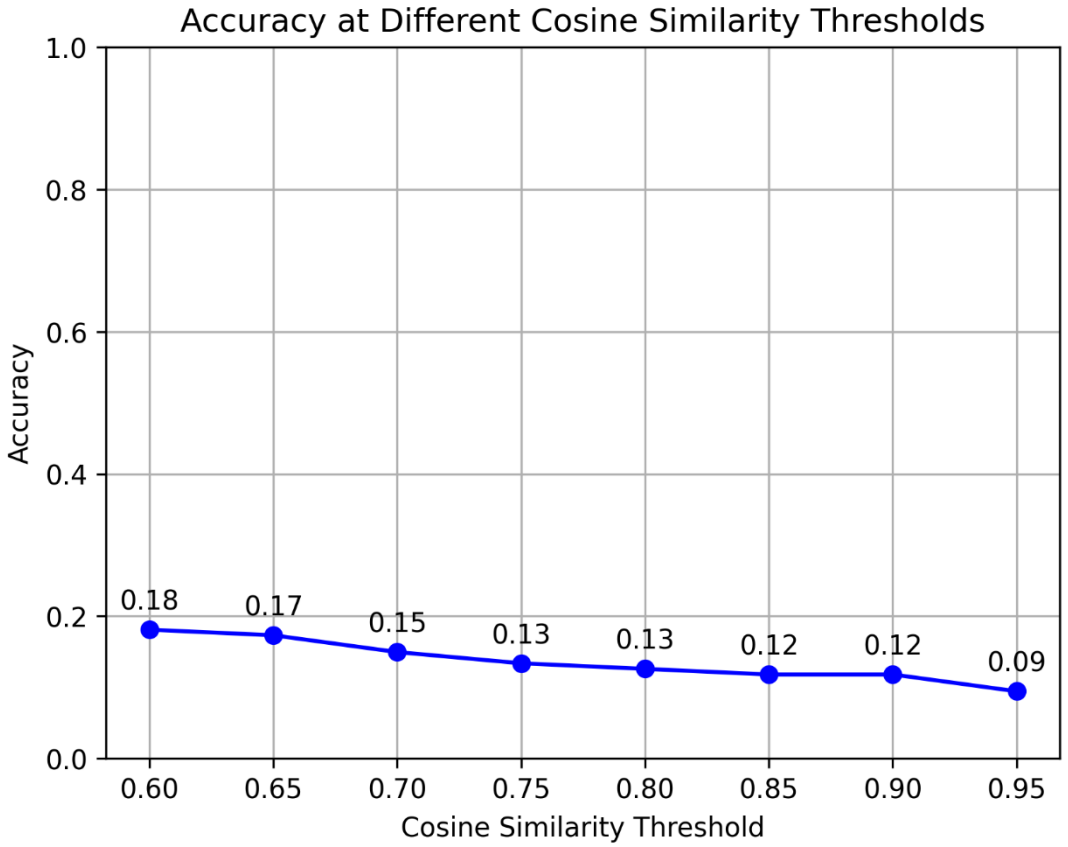
Kết quả được trình bày trong biểu đồ Hình 7, Hình 8, Hình 9, minh họa mối quan hệ giữa ngưỡng tương đồng cosine với độ chính xác, độ thu hút và khả năng tuân thủ logic của hệ điều hành:

Ở ngưỡng cao nhất (≥ 0.9) phản hồi của GPT Honeypot rất sát với thực tế trong những trường hợp đáp ứng mức tương đồng cao.

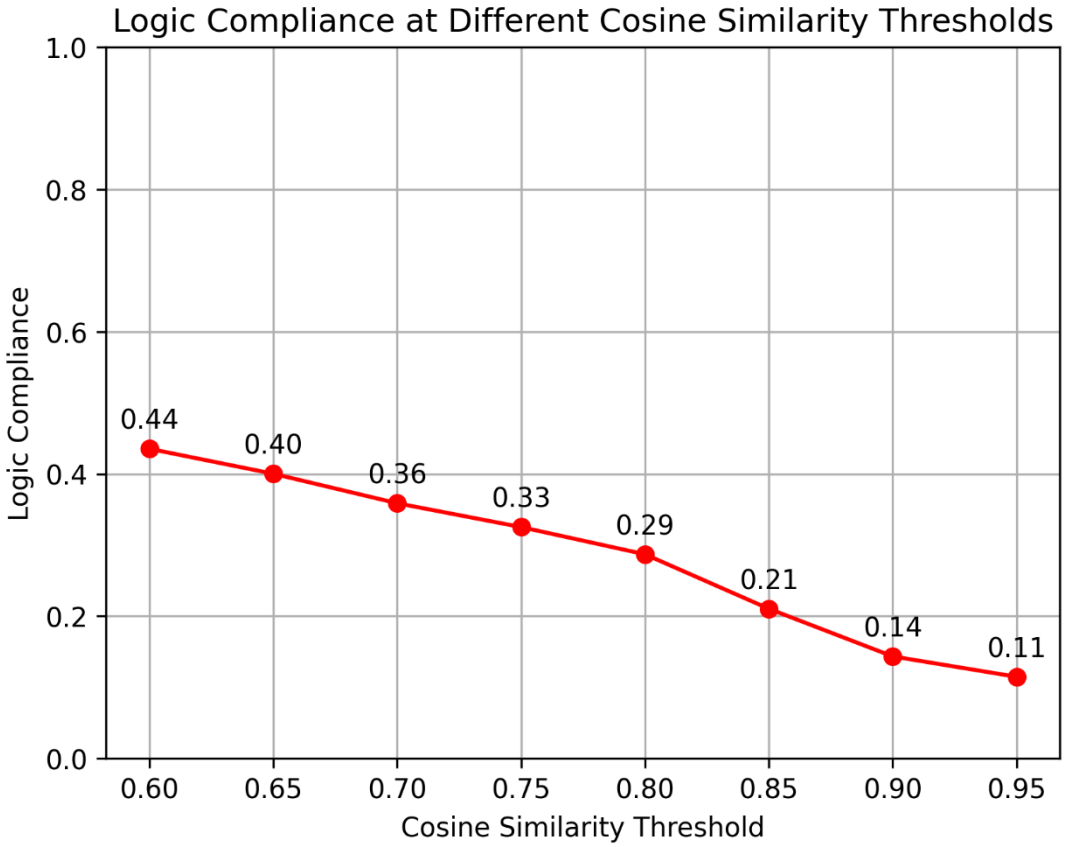
Khi giảm ngưỡng xuống (≥ 0.8) cho thấy khả năng của GPT Honeypot trong việc tái tạo logic hợp lý ở mức tương đồng trung bình.

Tại ngưỡng (≥ 0.7) thể hiện sự cải thiện trong việc thu hẹp khoảng cách giữa phản hồi của GPT Honeypot và máy thật.

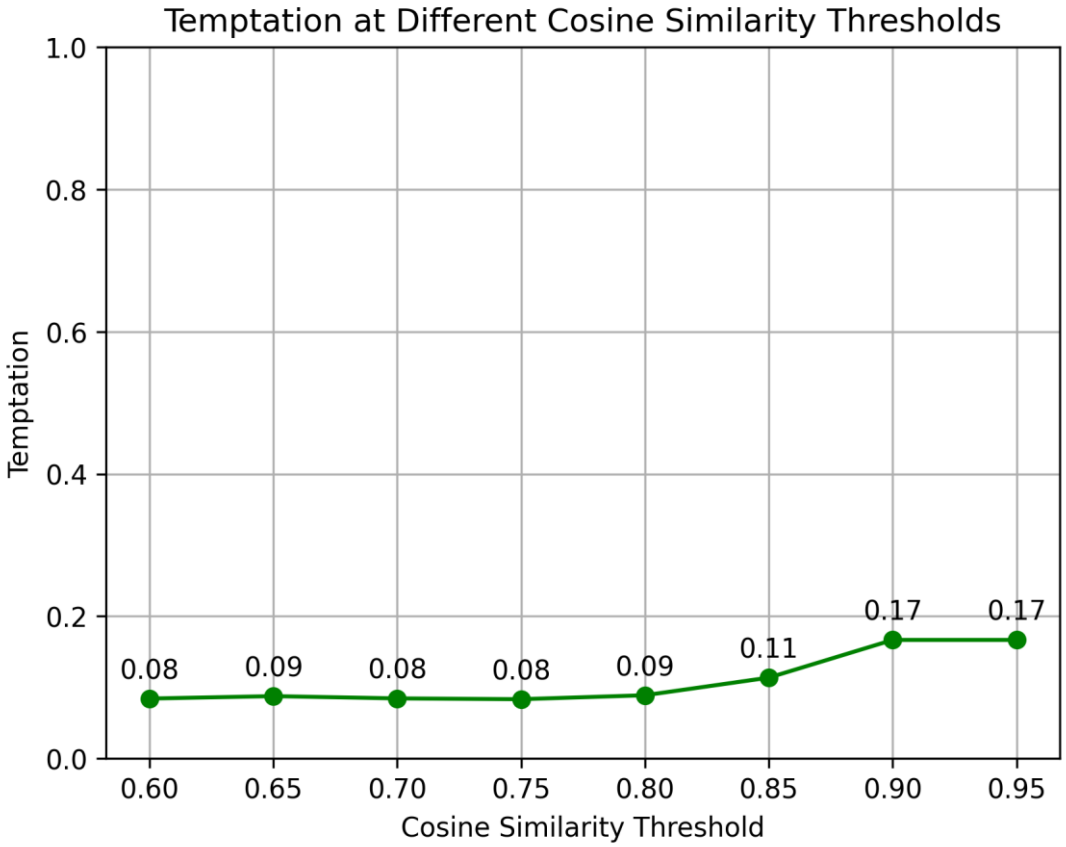
Cuối cùng, ở ngưỡng thấp nhất (≥ 0.6) cho thấy GPT Honeypot có khả năng tạo ra phản hồi tương đồng đáng kể với máy thật khi yêu cầu tương đồng được giảm bớt.



Hình 7. Hiệu suất khi thay đổi ngưỡng tương đồng



Hình 8. Độ thu hút khi thay đổi ngưỡng tương đồng



Hình 9. Tuân thủ logic hệ điều hành khi thay đổi ngưỡng tương đồng

Chương 6. KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

6.1. Kết luận

6.1.1. Tóm tắt vấn đề

Các honeypot truyền thống thường gặp khó khăn trong việc cân bằng giữa tính chân thực, khả năng tương tác lâu dài, và hiệu quả tài nguyên (được gọi là trilemma của honeypot). Những hạn chế này khiến chúng khó mô phỏng các cuộc tấn công phức tạp, đặc biệt khi đối mặt với các tác nhân xảo quyệt và chiến lược tấn công kéo dài. Để giải quyết vấn đề, mô hình SSH honeypot sử dụng LLM (Large Language Model) được phát triển, tích hợp khả năng xử lý ngôn ngữ tự nhiên của ChatGPT với quản lý linh hoạt prompt và giả lập giao thức terminal. Kiến trúc này nhằm nâng cao tính thực tế của honeypot, cho phép xử lý các chuỗi tương tác phức tạp trong khi tối ưu hóa tài nguyên và cải thiện khả năng phân tích hành vi của kẻ tấn công.

6.1.2. Phương pháp

Mô hình SSH honeypot sử dụng LLM được xây dựng với kiến trúc gồm ba thành phần chính: ChatGPT, Prompt Manager, và Terminal Protocol Proxy. Phương pháp hoạt động như sau:

- Terminal Protocol Proxy: Chịu trách nhiệm quản lý giao thức terminal, bao gồm thiết lập kết nối, giả lập dấu vân tay, đóng gói và phân tích các tin nhắn từ kẻ tấn công. Proxy này được xây dựng dựa trên lớp giao thức của Cowrie để hỗ trợ các giao thức như SSH và Telnet.
- Prompt Manager: Quản lý và xây dựng prompt linh hoạt, bao gồm cả nội dung tĩnh (cài đặt honeypot và nguyên tắc hoạt động) và động (lịch sử tương tác và trạng thái hệ thống). Phần này đảm bảo tính nhất quán và tối ưu hóa thông tin cung cấp cho ChatGPT, sử dụng chiến lược cắt tỉa để duy trì giới hạn chiều dài bối cảnh.

- ChatGPT: Đóng vai trò xử lý ngôn ngữ tự nhiên, phân tích các lệnh của kẻ tấn công và tạo phản hồi phù hợp. ChatGPT dựa trên các chiến lược "Chuỗi Suy Nghĩ" (Chain of Thought) để xử lý các tương tác phức tạp và dài hạn, bao gồm đánh giá tác động của lệnh lên hệ thống và xác định phản ứng tiếp theo.

Phương pháp này kết hợp sự linh hoạt trong tương tác của ChatGPT với các chiến lược quản lý thông minh của Prompt Manager và Terminal Protocol Proxy, giúp mô hình hóa các cuộc tấn công thực tế một cách hiệu quả và duy trì tài nguyên tối ưu.

6.1.3. Kết quả đạt được

Về độ tương đồng so với hệ thống thật (Hình 2): GPT Honey có số lượng phản hồi đạt độ tương đồng cao hơn hẳn 2 mô hình còn lại (Cowrie và LLMHoneypot).

Về hiệu suất (Hình 4): GPT Honey và Cowrie (0.14) cao hơn hẳn so với LLMHoneypot (0.07).

Về độ thu hút (Hình 4): GPT Honey (0.08) tuy cao hơn LLMHoneypot (0.06) nhưng vẫn thấp hơn Cowrie (0.12).

Về độ logic so với hệ điều hành (Hình 4): GPT Honey (0.68) cao hơn hẳn so với 2 mô hình còn lại (0.56 và 0.58).

Về hiệu suất qua các ngưỡng độ tương đồng (Hình 5): Ta thấy hiệu suất càng giảm khi ngưỡng độ tương đồng càng tăng, chứng minh rằng đa phần các lệnh thực hiện tấn công thành công và đúng với logic hệ điều hành có độ tương đồng không cao so với máy thật.

Qua các kết quả trên có thể thấy tuy các chỉ số còn thấp, nhưng khi so với các mô hình khác cũng không quá chênh lệch.

6.2. Hướng phát triển

Điều chỉnh mô hình để hiệu suất hoạt động tốt hơn, sau đó có thể kết hợp mô hình này với SSH proxy của honeypot Cowrie, tiến hành hiện thực hệ thống có thể triển khai chạy thử cho các ngữ cảnh tấn công khác nhau.

TÀI LIỆU THAM KHẢO

[1] Ziyang Wang; Jianzhou You; Haining Wang; Tianwei Yuan; Shichao Lv; Yang Wang, "HoneyGPT: Breaking the Trilemma in Terminal Honeypots with Large Language Model," 2024.

[2] "Hugging Face," [Online]. Available: https://huggingface.co/datasets/hotal/honeypot_logs. [Accessed 7 January 2025].

[3] Hakan T. Otal and M. Abdullah Canba, "LLM Honeypot: Leveraging Large Language Models as Advanced Interactive Honeypot Systems," 2024.

[4] "GeeksforGeeks," [Online]. Available: https://www.geeksforgeeks.org/cosine-similarity/. [Accessed 7 January 2025].

[5] "200lab," 22 April 2024. [Online]. Available: https://200lab.io/blog/transformer-cong-nghe-dang-sau-chatgpt-va-bard?srsltid=AfmBOoqkpbioXLwgTFqxA21Y2OkGWTCLxbfxIELa7-z1JZLS1KOpWwjO. [Accessed 8 January 2025].

[6] [Online]. Available: https://docs.cowrie.org/en/latest/README.html. [Accessed 8 January 2025].

[7] "GitHub," [Online]. Available: https://github.com/cowrie/cowrie. [Accessed 8 January 2025].

[8] [Online]. Available: https://communityhoneynetwork.readthedocs.io/en/v1.8/cowrie/#acknowledgements. [Accessed 8 January 2025].