

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

TRẦN LÊ MINH NGỌC

LƯU THỊ HUỲNH NHƯ

KHÓA LUẬN TỐT NGHIỆP

ĐÁNH GIÁ HIỆU QUẢ CỦA CÁC PHƯƠNG PHÁP PHÁT  
HIỆN VÀ GIẢI NÉN TẬP TIN THỰC THI BỊ ĐÓNG GÓI  
TRONG PHÁT HIỆN MÃ ĐỘC WINDOWS

**Evaluating the effectiveness of detecting and unpacking methods for  
packed executable files in Windows malware detection**

CỬ NHÂN NGÀNH AN TOÀN THÔNG TIN

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**

**TRẦN LÊ MINH NGỌC - 21521195**

**LƯU THỊ HUỲNH NHƯ' - 21521242**

**KHÓA LUẬN TỐT NGHIỆP**

**ĐÁNH GIÁ HIỆU QUẢ CỦA CÁC PHƯƠNG PHÁP PHÁT  
HIỆN VÀ GIẢI NÉN TẬP TIN THỰC THI BỊ ĐÓNG GÓI  
TRONG PHÁT HIỆN MÃ ĐỘC WINDOWS**

**Evaluating the effectiveness of detecting and unpacking methods for  
packed executable files in Windows malware detection**

**CỬ NHÂN NGÀNH AN TOÀN THÔNG TIN**

**GIẢNG VIÊN HƯỚNG DẪN**

**Ths. Đỗ Thị Thu Hiền**

**Ths. Nghi Hoàng Khoa**

**TP. HỒ CHÍ MINH, 2025**

# THÔNG TIN HỘI ĐỒNG CHẤM KHÓA LUẬN TỐT NGHIỆP

Hội đồng chấm khóa luận tốt nghiệp, thành lập theo Quyết định số 745/QĐ-ĐHCNTT ngày 27 tháng 06 năm 2025 của Hiệu trưởng Trường Đại học Công nghệ Thông tin.

Chủ tịch: TS. Nguyễn Ngọc Tự

Thư ký: ThS. Trần Thị Dung

Ủy viên: ThS. Nguyễn Khánh Thuật

# LỜI CẢM ƠN

Đầu tiên, nhóm thực hiện xin gửi lời cảm ơn sâu sắc đến cô Đỗ Thị Thu Hiền và thầy Nghi Hoàng Khoa đã tạo điều kiện, giúp nhóm học tập và có được những kiến thức cơ bản làm tiền đề giúp nhóm có thể hoàn thành được đồ án này. Nhờ sự hướng dẫn tận tình và chu đáo của thầy cô, nhóm đã học hỏi được nhiều kinh nghiệm và hoàn thành thuận lợi, đúng tiến độ cho đồ án của mình.

Trong quá trình thực hiện đồ án, nhóm chúng tôi luôn giữ một tinh thần cầu tiến, học hỏi và cải thiện từ những sai lầm, tham khảo từ nhiều nguồn tài liệu khác nhau và luôn mong tạo ra được sản phẩm chất lượng nhất có thể. Tuy nhiên, do vốn kiến thức còn hạn chế trong quá trình trau dồi từng ngày, nhóm không thể tránh được những sai sót, vì vậy chúng tôi mong rằng thầy cô sẽ đưa ra nhận xét một cách chân thành để chúng tôi học hỏi thêm kinh nghiệm nhằm mục đích phục vụ tốt các dự án khác trong tương lai.

Nhóm xin chân thành cảm ơn thầy và cô!

**Trần Lê Minh Ngọc**

**Lưu Thị Huỳnh Như**

# MỤC LỤC

TÓM TẮT KHOÁ LUẬN . . . . .	1
<b>1 TỔNG QUAN ĐỀ TÀI</b>	<b>2</b>
1.1 Lý do chọn đề tài . . . . .	2
1.2 Các nghiên cứu liên quan . . . . .	3
1.3 Vấn đề Khóa luận tập trung giải quyết . . . . .	4
1.4 Mục tiêu, đối tượng và phạm vi nghiên cứu . . . . .	5
1.4.1 Mục tiêu nghiên cứu . . . . .	5
1.4.2 Đối tượng nghiên cứu . . . . .	5
1.4.3 Phạm vi nghiên cứu . . . . .	5
<b>2 CƠ SỞ LÝ THUYẾT</b>	<b>7</b>
2.1 Tập tin thực thi Windows . . . . .	7
2.1.1 Portable Executable – PE là gì? . . . . .	7
2.1.2 Cấu trúc file PE . . . . .	7
2.2 Trình đóng gói . . . . .	11
2.2.1 Trình đóng gói (Packer) là gì? . . . . .	11
2.2.2 Cách đóng gói tập tin . . . . .	12
2.2.3 Các công cụ đóng gói phổ biến . . . . .	14
2.2.4 Lựa chọn tập dữ liệu dựa trên các trình đóng gói . . . . .	24
2.3 Các công cụ phát hiện và giải nén các tập tin thực thi Windows . . . . .	25
2.3.1 Unipacker . . . . .	25
2.3.2 ClamAV . . . . .	27
2.3.3 Detect It Easy . . . . .	31
2.3.4 PEiD . . . . .	33
2.4 Học máy . . . . .	35
2.4.1 Học máy là gì? . . . . .	35
2.4.2 Phân loại học máy . . . . .	35
2.4.3 Một số mô hình học máy . . . . .	37
2.4.4 Các chỉ số đánh giá mô hình học máy . . . . .	42

<b>3</b>	<b>PHƯƠNG PHÁP THỰC HIỆN</b>	<b>44</b>
3.1	Môi trường thực hiện . . . . .	44
3.2	Mô hình tổng quan phương pháp thực hiện . . . . .	44
3.3	Giai đoạn 1 - Đánh giá khả năng phát hiện các tệp bị đóng gói . . . . .	45
3.3.1	Mục tiêu và cách tiếp cận . . . . .	45
3.3.2	Quy trình thực hiện . . . . .	46
3.4	Giai đoạn 2 - Đánh giá khả năng giải nén các tệp đã đóng gói . . . . .	49
3.4.1	Mục tiêu và cách tiếp cận . . . . .	49
3.4.2	Quy trình thực hiện . . . . .	50
3.5	Giai đoạn 3 - Đánh giá hiệu quả của việc đào tạo các mô hình học máy	52
3.5.1	Mục tiêu và cách tiếp cận . . . . .	52
3.5.2	Quy trình thực hiện . . . . .	54
<b>4</b>	<b>KẾT QUẢ THỰC NGHIỆM</b>	<b>58</b>
4.1	Bộ dữ liệu . . . . .	58
4.2	Kết quả giai đoạn 1 - Đánh giá khả năng phát hiện các tệp bị đóng gói	59
4.3	Kết quả giai đoạn 2 - Đánh giá khả năng giải nén các tệp đã đóng gói	61
4.4	Kết quả so sánh tổng thể của các công cụ . . . . .	63
4.5	Kết quả giai đoạn 3 - Đánh giá hiệu quả của việc đào tạo các mô hình học máy . . . . .	65
4.5.1	Trích xuất đặt trưng . . . . .	65
4.5.2	Huấn luyện và mô hình học máy . . . . .	66
4.5.3	Kịch bản 1: Huấn luyện máy học với các file bị nén . . . . .	67
4.5.4	Kịch bản 2: Huấn luyện máy học với các file đã giải nén (bao gồm cả file tương đồng với file ban đầu và file không tương đồng)	68
4.5.5	Kịch bản 3: Huấn luyện máy học với file đã giải nén (chỉ với file tương đồng với file ban đầu) . . . . .	69
4.5.6	Tổng kết . . . . .	70
<b>5</b>	<b>KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN</b>	<b>73</b>
5.1	Kết luận . . . . .	73
5.2	Hướng phát triển . . . . .	74
	<b>TÀI LIỆU THAM KHẢO</b>	<b>76</b>

# DANH SÁCH HÌNH ẢNH

2.1	Cấu trúc một file PE . . . . .	8
2.2	Cấu trúc một tệp thực thi Windows đã bị đóng gói . . . . .	12
2.3	Công cụ nén tệp tin UPX . . . . .	15
2.4	Công cụ nén tệp tin ASPack . . . . .	16
2.5	Công nghệ nén tệp tin MPRESS . . . . .	18
2.6	Công nghệ nén tệp tin FSG . . . . .	19
2.7	Công nghệ nén tệp tin MEW . . . . .	21
2.8	Công nghệ nén tệp tin PETITE . . . . .	22
2.9	Công cụ Unipacker . . . . .	25
2.10	Các quy tắc yara nhận biết file bị nén bởi UPX của Unipacker . . . . .	26
2.11	Các quy tắc yara nhận biết file bị nén bởi PETITE của unipacker . . . . .	27
2.12	Các quy tắc yara nhận biết file bị nén bởi MPRESS của Unipacker . . . . .	27
2.13	Trình giải nén UPX của Unipacker . . . . .	28
2.14	Trình giải nén PETITE của Unipacker . . . . .	28
2.15	Công cụ ClamAV . . . . .	29
2.16	Các file chứa đoạn code liên quan đến UPX trong ClamAV . . . . .	29
2.17	Đoạn code nhận biết UPX của ClamAV . . . . .	30
2.18	Đoạn code giải nén upx_inflate2b . . . . .	30
2.19	Đoạn code giải nén upx_inflate2d . . . . .	31
2.20	Công cụ Detect It Easy . . . . .	31
2.21	Đoạn code nhận biết FSG bằng các signature của DiE . . . . .	33
2.22	Công cụ PEiD . . . . .	34
2.23	Minh họa cách hoạt động của mô hình Random Forest . . . . .	37
2.24	Minh họa cách hoạt động của mô hình XGBoost . . . . .	39
2.25	Minh họa cách hoạt động của mô hình K-Nearest Neighbors . . . . .	40
2.26	Minh họa siêu phẳng phân tách hai lớp dữ liệu của SVM trong không gian hai chiều và ba chiều . . . . .	42
3.1	Pipeline quy trình thực hiện nghiên cứu . . . . .	45
3.2	Sơ đồ tổng quan hóa quy trình thực thi giai đoạn 1 . . . . .	47

3.3	Phân tích tệp tin với PEiD phiên bản command line . . . . .	47
3.4	Kiểm tra thông tin tệp trong Unipacker . . . . .	48
3.5	Sơ đồ tổng quan hóa quy trình thực thi giai đoạn 2 . . . . .	50
3.6	Kết quả quét tệp tin bằng ClamAV . . . . .	51
3.7	Sơ đồ tổng quan hóa quy trình thực thi giai đoạn 3 . . . . .	54



# DANH SÁCH BẢNG BIỂU

3.1	Bảng so sánh sự khác nhau khi trích xuất đặc trưng giữa 3 kịch bản . .	56
4.1	Các packer mà công cụ có thể phát hiện được . . . . .	59
4.2	Thời gian chạy và độ chính xác của từng công cụ khi detect file . . . . .	60
4.3	Thời gian chạy và độ chính xác của từng công cụ khi unpack file . . .	62
4.4	So sánh tổng hợp của các công cụ . . . . .	64
4.5	Kết quả của các mô hình kịch bản 1 . . . . .	68
4.6	Kết quả của các mô hình kịch bản 2 . . . . .	69
4.7	Kết quả của các mô hình kịch bản 3 . . . . .	70
4.8	So sánh hiệu quả các mô hình học máy theo từng kịch bản huấn luyện	71

# TÓM TẮT KHÓA LUẬN

Trong khóa luận này, nhóm chúng tôi sẽ tập trung nghiên cứu các công cụ phát hiện và giải nén các tập tin thực thi Windows khi các tập tin này bị đóng gói để ngăn chặn việc phân tích ngược và che giấu hành vi bất thường.

Trong phạm vi của Khóa luận tốt nghiệp, chúng tôi sẽ tập trung nghiên cứu cấu trúc của các tập tin thực thi Windows (Portable Executable - PE), bao gồm các thành phần chính và cách chúng hoạt động. Bên cạnh đó, nhóm sẽ tìm hiểu các trình đóng gói phổ biến hiện nay trên thị trường, cơ chế hoạt động của chúng, cách chúng tác động đến các tệp tin PE, từ đó phân tích cách các trình đóng gói này bảo vệ mã nguồn và che giấu dữ liệu. Để phục vụ quá trình nghiên cứu, nhóm sử dụng bốn công cụ chính: Unipacker, ClamAV, PeiD và Detect It Easy (DIE). Đây đều là những công cụ phổ biến trong việc nhận diện và phân loại các trình đóng gói (packer). Hai trong số bốn công cụ trên còn có khả năng giải nén, giúp truy xuất mã gốc nhanh chóng và hiệu quả. Chúng tôi sẽ đánh giá hiệu quả của từng công cụ trong việc phát hiện và giải nén, sau đó đưa ra sự so sánh và nhận xét về các công cụ này.

Cuối cùng, sau khi thực hiện quá trình giải nén, các mẫu đã được xử lý sẽ được sử dụng làm đầu vào cho các mô hình học máy hiện hành nhằm đánh giá hiệu quả phát hiện mã độc. Nhờ đó không chỉ giúp kiểm chứng chất lượng đầu ra của các công cụ giải nén, mà còn góp phần đánh giá mức độ ảnh hưởng của các kỹ thuật đóng gói đến độ chính xác của mô hình phát hiện.

# Chương 1

## TỔNG QUAN ĐỀ TÀI

### 1.1 Lý do chọn đề tài

*“Trong 6 tháng đầu năm 2024, tình hình an toàn thông tin trên toàn cầu tiếp tục diễn biến phức tạp và đáng lo ngại. Các cuộc tấn công mạng gia tăng cả về số lượng và mức độ tinh vi, nhắm vào các cơ quan chính phủ, doanh nghiệp và tổ chức cá nhân. Số lượng các cuộc tấn công trên toàn thế giới đã tăng gấp 1,3 lần so với cùng kỳ 2023, riêng khu vực Việt Nam đã tăng gần gấp 2 lần và để lại nhiều ảnh hưởng nghiêm trọng đến cả uy tín, tài sản của nhiều doanh nghiệp, tổ chức.”* – Trung tâm An toàn thông tin VNPT (VNPT Cyber Immunity) ghi nhận. [3]

Hiện nay, vấn nạn an toàn thông tin ngày càng nghiêm trọng khi các loại mã độc và phần mềm độc hại liên tục được phát triển với những kỹ thuật tinh vi nhằm qua mặt các hệ thống bảo mật. Những cuộc tấn công mạng không chỉ gây tổn hại đến dữ liệu và tài sản số mà còn làm gián đoạn hoạt động của các tổ chức, doanh nghiệp, thậm chí đe dọa an ninh quốc gia. Thiệt hại từ các vụ tấn công mạng có thể lên đến hàng tỷ USD mỗi năm, bao gồm mất dữ liệu, gián đoạn kinh doanh và thiệt hại danh tiếng. Bên cạnh đó, đối với cá nhân, việc mất thông tin cá nhân hay bị xâm phạm quyền riêng tư đang trở thành nỗi lo thường trực trong thời đại mà đâu đâu cũng là Internet. Điều này không chỉ đe dọa sự an toàn của hệ thống thông tin mà còn đặt ra những thách thức lớn trong việc bảo vệ dữ liệu và tài sản trong kỷ nguyên kỹ thuật số. Như lời Robert Mueller, cựu Giám đốc FBI, từng nói: *“There are only two types of companies: those that have been hacked, and those that will be.”*. Câu nói này cũng đã nhấn mạnh mức độ nghiêm trọng và tất yếu của các cuộc tấn công mạng đối với mọi tổ chức và doanh nghiệp.

Một trong những kỹ thuật đã xuất hiện từ khá lâu và được các tin tặc sử dụng rộng rãi để che giấu mã độc là đóng gói (packing). Việc đóng gói không chỉ giúp giảm kích

thước tập tin mà còn che giấu mã nguồn, làm phức tạp quá trình phân tích và phát hiện mã độc của các chuyên gia an ninh mạng. Kỹ thuật này tạo ra những rào cản lớn trong việc nhận diện và ngăn chặn các cuộc tấn công mạng, đồng thời tiềm ẩn nguy cơ cao cho hệ thống thông tin của các doanh nghiệp và người dùng cá nhân. Do đó, việc nghiên cứu và phát triển các phương pháp hiệu quả để phát hiện và giải nén các tập tin thực thi đóng gói là vô cùng cần thiết. Việc này không chỉ giúp nâng cao khả năng bảo vệ hệ thống thông tin mà còn giảm thiểu những rủi ro từ các mối đe dọa ngày càng gia tăng khi thế giới đang chuyển đổi số hóa.

## 1.2 Các nghiên cứu liên quan

Trong bài nghiên cứu *“File packing from the malware perspective: Techniques, analysis approaches, and directions for enhancements”* (2022) của T. Muralidharan và đồng sự [7], họ đã tiến hành nghiên cứu toàn diện để phân tích vai trò của các kỹ thuật đóng gói trong việc ẩn giấu phần mềm độc hại và tăng độ khó của các hệ thống phát hiện hiện có. Các tác giả đã điều tra trên 24.000 tệp PE từ năm 2010 đến năm 2021 và phát hiện ra rằng tỷ lệ tệp bị đóng gói khá cao (khoảng 21%), trong đó, tỷ lệ tệp độc hại bị đóng gói cao hơn đáng kể so với tệp lành tính. Ngoài ra, vài báo phân tích và phân loại 23 phương pháp được đề xuất trong nghiên cứu học thuật để phát hiện tệp đóng gói, bao gồm các phương pháp phân tích tĩnh, động và kết hợp, các phương pháp có hoặc không có dùng máy học.

Nghiên cứu cũng chỉ ra rằng nhiều kỹ thuật đóng gói hiện đại sử dụng các cơ chế chống phân tích như chống gỡ lỗi, chống mô phỏng hoặc đóng gói nhiều lớp, khiến việc phân tích và giải nén ngày càng phức tạp. Cụ thể, bài báo này đề xuất một phương pháp mới để sử dụng dữ liệu chuỗi thời gian đa biến (multivariate time series data) làm đầu vào cho mô hình máy học để cải thiện độ chính xác khi xác định tệp đóng gói. Tuy nhiên, nghiên cứu này chỉ ở cấp độ đề xuất phương pháp và vẫn chưa triển khai thử nghiệm thực tế các mô hình học máy. Nó cũng thừa nhận rằng các trình đóng gói tùy chỉnh vẫn là một thách thức lớn đối với công nghệ phát hiện hiện tại.

Trong bài nghiên cứu *“When Malware is Packin’ Heat; Limits of Machine Learning Classifiers Based on Static Analysis Features”* (2020) [1], Aghakhani và đồng sự đã nghiên cứu về việc: liệu việc trích xuất các đặc trưng từ tệp bị đóng gói để huấn luyện mô hình học máy có đủ để giúp chúng phân loại chính xác phần mềm độc hại hay không? Và để trả lời vấn đề này, nhóm tác giả đã xây dựng hai bộ dữ liệu gồm các tập tin thực thi Windows đã được gán nhãn là độc hại hoặc lành tính, đồng thời xác định trạng thái đóng gói (packed/unpacked). Từ các tập tin này, họ trích xuất chín nhóm đặc trưng phân tích tĩnh phổ biến như: PE headers, PE sections, DLL imports,

API imports, Rich headers, n-gram bytes/opcodes,... và sử dụng nhiều thuật toán học máy như Random Forest, SVM và mạng nơ-ron để xây dựng mô hình phân loại.

Kết quả cho thấy rằng trong khi một số thông tin được bảo toàn sau khi đóng gói và có thể hữu ích để phát hiện phần mềm độc hại, thì các tính năng phân tích tĩnh thường không đủ mạnh để giúp các mô hình học máy hoạt động tốt với các trình đóng gói chưa từng thấy trước đó hoặc khi đối mặt với các kỹ thuật mã hóa mạnh. Ngoài ra, nghiên cứu cho thấy các mô hình có thể dễ dàng bị đánh lừa bởi các mẫu đối kháng (adversarial samples) làm giảm đáng kể độ chính xác. Đặc biệt, chúng chứng minh rằng một số mô hình học máy hiện tại chủ yếu học về sự hiện diện của trình đóng gói thay vì hành vi thực tế của phần mềm độc hại, dẫn đến tỷ lệ dương tính giả (*false positive rates*) cao khi gặp phải phần mềm lành tính nhưng bị đóng gói.

Một nghiên cứu khác gần đây của Gibert và cộng sự mang tên “*Assessing the impact of packing on static machine learning-based malware detection and classification systems*”(2025)[5] đã đánh giá trực tiếp tác động của các kỹ thuật đóng gói đến hiệu suất của các mô hình học máy tĩnh trong việc phát hiện và phân loại phần mềm độc hại. Nhóm tác giả đã áp dụng tám trình đóng gói khác nhau (như: UPX, MPress, Hyperion, Themida, Enigma, Mangle, v.v.) lên hai tập dữ liệu là Malimg và BODMAS, đồng thời huấn luyện và kiểm tra hiệu suất của nhiều mô hình học máy, bao gồm LightGBM (dựa trên đặc trưng), MalConv (end-to-end) và các mô hình nhận dạng hình ảnh như ResNet, EfficientNet và Swin Transformer.

Kết quả cho thấy các kỹ thuật đóng gói làm giảm đáng kể độ chính xác của các mô hình học máy, đặc biệt là các mô hình dựa trên hình ảnh. Ngay cả một mô hình mạnh mẽ như LightGBM cũng bị ảnh hưởng khi xử lý các mẫu bị đóng gói bằng các công cụ nén phần mềm độc hại mạnh mẽ như Mangle hoặc Hyperion.

### 1.3 Vấn đề Khóa luận tập trung giải quyết

Các tệp thực thi Windows bị đóng gói sẽ gây khó khăn trong quá trình phát hiện mã độc, dẫn đến kết quả cuối cùng không chính xác. Do đó, việc phát hiện và giải nén các tệp bị đóng gói đóng vai trò quan trọng tổng quát trong quá trình phân tích mã độc. Tuy nhiên, hiện nay có nhiều công cụ hỗ trợ phát hiện và giải nén các tệp đã đóng gói, nhưng hiệu quả của chúng chưa được đánh giá toàn diện.

Khóa luận này tập trung vào việc khảo sát, so sánh và đánh giá hiệu quả của một số công cụ phổ biến hiện nay như Unipacker, PEiD, Detect It Easy (DIE) và ClamAV trong việc phát hiện và giải nén các tệp thực thi Windows bị đóng gói. Đồng thời, Khóa luận cũng phân tích tác động của quá trình giải nén đến hiệu suất của các mô

hình học máy trong nhiệm vụ phân loại tệp là lành tính hay độc hại. Qua đó, hướng đến mục tiêu làm rõ liệu quá trình giải nén có thực sự cải thiện độ chính xác của mô hình phát hiện phần mềm độc hại hay không, từ đó đề xuất hướng ứng dụng phù hợp trong các hệ thống phân tích tĩnh dựa trên mô hình học máy.

## 1.4 Mục tiêu, đối tượng và phạm vi nghiên cứu

### 1.4.1 Mục tiêu nghiên cứu

Mục tiêu nghiên cứu chính của Khóa luận là:

- So sánh hiệu suất của các công cụ Unipacker, PEiD, Detect It Easy (DIE) và Clamav trong việc phát hiện và giải nén các tệp thực thi Windows bị đóng gói.
- Phân tích tác động của quá trình giải nén đến hiệu suất của các mô hình học máy trong nhiệm vụ phân loại tệp là lành tính hay độc hại. Qua đó, hướng đến mục tiêu làm rõ liệu quá trình giải nén có thực sự cải thiện độ chính xác của mô hình phát hiện phần mềm độc hại hay không.

### 1.4.2 Đối tượng nghiên cứu

Đối tượng nghiên cứu chính của Khóa luận là:

- Các tệp thực thi Windows (Portable Executable - PE): đây là loại định dạng phổ biến và cũng thường bị khai thác trong các cuộc tấn công mạng.
- Các trình đóng gói (packer): thường được hiểu là các công cụ nén tệp có khả năng giải nén tự động khi tệp được thực thi, đặc biệt là trong môi trường runtime (runtime packer). Các công cụ nhóm sẽ giới thiệu bao gồm: UPX, ASPack, MPRESS, PECompact FSG, MEW và PETITE.
- Cùng với đó là các công cụ được sử dụng để phát hiện và giải nén tệp tin như: Unipacker, ClamAV, Detect It Easy và PEiD.
- Cũng như các mô hình học máy phục vụ cho việc phát hiện phần mềm độc hại (malware detection).

### 1.4.3 Phạm vi nghiên cứu

Khóa luận này chỉ giới hạn trong việc khảo sát và đánh giá hiệu quả của bốn công cụ phổ biến trong việc phát hiện và giải nén các tệp thực thi Windows bị nén, bao

gồm: Unipacker, PEiD, Detect It Easy (DIE) và ClamAV và tập trung vào tập tin PE trên hệ điều hành Windows, không mở rộng sang các hệ điều hành hoặc định dạng tập tin khác. Ngoài ra, chỉ dừng lại ở mức sử dụng các công cụ này để kiểm tra khả năng hoạt động thực tế, bao gồm các tiêu chí như: khả năng phát hiện loại packer, khả năng giải nén thành công, tốc độ xử lý... chưa đi sâu vào viết hoặc cải tiến công cụ. Sau khi thực hiện giải nén, các tập tin được xử lý sẽ tiếp tục được đưa vào các mô hình học máy hiện hành để kiểm tra mức độ ảnh hưởng của việc đóng gói (và giải nén) đến hiệu quả phát hiện mã độc.

# Chương 2

## CƠ SỞ LÝ THUYẾT

### 2.1 Tập tin thực thi Windows

Tập thực thi Windows (Windows Executable) là các tệp chứa mã lệnh và dữ liệu cần thiết để hệ điều hành Windows nạp và thực thi một chương trình. Trong đó, file Portable Executable (PE) là một định dạng tiêu chuẩn được Windows sử dụng. Hãy cùng tìm hiểu một file PE sẽ có những gì.

#### 2.1.1 Portable Executable – PE là gì?

File **Portable Executable (PE)** là định dạng tệp nhị phân gốc của hệ điều hành Windows. Nó được thiết kế để lưu trữ các chương trình thực thi (.exe), thư viện liên kết động (.dll), kernel modules (.srv), và các loại tệp nhị phân khác. Định dạng này có tính độc lập nền tảng, nghĩa là nó sẽ cho phép các tệp PE hoạt động trên bất kỳ máy Windows nào, miễn là máy đó chạy đúng phiên bản hệ điều hành và kiến trúc bộ xử lý (x86 hoặc x64) mà tệp được biên dịch.

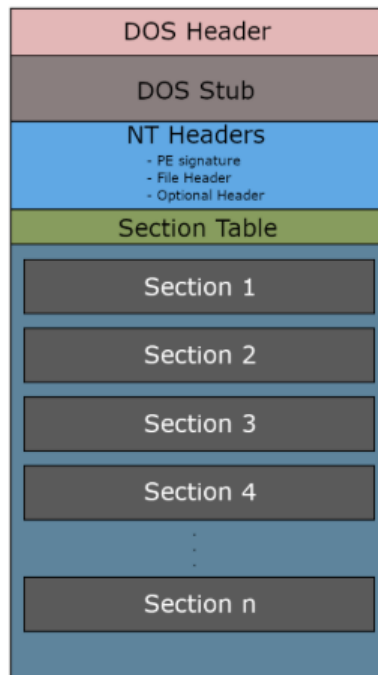
#### 2.1.2 Cấu trúc file PE

Định dạng file Portable Executable (PE) bao gồm nhiều thành phần và mỗi thành phần có một mục đích cụ thể. Các thành phần này được thể hiện trong Hình 2.1.

##### DOS Header

Mỗi tệp PE đều bắt đầu với một cấu trúc dài 64 byte gọi là DOS Header. Thành phần này giúp tệp PE được nhận dạng như một tệp thực thi MS-DOS. Mặc dù hệ điều hành Windows hiện đại không còn phụ thuộc vào MS-DOS, nhưng phần này vẫn được giữ lại để đảm bảo tính tương thích ngược và hỗ trợ các công cụ phân tích, đọc tệp PE.





**Hình 2.1:** Cấu trúc một file PE

DOS Header bao gồm các trường dữ liệu quan trọng, trong đó phổ biến nhất là:

- **e\_magic** (2 byte): là magic number xác định tệp thuộc định dạng DOS.
- **e\_lfanew** (4 byte): chứa offset đến PE Header. Đây là trường quan trọng nhất trong DOS Header, giúp hệ điều hành hoặc công cụ xử lý nhảy đến vị trí bắt đầu của PE Header trong tệp.

Ngoài ra, DOS Header còn chứa các trường khác cung cấp thông tin về định dạng và cách thực thi tệp trong môi trường MS-DOS, nhưng phần lớn chúng không được sử dụng trên hệ thống Windows hiện đại.

### DOS Stub

DOS Stub được thiết kế để hoạt động trên các hệ thống MS-DOS và có nhiệm vụ xử lý tình huống khi tệp PE được chạy trong môi trường không hỗ trợ định dạng PE, chẳng hạn như MS-DOS hoặc các hệ điều hành cũ không phải Windows. Khi chương trình được chạy trong chế độ DOS, DOS stub sẽ hiển thị một thông báo lỗi: *"This program cannot be run in DOS mode"* (Chương trình này không thể chạy trong chế độ DOS).

## NT Header

NT Header là một thành phần quan trọng trong cấu trúc tệp PE, chứa các thông tin cần thiết để hệ điều hành Windows có thể nạp và thực thi tệp. NT Header nằm ngay sau DOS Stub và bao gồm ba phần chính:

- **PE Signature:** là một giá trị 4 byte xác định tệp là tệp Portable Executable. Giá trị của PE Signature là 0x00004550 ("PE"), giúp phân biệt tệp PE với các định dạng tệp khác.
- **File Header:** một tiêu đề COFF tiêu chuẩn, cung cấp thông tin cơ bản về tệp PE để hệ điều hành hiểu cách xử lý tệp. PE Header có các trường quan trọng như:
  - Machine: Loại kiến trúc bộ xử lý (ví dụ: x86, x64)
  - NumberOfSections: Số lượng phần (sections) trong tệp.
  - TimeDateStamp: Thời gian tệp được tạo.
  - SizeOfOptionalHeader: Kích thước của Optional Header.
  - Characteristics: Các đặc điểm của tệp (ví dụ: tệp là thực thi, thư viện liên kết động, v.v.).
- **Optional Header:** mặc dù được gọi là "Optional Header" (tiêu đề tùy chọn) nhưng phần này lại bắt buộc đối với các tệp hình ảnh (Image Only) (như tệp .exe). Đây là phần quan trọng nhất của NT Header. Nó có các trường quan trọng như:
  - AddressOfEntryPoint: Điểm bắt đầu thực thi của tệp.
  - ImageBase: Địa chỉ cơ sở nơi hình ảnh sẽ được nạp vào bộ nhớ.
  - SectionAlignment: Cách các phần được căn chỉnh trong bộ nhớ.
  - Subsystem: Loại môi trường yêu cầu để chạy tệp (ví dụ: GUI hoặc console).
  - SizeOfImage: Tổng kích thước của tệp khi được nạp vào bộ nhớ

NT Header luôn bắt đầu tại offset được chỉ định bởi trường "e\_lfanew" trong DOS Header để đảm bảo rằng hệ điều hành có thể dễ dàng định vị NT Header dù kích thước của DOS Header hoặc DOS Stub thay đổi.

## Section Table

Section Table là một phần quan trọng của tệp PE, nằm ngay sau Optional Header trong cấu trúc của NT Header. Section Table là một mảng các mục, mỗi mục tương ứng với một Section Header. Mỗi Section Header cung cấp thông tin chi tiết về một phần (section) trong tệp PE, chẳng hạn như vị trí, kích thước, và các thuộc tính của phần đó.

Khi hệ điều hành nạp tệp PE vào bộ nhớ, nó đọc Section Table để xác định cách chia tệp thành các phần như mã thực thi, dữ liệu và tài nguyên,... Dựa trên thông tin này, các phần được ánh xạ vào bộ nhớ với địa chỉ và quyền truy cập tương ứng (đọc, ghi, thực thi) nhằm đảm bảo chương trình có thể sử dụng tài nguyên và dữ liệu đúng cách khi thực thi. Các phần như .rsrc sẽ được xử lý đặc biệt để quản lý tài nguyên chương trình như biểu tượng, chuỗi và giao diện, giúp tài nguyên được sử dụng đúng, hiệu quả và không bị sai sót.

*Cấu trúc của Section Header:* Mỗi Section Header trong Section Table chứa các trường dữ liệu quan trọng, bao gồm:

- Name (8 byte): Tên của phần, thường được đặt ngắn gọn như .text, .data, hoặc .rdata.
- VirtualSize (4 byte): Kích thước thực sự của phần khi được nạp vào bộ nhớ.
- VirtualAddress (4 byte): Địa chỉ tương đối của phần trong không gian địa chỉ ảo.
- SizeOfRawData (4 byte): Kích thước của phần trong tệp PE.
- PointerToRawData (4 byte): Offset (độ dời) từ đầu tệp đến dữ liệu thực tế của phần này.
- Characteristics (4 byte): Thuộc tính của phần, chẳng hạn như có thể thực thi, chỉ đọc, hoặc ghi được.

## Section

Và cuối cùng, trong file PE, Section là một phần cơ bản và quan trọng chứa các dữ liệu thực thi hoặc các tài nguyên cần thiết để chương trình hoạt động. Mỗi phần trong tệp PE có mục đích riêng biệt, từ mã thực thi, dữ liệu chương trình cho đến tài nguyên và thông tin gỡ lỗi. Các phần này được quản lý bởi hệ điều hành thông qua Section Table, nơi lưu trữ các thông tin mô tả về từng phần.

### a. Các loại Section phổ biến trong tệp PE

- **.text:** Đây là phần chứa mã thực thi của chương trình. Khi tệp PE được nạp vào bộ nhớ, phần này sẽ được ánh xạ vào vùng bộ nhớ có thể thực thi. ".text" sẽ chứa các chỉ thị mà CPU sẽ thực hiện khi chương trình chạy.
- **.data:** Phần này chứa dữ liệu toàn cục mà chương trình sử dụng và có thể thay đổi trong suốt quá trình thực thi. Các biến toàn cục, dữ liệu cần lưu trữ tạm thời trong khi chương trình đang chạy thường được lưu trữ trong phần ".data".

- **.rdata:** Chứa các dữ liệu chỉ đọc, chẳng hạn như chuỗi văn bản hằng số hoặc thông tin cấu hình mà chương trình không cần thay đổi trong suốt quá trình thực thi.
- **.rsrc:** Đây là phần chứa các tài nguyên mà chương trình cần, chẳng hạn như biểu tượng, đồ họa, văn bản giao diện người dùng, hoặc các tệp nhúng khác. “.rsrc” giúp chương trình truy xuất và sử dụng các tài nguyên này khi cần thiết.
- **.reloc:** Chứa thông tin về các địa chỉ cần được điều chỉnh nếu tệp PE được nạp vào một vị trí bộ nhớ không phải là vị trí mặc định, giúp chương trình hoạt động trên các địa chỉ bộ nhớ khác nhau mà không gặp sự cố.

b. *Vai trò của Section*

- **Chứa mã thực thi:** Phần “.text” chứa mã máy mà hệ điều hành sẽ thực thi khi chương trình được khởi động.
- **Chứa dữ liệu và tài nguyên:** Các phần như .data và .rsrc lưu trữ dữ liệu toàn cục và tài nguyên cần thiết cho chương trình, giúp chương trình dễ dàng truy xuất và sử dụng tài nguyên trong quá trình thực thi.
- **Tăng tính mở rộng:** Các phần như “.reloc” và “.rsrc” giúp chương trình có thể hoạt động linh hoạt hơn trong các môi trường khác nhau, ví dụ, khi được nạp vào các địa chỉ bộ nhớ khác nhau.
- **Quản lý bộ nhớ:** Các thuộc tính trong các Section Header giúp hệ điều hành quyết định quyền truy cập của từng phần, từ đó tăng tính bảo mật và hiệu quả trong việc quản lý bộ nhớ.

## 2.2 Trình đóng gói

### 2.2.1 Trình đóng gói (Packer) là gì?

Trình đóng gói (Packer) là công cụ nén tệp thực thi, cho phép tệp tự động được giải nén trong bộ nhớ khi chạy, còn gọi là "nén thực thi" (executable compression), đặc biệt là trong môi trường runtime (runtime packer).[4]

Mục đích ban đầu của kỹ thuật này là giảm kích thước tệp để tiết kiệm không gian lưu trữ, băng thông mạng và tránh yêu cầu người dùng phải giải nén thủ công. Trong giai đoạn internet còn chậm và bộ nhớ hạn chế, việc sử dụng packer rất phổ biến và hữu ích. Tuy nhiên, với sự phát triển của công nghệ lưu trữ và mạng tốc độ cao, lợi ích về dung lượng không còn quá quan trọng. Ngày nay, khi gặp các tệp thực thi đã

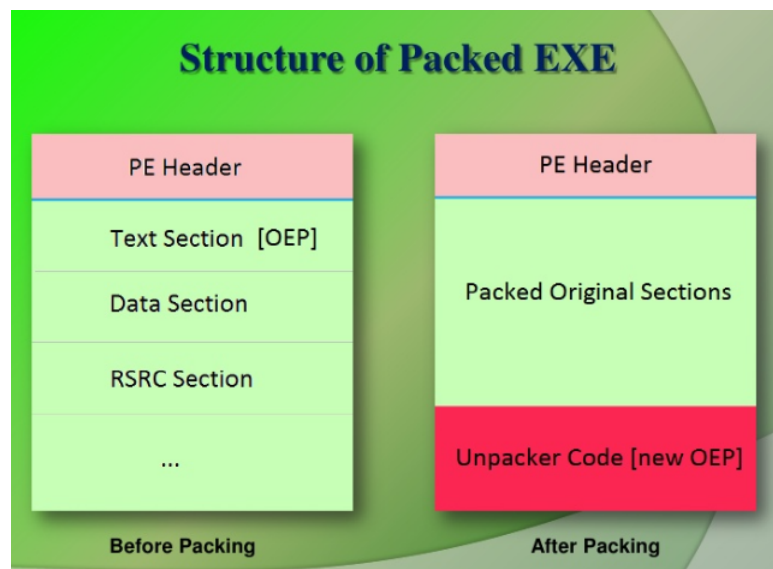
được đóng gói, người ta thường nghi ngờ rằng mục đích chính là để che giấu mã độc hoặc bảo vệ mã nguồn.

Việc sử dụng Packer là một lợi thế đặc biệt khi muốn ẩn mã độc (payload), giúp cho việc dịch ngược mã trở nên khó khăn hơn, làm tăng độ phức tạp khi nghiên cứu mã nguồn, giúp chúng dễ dàng vượt qua các công cụ phân tích và phát hiện phần mềm độc hại.

Nhờ sự hỗ trợ của các kỹ thuật này, mã độc có thể tồn tại lâu hơn trên hệ thống bị xâm nhập mà không bị phát hiện, gây khó khăn trong việc phát hiện và loại bỏ triệt để.

### 2.2.2 Cách đóng gói tập tin

Về bản chất, các trình đóng gói hiện nay được xem như những công cụ được sử dụng để che giấu tập tin độc hại. Các trình đóng gói này có thể mã hóa, nén hoặc đơn giản là thay đổi định dạng của tệp phần mềm độc hại để làm cho nó trông giống như một thứ hoàn toàn khác. Quá trình nén hoặc mã hóa của trình đóng gói đưa tập tin từ mã gốc của nó sang trạng thái mới bằng cách sử dụng các kỹ thuật làm rối "thử và đúng" (tried-and-true obfuscation).



**Hình 2.2:** Cấu trúc một tệp thực thi Windows đã bị đóng gói

Vậy kỹ thuật làm rối *Tried-and-True Obfuscation* là gì?

Kỹ thuật làm rối là một phần quan trọng trong hoạt động của trình đóng gói, được sử dụng để che giấu mã độc và gây khó khăn cho việc phân tích hoặc phát hiện. "Tried-and-True" ở đây ám chỉ việc áp dụng những phương pháp đã được kiểm chứng qua thời gian và chứng minh hiệu quả trong việc làm phức tạp quá trình phân tích mã. Dưới đây là một số kỹ thuật nổi bật được trình đóng gói sử dụng để làm rối mã độc:

### Xáo trộn mã (Code Scrambling)

Kỹ thuật xáo trộn mã là quá trình thay đổi trật tự các lệnh hoặc thêm các đoạn mã thừa vào mã nguồn ban đầu, nhằm tạo ra sự rối rắm, bao gồm một số kỹ thuật như:

- **Thay đổi trật tự lệnh:** Các lệnh quan trọng của mã độc có thể được sắp xếp lại một cách phi logic hoặc được lồng ghép với các lệnh không liên quan.
- **Thêm mã thừa:** Chèn các đoạn mã không có chức năng thực tế, ví dụ như vòng lặp vô dụng hoặc các lệnh không hoạt động để tăng độ dài mã nguồn, khiến các nhà phân tích phải mất thời gian loại bỏ chúng để tìm ra mã thực sự quan trọng.

### Sử dụng mã hóa phức tạp (Advanced Encryption)

Mã hóa là một trong những kỹ thuật mạnh mẽ nhất được sử dụng để làm rối mã độc. Hiện nay có rất nhiều cách để thực hiện mã hóa một chương trình, tuy nhiên, việc sử dụng những loại mã hóa đơn giản thường không khó để có thể giải mã. Vì vậy, một số cách mã hóa phức tạp ra đời, không chỉ làm rối mã độc mà còn đảm bảo rằng ngay cả khi tệp bị phát hiện, việc phân tích nội dung bên trong cũng trở nên rất khó khăn.

- **Mã hóa nhiều lớp:** Thay vì chỉ mã hóa một lần, mã độc có thể được mã hóa nhiều lớp, mỗi lớp sử dụng một thuật toán hoặc khóa mã hóa khác nhau.
- **Thuật toán khó phá giải:** Sử dụng các thuật toán mã hóa tiên tiến, phức tạp để làm cho việc giải mã thủ công trở nên gần như không thể nếu không có khóa giải mã chính xác.

### Tự sửa đổi mã (Self-Modifying Code)

Tự sửa đổi mã là một kỹ thuật độc đáo và hiệu quả trong việc làm rối, nơi mã độc có khả năng thay đổi chính nó như thay đổi cấu trúc hoặc nội dung trong quá trình thực thi để tránh bị nhận diện. Mã có thể tự giải mã, che giấu lệnh quan trọng hoặc thay đổi hành vi tùy theo môi trường, chẳng hạn như không hoạt động trên máy ảo để tránh bị phân tích.

Bằng cách tự sửa đổi, mã độc có thể vượt qua các công cụ phân tích tĩnh (static analysis) thường chỉ hoạt động trên mã cố định, buộc các nhà phân tích phải sử dụng các phương pháp phân tích động (dynamic analysis) phức tạp hơn. Nhờ đó, mã độc trở nên linh hoạt và khó bị phát hiện, làm tăng đáng kể độ khó trong việc nghiên cứu và phân tích.

### **Kết hợp các kỹ thuật để tăng hiệu quả**

Thông thường, các tác giả mã độc sẽ không chỉ sử dụng một kỹ thuật duy nhất mà kết hợp nhiều kỹ thuật làm rối để đạt hiệu quả tối ưu. Ví dụ: một mã độc có thể được mã hóa nhiều lớp, sau đó thêm các đoạn mã thừa và sử dụng mã tự sửa đổi khi thực thi.

Việc kết hợp nhiều kỹ thuật khiến cho quá trình phân tích mã trở nên khó khăn và tốn kém hơn, đồng thời làm giảm hiệu quả của các hệ thống bảo mật dựa trên mẫu nhận diện (signature-based detection).

### **2.2.3 Các công cụ đóng gói phổ biến**

#### **UPX**

UPX (Ultimate Packer for Executables) là một công cụ nén tệp thực thi mạnh mẽ và tiên tiến. Với khả năng giảm kích thước tệp của các chương trình và thư viện DLL xuống từ 50% đến 70%, UPX mang lại nhiều lợi ích như tiết kiệm không gian lưu trữ, giảm thời gian tải qua mạng, rút ngắn thời gian tải xuống, và tối ưu hóa chi phí phân phối cũng như lưu trữ. [10]

Điểm nổi bật của UPX là các chương trình và thư viện được nén bởi công cụ này vẫn hoàn toàn tự chứa, không cần phần mềm bổ sung để chạy và hoạt động chính xác như phiên bản gốc. Đặc biệt, việc nén không gây ra bất kỳ tác động nào đến hiệu suất runtime hoặc bộ nhớ đối với hầu hết các định dạng mà UPX hỗ trợ. UPX hỗ trợ nhiều định dạng thực thi khác nhau, bao gồm các chương trình và thư viện DLL trên hệ điều hành Windows, ứng dụng trên macOS, và tệp thực thi trên Linux. Điều này giúp UPX trở thành một công cụ linh hoạt, phù hợp với nhiều nền tảng và nhu cầu khác nhau.

- **Các ưu điểm nổi bật của UPX:**

- **Bảo mật:** UPX là phần mềm mã nguồn mở được công khai và tài liệu hóa trong nhiều năm, do đó các phần mềm bảo mật hoặc phần mềm diệt virus đáng tin cậy có thể kiểm tra bên trong các ứng dụng được nén bởi UPX để xác minh tính an toàn.



**Hình 2.3:** Công cụ nén tệp tin UPX

- Tỷ lệ nén xuất sắc: UPX thường nén hiệu quả hơn so với định dạng Zip. Công cụ này giúp giảm kích thước các tệp phân phối một cách đáng kể, tối ưu hóa dung lượng lưu trữ và thời gian tải xuống.
- Giải nén cực nhanh: Tốc độ giải nén đạt hơn 500 MB/giây trên bất kỳ máy tính hiện đại nào.
- Không tiêu tốn bộ nhớ: Các tệp thực thi nén bởi UPX được giải nén trực tiếp tại chỗ, không gây bất kỳ ảnh hưởng nào đến dung lượng bộ nhớ.
- Tính phổ quát: UPX hỗ trợ nhiều định dạng thực thi khác nhau, bao gồm các chương trình và thư viện DLL trên Windows, ứng dụng trên macOS, và tệp thực thi trên Linux.
- Tính di động: UPX được viết bằng ngôn ngữ C++ có khả năng tương thích tốt giữa các kiến trúc máy tính, không bị ảnh hưởng bởi sự khác biệt về endian (cách lưu trữ byte).

Với những tính năng vượt trội trên, không khó hiểu tại sao UPX được gọi là "Ultimate Packer for Executables" (Trình đóng gói tối ưu cho các tệp thực thi). Và đây cũng là một trong những packer được sử dụng phổ biến nhất hiện nay.

• **Một số nhược điểm của UPX:**

- Giới hạn kích thước tệp: UPX hoạt động tốt với các tệp thực thi nhỏ và vừa, nhưng khi nén các tệp lớn, hiệu quả nén có thể giảm đi, hoặc thậm chí gây ra lỗi nếu hệ thống không đủ tài nguyên để xử lý.
- Giới hạn trong môi trường nhúng: Trên các thiết bị nhúng hoặc hệ thống có tài nguyên hạn chế, việc nén/giải nén liên tục có thể gây ra vấn đề hiệu năng hoặc làm tăng độ trễ.



- Tương thích hạn chế: Một số phần mềm yêu cầu kiểm tra toàn vẹn (integrity check) có thể không hoạt động chính xác nếu sử dụng UPX. Ngoài ra, các chương trình sử dụng các kỹ thuật nén hoặc mã hóa đặc biệt khác có thể gặp xung đột khi kết hợp với UPX.

**Đặc trưng của packer:** Trong phần "Sections" hoặc "Headers", có thể tìm kiếm các dấu hiệu của UPX, chẳng hạn như một section có tên "UPX" như "UPX0", "UPX1 hoặc "UPX2".

### ASPack

ASPack là một giải pháp tiên tiến dành cho việc nén tệp thực thi Win32 (EXE) và bảo vệ chúng trước các hoạt động dịch ngược mã không chuyên. Công cụ này giúp giảm kích thước chương trình và thư viện Windows lên đến 70%, với tỷ lệ nén cao hơn tiêu chuẩn ZIP từ 10 - 20%, không chỉ tối ưu hóa dung lượng lưu trữ mà còn giảm thời gian tải qua mạng và thời gian tải xuống từ internet một cách đáng kể. [2]



Hình 2.4: Công cụ nén tệp tin ASPack

Ngoài khả năng nén mạnh mẽ, ASPack còn cung cấp lớp bảo mật bổ sung, bảo vệ chương trình và ứng dụng khỏi các phân tích trái phép, các công cụ gỡ lỗi và dịch ngược. Các chương trình được nén bằng ASPack vẫn duy trì đầy đủ chức năng và hiệu suất, hoạt động như trước mà không gây bất kỳ ảnh hưởng tiêu cực nào đến trải nghiệm người dùng. Công cụ này hoạt động độc lập và không yêu cầu phần mềm bổ sung, mang lại sự linh hoạt cho người dùng trong các dự án với yêu cầu đặc biệt.

- **Các ưu điểm nổi bật của ASPack:**

- Giảm kích thước tệp thực thi lên đến 70%, giúp tối ưu hóa dung lượng lưu trữ. Giảm thời gian tải qua mạng và thời gian tải xuống từ internet, nâng cao hiệu suất.
- Các ứng dụng Windows nhúng yêu cầu ít không gian lưu trữ hơn đáng kể.
- Bảo vệ tài nguyên và mã nguồn khỏi việc xem trộm, công cụ phân tích mã (disassemblers) và dịch ngược (decompilers).

- Tương thích với các tệp thực thi được tạo ra từ Microsoft Visual C++, Visual Basic, Embarcadero Delphi, C++ Builder và các trình biên dịch Win32 khác.
- **Một số nhược điểm của ASPack:**
  - Giới hạn đối với mã máy: ASPack chủ yếu tối ưu cho các tệp thực thi Win32 và có thể không tương thích hoàn toàn với các mã máy phức tạp.
  - Chỉ hỗ trợ các phiên bản Windows cũ: Mặc dù công cụ này hoạt động tốt trên các phiên bản Windows phổ biến như Windows XP, Vista, và Windows 7, nhưng nó có thể gặp vấn đề tương thích với các phiên bản Windows mới hơn như Windows 10 và Windows 11.
  - Hiệu suất giảm khi nén một số tệp lớn: Trong một số trường hợp, việc nén các tệp EXE lớn có thể gây giảm hiệu suất, đặc biệt khi giải nén chương trình, mặc dù thuật toán giải nén của ASPack được tối ưu hóa, nhưng quá trình này vẫn có thể tốn thời gian trong một số tình huống đặc biệt.

ASPack thích hợp nhất cho các ứng dụng Windows nhẹ, cần tối ưu hóa dung lượng và bảo mật mã nguồn, đồng thời hỗ trợ tốt các công cụ phát triển phần mềm truyền thống. Tuy ASPack là một công cụ mạnh mẽ và hiệu quả, nhưng nó cũng có những nhược điểm cần được cân nhắc khi quyết định sử dụng cho các dự án phát triển phần mềm hiện đại.

**Đặc trưng của packer:** Trong phần "Sections" hoặc "Headers", có thể tìm kiếm các dấu hiệu của ASPack, chẳng hạn như một section có tên ".aspack" hoặc ".adata". Ngoài ra, Entry Point thường không nằm trong section ".text" như thông thường mà nằm trong ".aspack", và entropy của các section này thường rất cao (trên 7.5), cho thấy dữ liệu đã bị nén hoặc mã hóa.

## MPRESS

MPRESS là một công cụ miễn phí dành cho việc nén các tệp thực thi, hỗ trợ nhiều định dạng như PE32, PE32+, .NET, và MAC-DARWIN[8]. Với khả năng giảm kích thước chương trình và thư viện, MPRESS giúp tối ưu hóa dung lượng tệp, đồng thời cải thiện thời gian khởi động khi ứng dụng được tải từ các thiết bị lưu trữ chậm hoặc qua mạng. Đây là giải pháp lý tưởng để cải thiện hiệu quả hoạt động của các ứng dụng.

Một trong những tính năng nổi bật của MPRESS là công nghệ giải nén tại chỗ (in-place decompression), cho phép giải nén các tệp thực thi mà không gây áp lực lên



**Hình 2.5:** Công nghệ nén tệp tin MPRESS

bộ nhớ hoặc ảnh hưởng tiêu cực đến hiệu suất. Ngoài ra, MPRESS sử dụng thư viện nén dữ liệu LZMAT với tốc độ cực kỳ nhanh. Kể từ phiên bản 2.00, phát hành ngày 21 tháng 3 năm 2009, MPRESS còn hỗ trợ nén tùy chọn bằng thuật toán LZMA, mang lại hiệu quả nén vượt trội hơn. Đây là công cụ hoàn toàn miễn phí, không yêu cầu bất kỳ chi phí bản quyền nào, phù hợp với mọi đối tượng sử dụng.

- **Các ưu điểm nổi bật của MPRESS:**

- Nén nâng cao cho nhiều loại tệp thực thi: Hỗ trợ nén các tệp .NET (any-CPU, x86, AMD64, IA64 EXE) với khả năng tương thích Microsoft Framework 1.1/2.0/3.0/4.0 mà không yêu cầu cài đặt .NET Framework. Đồng thời, cung cấp tính năng nén tối ưu cho các tệp PE32/PE32+ (EXE, DLL, OCX, v.v.) và ứng dụng macOS như mac-darwin-i386, mac-darwin-x86-64, mac-darwin-ub.
- Công nghệ nén hiệu quả và bảo mật: Sử dụng thuật toán LZMA tùy chọn, nén mã chương trình, dữ liệu và tài nguyên một cách tối ưu.
- Giải nén nhanh và hiệu quả: Với tốc độ giải nén lên đến 210 MB/giây trên AMD 2500+, MPRESS sử dụng công nghệ giải nén tại chỗ (in-place decompression), không tạo thêm áp lực bộ nhớ hay gây suy giảm hiệu suất.
- Hoạt động linh hoạt và toàn diện: Hỗ trợ UNICODE, hoạt động hoàn toàn minh bạch và tự chứa. Giao diện dòng lệnh cho phép tích hợp dễ dàng vào các tệp lệnh batch hoặc tệp makefile để tự động hóa quá trình nén.

- **Một số nhược điểm của MPRESS:**

- Tương thích hệ điều hành hạn chế: Mặc dù hỗ trợ nhiều phiên bản Windows từ 9x đến Windows 8, MPRESS không được cập nhật để tương thích tốt với các hệ điều hành mới hơn như Windows 10, Windows 11 hoặc các môi trường hiện đại hơn.
- Chỉ bảo vệ ở mức cơ bản: MPRESS cung cấp một số tính năng bảo vệ chống dịch ngược (reverse engineering), nhưng không hiệu quả trước các hacker chuyên nghiệp hoặc các công cụ mạnh hơn dành cho việc phân tích mã.

- Không có bản cập nhật thường xuyên: Phiên bản mới nhất (2.19) đã không được cập nhật trong nhiều năm, dẫn đến việc thiếu hỗ trợ cho các công nghệ hoặc định dạng mới.

MPRESS phù hợp nhất để nén các tệp thực thi và thư viện nhỏ, giảm dung lượng và tăng tốc khởi động trên các hệ thống cũ hoặc tài nguyên hạn chế. Nó cũng là lựa chọn tốt để bảo vệ cơ bản chống dịch ngược mà không phát sinh chi phí.

**Đặc trưng của packer:** Trong phần "Sections" hoặc "Headers", có thể tìm kiếm các dấu hiệu của MPRESS, chẳng hạn như một section có tên ".MPRESS1" hoặc ".MPRESS2"

## FSG

FSG (Fast Small Good) là một công cụ nén tệp thực thi nhỏ gọn và hiệu quả, được thiết kế để tối ưu hóa kích thước các tệp PE (Portable Executable) trên hệ điều hành Windows, được phát triển bởi Bartosz Gorski và xuất hiện lần đầu vào khoảng năm 2002. Với mục tiêu giảm thiểu kích thước tệp mà vẫn duy trì hiệu suất runtime, FSG đã được sử dụng rộng rãi trong các ứng dụng yêu cầu phân phối gọn nhẹ và nhanh chóng. Các tệp thực thi nén bằng FSG thường có tốc độ giải nén nhanh và không gây ảnh hưởng đáng kể đến hiệu suất runtime của ứng dụng. Với thiết kế tối giản, FSG dễ sử dụng và không yêu cầu cấu hình phức tạp, phù hợp với cả người dùng cơ bản và chuyên gia. Công cụ này tương thích tốt với nhiều phiên bản hệ điều hành Windows, hỗ trợ các tệp thực thi được tạo từ các ngôn ngữ lập trình phổ biến như C++, Delphi, và VB.



Hình 2.6: Công nghệ nén tệp tin FSG

- **Các ưu điểm nổi bật của FSG:**

- Dung lượng nhỏ: FSG không chỉ nén các tệp thực thi, mà chính công cụ này cũng có kích thước rất nhỏ, dễ dàng tích hợp vào các quy trình phát triển phần mềm.
- Giải nén nhanh: Tốc độ giải nén cao giúp tối ưu hóa thời gian khởi chạy chương trình, đặc biệt phù hợp với các ứng dụng yêu cầu tốc độ.

- Hiệu suất nén tốt: Mặc dù không phải là công cụ nén có tỷ lệ cao nhất, FSG vẫn đạt hiệu suất nén đáng kể, đặc biệt trên các tệp PE nhỏ hoặc vừa.
  - Đơn giản hóa việc phân phối: Nhờ khả năng giảm kích thước, FSG giúp tiết kiệm chi phí lưu trữ và tối ưu hóa việc phân phối phần mềm qua mạng.
- **Một số nhược điểm của FSG:**
- Giới hạn định dạng: FSG chủ yếu hỗ trợ các tệp PE trên Windows, không thể nén các định dạng hoặc kiến trúc khác như 64-bit (PE+).
  - Bị nhận diện nhầm: Do từng bị sử dụng bởi một số phần mềm độc hại, các tệp nén bởi FSG đôi khi bị phần mềm diệt virus đánh dấu nhầm là nguy hiểm.
  - Tính năng hạn chế: So với các công cụ nén tiên tiến như UPX, FSG thiếu các tùy chọn mở rộng hoặc hỗ trợ plug-in.
  - Tương thích không hoàn hảo: Một số ứng dụng sử dụng tính năng kiểm tra toàn vẹn (integrity check) có thể không hoạt động đúng khi được nén bởi FSG.

FSG là một công cụ nén tệp thực thi đơn giản, hiệu quả, và dễ sử dụng, phù hợp cho các ứng dụng nhỏ hoặc vừa cần giảm kích thước để tối ưu hóa việc phân phối. Mặc dù có một số hạn chế so với các công cụ nén hiện đại, FSG vẫn là một lựa chọn đáng cân nhắc trong các trường hợp cần giải pháp nén nhẹ và nhanh chóng.

**Đặc trưng của packer:** Trong phần "Sections" hoặc "Headers", có thể tìm kiếm các dấu hiệu của FSG, chẳng hạn như một section có tên ".FSG!"

## MEW

MEW (Minimal Executable Wrapper) là một công cụ nén tệp thực thi được thiết kế đặc biệt để xử lý các tệp nhỏ, nhưng cũng có thể nén hiệu quả các tệp lớn hơn[6]. Một trong những điểm mạnh của MEW là khả năng nén tệp EXE mà không làm mất tính tự chứa của chương trình, nghĩa là tệp nén vẫn có thể chạy mà không cần phần mềm bổ sung. MEW cũng cung cấp nhiều tùy chọn nén cho người dùng, bao gồm nén tài nguyên, xóa tài nguyên không cần thiết và hỗ trợ thuật toán LZMA để đạt được tỷ lệ nén tốt hơn.

MEW hỗ trợ các hệ điều hành Windows từ phiên bản Windows 95 đến Windows 7, mặc dù không hỗ trợ các hệ điều hành 64-bit, nhưng có thể hoạt động tốt trên các máy tính 32-bit. Công cụ này có thể nén các tệp EXE được viết bằng các ngôn ngữ lập trình phổ biến như C++ và Delphi.



Hình 2.7: Công nghệ nén tệp tin MEW

- **Các ưu điểm nổi bật của MEW:**

- Tỷ lệ nén tốt và tốc độ nén nhanh: MEW giúp giảm kích thước tệp EXE, tối ưu hóa dung lượng lưu trữ và thời gian tải xuống.
- Tính linh hoạt: MEW cung cấp nhiều tùy chọn nén và hỗ trợ thuật toán LZMA, giúp người dùng tối ưu hóa tỷ lệ nén.
- Đảm bảo tính tương thích: Tệp EXE nén bởi MEW vẫn có thể chạy bình thường mà không gặp phải vấn đề tương thích với hệ điều hành Windows.
- Mã nguồn mở: MEW là phần mềm mã nguồn mở, cho phép người dùng tùy chỉnh và sử dụng miễn phí.

- **Một số nhược điểm của MEW:**

- Giao diện người dùng chưa thân thiện: Giao diện của MEW không được tối ưu và thiếu các tài liệu hướng dẫn, điều này có thể gây khó khăn cho người dùng mới.
- Không hỗ trợ hệ điều hành 64-bit: MEW không hỗ trợ các hệ điều hành 64-bit, điều này khiến nó không phù hợp với các nền tảng hiện đại.
- Không có tệp trợ giúp: Ứng dụng không đi kèm với tài liệu hướng dẫn, người dùng phải tìm kiếm trực tuyến để hiểu cách sử dụng.

Với những tính năng nổi bật và khả năng nén hiệu quả, MEW là một công cụ hữu ích cho việc nén tệp EXE. Tuy nhiên, giao diện người dùng và sự hỗ trợ cho hệ điều hành 64-bit vẫn là những vấn đề cần được cải thiện để đạt được sự hoàn thiện hơn.

**Đặc trưng của packer:** Tên các section có thể bị thay đổi hoặc bất thường, ví dụ: .MEW, .pack, hoặc tên rút gọn không liên quan. Khi mở tệp bằng hex editor, chúng ta có thể thấy các chuỗi đặc trưng của MEW, như: MEW hoặc MicroJoin xuất hiện trong phần đầu hoặc cuối của tệp.

## PETITE

PETITE là một công cụ nén tệp thực thi miễn phí dành cho hệ điều hành Win32, bao gồm các phiên bản Windows 95/98/2000/NT/XP/Vista/7. Điểm nổi bật của Petite là khả năng tích hợp tính năng tự kiểm tra virus vào mỗi tệp nén, giúp bảo vệ các tệp thực thi khỏi sự nhiễm độc khi được sử dụng. Công cụ này có cả phiên bản giao diện người dùng (GUI) và dòng lệnh, phù hợp cho cả người dùng phổ thông và nhà phát triển. Petite cho tỷ lệ nén tốt, giúp giảm dung lượng tệp và giữ nguyên khả năng chạy của chương trình sau khi nén.



**Hình 2.8:** Công nghệ nén tệp tin PETITE

- **Các ưu điểm nổi bật của PETITE:**

- Tỷ lệ nén tốt: Petite giúp giảm kích thước tệp thực thi một cách hiệu quả, tối ưu hóa không gian lưu trữ và giảm thời gian tải tệp.
- Tính năng kiểm tra virus: Petite tích hợp khả năng tự kiểm tra virus trong mỗi tệp nén, đảm bảo bảo mật khi thực thi.
- Tính linh hoạt: Phiên bản giao diện người dùng và dòng lệnh của Petite đáp ứng nhu cầu của cả người dùng thông thường và nhà phát triển phần mềm.
- Tính tiện dụng: Petite có thể được sử dụng như một tiện ích mở rộng trong Windows Explorer, giúp người dùng nén tệp dễ dàng mà không cần mở ứng dụng.

- **Một số nhược điểm của PETITE:**

- Giao diện người dùng chưa tối ưu: Mặc dù phiên bản GUI của Petite giúp nén tệp dễ dàng, nhưng giao diện vẫn còn đơn giản và không phải là thân thiện nhất.
- Không hỗ trợ hệ điều hành 64-bit: Petite không hỗ trợ các hệ điều hành 64-bit, điều này khiến nó không phù hợp với nền tảng hiện đại.

- Tốc độ nén không phải lúc nào cũng nhanh: Mặc dù Petite nén hiệu quả, nhưng so với một số công cụ nén khác, tốc độ nén có thể chậm hơn đối với các tệp lớn.

Với khả năng nén hiệu quả và tính năng kiểm tra virus tích hợp, Petite là một công cụ hữu ích cho việc nén các tệp thực thi trên hệ điều hành Win32. Tuy nhiên, giao diện người dùng và sự hỗ trợ cho các hệ điều hành 64-bit cần được cải thiện để đáp ứng tốt hơn nhu cầu của người dùng hiện đại.

**Đặc trưng của packer:** Thường hiển thị tên “PETITE” trong phần nhận diện chữ ký (signature). Có các chuỗi như “Petite v2.xx” hoặc các đoạn mã liên quan đến công cụ nén này ở phần đầu hoặc gần cuối file.

## **XPack**

XPack là một trình đóng gói (packer) dành cho các tệp thực thi trên nền tảng Windows (PE file), với mục tiêu chính là bảo vệ mã nguồn và tăng cường khả năng chống dịch ngược (anti-reverse engineering). Mặc dù không phổ biến rộng rãi như UPX hoặc ASPack, XPack vẫn được các tác giả mã độc sử dụng do khả năng làm rối mã (obfuscation) và ẩn payload tương đối hiệu quả, gây khó khăn cho các công cụ phân tích tĩnh.

- **Các ưu điểm nổi bật của XPack:**

- Tạo ra các tệp thực thi khó phát hiện bởi các trình antivirus thông thường.
- Là một lựa chọn tốt để kiểm tra khả năng phát hiện dựa trên heuristic thay vì chỉ phụ thuộc vào signature.
- Có thể sử dụng để kiểm tra khả năng unpack thủ công hoặc bán tự động.

- **Một số nhược điểm của XPack:**

- Không được hỗ trợ giải nén tốt bởi hầu hết các công cụ unpack tự động.
- Không có giao diện người dùng thân thiện, thường chỉ tồn tại ở dạng bản đã được tích hợp trong các mẫu mã độc hoặc công cụ build mã độc.
- Thiếu tài liệu và không có nguồn tải chính thức – chủ yếu được trích xuất từ thực tế malware.

Trong môi trường nghiên cứu, XPack là một mẫu thử lý tưởng để đánh giá khả năng phát hiện các packer lạ của các công cụ như Detect It Easy hoặc ClamAV với signature mở rộng.



### 2.2.4 Lựa chọn tập dữ liệu dựa trên các trình đóng gói

Trong quá trình khảo sát, nhóm nhận thấy có rất nhiều trình đóng gói (packer) tồn tại trên thị trường, mỗi loại có đặc trưng riêng về kỹ thuật nén, độ phổ biến và mức độ làm rối mã. Tuy nhiên, để đảm bảo sự đa dạng, tính thực tiễn và khả năng triển khai được trong phạm vi khóa luận, nhóm lựa chọn 4 packer tiêu biểu là: UPX, MEW, PETITE và XPack.

Lý do chọn MEW, Petite, UPX, XPack thay vì các packer khác:

- Đảm bảo sự đa dạng trong kỹ thuật đóng gói
  - UPX: là packer phổ biến nhất hiện nay, mã nguồn mở, dễ sử dụng, có khả năng giải nén được bằng nhiều công cụ. Đây là một lựa chọn cơ bản – đại diện cho packer hợp pháp nhưng cũng bị lợi dụng bởi mã độc.
  - MEW và Petite: là packer cổ điển, được dùng phổ biến trong nhiều mẫu mã độc cũ. Các packer này thường dùng kỹ thuật mã hóa đơn giản và làm rối (obfuscation) – phù hợp để kiểm tra khả năng giải nén với các công cụ không hỗ trợ signature mới.
  - XPack: là một packer ít phổ biến hơn, không có nhiều thông tin, đại diện cho nhóm “exotic packers” – giúp kiểm tra khả năng phát hiện không phụ thuộc vào signature của các công cụ.

Việc lựa chọn này đảm bảo sự đa dạng về độ phổ biến và kỹ thuật, từ dễ đến khó, từ phổ biến đến hiếm gặp.

- Các công cụ bị hạn chế
  - Một số packer như FSG, ASPack, PECompact, Mpress mặc dù cũng phổ biến nhưng do lỗi thời nên không được hỗ trợ tốt và khó giải nén.
  - Một phần là do thời lượng thực hiện khóa luận có giới hạn nên một số packer có chức năng tương đồng sẽ bị loại bỏ.

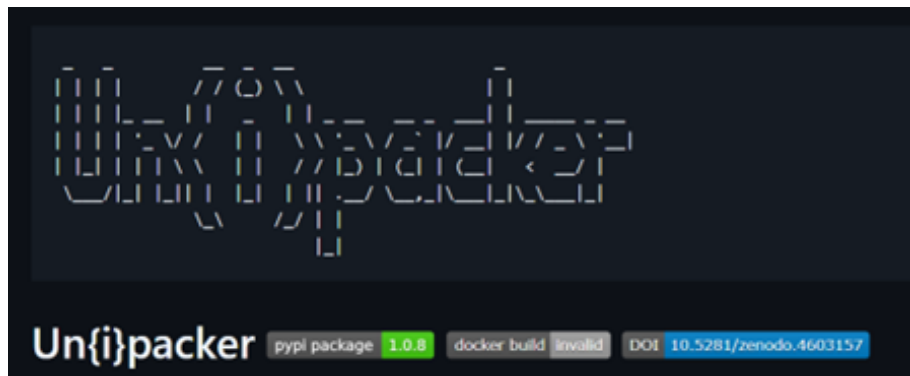
Chọn các packer dễ triển khai, dễ đóng gói hàng loạt và được hỗ trợ tốt sẽ giúp quá trình đánh giá chính xác và khả thi hơn.

Việc chọn 4 packer này không chỉ đảm bảo tính thực tiễn, khả năng triển khai kỹ thuật mà còn tối ưu hóa cho quá trình kiểm thử và đánh giá. Chúng đại diện cho các nhóm packer khác nhau và tạo ra môi trường đánh giá toàn diện nhưng khả thi.

## 2.3 Các công cụ phát hiện và giải nén các tập tin thực thi Windows

### 2.3.1 Unipacker

Unipacker là một công cụ phát hiện và giải nén tệp thực thi Windows bị nén một cách mạnh mẽ, chủ yếu được sử dụng trong việc phân tích bảo mật, đặc biệt khi cần truy xuất và giải nén các tệp thực thi đã được nén bằng các công cụ packer như UPX, MEW hoặc Petite mà khoogn cần phải trực tiếp chạy tệp. Công cụ này giúp phát hiện và giải nén tệp mà không làm mất dữ liệu hoặc ảnh hưởng đến chức năng của chúng. Nó hỗ trợ nhiều định dạng tệp nén và cung cấp các tùy chọn mạnh mẽ để xử lý tệp thực thi một cách hiệu quả.



Hình 2.9: Công cụ Unipacker

Lưu ý: Công cụ này được triển khai trên Linux và chỉ hỗ trợ các tệp thực thi định dạng PE32 (Portable Executable) của Windows và không hỗ trợ các tệp ELF (Executable and Linkable Format)[11].

#### Các packer được hỗ trợ bởi Unipacker

Unipacker có khả năng hỗ trợ đa dạng các trình nén tệp tin hiện tại trên thị trường:

- **ASPack**: Trình packer thương mại cao cấp với tỷ lệ nén cao.
- **FSG**: Phần mềm miễn phí, giải nén nhanh chóng.
- **MEW**: Thiết kế đặc biệt cho các tệp nhị phân nhỏ.
- **MPRESS**: Miễn phí, trình packer phức tạp hơn.

- **PEtite**: Trình packer miễn phí, tương tự như ASPack.
- **UPX**: Trình packer mã nguồn mở, hỗ trợ đa nền tảng.
- **YZPack**: (Không có mô tả cụ thể).

Các packer khác cũng có thể hoạt động nếu Unipacker đã hỗ trợ các hàm API cần thiết. Đối với những packer không được nhận diện, bạn sẽ cần chỉ định thủ công địa chỉ bắt đầu và kết thúc để mô phỏng.

### Cách phát hiện các trình nén tệp tin

Các dấu hiệu nhận dạng sẽ được tìm kiếm theo các quy tắc YARA (được lưu trữ trong file packer\_signatures.yar) và chọn ra packer thích hợp. Một số ví dụ cụ thể về dấu hiệu nhận biết packer thông qua quy tắc YARA được thể hiện trong các Hình 2.10, 2.11, 2.12.

```
rule upx{
  meta:
    description = "UPX packed file"
    date = "2019-01-05"
    strings:
      $upx = "upX" wide ascii
      $upx0 = "UPX0" wide ascii
      $upx1 = "UPX1" wide ascii
      $upx2 = "UPX2" wide ascii
      $upxx = "UPX!" wide ascii

    condition:
      pe32 and (1 of ($upx0, $upx1, $upx2)) or ($upxx or $upx)
}
```

**Hình 2.10:** Các quy tắc yara nhận biết file bị nén bởi UPX của Unipacker

### Quá trình giải nén các tệp tin

Hệ thống giải nén được thiết kế nhằm tự động xác định và xử lý các tệp thực thi Windows bị đóng gói. Quá trình bắt đầu bằng việc nhận diện loại trình đóng gói được sử dụng trong tệp thông qua YARA nêu trên, từ đó lựa chọn phương pháp giải nén phù hợp. Sau khi xác định, hệ thống tiến hành kiểm tra các vùng bộ nhớ hợp lệ, trích xuất dữ liệu từ các phần có chứa mã thực thi, và khôi phục nội dung gốc của tệp. Dữ liệu trong các phần (section) có chứa mã thực thi sẽ được trích xuất và tái cấu trúc lại để khôi phục nội dung ban đầu của tệp trước khi bị đóng gói. Hình 2.13, 2.14 là một trong những trình giải nén của Unipacker.

Cuối cùng, kết quả được ghi ra một tệp đã giải nén, sẵn sàng phục vụ cho việc phân tích sâu hơn như phát hiện mã độc hoặc huấn luyện mô hình học máy.

```
rule petite{
  meta:
    description = "PEtite packed file"
    date = "2019-01-16"
  strings:
    $petite = ".petite"
    $petite2 = "petite"
  condition:
    pe32 and ($petite or $petite2)
}
```

Hình 2.11: Các quy tắc yara nhận biết file bị nén bởi PETITE của unipacker

```
rule mpress{
  meta:
    description = "MPRESS packed file"
    date = "2019-02-28"
  strings:
    $mp = ".MPRESS1"
    $mp2 = ".MPRESS2"
  condition:
    pe32 and ($mp or $mp2)
}
```

Hình 2.12: Các quy tắc yara nhận biết file bị nén bởi MPRESS của Unipacker

### 2.3.2 ClamAV

ClamAV là một công cụ diệt virus mã nguồn mở được phát triển dưới giấy phép GPLv2 và được thiết kế đặc biệt để quét email trên các cổng thư điện tử[9]. Với khả năng phát hiện các loại trojan, virus, phần mềm độc hại và các mối đe dọa khác, ClamAV trở thành một lựa chọn lý tưởng để bảo mật hệ thống.

Và trong ngữ cảnh sử dụng cho các trình đóng gói tệp tin, ClamAV cũng hoàn thành rất xuất sắc trong việc phát hiện các trình đóng gói, tuy nhiên việc giải nén vẫn có một số hạn chế.

Một trong những ưu điểm lớn của ClamAV là hiệu suất cao và tính linh hoạt. Với dịch vụ quét đa luồng, ClamAV có thể xử lý nhiều tác vụ quét đồng thời, giúp tăng tốc quá trình bảo vệ hệ thống mà không làm giảm hiệu suất. Ngoài ra, công cụ này hỗ trợ

```
class UPXUnpacker(AutomaticDefaultUnpacker):

    def __init__(self, sample):
        super().__init__(sample)
        self.name = "UPX"
        self.allowed_sections = []
        self.dumper = ImportRebuilderDump()
        for s in self.secs:
            if s.SizeOfRawData > 0:
                self.allowed_sections += [s.Name]
        self.allowed_addr_ranges = self.get_allowed_addr_ranges()
```

Hình 2.13: Trình giải nén UPX của Unipacker

```
class PETiteUnpacker(AutomaticDefaultUnpacker):

    def __init__(self, sample):
        super().__init__(sample)
        self.name = "PETite"
        ep = sample.opt_header.AddressOfEntryPoint
        for s in self.secs:
            start_addr = s.VirtualAddress
            end_addr = s.VirtualSize + start_addr
            if start_addr <= ep <= end_addr:
                finish = end_addr

        # self.allowed_sections = ['.text']
        # self.allowed_addr_ranges = self.get_allowed_addr_ranges()
        self.allowed_addr_ranges.extend([(ep + self.BASE_ADDR, finish + self.BASE_ADDR)])
        self.dumper = PETiteDump()

    def is_allowed(self, address):
        for chunk in self.sample.allocated_chunks:
            if chunk not in self.allowed_addr_ranges:
                self.allowed_addr_ranges.append(chunk)
        return super().is_allowed(address)
```

Hình 2.14: Trình giải nén PETITE của Unipacker

nhiều định dạng tệp và ngôn ngữ chữ ký, cũng như khả năng giải nén tệp và lưu trữ, cho phép nó phát hiện mối đe dọa trong nhiều loại tệp khác nhau.

### Các packer được hỗ trợ bởi ClamAV

Tương tự như Unipacker, ClamAV cũng hỗ trợ đa dạng các trình nén tệp tin hiện tại trên thị trường. Tuy nhiên, có điểm nổi bật hơn là ClamAV hỗ trợ luôn cho các tệp tin PE 64 bits. Ngoài các công cụ đã giới thiệu ở trên như UPX, ASPack, FSG, Petite, MEW, ClamAV còn hỗ trợ thêm:

- **PeSpin:** Công cụ nén và bảo vệ tệp PE, tạo sự khó khăn cho việc phân tích.
- **NsPack:** Giảm dung lượng mà không ảnh hưởng đến khả năng thực thi.
- **wwpack32:** Công cụ nén PE mạnh mẽ, giữ nguyên tính tương thích và hiệu suất của tệp.

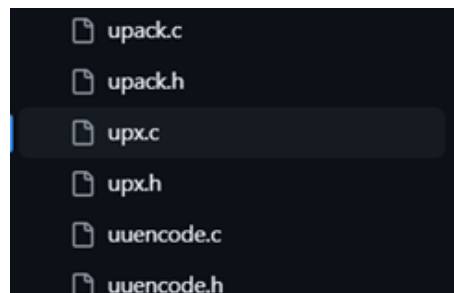


Hình 2.15: Công cụ ClamAV

### Cách phát hiện tệp tin bị nén và cách giải nén

ClamAV sẽ tổng hợp các signature và các dấu hiệu cho thấy một tệp tin bị đóng gói vào các tệp tin khác nhau tương ứng với từng trình đóng gói. Dưới đây là một ví dụ về công cụ UPX.

Để tìm thấy các tệp tin liên quan đến công cụ, chúng ta có thể đến địa chỉ “libclamav” và tìm các file như Hình 2.16.



Hình 2.16: Các file chứa đoạn code liên quan đến UPX trong ClamAV

Trong file “upx.c” sẽ một số dấu hiệu nhận biết tệp tin bị nén bởi UPX như sau:

- Phát hiện "magic bytes" của UPX: Đoạn mã kiểm tra các đặc trưng của file UPX bằng cách tìm kiếm các byte đặc biệt trong vùng dữ liệu của file. Cụ thể, mã tìm kiếm các byte `\x8d\xbe` trong đoạn mã máy của file, đây là một chỉ dấu phổ biến trong các file nén UPX. `\x8d\xbe` trong đoạn mã máy của file, đây là một chỉ dấu phổ biến trong các file nén UPX.
- Cấu trúc của PE header: Đoạn mã xác minh header của file PE. Trong khi kiểm tra các thành phần của PE header, mã tìm kiếm một số thông tin đặc trưng cho file nén UPX, như là các chỉ số trong bảng mục nhập, các đoạn mã cụ thể như

“lea edi, ...” và “mov eax, [edi]”, những đặc điểm này có thể là dấu hiệu của file nén UPX.

- Kiểm tra các chỉ số và section trong PE header: Mã kiểm tra các section của file PE và các thông số như `realstuffsiz`, `valign`, và `sectcnt` để đảm bảo rằng cấu trúc của file tuân theo chuẩn của UPX.

Hình 2.17 và Hình 2.18, 2.19 thể hiện đoạn code nhận biết và giải nén UPX của ClamAV.

```
clamav / libclamav / upx.c
Code Blame 612 lines (526 loc) · 20.8 KB
118  /* PE from UPX */
119
120  static int pefromupx(const char *src, uint32_t ssize, char *dst, uint32_t *dsize, uint32_t ep, uint32_t upx0, uint32_t upx1, uint32_t *
121  {
122      char *imports, *sections = NULL, *pehdr = NULL, *newbuf;
123      unsigned int sectcnt = 0, upd = 1;
124      uint32_t realstuffsiz = 0, valign = 0;
125      uint32_t foffset = 0x00 + 0xf8;
126
127      if ((dst == NULL) || (src == NULL))
128          return 0;
129
130      while ((valign = magic[sectcnt++])) {
131          if (CLI_ISCONTAINED(src, ssize - 5, src + ep - upx1 + valign - 2, 2) &&
132              src[ep - upx1 + valign - 2] == '\x8d' && /* lea edi, ... */
133              src[ep - upx1 + valign - 1] == '\xba') /* ... [esi + offset] */
134              break;
135      }
136
137      if (!valign && CLI_ISCONTAINED(src, ssize - 8, src + ep - upx1 + 0x00, 8)) {
138          const char *pt = &src[ep - upx1 + 0x00];
139          cli_dbgmsg("UPX: bad magic - scanning for imports\n");
140
141          while ((pt = cli_memstr(pt, ssize - (pt - src) - 8, "\xba\xbe", 2))) {
142              if (pt[6] == '\xba' && pt[7] == '\xb7') { /* lea edi, [esi+imports] / mov eax, [edi] */
143                  valign = pt - src + 2 - ep + upx1;
144                  break;
145              }
146              pt++;
147          }
148      }
149  }
```

Hình 2.17: Đoạn code nhận biết UPX của ClamAV

```
int upx_inflate2b(const char *src, uint32_t ssize, char *dst, uint32_t *dsize, uint32_t upx0, uint32_t upx1, uint32_t ep)
{
    int32_t backbytes, unp_offset = -1;
    uint32_t backsize, myebx = 0, scur = 0, dcur = 0, i, magic[] = {0x100, 0x110, 0xd5, 0};
    int oob;

    while (1) {
        while ((oob = doubleebx(src, &myebx, &scur, ssize)) == 1) {
            if (scur >= ssize || dcur >= *dsize)
                return -1;
            dst[dcur++] = src[scur++];
        }
    }
}
```

Hình 2.18: Đoạn code giải nén `upx_inflate2b`

Hàm `upx_inflate2b` và `upx_inflate2d`: Đây là các hàm thực hiện giải mã các phần của file đã được nén và phục hồi lại dữ liệu gốc.

- **`upx_inflate2b`**: Hàm này có thể liên quan đến việc giải nén dữ liệu nén theo thuật toán đặc biệt của UPX, có thể sử dụng các phương pháp như thuật toán nén LZMA, zlib, hoặc các cơ chế nén khác mà UPX áp dụng.

```
int upx_inflate2d(const char *src, uint32_t ssize, char *dst, uint32_t *dsize, uint32_t upx0, uint32_t upx1, uint32_t ep)
{
    int32_t backbytes, unp_offset = -1;
    uint32_t backsize, myebx = 0, scur = 0, dcur = 0, i, magic[] = {0x11c, 0x124, 0};
    int oob;

    while (1) {
        while ((oob = doubleebx(src, &myebx, &scur, ssize)) == 1) {
            if (scur >= ssize || dcur >= *dsize)
                return -1;
            dst[dcur++] = src[scur++];
        }
    }
}
```

Hình 2.19: Đoạn code giải nén upx\_inflate2d

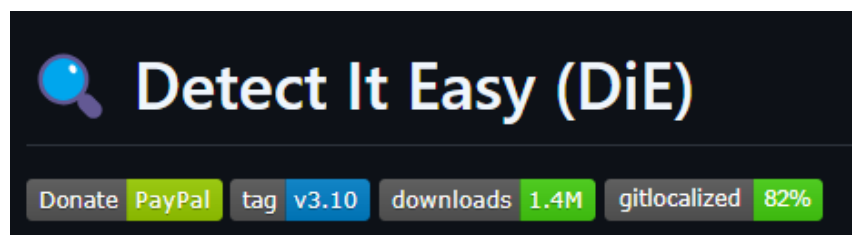
- **upx\_inflate2d**: Đây là hàm bổ sung, có thể là một phần của quy trình giải nén, giúp xử lý phần dữ liệu đã được giải nén một cách hoàn chỉnh hơn.

Bên cạnh đó còn có hàm decompress hoặc các hàm phụ trợ cung cấp một cách giải nén chung, thông qua các chỉ dẫn trong header PE và các byte chỉ định vị trí. Sau khi nhận diện file nén là UPX, hệ thống sẽ tiếp tục vào các bước giải nén thực sự.

Tương tự cho các trình đóng gói tệp tin khác.

### 2.3.3 Detect It Easy

Detect It Easy (DiE) là một công cụ mạnh mẽ dùng để nhận diện loại tệp tin, phổ biến trong giới phân tích mã độc, chuyên gia an ninh mạng và dịch ngược trên toàn thế giới. DiE hỗ trợ cả phân tích dựa trên signature và phân tích heuristic, giúp kiểm tra tệp hiệu quả trên nhiều nền tảng, bao gồm Windows, Linux và MacOS. Kiến trúc phát hiện linh hoạt khiến DiE trở thành một trong những công cụ đa năng nhất trong lĩnh vực này.



Hình 2.20: Công cụ Detect It Easy

DiE là công cụ phân tích mã độc mạnh nhờ hệ thống chữ ký linh hoạt và khả năng tùy chỉnh cao, giúp nâng cao độ chính xác so với các công cụ tĩnh truyền thống. Tuy nhiên, hạn chế lớn của DiE là chỉ phát hiện được trình đóng gói mà không thể giải nén tệp, gây khó khăn trong việc phục hồi mã gốc.



Detect It Easy cung cấp ba phiên bản khác nhau để người dùng có thể lựa chọn tùy theo nhu cầu sử dụng:

1. **die (Graphical Interface)**: Là phiên bản với giao diện đồ họa, giúp người dùng dễ dàng thao tác và trực quan hơn trong việc kiểm tra và phân tích các tệp.
2. **diel (Command-line version for batch processing)**: Phiên bản dòng lệnh này được thiết kế dành cho những người làm việc với khối lượng tệp lớn hoặc cần tự động hóa quy trình phân tích tệp.
3. **diel (Lightweight GUI version)**: Là một phiên bản nhẹ với giao diện đồ họa, được tối ưu cho các máy tính cấu hình thấp hoặc khi người dùng cần một công cụ đơn giản nhưng vẫn giữ được các tính năng quan trọng của DiE

Mỗi phiên bản đều được thiết kế để phục vụ các mục đích khác nhau, từ người dùng thông thường cho đến những chuyên gia cần xử lý lượng dữ liệu lớn một cách hiệu quả.

### Các packer được hỗ trợ bởi DiE

Detect It Easy (DiE) có khả năng phát hiện nhiều loại packer phổ biến như UPX, ASPack, FSG, PECompact, Mpress, MEW ... Ngoài ra còn có:

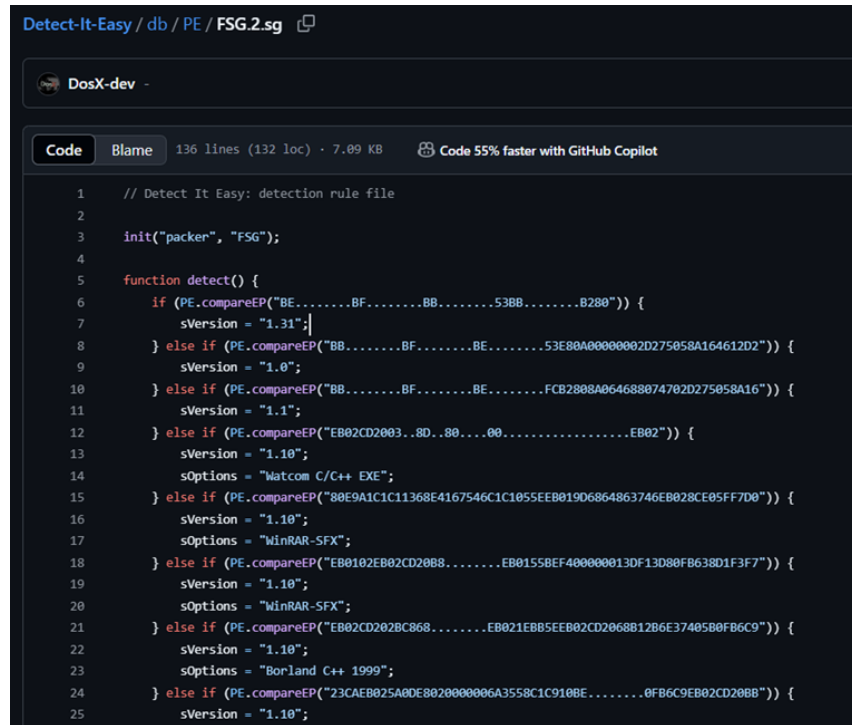
- **Themida**: một packer chuyên dụng để bảo vệ phần mềm khỏi việc phân tích.
- **kkrunchy**: một packer nhẹ, hiệu quả được thiết kế để nén các tệp EXE.
- **NSIS (Nullsoft Scriptable Install System)**: Mặc dù NSIS chủ yếu được sử dụng để tạo các trình cài đặt, nhưng nó cũng có thể nén tệp trong quá trình tạo trình cài đặt.

DiE cũng có thể phát hiện các packer tùy chỉnh hoặc các packer không phổ biến, nếu chúng tuân thủ các nguyên lý nén hoặc mã hóa được hỗ trợ.

### Cách phát hiện các trình đóng gói

Đối với các trình đóng gói, DiE sẽ viết các signature và các dấu hiệu nhận biết vào các file riêng biệt tương ứng với từng loại trình đóng gói. Các file này cũng được cập nhật liên tục để có thể thích nghi với việc các trình đóng gói tệp tin ngày càng nhiều và vô cùng tinh vi.

Đoạn mã trong Hình 2.21 là một rule file được sử dụng để phát hiện các tệp thực thi đã được nén hoặc đóng gói bằng FSG packer. Mỗi PE.compareEP so sánh điểm kết thúc (Entry Point) của tệp thực thi với các mẫu byte đặc trưng cho các phiên bản khác nhau của FSG.



```

Detect-It-Easy / db / PE / FSG.2.sg
DosX-dev

Code Blame 136 lines (132 loc) · 7.09 KB Code 55% faster with GitHub Copilot

1 // Detect It Easy: detection rule file
2
3 init("packer", "FSG");
4
5 function detect() {
6     if (PE.compareEP("BE.....BF.....BB.....53BB.....B280")) {
7         sVersion = "1.31";
8     } else if (PE.compareEP("BB.....BF.....BE.....53E80A0000002D275058A164612D2")) {
9         sVersion = "1.0";
10    } else if (PE.compareEP("BB.....BF.....BE.....FCB2808A064688074702D275058A16")) {
11        sVersion = "1.1";
12    } else if (PE.compareEP("EB02CD2003..8D..80.....00.....EB02")) {
13        sVersion = "1.10";
14        sOptions = "Watcom C/C++ EXE";
15    } else if (PE.compareEP("80E9A1C1C11368E4167546C1C1055EEB019D6864863746EB028CE05FF7D0")) {
16        sVersion = "1.10";
17        sOptions = "WinRAR-SFX";
18    } else if (PE.compareEP("EB0102EB02CD2088.....EB0155BEF400000013DF13D80F8638D1F3F7")) {
19        sVersion = "1.10";
20        sOptions = "WinRAR-SFX";
21    } else if (PE.compareEP("EB02CD202BC868.....EB021EBB5EEB02CD2068B12B6E37405B0F86C9")) {
22        sVersion = "1.10";
23        sOptions = "Borland C++ 1999";
24    } else if (PE.compareEP("23CAEB025A0DE8020000006A3558C1C910BE.....0FB6C9EB02CD2088")) {
25        sVersion = "1.10";

```

Hình 2.21: Đoạn code nhận biết FSG bằng các signature của DiE

- **PE.compareEP():** Đây là một hàm trong DiE dùng để so sánh điểm kết thúc của một tệp thực thi với một mẫu byte cụ thể. Mỗi mẫu byte là đặc trưng cho một phiên bản hoặc kiểu đóng gói của tệp.
- **Các mẫu byte:** Mỗi điều kiện kiểm tra trong mã so sánh một chuỗi byte cụ thể với tệp thực thi. Nếu điểm kết thúc của tệp khớp với mẫu, DiE sẽ nhận diện được loại packer và cung cấp thông tin về phiên bản.
- **Thông tin về phiên bản và tùy chọn:** Khi phát hiện ra tệp sử dụng packer FSG, DiE sẽ xác định phiên bản của packer (ví dụ: 1.0, 1.10, 2.0...) và có thể cung cấp thêm thông tin về các tùy chọn nén (chẳng hạn như "Borland C++", "Microsoft Visual C++", "MASM32").

Mỗi khi DiE gặp một tệp thực thi, nó sẽ kiểm tra điểm kết thúc của tệp này và so sánh với các mẫu byte đã được định nghĩa. Nếu khớp, nó sẽ trả về thông tin về packer và phiên bản.

### 2.3.4 PEiD

PEiD cung cấp giao diện người dùng (GUI) thân thiện, trực quan, và dễ sử dụng. Tỷ lệ phát hiện của công cụ nằm trong số tốt nhất so với các công cụ nhận diện hiện

có. Ngoài ra, PeiD còn hỗ trợ các chế độ quét đặc biệt giúp phát hiện những tệp đã bị chỉnh sửa hoặc chưa được nhận diện.



Hình 2.22: Công cụ PEiD

PEiD hỗ trợ quét nhiều tệp cùng lúc một cách hiệu quả. Người dùng chỉ cần kéo và thả các tệp vào giao diện chính của công cụ, sau đó một cửa sổ kết quả sẽ hiển thị thông tin chi tiết về từng tệp. Đặc biệt, PEiD hỗ trợ giao diện cập nhật chữ ký bên ngoài, cho phép người dùng tự cập nhật danh sách chữ ký để bắt kịp với các thay đổi trong môi trường.

### Các chế độ quét độc đáo trong PEiD

PEiD cung cấp ba chế độ quét khác nhau, mỗi chế độ phù hợp với các nhu cầu cụ thể của người dùng.

1. **Chế độ bình thường (Normal Mode):** Chế độ này quét điểm bắt đầu (Entry Point) của tệp PE để tìm các chữ ký đã được ghi nhận. Đây là phương pháp quét cơ bản mà hầu hết các công cụ nhận diện khác cũng sử dụng.
2. **Chế độ sâu (Deep Mode):** Chế độ này quét toàn bộ phần chứa Entry Point của tệp PE để phát hiện các chữ ký. Phương pháp này đảm bảo phát hiện được khoảng 80% các tệp đã bị chỉnh sửa hoặc mã hóa.
3. **Chế độ toàn diện (Hardcore Mode):** Đây là chế độ quét toàn bộ tệp PE để tìm tất cả các chữ ký đã ghi nhận. Chế độ này phù hợp khi các chế độ khác không đạt kết quả như mong đợi. Tuy nhiên, do các chữ ký nhỏ có thể xuất hiện nhiều lần trong các tệp khác nhau, kết quả quét có thể không chính xác.

Lưu ý: PEiD là một file thực thi trên hệ điều hành Windows, nhưng đừng lo, để chạy trên hệ điều hành Linux, chỉ cần sử dụng lệnh “wine peid.exe”.

## Các packer được hỗ trợ bởi PEiD

Hầu hết các trình đóng gói mà các công cụ được giới thiệu trên phát hiện được thì PEid cũng phát hiện được hết. Tuy nhiên, có một điểm đặc biệt ở PEiD, đó là việc lưu trữ signature cho các trình đóng gói. Mỗi trình đóng gói đều có cho mình rất nhiều signature với đa dạng các phiên bản được thu thập từ nhiều nguồn khác nhau do nhóm phát triển cam kết cập nhật signature thường xuyên để theo kịp với những thay đổi trong lĩnh vực công nghệ. Vì thế, việc phát hiện file PE bị nén trở nên tối ưu hơn.

PEiD hoạt động như một công cụ phân tích tĩnh (static analysis), tập trung vào việc so khớp signatures với mục tiêu chính là phát hiện các công cụ đóng gói được sử dụng. Điểm mạnh của nó là cơ sở dữ liệu signature phong phú và tốc độ quét nhanh, phù hợp để phân tích sơ bộ các tệp tin PE.

## 2.4 Học máy

### 2.4.1 Học máy là gì?

Học máy (Machine Learning) là một nhánh của trí tuệ nhân tạo (AI), sử dụng các thuật toán được huấn luyện trên tập dữ liệu để tạo ra các mô hình tự học, có khả năng dự đoán kết quả và phân loại thông tin mà không cần sự can thiệp của con người, nhằm thực hiện các nhiệm vụ một cách tự động và cải thiện hiệu suất cũng như độ chính xác thông qua kinh nghiệm và việc tiếp xúc với nhiều dữ liệu. Học máy hiện nay được ứng dụng trong rất nhiều lĩnh vực:

- **Thương mại điện tử & Giải trí:** Gợi ý sản phẩm, phim, nhạc dựa trên hành vi người dùng (Amazon, Netflix, Spotify).
- **Tài chính:** Phát hiện giao dịch gian lận, dự đoán điểm tín dụng.
- **Y tế:** Hỗ trợ chẩn đoán bệnh qua hình ảnh (X-ray, MRI).
- **Ngôn ngữ & Giao tiếp:** Dịch ngôn ngữ tự động (Google Translate), nhận diện và phản hồi giọng nói (Siri, Google Assistant).
- **An ninh mạng:** Phát hiện mã độc và tấn công bất thường, bảo vệ hệ thống bằng phân tích hành vi.

### 2.4.2 Phân loại học máy

Có nhiều loại học máy khác nhau được sử dụng để vận hành các sản phẩm và dịch vụ số hàng ngày. Mặc dù đều hướng đến mục tiêu chung – tạo ra máy móc và ứng

dụng có thể hoạt động không cần con người – nhưng mỗi loại lại sử dụng phương pháp riêng biệt.

Dưới đây là tổng quan về 4 loại học máy chính được sử dụng phổ biến hiện nay:

### **Học có giám sát (Supervised Learning)**

Thuật toán được huấn luyện trên tập dữ liệu đã gán nhãn, tức mỗi dữ liệu đều có “đáp án đúng”. Ví dụ: Cho thuật toán xem ảnh hoa đã có ghi loại hoa sẽ giúp nó nhận diện hoa khi gặp ảnh mới.

Thường dùng cho: Dự đoán và phân loại.

### **Học không giám sát (Unsupervised Learning)**

Thuật toán được huấn luyện trên tập dữ liệu không gán nhãn, nghĩa là không có sẵn đáp án. Thuật toán tự tìm ra mẫu, xu hướng, hoặc nhóm trong dữ liệu. Ví dụ: Phân tích dữ liệu mạng xã hội để tìm ra thói quen người dùng.

Thường dùng cho: Phân nhóm và khám phá mẫu dữ liệu lớn.

### **Học bán giám sát (Semi-supervised Learning)**

Kết hợp giữa dữ liệu có nhãn và không nhãn. Thuật toán được huấn luyện với một lượng nhỏ dữ liệu có nhãn, sau đó học tiếp với lượng lớn dữ liệu không nhãn. Ví dụ: Huấn luyện nhận diện giọng nói bằng một ít dữ liệu giọng nói có nhãn và nhiều dữ liệu chưa gán nhãn.

Thường dùng khi: Không có sẵn nhiều dữ liệu có nhãn.

### **Học tăng cường (Reinforcement Learning)**

Thuật toán học thông qua thử - sai và phản hồi (feedback) từ môi trường. Giống như trẻ em học qua trải nghiệm, thuật toán dần hiểu cách tối ưu hành động để đạt kết quả tốt nhất. Ví dụ: Thuật toán học chơi cờ qua hàng loạt ván và cải thiện chiến thuật theo kết quả thắng/thua.

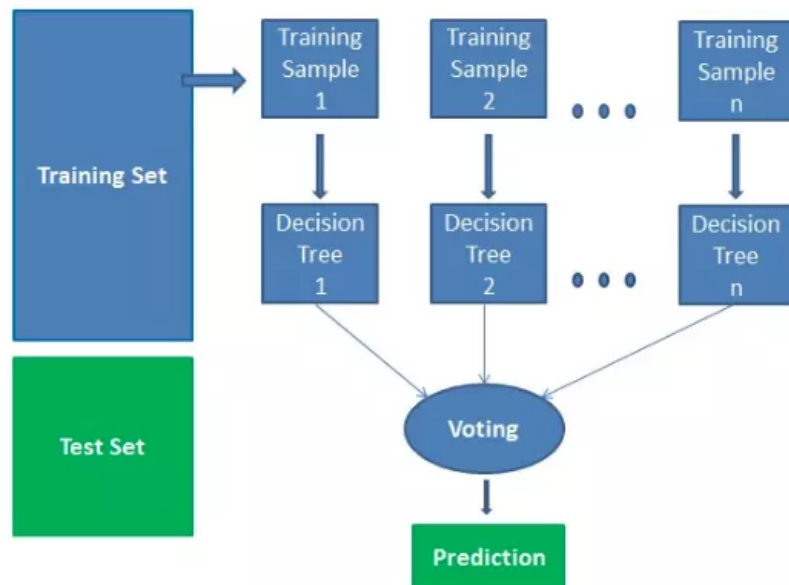
Thường dùng cho: Bài toán ra quyết định theo chuỗi hành động, như chơi game hoặc tóm tắt văn bản.

### 2.4.3 Một số mô hình học máy

#### Random Forest

Random Forest là một thuật toán học máy thuộc nhóm học có giám sát (supervised learning), dựa trên phương pháp học tổ hợp (ensemble learning). Thuật toán này được đề xuất bởi Leo Breiman vào năm 2001, và đã trở thành một trong những phương pháp phổ biến nhờ khả năng đạt độ chính xác cao và khắc phục được nhiều hạn chế của cây quyết định đơn lẻ.

Về cơ bản, Random Forest xây dựng nhiều cây quyết định từ các tập con ngẫu nhiên của dữ liệu huấn luyện và tổng hợp kết quả dự đoán từ các cây đó thông qua biểu quyết đa số (với bài toán phân loại) hoặc trung bình (với bài toán hồi quy). Việc sử dụng nhiều cây cùng lúc không chỉ tăng hiệu suất mà còn làm giảm rủi ro overfitting – một vấn đề phổ biến khi sử dụng mô hình cây đơn lẻ. Quy trình này được minh họa trong Hình 2.23.



Hình 2.23: Minh họa cách hoạt động của mô hình Random Forest

Ưu điểm:

- **Hiệu suất cao:** Đạt độ chính xác tốt trên nhiều loại tập dữ liệu, đặc biệt là khi có số lượng đặc trưng lớn hoặc dữ liệu bị nhiễu.
- **Khả năng tổng quát tốt:** Cơ chế bagging và chọn đặc trưng ngẫu nhiên giúp giảm overfitting hiệu quả hơn nhiều so với cây quyết định đơn.

- **Linh hoạt:** Hỗ trợ cả bài toán phân loại và hồi quy. Ngoài ra còn có thể dùng để phát hiện giá trị bất thường (anomaly detection).
- **Ít yêu cầu tiền xử lý:** Không cần chuẩn hóa hoặc chuyển đổi đặc trưng (scaling, normalization), vẫn hoạt động tốt với dữ liệu có phân phối không đồng đều.
- **Đánh giá tầm quan trọng của đặc trưng:** Cung cấp thông tin định lượng về mức độ ảnh hưởng của từng đặc trưng đầu vào tới kết quả dự đoán.

Nhược điểm:

- **Kích thước mô hình lớn:** Do số lượng cây lớn nên mô hình có thể chiếm nhiều bộ nhớ và tốn thời gian hơn khi dự đoán.
- **Khó giải thích:** Mặc dù mỗi cây riêng lẻ có thể dễ hiểu, nhưng tổ hợp nhiều cây làm cho toàn bộ mô hình trở thành "hộp đen", gây khó khăn trong việc phân tích kết quả.
- **Không tối ưu trong thời gian thực:** Nếu yêu cầu phản hồi nhanh như trong các hệ thống thời gian thực, Random Forest có thể không phù hợp.

## XGBoost

XGBoost (viết tắt của eXtreme Gradient Boosting) là một thuật toán học máy mạnh mẽ, được xây dựng dựa trên kỹ thuật Gradient Boosting, chuyên dùng cho các bài toán học có giám sát như phân loại (classification), hồi quy (regression) và xếp hạng (ranking). Nhờ hiệu suất vượt trội và khả năng xử lý dữ liệu lớn, XGBoost đã trở thành một trong những thuật toán phổ biến nhất trong lĩnh vực học máy, đặc biệt được ưa chuộng trong các cuộc thi trên nền tảng Kaggle.

Khác với các mô hình cây truyền thống, XGBoost tích hợp thêm nhiều cơ chế như regularization, xử lý giá trị thiếu, và tối ưu truy cập bộ nhớ, giúp cải thiện khả năng tổng quát hóa và tốc độ huấn luyện.

XGBoost là một mô hình học tổ hợp, xây dựng nhiều cây quyết định tuần tự. Mỗi cây mới được huấn luyện nhằm giảm lỗi của cây trước đó. Quá trình này gọi là boosting, và cụ thể hơn là gradient boosting. Các bước hoạt động chính gồm:

1. **Khởi đầu với dự đoán cơ bản:** Cây quyết định mô hình đầu tiên được đào tạo trên dữ liệu. Trong các tác vụ hồi quy, mô hình cơ sở này chỉ đơn giản là dự đoán giá trị trung bình của biến mục tiêu.
2. **Tính toán lỗi :** Sau khi đào tạo cây đầu tiên, lỗi giữa giá trị dự đoán và giá trị thực tế sẽ được tính toán.

3. **Huấn luyện cây tiếp theo:** Bước này cố gắng sửa các lỗi do cây đầu tiên gây ra.
4. **Lập lại quy trình :** Quy trình này tiếp tục với mỗi cây mới cố gắng sửa lỗi của các cây trước đó cho đến khi đạt được tiêu chí dừng.
5. **Kết hợp các dự đoán :** Dự đoán cuối cùng là tổng các dự đoán từ tất cả các cây.

Quy trình hoạt động của XGBoost được minh họa trong Hình 2.24.



**Hình 2.24:** Minh họa cách hoạt động của mô hình XGBoost

Ưu điểm:

- Có khả năng mở rộng và hiệu quả cho các tập dữ liệu lớn hàng triệu mẫu.
- Hỗ trợ xử lý song song và tăng tốc GPU để đào tạo nhanh hơn.
- Có thể tự động xử lý dữ liệu bị thiếu, giảm thiểu nhu cầu tiền xử lý.
- Sử dụng regularization (L1 & L2) để tránh tình trạng quá khớp và tăng khả năng khái quát hóa.

Nhược điểm:

- **Tốn tài nguyên:** Mô hình phức tạp, đòi hỏi RAM/CPU/GPU cao.
- **Dễ overfitting** nếu không điều chỉnh tham số cẩn thận (đặc biệt khi tập dữ liệu nhỏ).
- **Khó giải thích:** khả năng diễn giải mô hình tổng thể bị hạn chế hơn so với các phương pháp đơn giản hơn.



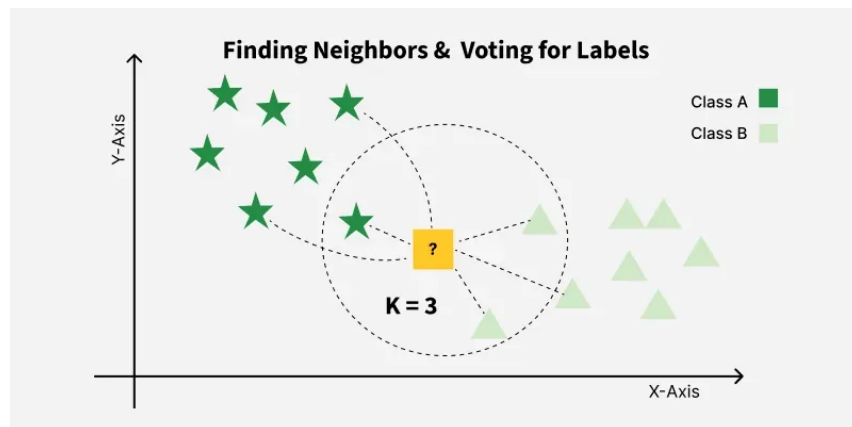
## K-Nearest Neighbors

K-Nearest Neighbors (KNN) là một thuật toán học có giám sát, thuộc nhóm lazy learning (học lười), vì nó không huấn luyện mô hình trong giai đoạn training mà chỉ lưu lại toàn bộ dữ liệu. Khi cần dự đoán, KNN sẽ tìm  $k$  điểm gần nhất với điểm mới trong tập huấn luyện để đưa ra kết quả. Thuật toán này có thể áp dụng cho cả phân loại (classification) và hồi quy (regression), nhưng thường được dùng nhiều hơn trong phân loại.

KNN hoạt động dựa trên nguyên lý: “điểm gần nhau thì có khả năng cùng nhãn”. Các bước cơ bản:

1. Tính khoảng cách giữa điểm cần dự đoán và tất cả các điểm trong tập huấn luyện (phổ biến nhất là Euclidean, ngoài ra còn có Manhattan, Minkowski, Hamming...).
2. Chọn  $k$  điểm gần nhất (có khoảng cách nhỏ nhất).
3. Phân loại:
  - Với classification: dự đoán nhãn theo đa số phiếu trong  $k$  lân cận.
  - Với regression: trả về trung bình giá trị của các điểm lân cận.

Quy trình hoạt động của KNN được minh họa trong Hình 2.25.



**Hình 2.25:** Minh họa cách hoạt động của mô hình K-Nearest Neighbors

Lựa chọn  $k$  phù hợp là quan trọng:

- $k$  nhỏ  $\rightarrow$  dễ nhiễu, overfitting
- $k$  lớn  $\rightarrow$  dễ underfitting, mất chi tiết

Ưu điểm:

- Thuật toán đơn giản, dễ dàng triển khai.
- Độ phức tạp tính toán nhỏ.
- Chỉ cần thiết lập số lượng hàng xóm ( $k$ ) và phương thức khoảng cách.

Nhược điểm:

- Với  $K$  nhỏ dễ gặp nhiễu dẫn tới kết quả đưa ra không chính xác.
- Chậm với dữ liệu lớn.
- Độ chính xác giảm khi dữ liệu có quá nhiều đặc trưng.
- Dễ overfitting nếu dữ liệu có nhiều hoặc số chiều cao (curse of dimensionality)

### Support Vector Machine

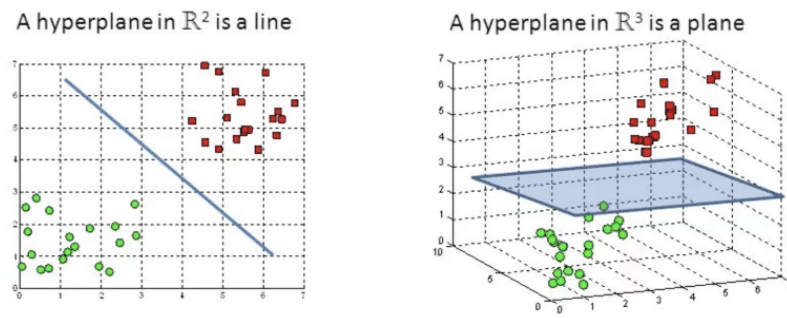
Support Vector Machine (SVM) là một thuật toán học máy có giám sát, được sử dụng phổ biến cho các bài toán phân loại và hồi quy. Mục tiêu chính của SVM là tìm một siêu phẳng (hyperplane) phân tách tối ưu các lớp dữ liệu bằng cách tối đa hóa khoảng cách (margin) giữa hai lớp. SVM đặc biệt hiệu quả trong các không gian có số chiều lớn và vẫn được ưa chuộng trong xử lý văn bản, phát hiện xâm nhập, nhận dạng mẫu,...

SVM được phát triển bởi Vladimir Vapnik và đồng nghiệp từ năm 1963, và được mở rộng mạnh mẽ vào thập niên 1990 với kỹ thuật kernel trick – cho phép phân loại dữ liệu phi tuyến bằng cách ánh xạ lên không gian chiều cao hơn.

Mục tiêu của SVM là tìm một siêu phẳng (hyperplane) trong không gian  $N$  chiều (với  $N$  là số đặc trưng) sao cho siêu phẳng này phân tách hai lớp dữ liệu một cách tối ưu. Cụ thể, SVM cố gắng chọn siêu phẳng có lề (margin) lớn nhất – tức là khoảng cách từ siêu phẳng đến các điểm dữ liệu gần nhất của hai lớp là lớn nhất có thể.

Trong không gian  $N$  chiều, siêu phẳng sẽ là một không gian con có  $N-1$  chiều. Các điểm nằm gần nhất với siêu phẳng (hoặc nằm trên ranh giới lề) được gọi là vector hỗ trợ (support vectors) - chính các điểm này quyết định vị trí và hướng của siêu phẳng. Việc tối ưu hóa siêu phẳng thực chất là tối đa hóa khoảng cách đến các vector hỗ trợ của hai lớp. Hình 2.26 minh họa siêu phẳng phân tách hai lớp dữ liệu trong không gian hai chiều và ba chiều.

Ưu điểm:



**Hình 2.26:** Minh họa siêu phẳng phân tách hai lớp dữ liệu của SVM trong không gian hai chiều và ba chiều

- Hiệu suất cao với dữ liệu nhiều chiều
- SVM chỉ tập trung vào các vector hỗ trợ (support vectors), giúp giảm thiểu mức sử dụng bộ nhớ so với các thuật toán khác.
- Hỗ trợ tốt cho cả phân loại nhị phân và phân loại đa lớp

Nhược điểm:

- Thời gian huấn luyện chậm
- Việc chọn đúng kernel và điều chỉnh các tham số như C đòi hỏi phải điều chỉnh cẩn thận, tác động đến các thuật toán SVM.
- Độ phức tạp của siêu phẳng ở các chiều cao hơn khiến SVM khó diễn giải hơn các mô hình khác.

#### 2.4.4 Các chỉ số đánh giá mô hình học máy

Để đánh giá chất lượng của mô hình đã huấn luyện, chúng ta cần những chỉ số phù hợp để đánh giá mô hình. Trước khi giới thiệu về các chỉ số đánh giá này, nhóm xin giới thiệu các chỉ số cơ bản là:

- TP (True Positive): Tổng số trường hợp dự báo khớp Positive.
- TN (True Negative): Tổng số trường hợp dự báo khớp Negative.
- FP (False Positive): Tổng số trường hợp dự báo các quan sát thuộc nhãn Negative thành Positive.

- FN (False Negative): Tổng số trường hợp dự báo các quan sát thuộc nhãn Positive thành Negative.

Một số chỉ số dùng để đánh giá mô hình phân loại như:

- Độ chính xác (Accuracy): tỷ lệ các trường hợp được dự báo đúng trên tổng số các trường hợp.

$$\text{Accuracy} = \frac{TP + TN}{\text{total sample}}$$

- Precision: cho chúng ta biết thực sự có bao nhiêu dự đoán Positive là thật sự True.

$$\text{Precision} = \frac{TP}{TP + FP}$$

- Recall: tỷ lệ dự báo chính xác các trường hợp positive trên toàn bộ các mẫu thuộc nhóm positive.

$$\text{Precision} = \frac{TP}{TP + FN}$$

- F1 Score: là trung bình điều hòa giữa precision và recall. Do đó nó đại diện hơn trong việc đánh giá độ chính xác trên đồng thời precision và recall.

$$\text{F1-score} = \frac{2 * \text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}}$$

# Chương 3

## PHƯƠNG PHÁP THỰC HIỆN

### 3.1 Môi trường thực hiện

Khi thực hiện thực nghiệm, nhóm đã sử dụng máy ảo Vcenter với cấu hình sau:

- Hệ điều hành: kali linux
- Dung lượng RAM: 6GB
- Số bộ xử lý (Processors): 4CPU
- Dung lượng ổ cứng: 400GB

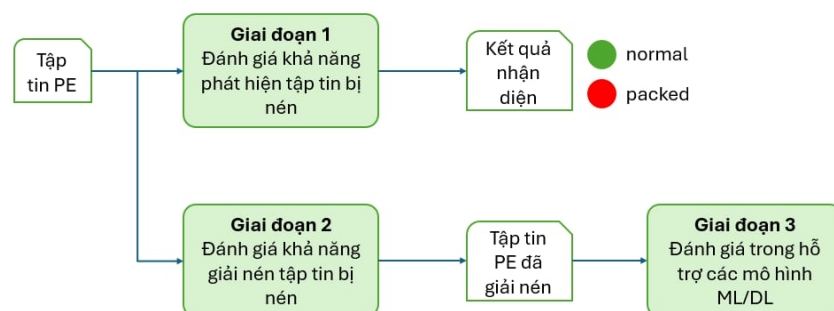
### 3.2 Mô hình tổng quan phương pháp thực hiện

Đề tài này nhằm mục đích đánh giá tổng quan về các công cụ hỗ trợ việc phát hiện và giải nén các tệp thực thi bị đóng gói, cùng với tác động của việc giải nén đối với khả năng phát hiện phần mềm độc hại của các mô hình học máy. Để đạt được mục tiêu này, nhóm đã đề xuất một quy trình thử nghiệm với ba giai đoạn chính: 1) đánh giá khả năng phát hiện các tệp bị đóng gói, 2) đánh giá khả năng giải nén và 3) phân tích tác động của việc giải nén đối với hiệu suất của các mô hình học máy.

Hình 3.1 cho thấy quy trình tổng quát của phương pháp triển khai.

Cụ thể: Dữ liệu đầu vào là một tệp thực thi ở định dạng PE (Portable Executable) đã bị đóng gói và được xử lý theo ba giai đoạn sau:

- Giai đoạn 1 - Đánh giá khả năng phát hiện các tệp bị đóng gói: Các tệp PE được đưa vào các công cụ phát hiện trình đóng gói để xác định xem chúng có được đóng gói hay không và ghi lại loại trình đóng gói (nếu có).



**Hình 3.1:** Pipeline quy trình thực hiện nghiên cứu

- Giai đoạn 2 - Đánh giá khả năng giải nén các tệp đã đóng gói: Các tệp bị đóng gói sẽ tiếp tục được xử lý bằng công cụ giải nén. Các tệp đã giải nén thành công sẽ được kiểm tra xem có các hành vi tương đồng với các tệp bị nén hay không để phân tích thêm.
- Giai đoạn 3 - Đánh giá hiệu quả của việc giải nén đối với đào tạo các mô hình học máy: Các tệp được chia thành 3 tập dữ liệu (các tệp bị đóng gói; các tệp đã giải nén (chỉ gồm những tệp đã được kiểm tra và xác định là tương đồng với tệp bị nén); các tệp đã giải nén (tất cả các tệp được giải nén bởi công cụ, bao gồm cả tệp tương đồng và không tương đồng)) sẽ được sử dụng để đào tạo các mô hình học máy. Bằng cách xây dựng ba kịch bản đào tạo, nhóm đánh giá sự khác biệt về độ chính xác, độ nhạy và khả năng phân loại của các tập dữ liệu khác nhau.

Quy trình trên không chỉ cho phép đánh giá toàn diện khả năng kỹ thuật của các công cụ phát hiện và giải nén mà còn đánh giá tác động thực tế của việc giải nén đối với hiệu suất của các hệ thống phát hiện phần mềm độc hại dựa trên học máy.

### 3.3 Giai đoạn 1 - Đánh giá khả năng phát hiện các tệp bị đóng gói

#### 3.3.1 Mục tiêu và cách tiếp cận

Giai đoạn đầu tiên của quy trình thử nghiệm nhằm đánh giá hiệu quả của một số công cụ phát hiện đóng gói trong việc xác định các tệp thực thi được đóng gói.

Tập dữ liệu được sử dụng trong thực nghiệm được trích xuất từ tập dữ liệu đi kèm của bài báo "*When Malware is Packin' Heat: Limits of Machine Learning Classifiers Based on Static Analysis Features*"[1]. Đây là tập dữ liệu được xây dựng đặc biệt để

kiểm tra tác động của các kỹ thuật đóng gói đối với khả năng phát hiện phần mềm độc hại bằng cách sử dụng phân tích tĩnh. Trong đề tài này, nhóm sử dụng 4 thư mục riêng biệt, mỗi thư mục tương ứng với một loại packer (UPX, MEW, XPack, Petite), mỗi nhóm chứa 2.000 tệp thực thi định dạng PE.

Bốn công cụ được sử dụng để đánh giá khả năng phát hiện trình đóng gói là: Detect It Easy (DiE), PEiD, ClamAV và Unipacker. Tất cả các công cụ đều được triển khai tự động thông qua script Python để đảm bảo tính nhất quán trong quá trình xử lý và giảm thiểu lỗi của con người. Kết quả đầu ra được lưu dưới dạng tệp .csv. Các tệp này được xử lý và sắp xếp thành các bảng thống kê để phân tích tỷ lệ phát hiện tệp bị đóng gói, tốc độ xử lý và tỷ lệ phát hiện chính xác tệp bị đóng gói bởi trình đóng gói nào. Kết quả sẽ được mô tả ở Chương 4

### 3.3.2 Quy trình thực hiện

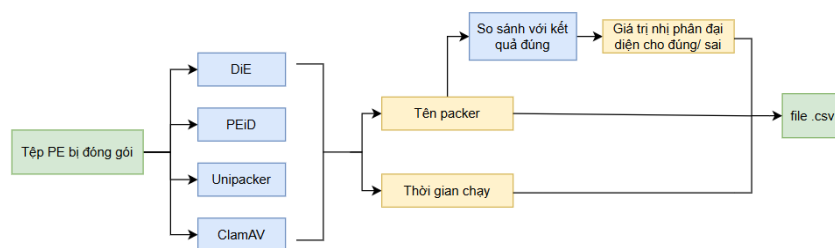
Để đánh giá chính xác và khách quan hiệu quả của các công cụ phát hiện packer, nhóm đã xây dựng một quy trình hoàn toàn tự động bằng script python. Ở giai đoạn này, tập dataset được sử dụng hoàn toàn là các tệp bị đóng gói (không chứa các tệp không bị đóng gói), vì vậy mục tiêu của quy trình là đánh giá khả năng xác định chính xác file có bị đóng gói hay không và bị đóng gói bởi trình đóng gói nào.

Để thử nghiệm, nhóm đã sử dụng bốn công cụ phát hiện packer: Detect It Easy (DiE), PEiD, ClamAV và Unipacker. Mỗi công cụ có phương thức hoạt động và định dạng đầu ra khác nhau, do đó cần phải gọi và xử lý độc lập thông qua các tập lệnh tự động. Ngôn ngữ lập trình được sử dụng là Python và bao gồm các thư viện chuẩn như os, subprocess, time,... Toàn bộ quy trình được chia thành bốn bước chính, mỗi công cụ được thực hiện lần lượt.

Hệ thống chứa một hàm xử lý chính *process\_folder()*, có trách nhiệm duyệt tất cả các tệp trong một thư mục dữ liệu nhất định. Đối với mỗi tệp .exe, hệ thống sẽ gọi bốn công cụ theo trình tự thông qua subprocess hoặc pexpect, tùy thuộc vào đặc điểm giao diện của công cụ. Mỗi lệnh gọi công cụ trả về:

- Tên trình đóng gói (nếu phát hiện).
- Thời gian xử lý tính bằng giây.
- Xác định trình đóng gói đúng hay sai (So sánh tên của trình đóng gói với tên thư mục chứa tệp bị đóng gói ban đầu)

Kết quả của từng công cụ sẽ được ghi vào tệp CSV tương ứng để thống kê và so sánh trong các bước tiếp theo. Cụ thể, đoạn script python này sẽ được hoạt động như Hình 3.2.



Hình 3.2: Sơ đồ tổng quan hóa quy trình thực thi giai đoạn 1

Phương pháp tương tác với từng công cụ cụ thể như sau:

#### a. Detect It Easy (DiE)

Đây là một công cụ phát hiện trình đóng gói phổ biến. Công cụ này có 3 phiên bản (được giới thiệu ở Chương 2 nhưng ở phần này, nhóm sẽ sử dụng phiên bản dòng lệnh được gọi thông qua lệnh 'diec' và trích xuất đầu ra từ các dòng chứa cụm từ "Packer:". VD:

```
$diec mew_packed.exe
Packer: MEW
```

Toàn bộ tương tác được xử lý bởi subprocess.run, bắt đầu tính thời gian trước khi lệnh được gọi và kết thúc sau khi nhận được đầu ra. Nếu đầu ra chứa chuỗi khớp với định dạng "Packer: <packer\_name>", kết quả sẽ được lưu cùng với thời gian xử lý. Nếu không, giá trị "None" sẽ được trả về.

#### b. PEiD

PEiD được gọi theo cách tương tự như DiE, vẫn sử dụng phiên bản dòng lệnh thông qua dòng lệnh 'peid <file\_path>'.

```
(kali@kali) - [~]
$ peid /home/kali/unipacker/Sample/sample/lbop20_UPX.exe
UPX Modified >> *$igBy Ahmed18
UPX
UPX
UPX
UPX (Delphi) Stub
```

Hình 3.3: Phân tích tệp tin với PEiD phiên bản command line

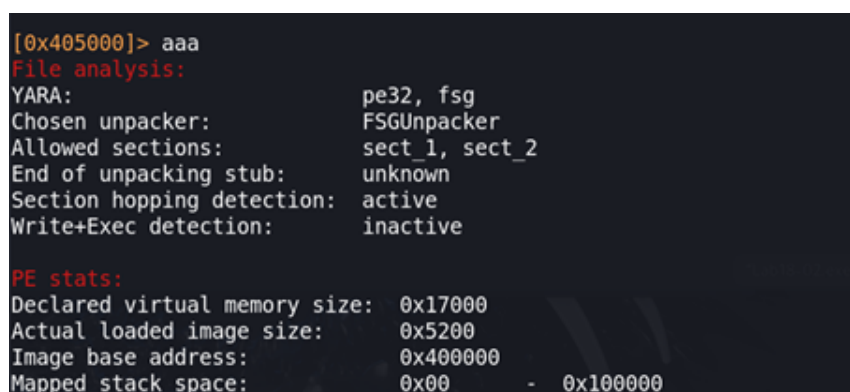
Một điểm khác biệt là PEiD có xu hướng trả về kết quả rỗng nếu không phát hiện được packer và khi đã phát hiện được sẽ trả về nhiều dòng chứa tên packer (như Hình 3.3). Do đó, đoạn script tự động sẽ kiểm tra chuỗi kết quả được trả về, nếu không có gì được trả về thì sẽ được ghi nhận giá trị là "None", ngược lại sẽ trả về dòng đầu tiên trong kết quả trả về.



**c. Unipacker**

Unipacker là một công cụ dòng lệnh tương tác và cần khá nhiều bước để tiến hành phân tích một tệp tin. Để tự động hóa tương tác với Unipacker, nhóm đã sử dụng thư viện pexpect trong Python. Quy trình bao gồm:

- (a) Khởi tạo quy trình shell.py (giao diện của Unipacker).
- (b) Gửi ID tùy chọn tương ứng với trình đóng gói cần phân tích.
- (c) Gửi đường dẫn tệp cần quét.
- (d) Gửi lệnh "aaa" để bắt đầu phân tích tệp.
- (e) Ghi lại thời gian xử lý và trích xuất kết quả từ dòng có chứa "Chosen unpacker:". Nếu không phát hiện trình đóng gói phù hợp, công cụ sẽ trả về "None". Hình 3.4 cho thấy kết quả trả về khi kiểm tra một tệp bằng Unipacker.



```
[0x405000]> aaa
File analysis:
YARA:                                pe32, fsg
Chosen unpacker:                      FSGUnpacker
Allowed sections:                     sect_1, sect_2
End of unpacking stub:                unknown
Section hopping detection:            active
Write+Exec detection:                 inactive

PE stats:
Declared virtual memory size: 0x17000
Actual loaded image size: 0x5200
Image base address: 0x400000
Mapped stack space: 0x00 - 0x100000
```

Hình 3.4: Kiểm tra thông tin tệp trong Unipacker

**d. ClamAV**

ClamAV thường được sử dụng với câu lệnh 'clamscan <file\_path>'. Tuy nhiên, để phân tích sâu hơn thì nhóm sẽ dùng thêm một số tùy chọn:

---

```
clamscan --debug --detect-pua=yes --leave-temp=yes <file_path>
```

---

Với file\_path là đường dẫn đến tệp tin cần phân tích để tiến hành quét.

- `--debug`: In ra thông tin gỡ lỗi chi tiết trong quá trình quét, có thể tìm kiếm tên các packer ở đây.
- `--detect-pua=yes`: Bật tính năng phát hiện PUA (Potentially Unwanted Applications) nhằm mở rộng phạm vi phát hiện.
- `--leave-temp=yes` (Chức năng này sẽ được làm rõ ở phần sau).

Sau khi ClamAV đã phân tích xong, kết quả sẽ được kiểm tra theo hai cách:

- Dựa vào tên packer trong PUA.Win.Packer.\* (nếu kết quả trả về có chứa cụm SCAN SUMMARY).
- Hoặc phân tích dòng log có chứa chuỗi như "libClamAV debug: UPX".

Vì khi sử dụng công cụ thường gặp lỗi hệ thống như “zsh: killed” nên quá trình thực thi sẽ được giới hạn bằng cách cho phép chương trình được lặp lại tối đa 5 lần.

Sau khi có được kết quả trong các file csv, nhóm sẽ thực hiện thống kê tự động bằng các hàm tính toán và kết quả sẽ được mô tả ở Chương 4.

## 3.4 Giai đoạn 2 - Đánh giá khả năng giải nén các tệp đã đóng gói

### 3.4.1 Mục tiêu và cách tiếp cận

Giai đoạn thứ hai của quy trình thử nghiệm nhằm đánh giá khả năng giải nén các tệp thực thi bị đóng gói bằng các công cụ giải nén mà nhóm đang có. Mục tiêu của giai đoạn này là kiểm tra xem các công cụ có thể tái tạo lại tệp gốc từ tệp đã bị đóng gói hay không, đồng thời đánh giá mức độ thành công, độ chính xác và độ tin cậy của quá trình giải nén.

Tập dữ liệu đầu vào là các tệp thực thi (định dạng PE) đã được đóng gói từ giai đoạn trước, bao gồm 4 thư mục tương ứng với 4 loại packer: UPX, MEW, XPack và Petite (tương tự như Giai đoạn 1).

Các công cụ được xem xét cho nhiệm vụ giải nén bao gồm:

- Unipacker: là một trong hai công cụ chính được sử dụng để giải nén. Unipacker được lựa chọn vì có khả năng phân tích tệp tại runtime, giải nén nhiều loại packer như UPX, MEW, Petite và tạo ra tệp unpacked có thể thực thi được.
- ClamAV: là công cụ giải nén thứ 2 được sử dụng. Mặc dù không phải là công cụ unpacker chuyên biệt, ClamAV vẫn được sử dụng để hỗ trợ quá trình giải nén thông qua cơ chế giải mã (bytecode engine) và phát hiện các signature ẩn bên trong tệp. Tuy nhiên, khả năng giải nén đầy đủ vẫn còn là một hạn chế.
- PEiD và Detect It Easy (DiE): không được sử dụng cho giai đoạn này, vì đây là các công cụ chỉ có khả năng phát hiện packer, không hỗ trợ chức năng giải nén hay khôi phục mã gốc.

Tương tự như Giai đoạn 1, tất cả quá trình giải nén bằng Unipacker và ClamAV sẽ được thực hiện tự động thông qua Python script. Khác một chút là giai đoạn này không chỉ lưu file csv kết quả mà còn chứa các tệp đã được giải nén thành công. Để đảm bảo tính chính xác và đánh giá hiệu quả, mỗi tệp sau khi unpack sẽ được kiểm tra tính thực thi bằng cách gọi chạy trực tiếp bằng Cuckoo - một môi trường sandbox. Các tệp được xem là giải nén thành công nếu chúng có thể thực thi và thực hiện đúng chức năng tương tự như tệp bị đóng gói.

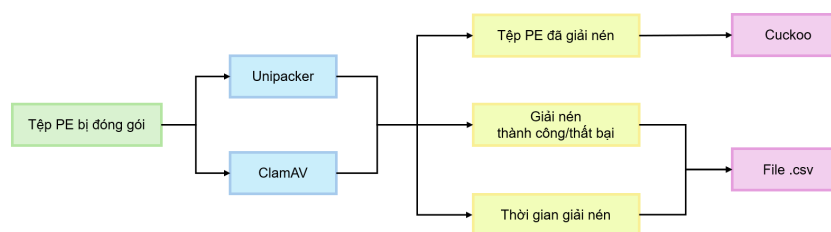
Kết quả của giai đoạn này sẽ cung cấp cơ sở cho giai đoạn ba, nơi nhóm đánh giá ảnh hưởng của việc giải nén đến hiệu quả của các mô hình học máy.

### 3.4.2 Quy trình thực hiện

Để đảm bảo tính nhất quán và hạn chế lỗi thủ công trong quá trình xử lý dữ liệu lớn, nhóm tiếp tục xây dựng một script Python tự động hóa toàn bộ quá trình giải nén cho các công cụ được sử dụng trong giai đoạn này. Quy trình được thiết kế sao cho có thể xử lý hàng nghìn tệp một cách tuần tự, ghi nhận kết quả đầu ra của từng công cụ, và lưu thông tin vào các tệp .csv để thuận tiện cho phân tích.

Quy trình giải nén được xây dựng dựa trên các thư viện chuẩn như os, subprocess, pexpect, shutil và hashlib, cho phép kiểm soát chi tiết việc gọi thực thi công cụ, xử lý file đầu ra, kiểm tra trạng thái unpack và ghi nhận kết quả.

Hình 3.5 mô tả cách thức hoạt động của quá trình giải nén.



**Hình 3.5:** Sơ đồ tổng quan hóa quy trình thực thi giai đoạn 2

Đầu ra của giai đoạn này bao gồm:

- Các tệp unpacked (nếu giải nén thành công).
- Báo cáo trạng thái unpack (thành công/thất bại, tốc độ xử lý) cho từng tệp, được lưu dưới dạng file csv.

Phương pháp tương tác với từng công cụ cụ thể như sau:

#### a. Giải nén bằng Unipacker

Không giống với các công cụ phát hiện tĩnh thông thường, Unipacker sử dụng

phân tích động (dynamic analysis) để theo dõi quá trình thực thi của tệp, từ đó phát hiện thời điểm mã gốc được giải nén vào bộ nhớ và tiến hành trích xuất. Đối với Unipacker, luồng thực thi được triển khai như sau:

- (a) Khởi tạo quy trình shell.py (giao diện của Unipacker).
- (b) Gửi ID tùy chọn tương ứng với trình đóng gói cần phân tích.
- (c) Gửi đường dẫn tệp cần quét.
- (d) Thực hiện lệnh sau bằng subprocess và đo thời gian chạy để xác định thời gian giải nén.

---

```
unipacker <file_path>-d <output_folder>
```

---

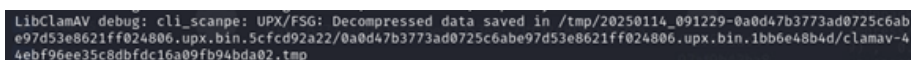
- (e) Nếu quá trình giải nén thành công, chương trình sẽ tìm chuỗi "Saved to ..." trong kết quả đầu ra để xác định đường dẫn của tệp tin đã giải nén.

#### b. Giải nén bằng ClamAV

Do không phải là công cụ unpack chuyên dụng nên ClamAV không tạo ra một tệp kết quả có định dạng .exe hoàn chỉnh như Unipacker. Kết quả của ClamAV được lưu lại dưới dạng một file .tmp trong bộ nhớ tạm.

Để tiến hành lấy tệp tin giải nén, chúng tôi thực hiện các bước sau:

- (a) Sử dụng tùy chọn "-leave-temp=yes" như đã giới thiệu ở trên để giữ lại toàn bộ bộ nhớ tạm này nhằm mục đích truy xuất file.
- (b) Thực hiện tìm kiếm trong đầu ra (stdout) của ClamAV các dòng chứa thông tin về file tạm chứa nội dung sau giải nén, cụ thể là chuỗi: "Decompressed data saved in /tmp/..." hoặc "Unpacked and rebuilt executable saved in /tmp/..." như Hình 3.6. Nếu tìm được dòng này, đoạn code sẽ di chuyển tệp .tmp về thư mục đích do người dùng chỉ định.



```
LibClamAV debug: cli_scanpe: UPX/FSG: Decompressed data saved in /tmp/20250114_091229-0a0d47b3773ad0725c6ab
e97d53e8621ff024806.upx.bin.5cfd92a22/0a0d47b3773ad0725c6abe97d53e8621ff024806.upx.bin.1bb6e48b4d/clamav-4
4ebf96ee35c8dbfcd16a09fb94bda02.tmp
```

**Hình 3.6:** Kết quả quét tệp tin bằng ClamAV

#### c. Kiểm tra chức năng giải nén bằng Cuckoo

Sau khi chạy các công cụ giải nén và có file kết quả, một vấn đề quan trọng đặt ra là liệu các tệp kết quả sau giải nén có duy trì được chức năng tương tự như tệp gốc đã bị đóng gói hay không. Để kiểm tra điều này, nhóm đã tiến hành chạy phân tích động trên Cuckoo Sandbox cho cả hai phiên bản: tệp bị đóng gói (bản gốc) và tệp sau khi được giải nén.

Hai tệp được xem là tương đương về mặt chức năng nếu danh sách API được gọi là tương đồng. Đây là một tiêu chí thường được sử dụng trong phân tích phần mềm độc hại để đánh giá sự giống nhau về hành vi. File sau khi giải nén thường chứa nhiều API runtime hơn file gốc, do việc load thêm thư viện và sử dụng API như LoadLibrary hay GetProcAddress tại thời điểm chạy. Điều này khiến tập API của file gốc trở thành tập con của file unpack, tức AB. Do đó, nhóm đề xuất sử dụng Overlap coefficient:

$$O(A,B) = \frac{|A \cap B|}{\min(|A|, |B|)}$$

### 3.5 Giai đoạn 3 - Đánh giá hiệu quả của việc đào tạo các mô hình học máy

#### 3.5.1 Mục tiêu và cách tiếp cận

Trong giai đoạn thứ ba của thử nghiệm, mục tiêu chính là đánh giá tác động của việc giải nén các tệp thực thi được đóng gói (PE) lên hiệu suất của các mô hình học máy trong phân loại phần mềm độc hại. Đây là một bước quan trọng để xác định liệu quá trình giải nén có thực sự cải thiện khả năng học tập và nhận diện của các thuật toán phân loại dựa trên đặc trưng tĩnh hay không.

Phân tích tĩnh (static analysis) là một kỹ thuật phổ biến trong lĩnh vực an ninh mạng và các mô hình học máy thường được đào tạo bằng các tính năng được trích xuất từ mã nhị phân mà không cần phải thực thi các tệp. Tuy nhiên, việc các tệp này bị đóng gói bằng các công cụ nén hoặc bảo vệ (packer/protector), có thể thay đổi hoặc ẩn nhiều đặc trưng có giá trị. Do đó, mô hình học máy có thể nhận được thông tin không đầy đủ hoặc nhiễu, dẫn đến hiệu suất kém. Do đó, giả thuyết được đề xuất trong đề tài này là việc giải nén các tệp bị đóng gói có thể khôi phục thông tin ẩn, do đó cải thiện độ chính xác của các mô hình học máy.

Để xác minh giả thuyết này một cách có hệ thống và khách quan, nhóm đã đề xuất ba kịch bản đào tạo mô hình sau, mỗi kịch bản tương ứng với một tình huống thực tế phổ biến trong quá trình phát hiện mã độc bằng phương pháp học máy:

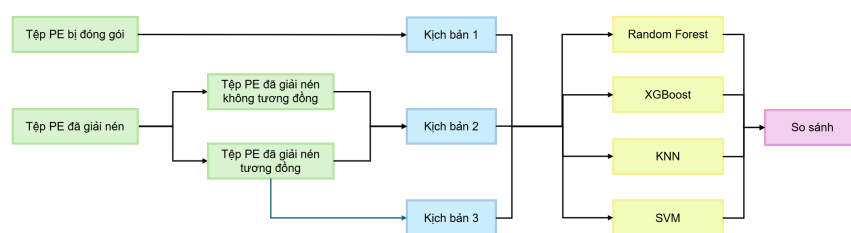
- Kịch bản 1 - Huấn luyện với dữ liệu đóng gói
  - Mô tả: Trong kịch bản này, toàn bộ tập dữ liệu đầu vào đều là những tệp thực thi PE bị đóng gói.

- Mục tiêu kiểm chứng: Kịch bản này phản ánh tình huống phổ biến mà các hệ thống học máy hiện tại phải xử lý trực tiếp dữ liệu "thô" (mà không trải qua giai đoạn giải nén file nếu file đã bị đóng gói).
- Kịch bản 2 - Huấn luyện với dữ liệu đã giải nén (bao gồm cả tệp tương đồng và không tương đồng với tệp gốc)
  - Mô tả: Trong kịch bản này, các tệp bị đóng gói được xử lý bằng một công cụ giải nén phù hợp (như Unipacker, ClamAV). Tuy nhiên, các tệp không được lọc dựa trên mức độ tương đồng của chúng với nội dung tệp gốc. Như vậy, một số tệp sau khi giải nén có thể khác biệt hoàn toàn về cấu trúc hoặc thông tin tĩnh, do việc giải nén không thành công hoặc công cụ không khôi phục được đúng nội dung ban đầu.
  - Mục tiêu kiểm chứng: Kiểm tra tính tổng quát hóa của mô hình: khi giải nén ra được cả các file không hoàn toàn giống bản gốc, mô hình có thể học được đặc trưng nào và hiệu quả phát hiện ra sao.
- Kịch bản 3 - Huấn luyện với dữ liệu giải nén có chọn lọc (chỉ giữ lại các tệp tương tự như tệp gốc)
  - Mô tả: Đây là kịch bản có mức độ xử lý đầu vào cao nhất. Sau khi giải nén, mỗi tệp được so sánh với tệp gốc tương ứng chỉ số Overlap. Chỉ những tệp đáp ứng ngưỡng tương đồng nhất định mới được giữ lại để huấn luyện. Điều này nhằm đảm bảo rằng dữ liệu đầu vào phản ánh đúng thông tin kỹ thuật ban đầu và không làm nhiễu mô hình trong quá trình học.
  - Mục tiêu kiểm chứng: Đây là điều kiện lý tưởng nhất, khi file được giải nén thành công và gần giống file gốc. Kiểm tra xem nếu công cụ giải nén hoạt động tốt, mô hình học máy có thể đạt được hiệu quả cao như khi dùng dữ liệu sạch ban đầu hay không.

Các kịch bản trên được xây dựng từ tình huống tệ nhất đến lý tưởng nhất, nhằm đánh giá dần hiệu quả mô hình học máy trong từng mức độ phục hồi thông tin sau giải nén, phản ánh đúng thực tế áp dụng mô hình trong các mức độ chất lượng dữ liệu khác nhau.

Hình 3.7 mô tả cách thức hoạt động của quá trình đào tạo máy học.

Mô hình học máy, kỹ thuật trích xuất đặc trưng, tập dữ liệu cụ thể và kết quả thực nghiệm chi tiết sẽ được giới thiệu trong Chương 4. Trong chương này, nhóm chỉ giới thiệu hướng chung và cấu trúc thử nghiệm để đặt nền tảng cho các bước triển khai kỹ thuật tiếp theo.



Hình 3.7: Sơ đồ tổng quan hóa quy trình thực thi giai đoạn 3

### 3.5.2 Quy trình thực hiện

Để đảm bảo đánh giá có hệ thống và khách quan, nhóm đã xây dựng một quy trình thực nghiệm bao gồm một loạt các bước, từ chuẩn bị dữ liệu đầu vào đến huấn luyện, sau đó so sánh hiệu suất theo các kịch bản mô hình khác nhau. Quy trình này không đi sâu vào cấu trúc kỹ thuật cụ thể, mà đóng vai trò là định hướng tổng thể cho quá trình triển khai ở chương sau.

Quy trình triển khai bao gồm bốn bước chính sau:

#### Bước 1: Chuẩn bị tập dữ liệu cho từng kịch bản

Dựa trên kết quả thu được trong giai đoạn giải nén (giai đoạn 2), nhóm tiếp tục tách và xây dựng ba tập dữ liệu huấn luyện, tương ứng với ba kịch bản đã đề cập ở trên:

- Tập dữ liệu 1: Chứa các tệp bị đóng gói, không giải nén.
- Tập dữ liệu 2: Chứa các tệp đã giải nén, không kiểm tra tính tương đồng.
- Tập dữ liệu 3: Chứa các tệp đã giải nén, thực hiện kiểm tra tính tương đồng và chỉ giữ lại các tệp đáp ứng yêu cầu.

Mỗi tệp trong ba tập dữ liệu được đánh dấu là "malicious" hoặc "benign".

#### Bước 2: Trích xuất đặc trưng tính

Sau các bước chuẩn hóa và gắn nhãn dữ liệu, nhóm đã trích xuất các đặc trưng từ các tệp thực thi PE, chuyển đổi dữ liệu nhị phân thành biểu diễn vector đặc trưng (feature vector) để huấn luyện mô hình học máy. Nhóm đã triển khai quy trình trích xuất đặc trưng bằng một đoạn script Python `extract_feature.py` do nhóm tự viết, sử dụng các thư viện chính như `pefile`, `pandas`, `Counter` và `array`. Script có khả năng xử lý hàng loạt các tệp tin .bin trong thư mục dữ liệu, trích xuất đặc trưng và lưu lại kết quả dưới dạng bảng .csv. Tổng thể, mỗi tệp tin PE sẽ được biểu diễn bằng một hàng

trong bảng đặc trưng, với các cột tương ứng là các đặc tính cụ thể (features). Quy trình trích xuất đặc trưng bao gồm ba nhóm thông tin chính:

- **Đặc trưng tổng quát (Generic features):** bao gồm độ entropy và kích thước tổng thể của tệp thực thi.
- **Đặc trưng cấu trúc PE (PE Header & Section features):** trích xuất từ header và các section như .text, .data, .rsrc, bao gồm entropy trung bình, số section, kích thước ảo (virtual size), đặc điểm (characteristics), v.v.
- **Đặc trưng API và DLL (API/DLL imports and exports):**
  - Toàn bộ các DLL được import sẽ được biểu diễn dưới dạng cột nhị phân (có hoặc không).
  - Các API phổ biến (xuất hiện trên 0.2% tổng số tệp) từ import/export cũng được mã hóa tương tự.

Tập đặc trưng đầu ra sẽ bao gồm cả:

- Các đặc trưng cố định (file size, PE headers, entropy, số lượng DLL/API),
- Các đặc trưng động (tập các API/DLL phổ biến được chọn theo tần suất xuất hiện trong dữ liệu).

Sự khác biệt trong chất lượng và cấu trúc đặc trưng đầu ra của ba kịch bản huấn luyện có được trình bày trong bảng 3.1.



Kịch bản	Mô tả tập dữ liệu	Số lượng đặc trưng	Ảnh hưởng
1	Tập thực thi bị nén	439	Một số đặc trưng bị che giấu: số lượng API thấp, entropy cao bất thường, nhiều trường trống hoặc sai lệch.
2	Tập đã được giải nén, bao gồm cả file tương đồng và không tương đồng với bản gốc	2973	Số lượng đặc trưng đầy đủ hơn, nhưng dữ liệu chứa nhiều do một số file unpack lỗi hoặc không đúng bản gốc.
3	Tập đã được giải nén, chỉ gồm những file tương đồng với bản gốc	2842	Chất lượng đặc trưng cao, phản ánh tốt hành vi thật của chương trình, đặc biệt ở phần API/DLL.

**Bảng 3.1:** Bảng so sánh sự khác nhau khi trích xuất đặc trưng giữa 3 kịch bản

Nhận xét:

- **Kịch bản 1** có entropy rất cao, cho thấy file có khả năng bị nén hoặc mã hóa mạnh → nhiều đặc trưng bị che giấu, số lượng API trích xuất được cực kỳ ít.
- **Kịch bản 2** cải thiện mạnh mẽ cả về số lượng API và độ chi tiết đặc trưng. Tuy nhiên, do không lọc file lỗi nên xuất hiện nhiều (một số đặc trưng có thể không phản ánh đúng hành vi).
- **Kịch bản 3** giữ lại hầu hết giá trị từ kịch bản 2 nhưng loại bỏ được dữ liệu nhiễu nhờ lọc theo hệ số Overlap, đảm bảo đặc trưng ổn định và tin cậy hơn cho huấn luyện.

Từ kết quả trên, có thể thấy việc trích xuất đặc trưng từ dữ liệu sau unpack là cần thiết và hiệu quả rõ rệt, giúp mô hình học máy có thêm thông tin quan trọng. Đặc biệt, việc lọc dữ liệu bằng tiêu chí tương đồng (Overlap 80%) trong kịch bản 3 không chỉ giảm nhiễu mà còn đảm bảo độ chính xác và tính đại diện của vector đặc trưng.

Sau khi giải nén, số lượng đặc trưng tăng đáng kể từ 439 (ở kịch bản 1) lên hơn 2.800 API (ở kịch bản 2 và kịch bản 3), cho thấy nhiều hành vi đã bị ẩn trước đó

đã được khôi phục. Một số API chỉ xuất hiện sau khi giải nén và có ảnh hưởng lớn đến quá trình huấn luyện gồm RegQueryValueW, MoveFileW, CreateDialogParamA, SetNamedSecurityInfoW, GetLastInputInfo... Những API này thường liên quan đến thao tác hệ thống, tương tác người dùng hoặc điều chỉnh bảo mật – đây là các hành vi đặc trưng của mã độc. Do đó, việc giải nén không chỉ giúp làm giàu vector đặc trưng mà còn khôi phục đúng các tín hiệu quan trọng giúp mô hình học máy phân loại hiệu quả hơn. Điều này lý giải tại sao các mô hình được huấn luyện trên dữ liệu đã giải nén thường đạt độ chính xác cao hơn.

### **Bước 3: Đào tạo và đánh giá các mô hình học máy**

Những file .csv chứa các đặc trưng đã được trích xuất của các file sẽ được sử dụng để huấn luyện một mô hình học máy chung. Chi tiết về mô hình và các tham số huấn luyện sẽ được giới thiệu trong chương tiếp theo, nhưng quy trình tổng thể bao gồm:

- Chia dữ liệu thành các tập huấn luyện và kiểm tra.
- Huấn luyện mô hình
- Đánh giá mô hình bằng các chỉ số như Accuracy, Precision, Recall, và F1-score.
- Lặp lại quy trình với nhiều mô hình khác nhau để kiểm chứng tính ổn định.

Sử dụng cùng một quy trình huấn luyện cho ba kịch bản đảm bảo rằng sự khác biệt về kết quả là do chất lượng dữ liệu đầu vào (có thực hiện giải nén hay không, có thực hiện lọc file tương đồng hay không), chứ không phải do thay đổi trong thuật toán hoặc tham số mô hình.

### **Bước 4: So sánh và phân tích kết quả**

Sau khi hoàn tất quy trình đào tạo, nhóm đã tổng hợp và so sánh hiệu suất của mô hình theo ba kịch bản.

# Chương 4

## KẾT QUẢ THỰC NGHIỆM

### 4.1 Bộ dữ liệu

Dataset được sử dụng cho phần này là tập dataset được công bố kèm theo bài báo "*When Malware is Packin' Heat; Limits of Machine Learning Classifiers Based on Static Analysis Features*" của Aghakhani và cộng sự [1] (có thể download dataset tại link sau: <https://github.com/ucsb-seclab/packware/tree/master>). Dataset bao gồm các tập tin PE được tổ chức thành các thư mục, trong đó mỗi thư mục tương ứng với một loại packer. Tên của packer được đặt làm tên thư mục, giúp dễ dàng phân loại và quản lý. Chi tiết dataset như sau:

- Dataset gốc:
  - Thư mục xpack: Chứa 9980 tệp đã được đóng gói bởi packer xpack.
  - Thư mục mew: Chứa 7350 tệp đã được đóng gói bởi packer mew.
  - Thư mục petite: Chứa 6530 tệp đã được đóng gói bởi packer petite.
  - Thư mục upx: Chứa 5200 tệp đã được đóng gói bởi packer upx.
- Dataset thực nghiệm:
  - Thư mục xpack: Chứa 2000 tệp đã được đóng gói bởi packer xpack.
  - Thư mục mew: Chứa 2000 tệp đã được đóng gói bởi packer mew.
  - Thư mục petite: Chứa 2000 tệp đã được đóng gói bởi packer petite.
  - Thư mục upx: Chứa 2000 tệp đã được đóng gói bởi packer upx.

Trong quá trình thử nghiệm, ban đầu nhóm đã lên kế hoạch sử dụng toàn bộ tập dữ liệu gốc. Tuy nhiên, trong giai đoạn triển khai thực nghiệm, nhóm phát hiện ra rằng một số lượng lớn tệp không thể được thực thi ngay từ đầu, điều này ảnh hưởng

trực tiếp đến việc đánh giá hiệu quả giải nén. Vì các tệp thực thi không được kiểm tra trước khi áp dụng pipeline nên nhóm không thể xác định nguyên nhân gây ra lỗi là do sự cố ở chính tệp gốc hay chính công cụ giải nén, dẫn đến việc cần phải kiểm tra lại dữ liệu bằng Cuckoo - một môi trường sandbox, điều này tốn thời gian và ảnh hưởng đến tiến trình chung.

Sau khi rút kinh nghiệm từ việc này, nhóm đã quyết định giới hạn tập dữ liệu thử nghiệm ở mức 2.000 tệp cho mỗi công cụ đóng gói, sàng lọc lựa chọn các tệp này chặt chẽ hơn để đảm bảo rằng các tệp này có thể thực thi được, đồng thời để các thư mục có một số lượng mẫu chung để có thể phản ánh chính xác hiệu quả của quá trình giải nén.

## 4.2 Kết quả giai đoạn 1 - Đánh giá khả năng phát hiện các tệp bị đóng gói

Mỗi công cụ sẽ hỗ trợ những loại packer khác nhau, nên nhóm chỉ thực hiện thực nghiệm và lấy số liệu đối với những packer mà tài liệu của công cụ xác nhận rằng có thể nhận biết được. Bảng 4.1 thể hiện khả năng phát hiện các loại packer được sử dụng trong tập dữ liệu của các công cụ.

	DIE	PEiD	ClamAV	Unipacker
xpack	✓	✓		
MEW	✓	✓	✓	✓
PETITE	✓	✓	✓	✓
UPX	✓	✓	✓	✓

**Bảng 4.1:** Các packer mà công cụ có thể phát hiện được

Với số lượng trình đóng gói mà nhóm sử dụng để thực nghiệm, DiE và PEiD đều có thể phát hiện được hết bởi dataset của hai công cụ này là rất lớn và được cập nhật liên tục. Còn ClamAV và Unipacker có vẻ kém lợi thế hơn một chút.

Đối với mỗi loại packer, mỗi công cụ lại có thời gian chạy và độ chính xác khác nhau. Nhằm làm rõ sự khác biệt này, nhóm thực hiện đã tiến hành đo lường thời gian chạy trung bình, tỷ lệ phát hiện và độ chính xác của từng công cụ đối với từng loại packer. Kết quả được trình bày chi tiết trong Bảng 4.2 dưới đây, thể hiện sự chênh lệch rõ rệt giữa các công cụ cả về hiệu năng và độ tin cậy trong việc phát hiện file được nén/đóng gói. Những khác biệt này là cơ sở quan trọng để đánh giá ưu – nhược điểm

của từng công cụ trong thực tiễn triển khai hệ thống phân tích mã độc, đặc biệt là khi lựa chọn công cụ phù hợp với các môi trường giới hạn tài nguyên, yêu cầu thời gian thực, hoặc mục tiêu phát hiện cụ thể.

		DIE	PEID	ClamAV	Unipacker
Thời gian chạy	xpack	0.20s	0.34s	✗	✗
	UPX	0.27s	0.41s	39.49s	0.22s
	MEW	0.24s	0.38s	39.32s	0.20s
	PEtite	0.25s	0.41s	27.76s	0.21s
	<b>Trung bình</b>	<b>0.24s</b>	<b>0.39s</b>	<b>35.52s</b>	<b>0.21s</b>
Phần trăm phát hiện	xpack	74.7%	94.20%	✗	✗
	UPX	100%	99.95%	87.41%	99.90%
	MEW	100%	100%	98.60%	99.70%
	PEtite	100%	100%	66.15%	99.74%
	<b>Trung bình</b>	<b>93.68%</b>	<b>98.54%</b>	<b>84.05%</b>	<b>99.78%</b>
Độ chính xác	xpack	99.67%	79.14%	✗	✗
	UPX	99.95%	96.60%	100%	100%
	MEW	100%	99.80%	100%	99.60%
	PEtite	99.95%	100%	100%	97.50%
	<b>Trung bình</b>	<b>99.89%</b>	<b>93.89%</b>	<b>100%</b>	<b>99.03%</b>

**Bảng 4.2:** Thời gian chạy và độ chính xác của từng công cụ khi detect file

*Nhận xét:*

- Unipacker nổi bật nhất:
  - Là công cụ nhanh nhất (0.21s trung bình), phù hợp cho xử lý khối lượng lớn.
  - Có tỷ lệ phát hiện cao nhất (99.78%), vượt trội hơn các công cụ khác.
  - Độ chính xác gần như tuyệt đối (99.03%), cho thấy kết quả tin cậy.
  - Là công cụ mạnh và toàn diện, phù hợp làm lựa chọn chính trong hệ thống phát hiện packer.

- DIE ấn tượng với độ chính xác:
  - Tuy thời gian chạy không phải nhanh nhất, nhưng vẫn ở mức tốt (0.24s).
  - Độ chính xác cao nhất trong tất cả các công cụ (99.89%), đặc biệt hiệu quả khi cần nhận diện đúng loại packer.
  - Tỷ lệ phát hiện đạt mức khá (93.68%), đủ đáp ứng các yêu cầu cơ bản.
- PEiD cân bằng:
  - Là công cụ có hiệu suất tốt và ổn định: thời gian xử lý nhanh (0.39s), phát hiện tốt (98.54%), độ chính xác cao (93.89%).
  - Phù hợp làm công cụ hỗ trợ để kiểm chứng chéo kết quả.
- ClamAV chưa thực sự phù hợp cho phát hiện packer:
  - Là công cụ có thời gian chạy lâu nhất ( 35 giây mỗi file), gây ảnh hưởng hiệu suất.
  - Tỷ lệ phát hiện thấp (84.05%) và thiếu dữ liệu đối với một số packer như xpack.
  - Mặc dù độ chính xác cao (100%) với các mẫu mà nó phát hiện, ClamAV có vẻ phù hợp hơn trong phân tích mã độc tổng thể, không chuyên sâu vào phát hiện packer.

### 4.3 Kết quả giai đoạn 2 - Đánh giá khả năng giải nén các tệp đã đóng gói

Trong số bốn công cụ mà nhóm nghiên cứu, chỉ có hai công cụ là ClamAV và Unipacker là có khả năng giải nén tệp tin. Nhằm giúp cho việc so sánh 2 công cụ, nhóm thực hiện đã tiến hành đo lường thời gian chạy trung bình, tỷ lệ phát hiện và độ chính xác của từng công cụ đối với từng loại packer. Kết quả được trình bày chi tiết trong Bảng 4.3 dưới đây.

*Nhận xét:*

PEiD và Detect It Easy (DIE) là các công cụ phân tích tĩnh có khả năng nhận diện packer chứ không thực hiện giải nén tệp thực thi. Chức năng của chúng chỉ dừng lại ở việc so khớp chữ ký với các mẫu đã biết để xác định loại packer đang được sử dụng, mà không hỗ trợ việc trích xuất mã gốc hay tái tạo lại tệp tin sau khi đã bị đóng gói.

		ClamAV	Unipacker
Thời gian chạy	xpack	<b>X</b>	<b>X</b>
	UPX	35.48s	17.35s
	MEW	38.83s	209.04s
	PEtite	0.01s	22.16s
	<b>Trung bình</b>	<b>24.78s</b>	<b>82.85s</b>
Phần trăm phát hiện	xpack	<b>X</b>	<b>X</b>
	UPX	87.41%	97.30%
	MEW	98.60%	93.90%
	PEtite	0.05%	94.50%
	<b>Trung bình</b>	<b>62.02%</b>	<b>95.23%</b>
Độ chính xác	xpack	<b>X</b>	<b>X</b>
	UPX	15.40%	59.48%
	MEW	73.39%	78.54%
	PEtite	0.00%	59.30%
	<b>Trung bình</b>	<b>29.6%</b>	<b>65.77%</b>

**Bảng 4.3:** Thời gian chạy và độ chính xác của từng công cụ khi unpack file

Dựa trên kết quả ở bảng 4.3, có thể thấy rằng Unipacker là công cụ có khả năng giải nén tốt nhất trong số các công cụ được khảo sát với tỷ lệ phát hiện trung bình lên tới 95.23% và độ chính xác trung bình 65.77%. Xét về thời gian giải nén, tuy ClamAV nhanh hơn nhưng chỉ đạt 62.02% phát hiện và 29.6% chính xác, thậm chí độ chính xác bằng 0 đối với packer PETITE.

Có thể thấy, Unipacker có thể giải nén nhiều định dạng packer khác nhau một cách ổn định, và mặc dù mất nhiều thời gian hơn, nhưng kết quả đầu ra có chất lượng tốt hơn và phù hợp hơn để sử dụng cho các bước phân tích tiếp theo.

- Lựa chọn công cụ phù hợp:
  - Unipacker là lựa chọn tốt nhất khi cần giải nén file thực thi để phục vụ phân tích sâu hoặc huấn luyện mô hình học máy.

- ClamAV có thể sử dụng cho kiểm tra nhanh nhưng cần thận trọng với kết quả.
- Độ chính xác gần như tuyệt đối (99.03%), cho thấy kết quả tin cậy.
- PEiD và DIE nên dùng như công cụ hỗ trợ bước đầu để nhận diện packer, không dùng để unpack.

#### **4.4 Kết quả so sánh tổng thể của các công cụ**

Kết hợp lý thuyết và kết quả trên, ta có Bảng 4.4 so sánh các công cụ như sau (chỉ xét trên các loại packer mà công cụ có thể phát hiện).



	<b>DIE</b>	<b>PEiD</b>	<b>ClamAV</b>	<b>Unipacker</b>
<b>Cài đặt dễ hay khó?</b>	Dễ	Dễ	Dễ	Dễ
<b>Môi trường sử dụng</b>	Windows, Linux, macOS	Windows, Linux	Windows, Linux, macOS	Linux
<b>Ưu điểm</b>	Phát hiện heuristic và signature linh hoạt	Cơ sở dữ liệu signature phong phú; tốc độ nhanh; giao diện trực quan	Hỗ trợ phát hiện virus và malware trên nhiều nền tảng	Hỗ trợ phân tích runtime
<b>Nhược điểm</b>	Không hỗ trợ giải nén	Phụ thuộc vào signature, không phát hiện được các packer mới	Tập trung vào malware hơn là phân tích tệp nén	Chỉ hỗ trợ phát hiện một số loại packer cụ thể
<b>Có phát hiện file nén không?</b>	Có thể phát hiện 93.68% file nén	Có thể phát hiện 98.54% file nén	Có thể phát hiện 84.05% file nén	Có thể phát hiện 99.78% file nén
<b>Phát hiện đúng hay sai?</b>	Đúng 99.89%	Đúng 93.89%	Đúng 100%	Đúng 99.03%
<b>Thời gian detect file</b>	Nhanh	Nhanh	Chậm	Nhanh
<b>Có thể giải nén không?</b>	Không	Có thể nếu dùng plugins và chỉ 1 ít packer	Có thể nhưng khả năng giải nén kém	Có thể tương đối tốt với 1 vài packer
<b>Thời gian giải nén file</b>	Không giải nén	Không giải nén	Nhanh	Chậm
<b>Độ chính xác khi giải nén</b>	Không giải nén	Không giải nén	Khá tệ	Tốt

**Bảng 4.4:** So sánh tổng hợp của các công cụ

*Nhận xét:*

Bảng 4.4 cho thấy kết quả so sánh bốn công cụ thường dùng để phát hiện và giải nén các tệp thực thi được đóng gói: Detect It Easy (DIE), PEiD, ClamAV và Unipacker. Tiêu chí đánh giá bao gồm khả năng phát hiện packer, độ chính xác khi phát hiện, khả năng giải nén, thời gian xử lý (thời gian phát hiện và giải nén file), đồng thời còn có các tiêu chí như: tính thân thiện với người dùng (cài đặt dễ hã y khó), môi trường sử dụng,...

Nhìn chung, tất cả các công cụ đều tương đối dễ cài đặt và sử dụng, hỗ trợ nhiều nền tảng (Windows, Linux và macOS), nhưng có sự khác biệt đáng kể về khả năng giải nén và độ chính xác trong việc xác định các chương trình bị đóng gói.

Cụ thể:

- Điểm mạnh của DIE nằm ở khả năng hỗ trợ phát hiện heuristic và signature linh hoạt, tốc độ xử lý nhanh, nhưng không hỗ trợ giải nén. Công cụ này phù hợp để phát hiện nhanh các tệp có dấu hiệu được đóng gói, nhưng không thể sử dụng cho các quy trình giải nén.
- PEiD có giao diện trực quan và có thể phát hiện chính xác các chương trình được đóng gói theo signature, nhưng công cụ này phụ thuộc nhiều vào các plugin và chỉ hỗ trợ giải nén một số chương trình được đóng gói nhất định - điều này hạn chế khả năng sử dụng của công cụ này trong các môi trường tự động.
- ClamAV là một công cụ diệt vi-rút nguồn mở có tỷ lệ phát hiện cao và độ chính xác tuyệt đối trong các thử nghiệm, nhưng khả năng hỗ trợ giải nén còn hạn chế. Ngoài ra, thời gian xử lý lâu của ClamAV khiến nó không phù hợp với các pipeline hiệu suất cao.
- Unipacker là công cụ duy nhất trong nhóm có khả năng giải nén thực sự, hỗ trợ giải nén nhiều trình đóng gói phổ biến. Mặc dù tỷ lệ phát hiện không cao tuyệt đối, nhưng nó giải nén thành công, có chất lượng tốt, xử lý nhanh và có thể được tích hợp vào pipeline.

## **4.5 Kết quả giai đoạn 3 - Đánh giá hiệu quả của việc đào tạo các mô hình học máy**

### **4.5.1 Trích xuất đặt trưng**

Sau khi phân tích kỹ lưỡng các nghiên cứu liên quan, nhóm đã trích xuất những nhóm đặc trưng tĩnh (static analysis features) sau:

- PE Header: Trích xuất các đặc trưng từ `pe.OPTIONAL_HEADER` (gồm 1 số đặc trưng như: `SizeOfHeaders`, `AddressOfEntryPoint`, `ImageBase`,...) và đặc trưng từ `pe.FILE_HEADER` (gồm các đặc trưng như: `NumberOfSections` và `SizeOfOptionalHeader`).
- PE Sections: Với mỗi section, trích xuất các đặc trưng như: `size`, `virtualSize`, `entropy`, `virtualAddress`.
- DLL Imports: Tạo đặc trưng nhị phân cho mỗi thư viện DLL được sử dụng.
- API Imports: Tương tự như DLL, tạo đặc trưng nhị phân cho mỗi API được sử dụng.

## 4.5.2 Huấn luyện và mô hình học máy

### 1. Random Forest

Sau khi chia tập dữ liệu thành hai phần theo tỷ lệ 70:30, với quy mô dữ liệu và số lượng đặc trưng ở mức ít, chúng tôi huấn luyện mô hình với tham số `n_estimators=100` và `max_depth=7` như một cấu hình tối ưu hóa cân bằng giữa độ chính xác và tốc độ huấn luyện.

Sau khi huấn luyện mô hình, chúng tôi phân tích độ quan trọng của đặc trưng bằng thuộc tính `feature_importances_` của Random Forest. Các đặc trưng được sắp xếp theo mức độ ảnh hưởng đến quyết định của mô hình. Trong thực nghiệm, chúng tôi lựa chọn 20 đặc trưng quan trọng nhất để đánh giá lại mô hình sau khi giảm chiều dữ liệu.

### 2. XGBoost

Trước tiên, chúng tôi chuẩn hóa dữ liệu nhãn bằng cách mã hóa nhị phân: `benign = 0`, `malicious = 1` để phù hợp với yêu cầu đầu vào của mô hình XGBoost.

Sau đó, chúng tôi áp dụng phương pháp chọn lọc đặc trưng có giám sát sử dụng thống kê chi-squared ( $\chi^2$ ). Chúng tôi chọn ra 20 đặc trưng quan trọng nhất có liên quan cao với nhãn mục tiêu.

Mô hình XGBoost được cấu hình với các siêu tham số như sau:

- `n_estimators`: 100
- `max_depth`: 4
- `learning_rate`: 0.05
- `subsample`: 0.8
- `gamma`: 0.7

- `reg_lambda: 5`
- `eval_metric: 'logloss'`

### 3. K-Nearest Neighbors

Giống với XGBoost, nhãn dữ liệu cũng được mã hóa thành giá trị nhị phân và sử dụng phương pháp thông kê chi-squared để lựa chọn 20 đặc trưng quan trọng nhất.

Chúng tôi sử dụng kỹ thuật cross-validation ( $k\text{-fold} = 5$ ) để tìm giá trị  $k$  tối ưu cho số lượng hàng xóm gần nhất. Thang giá trị từ  $k = 3$  đến  $k = 10$  được kiểm tra và mô hình được đánh giá bằng chỉ số F1-macro nhằm đảm bảo độ công bằng giữa các lớp.

Sau khi xác định được  $k$  tối ưu, mô hình KNN được huấn luyện với các tham số như sau:

- `n_neighbors: k` tốt nhất vừa tìm được
- `weights: 'distance'`
- `p: 2`
- `metric: 'minkowski'`

### 4. Support Vector Machine

Để huấn luyện mô hình, nhãn được ánh xạ nhị phân như những mô hình trước, sau đó sử dụng phương pháp SelectKBest với thông kê chi-squared để chọn ra 20 đặc trưng quan trọng nhất.

Vì SVM rất nhạy cảm với tỷ lệ của đặc trưng, nên chúng tôi tiến hành chuẩn hóa dữ liệu đầu vào bằng phương pháp z-score normalization (chuẩn hóa trung bình 0, độ lệch chuẩn 1). Sau đó, tập dữ liệu được chia theo tỷ lệ 80:20 để huấn luyện và đánh giá mô hình.

Mô hình SVC (SVM classifier) sử dụng kernel Gaussian (RBF) với các tham số đã tinh chỉnh thủ công:

- `C: 10`
- `gamma: 0.1`
- `kernel: 'rbf'`

#### 4.5.3 Kịch bản 1: Huấn luyện máy học với các file bị nén

Kết quả của việc huấn luyện mô hình được thể hiện đầy đủ ở Bảng 4.5.

	Random Forest	XGBoost	KNN	SVM
Accuracy	0.89	0.85	0.85	0.79
Precision	0.87	0.83	0.86	0.79
Recall	0.91	0.87	0.84	0.77
F1 Score	0.89	0.85	0.85	0.78

**Bảng 4.5:** Kết quả của các mô hình kịch bản 1

Trong kịch bản 1, dữ liệu là các file PE bị nén bởi packer, nghĩa là:

- Đặc trưng quan trọng của file gốc (như chuỗi API, cấu trúc section, byte entropy...) đã bị làm méo mó hoặc ẩn đi.
- Một số thông tin phân biệt giữa các lớp (benign/malicious hoặc các packer khác nhau) bị che giấu hoặc biến dạng.

Kết quả khá tốt, nhưng không phải cao nhất vì mô hình học máy học trực tiếp trên đặc trưng bị biến đổi bởi packer → một số đặc trưng quan trọng có thể bị ẩn đi:

- Accuracy dao động từ 0.79–0.89.
- Recall và Precision khá cao với Random Forest (0.91, 0.87) nhưng thấp ở SVM (0.77, 0.79) do Random Forest có khả năng xử lý tốt dữ liệu nhiễu và tự động chọn đặc trưng quan trọng, trong khi SVM kém hiệu quả vì phụ thuộc vào ranh giới phân lớp rõ ràng).

#### 4.5.4 Kịch bản 2: Huấn luyện máy học với các file đã giải nén (bao gồm cả file tương đồng với file ban đầu và file không tương đồng)

Kết quả của việc huấn luyện mô hình được thể hiện đầy đủ ở Bảng 4.6.

	Random Forest	XGBoost	KNN	SVM
Accuracy	0.90	0.90	0.87	0.79
Precision	0.87	0.92	0.88	0.79
Recall	0.91	0.86	0.85	0.77
F1 Score	0.89	0.89	0.86	0.78

**Bảng 4.6:** Kết quả của các mô hình kịch bản 2

Trong kịch bản 2, dữ liệu được sử dụng để huấn luyện là các file PE đã được giải nén bằng Unipacker, bao gồm cả file tương đồng (giải nén tốt, gần giống file gốc) và file không tương đồng (giải nén không hoàn hảo).

- Việc giải nén giúp phục hồi một phần đặc trưng bị ẩn, như chuỗi API, cấu trúc vùng nhớ... → giúp mô hình học được thông tin rõ ràng hơn so với dữ liệu bị nén.
- Tuy nhiên, do có cả file giải nén không hoàn hảo, nên dữ liệu có thể bị nhiễu, làm ảnh hưởng đến độ ổn định và hiệu năng mô hình.

Kết quả cải thiện so với kịch bản 1 ở hầu hết các mô hình, đặc biệt là XGBoost và KNN:

- Accuracy tăng lên đến 0.90 với Random Forest và XGBoost.
- Precision của XGBoost đạt 0.92 – cao nhất trong cả ba kịch bản.
- Tuy nhiên, F1 Score của SVM vẫn thấp (0.78) do vẫn bị ảnh hưởng bởi dữ liệu không nhất quán từ các file giải nén không thành công.

#### 4.5.5 Kịch bản 3: Huấn luyện máy học với file đã giải nén (chỉ với file tương đồng với file ban đầu)

Kết quả của việc huấn luyện mô hình được thể hiện đầy đủ ở Bảng 4.7.

	Random Forest	XGBoost	KNN	SVM
Accuracy	0.88	0.88	0.84	0.81
Precision	0.86	0.88	0.87	0.83
Recall	0.93	0.91	0.84	0.83
F1 Score	0.89	0.90	0.85	0.83

**Bảng 4.7:** Kết quả của các mô hình kịch bản 3

Trong kịch bản 3, dữ liệu huấn luyện là các file PE đã được giải nén tốt (tương đồng với file gốc), nghĩa là:

- Chỉ sử dụng những file mà quá trình giải nén bằng Unipacker cho kết quả chất lượng cao, với chuỗi API và đặc trưng tương tự file gốc.
- Tập dữ liệu trở nên sạch, nhất quán, ít nhiễu hơn nhiều so với kịch bản 2.

Kết quả:

- Mô hình đạt hiệu suất cao và ổn định nhất:
  - Accuracy và F1 Score đều cao trên hầu hết các mô hình (F1 0.83).
  - Các chỉ số giữa các mô hình gần nhau hơn, cho thấy dữ liệu có tính phân lớp rõ ràng.
- Lý do:
  - Dữ liệu nhất quán và giàu đặc trưng phân biệt giúp tất cả mô hình, từ đơn giản (KNN) đến phức tạp (XGBoost, RF), học hiệu quả hơn.
  - SVM cải thiện rõ rệt vì tập dữ liệu đã được lọc bỏ nhiễu → ranh giới giữa các lớp rõ ràng hơn, phù hợp với cơ chế hoạt động của SVM.
  - XGBoost và KNN tiếp tục duy trì hiệu năng cao nhờ khai thác triệt để các đặc trưng chính xác từ file giải nén tốt.

#### 4.5.6 Tổng kết

Từ kết quả thu được ở trên, nhóm đã tổng hợp lại thành bảng 4.8

Kịch bản	Chỉ số	Random Forest	XGBoost	KNN	SVM
Kịch bản 1	Accuracy	0.89	0.85	0.85	0.79
	Precision	0.87	0.83	0.86	0.79
	Recall	0.91	0.87	0.84	0.77
	F1 Score	0.89	0.85	0.85	0.78
Kịch bản 2	Accuracy	0.90	0.90	0.87	0.79
	Precision	0.87	0.92	0.88	0.79
	Recall	0.91	0.86	0.85	0.77
	F1 Score	0.89	0.89	0.86	0.78
Kịch bản 3	Accuracy	0.88	0.88	0.84	0.81
	Precision	0.86	0.91	0.87	0.83
	Recall	0.93	0.91	0.84	0.83
	F1 Score	0.89	0.90	0.85	0.83

**Bảng 4.8:** So sánh hiệu quả các mô hình học máy theo từng kịch bản huấn luyện

Quan sát bảng 4.8 có thể thấy hiệu quả của các mô hình học máy thay đổi khá rõ giữa ba kịch bản.

- Trong kịch bản 1, mô hình Random Forest hoạt động tốt nhất với Recall đạt được đến 0.91, độ chính xác và F1 score đều đạt đến 0.89. XGBoost và KNN có kết quả tương đồng với nhau, tuy nhiên SVM lại có kết quả kém hơn đáng kể ở mọi chỉ số. Điều này cho thấy các file bị đóng gói không đủ thông tin về các đặc trưng để SVM hoạt động hiệu quả, dẫn đến kết quả chênh lệch nhau khá nhiều ở các mô hình.
- Trong kịch bản 2, nhóm đã thấy sự cải thiện đáng kể khi họ sử dụng các tệp đã giải nén để đào tạo. Cả Random Forest và XGBoost đều đạt được độ chính xác là 0.90. Đồng thời cả hai đều có sự cải thiện ở các chỉ số.
- Đến kịch bản 3, khi nhóm chỉ dùng các tệp unpacked có tính tương đồng cao với tệp gốc, tuy chỉ số của các mô hình không tăng lên đáng kể, nhưng SVM lại cải thiện kết quả rõ rệt trong kịch bản này, dù kết quả rất tệ trong 2 kịch bản trước.



Từ ba kịch bản trên có thể rút ra kết luận quan trọng:

Quá trình giải nén rõ ràng có ảnh hưởng tích cực đến hiệu quả của mô hình phát hiện mã độc, đặc biệt với các mô hình học máy tĩnh. Việc unpack giúp khôi phục lại các đặc trưng bị che giấu do đóng gói, từ đó cải thiện cả độ chính xác, độ bao phủ (recall) và độ tin cậy của mô hình. Điều này khẳng định vai trò thiết yếu của bước tiền xử lý unpack trong các hệ thống phân tích mã độc hiện đại.

## Chương 5

# KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

### 5.1 Kết luận

Trong lĩnh vực phát hiện và giải nén các tệp tin thực thi Windows bị đóng gói, cả bốn công cụ Unipacker, ClamAV, Detect It Easy (DIE) và PEiD đều đã khẳng định được giá trị và khả năng vượt trội của mình. Mỗi công cụ đều mang đến những dòng lệnh, những cách thức và phương pháp riêng biệt nhưng nhìn chung, chúng đều hướng tới một mục đích chung, đó là hỗ trợ các chuyên gia bảo mật, các nhà nghiên cứu, các doanh nghiệp và cá nhân thực hiện các nhiệm vụ như phát hiện, giải mã và phân tích mã độc.

Unipacker là công cụ chuyên dụng cho việc giải nén các tệp tin bị đóng gói. Với khả năng tự động hóa cao, nó giảm thiểu đáng kể thời gian xử lý thủ công, giúp các chuyên gia nhanh chóng tiếp cận với mã nguồn gốc của các tệp tin để phân tích sâu hơn. Tuy nhiên, Unipacker đôi khi có thể gặp khó khăn với các tệp tin sử dụng trình đóng gói mới hoặc ít phổ biến hơn, điều này đòi hỏi sự kết hợp với các công cụ khác để đạt được hiệu quả tối ưu.

ClamAV nổi tiếng với khả năng phát hiện mã độc và xử lý các tệp tin bị nhiễm, không chỉ dừng lại ở chức năng quét mà còn hỗ trợ giải nén một số tệp tin bị đóng gói. Ưu điểm lớn của ClamAV là khả năng cập nhật cơ sở dữ liệu liên tục, giúp nó nhận diện các mối đe dọa mới một cách nhanh chóng. Và đặc biệt, trong việc giải nén, Clamav vẫn chưa thực sự thuận thực và khiến cho việc truy xuất dữ liệu gốc vẫn còn khó khăn.

Detect It Easy (DIE) lại tập trung vào việc nhận diện định dạng tệp tin và các trình đóng gói, cryptor, hoặc compiler. Công cụ này nổi bật nhờ giao diện thân thiện, khả năng tùy chỉnh cao, và hỗ trợ mở rộng với các plugin. Detect It Easy cung cấp thông tin chi tiết về cấu trúc tệp tin, giúp người dùng nắm bắt được những đặc điểm quan

trọng mà không cần đi sâu vào phân tích mã. Điều này làm cho DIE trở thành một phần không thể thiếu trong việc phát hiện các trình đóng gói phổ biến và ít phổ biến hơn.

PEiD là một công cụ đã quá quen thuộc trong việc phát hiện các trình đóng gói nhờ khả năng nhận diện hàng trăm signature khác nhau. Với ba chế độ quét linh hoạt – Normal, Deep, và Hardcore – PEiD có thể xử lý cả các tệp tin bị chỉnh sửa hoặc biến đổi phức tạp. Bên cạnh đó, các tính năng như xem thông tin chi tiết của tệp tin PE, tích hợp trình xem hex, và hỗ trợ plugin giúp PEiD mở rộng khả năng phân tích của mình. Dù đã ra mắt từ lâu, công cụ này vẫn giữ được sự hữu ích nhờ tính ổn định và hiệu quả.

Việc giải nén các tệp tin bị đóng gói không chỉ có ý nghĩa trong quá trình phân tích thủ công, mà còn đóng vai trò quan trọng trong việc nâng cao hiệu quả của các mô hình học máy trong lĩnh vực an ninh mạng. Các tệp PE sau khi được giải nén thành công sẽ khôi phục lại nhiều đặc trưng quan trọng vốn bị che giấu bởi các trình đóng gói — chẳng hạn như chuỗi API, cấu trúc section, hoặc entropy vùng mã. Những đặc trưng này là yếu tố cốt lõi để các mô hình học máy có thể phân loại, nhận diện hoặc phát hiện mã độc một cách chính xác hơn.

Bên cạnh đó, giải nén cũng đóng vai trò như một bước tiền xử lý quan trọng, giúp chuyển đổi dữ liệu từ dạng khó phân tích (do nén) sang dạng dễ khai thác hơn. Việc giải nén không chỉ là bước hỗ trợ về mặt kỹ thuật, mà còn mang tính chiến lược trong việc khai thác và huấn luyện các hệ thống thông minh phát hiện mã độc hiện đại, góp phần tăng cường khả năng phòng thủ và phản ứng nhanh trước các mối đe dọa ngày càng tinh vi.

## 5.2 Hướng phát triển

Nhìn chung, mặc dù mỗi công cụ đều có điểm mạnh và hạn chế riêng, sự kết hợp của chúng giúp khắc phục được nhiều vấn đề trong việc phát hiện và giải nén tệp tin thực thi bị đóng gói. Sự phát triển không ngừng của công nghệ đóng gói và mã hóa tệp tin đòi hỏi các công cụ này phải được cải tiến liên tục. Tuy nhiên, với những tính năng hiện tại, chúng đã và đang hỗ trợ đắc lực cho các chuyên gia bảo mật, góp phần đảm bảo an toàn thông tin trước những mối đe dọa ngày càng tinh vi và đa dạng.

Mặc dù các công cụ như Unipacker, ClamAV, Detect It Easy (DIE) và PEiD đã chứng minh được hiệu quả nhất định trong việc phát hiện và giải nén các tệp tin thực thi bị đóng gói, tuy nhiên, mỗi công cụ lại có những điểm mạnh và hạn chế riêng. Việc khai thác hiệu quả từng công cụ ở những bước phù hợp trong quy trình phân tích thay vì

sử dụng độc lập là một hướng phát triển thực tế và cần thiết, đặc biệt trong bối cảnh dữ liệu ngày càng phức tạp và tấn công mạng ngày càng tinh vi.

Một trong những hướng đi tiềm năng là xây dựng một hệ thống tích hợp các công cụ này thành một pipeline tự động, nơi các thành phần hỗ trợ lẫn nhau theo từng bước hợp lý. Cụ thể:

- PEiD và Detect It Easy có thể đảm nhận nhiệm vụ nhận diện trình đóng gói bằng cách đối chiếu signature hoặc phân tích cấu trúc PE.
- Unipacker được sử dụng để giải nén các tệp tin đã được nhận diện — nhờ khả năng tự động và tốc độ xử lý cao.
- ClamAV có thể đóng vai trò kép: vừa giải nén với một số định dạng hỗ trợ, vừa quét mã độc trên các tệp gốc hoặc đã giải nén.

Từ kết quả thực nghiệm các kịch bản huấn luyện mô hình học máy trước đó, có thể thấy rằng việc giải nén thành công các tệp tin đóng gói giúp khôi phục lại đặc trưng quan trọng, từ đó làm tăng đáng kể hiệu suất và độ chính xác của mô hình học máy. Vì vậy, tích hợp giải nén chất lượng vào giai đoạn tiền xử lý là điều cực kỳ quan trọng trong các hệ thống phát hiện tự động.

Ngoài ra, một hệ thống hoàn chỉnh có thể được phát triển với giao diện người dùng trực quan (GUI), cho phép người dùng dễ dàng kéo – thả tệp tin, theo dõi kết quả quét và phân tích từng bước một cách thuận tiện. Hệ thống này cũng nên hỗ trợ quét hàng loạt (batch scan), phù hợp với nhu cầu xử lý số lượng lớn tệp tại các trung tâm phân tích hoặc doanh nghiệp an ninh mạng. Việc tích hợp với các hệ thống giám sát an ninh (SIEM) sẽ giúp phát hiện và phản ứng kịp thời trước các mối đe dọa trong thời gian thực. Bên cạnh đó, cần trang bị cơ chế cập nhật signature tự động từ các nguồn uy tín hoặc mở rộng theo hướng cộng đồng đóng góp, nhằm đảm bảo khả năng nhận diện các trình đóng gói và mã độc mới một cách hiệu quả và liên tục.

Thay vì chỉ tập trung vào việc lựa chọn từng công cụ riêng lẻ, hướng phát triển hiệu quả hơn là xây dựng một hệ thống tích hợp, nơi các công cụ có thể phối hợp với nhau một cách linh hoạt và tự động. Khi đó, quá trình giải nén không chỉ trở thành bước hỗ trợ kỹ thuật, mà còn đóng vai trò quan trọng trong việc nâng cao hiệu quả phân tích mã độc và cải thiện chất lượng dữ liệu đầu vào cho các mô hình học máy. Một hệ thống như vậy, nếu được thiết kế thông minh, dễ sử dụng và luôn cập nhật, chắc chắn sẽ mang lại giá trị thực tiễn cao cho cả chuyên gia bảo mật lẫn hệ thống phòng thủ tự động.

# TÀI LIỆU THAM KHẢO

- [1] Hojjat Aghakhani et al. “When Malware is Packin’ Heat; Limits of Machine Learning Classifiers Based on Static Analysis Features”. In: *Network and Distributed Systems Security (NDSS) Symposium*. Internet Society, 2020.
- [2] *ASPack - exe file compressor allows to protect application from reverse engineering decompilation and disassemblers*. Truy cập: 2024-06-20. url: <https://www.aspack.com/>.
- [3] BLUE<sub>T</sub>EAM. *BÁO CÁO TÌNH HÌNH ATTT (06 tháng đầu năm 2024)*. Chia sẻ kỹ thuật. Truy cập: 2024-06-20. VNPT Cyber Immunity, 2024.
- [4] R. Crew. *Phân biệt vai trò của Packer, Crypter, và Protector trong các phần mềm độc hại*. Truy cập: 2024-06-20. 2020. url: [insecclab.uit.edu.vn](http://insecclab.uit.edu.vn).
- [5] Daniel Gibert et al. “Assessing the impact of packing on static machine learning-based malware detection and classification systems”. In: *Computers & Security* 156 (2025), p. 104495.
- [6] MEW. Truy cập: 2024-06-20. 2004. url: <http://www.softpedia.com/>.
- [7] Trivikram Muralidharan et al. “File Packing from the Malware Perspective: Techniques, Analysis Approaches, and Directions for Enhancements”. In: *ACM Computing Surveys* 55.1 (2022), pp. 1–48.
- [8] requaos. *mPress, Github*. Truy cập: 2024-06-20. 2009. url: <https://github.com/requaos/mPress>.
- [9] C. Team. *ClamAV*. Truy cập: 2024-06-20. url: <https://www.clamav.net/>.
- [10] The UPX Team. *UPX*. Truy cập: 2024-06-20. 2024. url: <https://upx.github.io/>.
- [11] *Unipacker: Automatic and platform-independent unpacker for Windows binaries based on emulation, Github*. Truy cập: 2024-06-20. 2014. url: <https://github.com/unipacker/>.