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

NGUYỄN THÁI DƯƠNG
NGUYỄN TRẦN ĐỨC ANH

ĐỒ ÁN CHUYÊN NGÀNH

PHƯƠNG PHÁP PHÁT HIỆN MÃ ĐỘC NÉN TRÊN NỀN TẢNG WINDOWS

A METHOD OF PACKED WINDOWS MALWARE DETECTION

SINH VIÊN NGÀNH AN TOÀN THÔNG TIN

GIẢNG VIÊN HƯỚNG DẪN:
ThS. Phan Thế Duy

TP.Hồ Chí Minh - 2024

LỜI CẢM ƠN

Trong quá trình nghiên cứu và hoàn thành khóa luận, nhóm đã nhận được sự định hướng, giúp đỡ, các ý kiến đóng góp quý báu và những lời động viên của các giáo viên hướng dẫn và giáo viên bộ môn. Nhóm xin bày tỏ lời cảm ơn tới thầy Phan Thế Duy đã tận tình trực tiếp hướng dẫn, giúp đỡ trong quá trình nghiên cứu.

Nguyễn Thái Dương **Nguyễn Trần Đức Anh**

MỤC LỤC

DANH MỤC CÁC KÝ HIỆU, CÁC CHỮ VIẾT TẮT

Ký hiệu	Giải thích
y	Label
w	Weak Learner
W	Tổng các Weak Learner
c	Điểm tự tin của Weak Learner
η	Tốc độ học
θ	Trọng số cho mạng nơ-ron
$L(\theta)$	Hàm mất mát
PE	Portable Executable
DLL	Dynamic Link Library
D(x)	Ước tính của bộ phân biệt cho dữ liệu đầu vào là x
$\mathbb{E}_x$	Kỳ vọng trên trường hợp dữ liệu thật
G(z)	Đầu ra của bộ sinh với đầu vào z
D(G(z))	Ước tính của bộ phân biệt cho dữ liệu đầu vào từ bộ sinh
$\mathbb{E}_z$	Kỳ vọng trên trường hợp dữ liệu giả

DANH MỤC CÁC HÌNH VẼ

DANH MỤC CÁC BẢNG BIỂU

TÓM TẮT ĐỒ ÁN

Tính cấp thiết của đề tài nghiên cứu:

Trong những năm gần đây, những đề tài nghiên cứu về mã độc đã không ngừng cải tiến các. Các công trình này đã tích hợp thành công mô hình học máy vào việc phát hiện và phân loại các mẫu mã độc. Tuy nhiên, những cải tiến vượt bậc trong lĩnh vực này, lại đi kèm với nhiều thách thức khi thủ thuật tấn công cũng trở nên tinh vi hơn. Chẳng hạn như nghiên cứu của Aghakhani và cộng sự đã chỉ ra rằng việc nén mã độc bằng những bộ nén sẽ khiến cho việc mô hình nhận diện mã độc bị sai. Đồng thời việc mô hình nhận diện mã độc có khả năng chống lại tấn công đối kháng cũng là một thách thức.

Sau khi nghiên cứu hướng sử dụng học máy và XAI để nhận diện mã độc nén, chúng tôi nhận thấy đây là một đề tài có tiềm năng. Nhóm cũng nhận thấy hầu hết các công trình nhận diện mã độc nén đều gặp nhiều hạn chế.

CHƯƠNG 1. TỔNG QUAN

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

1.1. Giới thiệu vấn đề

Trong bối cảnh mã độc đang trở nên thịnh hành hơn, cũng vì vậy các bài nghiên cứu về ứng dụng các kỹ thuật học máy để phát hiện và phân loại mã độc được ra đời[4]. Việc áp dụng học máy giúp cho quá trình tự động hóa và phát hiện mã độc trở nên dễ dàng hơn, một số nghiên cứu về mô hình nhận diện mã độc xoay quanh bộ dữ liệu EMBER[3] có tỷ lệ phát hiện mã độc rất cao, chẳng hạn như mô hình DeepMalNet[7] có kết quả phát hiện mã độc lên đến 98%.

Tuy nhiên, để qua mặt các công cụ phân tích mã độc tĩnh, phần mềm độc hại có thể sử dụng phương pháp nén. Công trình nghiên cứu trong bài báo "When Malware is Packin' Heat"[1] của nhóm Aghakhani chỉ ra rằng những bộ nén lưu giữ những thông tin "hữu ích" cho việc nhận diện mã độc, tuy nhiên những thông tin này không thực sự hữu ích trong việc phân tích hành vi của mã độc, công trình này nghiên cứu về những hạn chế của việc sử dụng học máy trong việc phân tích thuộc tính tĩnh. Kết quả từ những nghiên cứu này chỉ ra rằng, việc trích xuất thuộc tính từ các tệp nén là không đủ để mô hình học máy có thể nhận diện được những bộ nén mới, chưa từng xuất hiện trước đây, áp dụng học máy một cách "ngây thơ" sẽ làm tăng đáng kể các kết quả dương tính giả, từ đó việc nhận diện những file mã độc đã được nén trong thực tế sẽ không chính xác.

Sau khi thực nghiệm lại các nghiên cứu về việc áp dụng học máy để nhận diện mã độc, nhóm nhận thấy rằng những bộ nén khác nhau sẽ có cách trích xuất

đặc trưng khác nhau, trong đó những thuộc tính như file size khá quan trọng trong viện nhận dạng những mẫu đã được nén. Ngoài ra, những mô hình học máy chỉ có khả năng nhận diện được những bộ nén đã được học hoặc những bộ nén có chức năng gần giống. Nhóm chúng tôi quyết định áp dụng Explainable AI[6] để đánh giá lại những thuộc tính quan trọng trong việc xác định kết quả nhận diện mã độc của những mô hình học máy, đồng thời ứng dụng thêm việc tạo những mẫu đối kháng[2] để quan sát kết quả của mô hình học máy sau khi bị tấn công.

1.2. Giới thiệu những nghiên cứu liên quan

1.2.1. Trình phát hiện mã độc nén

Những trình phát hiện mã độc nén mà nhóm xây dựng đều được áp dụng phương pháp phân tích tĩnh và được học trên 2 bộ dữ liệu là Wild dataset và Lab dataset được cung cấp trong công trình nghiên cứu của nhóm tác giả Aghakhani[1]. Bộ dữ liệu Wild dataset gồm có 50724 mẫu và 56543 thuộc tính. Bộ dữ liệu Lab dataset là bộ dữ liệu được nén từ bộ dữ liệu Wild dataset bằng 10 bộ nén khác nhau với 341444 mẫu.

1.2.2. Explainable Artificial Intelligence

Explainable Artificial Intelligence (XAI) là các kỹ thuật được sử dụng để diễn giải cho người dùng hiểu hơn về quá trình đưa ra quyết định và cách thức hoạt động của một mô hình học máy từ đó có thể cải thiện được hiệu xuất của mô hình dựa trên những kết quả đó.

1.2.3. Mẫu đối kháng

Mẫu đối kháng là những trường hợp dữ liệu được chỉnh sửa, sao cho các mô hình học máy, đặc biệt là các mô hình học sâu như mạng nơ-ron đưa ra dự đoán

sai lầm. Những thay đổi này thường rất nhỏ, thậm chí không thể nhận thấy bằng mắt thường, nhưng lại có thể đánh lừa các mô hình này một cách hoàn toàn.

1.3. Mục tiêu, đối tượng, và phạm vi nghiên cứu

1.3.1. Mục tiêu nghiên cứu

Nghiên cứu về khả năng nhận diện mã đọc nén của mô hình học máy và chỉ ra được những hạn chế của mô hình học máy trong việc nhận diện mã độc nén.

1.3.2. Đối tượng nghiên cứu

Đối tượng nghiên cứu:

- Mã độc
- Trình phát hiện mã độc dựa trên học máy
- Mã độc nén

1.3.3. Phạm vi nghiên cứu

Đánh giá hiệu năng của mô hình học máy trong mã độc nén, phân tích những tính năng quan trọng trong việc nhận diện mã độc bằng XAI, tạo ra các mẫu đối kháng bằng cách sử dụng mạng đối nghịch tạo sinh (GAN) để đánh giá lại kết quả của mô hình.

1.3.4. Cấu trúc của đồ án

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

- Chương 1: Giới thiệu tổng quan về đề tài và những nghiên cứu liên quan.
- Chương 2: Trình bày cơ sở lý thuyết.

- Chương 3: Trình bày mô hình nhận diện mã độc nén.
- Chương 4: Trình bày thực nghiệm và đánh giá.
- Chương 5: Kết luận và hướng phát triển của đề tài.

CHƯƠNG 2. CƠ SỞ LÝ THUYẾT

Chương này trình bày cơ sở lý thuyết của nghiên cứu: Bao gồm tệp thực thi, mô hình phân tích mã độc, Explainable AI và mẫu đối kháng.

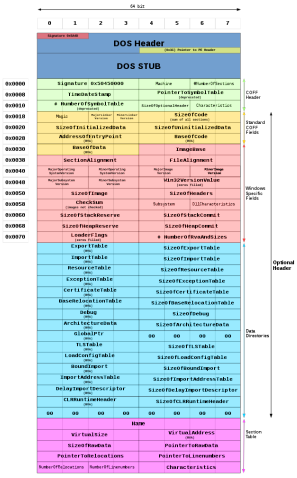
2.1. Tệp thực thi PE

2.1.1. Cấu trúc tệp thực thi

Trong nghiên cứu này chúng tôi chỉ sử dụng những tệp thực thi có định dạng PE.

Định dạng tệp PE, là một loại tệp thực thi tiêu chuẩn trên hệ hiệu hành Windows, tên đầy đủ là Portable Executable (PE), định dạng tệp tin PE là định dạng của tệp thực thi executables(.exe) và dynamic link library(.dll), được sử dụng cho cả 2 phiên bản hệ điều hành 32 bit và 64 bit. Định dạng này là một cấu trúc đóng gói dữ liệu cho trình tải hệ điều hành Windows(Windows OS loader) để quản lý và thực thi những mã lệnh. Định dạng tập tin PE có các vai trò cơ bản sau:

Cấu trúc chung của định dạng tệp tin PE sẽ có 2 phần là header và section, phần header dùng để lưu các giá trị định dạng và offset của các section trong phần section.



Hình 2.1: *Cấu trúc định dạng tệp tin PE*

2.1.1.1. Trường DOS Header

Trường này được thực thi bất cứ khi nào tập tin chạy trong môi trường MS-DOS, mục tiêu của nó là để thông báo tập tin này không thể hoạt động trong môi trường trên. Chi tiết hơn, trường tiêu đề DOS chiếm 64 bytes đầu tiên trong tập tin chứa 2 giá trị quan trọng là e_magic (0x5a4b) và e_lfanew là một DWORD (4bytes), giá trị nằm trong 4 bytes này chứa khoảng cách (offset) đến Trường tiêu đề tập tin PE. 64 bytes đầu tiên của tệp thực thi PE được chiếm bởi DOS header, điều này giúp cho tệp tin có thể thực thi được trong môi trường MS-DOS. Trường này có 2 giá trị quan trọng là e_magic là chữ ký của tệp thực thi PE và và e_lfanew là một DWORD chứa offset của PE Header so với vị trí đầu tệp.

2.1.1.2. Trường DOS STUB

Trường này là một chương trình DOS EXE dùng để thông báo lỗi: “This is program cannot be run in DOS mode” khi chương trình không tương thích.

2.1.1.3. Trường PE Header

Trường PE header bao gồm các thông tin cần thiết để tải chương trình lên bộ nhớ. Cấu trúc này gồm 3 phần được định nghĩa trong windows.inc:

- **Signature:** là 1 DWORD bắt đầu của PE Header chứa chữ ký : 50h, 45h, 00h, 00h.
- **IMAGE_FILE_HEADER:** Trường này bao gồm 20 bytes tiếp theo của PE Header, phần này chứa thông tin về sơ đồ bố trí vật lý và các đặc tính của tệp tin.
- **IMAGE_OPTIONAL_HEADER:** Gồm 224 bytes tiếp theo sau FILE_HEADER Chứa thông tin về sơ đồ logic trong tệp tin PE. Trường này chứa một số thông tin quan trọng giành cho việc chỉnh sửa tệp tin
  - **Magic (2 bytes)**: Xác định là tệp tin 32 bit (0B 01) hay 64 bit (0B 20)
  - **AddressOfEntryPoint (4bytes):** Chứa địa chỉ ảo tương đối (RVA) của câu lệnh đầu tiên sẽ được thực thi khi chương trình PE loader sẵn sàng để chạy tệp thực thi PE (.text). Nếu muốn chương trình bắt đầu từ một địa chỉ khác (để thực thi câu lệnh với mục đích khác) thì cần thay đổi địa chỉ này về địa chỉ tương đối của câu lệnh muốn thực thi.
  - **ImageBase:** địa chỉ nạp được ưu tiên cho tệp tin PE.
  - **Section Alignment:** Phần liên kết của các Section trong bộ nhớ.
  - **File Alignment:** Phần liên kết của các Section trong tệp tin. Tương tự như SectionAlignment nhưng áp dụng với tệp tin.
  - **SizeOfImage:** Toàn bộ kích thước của Pe image trong bộ nhớ, là tổng của tất cả các headers và sections được liên kết tới Section Alignment
  - **SizeOfHeaders:** Kích thước của tất cả các headers + section table = kích thước tệp tin trừ đi tổng kích thước của các section trong tệp tin.

– **Data Directory:** là một mảng gồm 16 phần tử, trong đó mỗi phần liên quan đến một cấu trúc dữ liệu quan trọng trong tệp tin PE.

2.1.1.4. Phân đoạn bảng (Section Table)

Section Table là thành phần kế tiếp theo sau PE Header, Section Table bao gồm một mảng những cấu trúc IMAGE_SECTION_HEADER, mỗi phần từ trong đó chứa thông tin về một phân đoạn trong tệp tin PE. Một số thành phần quan trọng:

- **Kích thước ảo (VirtualSize):** Kích thước thật của dữ liệu sau khi được tải lên bộ nhớ.
- **Địa chỉ ảo (VirtualAddress):** Giá trị ánh xạ của các section sau khi được tải lên bộ nhớ.
- **SizeOfRawSection:** Kích thước của section data có trên ổ đĩa.
- **PointerOfRawSection:** là offset từ đầu tệp tin cho tới section data.
- **Đặc tính (Characteristic):** Chứa đặc tính của section: thực thi, dữ liệu khởi tạo ...

2.1.1.5. Phân đoạn tập tin PE (Section PE File)

Phần sections của một tệp tin PE chứa dữ liệu cần thiết để có thể thực thi được tệp tin. Mỗi phần của PE section đều có chứa tiêu đề riêng (Section Header) được lưu trữ trong Section Table. Section gồm có:

- .text: Phần .text là phân đoạn chứa mã thực thi của tệp tin PE.
- .data: Chứa dữ liệu, các biến được khởi tạo trong mã lệnh.
- .rdata: Chứa những dữ liệu chỉ đọc, các biến const.

- .idata: Chứa bảng nhập (Import Tables). Phần này được trình tải sử dụng để xác định sẽ tải những DLL nào và các chức năng được sử dụng từ mỗi DLL.
- .edata: Chứa những thư mục xuất (Export Directory). Phần này bao gồm những hàm có chức năng xuất.
- .rsrc: Chứa tài nguyên mà một chương trình sử dụng. Bao gồm hình ảnh, biểu tượng, ...

2.2. Mô hình phát hiện mã độc

Mục đích của phát hiện mã độc là đưa ra những cảnh báo sớm để có cơ chế ngăn chặn kịp thời trước khi các tập tin mã độc thực thi và gây ra các tổn hại tới hệ thống. Vì thế, vai trò của phát hiện mã độc là rất quan trọng trong việc giảm thiểu thiệt hại hoặc ngăn chặn các tập tin này kịp thời.

2.2.1. Phương pháp phân tích tĩnh

Đối với việc phân tích mã độc thường chia ra làm 2 loại là phân tích động và phân tích tĩnh, trong đề tài này chúng tôi sẽ tập trung vào đề tài phân tích tĩnh. Kỹ thuật phát hiện mã độc dựa trên phương pháp phân tích tĩnh có đặc điểm là phát hiện mã độc mà không cần phải chạy hay thực thi bất kỳ đoạn mã nào, mà thay vào đó những thuộc tính của tệp thực thi được trích xuất ra bằng một số kỹ thuật chuyên sâu chẳng hạn như trích xuất từ trường Header, trích xuất từ các đoạn mã được tải lên, trích xuất những thư viện được nhập vào (Imported Libraries), ...[5]

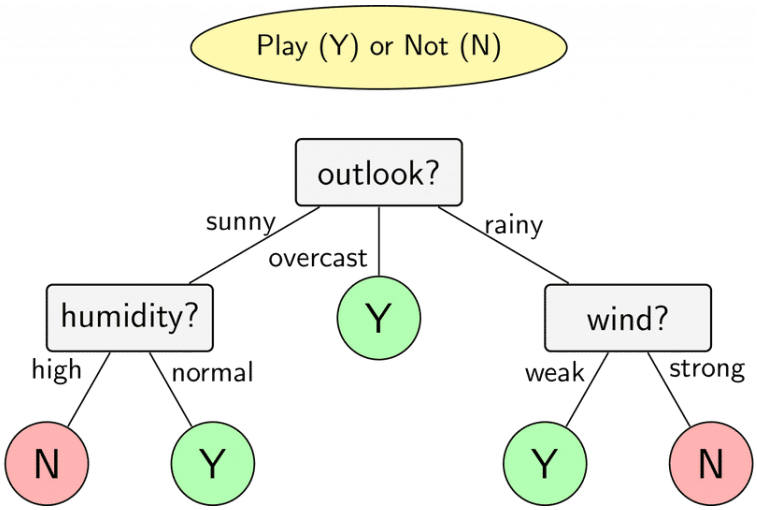
2.2.2. Mô hình phát hiện mã độc dựa trên học máy

Từ kỹ thuật phân tích tĩnh, các nhà nghiên cứu đã tạo ra được những mô hình học máy nhằm phát hiện mã độc bằng cách học theo những thuộc tính

đã được phân tích. Mô hình phát hiện mã độc dựa trên học máy cung cấp khả năng phát hiện một cách tự động dựa theo các thuật toán học máy phổ biến được ứng dụng trong việc phát hiện mã độc có thể kể tới như: Cây quyết định (Decesion Tree), Máy véc-tơ hỗ trợ (SVM), Naive Bayes, Rừng ngẫu nhiên (Random Forest), Mạng nơ-ron (Neural Networks), ... Trong phạm vi của đề tài, chúng tôi sẽ xây dựng mô hình học máy dựa theo những thuật toán Decesion Tree, Random Forest, Gradient Boosting, AdaBoost và 1 mạng nơ-ron thần kinh tích chập (Convolutional neural network).

2.2.2.1. Thuật toán cây quyết định

Decision Tree là một thuật toán supervised learning (học có giám sát) Trong lý thuyết quyết định, một cây quyết định là một đồ thị của các quyết định và các hậu quả có thể xảy ra của quyết định đó.

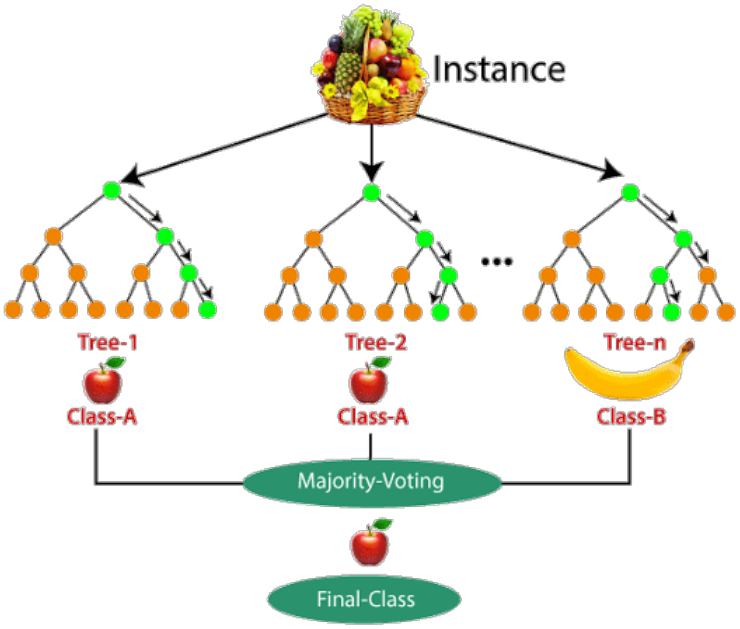


Hình 2.2: *Ví dụ về thuật toán cây quyết định*

Hình 2.22.2 là một ví dụ 1 về cây quyết định. Giả sử dựa theo thời tiết mà một gia đình có quyết định đi chơi hay không ? Dựa theo mô hình trên, nếu trời nắng, độ ẩm bình thường thì khả năng gia đình này đi chơi cao. Còn nếu trời nắng, độ ẩm cao thì khả năng sẽ không đi.

2.2.3. Thuật toán rừng ngẫu nhiên

Nếu như thuật toán cây quyết định tượng trưng cho sự quyết định bởi một cây thì Random Forest sẽ gộp nhiều cây quyết định lại với nhau, mỗi quyết định của mỗi cây sẽ được tổng hợp lại và đưa ra quyết định cuối cùng. Giả sử như



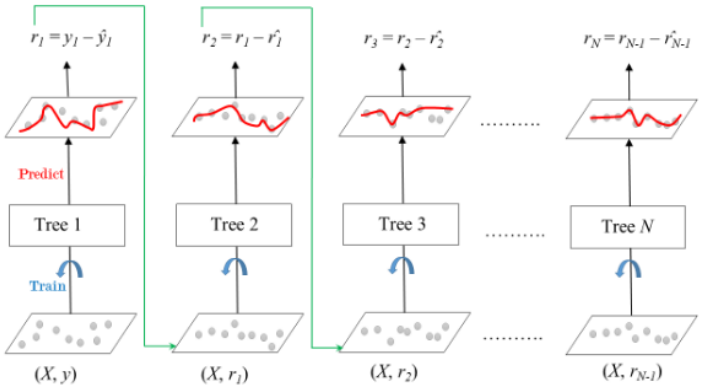
Hình 2.3: *Ví dụ về thuật toán rừng ngẫu nhiên*

có 2 cây quyết định đưa ra quyết định là táo, cây cuối cùng lại đưa ra chuối thì kết quả cuối cùng sẽ được tổng hợp lại đưa ra quyết định đó là táo.

2.2.4. Thuật toán tăng cường độ dốc (Gradient Boosting)

Gradient Boosting là một thuật toán tăng cường mạnh mẽ, kết hợp những mô hình học yếu thành mô hình học mạnh mẽ, trong đó mỗi mô hình mới được huấn luyện để giảm thiểu hàm mất mát (loss function) của mô hình trước đó bằng cách sử dụng phương pháp descent. Trong mỗi lần lặp, thuật toán tính toán độ dốc của hàm mất mát theo các dự đoán hiện tại và sau đó huấn luyện một mô hình mới để giảm thiểu độ dốc này. Các dự đoán của mô hình mới sau đó được thêm vào tập hợp và quá trình này được lặp lại cho đến khi đáp ứng

tiêu chí dừng.



Hình 2.4: *Ví dụ về thuật toán Gradient Boosting*

Thuật toán Gradient Boosting nhằm để giải quyết bài toán tối ưu sau:

$$min_{c_n, w_n} L(y, W_{n-1} + c_n w_n)$$

Trước tiên cần phải nói về công thức cập nhật tham số mô hình theo hướng giảm của đạo hàm (Gradient Descent):

$$\theta_n = \theta_{n-1} - \eta \frac{\partial}{\partial \theta} L(\theta_{n-1})$$

Nếu coi các mô hình boosting là một hàm số W thì mỗi hàm learner có thể coi là một tham số w. Đến đây, để cực tiểu hóa hàm mất mát $L(y, W)$ sẽ có công thức:

$$W_n = W_{n-1} - \eta \frac{\partial}{\partial \theta} L(W_{n-1})$$

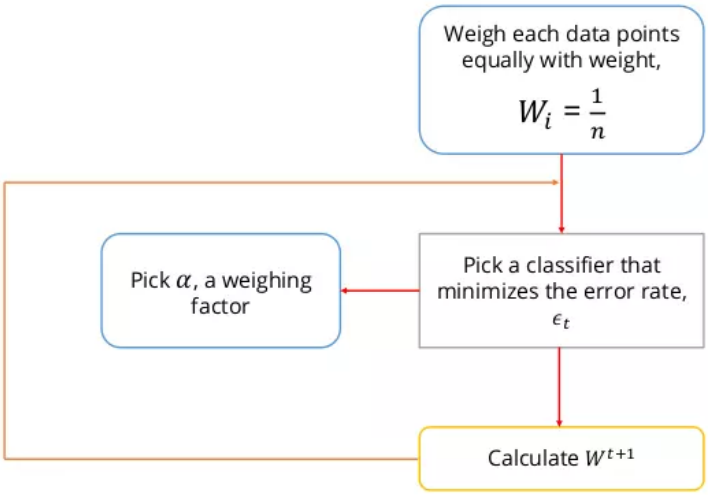
Dựa vào sự liên hệ giữa 2 công thức sẽ rút gọn được như sau:

$$c_n w_n \approx -\eta \frac{\partial}{\partial w} L(W_{n-1})$$

Với mỗi w_n là mô hình tiếp theo được thêm vào thuật toán. Mô hình sẽ cần học để phù hợp với giá trị $-\eta \frac{\partial}{\partial w} L(W_{n-1})$. Giá trị này có tên gọi là pseudo-residuals và được sử dụng làm nhãn cho mô hình tiếp theo.

2.2.5. Thuật toán AdaBoost

AdaBoost (được coi là 1 trường hợp đặc biệt của Gradient Boosting) là thuật toán với ý tưởng đơn giản là sử dụng các cây quyết định(1 gốc, 2 lá) để đánh trọng số cho các điểm dữ liệu, từ đó cải thiện hiệu suất của các mô hình yếu để tạo ra một mô hình mạnh. Các bước triển khai thuật toán AdaBoost dựa theo



Hình 2.5: *Ví dụ về thuật toán AdaBoost*

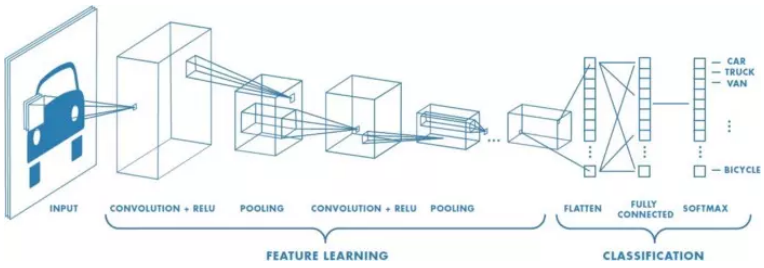
hình 2.5 được triển khai như sau:
Đối với giải thuật AdaBoost bước đầu cần khởi tạo trọng lượng cho từng input, thuật toán trên nhằm để khởi tạo trọng lượng ban đầu là bằng nhau và bằng $\frac{1}{n}$ cho mỗi điểm dữ liệu:

$$w_i^1 = \frac{1}{n}, i \in \{1, 2, ..., n\} \rightarrow \Sigma_i w_i^1 = 1$$

Với mỗi vòng lặp sẽ huấn luyện mô hình học yếu mơi w_i mới được thêm vào. Tính giá trị mất mát và từ đó tính ra được giá trị điểm tự tin c_i của mô hình vừa mới huấn luyện. Cập nhật lại mô hình chính $W = W + c_i w_i$. Bước cuối cùng sẽ đánh giá lại trọng số cho các điểm dữ liệu và tiếp tục tiến tới mô hình mới $i + 1$.

2.2.6. Mạng nơ-ron tích chập

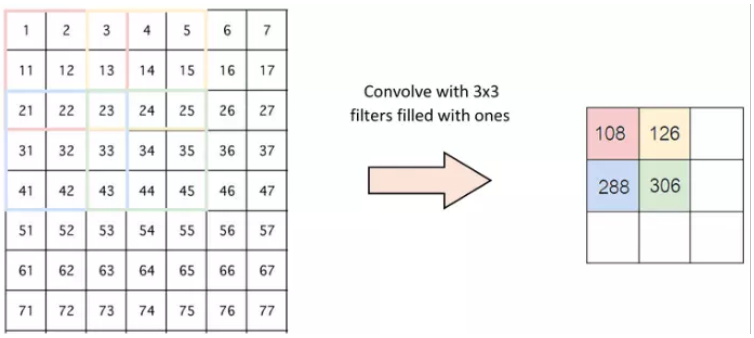
Trong những mô hình nơ-ron, mạng nơ-ron tích chập(CNN) được dùng cho việc nhận dạng và phân loại hình ảnh. Mạng nơ-ron tích chập phân loại bằng cách lấy một dữ liệu đầu vào, xử lý và phân loại nó thành 1 nhãn nhất định (ví dụ: nam, nữ,...).Sau khi mô hình được huấn luyện, mỗi dữ liệu đầu vào sẽ được chuyển qua những lớp tích chập (convolutional layer) với các bộ lọc (Kernals), tổng hợp lại các lớp được kết nối đầy đủ (Full Connected) và áp dụng hàm Softmax để phân loại đối tượng.



Hình 2.6: *Ví dụ về thuật mạng nơ-ron tích chập*

Trong mạng CNN có những thành phần chính sau đây:

- Lớp đầu vào: Là Lớp thể hiện cho những đầu vào của mạng.
- Lớp tích chập: Tích chập là lớp đầu tiên để trích xuất các tính năng từ dữ liệu đầu vào. Lớp này duy trì mối quan hệ giữa các pixel bằng cách tìm hiểu các thuộc tính của dữ liệu.
- Bước nhảy (Stride): Stride là số pixel thay đổi trên ma trận đầu vào. Khi stride là 1 thì ta di chuyển các kernel 1 pixel. Khi stride là 2 thì ta di chuyển các kernel đi 2 pixel và tiếp tục như vậy.
- Đường viền (Padding): Trong trường hợp kernal không phù hợp thì lớp padding sẽ được sử dụng để thêm các trọng số vào đường biên của dữ liệu.
- Lớp gộp (Pooling Layer): Lớp gộp thường được sử dụng để giảm bớt số lượng khi dữ liệu qúa lớn. Có 3 loại lớp gộp là Max Pooling, Average Pooling và



Hình 2.7: *Ví dụ về bước nhảy trong mạng CNN*

Sum Pooling:

- Max Pooling: Lấy phần tử lớn nhất từ ma trận đối tượng.
- Average Pooling: Lấy trung bình tổng các ma trận đối tượng.
- Sum Pooling: Tổng tất cả các phần tử trong ma trận.

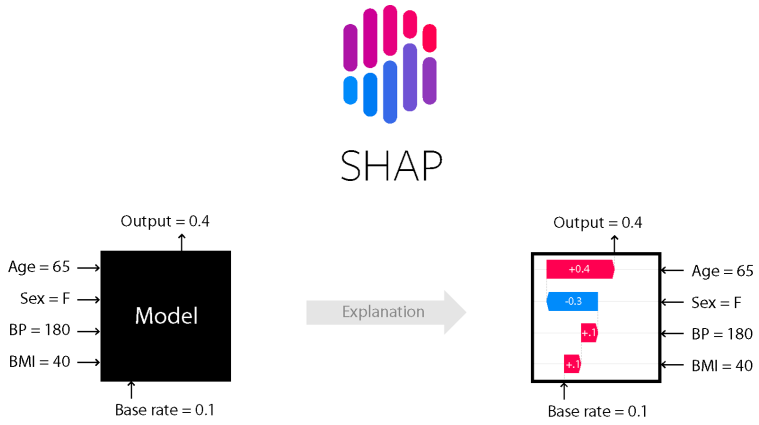
Mạng nơ-ron tích chập có những ưu điểm sau so với mạng nơ-ron truyền thống: Lượng tham số ít hơn nhiều so với mạng kết nối đầy đủ. Các lớp tích chập (CONV), các lớp gộp giúp trích xuất đặc trưng của ảnh khiến ma trận ảnh đi qua mỗi lớp nhỏ dần đi, vì thế lượng tham số cần học cũng giảm theo. Ngoài ra vì loại đi những dữ liệu thừa có ảnh hưởng đến vị trí của đối tượng nên mạng nhận diện chính xác một đối tượng nằm ở các vị trí khác nhau trong một ảnh.

2.3. Explainable AI

Explainable AI hay XAI là một lĩnh vực mới trong những nghiên cứu về học máy. Có thể thấy rằng khi một mô hình AI đưa ra quyết định, mọi người sẽ không biết rằng tại sao mô hình lại đưa ra quyết định như vậy, điều gì dẫn đến kết quả đó. Từ đó XAI được ra đời nhằm giải quyết câu hỏi này.

2.3.1. XAI với SHAP (SHapley Additive exPlanations)

SHAP là một phương pháp được sử dụng bởi những người làm việc với các mô hình học máy hoặc những nhà phát triển AI để giải thích các dự đoán của mô hình. Điểm mạnh của SHAP so với các giải pháp XAI khác đó là có thể được sử dụng để giải thích các dự đoán của bất kỳ mô hình nào. SHAP được phát triển và tổng hợp từ nhiều công cụ XAI khác như LIME, DeepLift,... Ý tưởng chính của SHAP là tính toán các giá trị SHAP (SHAP values) cho từng thuộc tính của tập dữ liệu được sử dụng để huấn luyện và kiểm tra mô hình học máy. Những thuộc tính này sẽ được diễn giải với giá trị SHAP thể hiện tác động của tính năng đó trong việc tạo ra dự đoán do mô hình đưa ra.

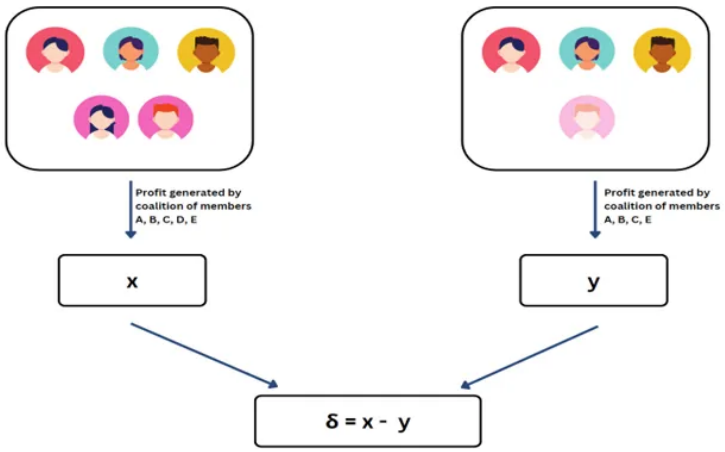


Hình 2.8: *Tổng quan về bộ công cụ SHAP*

2.3.2. Giá trị SHAP (SHAP Values)

Giá trị Shapley được lấy khái niệm từ lý thuyết của trò chơi hợp tác. Mục tiêu của giá trị Shapley là đo lường sự đóng góp của mỗi người chơi cho trò chơi. Khái niệm việc tính toán các giá trị Shapley dựa theo một trò chươi trong đó 'n' người chơi tham gia vào trò chơi với mục đích đạt được phần thưởng 'a' và phần thưởng này được phân bổ công bằng cho mỗi người chơi theo đóng góp của từng cá nhân. Hãy tưởng tượng một nhóm người (A, B, C, D, E) đang thực hiện một dự án nhằm thu được lợi nhuận (P) cho công ty. Để phân chia lợi

nhuận của công ty một cách đồng đều cho 5 người dựa trên nỗ lực của họ để đạt được lợi nhuận đó, chúng ta cần tính toán phần đóng góp của từng cá nhân để đạt được lợi nhuận (P). Đóng góp này là chính là giá trị Shapley cho mỗi người trong nhóm hoặc trong trường hợp học máy sẽ là từng thuộc tính. Để tính giá trị Shapley của một người 'A' trong nhóm, chênh lệch giữa lợi nhuận được tạo ra khi có thành viên đó và khi vắng mặt thành viên đó sẽ được tính. Sự khác

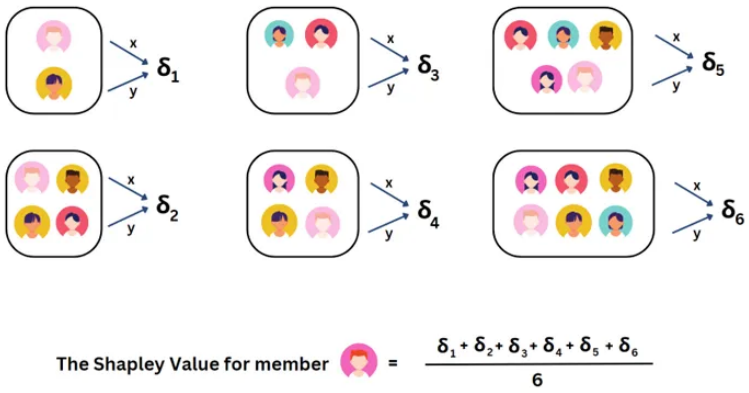


Hình 2.9: *Tính toán phân phối cận biên cho một thuộc tính*

biệt này được gọi là 'đóng góp cận biên' của thành viên 'A' cho nhóm hiện tại. Tất cả các nhóm khác nhau mà thành viên 'A' có mặt tượng trưng cho sự liên kết của dữ liệu. Do đó, để tính toán tất cả các khoản đóng góp cận biên cho thành viên này ('A'), dựa trên tất cả các liên kết có thể có của họ. Giá trị trung bình của tất cả các phân phối cận biên là 'Giá trị Shapley' cho thành viên 'A' này.

2.3.3. Ứng dụng SHAP vào phát hiện mã độc dựa theo phương pháp phân tích tĩnh

Trong ngữ cảnh phát hiện mã độc, Phương pháp XAI có khả năng cung cấp cho các nhà phân tích mã độc một số giải thích về lý do tại sao một mẫu nhất định được phân loại là độc hại hoặc lành tính. Hơn nữa đối với phương pháp phân tích tĩnh, những thuộc tính được trích xuất ra từ mẫu mã độc, qua quá



Hình 2.10: *Tính toán giá trị SHAP một thuộc tính*

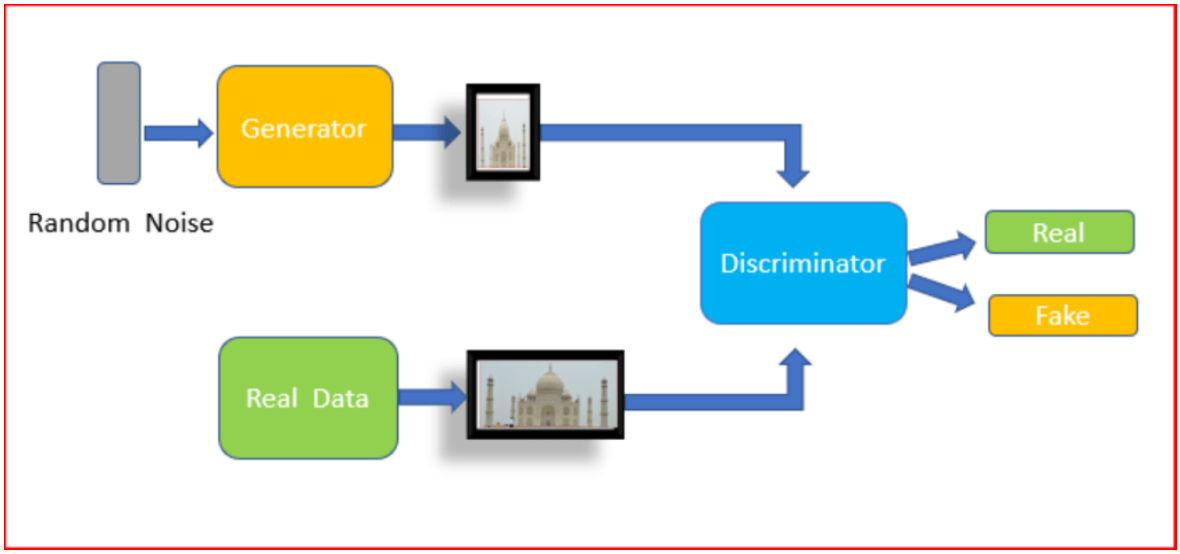
trình huấn luyện sẽ là những nhân tố quyết định tại sao mô hình đưa ra kết quả như vậy. Vì thế bộ công cụ SHAP được sử dụng tại đây để giải quyết vấn đề về những thuộc tính nào sẽ quyết định một mẫu là độc hại hay lành tính và từ đó giúp chúng ta có cái nhìn khái quát hơn về ngữ cảnh áp dụng học máy vào mã độc nén.

2.4. Mạng đối nghịch tạo sinh (GAN)

GAN thuộc nhóm generative model, có nghĩa là GAN là mô hình có khả năng sinh ra dữ liệu mới. Mô hình có tên mạng đối nghịch tạo sinh là do được cấu thành từ 2 mạng gọi là bộ sinh (Generator) và bộ phân biệt (Discriminator), luôn đối nghịch nhau trong quá trình huấn luyện mạng GAN.

2.4.1. Nguyên lý hoạt động của GAN

Nguyên lý hoạt động của GAN có thể được hiểu như một trò chơi zero_sum _game giữa hai đối thủ, thiệt hại của người này chính là lợi ích của người kia. Ở đây, bộ sinh là đối thủ đang cố gắng tạo ra dữ liệu giả giống dữ liệu thật đến mức có thể đánh lừa được bộ phân biệt và bộ phân biệt là đối thủ đang cố gắng phân biệt dữ liệu thật và dữ liệu giả một cách chính xác nhất. Khi bộ sinh tạo ra dữ liệu giả giống dữ liệu thật hơn, thì bộ phân biệt sẽ khó phân biệt hơn.



Hình 2.11: *Nguyên tắc hoạt động chính của GAN*

Điều này khiến bộ sinh phải cố gắng tạo ra dữ liệu giả giống dữ liệu thật hơn nữa. Quá trình này diễn ra lặp đi lặp lại cho đến khi bộ sinh có thể tạo ra dữ liệu giả giống dữ liệu thật đến mức bộ phân biệt không thể phân biệt được.

2.4.2. Cấu trúc của GAN

GAN cấu tạo gồm 2 mạng là Generator và Discriminator. Trong khi bộ sinh sinh ra các dữ liệu giống như thật thì bộ phân biệt cố gắng phân biệt đâu là dữ liệu được sinh ra từ Generator và đâu là dữ liệu thật.

2.4.2.1. Bộ sinh (Generator)

Bộ sinh là mô hình học cách tạo dữ liệu giả bằng cách nhận phản hồi từ bộ phân biệt. Bộ sinh học cách làm bộ phân biệt nhận dạng dữ liệu nó tạo ra là thật

2.4.2.2. Bộ phân biệt (Discriminator)

Bộ phân biệt là mô hình phân loại nhị phân làm nhiệm vụ phân loại dữ liệu thật và dữ liệu giả. Dữ liệu thật được lựa chọn từ tập dữ liệu huấn luyện và dữ

liệu giả được tạo ra từ bộ sinh. Tỷ lệ lựa chọn dữ liệu thật và dữ liệu giả để đưa vào huấn luyện Bộ phân biệt thường là 50%:50% để không bị mất cân bằng mẫu.

2.4.2.3. Hàm mất mát (Loss Function)

Hàm mất mát của GAN là một hàm kết hợp đồng thời giữa mục tiêu của bộ phân biệt và mục tiêu của bộ sinh.

$$\min_G \max_D V(D,G) = \underbrace{\mathbb{E}_{x\sim p_{\text{data}}(x)}[\log D(x)]}_{\text{log-probability that D predict x is real}} + \underbrace{\mathbb{E}_{z\sim p_z(z)}[\log(1-D(G(z)))]}_{\text{log-probability D predicts G(x) is fake}}$$

Từ hàm mất mát của GAN có thể thấy là việc huấn luyện bộ sinh và bộ phân biệt đối nghịch nhau, trong khi D cố gắng maximize loss thì G cố gắng minimize loss.

CHƯƠNG 3. THÍ NGHIỆM VÀ ĐÁNH GIÁ

Ở chương này, chúng tôi đưa ra những thí nghiệm, những đánh giá và kết quả của việc mô hình phát hiện mã độc nén, áp dụng XAI và tấn công đối kháng bằng cách sử dụng bộ sinh đối kháng (GAN).

3.1. Tập dữ liệu

Các thí nghiệm phát hiện mã độc nén mà nhóm xây dựng đều được áp dụng phương pháp phân tích tĩnh và được học trên 2 bộ dữ liệu là Wild dataset và Lab dataset được cung cấp trong công trình nghiên cứu của nhóm tác giả Aghakhani[1].

3.1.1. Wild Dataset

Tập dữ liệu được lấy từ các mẫu tệp thực thi PE (Windows x86) tồn tại trong thực tế. Cụ thể, những mẫu được lấy từ: (i) các phần mềm anti-malware thương mại, (ii) EMBER, một tập dữ liệu chuẩn được dán nhãn, được dùng để đào tạo các mô hình học máy nhằm phát hiện phần mềm độc hại trên Windows. Sau đó, gán nhãn malware/benign bằng cách kiểm tra lại các mẫu bằng những công cụ có độ tin cậy cao trên Virus Total, sau đó chọn lọc những mẫu có sự thống nhất về nhãn giữa kết quả từ Virus Total và nguồn của mẫu. Tiếp theo, gán nhãn packed/unpacked bằng các phương pháp phân tích, nhờ vào dấu hiệu giải nén khi thực thi tệp, điều đó tăng độ chính xác cho nhãn. Trong đó bao gồm: (i) sandbox từ các anti-malware thương mại, (ii) Deep Packer Inspector, (iii) các công cụ khác (Manalyze, Exeinfo PE, yara rules, PEiD, F-Prot). Cuối cùng, cho ra 50,724 mẫu, bao gồm 4,396 unpacked benign, 12,647 packed benign và 33,681 packed malicious.

3.1.2. Lab Dataset

Tập dữ liệu được tạo ra bằng cách thực hiện nén 50,724 tệp thực thi trong Wild Dataset bằng 10 bộ nén khác nhau (bao gồm Dolphin dropper 3, Obsidium, PELock, Themida, PECompact, Petite, UPX, kkrunchy, MPRESS, và tElock) thu được 341,444 tệp thực thi đã được nén. Sau đó, thực hiện xác minh các tệp đã được nén có thể hiện hành vi ban đầu của chúng trong quá trình thực thi hay không bằng cách phân tích trong Cuckoo Sandbox và so sánh với hành vi của tệp đó trước khi nén. Kết quả cho rằng 94,56% các tệp thực thi đã nén thể hiện hành vi giống như các tệp gốc.

3.1.3. Thuộc tính

Bộ dữ liệu được chia thành nhóm 9 thuộc tính gồm:

- PE headers: Có 28 thuộc tính thuộc nhóm này, trong đó 12 thuộc tính được

trích xuất từ Optional Header và COFF Header, 16 thuộc tính còn lại ở dưới dạng binary và được trích xuất trong trường đặc trưng của COFF Header, mỗi tính năng sẽ biểu diễn cờ tương ứng có được đặt cho tệp thực thi hay không.

- PE sections: Có tổng cộng 570 thuộc tính thuộc nhóm này, với mỗi trường trong PE Sections, dữ liệu được trích xuất 8 thuộc tính từ mỗi trường và với mỗi trường đặc trưng trong section header, nhóm tác giả đã tạo ra thêm 32 thuộc tính cho mỗi bit flag

- DLL imports: Có 4305 thuộc tính thuộc nhóm này. Với những tệp thực thi hầu như đều được liên kết với thư việc liên kết động DLL. Với mỗi thư viện mà được tệp thực thi sử dụng sẽ được trích xuất ra dưới dạng nhị phân.

- API imports: Tương tự như thuộc tính DLL imports, với mỗi API mà tệp thực thi sử dụng sẽ được trích xuất dưới dạng binary, tổng cộng có 19168 thuộc tính, nhóm này có 19168 thuộc tính.

- Rich Header: Những thuộc tính này được trích xuất từ trường Rich Header, nhóm này gồm có 66 thuộc tính.

- Byte n-grams: Byte n-ngrams là nhóm có 13000 thuộc tính. Thuộc tính này được trích xuất ra bằng cách chọn 6 gram xuất hiện ở hơn 1% số mẫu trong tập hợp, dẫn đến 204.502 tính năng. Sau đó, nhóm tác giả đã chọn 13.000 tính năng n-gram hàng đầu dựa trên thước đo Tăng cường Thông tin[1]

- Opcode n-grams: Với thuộc tính này, nhóm tác giả sự dụng trình biên dịch ngược (disassembler) Capstone để chuyển các tệp thực thi thành những chuỗi opcode, tổng cộng có 2500 thuộc tính

- Strings: Thuộc tính này là nhóm những chuỗi có thể in được trong tệp thực thi, được tạo ra bằng cách sử dụng phần mềm GNU strings để trích xuất những chuỗi in được dài nhất 4 ký tự, tổng cộng có 16900 thuộc tính.

- File generic: Thuộc tính này được tạo ra bằng cách tính kích thước và entropy của từng tệp, vì vậy nhóm này chỉ có 2 thuộc tính.

3.2. Kịch bản triển khai

Như đã trình bày trong Chương 2, nhóm chúng tôi sẽ sử dụng những thuật toán như Decesion Tree, Random Forest, Gradient Boosting, AdaBoost và 1 mạng nơ-ron tích chập để triển khai 6 kịch bản với mục đích xây dựng mô hình phát hiện mã độc nén và đánh giá mô hình.

Những câu hỏi được đặt ra trong quá trình nghiên cứu là

- **C**âu hỏi 1: Liệu mô hình có nhẫm lẫn giữa bộ nén và mã độc không sẽ được thực nghiệm trong thí nghiệm 1 và 2.
- **C**âu hỏi 2: Tại sao mô hình phân loại học máy lại đưa ra những nhận định như vậy? Những thuộc tính quan trọng góp phần trong những nhận định đó? Được trả lời tại thí nghiệm 3.
- **C**âu hỏi 3: Với mô hình phân loại học máy khác nhau thì SHAP có đưa ra những nhận định như giống nhau không? Được trả lời thông qua thí nghiệm 4.
- **C**âu hỏi 4: Liệu các bộ nén khác nhau có gây cản trở trong việc sử dụng mô hình phân loại học máy không? Được trả lời tại thì nghiệm 5, 6, 7.
- **C**âu hỏi 5: Liệu các bộ nén có chặn được mô hình phân loại học máy chỉ sử dụng phân tích tĩnh hay không? Được trả lời thông qua thí nghiệm 2, 6, 8.
- **C**âu hỏi 6: Khi bị tấn công đối kháng, mô hình có còn nhận diện được không? Được trả lời thông qua thí nghiệm 9.

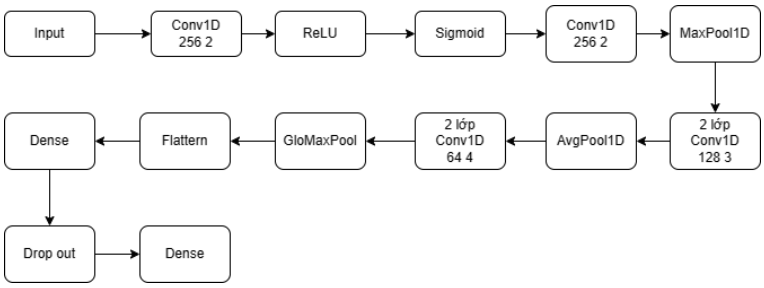
3.2.1. Trình phát hiện mã độc nén

Để tra lời câu hỏi đầu tiên, đầu tiên chúng tôi xây dựng thí nghiệm 1, là một mô hình dựa theo bộ dữ liệu Wild với chỉ 2 nhãn là (Packed Malware và Unpacked Benign) mục đích của việc này là kiểm tra xem nếu không được học về Packed Benign (Mẫu lành tính đã bị nén) thì mô hình có bị nhầm lẫn những tệp bị nén độc thành mã độc hay không.

Trước khi trình bày về kết quả, chúng tôi muốn nói rõ hơn về cấu trúc mô hình mạng nơ-ron của nhóm đề xuất như sau:

- **I**nput layer nhận đầu vào có kích thước (timesteps, features), trong đó imesteps là số mẫu được đưa vào, features là số lượng đặc trưng được lọc ra.
- **L**ớp tích chập đầu tiên sử dụng 256 bộ lọc (filters) với kích thước kernel là 2 và hàm kích hoạt 'relu'. Sau đó, một hàm kích hoạt 'sigmoid' được áp dụng sau layer này.
- **L**ớp tích chập thứ hai cũng có 256 filters và kernel size là 2.
- **M**ột lớp Max Pooling 1D được đặt sau đó với pool size là 2 để giảm kích thước của đầu ra.
- **T**iếp theo là hai lớp tích chập thứ ba và tư có 128 filters và kernel size là 3.
- **T**iếp theo áp dụng một lớp Average Pooling 1D sau lớp tích chập thứ 4 với pool size là 2 để giảm kích thước của đầu ra.
- **L**ớp tích chập thứ 5 và 6 có 64 filters và kernel size là 4.
- **T**iếp theo sẽ sử dụng 1 lớp Global MaxPooling 1D để chọn ra giá trị lớn nhất từ toàn bộ đầu ra của lớp tích chập cuối cùng.
- **N**hững lớp trên sẽ được qua một lớp Flatten để làm phẳng dữ liệu thành một chiều trước khi đưa vào các lớp Fully Connected (Dense). Lớp Fully

Connected sẽ gồm 1 lớp Dense với đầu vào là 64 nơ-ron với hàm kích hoạt 'relu', một lớp dropout với tỷ lệ 0.35 để tránh overfit và 1 lớp Dense cuối cùng sử dụng hàm kích hoạt sigmoid để sử dụng cho bài toán phân loại hai nhãn.



Hình 3.1: *Mô hình mạng CNN được áp dụng*

Tiếp theo chúng tôi sẽ trình bày kết quả của thí nghiệm trên, sau khi hoàn tất huấn luyện mô hình dựa theo những thuật toán trên, mô hình có độ chính xác rơi vào khoảng từ 81% đến 86%.

Experiment 1	**Type of dataset**	**Accuracy**	**Precision**	**Recall**	**F1-score**
Random Forest	Packed MW và Unpacked Benign (Wild)	0,85	0,81	0,92	0,86
Decision Tree	Packed MW và Unpacked Benign (Wild)	0,81	0,76	0,91	0,83
Gradient Boosting	Packed MW và Unpacked Benign (Wild)	0,85	0,78	0,96	0,86
Ada Boost	Packed MW và Unpacked Benign (Wild)	0,86	0,81	0,96	0,88
CNN	Packed MW và Unpacked Benign (Wild)	0,86	0,82	0,91	0,87

Bảng 3.1: *Kết quả thí nghiệm 1*

Tiếp theo, chúng tôi triển khai thí nghiệm 2 mô hình được học đầy đủ nhãn hơn với 4 nhãn (Packed Malware, Unpacked Benign, Packed Benign, Unpack Malware) để so sánh. Kết quả lúc này đã tốt hơn, rơi vào khoảng 92% đến 95%.

Experiment 2	**Type of dataset**	**Accuracy**	**Precision**	**Recall**	**F1-score**
Random Forest	Packed MW và Unpacked Benign (Wild)	0,93	0,93	0,9	0,91
Decision Tree	Packed MW và Unpacked Benign (Wild)	0,92	0,94	0,93	0,93
Gradient Boosting	Packed MW và Unpacked Benign (Wild)	0,95	0,96	0,93	0,95
Ada Boost	Packed MW và Unpacked Benign (Wild)	0,94	0,96	0,93	0,94
CNN	Packed MW và Unpacked Benign (Wild)	0,93	0,89	0,91	0,9

Bảng 3.2: *Kết quả thí nghiệm 2*

Dựa vào kết quả của bảng 3.1 và 3.2, kết quả chỉ ra rằng với mô hình học thiếu nhãn Packed Benign, tỷ lệ phát hiện của mô hình sẽ thấp hơn so với việc được học đầy đủ 4 nhãn và những trọng số như dương tính giả của thí nghiệm 1 có kết quả thấp hơn thí nghiệm 2

Từ đó có thể kết luận rằng mô hình học thiếu nhãn sẽ có khả năng bị nhầm những mẫu nén thành mẫu mã độc và với những mô hình không được học về dữ liệu mã độc nén, tỷ lệ này nhầm lẫn này cũng sẽ xảy ra.

3.2.2. Những thuộc tính quan trọng của mô hình nhận diện mã độc

Để tìm ra được những thuộc tính mà mô hình học máy xem là quan trọng trong việc nhận diện mã độc, chúng tôi áp dụng công cụ SHAP đã được trình bày ở chương 2 để triển khai.

Để trả lời câu hỏi này chúng tôi thực hiện thí nghiệm 3 với mô hình gốc là từ thí nghiệm 2 đã được nêu ở trên (huấn luyện mô hình với 4 nhãn Packed Malware, Unpacked Benign, Packed Benign, Unpack Malware) và sử dụng thuật toán rừng ngẫu nhiên để phân tích.

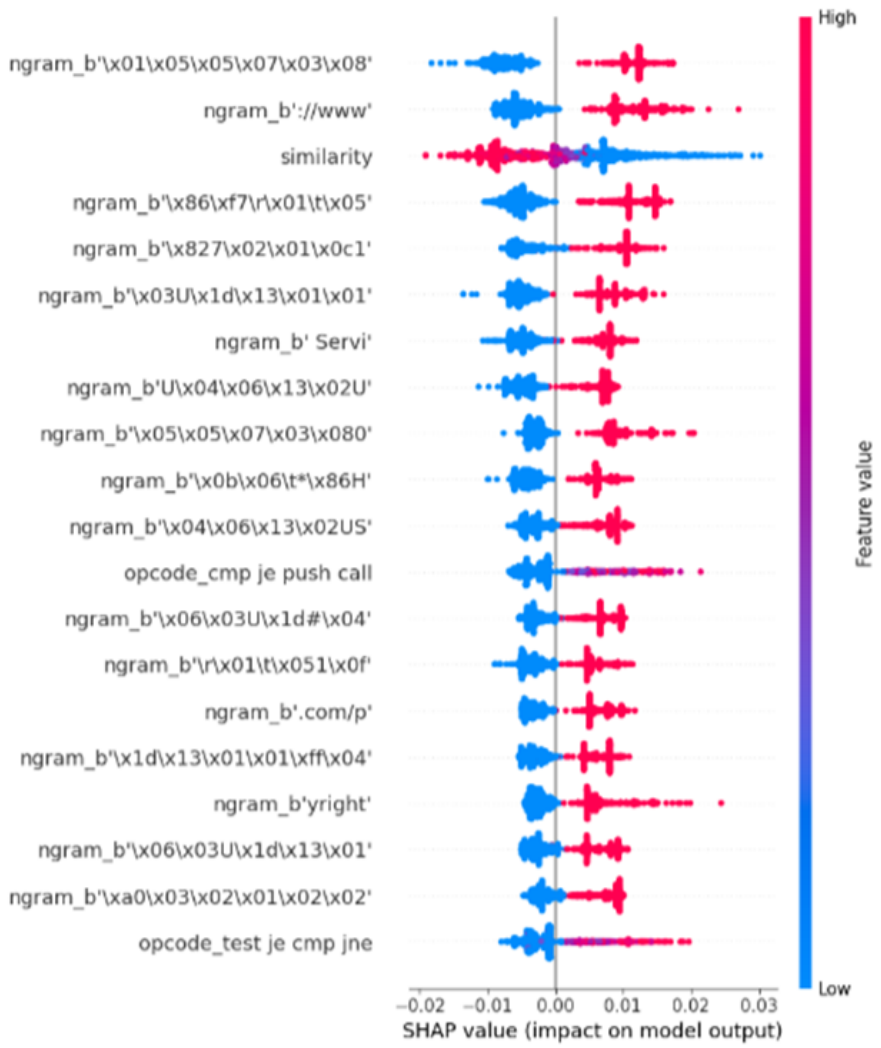
Kết quả chi tiết của thí nghiệm như sau:

Thí nghiệm 3 chỉ ra rằng, những thuộc tính thuộc nhóm bytes-ngram đóng vai trò quan trọng trong việc nhận diện một mẫu là mã độc đối với thuật toán rừng ngẫu nhiên. Vậy nếu chúng ta áp dụng những thuật toán học máy khác thì kết quả có giống vậy không. Để trả lời câu hỏi chúng tôi thực nghiệm thí nghiệm thứ 4 giống với thí nghiệm 3 nhưng thay vì dùng thuật toán rừng ngẫu nhiên, chúng tôi chọn cây quyết định và Gradient Boosting.

Đối với kết quả của thực nghiệm, mỗi mô hình học máy đều có quyết định khác nhau trong việc nhận diện một mẫu mã độc, vì vậy khi này những thuộc tính quan trọng đối với từng thuật toán cũng khác nhau. Sau đây là danh sách 20 thuộc tính đứng đầu và quan trọng nhất trong mỗi thuật toán từ thí nghiệm 3 và 4:

- **R**andom Forest:

```
ngram_b'ation1'
ngram_b'\x06\x13\x02US1'
ngram_b'\x04\x06\x13\x02US'
ngram_b'\r\x01\t\x061\x82'
similarity
ngram_b'0\x82\x01\n\x02\x82'
```



Hình 3.2: *Bảng kết quả của thí nghiệm 3*

```
ngram_b'//www.'
opcode_test je lea
ngram_b'porati'
ngram_b'\n+\x06\x01\x04\x01'
ngram_b'\x00\x03\x82\x01\x0f\x00'
ngram_b'\r\x01\t\x051\x0f'
ngram_b'\x031\x0b\x06\t*'
ngram_b'+\x06\x01\x05\x05\x07'
ngram_b'\x1c\x06\t*\x86H'
ngram_b'0\x18\x06\t*\x86'
ngram_b'0\x1c\x06\t*\x86'
opcode_jmp mov push push
ngram_b'7\x02\x01\x0b1\x0e'
ngram_b'\x07\x010\x1c\x06\t'
```

- **D**ecesion Tree:

```
ngram_b'\x05\x05\x07\x03\x080'
opcode_test je lea
similarity
pesectionProcessed_sectionsMaxEntropy
opcode_add mov add add
ngram_b'\xe1\x00\x00\x00\x00\x00'
string_b'u=;]'
pesection_2_rawAddress(pointerToRawData)
ngram_b'\x00m\x00p\x00a'
ngram_b'ttp://'
string_b'GetNativeSystemInfo'
opcode_and xor
```

opcode_call int3 push mov

string_b'‘DATA'

pesectionProcessed_entrypointSection_characteristics_bit31

imp_process32next

pesectionProcessed_sectionsMaxSize

string_b'el32'

ngram_b'indows'

- **G**radient Boosting:

pesection_2_entropy

ngram_b'ation1'

ngram_b'\x05\x05\x07\x03\x080'

pesectionProcessed_entrypointSection_entropy

opcode_test je lea

pesectionProcessed_resources_nb

similarity

string_b'VB5!'

pesection_2_rawAddress(pointerToRawData)

shell32.dll

ngram_b'm\x00p\x00a\x00'

ngram_b'\x00n\x00y\x00N'

ngram_b'Copyri'

pesection_1_entropy

pesectionProcessed_resourcesMeanSize

opcode_test je push mov

ngram_b'osoft.'

pesection_5_virtualAddress

ngram_b' Code '

```
opcode_jmp push pop push
```

3.2.3. Thí nghiệm với những bộ nén

Để kiểm tra tính thực tiễn của mô hình phát hiện mã độc nén trong thí nghiệm 2, chúng tôi thực hiện thí nghiệm liên quan đến những bộ nén, dữ liệu được dùng bây giờ là dữ liệu Lab, là dữ liệu của bộ dữ liệu Wild được nén lại bằng 10 bộ nén khác nhau. Những thí nghiệm với bộ nén đều được dùng thuật toán rừng ngẫu nhiên.

Thí nghiệm 5 được tạo ra để huấn luyện mô hình học máy có khả năng nhận diện được 10 bộ nén đã nêu ở trên, dữ liêu huấn luyện và kiểm tra chi tiết như sau:

	Số lượng mẫu huấn luyện	**Số lượng mẫu kiểm tra**
Mỗi bộ nén lấy 10000 mẫu và chia làm 4 nhãn (Packed MW, Packed Benign, Unpacked Benign, Unpacked Malware)	7000	3000

Bảng 3.3: *Dữ liệu được sử dụng cho thí nghiệm 5*

Sau khi huấn luyện mô hình với mỗi bộ nén, tỷ lệ phát hiện mã độc trên mỗi bộ nén rơi vào khoảng từ 72% đến 94%.

Packer	**Type of dataset**	**Accuracy**	**Precision**	**Recall**	**F1-score**
Dolphin dropper 3	Dolphin dropper 3 (malware, benign)	0,72	0,73	0,71	0,72
Obsidium	Obsidium (malware, benign)	0,91	0,89	0,93	0,91
PELock	PELock (malware, benign)	0,94	0,91	0,96	0,94
PECompact	PECompact (malware, benign)	0,89	0,89	0,89	0,89
PEtite	PEtite (malware, benign)	0,94	0,92	0,96	0,94
UPX	UPX (malware, benign)	0,93	0,93	0,93	0,93
kkrunchy	kkrunchy (malware, benign)	0,9	0,88	0,92	0,9
mpress	mpress (malware, benign)	0,91	0,89	0,94	0,91
telock	telock (malware, benign)	0,85	0,85	0,85	0,85
Themida	Themida (malware, benign)	0,9	0,9	0,9	0,9

Bảng 3.4: *Kết quả của thí nghiệm 5*

Ở thí nghiệm 6, chúng tôi tìm ra những thuộc tính quan trọng đối với từng bộ nén để chỉ ra rằng những bộ nén khác nhau đều có những thuộc tính quan trọng khác nhau. Trong thí nghiệm này, chúng tôi sẽ lấy ra 50 thuộc tính quan trọng hàng đầu với từng bộ nén và thống kê chúng thuộc nhóm thuộc tính nào trong 9 nhóm.

Packer	API import	dll	rich header	pe sections	pe header	strings	byte ngram	opcode ngram	generic
Dolphin dropper 3	0	0	0	0	0	48	0	0	2
Obsidium	0	0	0	19	3	0	27	0	1
PELock	0	0	0	0	0	0	49	0	1
PECompact	0	0	4	22	4	0	17	0	2
PEtite	0	1	3	8	1	0	37	0	0
UPX	2	1	3	19	5	0	18	0	2
kkrunchy	0	0	0	23	5	0	19	1	2
mpress	0	1	0	3	0	0	45	0	1
telock	0	0	3	39	3	0	4	0	1
Themida	0	0	0	0	0	0	0	50	0

Bảng 3.5: *Kết quả của thí nghiệm 6*

Dựa vào kết quả của thí nghiệm 6 trong 3.5, mỗi mô hình phát hiện mã độc dựa theo từng bộ nén đều cho ra thông tin về những thuộc tính quan trọng khác nhau, chẳng hạn như bộ nén telock có 39 thuộc tính thuộc nhóm pe sections là quan trọng, còn bộ né pelock thì lại không có thuộc tính nào là quan trọng trong nhóm pe sections này cả.

Tiếp theo trong thí nghiệm 7, chúng tôi kiểm thử mô hình học máy từ thí nghiệm 2 xem liệu rằng mô hình có khả năng nhận diện được những mã độc nén mới mà mô hình chưa từng gặp không. Kết quả trả ra cho thấy rằng tỷ lệ phát hiện mã độc lúc này rất thấp, cao nhất chỉ có 77% và thấp nhất là 48%.

Packer	Train data	Test data	Accuracy	Precision	Recall	F1-score
Dolphin dropper 3	Packed MW, Packed Benign, Unpacked Benign (Wild dataset)	Dolphin dropper 3 (malware, benign)	0,5	0,5	1	0,67
Obsidium	Packed MW, Packed Benign, Unpacked Benign (Wild dataset)	Obsidium (malware, benign)	0,56	0,53	1	0,69
PELock	Packed MW, Packed Benign, Unpacked Benign (Wild dataset)	PELock (malware, benign)	0,6	0,56	1	0,71
PECompact	Packed MW, Packed Benign, Unpacked Benign (Wild dataset)	PECompact (malware, benign)	0,71	0,63	0,98	0,77
PEtite	Packed MW, Packed Benign, Unpacked Benign (Wild dataset)	PEtite (malware, benign)	0,59	0,55	1	0,71
UPX	Packed MW, Packed Benign, Unpacked Benign (Wild dataset)	UPX (malware, benign)	0,77	0,7	0,94	0,8
kkrunchy	Packed MW, Packed Benign, Unpacked Benign (Wild dataset)	kkrunchy (malware, benign)	0,5	0,5	1	0,67
mpress	Packed MW, Packed Benign, Unpacked Benign (Wild dataset)	mpress (malware, benign)	0,48	0,48	0,74	0,58
telock	Packed MW, Packed Benign, Unpacked Benign (Wild dataset)	telock (malware, benign)	0,52	0,51	1	0,68
Themida	Packed MW, Packed Benign, Unpacked Benign (Wild dataset)	Themida (malware, benign)	0,8	0,72	0,99	0,84

Bảng 3.6: *Kết quả của thí nghiệm 7*

Vậy có thể kết luận rằng nếu không được học những bộ nén mới thì mô hình sẽ mất đi khả năng nhận diện mã độc và đây cũng là một thách thức trong việc nghiên cứu những mô hình phát hiện mã độc nén tương tự.

Về thí nghiệm 8, chúng tôi sẽ kiểm thử mô hình ở thí nghiệm 5 với dữ liệu từ thí nghiệm 2 để xem mô hình được học từ những bộ nén có khả năng nhận diện những mã độc tương tự hay không, kết quả lúc này cũng khá khả quan, tỷ lệ chính xác trong khoảng 63% đến 89%.

Packer	Test data	Train data	Accuracy	Precision	Recall	F1-score
Dolphin dropper 3	Dolphin dropper 3 (malware, benign)	Packed MW, Packed Benign, Unpacked Benign (Wild dataset)	0,63	0,75	0,37	0,5
Obsidium	Obsidium (malware, benign)	Packed MW, Packed Benign, Unpacked Benign (Wild dataset)	0,84	0,92	0,74	0,82
PELock	PELock (malware, benign)	Packed MW, Packed Benign, Unpacked Benign (Wild dataset)	0,86	0,86	0,86	0,86
PECompact	PECompact (malware, benign)	Packed MW, Packed Benign, Unpacked Benign (Wild dataset)	0,82	0,8	0,86	0,83
PEtite	PEtite (malware, benign)	Packed MW, Packed Benign, Unpacked Benign (Wild dataset)	0,85	0,85	0,86	0,85
UPX	UPX (malware, benign)	Packed MW, Packed Benign, Unpacked Benign (Wild dataset)	0,89	0,87	0,91	0,89
kkrunchy	kkrunchy (malware, benign)	Packed MW, Packed Benign, Unpacked Benign (Wild dataset)	0,75	0,87	0,59	0,7
mpress	mpress (malware, benign)	Packed MW, Packed Benign, Unpacked Benign (Wild dataset)	0,82	0,86	0,77	0,81
telock	telock (malware, benign)	Packed MW, Packed Benign, Unpacked Benign (Wild dataset)	0,72	0,7	0,76	0,73
Themida	Themida (malware, benign)	Packed MW, Packed Benign, Unpacked Benign (Wild dataset)	0,84	0,8	0,9	0,85

Bảng 3.7: *Kết quả của thí nghiệm 8*

3.2.4. Mẫu đối kháng

Để kiểm tra xem ảnh hưởng của mẫu đối kháng lên mô hình học máy/học sâu phân loại mã độc, chúng tôi thực hiện thí nghiệm xây dụng mô hình GAN để sinh ra mẫu đối kháng. Sau đó thực hiện kiểm tra bằng mô hình phân loại CNN đã được đề cập ở các thí nghiệm 2.

GAN gồm 2 mô hình chính là bộ sinh và bộ phân biệt:

- Bộ sinh là một mạng lưới nơ-ron có đầu vào là noise véc-tơ ngẫu nhiên có

kích thước bằng dữ liệu huấn luyện, dữ liệu này được chọn ngẫu nhiên từ tập dữ liệu Wild được gán nhãn độc hại. Đầu ra của bộ sinh là dữ liệu giả được tạo ra.

- Bộ phân biệt là một mạng lưới nơ-ron khác nhận đầu vào là dữ liệu được sinh ra từ bộ sinh và dữ liệu thật. Đầu ra của bộ phân biệt là dự đoán mẫu nào là thật,

Quá trình huấn luyện mô hình sẽ được thực hiện xen kẽ:

- Bước 1: Huấn luyện bộ phân biệt. Đầu tiên tạo ra một dữ liệu sao cho một nửa là thật nhãn 1 và một nửa là giả nhãn 0. Sau đó đưa vào bộ phân biệt để huấn luyện.

- Bước 2: Huấn luyện bộ sinh. Đầu tiên khởi tạo dữ liệu đầu vào là véc-tơ noise. Gán nhãn 1 cho bộ sinh và huấn luyện để bộ sinh cố gắng tạo ra dữ liệu giống thật.

Sau khi hoàn thành huấn luyện GAN, kiểm tra mô hình phân loại CNN đã được đề cập ở thí nghiệm 2. Dùng dữ liệu được tạo ra, 46 mẫu gán nhãn là độc hại cùng với mẫu lành tính được lấy từ tập dữ liệu Wild, bao gồm 46 mẫu lành tính cả nén và không nén. Dữ liệu được dùng để huấn luyện mô hình phân loại CNN bao gồm 274 mẫu, trong đó có 137 mẫu độc hại được nén và 137 mẫu lành tính cả nén và không nén.

	Accuracy	**Precision**	**Recall**	**F1-score**
Wild dataset	0.87	0.86	0.84	0.83
GAN dataset	0.5	0.25	0.5	0.33

Bảng 3.8: *Kết quả của thí nghiệm 9*

Kết quả mô hình nhận dạng toàn bộ mẫu kiểm tra đều là lành tính. Kết quả này cho thấy mô hình phân loại đã đưa ra nhận định sai lầm cho những mẫu giả được tạo ra.

3.3. Kết quả thí nghiệm

Ở mục này, nhóm chúng tôi sẽ trả lời 6 câu hỏi được đặt ra ban đầu và cuối cùng, sẽ đưa ra một số kết luận.

Câu hỏi 1: Liệu mô hình có nhẫm lẫn giữa bộ nén và mã độc không?

⇒ Đối với một mô hình truyền thống, không được học về những bộ nén thì khả năng mô hình bị nhầm lẫn giữa những bộ nén và mã độc là rất cao.

Câu hỏi 2: Tại sao mô hình phân loại học máy lại đưa ra những nhận định như vậy? Những thuộc tính quan trọng góp phần trong những nhận định đó?

⇒ Những mô hình học máy đưa ra nhận định dựa theo những thuộc tính mà mô hình học được và dựa theo độ ưu tiên của từng thuộc tính có đóng góp cao trong việc đưa ra kết quả cuối cùng hay không.

Câu hỏi 3: Với mô hình phân loại học máy khác nhau thì SHAP có đưa ra những nhận định như giống nhau không?

⇒ Với từng loại thuật toán khác nhau, việc đưa ra quyết định và chọn lọc tính năng cũng khác nhau, từ đó dẫn đến việc các giá trị SHAP của từng loại thuật toán cũng khác nhau.

Câu hỏi 4: Liệu các bộ nén khác nhau có gây cản trở trong việc sử dụng mô hình phân loại học máy không?

⇒ Các bộ nén chắc chắn sẽ gây ra cản trở cho mô hình vì những bộ nén có giữ lại một số thuộc tính quan trọng trong việc nhận diện mã độc, tuy nhiên khi sử dụng mô hình ban đầu để nhận diện những mẫu đã được nén thì không còn chính xác như ban đầu.

Câu hỏi 5: Liệu các bộ nén có chặn được mô hình phân loại học máy chỉ sử dụng phân tích tĩnh hay không?

⇒ Hoàn toàn có thể vì đối với những mô hình học máy phát hiện mã độc dựa theo phân tích tĩnh, kể cả những mô hình đó đã được học dữ liệu về những bộ nén thì những mô hình này cũng chỉ có thể nhận diện được bộ nén đó và các bộ nén gần giống còn những mẫu mới chưa từng gặp thì tỷ lệ nhận diện đúng

sẽ không cao. Câu hỏi 6: Khi bị tấn công đối kháng, mô hình có còn nhận diện được không?

⇒ Mô hình đưa ra dự đoán sai lầm với những mẫu đối kháng được sinh ra từ GAN. Đây là vấn đề quan trọng, cần được giải quyết đối với mô hình dùng để phát hiện mã độc.

Qua những thực nghiệm và những câu hỏi được đặt ra, chúng tôi thấy rằng việc áp dụng học máy cho việc nhận diện mã độc nén đang là một vấn đề nan giải và cần được nghiên cứu kỹ càng hơn để có thể đưa ra thêm nhiều biện pháp phòng thủ.

CHƯƠNG 4. KẾT LUẬN

Ở chương này, chúng tôi đưa ra những kết luận về nghiên cứu, những hạn chế, và đồng thời đưa ra hướng cải thiện và phát triển.

4.1. Kết luận

Đề tài nghiên cứu về mã độc nén là một đề tài hay tuy nhiên lại đầy thách thức trong thực tế. Việc áp dụng học máy để nhận diện mã độc nén theo phương pháp phân tích tĩnh là không đủ để có thể nhận diện những mấu mã độc nén mới.

Qua việc xây dựng hệ thống tạo mẫu đột biến này, nhóm chúng tôi đã hiểu sâu hơn về các hướng nghiên cứu liên quan, hiểu được các hạn chế để góp phần cải thiện dần. Khóa luận này đã đạt được một số kết quả như sau:

- Tìm hiểu về cơ chế phát hiện mã độc của các trình phát hiện theo phương pháp phân tích tĩnh.
- Tìm hiểu về các đặc tính của tệp thực thi (PE) và phương pháp tạo mã độc nén.

- Tìm hiểu và xây dựng những mô hình phát hiện mã độc cũng như mã độc nén.
- Nghiên cứu về Explainable AI để giải thích mô hình học máy
- Thực hiện tạo mẫu đối kháng bằng bộ sinh đối kháng và tấn công mô hình học máy.

Kết quả mà nhóm thu được qua thực nghiệm cho thấy việc áp dụng học máy cho việc phát hiện mã độc nén dựa theo phương pháp phân tích tĩnh là khả thi, tuy nhiên vẫn chưa đủ mạnh. Đồng thời, nhóm cũng chỉ ra được những hạn chế của học máy trong việc phát hiện mã độc nén và những mẫu đối kháng.

4.2. Hướng phát triển

Ứng dụng XAI kết hợp với mẫu đối kháng để tạo ra những mấu đối kháng mã độc nén mạnh mẽ hơn. Ngoài ra còn có thể áp dụng việc kết hợp XAI và mẫu đối kháng để tạo ra những tệp thực thi trong thực tế nhằm áp dụng vào thực tiễn.

Ngoài ra phương pháp phòng thủ chống lại mã độc nén và tấn công đối kháng cũng rất quan trọng nên cần phải được nghiên cứu sâu hơn.

TÀI LIỆU THAM KHẢO

Tiếng Anh:

[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 2020*, 2020.

[2] Ebtihaj Alshahrani et al. (2022), "Adversarial attacks against supervised machine learning based network intrusion detection systems", *Plos one*, 17 (10), e0275971.

[3] Hyrum S Anderson and Phil Roth (2018), "Ember: an open dataset for training static pe malware machine learning models", *arXiv preprint arXiv:1804.04637*.

[4] Daniel Gibert, Carles Mateu, and Jordi Planes (2020), "The rise of machine learning for detection and classification of malware: Research developments, trends and challenges", *Journal of Network and Computer Applications*, 153, p. 102526.

[5] Muhammad Ijaz, Muhammad Hanif Durad, and Maliha Ismail, "Static and dynamic malware analysis using machine learning", in: *2019 16th International bhurban conference on applied sciences and technology (IBCAST)*, IEEE, 2019, pp. 687–691.

[6] Aditya Kuppa and Nhien-An Le-Khac (2021), "Adversarial xai methods in cybersecurity", *IEEE transactions on information forensics and security*, 16, pp. 4924–4938.

[7] R Vinayakumar and KP Soman (2018), "DeepMalNet: evaluating shallow and deep networks for static PE malware detection", *ICT express*, 4 (4), pp. 255–258.