**HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG**

---

**ĐẬU ĐỨC SIÊU**

# PHÁT HIỆN MÃ ĐỘC DỰA TRÊN PHÂN TÍCH MẪU

**LUẬN VĂN THẠC SĨ KỸ THUẬT**

**(Theo định hướng ứng dụng)**

**HÀ NỘI – 2021**

**HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG**

---

**ĐẬU ĐỨC SIÊU**

# PHÁT HIỆN MÃ ĐỘC DỰA TRÊN PHÂN TÍCH MẪU

**CHUYÊN NGÀNH : HỆ THỐNG THÔNG TIN**

**MÃ SỐ: 8.48.01.04**

**LUẬN VĂN THẠC SỸ KỸ THUẬT (HỆ THỐNG THÔNG TIN)**

**NGƯỜI HƯỚNG DẪN KHOA HỌC**

**TS. PHẠM HOÀNG DUY**

**HÀ NỘI – 2021**

## LỜI CAM ĐOAN

Tôi xin cam đoan luận văn về đề tài "Phát hiện mã độc dựa trên phân tích mẫu." là công trình nghiên cứu cá nhân của tôi trong thời gian qua. Mọi số liệu sử dụng phân tích trong luận văn và kết quả nghiên cứu là do tôi tự tìm hiểu, phân tích một cách khách quan, trung thực, có nguồn gốc rõ ràn. Tôi xin chịu hoàn toàn trách nhiệm nếu có sự không trung thực trong thông tin sử dụng trong luận văn

*Hà Nội, tháng 8 năm 2021*

**HỌC VIÊN**

**ĐẬU ĐỨC SIÊU**

# LỜI CẢM ƠN

Trước tiên, em xin chân thành cảm ơn các thầy cô trong Khoa Công Nghệ Thông Tin và toàn thể cán bộ của Học viện Công nghệ Bưu chính Viễn thông đã quan tâm và tạo mọi điều kiện thuận lợi cho em trong quá trình thực hiện đồ án.

Em xin gửi lời biết ơn sâu sắc nhất tới Thầy giáo TS. Phạm Hoàng Duy đã tận tình chỉ bảo, định hướng cho em trong suốt quá trình học tập và thực hiện đồ án này, đồng thời giúp em có tiếp cận với phương pháp tư duy và nghiên cứu khoa học.

Cuối cùng, em xin cảm ơn gia đình em, bạn bè em, những người đã luôn ở bên cạnh, quan tâm, giúp đỡ, động viên em để em có thể hoàn thành được đồ án này.

Em xin chân thành cảm ơn!

*Hà Nội, tháng 8 năm 2021*

**HỌC VIÊN**

**ĐẬU ĐỨC SIÊU**

# MỤC LỤC

## DANH SÁCH THUẬT NGỮ, CHỮ CÁI VIẾT TẮT

| Ký hiệu | Tên Tiếng Anh | Ý nghĩa Tiếng Việt |
|---|---|---|
| PAYL | Payload-based signatures | Chữ ký dựa trên tải |
| PIS | Privacy-Invasive Software | Phần mềm xâm phạm quyền riêng tư |
| DLL | Dynamic Link Library | Thư viện liên kết động |
| PE | Portable Executable | Có thể thực thi được |
| MBR | Master Boot Record | Bản ghi khởi động |

# DANH SÁCH BẢNG BIỂU

# DANH SÁCH HÌNH VẼ

# MỞ ĐẦU

Phần mềm độc hại đang là một mối đe dọa rất lớn về bảo mật trong thời đại kỹ thuật số ngày nay. Người dùng máy tính, các công ty và chính phủ đang chịu các cuộc tấn công sử dụng các phần mềm độc hại gia tăng theo cấp số nhân: Năm 2008, tổng các cuộc tấn công là khoảng 25 triệu, đến năm 2014, thiệt hại con số này đã tăng gấp 4 lần, lên tới 325 triệu và đến 2017 là gần 600 triệu [1]. Phân tích phần mềm độc hại trở thành một thành phần quan trọng của cơ chế bảo vệ. Phương pháp phân tích tĩnh mã độc cổ điển đã đem lại những hiệu quả cao tuy nhiên nhiều phần mềm độc hại gần đây sử dụng các kỹ thuật đa hình, biến hình và các kỹ thuật lẩn tránh khác để thay đổi hành vi của phần mềm độc hại một cách nhanh chóng và tạo ra một số lượng lớn phần mềm độc hại.

Trong thập kỷ qua, rất nhiều nghiên cứu đã được thực hiện, sử dụng các phương pháp khai phá dữ liệu trên cả phân tích tĩnh và động. Một trong những dạng phần mềm độc hại mới nhất được gọi là phần mềm độc hại có mục tiêu, chưa có nhiều nghiên cứu về nó. Phần mềm độc hại có mục tiêu, là một tập hợp của Mối đe dọa liên tục nâng cao (APT), đang phát triển về số lượng và độ phức tạp trong những năm gần đây. Tấn công mạng có mục tiêu (thông qua phần mềm độc hại được nhắm mục tiêu) ngày càng đóng một vai trò độc hại trong việc phá vỡ hệ thống tài chính và xã hội trực tuyến. APT được thiết kế để ăn cắp bí mật công ty / quốc gia và / hoặc gây tổn hại đến lợi ích quốc gia / công ty. Rất khó để nhận ra phần mềm độc hại có mục tiêu bằng các công cụ chống vi-rút, Hệ thống phát hiện xâm nhập(Intrusion Detection System - IDS), Hệ thống phòng chống xâm nhập IPS và các công cụ phát hiện phần mềm độc hại tùy chỉnh. Những kẻ tấn công tận dụng các kỹ thuật tấn công xã hội (Social Engineering) cùng với một hoặc nhiều lỗ hổng chưa được phát hiện để thực hiện triển khai các phương thức tấn công . Cùng với đó, sự ra đời gần đây của Mã hóa khóa và Ransomware(phần mềm sau khi lây nhiễm vào máy tính, mã hóa hoặc chặn truy cập dữ liệu) đặt ra những mối đe dọa nghiêm trọng đối với các tổ chức/quốc gia cũng như cá nhân.

Từ nhu cầu phát triển phân tích mã độc trên, luận văn sẽ tập trung nghiên cứu phương pháp phát hiện mã độc dựa trên phân tích mẫu với nội dung được trình bày như sau:

**Chương 1: Mã độc và một số phương pháp phân tích mã độc**

Giới thiệu chương về các khái niệm cơ bản về mã độc và phân tích mã độc cũng như một số phương pháp, công cụ xác định mã độc hiện hành.

**Chương 2: Mô hình phát hiện mã độc**

Giới thiệu chương: Chương này đưa ra mô hình chung cho việc xử lý các dữ liệu độc hại, giới thiệu cách thức để trích xuất dữ liệu sang dạng phân tích được, cuối cùng là khai phá các mẫu tuần tự để tìm ra ngưỡng phù hợp để xác định mã độc và phân tích xem dữ liệu có phải là độc hại hay không.

**Chương 3: Thử nghiệm và đánh giá**

Giới thiệu chương: Chương này giới thiệu về tập dữ liệu, cách thức thực hiện và triển khai mô hình phân tích mã độc

Nội dung chương sẽ giới thiệu quá trình thu thập dữ liệu thử nghiệm, xử lý và trích chọn đặc trưng, huấn luyện các mô hình thuật toán học máy, đưa ra kết quả và nhận xét đánh giá.

# CHƯƠNG I: TỔNG QUAN VỀ MÃ ĐỘC VÀ CÁC PHƯƠNG PHÁP PHÁT HIỆN

*Chương I trình bày khái niệm về mã độc, lịch sử phát triển và các loại mã độc phổ biến. Ngoài ra, trong chương này cũng sẽ đi tìm hiểu về các phương pháp phân tích và phát hiện mã độc cùng các nghiên cứu có liên quan.*

## 1.1 Tổng quan về mã độc

### *1.1.1. Định nghĩa*

Mã độc hay Malware (Malicious software) là một khái niệm chung dùng để chỉ các phần mềm độc hại được viết với mục đích có thể lây lan phát tán (hoặc không lây lan, phát tán) trên hệ thống máy tính và internet, nhằm thực hiện các hành vi bất hợp pháp nhằm vào người dùng cá nhân, cơ quan, tổ chức. Thực hiện các hành vi chuộc lợi cá nhân, kinh tế, chính trị hoặc đơn giản là để thỏa mãn ý tưởng và sở thích của người viết.

### *1.1.2. Lịch sử phát triển*

Lịch sử phát triển của mã độc gắn liền với lịch sử phát triển máy tình và mạng máy tính. Các virus đầu tiên là trò đùa lành tính; virus độc hại không được công khai cho đến đầu những năm 1980. Đầu tiên là Sâu (worms) , được tạo ra vào cuối năm 1970, cũng là lành tính, nhằm thực hiện bảo trì hệ thống. Mã độc đã không trở nên phổ biến cho đến cuối những năm 1980. Trong thời gian đó, hình thức phổ biến nhất là virus, đặc biệt là là các virus nhiễm vào MBR. Tại thời điểm đó, người viết virus cũng tạo ra một số kỹ thuật khiến virus của họ có thể tránh bị phát hiện. Năm 1988, Morris worm khét tiếng đã được phát hành, phá hoại hàng ngàn máy tính nối mạng. Trojan bắt đầu lộ diện vào giữa năm 1980.

Trong những năm đầu thập niên 1990, tình hình phần mềm độc hại phần lớn vẫn không thay đổi, với virus biên soạn tiếp tục là hình thức phổ biến của mã độc. Tuy nhiên, trong nửa sau của năm 1990, một số thay đổi quan trọng trong máy tính tạo ra cơ hội mới cho các phần mềm độc hại. Thứ nhất, số lượng máy tính cá nhân tăng lên rất nhiều. Ngoài ra, việc sử dụng các ứng dụng Thư điện tử và phần mềm với ngôn ngữ vĩ mô, chẳng hạn như xử lý văn bản và bảng tính, trở nên phổ biến. Theo đó, người viết virus bắt

đầu phát triển virus hiểu và lan truyền qua thư điện tử, cũng như phát triển Sâu với khả năng tương tự. Hai vụ tấn công giải thích phần mềm độc hại, virus Melissa (năm 1999) và Love-Letter worm (năm 2000), mỗi vụ tấn công ảnh hưởng hàng triệu hệ thống.

Kể từ năm 2000, Worms trở thành một dạng phần mềm độc hại phổ biến. Những người viết virus thường thích worms hơn virus vì worms có thể lây lan nhanh hơn nhiều. Trong số các loại virus, virus khởi động đã trở nên khá phổ biến, chủ yếu là do việc sử dụng suy giảm của đĩa mềm; virus macro đã trở thành loại virus phổ biến nhất. Năm 2001, lần đầu tiên loại virus tấn công trộn, Nimda, đã được phát hành, gây ảnh hưởng nghiêm trọng. Nimda có đặc điểm của virus, worms, và viru di động. Gần đây hơn, mã độc tấn công điện thoại di động đã trở nên ngày càng phổ biến, chủ yếu là do sự phổ biến của các trình duyệt Web và HTML dựa trên thư điện tử; Tuy nhiên, mã độc trên các thiết bị di động vẫn không phổ biến như worms. Một xu hướng khác là nhiều trường hợp của các phần mềm độc hại, bao gồm worms, trojan và mã độc di động, cung cấp các công cụ tấn công, chẳng hạn như các rootkit, keystroke logger, và backdoors, để hệ thống bị nhiễm.

Trong những năm gần đây, cùng với sự phát triển của tiền điện tử, các cuộc tấn công tống tiền bằng mã độc bùng nổ. Nổi bật nhất có thể kể đến cuộc tấn công WannaCry (năm 2017) lây nhiễm cho 250000 máy tính. Sau khi các máy tính bị lây nhiễm, mã độc đã thực hiện mã hoá toàn bộ dữ liệu của người dùng và bắt người dùng phải nộp cho kẻ tấn công một khoản tiền bằng bitcoin để có thể lấy lại được dữ liệu.

Với khả năng của các tin tặc, mã độc ngày ngay có thể xâm nhập bằng cách phá vỡ các hàng rào an toàn của hệ điều hành hay xâm nhập vào các lỗ hổng của các phần mềm nhất là các phần mềm thư điện tử, rồi từ đó lan tỏa khắp nơi theo các nối kết mạng hay qua thư điện tử. Do đó, việc truy tìm ra nguồn gốc phát tán virus sẽ càng khó hơn nhiều. Chính Microsoft, hãng phần mềm tạo ra các phần mềm phổ biến, cũng là một nạn nhân. Họ đã phải nghiên cứu, sửa chữa và phát hành rất nhiều các phần mềm nhằm sửa các khiếm khuyết của phần mềm cũng như phát hành các cập nhật của gói dịch vụ (service pack) nhằm giảm hay vô hiệu hóa các tấn công của mã độc. Nhưng dĩ nhiên với các phần mềm có hàng triệu dòng mã nguồn thì mong ước chúng hoàn hảo theo ý nghĩa của sự an

toàn chỉ có trong lý thuyết. Đây cũng là cơ hội cho các nhà sản xuất các loại phần mềm bảo vệ, sửa lỗi phát triển.

Trong tương lai không xa, mã độc được dự báo sẽ có thêm các bước biến đổi khác, nó bao gồm mọi điểm mạnh sẵn có (polymorphic, hay tấn công bằng nhiều cách thức, nhiều kiểu) và còn kết hợp với các thủ đoạn khác của phần mềm gián điệp (spyware). Đồng thời chúng có thể tấn công vào nhiều hệ điều hành khác nhau chứ không nhất thiết nhắm vào một hệ điều hành độc nhất như trong trường hợp của Microsoft Windows hiện nay. Và có lẽ mã độc sẽ không hề (thậm chí là không cần) thay đổi phương thức tấn công: lợi dụng điểm yếu của máy tính cũng như chương trình.

### *1.1.3. Các loại mã độc phổ biến*

#### *a) Vi rút khởi động*

Virus khởi động, là loại virus lây vào phân vùng khởi động hoặc bản ghi gốc của ổ đĩa cứng. Đây là các khu vực đặc biệt chứa các dữ liệu để khởi động hệ thống, nạp các phân vùng.

Vi rút khởi động được thực thi trước khi hệ điều hành được nạp lên. Vì vậy, nó hoàn toàn độc lập với hệ điều hành. Virus khởi động có nhược điểm là khó viết do không thể sử dụng các dịch vụ, chức năng có sẵn của hệ điều hành và kích thước virus bị hạn chế bởi kích thước của các sector (mỗi sector chỉ có 512 byte).

Ngày nay gần như không còn thấy sự xuất hiện của Virus khởi động do đặc điểm lây lan chậm và không phù hợp với thời đại Internet.

#### *b) Virus tác vụ (Macro virus)*

Đây là loại virus đặc biệt tấn công vào chương trình trong bộ Microsoft Office của Microsoft: Word, Excel, Powerpoint. Macro là tính năng hỗ trợ trong bộ công cụ văn phòng Microsoft Office cho phép người sử dụng lưu lại các công việc cần thực hiện lại nhiều lần. Thực tế hiện nay cho thấy virus macro gần như đã "tuyệt chủng".

#### *c) Virus script*

Đây là loại virus được viết bằng các ngôn ngữ script (kịch bản) như VBScript, JavaScript, Batch script. Những loại virus này thường có đặc điểm dễ viết, dễ cài đặt. Chúng thường tự lây lan sang các file script khác, thay đổi nội dung cả các file html để

thêm các thông tin quảng cáo, chèn banner … Đây cũng là một loại virus phát triển nhanh chóng nhờ sự phổ biến của Internet.

***d) Virus thực thi***

Virus này chuyên lây vào các file thực thi (ví dụ file có phần mở rộng .com, .exe, .dll) một đoạn mã để khi file được thực thi, đoạn mã virus sẽ được kích hoạt trước và tiếp tục thực hiện các hành vi phá hoại, lây nhiễm.

Loại virus này có đặc điểm lây lan nhanh và khó diệt hơn các loại virus khác do phải xử lý cắt bỏ, chỉnh sửa file bị nhiễm.

Virud thực thi có nhược điểm là chỉ lây vào một số định dạng file nhất định và phụ thuộc vào hệ điều hành. Chúng vẫn tồn tại tới ngày nay với những biến thể ngày càng trở nên nguy hiểm, phức tạp hơn.

***e) Virus gián điệp (Trojan )***

Tên của loại virus này được lấy theo một điển tích cổ. Trong cuộc chiến với người Tơ-roa, các chiến binh Hy Lạp sau nhiều ngày không thể chiếm được thành đã nghĩ ra một kế, giảng hòa rồi tặng người dân thành Tơ-roa một con ngựa gỗ khổng lồ. Sau khi ngựa gỗ được đưa vào thành, đêm đến các chiến binh Hy Lạp từ trong ngựa gỗ chui ra đánh chiếm thành.

Đây cũng chính là cách mà các Vi rút gián điệp (gọi tắt là Trojan) áp dụng: các đoạn mã của Trojan được "che giấu" trong các loại virus khác hoặc trong các phần mềm máy tính thông thường để bí mật xâm nhập vào máy nạn nhân. Khi tới thời điểm thuận lợi chúng sẽ tiến hành các hoạt động ăn cắp thông tin cá nhân, mật khẩu, điều khiển máy tính nạn nhân … Bản chất của Trojan là không tự lây lan mà phải sử dụng phần mềm khác để phát tán.

Dựa vào cách hoạt động ta có thể phân chia Trojan thành các loại sau: BackDoor, Adware và Spyware.

***f) BackDoor***

Backdoor (cửa hậu) trong phần mềm hay hệ thống máy tính thường là một cổng không được thông báo rộng rãi, cho phép người quản trị xâm nhập hệ thống để tìm nguyên nhân gây lỗi hoặc bảo dưỡng. Ngoài ra nó cũng dùng để chỉ cổng bí mật mà

hacker và gián điệp dùng để truy cập bất hợp pháp. Nhờ đó họ có thể đến và đi tùy ý, cho phép truy cập hệ thống từ xa. Mã độc cài trên hệ thống được gọi là Trojan truy cập từ xa (RAT), dùng để cài malware trên máy hoặc đánh cắp dữ liệu. Kẻ tấn công có thể cài các phần mềm BackDoor  lên nhiều máy tính khác nhau thành một mạng lưới các máy bị điều khiển sau đó thực hiện các vụ tấn công từ chối dịch vụ (DoS – Denial of Service).

***g) Adware và Spyware***

Đây là loại Trojan khi xâm nhập vào máy tính với mục đích quảng cáo hoặc "gián điệp". Chúng đưa ra các quảng cáo, mở ra các trang web, thay đổi trang mặc định của trình duyệt … gây khó chịu cho người sử dụng. Các phần mềm này cài đặt các phần mềm ghi lại thao tác bàn phím , ăn cắp mật khẩu và thông tin cá nhân …

***h) Worm***

Cùng với các loại mã độc máy tính như Trojan, WannaCry, Worm (sâu máy tính) là loại virus phát triển và lây lan mạnh mẽ nhất hiện nay nhờ mạng Internet.

Vào thời điểm ban đầu, Worm được tạo ra chỉ với mục đích phát tán qua thư điện tử. Khi lây vào máy tính, chúng thực hiện tìm kiếm các sổ địa chỉ, danh sách thư điện tử trên máy nạn nhân rồi giả mạo các thư điện tử để gửi bản thân chúng tới các địa chỉ thu thập được.

Các thư điện tử do worm tạo ra thường có nội dung "giật gân", hoặc "hấp dẫn", hoặc trích dẫn một thư điện tử nào đó ở máy nạn nhân để ngụy trang. Điều này khiến các thư điện tử giả mạo trở nên "thật" hơn và người nhận dễ bị đánh lừa hơn. Nhờ những thư điện tử giả mạo đó mà Worm lây lan mạnh mẽ trên mạng Internet theo cấp số nhân.

Bên cạnh Worm lây lan theo cách truyền thống sử dụng thư điện tử, Worm hiện nay còn sử dụng phương pháp lân lan qua ổ USB. Thiết bị nhớ USB đã trở nên phổ biến trên toàn thế giới do lợi thế kích thước nhỏ, cơ động và trở thành phương tiện lây lan lý tưởng cho Worm.

Dựa đặc điểm lây lan mạnh mẽ của Worm, những kẻ viết virus đã đưa thêm vào Worm các tính năng phá hoại, ăn cắp thông tin…, Worm thường được sử dụng cùng với các phần mềm độc hại khác như BackDoor, Adware…

***i) Rootkit***

Rootkit ra đời sau các loại virus khác, nhưng rootkit lại được coi là một trong những loại virus nguy hiểm nhất.

Bản thân rootkit không thực sự là virus, đây là phần mềm hoặc một nhóm các phần mềm máy tính được giải pháp để can thiệp sâu vào hệ thống máy tính (nhân của hệ điều hành hoặc thậm chí là phần cứng của máy tính) với mục tiêu che giấu bản thân nó và các loại phần mềm độc hại khác.

Với sự xuất hiện của rootkit, các phần mềm độc hại như trở nên "vô hình" trước những công cụ thông thường thậm chí vô hình cả với các phần mềm diệt virus. Việc phát hiện mã độc và tiêu diệt virus trở nên khó khăn hơn rất nhiều trước sự bảo vệ của rootkit – vốn được trang bị nhiều kĩ thuật mới hiện đại.

Xuất hiện lần đầu trên hệ thống Unix từ khá lâu, nhưng kể từ lần xuất hiện "chính thức" trên hệ điều hành Windows vào năm 2005, Rootkit đang dần trở nên phổ biến và trở thành công cụ che giấu hữu hiệu cho các loại phần mềm độc hại khác.

***j) Botnet***

Từ "botnet" là sự kết hợp của hai từ, "robot" **và** "network"**.** Ở đây, một tên tội phạm mạng thực hiện vai trò của một "botmaster" sử dụng virus để xâm phạm bảo mật của một số máy tính và kết nối chúng vào mạng vì mục đích xấu. Mỗi máy tính trên mạng hoạt động như một "bot", và được kẻ xấu kiểm soát để lây truyền mã độc, spam hoặc nội dung độc hại nhằm khởi động cuộc tấn công. Botnet còn được gọi là đội quân zombie vì các máy tính liên quan đang được điều khiển bởi một người khác không phải chủ sở hữu của chúng.

***k) Keylogger***

Ý tưởng đằng sau phần mềm độc hại này là ghi lại tất cả các phím do người dùng nhấn và lưu trữ tất cả dữ liệu, bao gồm mật khẩu, số thẻ ngân hàng và thông tin nhạy cảm khác.

***l) Ransomware***

Loại phần mềm độc hại này nhằm mục đích mã hóa tất cả dữ liệu trên máy và yêu cầu nạn nhân chuyển tiền để lấy khóa giải mã. Thông thường, một máy bị nhiễm phần mềm ransomware bị "đóng băng" vì người dùng không thể mở bất kỳ tệp nào, ảnh trên

màn sẽ là thông tin về các yêu cầu của kẻ tấn công.

## 1.2 Các phương pháp phát hiện mã độc

### *1.2.1 Các kỹ thuật phân tích mã độc*

Để phát hiện một chương trình có phải mã độc không cần phải thực hiện quá trình phân tích mã chương trình. Có hai phương pháp chính để thực hiện phân tích mã độc gồm: Phân tích tĩnh và Phân tích động.

***a) Phân tích tĩnh***

Phân tích tĩnh là kỹ thuật sử dụng các công cụ để đọc một phần hoặc toàn bộ mã nguồn của chương trình độc hại và từ đó cố gắng suy ra được đặc tính hành vi của chương trình. Chương trình độc hại có thể được viết bằng nhiều ngôn ngữ khác nhau, phổ biến nhất là assembly. Vì vậy có nhiều phương pháp phân tích tĩnh khác nhau như:

- *Kiểm tra định dạng tệp*: dữ liệu thô có thể cung cấp thông tin hữu ích. Ví dụ, các tệp Windows PE (thực thi) có thể cung cấp nhiều thông tin về thời gian biên dịch, bảng import, export v.v.
- *String Extraction*: đề cập đến việc kiểm tra đầu ra phần mềm (ví dụ: trạng thái hoặc thông báo lỗi) để có thể đánh giá thông tin về hoạt động của phần mềm độc hại.
- *Fingerprinting*: bao gồm tính toán băm mật mã, các giá trị biến môi trường, chẳng hạn như username, password, registry, …
- *AV scanning*: nếu tệp được kiểm tra là phần mềm độc hại nổi tiếng, rất có thể tất cả các trình quét vi-rút sẽ có thể phát hiện được. Mặc dù nó có vẻ không mang lại kết quả cao nhưng cách phát hiện này thường được các nhà cung cấp sử dụng.
- *Disassembly*: đề cập đến việc dịch ngược mã máy thành ngôn ngữ con người có thể hiểu và suy ra logic và ý định phần mềm. Đây là phương pháp phân tích tĩnh phổ biến và đáng tin cậy nhất.

Phân tích tĩnh thường dựa vào một số công cụ nhất định. Ngoài việc phân tích 1 số thông tin cơ bản, họ có thể cung cấp thông tin về kỹ thuật bảo vệ được phần mềm độc hại sử dụng. Ưu điểm của phân tích tĩnh là có thể tìm ra tất cả kịch bản thực thi có thể có của mã độc mà không bị hạn chế về bất kỳ điều kiện gì. Hơn nữa, phân tích tĩnh an toàn hơn

phân tích động bởi không cần thực thi mã độc trực tiếp, vì thế sẽ không gây nguy hiểm cho hệ thống. Tuy nhiên phân tích tĩnh lại tốn rất nhiều thời gian, vì thế phân tích tĩnh thường không được sử dụng trong thực tế mà thường dùng để nghiên cứu, ví dụ khi nghiên cứu chữ ký cho các mã độc zero-day

***b) Phân tích động***

Không giống với phân tích tĩnh ở chỗ, các hành vi của mã độc được giám sát trong khi nó đang thực thi, từ đó có thể tìm hiểu được thuộc tính và mục đích của mã độc. Thông thường mã độc sẽ được thực thi trong môi trường ảo (vd. Sandbox). Trong quá trình phân tích sẽ phát hiện tất cả hành vi của mã độc, như mở tệp tin, tạo mutexes, ... và kiểu phân tích này sẽ nhanh hơn phân tích tĩnh rất nhiều. Tuy nhiên, phân tích động chỉ biết được hành vi của mã độc trong hệ thống ảo dùng để kiểm tra, ví dụ kết quả thu được khi thực thi hai mã độc giống nhau trong môi trường Windows 7 và Windows 8.1 sẽ khác nhau

### *1.2.2 Các phương pháp hiện mã độc*

Kỹ thuật phân tích mã độc là bước đầu trong quá trình phát hiện mã độc. Chính vì lý do này mà phương pháp phát hiện mã độc cũng được chia thành hai nhóm chính dựa trên các kỹ thuật phân tích mã độc phổ biến. Các phương pháp này gồm: Phương pháp phát hiện dựa trên chữ ký và Phương pháp phát hiện dựa trên hành vi.

**a) Phương pháp phát hiện dựa trên chữ ký**

Phân tích dựa trên chữ ký là một phương pháp tĩnh dựa trên các chữ ký được xác định trước. Đây có thể là tệp vân tay, ví dụ: MD5 hoặc SHA1 băm, chuỗi tĩnh, siêu dữ liệu tệp.

Kịch bản phát hiện, trong trường hợp này, sẽ như sau: khi một tệp đến hệ thống, nó được phân tích tĩnh bởi phần mềm chống vi-rút. Nếu bất kỳ chữ ký nào được khớp, cảnh báo được kích hoạt, cho biết tệp này là đáng ngờ. Đa phần phân tích kiểu này là đủ vì các mẫu phần mềm độc hại nổi tiếng thường có thể được phát hiện dựa trên giá trị băm.

**b) Phương pháp phát hiện dựa trên hành vi**

Tuy nhiên, những kẻ tấn công bắt đầu phát triển phần mềm độc hại theo cách nó

có thể thay đổi chữ ký của nó. Tính năng phần mềm độc hại này được gọi là đa hình (polymorphism). Rõ ràng, phần mềm độc hại như vậy không thể được phát hiện bằng cách sử dụng chữ ký. Hơn nữa, các loại phần mềm độc hại mới cũng không thể phát hiện bằng cách sử dụng chữ ký, cho đến khi chữ ký được tạo ra. Do đó, các nhà cung cấp AV phải đưa ra một cách khác để phát hiện - dựa trên hành vi.

Trong phương pháp này, hành vi thực tế của phần mềm độc hại được quan sát trong quá trình thực thi, tìm kiếm các dấu hiệu của hành vi nguy hiểm: sửa đổi tệp lưu trữ, registry, thiết lập kết nối đáng ngờ,.v.v… Bản thân, mỗi hành động này không thể là dấu hiệu của phần mềm độc hại, nhưng sự kết hợp của chúng có thể làm tăng mức độ đáng ngờ của tệp.

Mức độ chính xác của phát hiện dựa trên hành vi dựa vào quá trình thực thi. Tốt nhất là sử dụng môi trường ảo, ví dụ như sandbox, để chạy tệp tin và giám sát hành vi của nó. Mặc dù phương pháp này tốn nhiều thời gian hơn, nhưng nó an toàn hơn, do tệp tin được kiểm tra trước khi chạy thực tế. Ưu điểm chính của phát hiện dựa trên heuristics là nó không chỉ phát hiện các mã độc đã biết mà còn phát hiện được các cuộc tấn công zero-day và các loại virus đa hình. Tuy nhiên, một số loại mã độc có khả năng phát hiện môi trường ảo, nó sẽ không thực thi các hành vi độc hại trong môi trường sandbox. Hơn nữa, trên thực tế, với lượng mã độc đang ngày một gia tăng, phương pháp này không thực sự hiệu quả trước các loại mã độc mới.

### 1.3 Các nghiên cứu liên quan

Như đã giới thiệu ở trên phần lớn các hệ thống phát hiện mã độc đều dựa vào signature hoặc là chính bằng sức người. Những hệ thống hoạt động như thế công suất cũng như độ hiệu quả không hề cao và có thể dẫn đến nhiều sai sót.

Trước sự phát triển nhanh chóng của các loại mã độc đa hình, việc phát triển một phương pháp phát hiện mới là rất cần thiết. Một trong các giải pháp là kết hợp phát hiện dựa trên hành vi với các kỹ thuật học máy để tăng độ chính xác.

Ngoài ra, học máy cũng giúp tự động hóa quá trình phát hiện mã độc. Dựa trên việc “học” các đặc trưng mẫu, các kỹ thuật học máy có thể tự động thêm các đặc trưng

mới và tối ưu hóa tập đặc trưng, loại bỏ các đặc trưng dư thừa. Tuy nhiên, với lượng dữ liệu khổng lồ, để có kết quả tốt nhất, học máy cần kết hợp với các công nghệ khác như Big Data và Điện toán đám mây.

Mặc dù chưa được ứng dụng rộng rãi, ý tưởng sử dụng học máy trong việc phát hiện mã độc không còn mới. Một số nghiên cứu về lĩnh vực này đã được thực hiện nhằm kiểm tra tính chính xác của các phương pháp khác nhau.

Wang và Stolfo [4] trình bày PAYL, một công cụ tính toán tải trọng dự kiến cho mỗi dịch vụ (cổng) trên hệ thống. Phân phối tần số byte được tạo cho phép phát triển mô hình cho từng dịch vụ của máy chủ lưu trữ. Mô hình này được tính toán trong giai đoạn học tập. Máy dò so sánh trọng tải đến với mô hình, đo khoảng cách giữa hai loại.

Khoảng cách không chỉ tính đến các giá trị trung bình của một vectơ đặc trưng mà còn tính đến phương sai và hiệp phương sai mang lại một thước đo thống kê mạnh hơn về độ tương tự. Nếu trọng tải đến quá xa so với mô hình (giá trị khoảng cách lớn), thì trọng tải được coi là độc hại.

Boldt và Carlson [3] đưa ra khái niệm về phần mềm xâm phạm quyền riêng tư (PIS). Phần mềm quảng cáo và phần mềm gián điệp là những loại phần mềm xâm phạm quyền riêng tư chính. Thông thường, PIS được sử dụng như một phần của phần mềm chia sẻ tệp. Boldt và Carlson sử dụng Bộ công cụ FTK để giúp xác định PIS. Cách tiếp cận cơ bản bao gồm ban đầu tạo ra một hệ thống không có PIS, một hệ thống "sạch". Ảnh chụp nhanh của hệ thống sạch được coi là đường cơ sở của hệ thống. Ảnh chụp nhanh mô tả hệ thống tệp của máy chủ đích. Sau khi đường cơ sở được ghi lại, một số hành động được thực hiện để có khả năng giải phóng PIS trên máy chủ đích. Ví dụ, hành động có thể là lướt World Wide Web. Ảnh chụp nhanh sẽ được thực hiện đều đặn. Ad-Aware là công cụ loại bỏ PIS phổ biến nhất và do đó, các tác giả đã chọn đánh giá Ad-Aware bằng cách sử dụng các kỹ thuật phân tích tĩnh và pháp y. Thông qua việc sử dụng kỹ thuật của họ, Boldt và Carlson nhận thấy rằng Ad-Aware tạo ra các kết quả dương tính giả cũng nhưPhân tích tệp tin

Alazab [5] đã đề xuất phương pháp sử dụng API để biểu diễn đặc trưng của mã độc. Thuật toán Support Vector Machine cho kết quả tốt nhất với tỉ lệ 97.6%, tỉ lệ dương

tính sai là 0.025.

Baldangombo và cộng sự đã giới thiệu phương pháp trích chọn đặc trưng dựa trên tiêu đề PE, các thư viện DLL và các hàm chức năng API [6]. Các thuật toán được sử dụng bao gồm Naïve Bayes, Cây quyết định J48, và Support Vector Machines. Thuật toán có kết quả tốt nhất là J48, với tỉ lệ chính xác lên tới 99%.

Dragos Gavrilut [7] đã đề xuất hệ thống phát hiện mã độc dựa trên các thuật toán perceptron cải tiến. Với các thuật toán khác nhau, độ chính xác dao động trong khoảng 69.90% - 96.18%. Tuy nhiên thuật toán có độ chính xác cao nhất cũng có nhiều kết quả dương tính sai nhất. Thuật toán cân đối nhất có tỉ lệ dương tính sai thấp và có độ chính xác là 93.01%.

Singhal và Raul đã thảo luận về phương pháp phát hiện dựa trên thuật toán Random Forest cải tiến kết hợp với Information Gain để biểu diễn đặc trưng tối ưu hơn [8]. Tập dữ liệu được tác giả sử dụng chỉ bao gồm tệp tin thực thi, vì thế việc trích chọn đặc trưng đơn giản hơn. Tỷ lệ phát hiện là 97% và tỉ lệ dương tính sai là 0.03.

Kết quả đưa ra của các nghiên cứu ở trên đều không giống nhau, do chưa có một phương pháp thống nhất trong việc phát hiện cũng như biểu diễn đặc trưng. Độ chính xác của từng trường hợp còn phụ thuộc vào các loại mã độc được dùng để lấy mẫu và quá trình chạy thực tế.

## 1.4 Kết luận chương

Chương I đã trình bày về khái niệm mã độc, lịch sử hình thành mã độc, phân loại và giới thiệu các một số loại mã độc phổ biến. Có thể thấy, mã độc ngày càng phát triển mạnh mẽ với nhiều biến thể khác khác nhau. Chính vì vậy, chương I cũng đã đưa ra các kỹ thuật phân tích mã độc làm cơ sở cho các phương pháp phát hiện mã độc. Trong chương II, luận văn sẽ tìm hiểu sâu hơn vào học máy và kỹ thuật phân tích, trích xuất mã lệnh (lệnh vận chuyển dữ liệu) của mã độc nhằm mục đích xây dựng phương pháp phát hiện mã độc dựa trên phân tích mẫu.

# CHƯƠNG II: MÔ HÌNH PHÁT HIỆN MÃ ĐỘC

*Chương II giới thiệu tổng quan về học máy và các thuật toán học máy phổ biến. Tiếp đó, chương sẽ trình bày về kỹ thuật phân tích mã độc để trích xuất mã lệnh và xây dựng phương pháp phát hiện mã độc dựa trên phân tích mẫu kết hợp với học máy.*

## 2.1 Tổng quan về học máy

### *2.1.1 Khái niệm học máy*

Một trong những khác biệt lớn nhất giữa con người và máy tính là khả năng tự học hỏi từ kinh nghiệm hay dữ liệu đã có. Muốn thực hiện được một chức năng nào đó, máy tình phải được lập trình những logic phức tạp để có thể thực hiện theo một quy trình có sẵn.

Cùng với sự phát triển của khoa học công nghệ, con người càng mong muốn giảm thiểu quá trình lập trình logic phức tạp cho máy tình mà muốn máy tính cũng có khả năng tự học hỏi như con người. Từ đó, nhiều nhà nghiên cứu đã tập trinh nhiều hơn vào lĩnh vực trí tuệ nhân tạo.

Học máy (hay Machine Learning) là một công nghệ phát triển từ lĩnh vực trí tuệ nhân tạo. Các thuật toán học máy là các chương trình máy tính có khả năng học hỏi về cách hoàn thành các nhiệm vụ và cách cải thiện hiệu suất theo thời gian.

Học máy vẫn đòi hỏi sự đánh giá của con người trong việc tìm hiểu dữ liệu cơ sở và lựa chọn các kĩ thuật phù hợp để phân tích dữ liệu. Đồng thời, trước khi sử dụng, dữ liệu phải sạch, không có sai lệch và không có dữ liệu giả.

Các mô hình học máy yêu cầu lượng dữ liệu đủ lớn để "huấn luyện" và đánh giá mô hình. Trước đây, các thuật toán học máy thiếu quyền truy cập vào một lượng lớn dữ liệu cần thiết để mô hình hóa các mối quan hệ giữa các dữ liệu. Sự tăng trưởng trong dữ liệu lớn (big data) đã cung cấp các thuật toán học máy với đủ dữ liệu để cải thiện độ chính xác của mô hình và dự đoán.

### *2.1.2 Các phương học máy phổ biến*

Dự theo phương thức học, các thuật toán Machine Learning thường được chia làm 4 nhóm: Học có giám sát, Học phi giám sát, Học bán giám sát và học tăng cường. Ngoài ra có một số cách phân nhóm không có Học bán giám sát  hoặc Học tăng cường.

- Supervised learning (hay học có giám sát): là phương pháp sử dụng những dữ liệu đã được gán nhãn từ trước để suy luận ra quan hệ giữa đầu vào và đầu ra. Các dữ liệu này được gọi là dữ liệu huấn luyện và chúng là cặp các đầu vào-đầu ra. Học có giám sát sẽ xem xét các tập huấn luyện này để từ đó có thể đưa ra dự đoán đầu ra cho 1 đầu vào mới chưa gặp bao giờ. Ví dụ dự đoán giá nhà, phân loại thư điện tử.
- Unsupervised learning (hay học phi giám sát): Khác với học có giám sát, học phi giám sát sử dụng những dữ liệu chưa được gán nhãn từ trước để suy luận. Phương pháp này thường được sử dụng để tìm cấu trúc của tập dữ liệu. Tuy nhiên lại không có phương pháp đánh giá được cấu trúc tìm ra được là đúng hay sai. Ví dụ như phân cụm dữ liệu, trích xuất thành phần chính của một chất nào đó.
- Semi-supervised learning (hay học nửa giám sát): là một lớp của kỹ thuật học máy, sử dụng cả dữ liệu đã gán nhãn và chưa gán nhãn để huấn luyện – điển hình là một lượng nhỏ dữ liệu có gán nhãn cùng với lượng lớn dữ liệu chưa gán nhãn. Học nửa giám sát đứng giữa học không giám sát (không có bất kỳ dữ liệu có nhãn nào) và có giám sát (toàn bộ dữ liệu đều được gán nhãn). Nhiều nhà nghiên cứu nhận thấy dữ liệu không gán nhãn, khi được sử dụng kết hợp với một chút dữ liệu có gán nhãn, có thể cải thiện đáng kể độ chính xác. Để gán nhãn dữ liệu cho một bài toán học máy thường đòi hỏi một chuyên viên có kỹ năng để phân loại bằng tay các ví dụ huấn luyện. Chi phí cho quy trình này khiến tập dữ liệu được gán nhãn hoàn toàn trở nên không khả thi, trong khi dữ liệu không gán nhãn thường tương đối rẻ tiền. Trong tình huống đó, học nửa giám sát có giá trị thực tiễn lớn lao.
- Reinforcement learning (hay học tăng cường): Phương pháp học tăng cường tập trung vào việc làm sao để cho 1 tác tử trong môi trường có thế hành động sao cho lấy được phần thưởng nhiều nhất có thể. Khác với học có giám sát nó không có cặp dữ liệu gán nhãn trước làm đầu vào và cũng không có đánh giá các hành động là đúng hay sai.

### *2.1.3 Quy trình huấn luyện mô hình học máy*

Quá trình xây dựng một mô hình học máy cơ bản cần được dựa theo các bước như sau:

- **Chuẩn bị dữ liệu:** Đầu tiên cần sử dụng các phương pháp, kỹ thuật khác nhau để thu thập được một tập dữ liệu phù hợp với bài toán cần giải quyết. Tập dữ liệu này cần đủ lớn và có tính đa dạng.
- **Chọn mô hình**: Chọn một mô hình thống kê cho tập dữ liệu. Ví dụ như mô hình thống kê Bernoulli, mô hình phân phối chuẩn.
- **Tìm tham số**: Các mô hình thống kê có các tham số tương ứng, nhiệm vụ lúc này là tìm các tham số này sao cho phù hợp với tập dữ liệu nhất có thể. Đây cũng chính là quá trình huấn luyện mô hình.
- **Suy luận**: Sau khi có được mô hình và tham số, có thể dựa vào đó để đưa ra suy luận cho một đầu vào mới.

Để một mô hình đạt được hiệu quả cao, quá trình chuẩn bị dữ liệu gần như là quá trình quan trọng nhất, đây cũng được coi là quá trình tiên quyết. Dữ liệu sau khi thu thập được cần phải:

a) **Chuẩn hoá**: Tất cả các dữ liệu đầu vào đều cần được chuẩn hoá để máy tính có thể xử lý được. Quá trình chuẩn hoá bao gồm số hoá dữ liệu, điều chỉnh thông số cho phù hợp với bài toán. Việc chuẩn hoá ảnh hưởng trực tiếp tới tốc độ huấn luyện cũng như cả hiệu quả huấn luyện. Có nhiều phương pháp chuẩn hoá dữ liệu khác nhau, phổ biến nhất có thể kể đến các phương pháp sau:

- Trung tâm hóa dữ liệu (centering data): Là phương pháp đưa dữ liệu về xoay quanh giá trị trung tâm của dữ liệu thông qua công thức:

$$x' = x - average(x)$$

với $x$ là dữ liệu gốc, $average(x)$ là giá trị trung bình của toàn tập dữ liệu, $x'$ là dữ liệu mới thu được.

- Chuẩn hóa min-max (rescaling): là phương pháp đơn giản nhất trong việc tăng/giảm phạm vi của đặc trưng bằng việc tăng/giảm về phạm vi [0,1] hoặc [-1,1]. Công thức chung được cho như sau:

$$x' = \frac{x - min(x)}{(x) - min(x)}$$

với $max(x), min(x)$ là giá trị lớn nhất và nhỏ nhất của toàn tập dữ liệu.

- Co giãn trung bình (mean normalization): Tương tự như phương pháp rescaling, phương pháp co giãn xoay quanh trung bình có giá trị nằm trong khoảng [-0.5, 0.5] và được cho bởi công thức:

$$x' = \frac{x - average(x)}{(x) - min(x)}$$

- Chính quy hóa (standardisation): Được tính bằng công thức:

$$x' = \frac{x - average(x)}{std(x)}$$

với $std(x)$ là độ lệch chuẩn của toàn tập dữ liệu.

b) **Trích chọn đặc trưng:** Mục đích của quá trình này là thu được một tập dữ liệu chi tiết và không dư thừa. Các đặc trưng phải biểu diễn thông tin quan trọng và liên quan tới tập dữ liệu nếu không kết quả dự đoán sẽ không chính xác. Vì thế, trích xuất đặc trưng là một nhiệm vụ không rõ ràng, cần rất nhiều nghiên cứu và thử nghiệm. Hơn nữa, với mỗi lĩnh vực, đặc trưng của dữ liệu cũng khác nhau, nên không có phương pháp chung cho việc trích xuất đặc trưng.

c) **Phân chia**: Việc mô hình được chọn rất khớp với tập dữ liệu đang có không có nghĩa là giả thuyết trước đó là đúng mà có thể xảy ra tình huống dữ liệu thật lại không khớp. Vấn đề này trong học máy được gọi là khớp quá (*Overfitting*). Vì vậy khi huấn luyện phải phân chia dữ liệu ra thành 3 loại để có thể kiểm chứng được phần nào mức độ tổng quát của mô hình. Cụ thể 3 loại đó là:

- **Tập huấn luyện** (*Training set*): Chiếm 60%. Dùng để học khi huấn luyện.
- **Tập kiểm chứng** (*Cross validation set*): Chiếm 20%. Dùng để kiểm thử mô hình khi huấn luyện.
- **Tập kiểm tra** (*Test set*): Chiếm 20%. Dùng để kiểm tra xem mô hình đã phù hợp chưa sau khi huấn luyện. Dữ liệu tập kiểm tra cần đảm bảo là độc lập hoàn toàn với tập huấn luyện và tập kiểm thử.

## 2.2 Một số kỹ thuật học máy phổ biến

### *2.2.1 Đinh lý Bayes*

Naive Bayes là một thuật toán dựa trên định lý Bayes về lý thuyết xác suất để đưa ra các phán đoán cũng như phân loại dữ liệu dựa trên các dữ liệu được quan sát và thống kê, được ứng dụng rất nhiều trong các lĩnh vực Machine learning dùng để đưa các dự đoán có độ chính xác cao, dựa trên một tập dữ liệu đã được thu thập. Ý tưởng chính của thuật toán là tính toán xác suất của mỗi đặc trưng một cách độc lập, sau đó đưa ra dự đoán dựa trên định lý Bayes.

Định lý Bayes cho phép tính xác suất xảy ra của một sự kiện ngẫu nhiên $A$ khi biết sự kiện liên quan $B$ đã xảy ra. Xác suất này được ký hiệu là $P(A|B)$, và đọc là "xác suất của $A$ nếu có $B$". Đại lượng này được gọi xác suất có điều kiện hay xác suất hậu nghiệm vì nó được rút ra từ giá trị được cho của $B$ hoặc phụ thuộc vào giá trị đó. Công thức của định lý Bayes như sau:

$$P(B) = \frac{P(B|A)P(A)}{P(B)}$$

Như vậy, $P(B)$ sẽ phụ thuộc 3 yếu tố:

2 Xác suất xảy ra $A$ của riêng nó, không quan tâm đến $B$. Kí hiệu là $P(A)$.

3 Xác suất xảy ra $B$ của riêng nó, không quan tâm đến $A$. Kí hiệu là $P(B)$.

4 Xác suất xảy ra $B$ khi biết $A$ xảy ra. Kí hiệu là $P(B|A)$. Đại lượng này gọi là khả năng (likelihood) xảy ra $B$ khi biết $A$ đã xảy ra.

Định lý Bayes được mở rộng ra khi sự kiện $A$ phụ thuộc vào xác suất xảy ra của các sự kiện $B_1, B_2, \dots, B_n$ bằng công thức sau:

$$P(B) = \frac{(P(B_1|A) \times P(B_2|A) \times \dots \times P(B_n|A)) \times P(A)}{P(B_1) \times P(B_2) \times \dots \times P(B_n)}$$

Ứng dụng định lý Bayes, thuật toán Naive Bayes hoạt động như sau:

Gọi $D$ là tập dữ liệu huấn luyện, trong đó mỗi phần tử dữ liệu $X$ được biểu diễn bằng một vector chứa $n$ giá trị thuộc tính $A_1, A_2, \dots, A_n = \{x_1, x_2, \dots, x_n)$

Giả sử có m lớp $C_1, C_2, \dots, C_m$. Cho một phần tử dữ liệu $X$, bộ phân lớp sẽ gán nhãn

cho $X$ là lớp có xác suất hậu nghiệm lớn nhất. Cụ thể, bộ phân lớp Bayes sẽ dự đoán $X$ thuộc vào lớp $C_i$ nếu và chỉ nếu: $P(X) > P(C_j|X)(1 \leq i, j \leq m, i \neq j)$ với các giá trị xác suất được tính dựa trên định lý Bayes.

Để tìm xác suất lớn nhất, ta nhận thấy các giá trị $P(X)$ là giống nhau với mọi lớp nên không cần tính. Do đó ta chỉ cần tìm giá trị lớn nhất của $P(X) \times P(C_i)$. Chú ý rằng $P(X)$ được ước lượng bằng $\frac{|D_i|}{|D|}$, trong đó $D_i$ là tập các phần tử dữ liệu thuộc lớp $C_i$. Nếu xác suất tiền nghiệm $P(C_i)$ cũng không xác định được thì ta coi chúng bằng nhau $P(C_1) = P(C_2) = \cdots = P(C_m)$, khi đó ta chỉ cần tìm giá trị $P(X)$ lớn nhất.

Khi số lượng các thuộc tính mô tả dữ liệu là lớn thì chi phí tính toàn $P(X)$ là rất lớn, dó đó có thể giảm độ phức tạp của thuật toán Naive Bayes giả thiết các thuộc tính độc lập nhau. Khi đó ta có thể tính: $P(C_i) = P(C_i) \times P(C_i) \times \ldots \times P(x_n|C_i)$

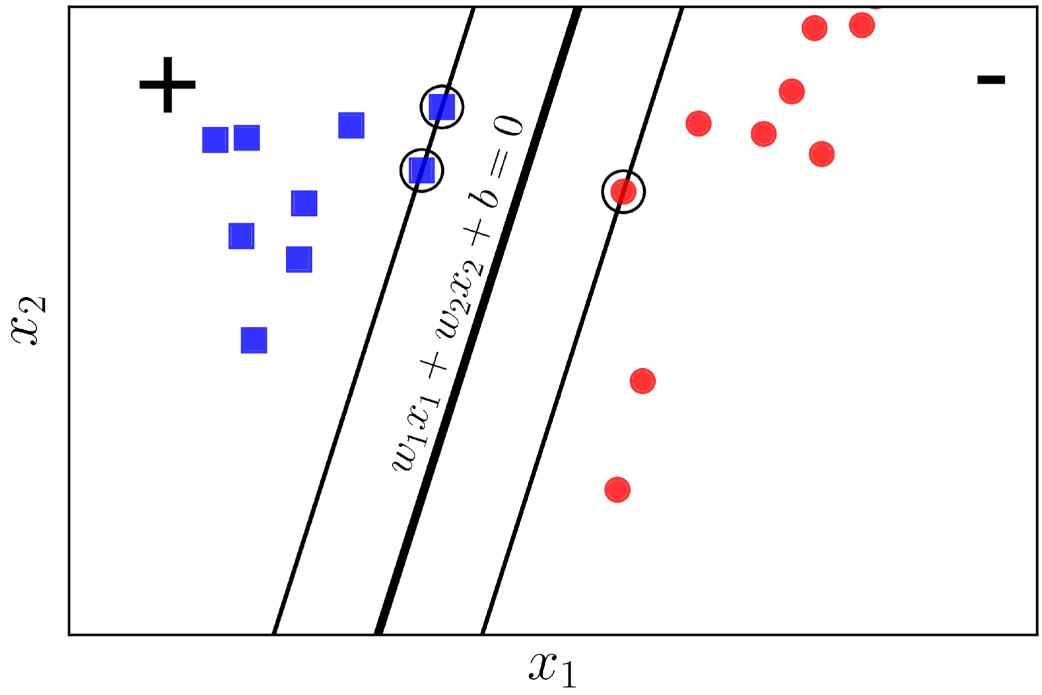
Ưu điểm của thuật toán là đơn giản và dễ hiểu. Hơn nữa, nó phù hợp với các tập dữ liệu có nhiều đặc trưng khác nhau, bởi dự đoán phụ thuộc vào xác suất của các đặc trưng. Ngoài ra, thuật toán tiêu tốn ít tài nguyên, có hiệu năng cao, không cần tính toán các hệ số phụ như các thuật toán khác.

### *2.2.2 Support Vector Machine*

Support Vector Machines (SVM) là một thuật toán phổ biến thường được dùng trong các bài toán phân lớp. Ý tưởng chính là tìm kiếm một siêu mặt phẳng phân chia các lớp một cách tối ưu nhất. Khoảng cách giữa support vector và siêu mặt phẳng được gọi là khoảng cách biên (margin).

Khoảng cách biên càng lớn thì mặt phẳng quyết định càng tốt đồng thời việc phân loại càng chính xác. Mục đích thuật toán SVM tìm ra được khoảng cách biên lớn nhất để tạo kết quả phân lớp tốt.

Dữ liệu huấn luyện của mô hình SVM là tập các cặp dữ liệu $\{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$ với $x_i \in R^d, d$ là số chiều của dữ liệu, đại diện cho toạ độ trong không gian của dữ liệu đầu vào, $y_i$ là nhãn của dữ liệu đó và $n$ là số lượng dữ liệu đầu vào. Ví dụ như hình sau:




*Hình II-1: Ví dụ SVM bài toán phân loại 2 lớp (nguồn: machinelearningcoban.com)*

Dữ liệu đầu vào thuộc mặt phẳng 2 chiều, có nhãn $y_i = 1$ đại diện bởi các điểm hình vuông màu xanh hoặc $y_i = -1$ đại diện bởi các điểm hình tròn màu đỏ và có đường thẳng $w^T x + b = w_1 x_1 + w_2 x_2 + b = 0$ làm mặt phẳng phân chia lớp. Để ý thấy khoảng cách từ một cặp dữ liệu bất kỳ $(x_n, y_n)$ bất kỳ tới mặt phẳng phân chia lớp là:

$$\frac{y_n(w^T x_n + b)}{||w||_2}$$

Như vậy margin sẽ là khoảng cách gần nhất từ một điểm dữ liệu tới mặt phẳng phân chia lớp. Margin được tính như sau:

$$margin = \frac{y_n(w^T x_n + b)}{||w||_2}$$

Quá trình huấn luyện SVM chính là đi tối ưu để tìm $w$ và $b$ sao cho giá trị margin là lớn nhất:

$$(w, b) = arg\left\{\frac{y_n(w^T x_n + b)}{||w||_2}\right\} = arg\{\frac{1}{||w||_2}[y_n(w^T x_n + b)]\}$$

Thuật toán SVM thường cho kết quả khá chính xác, đặc biệt là đối với các tập dữ liệu "sạch". Hơn nữa, nó còn phù hợp với các tập dữ liệu nhiều chiều, kể cả khi số chiều nhiều hơn số lượng mẫu. Nó cũng hiệu quả với các tập dữ liệu có nhiều nhiễu hoặc chồng chéo nhau. Tuy nhiên, thời gian huấn luyện có thể rất lâu.

### *2.2.3 Decision Tree*

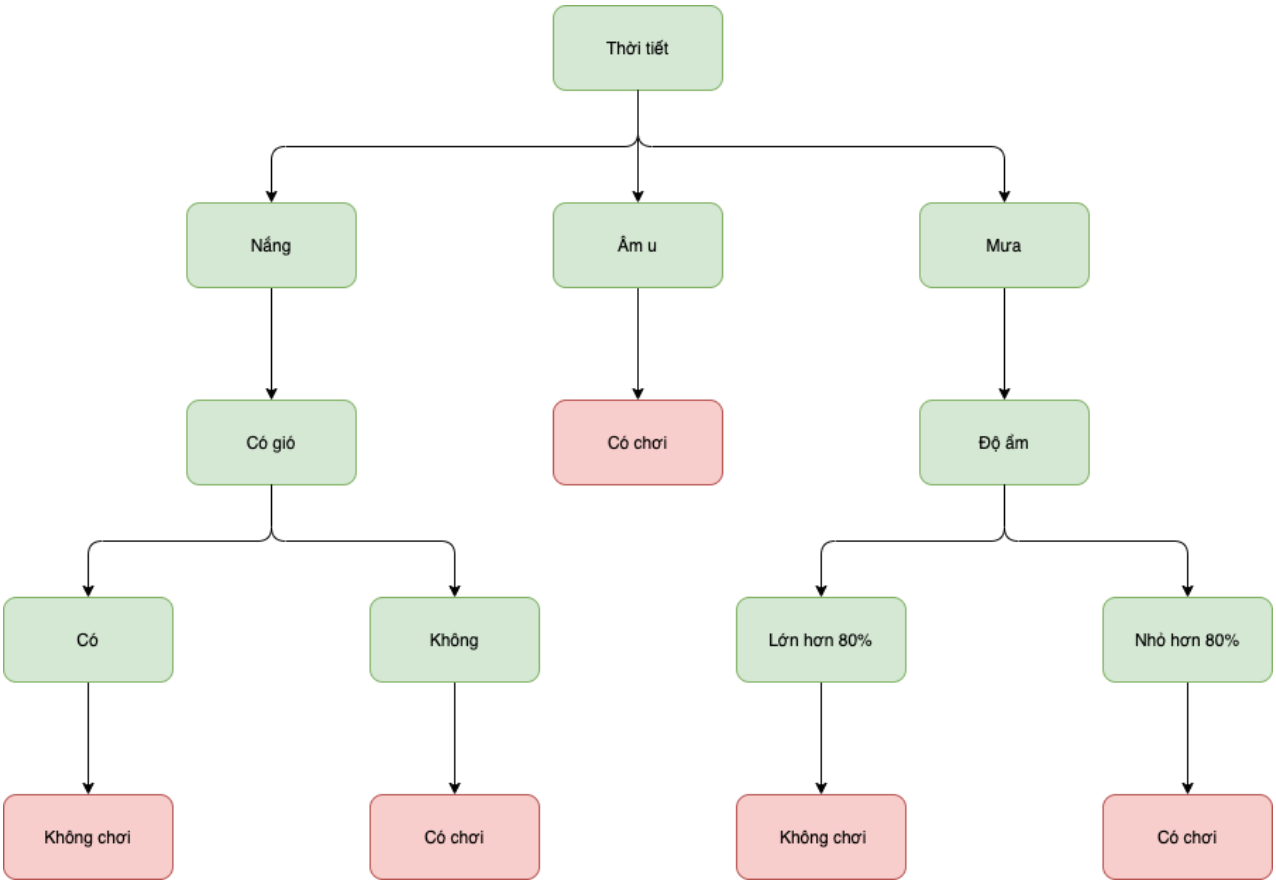
Decision Tree (Cây quyết định) là một cây phân cấp có cấu trúc được dùng để phân lớp các đối tượng dựa vào dãy các luật. Khi cho dữ liệu về các đối tượng gồm các thuộc tính cùng với lớp (classes) của nó, cây quyết định sẽ sinh ra các luật để dự đoán lớp của các đối tượng chưa biết. Các quy tắc này được trích ra dựa trên bộ các đặc trưng của các dữ liệu huấn luyện. Trong cây quyết định, các lá đại diện cho các lớp (nhãn), mỗi nút con trong cây và các nhánh cây của nó biểu diễn sự kết hợp của các đặc trưng để dẫn dắt tới việc phân lớp. Như vậy, việc phân loại một đối tượng sẽ bắt đầu với việc kiểm tra giá trị của nút gốc, sau đó tiếp tục đi xuống dưới theo các nhánh cây tương ứng với các giá trị đó. Quá trình này được lặp đi lặp lại đối với từng nút, cho đến khi không thể đi tiếp được nữa và chạm đến nút lá. Mục đích của thuật toán là đạt được kết quả chính xác nhất với số lần lựa chọn ít nhất.

Dưới đây là một ví dụ về cây quyết định. Thông tin dữ liệu được cho như bảng dưới đây:

| **Thuộc tính** | | | | **Phân loại (Có chơi tennis)** |
|---|---|---|---|---|
| **Thời tiết** | **Nhiệt độ (Độ C)** | **Độ ẩm (%)** | **Có gió** | |
| **Mưa** | 32 | 85 | Không | Không |
| **Mưa** | 34 | 82 | Có | Không |
| **Âm u** | 31 | 90 | Không | Có |
| **Nắng** | 26 | 87 | Không | Có |
| **Nắng** | 23 | 67 | Không | Có |
| **Âm u** | 18 | 80 | Có | Không |
| **Mưa** | 24 | 88 | Có | Có |

| | | | | |
|---|---|---|---|---|
| **Mưa** | 28 | 84 | Không | Không |
| **Nắng** | 20 | 70 | Không | Có |

*Bảng II-1: Dữ liệu chơi tennis*




*Hình II-2: Ví dụ Decision Tree cho bài toán chơi tennis*

Dữ liệu đầu vào được chia thành 2 nhãn là có chơi tennis và không chơi tennis. Ở đây, cây quyết định có thể sinh ra các luật tương ứng với các nút và lá của cây như sau:

- Nếu trời âm u: Có chơi
- Nếu trời nắng:

o Nếu có gió: Có chơi

o Nếu không có gió: Không chơi

- Nếu trời mưa:

o Nếu độ ẩm lớn hơn 80%: Không chơi

o Nếu độ ẩm nhỏ hơn 80%: Có chơi

Thuật toán thường dùng trong cây quyết định là ID3 (Iterative Dichotomiser 3).

Thuật toán này dựa trên khái niệm Entropy và Information Gain (Độ lợi thông tin). Ở đây entropy chỉ mức độ không chắc chắn của dữ liệu. Thuật toán ID3 có thể được mô tả như sau: bắt đầu từ nút gốc, tại mỗi bước, thuộc tính tốt nhất sẽ được chọn ra sao cho độ lợi thông tin là lớn nhất. Với mỗi thuộc tính được chọn, ta chia dữ liệu vào các nút con tương ứng với các giá trị của thuộc tính đó rồi tiếp tục áp dụng phương pháp này cho mỗi nút con. Các bước thực hiện của thuật toán này như sau:

1. Giả sử tập dữ liệu $S$ có $N$ phần tử và $C$ lớp khác nhau, trong đó $N_c$ điểm thuộc lớp $c$. Entropy của nút này sẽ được tính theo công thức:

$$H(S) = -\sum_{c=1}^{C} \frac{N_c}{N} log(\frac{N_c}{N})$$

Giả sử thuộc tính được chọn là $x$, các điểm dữ liệu trong $S$ được phân thành $K$ nút con với số điểm trong mỗi nút con tương ứng là $m_1, m_2, \dots, m_K$. Ta có:

$$H(x, S) = \sum_{k=1}^{K} \frac{m_k}{N} H(S_k)$$

2. Độ lợi thông tin $G(x, S)$ sẽ được tính dựa vào sự chênh lệch giữa entropy ban đầu và entropy của mỗi nhánh.

$$G(x, S) = H(S) - H(x, S)$$

3. Thuộc tính có độ lợi thông tin lớn nhất sẽ được chọn làm nút quyết định.

4. Nếu một trong các nhánh của nút quyết định được chọn có entropy bằng 0, nó sẽ trở thành nút lá. Các nhánh khác sẽ tiếp tục được phân chia.

5. Thuật toán sẽ chạy đến khi không thể phân chia được nữa thì dừng.

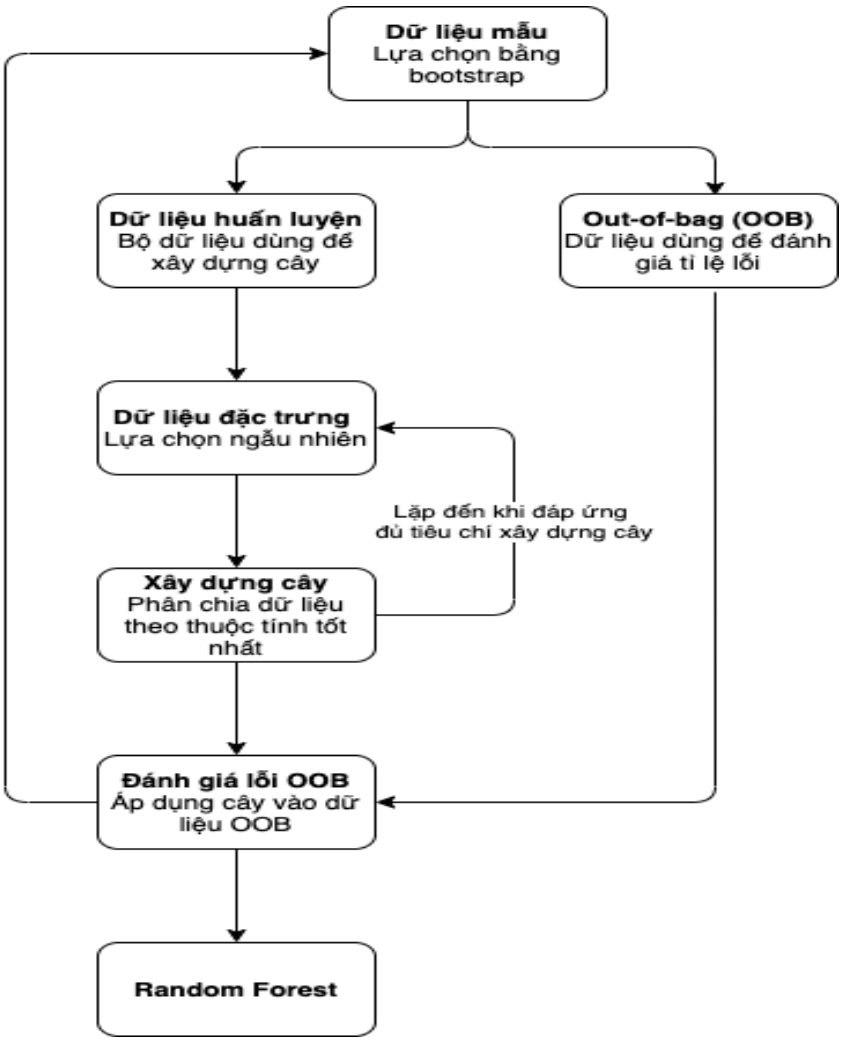
Cây quyết định là một thuật toán phổ biến bởi nó đơn giản và có thể xử lý tốt các tập dữ liệu lớn và có nhiều dữ liệu nhiễu. Một ưu điểm khác của cây quyết định là người ta có thể theo dõi quá trình lựa chọn một cách tường minh. Điều này khiến cho thuật toán này trở nên phổ biến cho các bài toán như chẩn đoán y tế, lọc thư rác, sàng lọc an ninh.

### *2.2.4 Random Forest*

Random Forest là một thành viên trong chuỗi thuật toán cây quyết định. Ý tưởng của

Random Forest là tạo ra một vài cây quyết định. Các cây quyết định này sẽ chạy và cho kết quả độc lập. Câu trả lời được dự đoán bởi nhiều cây quyết định nhất sẽ được Random Forest lựa chọn. Để đảm bảo các cây quyết định không giống nhau, Random Forest sẽ ngẫu nhiên chọn ra một tập con các đặc trưng ở mỗi nút (thay cho toàn bộ). Các tham số còn lại được sử dụng trong Random Forest giống như trong cây quyết định.

Cụ thể các bước của thuật toán được trình bày như sơ đồ dưới đây:




*Hình II-3: Sơ đồ thuật toán Random Forest*

1. Các cây được xây dựng dựa trên 2/3 dữ liệu của tập dữ liệu huấn luyện. Dữ liệu được lựa chọn ngẫu nhiên.

2. Một số biến dự đoán được chọn ngẫu nhiên từ tổng số các biến dự đoán. Sau đó, cách phân chia tốt nhất của các biến được lựa chọn sẽ được dùng để phân chia nút. Theo

mặc định, số lượng biến được chọn sẽ là căn bậc hai của tổng số các thuộc tính dùng để dự đoán và không đổi đối với các cây.

3. Tỷ lệ dự đoán sai được tính toán dựa vào phần dữ liệu còn lại (dữ liệu out-of-bag).
4. Mỗi cây huấn luyện sẽ đưa ra một kết quả phân loại, được gọi là "bỏ phiếu". Lớp nhận được nhiều "phiếu" nhất sẽ được chọn là kết quả cuối cùng.

Random Forest thừa kế rất nhiều ưu điểm của thuật toán cây quyết định. Random Forest có thể dùng cho cả hai bài toán phân loại và hồi quy, bởi nó đơn giản và dễ thích nghi, kết quả đưa ra cũng chính xác hơn. Tuy nhiên, không như cây quyết định, cấu trúc của Random Forest rất phức tạp nên không thể hiểu được cơ chế hoạt động bên trong của thuật toán. Ngoài ra, Random Forest cũng ổn định hơn so với cây quyết định. Đối với cây quyết định, chỉ cần dữ liệu bị sửa đổi một chút thì cả cây cũng sẽ bị thay đổi, làm giảm độ chính xác. Còn với thuật toán Random Forest, do nó được kết hợp từ rất nhiều cây quyết định nên nó sẽ ổn định hơn.

## 2.3 Một số phương pháp trích chọn đặc trưng phổ biến với bài toán phát hiện mã độc

### 2.3.1 Trích chọn đặc trưng dựa trên PE Header

Các file mã độc thường có định dạng PE, đây là một định dạng riêng của Win32. Để có thể thực thi trên máy tính, nội dung file PE được chia thành các thành phần và có mối liên hệ mật thiết với nhau. Nắm rõ cấu trúc PE sẽ giúp hiểu được cơ chế thực thi của một chương trình, từ việc tổ chức tới việc load lên bộ nhớ, các tài nguyên sử dụng, …. Hơn nữa, khi muốn sửa đổi một file, ví dụ như thêm vào một số đoạn mã, chỉnh sửa một số thành phần nhưng vẫn muốn chương trình thực thi bình thường. Do đó, cần phải nắm rõ cấu trúc PE file, mối liên hệ giữa các thành phần trong file để có thể nhanh chóng thay đổi file và thoả mãn yêu cầu đề ra. Trong đó PE Header là phần chứa nhiều thông tin quan trọng nhất của 1 file.

PE Header là cấu trúc IMAGE_NT_HEADERS bao gồm các thông tin cần thiết cho quá trình loader load file lên bộ nhớ. Cấu trúc này gồm 3 phần được định nghĩa trong windows.inc:

- **Signature**: là 1 DWORD bắt đầu PE Header chứa chữ ký PE: 50h, 45h, 00h, 00h.

- **FileHeader**: 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 file. Trong trường này chúng ta cần chú ý tới trường NumberOfSections, đây là trường chứa số section của file. Nếu muốn thêm/xoá section trong PE file, ta cần thay đổi tương ứng trường này.
- **OptionalHeader**: bao gồm 224 bytes tiếp theo sau FileHeader. Cấu trúc này được định nghĩa trong windows.inc, đây là phần chứa thông tin về sơ đồ logic trong PE file.

Như vậy việc trích xuất thông tin của phần PE Header có thể cho ta được cái nhìn tổng quan về định dạng, nhiệm vụ, logic thực thi của file. Đây cũng chính là các đặc trưng thường được trích xuất để phân loại một file bình thường hay file chứa mã độc. Tuy nhiên, các đặc trưng này có nhược điểm là kẻ tấn công có thể thay đổi thông tin một file mã độc sao cho giống với file bình thường.

### 2.3.2 Trích chọn đặc trưng dựa trên ảnh nhị phân

Các file dù là định dạng gì đều có thể được biểu diễn dưới dạng nhị phân. Vì vậy, có thể chuyển đổi dữ liệu nhị phân của một file thành ảnh nhị phân. Cụ thể hơn có thể biến đổi theo 2 cách khác nhau như sau:

- Dữ liệu nhị phân ban đầu được chuyển về dạng ma trận nhị phân có kích thước phù hợp với bài toán. Sau đó ma trận được đưa về ảnh nhị phân, tức là mỗi pixel trong ảnh tương ứng với 1 bit của dữ liệu (chỉ có 2 giá trị 0 và 1 đại diện tương ứng cho đen và trắng).
- Dữ liệu nhị phân được đọc dưới dạng từng byte (tương ứng với 8 bit – có giá trị từ 0 đến 255). Tiếp đó dữ liệu ở dạng byte cũng được đưa về dạng ma trận phù hợp với bài toán. Cuối cùng là được chuyển đổi thành ảnh nhị phân xám, mỗi pixel tương ứng với 1 byte (giá trị của pixel tương ứng từ 0 đến 255 đại diện cho độ xám của pixel).

Sau khi đã thu được ảnh nhị phân từ dữ liệu nhị phân, có thể sử dụng các phương pháp trích chọn đặc trưng cho ảnh để trích xuất đặc trưng cho bài toán phát hiện mã độc.

### 2.3.3 Trích chọn đặc trưng dựa trên mã nguồn

Có thể sử dụng các công cụ dịch ngược để lấy được nội dung mã nguồn của mã độc. Mã nguồn của một chương trình thực chất là một file văn bản, nội dung văn bản thể hiện logic của chương trình, trong đó khi muốn thực hiện logic nào đó, các hàm, các câu lệnh phải viết theo 1 trình tự nhất định. Vì lý do này, có thể sử dụng các phương pháp trình

chọn đặc trưng cho văn bản như n-gram, tf-idf để trích xuất đặc trưng từ mã nguồn của một chương trình.

## 2.4 Phương pháp phát hiện mã độc dựa trên phân tích mẫu

### 2.4.1 Cấu trúc đoạn mã Assembly

Các chương trình mã độc thường được xây dựng từ các đoạn mã Assembly có cấu trúc 4 trường riêng biệt cách nhau bởi dấu tab hoặc dấu cách như ví dụ dưới đây:

```
1 | Label Opcode  Operands    Comment
2 | Func  MOV     R0, #100    ; đặt R0 bằng 100
3 |               BX  LR      ; hàm return
```

*Hình II-4: Ví dụ cấu trúc đoạn mã Assembly*

- **Label:** nằm ở cột đầu tiên dùng để xác định vị trí trong bộ nhớ của tập lệnh hiện tại, bắt buộc phải chọn tên duy nhất cho mỗi label.
- **Opcode:** là mã máy chỉ cho bộ xử lý lệnh nào cần phải thực hiện.
- **Operand:** là toán hạng xác định vị trí của dữ liệu để thực hiện lệnh. Với tập lệnh Thumb thì có 0,1,2,3, hoặc 4 operand (toán hạng) cách nhau bằng dấu phẩy.
- **Comment:** là phần chú thích, nó thường được bỏ qua khi biên dịch code, nhưng nó sẽ mô tả giúp cho bạn có thể hiểu được cách phần mềm hoạt động.

### 2.4.2 Công cụ trích xuất và phân tích mã lệnh của mã độc

Theo như cấu trúc của đoạn mã assembly, có thể thấy thông tin về mã lệnh là thông tin quan trọng nhất. Với tập dữ liệu các mã lệnh có thể dự đoán được hành vi của một chương trình. Các chương trình mã độc thường sẽ có một số đoạn mã lệnh có thứ tự giống nhau vì thực hiện các hành vi mục đích tương tự nhau. Như vậy, việc đọc được đoạn mã assembly của một chương trình và trích xuất được dữ liệu mã lệnh của chương trình là bước đầu tiên và quan trọng nhất trong quá trình phát hiện mã độc.

Một chương trình bình thường trung bình có khoảng 1,5 triệu mã lệnh với khoảng 192 loại mã lệnh khác nhau. Trong số các mã lệnh này, có 72 loại mã lệnh mà chiếm đến hơn 99,8% tổng số mã lệnh có trong một chương trình, với 14 loại mã lệnh chiếm xấp xỉ 90% và nổi bật nhất trong số đó là 5 loại mã lệnh chiếm hơn 64% là mov, push, call, pop, cmp. Với các chương trình độc hại thường có khoảng 665,000 mã lệnh với tổng số 141

loại mã lệnh khác nhau được tìm thấy. Tương tự như với chương trình bình thường, 14 loại mã lệnh phổ biến nhất chiếm hơn 92% và 5 loại mã lệnh chính chiếm xấp xỉ 65% tổng số mã lệnh. Bảo dưới đây so sánh mật độ xuất hiện (%) của 14 loại mã lệnh phổ biến nhất trong chương trình bình thường và 7 loại mã độc khác nhau.

| Mã lệnh | Bình thường | Kernel RK | User RK | Tools | Bot | Trojan | Virus | Worms |
|---|---|---|---|---|---|---|---|---|
| **mov** | 25,3 | 37 | 29 | 25,4 | 34,6 | 30,5 | 16,1 | 22,2 |
| **push** | 19,5 | 15,6 | 16,6 | 19 | 14,1 | 15,4 | 22,7 | 20,7 |
| **call** | 8,7 | 5,5 | 8,9 | 8,2 | 11 | 10 | 9,1 | 8,7 |
| **pop** | 6,3 | 2,7 | 5,1 | 5,9 | 6,8 | 7,3 | 7 | 6,2 |
| **cmp** | 5,1 | 6,4 | 4,9 | 5,3 | 3,6 | 3,6 | 5,9 | 5 |
| **jz** | 4,3 | 3,3 | 3,9 | 4,3 | 3,3 | 3,5 | 4,4 | 4 |
| **lea** | 3,9 | 1,8 | 3,3 | 3,1 | 2,6 | 2,7 | 5,5 | 4,2 |
| **test** | 3,2 | 1,8 | 3,2 | 3,7 | 2,6 | 3,4 | 3,1 | 3 |
| **jmp** | 3 | 4,1 | 3,8 | 3,4 | 3 | 3,4 | 2,7 | 4,5 |
| **add** | 3 | 5,8 | 3,7 | 3,4 | 2,5 | 3 | 3,5 | 3 |
| **jnz** | 2,6 | 3,7 | 3,1 | 3,4 | 2,2 | 2,6 | 3,2 | 3,2 |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **retn** | 2,2 | 1,7 | 2,3 | 2,9 | 3 | 3,2 | 2 | 2,3 |
| **xor** | 1,9 | 1,1 | 2,3 | 2,1 | 3,2 | 2,7 | 2,1 | 2,3 |

*Bảng II-2: Thống kê phân bố các loại mã lệnh phổ biến nhất trong các loại chương trình*

Có nhiều công cụ giúp đọc được đoạn mã assembly của một chương trình như nasm, gdb, objdump, strace. Công cụ được sử dụng nhiều nhất là objdump. Objdump là công cụ giúp bạn xem các thông tin quan trọng trong các file object hay file executable.

Để thực hiện đọc đoạn mã assembly của một chương trình, với công cụ objdump, người dùng có thể sử dụng lệnh:

objdump –M <mã kiến trúc bộ vi xử lý chạy chương trình> -D <đường dẫn tới chương trình> > <đường dẫn tới nơi muốn lưu trữ file báo cáo>

Objdump hỗ trợ nhiều kiến trúc hệ bộ vi xử lý khác nhau như: x86-64, x86-64, i386, i8086, att, intel, att-mnemonic, intel-mnemonic, addr64, addr16, data32, data16, suffix, amd64, intel64.

Ví dụ, thực hiện biên dịch một chương trình “demo” chạy trên kiến trúc intel, người dùng thực hiện gõ lệnh: objdump –M intel –D demo  > demo.txt. Kết quả thu được sẽ được lưu trong file demo.txt với nội dung như sau:

*Hình II-5: Ví dụ mã Assembly được đọc bằng công cụ Objdump*

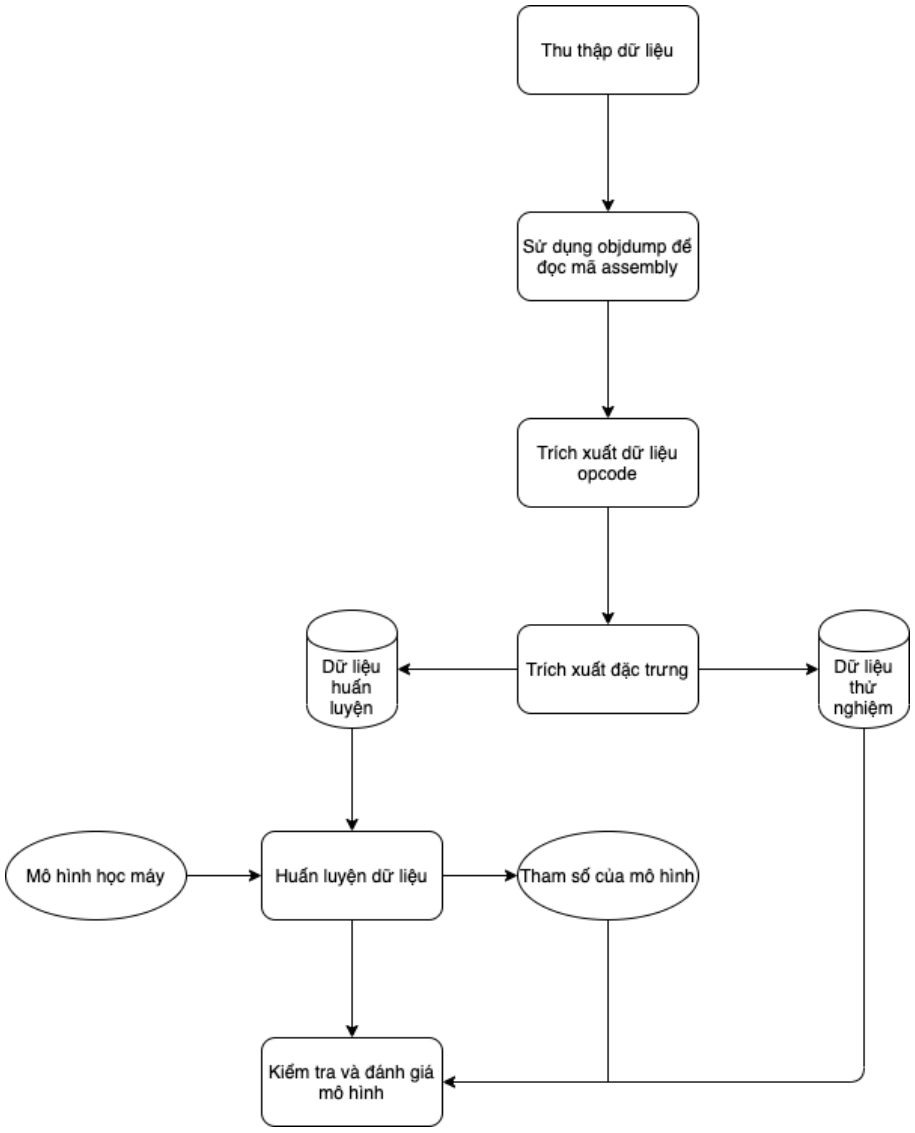
```
Disassembly of section .interp:

0000000000400238 <.interp>:
  400238:       2f                      (bad)
  400239:       6c                      ins    BYTE PTR es:[rdi],dx
  40023a:       69 62 36 34 2f 6c 64    imul   esp,DWORD PTR [rdx+0x36],0x646c2f34
  400241:       2d 6c 69 6e 75          sub    eax,0x756e696c
  400246:       78 2d                   js     400275 <_init-0x373>
  400248:       78 38                   js     400282 <_init-0x366>
  40024a:       36 2d 36 34 2e 73       ss sub eax,0x732e3436
  400250:       6f                      outs   dx,DWORD PTR ds:[rsi]
  400251:       2e 32 00                xor    al,BYTE PTR cs:[rax]

Disassembly of section .note.ABI-tag:

0000000000400254 <.note.ABI-tag>:
  400254:       04 00                   add    al,0x0
  400256:       00 00                   add    BYTE PTR [rax],al
  400258:       10 00                   adc    BYTE PTR [rax],al
  40025a:       00 00                   add    BYTE PTR [rax],al
  40025c:       01 00                   add    DWORD PTR [rax],eax
  40025e:       00 00                   add    BYTE PTR [rax],al
  400260:       47                      rex.RXB
  400261:       4e 55                   rex.WRX push rbp
  400263:       00 00                   add    BYTE PTR [rax],al
  400265:       00 00                   add    BYTE PTR [rax],al
  400267:       00 02                   add    BYTE PTR [rdx],al
  400269:       00 00                   add    BYTE PTR [rax],al
  40026b:       00 06                   add    BYTE PTR [rsi],al
  40026d:       00 00                   add    BYTE PTR [rax],al
  40026f:       00 20                   add    BYTE PTR [rax],ah
```

```
 400271:       00 00                    add    BYTE PTR [rax],al
       ...

Disassembly of section .note.gnu.build-id:

0000000000400274 <.note.gnu.build-id>:
 400274:       04 00                    add    al,0x0
 400276:       00 00                    add    BYTE PTR [rax],al
 400278:       14 00                    adc    al,0x0
 40027a:       00 00                    add    BYTE PTR [rax],al
 40027c:       03 00                    add    eax,DWORD PTR [rax]
 40027e:       00 00                    add    BYTE PTR [rax],al
 400280:       47              rex.RXB
 400281:       4e 55                    rex.WRX push rbp
 400283:       00 57 27                 add    BYTE PTR [rdi+0x27],dl
 400286:       42              rex.X
 400287:       43 b7 9f                 rex.XB mov r15b,0x9f
 40028a:       4d 3f                    rex.WRB (bad)
 40028c:       e7 81                    out    0x81,eax
 40028e:       87 84 84 9b 57 de a8     xchg   DWORD PTR [rsp+rax*4-0x5721a865],eax
 400295:       9a              (bad)
 400296:       77 ec                    ja     400284 <_init-0x364>
```

Sau khi đã thu được đoạn mã assembly của chương trình, hoàn toán có thể thực hiện quá trình trích xuất thông tin dữ liệu mã lệnh của chương trình để thực hiện phân tích và phát hiện xem chương trình có phải chương trình mã độc không.

### *2.4.3 Phương pháp phát hiện mã độc dựa trên phân tích mẫu*

Mô hình tổng quan của phương pháp phát hiện mã độc dựa trên học máy được thể hiện trên hình sau:




*Hình II-6: Sơ đồ phương pháp phát hiện mã độc dựa trên phân tích mẫu*

Bước đầu tiên của bất kỳ bài toán ứng dụng học máy cũng là thực hiện quá trình thu thập dữ liệu cho bài toán. Dữ liệu cần đảm bảo đủ lớn và có tính đa dạng về dữ liệu. Ngoài ra, để mô hình đạt kết quả tốt, phân bố của các lớp phải là xấp xỉ nhau. Với bài toán phát hiện mã độc, bước đầu cần phải thu thập các chương trình mã độc cùng với đó là các chương trình bình thường. Số lượng chương trình mã độc và chương trình bình thường cần phải là tương đương.

Dữ liệu sau khi đã thu thập sẽ được trộn vào nhau và được thực hiện quá trình dịch ngược để lấy được các đoạn mã assembly. Quá trình này sẽ sử dụng công cụ objdump để dịch ngược, tương ứng với mỗi chương trình trong tập dữ liệu cần phải dịch ra một file

mã assembly tương ứng. Các file mã assembly này sau đó sẽ được trích lấy thông tin mã lệnh.

Từ đây, với mỗi chương trình sẽ thu được tập hợp mã lệnh của chương trình. Dữ liệu mã lệnh này sau đó sẽ được chọn ra các mã lệnh phổ biến nhất để làm đặc trưng, với mỗi loại mã lệnh là một đặc trưng. Giá trị của mỗi đặc trưng là tần suất xuất hiện của mã lệnh trong toàn bộ mã assembly của chương trình. Để quá trình huấn luyện sau đó được thực hiện nhanh hơn và có độ chính xác cao, các đặc trưng của dữ liệu sau đó cần được chuẩn hoá.

Toàn bộ dữ liệu đã được trích chọn đặc trưng sau đó sẽ được kia thành 2 tập dữ liệu: tập dữ liệu huấn luyện và tập dữ liệu thử nghiệm. Dữ liệu được chia theo tỉ lệ 80%:20% tương ứng với dữ liệu huấn luyện và dữ liệu thử nghiệm. Tuy nhiên, trong mỗi tập dữ liệu vẫn phải đảm bảo phân bố của chương trình mã độc và chương trình bình thường là tương đương nhau.

Các mô hình học máy phổ biến sau đó được sử dụng để huấn luyện và tìm ra các ngưỡng tham số của mô hình. Chính các ngưỡng tham số này kết hợp với mô hình sẽ phân loại và phát hiện dữ liệu mới thuộc nhóm chương trình mã độc hay chương trình bình thường. Cuối cùng, để đánh giá tính hiệu quả của phương pháp này, dữ liệu thử nghiệm sẽ được sử dụng để đánh giá các mô hình học máy.

## 2.5 Kết luận chương

Chương II đã trình bày khái niệm về học máy và các thuật toán nổi bật trong học máy. Ngoài ra, chương đã giới thiệu kỹ thuật phân tích, trích xuất mã lệnh của một chương trình mã độc. Áp dụng kết hợp học máy, chương đã trình bày về các bước trong phương pháp phát hiện mã độc sử dụng phân tích mẫu. Trong chương tiếp theo, luận văn sẽ trình bày về quá trình thực nghiệm để đánh giá hiệu quả của phương pháp trên.

# CHƯƠNG III: THỬ NGHIỆM VÀ ĐÁNH GIÁ

*Chương III trình bày về quá trình thực nghiệm thông qua các bước thu thập dữ liệu, tiền xử lý trích chọn đặc trưng của dữ liệu, phân chia dữ liệu và cài đặt môi trường để tiến hành thực nghiệm. Tiếp đến, trình bày các kết quả thực nghiệm và đánh giá hiệu quả của mô hình phát hiện mã độc được trình bày trong chương 2.*

## 3.1 Thu thập dữ liệu và tiền xử lý dữ liệu

### *3.1.1 Thu thập dữ liệu*

Để thực nghiệm phương pháp phát hiện mã độc dựa trên phân tích mẫu, luận văn thu thập được 2736 file dữ liệu mẫu từ hai nguồn online là virustotal và virusshare. Trong 2736 file dữ liệu mẫu này có 1738 file mã độc và 998 file bình thường. Các file bình thường chủ yếu thuộc định dạng .exe, ngoài ra cũng có nhiều định dạng khác như: .doc, .xls, .pdf, .png, … nhằm làm tăng tính đa dạng cho dữ liệu và đây cũng là các định dạng file thường được sử dụng để phát tán mã độc. Các file mã độc thu thập được thuộc các dạng chính như: trojan, worm, virus, adware, spyware, … Các file mã độc sau đó sẽ được gán nhãn là "mã độc", còn các file bình thường sẽ được gán nhãn "bình thường".

Tuy nhiên, số lượng các mẫu thu thập còn rất ít so với thực tế, nhưng trong quá trình thu thập những mẫu này gặp rất nhiều khó khăn. Đa phần các mẫu mã độc gần như không ai chia sẻ hoặc có chia sẻ thì cũng rất hạn chế và việc tải từ các trang cũng bị giới hạn không phải muốn lấy bao nhiêu cũng được.

### *3.1.2 Tiền xử lý dữ liệu*

Sau khi thu thập được dữ liệu mẫu, luận văn sẽ sử dụng công cụ objdump để đọc từng tập dữ liệu mẫu. Nội dung mã assembly của từng chương trình sau đó sẽ được lưu thành một file .txt. Do đa phần các mẫu dữ liệu thu thập được chạy trên hệ điều hành Windows bao gồm cả 64bits và 32bits, các máy tính hiện nay sử dụng phổ biến là kiến trúc vi xử lý của intel, nên luận văn sử dụng objdump với tham số kiến trúc bộ vi xử lý là intel (tức là dịch ngược ra mã assembly sử dụng trên các máy tính sử dụng bộ vi xử lý của intel, không quan trọng phiên bản 64bits hay 32bits). Sau đó tiến hành đếm các hàm có trong file txt sinh ra từ objdump để làm đầu vào cho quá trình huấn luyện

Quá trình tiền xử lý này diễn ra nhanh hơn nhiều so với các cách phân tích mã độc khác do chỉ việc dịch ngược để tìm mã assembly của chương trình mà không phải xây dựng môi trường ảo và đợi các file chương trình chạy hết toàn bộ như các cách phân tích động.

### *3.1.3 Trích chọn đặc trưng dữ liệu*

Để mô hình đạt độ chính xác cao, quá trình trích chọn đặc trưng là vô cùng quan trọng. Dữ liệu mã assembly tương ứng của từng chương trình được xử lý để trích xuất ra duy nhất thông tin danh sách mã lệnh được sử dụng trong từng dòng lệnh. Sau khi có được danh sách toàn bộ mã lệnh của chương trình, luận văn tìm ra danh sách các loại mã lệnh đã xuất hiện trong chương trình và số lần xuất hiện của từng loại mã lệnh.

Có thể dễ thấy, doanh sách các loại mã lệnh và số lần xuất hiện của từng loại mã lệnh của mỗi chương trình là khác nhau, trừ khi 2 chương trình giống nhau hoàn toán. Điều này dẫn đến việc cần có một bộ mã lệnh chuẩn để thống nhất giữa tất cả các chương trình. Bộ mã lệnh chuẩn này cũng chính là bộ đặc trưng dữ liệu cho toàn bộ dữ liệu đã thu thập được. Giá trị của từng đặc trưng chính là số lần xuất hiện của từng mã lệnh trong mỗi chương trình.

Như trình bày ở trên, đa số các chương trình bất kể là mã độc hay bình thường sẽ có 14 loại mã lệnh là phổ biến nhất chiếm hơn 90% tổng số mã lệnh của một chương trình. Để tăng độ chính xác của của mô hình phát hiện, đồ án sử dụng thêm 15 loại mã lệnh khác có tần suất xuất hiện là nhiều nhất và khác 14 loại mã lệnh ở trên để làm danh sách đặc trưng chuẩn cho toàn bộ dữ liệu. Như vậy, luận văn thu được tổng cổng 29 đặc trưng cho mỗi chương trình. Danh sách 29 đặc trưng này là: mov, push, call, pop, cmp, jz, lea, test, jmp, add, jnz, retn, xor, and, bt, fdivp, fild, fstcw, imul, int, nop, pushf, rdtsc, sbb, setb, setle, shld, std và bad.

Tương ứng với mỗi chương trình, luận văn có được nhãn của chương trình là “mã độc” hay “bình thường”. Kết hợp với danh sách đặc trưng tương ứng của mỗi chương trình, luận văn thu được file dữ liệu với mỗi dòng tương ứng với một chương trình, 29 giá trị đầu tiên trên mỗi dòng là 29 đặc trưng của chương trình, giá trị cuối cùng là nhãn của chương trình. Dữ liệu sau đó được lưu ở một file csv cuối cùng có nội dung như hình

dưới đây:

| mov | push | call | pop | cmp | jz | lea | test | jmp | add | jnz | retn | xor | and | bt | fdivp | fild | fstcw | imul | int | nop | pushf | rdtsc | sbb | setb | setle | shld | std | (bad) | labels |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 3362 | 491 | 840 | 495 | 581 | 0 | 317 | 548 | 482 | 3477 | 0 | 0 | 388 | 990 | 1 | 0 | 1 | 0 | 308 | 85 | 340 | 17 | 0 | 150 | 1 | 0 | 0 | 534 | 938 | b |
| 74728 | 31431 | 14363 | 7866 | 9853 | 0 | 14338 | 7709 | 8331 | 45814 | 0 | 0 | 5219 | 3861 | 19 | 1 | 14 | 0 | 648 | 5453 | 3404 | 52 | 0 | 782 | 127 | 11 | 3 | 46 | 909 | m |
| 9782 | 12950 | 3422 | 6773 | 5324 | 0 | 3121 | 2125 | 1163 | 79531 | 0 | 0 | 6599 | 3344 | 20 | 0 | 28 | 0 | 1569 | 339 | 1580 | 67 | 0 | 931 | 22 | 1 | 0 | 1366 | 578 | m |
| 8224 | 1444 | 1757 | 1512 | 988 | 0 | 534 | 922 | 1203 | 8947 | 0 | 0 | 1623 | 1582 | 0 | 0 | 2 | 0 | 727 | 294 | 865 | 17 | 0 | 279 | 9 | 0 | 0 | 508 | 2116 | b |
| 8510 | 2047 | 1241 | 985 | 1720 | 0 | 1727 | 505 | 618 | 2186 | 0 | 0 | 1339 | 1183 | 8 | 0 | 1 | 0 | 382 | 349 | 290 | 1 | 0 | 24 | 30 | 1 | 2 | 178 | 57 | m |
| 5912 | 609 | 1340 | 690 | 699 | 0 | 319 | 780 | 711 | 4064 | 0 | 0 | 773 | 1833 | 1 | 0 | 3 | 0 | 367 | 252 | 571 | 5 | 0 | 172 | 2 | 0 | 0 | 771 | 994 | b |
| 6712 | 3356 | 1242 | 974 | 1682 | 0 | 1699 | 500 | 620 | 2794 | 0 | 0 | 1301 | 1221 | 8 | 1 | 1 | 0 | 327 | 287 | 281 | 1 | 0 | 23 | 30 | 0 | 2 | 189 | 53 | m |
| 8978 | 3992 | 2021 | 1468 | 1677 | 0 | 1058 | 808 | 943 | 3054 | 0 | 0 | 1171 | 607 | 7 | 0 | 1 | 0 | 161 | 325 | 402 | 1 | 0 | 28 | 17 | 1 | 2 | 182 | 986 | m |
| 8373 | 6713 | 2262 | 1984 | 2422 | 0 | 1424 | 866 | 976 | 3818 | 0 | 0 | 3448 | 2316 | 8 | 1 | 2 | 0 | 665 | 457 | 406 | 2 | 0 | 34 | 31 | 0 | 2 | 190 | 986 | m |
| 137 | 30 | 16 | 30 | 17 | 0 | 5 | 11 | 20 | 312 | 0 | 0 | 17 | 63 | 0 | 0 | 0 | 0 | 5 | 3 | 17 | 0 | 0 | 11 | 0 | 0 | 0 | 2 | 49 | b |
| 15472 | 10930 | 3485 | 1840 | 3730 | 0 | 2345 | 855 | 1011 | 6804 | 0 | 0 | 3836 | 4995 | 10 | 1 | 1 | 0 | 363 | 1614 | 407 | 1 | 0 | 31 | 22 | 0 | 2 | 191 | 986 | m |
| 5510 | 3214 | 1120 | 814 | 1321 | 0 | 975 | 454 | 562 | 2410 | 0 | 0 | 1247 | 924 | 6 | 1 | 1 | 0 | 268 | 405 | 288 | 2 | 0 | 29 | 26 | 0 | 2 | 187 | 982 | m |
| 2451 | 893 | 118 | 781 | 680 | 0 | 146 | 377 | 181 | 890 | 0 | 0 | 566 | 2230 | 1 | 5 | 32 | 0 | 218 | 166 | 91 | 77 | 0 | 523 | 2 | 0 | 1 | 87 | 2303 | b |
| 929 | 156 | 242 | 167 | 112 | 0 | 63 | 109 | 150 | 1234 | 0 | 0 | 150 | 92 | 0 | 0 | 1 | 0 | 36 | 483 | 107 | 1 | 0 | 43 | 0 | 0 | 0 | 39 | 284 | b |
| 130615 | 13798 | 14522 | 10065 | 17838 | 0 | 15328 | 11028 | 12788 | 52089 | 0 | 0 | 5328 | 4629 | 13 | 7 | 51 | 0 | 1223 | 257 | 23393 | 173 | 9 | 1972 | 57 | 11 | 15 | 47 | 1627 | m |
| 79324 | 34706 | 15885 | 7916 | 9904 | 0 | 16727 | 7790 | 8688 | 49169 | 0 | 0 | 5226 | 4040 | 22 | 1 | 27 | 0 | 629 | 5513 | 3513 | 63 | 0 | 763 | 139 | 11 | 3 | 49 | 953 | m |
| 56101 | 4839 | 5610 | 4091 | 7676 | 0 | 6246 | 6229 | 5594 | 25613 | 0 | 0 | 2345 | 3709 | 24 | 1 | 5 | 0 | 609 | 5725 | 3640 | 23 | 0 | 719 | 63 | 6 | 3 | 35 | 542 | m |
| 41 | 23 | 16 | 20 | 8 | 0 | 4 | 6 | 14 | 265 | 0 | 0 | 11 | 8 | 0 | 0 | 0 | 0 | 4 | 2 | 15 | 0 | 0 | 10 | 0 | 0 | 0 | 2 | 34 | b |
| 72 | 187 | 1 | 176 | 94 | 0 | 2 | 14 | 4 | 1019 | 0 | 0 | 71 | 52 | 0 | 0 | 2 | 0 | 116 | 6 | 1 | 3 | 0 | 85 | 0 | 0 | 0 | 2 | 7 | m |
| 74728 | 31431 | 14363 | 7866 | 9859 | 0 | 14338 | 7709 | 8331 | 45814 | 0 | 0 | 5270 | 3862 | 19 | 1 | 14 | 0 | 650 | 5453 | 3404 | 52 | 0 | 782 | 127 | 11 | 3 | 46 | 909 | m |
| 1056 | 5246 | 107 | 4376 | 1308 | 0 | 79 | 208 | 107 | 23822 | 0 | 0 | 1218 | 702 | 0 | 0 | 45 | 0 | 796 | 125 | 67 | 439 | 0 | 1953 | 0 | 0 | 0 | 31 | 197 | m |
| 34 | 17 | 6 | 20 | 8 | 0 | 3 | 5 | 14 | 252 | 0 | 0 | 12 | 8 | 0 | 0 | 0 | 0 | 5 | 2 | 16 | 0 | 0 | 8 | 0 | 0 | 0 | 1 | 34 | b |
| 6664 | 1757 | 1066 | 771 | 1075 | 0 | 634 | 473 | 576 | 2082 | 0 | 0 | 883 | 599 | 8 | 0 | 1 | 0 | 172 | 266 | 293 | 1 | 0 | 23 | 23 | 1 | 2 | 178 | 982 | m |
| 476 | 150 | 166 | 121 | 73 | 0 | 38 | 62 | 190 | 1258 | 0 | 0 | 104 | 83 | 3 | 0 | 1 | 0 | 123 | 39 | 30 | 0 | 0 | 61 | 0 | 0 | 0 | 153 | 239 | b |
| 50 | 26 | 14 | 20 | 11 | 0 | 6 | 5 | 24 | 324 | 0 | 0 | 14 | 8 | 0 | 0 | 0 | 0 | 6 | 2 | 15 | 1 | 0 | 7 | 0 | 0 | 0 | 3 | 43 | b |

*Hình III-1: File dữ liệu đặc trưng của bộ dữ liệu mã độc thu thập được*

## 3.2 Cài đặt và thử nghiệm

### *3.2.1 Cài đặt môi trường thực nghiệm*

#### a) Môi trường thực nghiệm

Để thực hiện quá trình thực nghiệm, luận văn sử dụng máy tính có cấu hình tiêu chuẩn như sau để thực hiện quá trình huấn luyện các giải thuật:

- Vi xử lý: Intel i5
- Dung lượng Ram: 4GB
- Hệ điều hành: Ubuntu Desktop 18.04 phiên bản x64 (64 bits)

Ngoài ra, luận văn sử dụng ngôn ngữ lập trình Python 3 và các thư viện Numpy, Pandas và scikit-learn để cài đặt và huấn luyện các môi. Trong đó:

- Numpy là một thư viện lõi phục vụ cho khoa học máy tính của Python, hỗ trợ cho việc tính toán các mảng nhiều chiều, có kích thước lớn với các hàm đã được tối ưu áp dụng lên các mảng nhiều chiều đó. Numpy đặc biệt hữu ích khi thực hiện các hàm liên quan tới Đại Số Tuyến Tính.
- Pandas là một thư viện mã nguồn mở, hỗ trợ đắc lực trong thao tác dữ liệu.

Đây cũng là bộ công cụ phân tích và xử lý dữ liệu mạnh mẽ của ngôn ngữ lập trình python. Thư viện này được sử dụng rộng rãi trong cả nghiên cứu lẫn phát triển các ứng dụng về khoa học dữ liệu. Thư viện này sử dụng một cấu trúc dữ liệu riêng là Dataframe. Pandas cung cấp rất nhiều chức năng xử lý và làm việc trên cấu trúc dữ liệu này.

- Scikit-learn là một thư viện mã nguồn mở trong ngành machine learning, rất mạnh mẽ và thông dụng với cộng đồng Python, được thiết kế trên nền NumPy và SciPy. Scikit-learn cung cấp gần như toàn bộ các thuật tuán về học máy, ngoài ra cũng cung cấp sẵn các công cụ về tiền xử lý dữ liệu và đánh giá độ chính xác của mô hình.

**b) Cài đặt thực nghiệm**

Với toàn bộ dữ liệu đã thu thập và trích xuất đặc trưng ở trên, luận văn chia dữ liệu thành 2 tập con là: tập dữ liệu huấn luyện và tập dữ liệu thử nghiệm cho mục đích huấn luyện và đánh giá mô hình theo tỉ lệ tương ứng là 80% và 20%. Cụ thể dữ liệu trong hai tập được phân bố như bảng sau:

| Dữ liệu | Mã độc | Bình thường | Tổng |
|---|---|---|---|
| **Huấn luyện** | 1382 | 806 | 2188 |
| **Thử nghiệm** | 356 | 192 | 548 |

*Bảng III-1: Phân bố 2 tập dữ liệu huấn luyện và thử nghiệm*

Để tăng tốc độ huấn luyện và độ chính xác của mô hình, luận văn sử dụng phương pháp Chính quy hóa (standardisation) như đã trình bày ở chương 2 để chuẩn hoá dữ liệu.

Luận văn sẽ sử dụng nhiều thuật toán khác nhau cho bài toán phát hiện mã độc như: Navie-Baye, SVM, Decision Tree, Random Forest. Với mỗi thuật toán, sẽ cho những kết quả khác nhau. Do đó, mô hình sẽ chạy thực nghiệm với toàn bộ các thuật toán, đánh giá và lựa chọn ra thuật toán có độ chính xác cao nhất phù hợp với bài toán phát hiện mã độc.

### *3.2.2 Phương pháp đánh giá*

Để đánh giá độ chính xác của từng thuật toán học máy, luận văn sử dụng các

phương pháp sau:

**a) Accuracy**

Accuracy là cách đánh giá đơn giản nhất và hay được sử dụng nhất. Phương pháp đánh giá này dựa trên công thức đơn giản là tỉ lệ số mẫu được dự đoán đúng so với tổng số mẫu có trong tập dữ liệu thử nghiệm. Công thức cụ thể như sau:

$$acc = \frac{total(correctly\ samples)}{total(samples)}$$

Phương pháp này không quan tâm đến độ chính xác của từng nhãn mà chỉ quan tâm số mẫu được dự đoán đúng nhãn. Các phương pháp tiếp theo sẽ đánh giá chi tiết hơn dựa trên kết quả dự đoán của từng nhãn.

**b) Precision và Recall**

Với một cách xác định một lớp là positive, Precision được định nghĩa là tỉ lệ số điểm true positive trong số những điểm được phân loại là positive (TP + FP). Recall được định nghĩa là tỉ lệ số điểm true positive trong số những điểm thực sự là positive (TP + FN). Một cách toán học, Precision và Recall là hai phân số có tử số bằng nhau nhưng mẫu số khác nhau:

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

$$Recall = \frac{TP}{TP + FN}$$

Trong đó:

- TP: số lượng các bản ghi gán nhãn "bình thường" được phân loại đúng.
- TN: số lượng các bản ghi gán nhãn "mã độc" được phân loại đúng.
- FP: số lượng các bản ghi gán nhãn "mã độc" bị phân loại sai thành "bình thường".
- FN: số lượng các bản ghi gán nhãn "bình thường" bị phân loại sai thành "mã độc".

Precision cao đồng nghĩa với việc độ chính xác của các điểm tìm được là cao. Recall cao đồng nghĩa với việc True Positive Rate cao, tức tỉ lệ bỏ sót các điểm thực sự positive là thấp.

**c) F1 score**

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. Công thức tính:

$$\frac{2}{F_1} = \frac{1}{Precision} + \frac{1}{Recall}$$

Hay:

$$F_1 = 2\frac{1}{\frac{1}{Precision} + \frac{1}{Recall}} = 2\frac{Recision * Recall}{Precison + Recall}$$

F1 – score có giá trị nằm trong nửa khoảng (0; 1]. F1 càng cao, bộ phân lớp càng tốt.

**d) Ma trận nhầm lẫn**

Ma trận nhầm lẫn là một ma trận thể hiện có bao nhiêu điểm dữ liệu thực sự thuộc vào một lớp, và được dự đoán là rơi vào một lớp. Ma trận có kích thước mỗi chiều bằng số lượng lớp dữ liệu. Giá trị tại hàng thứ i, cột thứ j là số lượng điểm lẽ ra thuộc vào class i nhưng lại được dự đoán là thuộc vào lớp j. Tổng các phần tử trong toàn ma trận này chính là số điểm trong tập kiểm thử. Các phần tử trên đường chéo của ma trận là số điểm được phân loại đúng của mỗi lớp dữ liệu.

Một ma trận nhầm lẫn gồm 4 chỉ số sau đối với mỗi lớp phân loại:

Để phù hợp, ta sẽ sử dụng lại bài toán phát hiện mã độc để giải thích 4 chỉ số này. Trong bài toán ta có 2 lớp: lớp mã độc được đoán Positive và lớp bình thường được đoán là Negative:

- **TP (True Positive)**: dự đoán đúng là file bình thường
- **TN (True Negative)**: dựa đoán đúng là file mã độc.
- **FP (False Positive)**: Dự đoán mã độc và trên thực tế là bình thường.
- **FN (False Negative)**: Dự đoán bình thường và thực tế mã độc.

## 3.3 Kết quả đánh giá

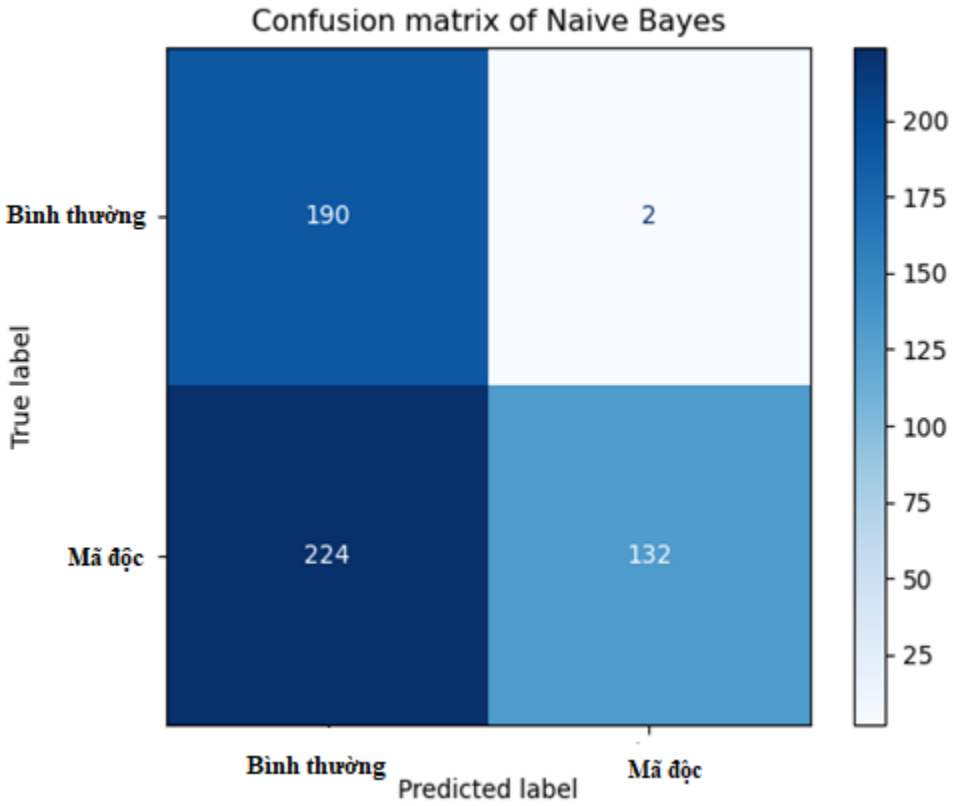
### *3.3.1 Naive Bayes*

Thuật toán Naive Bayes cho kết quả phân loại mã độc ở mức khá thấp, đạt

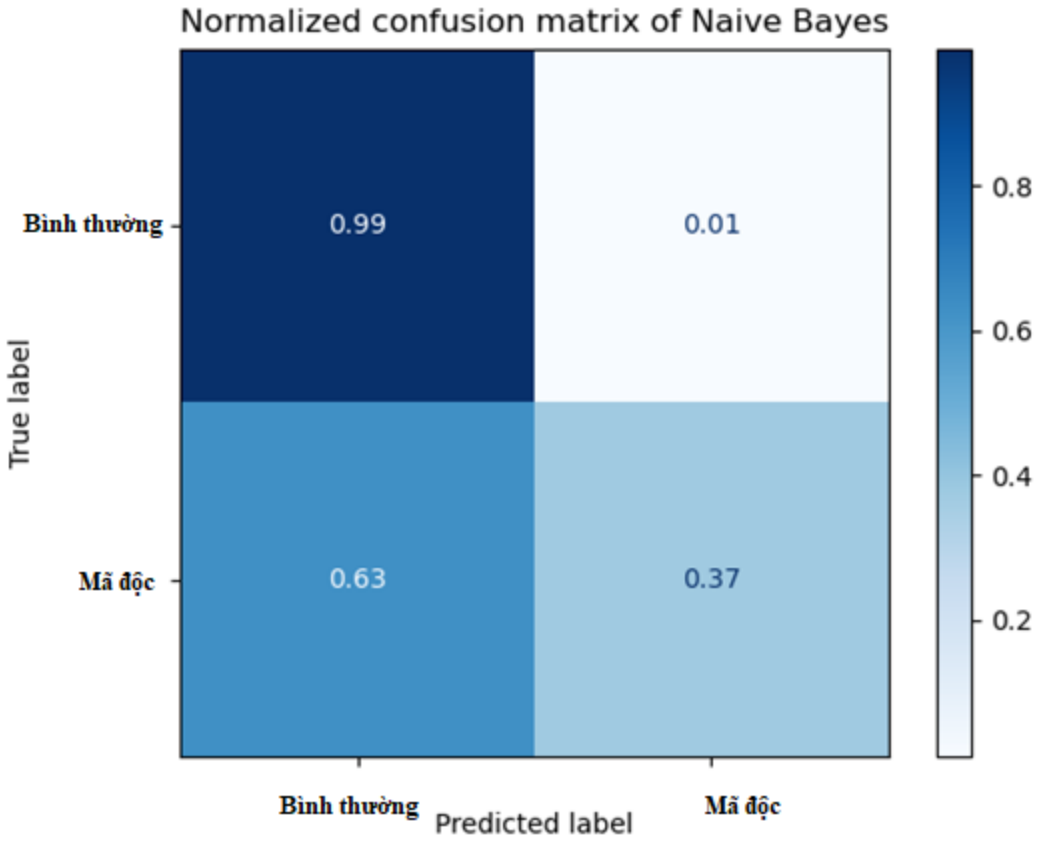
58,76%. Kết quả chi tiết như sau:

| Precision | Recall | F1 score | Accuracy |
|---|---|---|---|
| 98,96% | 45,89% | 62,71% | 58,76% |

*Bảng III-2: Kết quả thực nghiệm của thuật toán Naive Bayes*




*Hình III-2: Ma trận nhầm lẫn của thuật toán Naive Bayes*




*Hình III-3: Ma trận nhầm lẫn bình thường hóa của thuật toán Naive Bayes*

Nhìn vào thông tin ma trận nhầm lẫn, có thể thấy với nhãn dữ liệu bình thường, thuật toán cho kết quả nhận diện rất tốt khi nhận đúng đến 190 chương trình là chương trình bình thường. Tuy nhiên, với nhãn mã độc, thuật toán lại nhận nhầm tới hơn 60% các chương trình mã độc thành chương trình bình thường và chỉ có 132 chương trình là được nhận đúng.

Điều này có thể được lý giải bởi trong thuật toán Naive Bayes, kết quả được dự đoán từ xác suất xuất hiện độc lập của từng đặc trưng, điều này trong nhiều trường hợp là không đúng vì có nhiều đặc trưng, phân bố trong dữ liệu bình thường và dữ liệu mã độc là gần giống nhau.

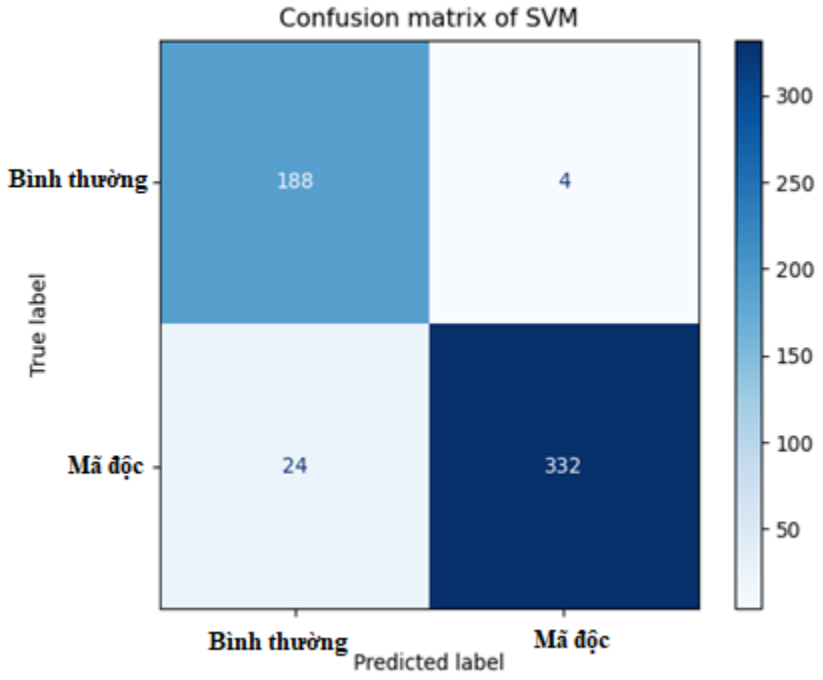
### *3.3.2 Support Vector Machine*

Đối với thuật toán SVM, kết quả thu được tuy không cao nhưng vẫn tương đối tốt với độ chính xác 94,89%.

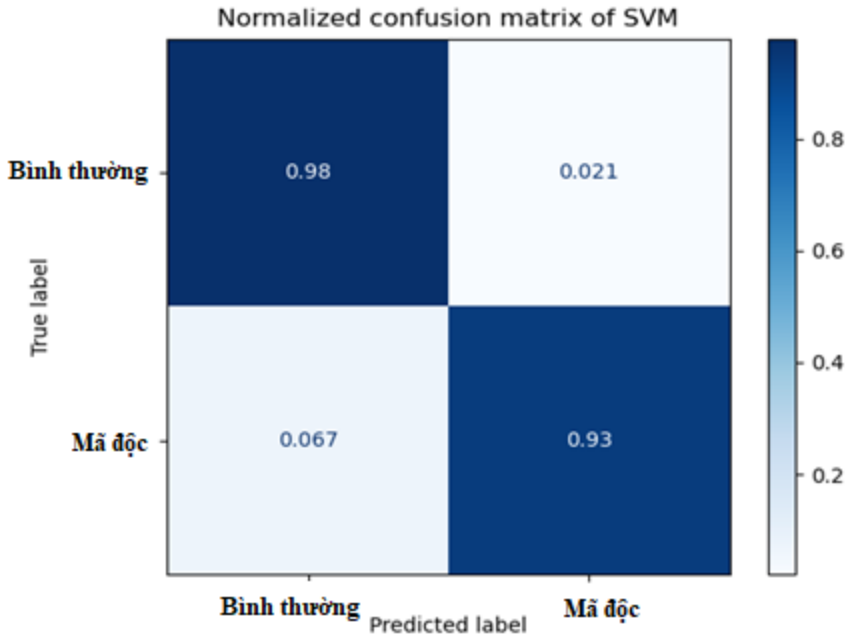
| Precision | Recall | F1 score | Accuracy |
|---|---|---|---|
| 97,92% | 88,68% | 93,07% | 94,89% |

*Bảng III-3: Kết quả thực nghiệm của thuật toán SVM*

Khi nhìn vào ma trận nhầm lẫn của thuật toán như hình dưới đây, có thể thấy thuật toán vẫn nhận nhầm khoảng 6% chương trình mã độc thành chương trình bình thường. Còn ở chiều ngược lại, thuật toán chỉ nhầm lẫn 4 chương trình bình thường thành chương trình mã độc.




*Hình III-6: Ma trận nhầm lẫn của thuật toán SVM*



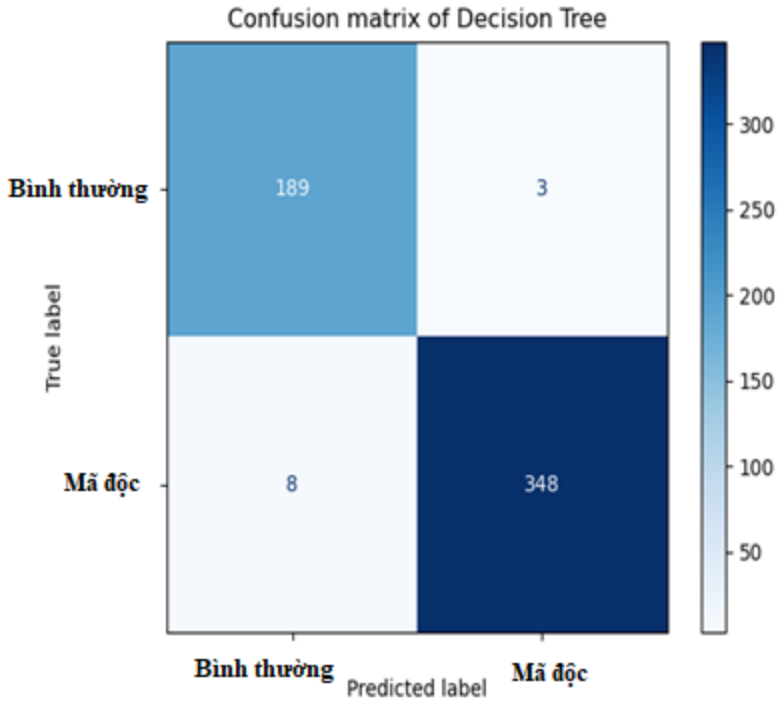

*Hình III-7: Ma trận nhầm lẫn bình thường hóa của thuật toán SVM*

### *3.3.3 Decision Tree*

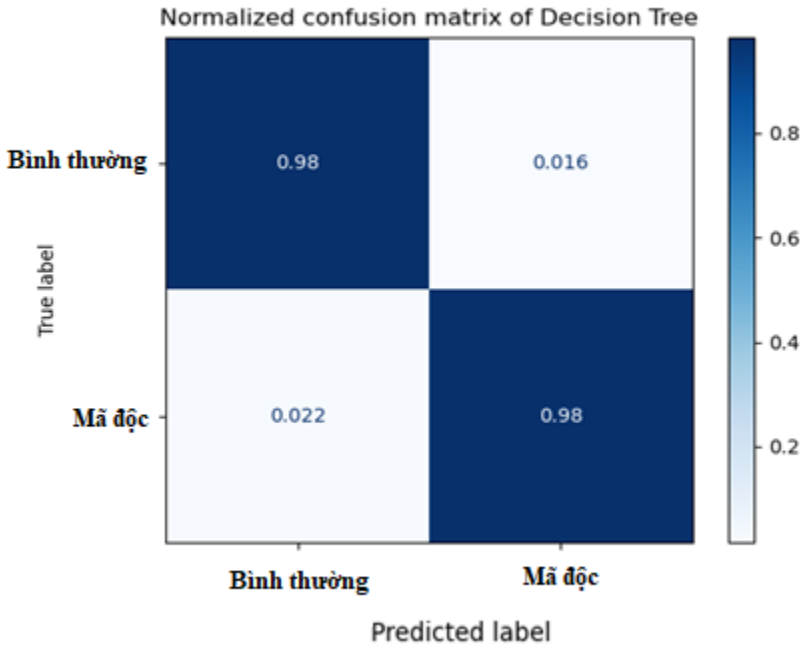
Kết quả thu được khi sử dụng thuật toán Decision Tree cũng đạt kết quả xấp xỉ 98%. Kết quả cụ thể như bảng dưới đây:

| Precision | Recall | F1 score | Accuracy |
|---|---|---|---|
| 98,44% | 95,94% | 97,17% | 97,99% |

*Bảng III-4: Kết quả thực nghiệm của thuật toán Decision Tree*




*Hình III-8: Ma trận nhầm lẫn của thuật toán Decision Tree*




*Hình III-9: Ma trận nhầm lẫn bình thường hóa của thuật toán Decision Tree*

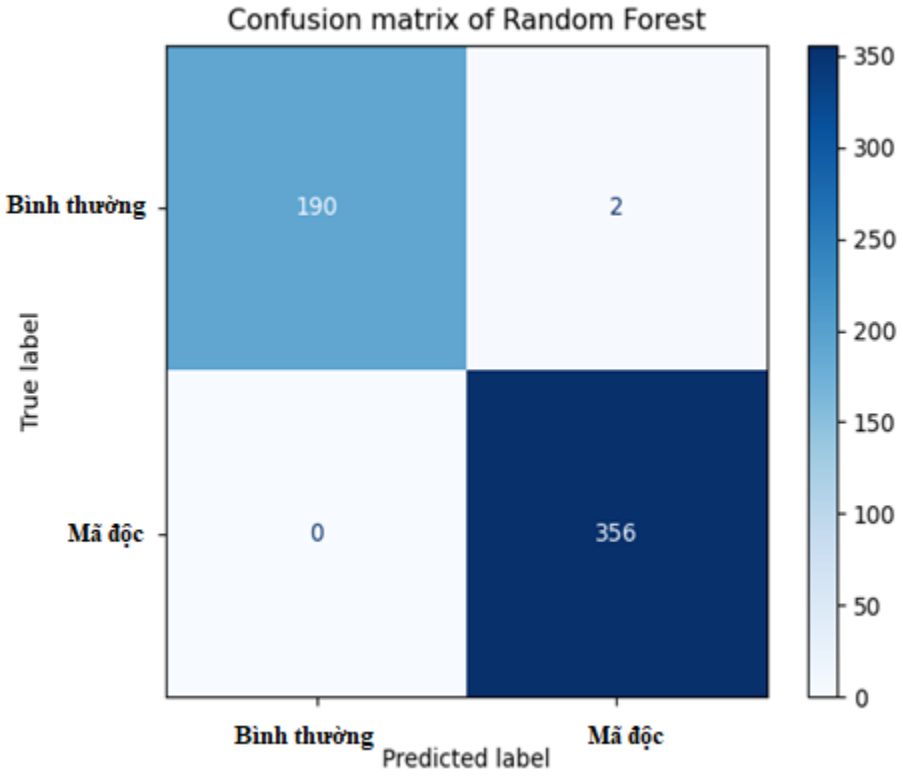
Cụ thể hơn, thuật toán nhận chính xác 189 chương trình bình thường và 348 chương trình mã độc. Số chương trình bị nhận nhầm chỉ chiếm khoảng 1-2% ở mỗi nhãn.

### *3.3.4 Random Forest*

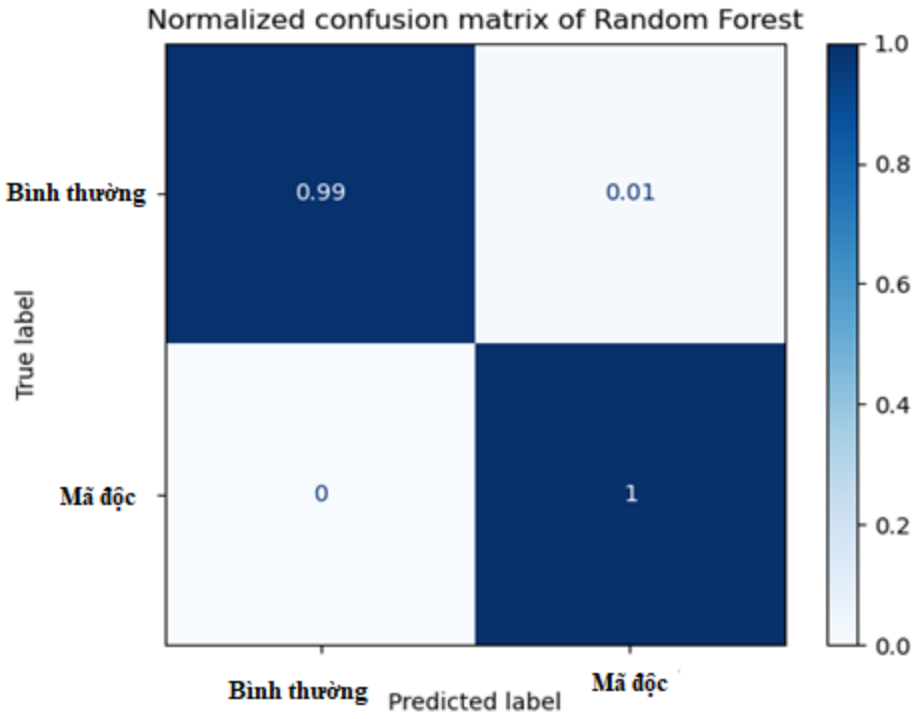
Thuật toán Random Forest cho kết quả thực nghiệm là cao nhất với độ chính xác lên đến 99,64% cao hơn cả thuật toán Decision Tree.

| Precision | Recall | F1 score | Accuracy |
|---|---|---|---|
| 98,96% | 100% | 99,48% | 99,64% |

*Bảng III-5: Kết quả thực nghiệm của thuật toán Random Forest*




*Hình III-10: Ma trận nhầm lẫn của thuật toán Random Forest*




*Hình III-11:Ma trận nhầm lẫn bình thường hóa của thuật toán Random Forest*

Có thể thấy, thuật toán này không nhận nhầm bất kì chương trình mã độc nào thành chương trình bình thường. Ngược lại, chỉ có 2 chương trình bình thường bị nhận nhầm thành chương trình mã độc. Điều này hoàn toàn dễ hiểu vì Random Forest có thể được coi là một thuật toán nâng cấp của thuật toán Decision Tree, trong khi thuật toán Decision Tree cho kết quả đã rất cao.

### 3.4 Nhận xét

| Phương pháp | F1 score | Accuracy |
|---|---|---|
| Naive Bayes | 62,71% | 58,76% |
| SVM | 93,07% | 94,89% |
| Decision Tree | 97,17% | 97,99% |
| Random Forest | 99,48% | 99,64% |

*Hình III-12: Tổng kết kết quả của các thuật toán*

Qua tất cả các thực nghiệm có thể đưa ra một số nhận xét như sau:

- Thuật toán Naive Bayes cho kết quả tệ nhất, có quá nửa chương trình độc

hại bị nhận nhầm thành chương trình bình thường. Điều này là hoàn toàn không thể chấp nhận được đối với một phương pháp phát hiện mã độc. Vì vậy không thể áp dụng phương pháp này vào bài toàn phát hiện mã độc.

- Thuật toán SVM cho kết quả khá tốt với độ chính xác xấp xỉ 95%. Tuy nhiên so với các phương pháp khác, độ chính xác này chưa đạt kỳ vọng phát hiện mã độc khi còn nhiều chương trình độc hại bị nhận nhầm thành chương trình bình thường. Thêm vào đó thuật toán SVM có thời gian chạy lâu hơn các phương pháp khác.
- Thuật toán Decision Tree cho độ chính xác vào khoảng 98%, số lượng chương trình mã độc bị nhận nhầm thành chương trình bình thường cũng rất ít chỉ khoảng 1-2%.
- Được coi như một thuật toán nâng cấp hơn Decision Tree, Random Forest đã thể hiện được ưu điểm của mình khi cho kết quả là tốt nhất với hơn 99%. Với thuật toán này, trong thực nghiệm không có một chương trình độc hại nào bị nhận nhầm thành chương trình bình thường. Chỉ có khoáng 1% chương trình bình thường bị nhận nhầm thành chương trình mã độc, nhưng đây là điều chấp nhận được. Đây cũng chính là thuật toán luận văn khuyến nghị xử dụng trong bài toán phát hiện mã độc.

### 3.5 Kết luận chương

Như vậy, chương III đã trình bày quá trình thử nghiệm, bao gồm các bước thực hiện, kết quả thực nghiệm, đánh giá và nhận xét về các kỹ thuật học máy sử dụng trong phát hiện mã độc. Luận văn cũng dừng lại ở việc thực hiện với các file thực thi. Trong 5 thuật toán đã đề cập ở trên, luận văn kiến nghị sử dụng Random Forest vào bài toán phát hiện mã độc thực tế, với độ chính xác 99.64%.

# KẾT LUẬN

**Kết quả đạt được:**

Luận văn đã nghiên cứu về phương pháp phát hiện mã độc dựa trên phân tích mẫu và đã đạt được một số kết quả sau:

Trình bày các kiến thức khái quát về mã độc như khái niệm, các dạng mã độc, lịch sử phát triển của mã độc. Ngoài ra, đồ án cũng giới thiệu một số kỹ thuật phát hiện mã độc dưa trên chữ kí và dựa trên hành vi.

Đưa ra khái niệm về học máy, các phương pháp học máy cũng như một số kỹ thuật học máy phổ biến hiện nay là Navie Bayes, Support Vector Machine, Decision Tree và Random Forest. Cũng với đó là giới thiệu một số phương pháp trích chọn đặc trưng cho bài toán phát hiện mã độc.

Trình bày kỹ thuật trích xuất và phân tích mã lệnh của mã độc. Từ đó xây dựng mô hình phát hiện mã độc dựa trên phân tích mẫu.

· Thu thập dữ liệu và xây dựng môi trường thực nghiệm, đánh giá kết quả phương pháp phát hiện mã đọc dựa trên phân tích mẫu.

**Hướng phát triển trong tương lai:**

Trên cơ sở các kiến thức tìm hiểu được và các kinh nghiệm cũng như kết quả trong quá trình thử nghiệm, luận văn có thể được cải thiện và nâng cao theo hướng:

· Dữ liệu hiện tại thu được cho quá trình thực nghiệm còn nhỏ. Luận văn sẽ thu thập thêm dữ liệu với kích thước lớn hơn, đa dạng về loại mã độc và các định dạng file chạy trên nhiều môi trường khác nhau.

· Chuyển từ mô hình phân loại 2 lớp cho bài toán phát hiện mã độc sang mô hình nhận nhiện nhiều nhãn mã độc khác nhau. Thử nghiệm một số mô hình học sâu để đạt kết quả cao hơn.

· Nghiên cứu phương pháp trích chọn đặc trưng mới dựa trên mã lệnh của mã độc.

# TÀI LIỆU THAM KHẢO


[1] Amit Sahu Prachi Parwar Deepak Agrawal An Analysis to Detect Malware using Machine Learning

[2] Aditya Mathur - A_survey_of_malware_detection_techniques

[3] M. Boldt and B. Carlsson. Analysing privacy-invasive software using computer forensic methods. http: // www. e-evidence. info/ b. html , January 2006.

[4] K. Wang and S. J. Stolfo. Anomalous payload-based network intrusion detection. In Proceedings of the 7th International Symposium on (RAID), pages 201–222, September 2004.

[5] https://www.researchgate.net/publication/49285561_Malware_Detection_Based_on_Structural_and_Behavioural_Features_of_API_Calls

[6] https://www.researchgate.net/figure/PE-Miner-programs-main-GUI_fig2_255787076

[7] https://www.researchgate.net/publication/224093090_Malware_Detection_Using_Perceptrons_and_Support_Vector_Machines

[8] https://www.researchgate.net/publication/224951943_Malware_Detection_Module_using_Machine_Learning_Algorithms_to_Assist_inCentralized_Security_in_Enterprise_Networks

[9] V. Shijoa, A. Salim, International Conference on Information and Communication Technologies (ICICT 2014)

[10] W. Li, K. Wang, S. Stolfo, and B. Herzog. Fileprints: Identifying file types by n-gram analysis. 6th IEEE Information Assurance Workshop, June 2005

[11] Alazab, Mamoun, Sitalakshmi Venkatraman, Paul Watters, and Moutaz Alazab. (2011). Zero-day Malware Detection based on Supervised Learning Algorithms of API call Signatures. Proceedings of the 9-th Australasian Data Mining Conference, 171-181

[12] Baldangombo Usukhbayar, Nyamjav Jambaljav, Shi-Jinn Horng. (2013). A Static Malware Detection System Using Data Mining Methods. Cornell University.

[13] Gavrilut, Dragos, Mihai Cimpoesu, Dan Anton, Liviu Ciortuz. (2009). Malware Detection Using Machine Learning. The International Multiconference on Computer Science and Information Technology.

[14] Priyank Singhal, Nataasha Raul. (2015). Malware Detection Module using Machine Learning Algorithms to Assist in Centralized Security in Enterprise Networks.

[15] Badr Hssina, Abdelkarim MERBOUHA, Hanane Ezzikouri, Mohammed Erritali , A comparative study of decision tree ID3 and C4.5., 2014, (IJACSA) International Journal of Advanced Computer Science and Applications, Special Issue on Advances in Vehicular Ad Hoc Networking and Applications