ĐẠI HỌC THÁI NGUYÊN
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

BÁO CÁO TỔNG KẾT

ĐỀ TÀI KHOA HỌC VÀ CÔNG NGHỆ CẤP ĐẠI HỌC

XÂY DỰNG PHẦN MỀM DIỆT VIRUS ICTUAV

Mã số: ĐH2016-TN07-01

Chủ nhiệm đề tài: ThS. TRỊNH MINH ĐỨC

THÁI NGUYÊN, 05/2019

ĐẠI HỌC THÁI NGUYÊN
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

BÁO CÁO TỔNG KẾT
ĐỀ TÀI KHOA HỌC VÀ CÔNG NGHỆ CẤP ĐẠI HỌC

XÂY DỰNG PHẦN MỀM DIỆT VIRUS ICTUAV

Mã số: ĐH2016-TN07-01

Xác nhận của cơ quan chủ trì **Chủ nhiệm đề tài**

THÁI NGUYÊN, 05/2019

ĐẠI HỌC THÁI NGUYÊN
TRƯỜNG ĐẠI HỌC CNTT&TT

DANH SÁCH THÀNH VIÊN THỰC HIỆN ĐỀ TÀI

TT	Họ và tên	Đơn vị công tác và lĩnh vực chuyên môn	Ghi chú
1	ThS. Trịnh Minh Đức	Đơn vị công tác: Trường Đại học Công nghệ thông tin và Truyền thông Chuyên môn: CNTT	
2	ThS. Lê Khánh Dương	Đơn vị công tác: Trường Đại học Công nghệ thông tin và Truyền thông Chuyên môn: CNTT	
3	ThS. Nguyễn Tuấn Hiệp	Đơn vị công tác: Trường Đại học Công nghệ thông tin và Truyền thông Chuyên môn: CNTT	
4	ThS. Vũ Việt Dũng	Đơn vị công tác: Trường Đại học Công nghệ thông tin và Truyền thông Chuyên môn: CNTT	
5	ThS. Võ Văn Trường	Đơn vị công tác: Trường Đại học Công nghệ thông tin và Truyền thông Chuyên môn: CNTT	

ĐƠN VỊ PHỐI HỢP CHÍNH

Tên đơn vị trong và ngoài nước	Nội dung phối hợp nghiên cứu	Họ và tên người đại diện đơn vị
Bộ môn An toàn hệ thống thông tin – Trường Đại học CNTT&TT	- Hỗ trợ cơ sở vật chất, thiết bị, phòng nghiên cứu. - Phối hợp nghiên cứu.	ThS. Lê Khánh Dương

MỤC LỤC

DANH MỤC TỪ VIẾT TẮT

Chữ viết tắt	Tiếng Anh	Tiếng Việt
API	Application Programming Interface	Giao diện lập trình ứng dụng
CRC	Cyclic Redundancy Check	Kiểm dư chu trình
ELF	Distributed Denial of Service	Từ chối dịch vụ phân tán
KNN	K-Nearest Neighbors	K láng giềng gần nhất
LSASS	Local Security Authority Subsystem Service	Dịch vụ hệ thống con cấp quyền
MD 5	Message-Digest algorithm 5	Hàm băm MD5
PE	Portable Executable	Tệp thực thi di động
RAT	Remote Administration Tool	Công cụ quản trị từ xa
TF	Term Frequency	Tần số từ

DANH MỤC CÁC HÌNH VẼ

DANH MỤC CÁC BẢNG BIỂU

ĐẠI HỌC THÁI NGUYÊN

TRƯỜNG ĐẠI HỌC CNTT&TT

THÔNG TIN KẾT QUẢ NGHIÊN CỨU

1. Thông tin chung

- Tên đề tài: XÂY DỰNG PHẦN MỀM DIỆT VIRUS ICTUAV
- Mã số: ĐH2016-TN07-01
- Chủ nhiệm: ThS. Trịnh Minh Đức
- Cơ quan chủ trì: Trường Đại học Công nghệ thông tin và Truyền thông
- Thời gian thực hiện: 24 tháng

2. Mục tiêu

Đưa ra đánh giá về các loại mã độc, tìm hiểu các phương pháp phân tích, trích rút đặc trưng mã độc từ đó xây dựng một phần mềm có khả năng phát hiện và loại bỏ mã độc

3. Kết quả nghiên cứu

- Đề tài đã thực hiện tìm hiểu về lập trình Win 32 API, Named Pipes.
- Nghiên cứu các vấn đề liên quan đến các loại mã độc, cách thức lây lan phá hoại của mã độc, các phương pháp phát hiện và loại bỏ mã độc đồng thời nghiên cứu về các phương pháp trích rút đặc trưng mã độc qua đó đề xuất được một giải pháp trích trọn đặc trưng cho bài toán phát hiện mã độc.
- Bên cạnh đó đề tài cũng đưa ra các kỹ thuật để xây dựng cơ sở dữ liệu mẫu mã độc.

4. Sản phẩm

- **Bài báo đăng tạp chí, kỷ yếu trong nước: 03**

1. Võ Văn Trường, Trịnh Minh Đức, Nguyễn Văn Vinh, Lê Khánh Dương (2016), "Đề xuất giải pháp trích chọn đặc trưng cho các thuật toán phân lớp dữ liệu trong kỹ thuật học máy giám sát và ứng dụng hiệu quả vào bài toán phát hiện mã độc", *Tạp chí Công nghệ thông tin và truyền thông – Bộ TT&TT*, 529 (719), tr. 38-46.

2. Võ Văn Trường, Trịnh Minh Đức, Nguyễn Văn Vinh, Lê Khánh Dương (2016), "Đề xuất giải pháp trích chọn đặc trưng cho các thuật toán phân lớp dữ liệu trong kỹ thuật học máy giám sát và ứng dụng hiệu quả vào bài toán phát hiện mã độc", *Kỷ*

yếu hội thảo Một số vấn đề chọn lọc về an toàn an ninh thông tin lần thứ 1, Học viện kỹ thuật mật mã.

3. Trịnh Minh Đức, Đinh Khánh Linh, Lê Khánh Dương, Võ Văn Trường (2019), "Một số kỹ thuật tạo cơ sở dữ liệu mẫu mã độc", *Tạp chí khoa học và công nghệ - Đại học Thái Nguyên* (đã được chấp nhận đăng).

- **Đề tài sinh viên NCKH: 01**

Nguyễn Hoàng Thắng. "Nghiên cứu và xây dựng hệ thống quét mã độc trực tuyến". Đề tài khoa học cấp sinh viên năm 2018. Kết quả nghiệm thu đạt loại Tốt theo QĐ số 31/QĐ-DH CNTT&TT, ngày 07/01/2019 của trường đại học CNTT&TT về việc công nhận kết quả thực hiện các đề tài KH&CN cấp cơ sở và sinh viên năm 2018.

- **Phần mềm diệt virus ICTUAV**

5. Hiệu quả

- Giáo dục, đào tạo: Sản phẩm của đề tài sẽ là tài liệu tham khảo cho sinh viên học các môn học liên quan đến chuyên ngành an toàn thông tin có thể hiểu được cơ chế hoạt động và các phương pháp phát hiện mã độc.
- Kinh tế, xã hội: Phần mềm có khả năng phát hiện và loại bỏ các loại mã độc, giúp bảo vệ người dùng tránh được các nguy cơ bị đánh cắp mật khẩu, tài khoản ngân hàng, thông tin cá nhân … tránh được các thiệt hại về kinh tế.

6. Khả năng áp dụng và phương thức chuyển giao kết quả nghiên cứu

Phần mềm đã được sử dụng tại công ty Lumi Việt Nam và đang tiếp tục được nâng cấp để có thể chuyển giao đến các đơn vị khác.

Ngày tháng năm 2019

Cơ quan chủ trì	**Chủ nhiệm đề tài**

INFORMATION ON RESEARCH RESULTS

1. General information

- Project title: BUILDING THE ICTUAV ANTI-VIRUS SOFTWARE.
- Code number: ĐH2016-TN07-01.
- Coordinator: Msc. Trinh Minh Duc.
- Implementing institution: Thai Nguyen University of Information Technology and Communication.
- Duration: 24 months.

2. Objective(s)

Giving evaluations on types of malicious softwares, studying analysis methods, feature extractions of malwares from there building a software which is capable of detecting malwares.

3. Research results

- The subject studied about Win 32 API, Named Pipes
- Types of malwares, spreading and detructive actions of malwares
- Studying about methods for detecting and removing malwares
- Feature extraction methods
- Proposing a feature extraction algorithm for malware detection

4. Products

- **Published articles in the country's magazines: 03**.

1. Vo Van Truong, Trinh Minh Duc, Nguyen Van Vinh, Le Khanh Duong (2016), "Proposing a feature extraction method for malware detection problem", *Journal of Information & Communications Technology*, 529 (719), pp. 38-46.
2. Vo Van Truong, Trinh Minh Duc, Nguyen Van Vinh, Le Khanh Duong (2016), "Proposing a feature extraction method for malware detection problem", *In proceedings of SoIS 2016,* ACT.
3. Trinh Minh Duc, Dinh Khanh Linh, Le Khanh Duong, Vo Van Truong (2019), "A number of techniques to create malware database", *TNU Journal of Science and Technology* (accepted to publish).

- **Student research topic: 01.**

Nguyen Hoang Thang. "Studying and building an online malware scanning system". Student research topic 2018. Achieving the good result.

- **The anti-virus software ICTUAV**

5. Effects

- Education: Products of the project will be reference material sources for students who study subjects related to Information security
- Economy and society: The software is capable of detecting and removing malwares, which protects users from avoiding risks such as loss of passwords, bank accounts…

6. The ability to applying and methods of transferring the results of research

The software is used by Lumi Company and being updated more functions

MỞ ĐẦU

1. Tổng quan tình hình nghiên cứu thuộc lĩnh vực của đề tài ở trong và ngoài nước

Ngoài nước: Những năm gần đây đã chứng kiến sự phát triển nhanh chóng của các phần mềm độc hại cả về số lượng và chủng loại. Trong năm 1992 số lượng mã độc đã tăng từ 1000 lên 2300, năm 2002 có đến 60000 loại mã độc và các biến thể của chúng được phát hiện. Ngày này số lượng này đã tăng lên trên hơn 847 triệu mẫu mã độc tính đến cuối năm 2018. Trước sự phát triển nhanh chóng và sự tàn phá nặng nề của các phần mềm độc hại, trên thế giới đã có rất nhiều nghiên cứu về các phương pháp phát hiện và loại bỏ các phần mềm độc hại, nhưng nhìn chung các nghiên cứu này đều xoay quanh hai phương pháp chính phát hiện dựa trên sự bất thường hay dị thường và dựa trên dấu hiệu.

Trong [33] Schultz và nhóm tác giả đã lần đầu tiên giới thiệu khái niệm khai phá dữ liệu cho việc phát hiện mã độc. Họ đã sử dụng ba đặc trưng tĩnh khác nhau cho việc phân loại mã độc: Portable executable (PE), strings và các chuỗi byte.

Natajai [26] đã đề xuất một phương pháp trực quan hóa và phân loại mã độc sử dụng các kỹ thuật xử lý ảnh, phương pháp này hình dung các tập tin mã độc như là những ảnh xám. Sử dụng ký thuật K láng giềng gần nhất với khoảng cách Euclide cho việc phân loại mã độc. Trong [27] các tác giả đã so sánh sự phân tích dựa trên kết cấu nhị phân (dựa trên kỹ thuật xử lý ảnh) với phân tích động. Họ đã phát hiện ra rằng sự phân loại sử dụng phương pháp này nhanh hơn, khả năng mở rộng hơn và có thể so sánh với phân tích động về sự chính xác.

Kong và nhóm tác giả [23] đã trình bày một framework cho việc phân loại mã độc tự động dựa trên thông tin cấu trúc của mã độc.

Vấn đề quan trọng nhất để phát hiện đúng mã độc là cần phải có phương pháp trích rút đặc trưng thực sự hiệu quả, trong [38] Tony Abou-Assaleh đã đề xuất một phương pháp phát hiện mã độc mới sử dụng dấu hiệu đặc trưng byte n-grams, các thí nghiệm của Abou-Assaleh đã đạt được tỉ lệ chính xác 98%. Trong [32] Santos và nhóm tác giả đã sử dụng đặc trưng opcode n-gram cho phương pháp phát hiện của họ.

2. Tính cấp thiết của đề tài

Ngày nay mạng Internet đã trở nên phổ biến trên toàn thế giới, Internet đã lan tỏa vào mọi ngóc ngách của đời sống xã hội, bất kỳ ai cũng có thể truy cập Internet

để tìm kiếm thông tin phục vụ cho nhu cầu của họ. Cùng với sự phát triển nhanh chóng của Internet là những mối nguy hiểm, rủi ro tiềm tàng khi sử dụng như đánh cắp mật khẩu, chiếm đoạt tài khoản ngân hàng, đánh cắp thông tin cá nhân, tống tiền…Những rủi ro này phần lớn bắt nguồn từ những phần mềm độc hại, các loại mã độc, virut máy tính… Chính vì thế, vấn đề nghiên cứu và xây dưng một phần mềm phát hiện và loại bỏ mã độc máy tính là một việc vô cùng cấp thiết.

Theo Báo cáo tổng kết của Kaspersky năm 2014 (Kaspersky Security Bulletin 2014), có 1.4 triệu vụ tấn công người dùng bằng mã độc trên Android năm 2014, tăng gấp 4 lần so với năm 2013. Trong đó, Việt Nam đứng thứ 6 trên toàn thế giới về số người dùng thiết bị di động bị mã độc tấn công. Nguy cơ mất an toàn thông tin đang ở mức đáng báo động khi Việt Nam có gần 50% số người dùng có nguy cơ nhiễm mã độc khi sử dụng Internet trên máy tính, xếp hạng 4 trên toàn thế giới; và đứng đầu thế giới với gần 70% người dùng máy tính dễ bị nhiễm mã độc, phần mềm độc hại cục bộ (qua USB, thẻ nhớ,…).

Ngoài ra, Microsoft ước tính rằng có khoảng 80% máy tính tại Việt Nam nhiễm các loại mã độc và phần mềm độc hại. Theo báo cáo gần đây của Hiệp hội An toàn Thông tin Việt Nam (VNISA), phần lớn các cơ quan tổ chức tại Việt Nam cho phép dùng thiết bị cá nhân (di động và máy tính bảng) truy cập vào mạng lưới tại nơi làm việc nhưng có tới 74% trong số thiết bị không hề sử dụng bất kỳ biện pháp bảo mật thông tin nào. Những thông số này đã dấy lên một mối lo ngại rất lớn và cũng đặt ra một áp lực không hề nhỏ cho các lãnh đạo, chuyên gia về công nghệ thông tin tìm ra giải pháp để đối phó với tình trạng mất an toàn thông tin trong môi trường hiện nay.

Theo báo cáo gần đây của Hiệp hội An toàn Thông tin Việt Nam phần lớn các cơ quan tổ chức tại Việt Nam cho phép dùng thiết bị cá nhân (di động và máy tính bảng) truy cập vào mạng lưới tại nơi làm việc nhưng có tới 74% trong số thiết bị không hề sử dụng bất kỳ biện pháp bảo mật thông tin nào. Từ thực tế này, tôi đề xuất đề tài: “Xây dựng phần mềm diệt virus ICTUAV”. Đây sẽ là một đề tài có ý nghĩa thực tiễn và ứng dụng rất cao, với phần mềm này chúng tôi hy vọng sẽ giúp bảo vệ người dùng trước những nguy hiểm, rủi ro khi truy cập máy tính và Internet.

3. Mục tiêu của đề tài

Đưa ra đánh giá về các loại mã độc, virus máy tính, tìm hiểu các phương pháp phân tích, trích rút đặc trưng mã độc từ đó xây dựng một phần mềm có khả năng phát hiện và loại bỏ mã độc.

4. Cách tiếp cận và phương pháp nghiên cứu

4.1. Cách tiếp cận

- Nghiên cứu trong tài liệu (từ các bài báo, tạp chí...)
- Tham gia các diễn đàn, hội thảo, xê mi na về an toan thông tin trong nước.
- Thử nghiệm trên một hệ thống thực tế.

4.2. Phương pháp nghiên cứu

- Về mặt lý thuyết: Nghiên cứu tổng quan về các loại mã độc, cách thức lây lan phá hoại của chúng. Nghiên cứu các phương pháp phát hiện mã độc phổ biến hiện nay.
- Về mặt thực nghiệm: dựa trên cơ sở lý thuyết đề xuất phương pháp trích rút đặc trưng cho bài toán phát hiện mã độc, đồng thời tiến hành xây dựng một phần mềm có khả năng phát hiện mã độc.

5. Đối tượng và phạm vi nghiên cứu

5.1. Đối tượng nghiên cứu

- Nghiên cứu các phương pháp phân tích mã độc
- Nghiên cứu các phương pháp phát hiện và loại bỏ mã độc
- Nghiên cứu các phương pháp trích rút đặc trưng mã độc
- Nghiên cứu các phương pháp tạo cơ sở dữ liệu mẫu mã độc
- Nghiên cứu phần mềm mã nguồn mở ClamAV

5.2. Phạm vi nghiên cứu

- Các vấn đề liên quan đến phân tích mã độc
- Các vấn đề liên quan đến mã độc
- Các vấn đề về trích rút đặc trưng mã độc
- Các tiêu chí đánh giá hiệu quả, chất lượng của các phương pháp phát hiện và trích rút đặc trưng

6. Nội dung nghiên cứu

- Tìm hiểu lập trình Win32 API
- Tìm hiểu Named Pipes
- Tìm hiểu về các loại mã độc, cách thức lây lan phá hoại của chúng.
- Tìm hiểu các phương pháp phát hiện và loại bỏ mã độc.
- Tìm hiểu các phương pháp trích rút đặc trưng mã độc
- Xây dựng CSDL mã độc

- Nghiên cứu và tìm hiểu về Clamav, một phần mềm tự do nguồn mở cho việc phát hiện và loại bỏ các loại mã độc.
- Xây dựng phần mềm phát hiện và loại bỏ mã độc dựa trên Clamav

CHƯƠNG 1: TỔNG QUAN VỀ MÃ ĐỘC

1.1. Giới thiệu về mã độc máy tính

Mã độc hay phần mềm độc hại là các chương trình máy tính có chứa bên trong nó nội dung các mã độc hại được tạo ra với mục đích thực hiện các hành vi bất hợp pháp như: truy cập trái phép, đánh cắp thông tin người dùng, lây lan thư rác, thậm chí thực hiện các hành vi tống tiền (ransomware), tấn công và gây tổn thương cho các hệ thống máy tính… nhằm chuộc lợi cá nhân , hoặc các lợi ích về kinh tế, chính trị hay đơn giản chúng có khi được tạo ra chỉ là một trò đùa ác ý nào đó. Bất kỳ một phần mềm nào là lý do làm tổn thương, phá vỡ đến tính bí mật, tính toàn vẹn và tính sẵn sàng của dữ liệu người dùng, máy tính hoặc môi trường mạng đều có thể được xem như các mã độc. Ngày nay ,cùng với sự phát triển của công nghệ thông tin các ứng dụng máy tính, các phần mềm hệ thống không ngừng thay đổi và phát triển mạnh mẽ, mã độc ban đầu từ chỗ chỉ là những chương trình nhỏ có khả năng tự sao chép đến nay mã độc và các biến thể của nó đang ngày càng trở nên đa dạng, tinh vi mức độ nguy hiểm ngày càng gia tăng đang trở thành một nguy cơ đe dọa trực tiếp đến an toàn, an ninh thông tin.

1.2. Phân loại mã độc

Ban đầu các phần mềm độc hại được tạo ra bằng cách sống kí sinh và lây nhiễm trên các vật chủ là các chương trình có chứa các nội dung thực thi. Các đoạn mã máy nhiễm độc được lây lan tồn tại trong một vài chương trình ứng dụng, các chương trình tiện ích hoặc các hệ thống máy tính thậm chí ngay chính trên các mã được dùng để khởi động máy tính. Các phần mềm độc hại này được xác định và phân loại bởi mục đích độc hại hay các hành vi của nó, có thể kể tên như: Virus, Trojan, Logic Bombs, BackDoors…Các dạng chương trình độc hại kiểu khác là các chương trình mã độc mà tự chúng có khả năng thực thi một các độc lập trên các phần mềm hệ thống mà không cần kí sinh trên các vật chủ là các chương trình ứng dụng hay tệp tin. Các mã độc dạng này thường có khả năng tự nhân bản chúng lợi dụng các lỗ hổng bảo mật để tấn công hay thực hiện các hành vi độc hại, đại diện cho các dạng mã độc này là các sâu máy tính (worm).

Một cách khác để phân biệt các loại mã độc khác nhau là dựa trên mục đích và các hành vi của chúng. Ngày nay mã độc đang ngành càng phát triển và tiến hóa không ngừng trong đó nổi lên là các dạng mã độc mà có khả năng ngụy trang và lén lút thực hiện các hành vi đánh cắp thông tin, thực hiện nghe lén hay thâm nhập như là các phần mềm gián điệp hay các chương trình khai thác lỗ hổng bảo mật trên máy

tính người dùng trong thời gian dài. Một số loại mã độc khác chủ yếu được thực thi mà hành vi của nó tập trung vào mục đích lây lan và phá hoại gây nhiễm cho hệ thống máy tính.

Như vậy có thể nói phần mềm độc hại có thể được tạm chia thành các loại khác nhau theo phương thức và hoạt động của chúng, Các phần mềm chống virus đôi khi không quan tâm đến tên của nó, có khả năng phát hiện ra các kiểu mã độc như sau:

Có 3 đặc điểm thường liên quan với các loại phần mềm độc hại là:

- Mã độc tự nhân bản và nỗ lực lây nhiễm bằng việc tạo các bản sao chép mới hoặc các thể hiện của chính nó. Mã độc cũng có thể được lây nhiễm một cách bị động thông qua thao tác vô ý của người dùng tạo ra các sự kiện kích hoạt hay sao chép nhưng điều này không được coi là tự nhân bản.
- Sự tăng trưởng về số lượng của phần mềm độc hại mô tả sự thay đổi tổng thể trong đó có số lượng lớn các trường hợp là do tự nhân bản, Những mã độc mà không tự nhân bản sẽ luôn luôn có một sự tăng trưởng bằng không nhưng mã độc với một sự tăng trưởng bằng không lại có thể là những mã độc tự nhân bản.
- Phần mềm độc hại ký sinh yêu cầu một số chương trình mã thực thi khác để tồn tại. Thực thi trong ngữ cảnh này nên được hiểu chung là bao gồm bất cứ thứ gì có thể thực thi được, như là các khối khởi động trên đĩa, mã nhị phân trong các ứng dụng và các mã biên dịch. Nó cũng bao gồm các mã nguồn như là các ngôn ngữ kịch bản, hay là các mã yêu cầu biên dịch trước khi chúng được thực thi.

1.2.1. Virus máy tính

Virus là một loại mã độc có các đặc điểm như tự nhân bản, chúng ký sinh trên các vật chủ, virus là một chương trình độc hại mà khi được thực thi nó sẽ cố gắng sao chép chính nó vào bên trong một mã thực thi khác. Khi thành công thì mã chương trình được gọi là mã bị lây nhiễm. Các mã lây nhiễm khi chạy có thể tiếp tục lây nhiễm sang các mã mới. Sự tự sao chép này tồn tại trong các mã thực thi là một đặc tính xác định quan trọng của một virus. Khi virus được lây nhiễm nó có thể thực hiện một loạt các hành vi như thay đổi, xóa, hay sao chép các tệp tin cũng như phát tán chúng trên các hệ thống máy tính.

Hiểu theo cách truyền thống thì Virus có thể lây lan bên trong một máy tính duy nhất hoặc có thể di chuyển từ một máy tính đến máy tính khác bằng việc sử dụng các

phương tiện vận chuyển đa phương tiện của con người như là đĩa mềm, CD-ROM, DVD-ROM, hoặc USB,… Nói cách khác Virus không lây lan thông qua mạng máy tính thay vào đó mạng máy tính là một miền hoạt động của các sâu máy tính. Tuy nhiên Virus hiểu theo cách truyền thống đôi khi được dùng để chỉ bất kỳ phần mềm độc hại nào có khả năng tự sao chép.

Virus có thể được bắt gặp trong các giai đoạn khác nhau của sự tự nhân bản. Một mầm mống nảy sinh là hình thức ban đầu của Virus trước khi diễn ra bất kỳ một sự tự sao chép nào, một Virus mà thất bại trong việc tự tái tạo lại được gọi là một dự định, điều này có thể xảy ra như là kết quả của lỗi trong các virus hoặc sự gặp phải một phiên bản bất ngờ của hệ điều hành. Một virus có thể trú ẩn, nơi mà nó hiện diện nhưng chưa lây nhiễm hay làm bất cứ điều gì ví dụ một cửa sổ có chứa virus có thể ở trên một máy chủ tập tin chạy trên hệ điều hành Unix và không có hiệu lực ở đó, nhưng có thể được xuất và di chuyển sang các máy Windows về cơ bản một Virus được trải qua 4 giai đoạn, giai đoạn 1 là trú ẩn như đã trình bày ở trên, giai đoạn 2 virus thực hiện lây nhiễm bằng việc việc sao chép chính nó vào các tệp tin hay chương trình máy tính khác, giai đoạn 3 virus được kích hoạt để chuẩn bị chạy các đoạn mã độc, giai đoạn 4 virus thực thi các chức năng của nó theo đúng kịch bản mà kẻ đã tạo ra nó.

Một virus máy tính gồm 3 thành phần chính như sau:

Cơ chế lây nhiễm: Làm thế nào để virus có thể lây lan bằng cách sửa đổi mã khác để chứa một sao chép của virus, các cách chính xác thông qua đó một loại virus lây lan được gọi là vector lây nhiễm của nó. Đây không phải là duy nhất, một loại virus lây nhiễm theo nhiều cách được gọi là đa phương.

Kích hoạt: Các cách quyết định xem có hay không để mở một payload.

Payload: là mã lệnh hay cái mà virus thực hiện bên cạnh việc lây nhiễm.

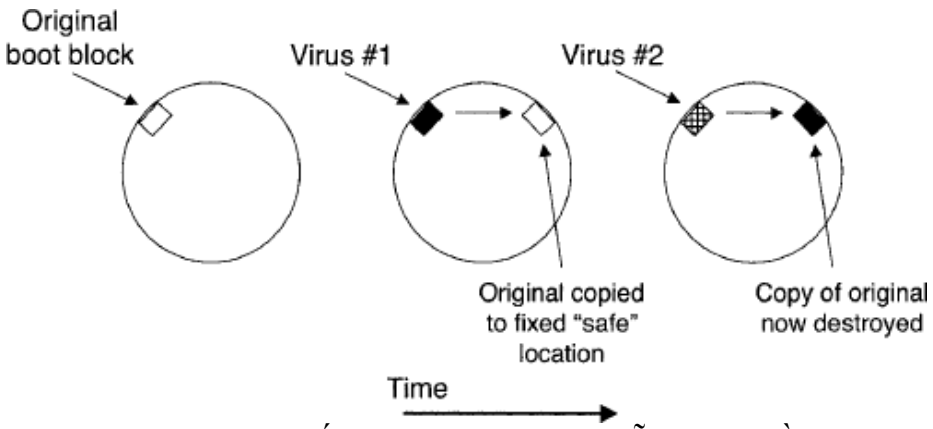
Ngoại trừ cơ chế lây nhiễm, 2 thành phần còn lại có thể tùy chỉnh do vậy lây nhiễm là một trong những chìa khóa định nghĩa các đặc điểm của một virus. Trong sự thiếu vắng tính chất lây nhiễm nếu mã độc hại chỉ chứa thành phần kích hoạt và payload trường hợp này được gọi là mã độc logic bomb.

Virus có thể được phân loại bằng nhiều cách, dựa vào mục tiêu cố gắng lây nhiễm của virus, và dựa vào các phương thức mà virus sử dụng để che giấu bản thân nó với các hệ thống phát hiện và các phần mềm chống virus.

1.2.1.1. Phân loại virus dựa vào các hình thức lây nhiễm

1.2.1.1.1. Virus lây nhiễm Boot-Sector

Virus lây nhiễm Boot-Sector là một virus lây nhiễm bằng việc sao chép chính nó đến khối khởi động bằng cách như vậy sau khi hệ điều hành máy tính tiến hành khởi động nó sẽ thực thi các đoạn mã virus bị chèn vào chứ không phải là các đoạn mã khởi động thông thường. Virus có thể sao chép nội dung của khối khởi động cũ ở nơi khác trên đĩa đầu tiên hoặc nơi cố định an toàn trên ổ đĩa để virus có thể chuyển điều khiển đến nó sau khi hoàn thành xử lý tiến trình khởi động.



Hình 1.1. Khối khởi động bị lây nhiễm bởi nhiều virus [20]

Phương pháp này sẽ gặp vấn đề nếu như máy tính bị nhiễm nhiều lần bởi những loại virus khác nhau khi chúng cùng sử dụng một địa điểm an toàn để lưu trữ đoạn mã khởi động, do ở lần tiếp theo virus sẽ tiếp tục sao chép đoạn mã virus trước đó đến vùng an toàn vô tình ghi đè lên đoạn mã nạp chương trình thực sự do vậy bị xóa và chúng sau khi thực hiện các chức năng của mình không có cách nào có thể quay trờ lại khởi động máy tính, đây là một tổn hại không chủ ý được gây ra bởi virus, ngày nay virus boot không còn được thấy xuất hiện nhiều do tốc độ lây lan chậm và khả năng tiến hóa không cao của nó.

1.2.1.1.2. Virus lây nhiễm tập tin

Virus lây nhiễm tập tin hệ điều hành có một khái niệm về tập tin được thực thi, trong một ý nghĩa rộng hơn các tập tin thực thi cũng có thể bao gồm các tập tin mà có thể chạy bởi các dòng lệnh người sử dụng như "shell". Một tập tin lây nhiễm là một virus lây lan là những tập tin được hệ điều hành hay "shell" xem là để thực thi nó có thể bao gồm các tập tin "batch" và mã kịch bản "shell" nhưng các mã nhị phân thực thi là những mã thực thi phổ biến nhất.

Có 2 vấn đề cần quan tâm chính trong lây nhiễm tập tin đó là:

1. Virus thì nằm ở đâu ?
2. Virus được thực thi như thế nào khi mà các tập tin lây nhiễm chạy ?

Để tiến hành lây nhiễm virus thực hiện ác công việc như sau:

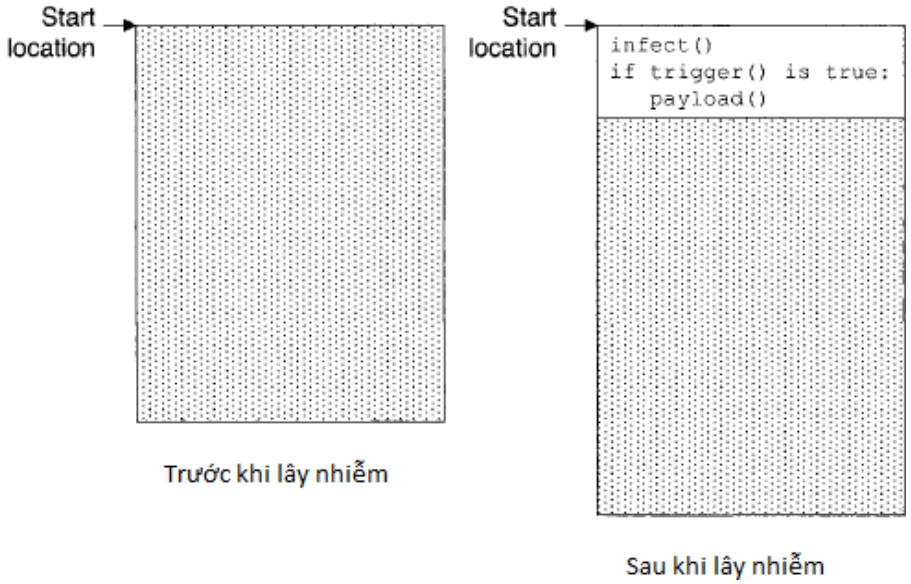
- Tìm kiếm đối tượng tệp thực thi để lây nhiễm
- Nạp các mã độc lây nhiễm vào các tập tin tìm được
- Lưu trú và đảm bảo nó tồn tại duy nhất trong tập tin bị lây nhiễm
- Tiếp tục tìm kiếm các tệp tin khác để tiến hành lẫy nhiễm mã độc

Mỗi một loại virus sẽ có những cách thức để thực hiện lây nhiễm khác nhau song chúng đều được trải qua các nguyên lý bao gồm 3 giai đoạn là:

- Lây nhiễm và chiếm quyền điều khiển
- Chạy các chức năng của nó là các đoạn mã độc được gắn vào tệp tin lây nhiễm
- Trao trả quyền điều khiển cho tệp và đảm bảo tính toàn vẹn ban đầu của dữ liệu.

Thông thường virus được tiến hành đặt ở hai nơi, nơi bắt đầu một tệp tin và nơi kết thúc hoặc một vị trí nào đó trong tệp tin, các kĩ thuật thực hiện của virus được trình bày như sau:

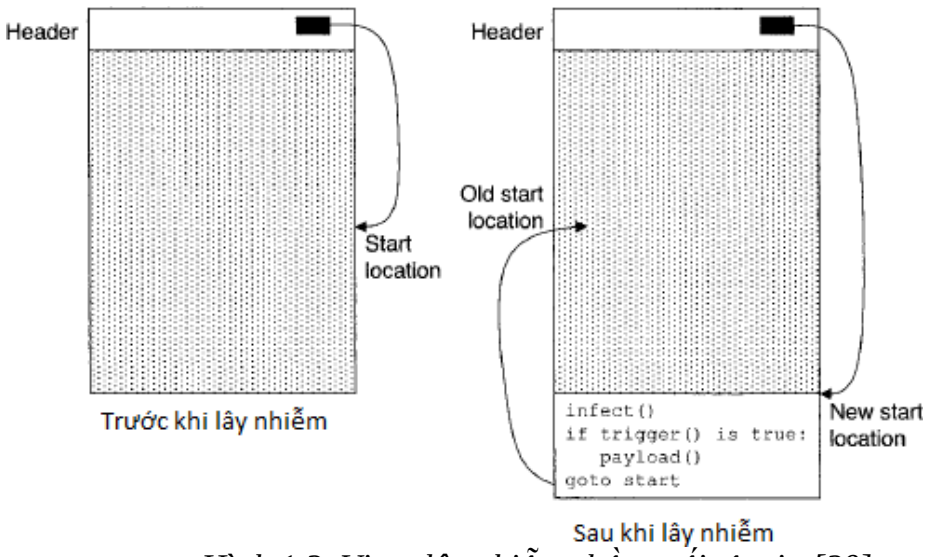
Lây nhiễm phần đầu của tập tin: các tập tin đơn giản thường có định dạng như .EXE, .COM, MS-DOS toàn bộ các tập tin được kết hợp bởi phần mã và phần dữ liệu. Khi được thực thi toàn bộ tập tin sẽ được nạp lên bộ nhớ và thực thi sẽ được bắt đầu bởi lệnh nhảy đến phần đầu tiên nạp tập tin đó. Trong trường hợp này virus được đặt ở vị trí bắt đầu của tập tin nơi mà các tập tin nhận quyền điều khiển đầu tiên khi các tập tin bị lây nhiễm chạy nó sẽ thực thi các đoạn mã độc đầu tiên.



Hình 1.2 virus lây nhiễm phần đầu tệp tin [20]

Lây nhiễm phần cuối của tập tin: gắn thêm các mã vào phần cuối của một tập tin thì cực kì đơn giản, một virus đặt ở vị trí kết thúc một tập tin bằng cách nào nó có thể chiếm đoạt quyền điều khiển, có 2 phương pháp cơ bản như sau:

Các mã lệnh chương trình gốc có thể được lưu lại, và được thay thế bởi một lệnh nhảy đến các mã của virus, sau khi thực thi xong virus sẽ sử dụng quyền điều khiển trỏ đến các mã ban đầu mà nó đã lây nhiễm. Virus có thể sẽ cố gắng chạy trực tiếp các mã lệnh chương trình gốc ngay tại khu vực nó đã lưu hoặc virus có thể phục hồi các mã lệnh gốc này về trạng thái ban đầu và chạy nó.



Hình 1.3. Virus lây nhiễm phần cuối tệp tin [20]

Nhiều các tập tin thực thi có định dạng chỉ rõ vị trí khởi động trong phần đầu của tập tin. Virus có thể thay đổi vị trí khởi động này trỏ đến phần mã của nó và thực thi sau đó lệnh nhảy trở lại vị trí khởi động gốc khi chúng hoàn thành.

Ghi đè vào tập tin: nhằm tránh sự thay đổi về kích thước của một tập tin một vài virus thực hiện ghi đè các phần mã của nó lên các phần mã của tập tin gốc. Việc ghi đè một cách không đúng đắn sẽ gây ra sự thay đổi cũng như phá vỡ cấu trúc của tệp tin ban đầu điều này vô tình làm cho các chương trình chống virus dễ dàng phát hiện ra chúng, có một vài cách mà mức độ rủi ro và phức tạp khác nhau: phương pháp đầu tiên virus hoạt động giống như Boot-virus nó thực hiện lưu các phần nội dung của mã chương trình ban đầu ở một nơi nào đó và sau đó thực hiện ghi đè lên một phần của tập tin, cách thứ 2 là virus có thể thực hiện thao tác nén các dữ liệu và mã chương trình ban đầu sau đó tiến hành ghi đè lên vị trí trống được tạo ra và khi virus thực hiện xong chức năng của nó thì sẽ thực hiện giải nén phần mã gốc để phục hồi lại tập tin ban đầu.

Chèn vào tập tin: dựa vào cấu trúc của các tệp tin khác nhau virus sử dụng kỹ thuật chèn mã của nó vào các vùng còn trống không được sử dụng, các vị trí này thường là các vị trí bên trong của tệp tin, về cơ bản nó cũng có thể chiếm quyền điều khiển để khi tệp tin khởi động sẽ trỏ về đúng vị trí các đoạn mã virus đã chèn, sau khi thực hiện xong các công việc của mình, virus sẽ trỏ lại lệnh thực thi đúng như vị trí ban đầu của tệp tin.

Không nằm trong tập tin: các virus lây nhiễm dạng này hay còn gọi là các virus đồng dạng không giống như các cách lây nhiễm như bên trên loại virus này không làm thay đổi mã của chương trình gốc chúng chủ yếu tồn tại dưới hình thức ngụy trang để có những đặc điểm giống như các tệp tin thực thi là mục tiêu của chúng, chúng tỏ ra là một tệp tin bình thường nào đó nằm trên hệ điều hành và bằng cách chiếm quyền thực thi trước các chương trình gốc dựa trên các cơ chế về quyền ưu tiên trong hệ điều hành hay các hoạt động tìm kiếm shell:

Thông thường các chương trình thực thi sẽ nằm trên các đường dẫn tìm kiếm của hệ điều hành, lợi dụng các cơ chế tìm kiếm ưu tiên virus có thể sao chép chính nó vào các đường dẫn này và tạo tên giống với tên của chương trình mà nó muốn lây nhiễm, sau đó chiếm quyền ưu tiên để hệ điều hành thực thi nó trước khi chương trình gốc thực sự được thực thi.

Các tệp tin mục tiêu cũng có thể bị thay đổi tên và các virus đồng dạng có thể sẽ lấy lại tên gốc của tệp tin mục tiêu đó.

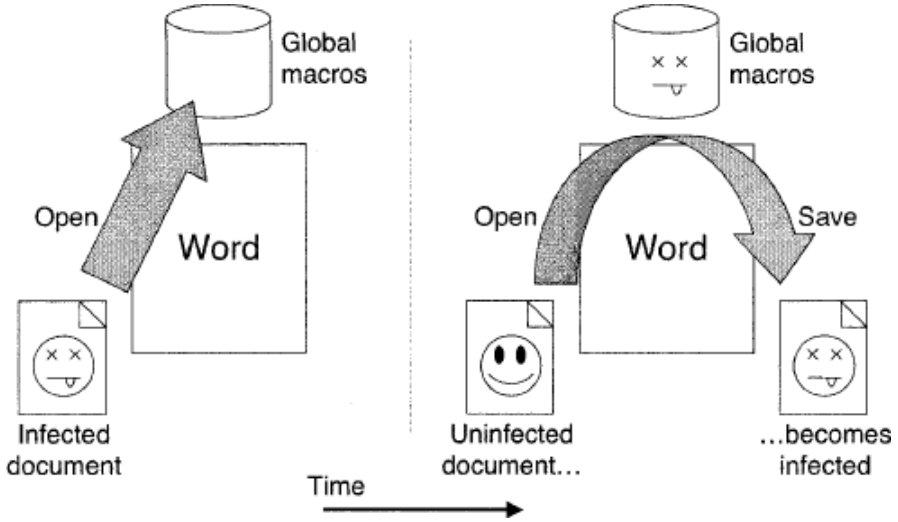
Các định dạng tập tin ELF thường được sử dụng trên các hệ thống Unix có một trình Virus đồng dạng thậm chí cũng có khả năng hoạt động dựa trên các môi trường giao diện đồ họa, một biểu tượng của ứng dụng mục tiêu có thể bị che dấu bởi một biểu tượng của virus đồng dạng. Khi một người sử dụng click chọn vào biểu tượng mà họ nghĩ là biểu tượng của ứng dụng nhưng thay vào đó một virus đồng dạng sẽ thực thi.

Đối với hệ điều hành Windows, virus sẽ tự ghi các nội dung của nó vào một số địa chỉ trong Registry để những lần khởi động tiếp theo nó sẽ được gọi đến và kích hoạt trở lại. Đây là một hình thức sử dụng tìm kiếm shell đối với các tập tin thực thi sau khi chúng được lây nhiễm vào mày tính cớ chế này cho phép virus được thực thi và lây lan các hoạt động của chúng mỗi lần máy tính bật hay khởi động lại.

Kiểm tra tính duy nhất: để đảm bảo tăng tốc độ lây nhiễm thì virus cần phải tránh việc lây nhiễm lặp lại trên các tập tin hay vùng nhớ đã chứa nó vì thế chúng cần phải có cơ chế đảm bảo sự tồn tại duy nhất trên bộ nhớ cũng như trong các tệp tin lây nhiễm, một số phương pháp để thực hiện điều này như sau: trước khi tiến hành lây nhiễm nó thực hiện kiểm tra một đoạn mã trên vùng nhớ và so sánh với đoạn mã virus nếu có sự khác biệt thì nó sẽ tiến hành lây nhiễm lên vùng nhớ này. Hoặc virus có thể xây dựng thêm chức năng cho phép kiểm tra để trước khi lây nhiễm chức năng này sẽ được gọi quyết định có hay không sự tồn tại của nó trong bộ nhớ được dựa vào các giá trị trả về trong thanh ghi hoặc các biến hệ thống sau khi chức năng được gọi.

1.2.1.1.3. Virus Macro

Một số ứng dụng cho phép chứa các tập tin dữ liệu, như các trình xử lý văn bản cho phép nhúng các hàm macro bên trong chúng, Macro là một đoạn mã nhỏ được viết bởi một ngôn ngữ chuẩn được biên dịch bởi ứng dụng, ngôn ngữ này cung cấp đầy đủ các hàm để viết một virus. Do vậy virus Macro chiếm được nhiều lợi thế hơn là virus lây nhiễm tập tin.



Hình 1.4. Mô tả hoạt động của virus macro [20]

Khi một tài liệu chứa một macro được nạp bởi ứng dụng, macro có thể là lý do gây ra việc chạy tự động, cho phép kiểm soát điều khiển đến các virus macro. Một vài ứng dụng có thể tạo ra các cảnh báo người dùng về sự xuất hiện của marco trong văn bản tuy nhiên những cảnh báo này thường bị bỏ qua một cách dễ dàng. Trong hệ điều hành Windows thì Virus macro chủ yếu là các đoạn mã được viết bởi ngôn ngữ lập trình Visual Basic đây là một ngôn ngữ được sử dụng trong phần mềm thuộc bộ Microsoft Office như Word, PowerPoint, Excel.

Một tài liệu bị lây nhiễm thường bởi 2 kiểu macro với các tính chất:

AutoOpen: là các đoạn mã macro sẽ tự động chạy khi mà một tập tin được mở các macro này sẽ chiếm được quyền kiểm soát tập tin bị lây nhiễm.

FileSaveAs: là các đoạn mã lưu tập tin trong macro sẽ chạy khi mà tác vụ “File save as” trong trình đơn được lựa chọn. Nói cách khác đoạn mã này có thể được sử dụng để lây nhiễm đến bất kì tài liệu nào chưa bị nhiễm virus mà được lưu bởi người dùng.

Đứng trên quan điểm kỹ thuật, ngôn ngữ cho macro thì dễ dàng được sử dụng hơn các ngôn ngữ lập trình bậc thấp vì vậy lợi thế của macro làm giảm các rào cản tạo ra một virus.

1.2.1.2 Phân loại virus dựa trên các chiến lược ẩn náu

Một cách khác để phân loại virus là dựa vào các cách làm thế nào để chúng có thể ẩn náu và qua mặt được người dùng cũng như các phầm mềm diệt virus.

1.2.1.2.1. Virus mã hóa

Đối với một virus được mã hóa ý tưởng là phần thân của virus bao gồm phần lây nhiễm, kích hoạt và payload sẽ được mã hóa thông qua một vài cách thức, mục đích là gây ra sự khó khăn để có thể phát hiện ra chúng, khi phần thân của virus ở dạng mã hóa nó sẽ không thực thi cho đến khi được giải mã, sau một vòng giải mã sẽ giải mã phần thân của virus và chuyển điều khiển trỏ đến nó, bằng cách này virus giải mã sẽ cung cấp ít thông tin về chúng hơn với các phần mềm phát hiện mã độc. Một quá trình giải mã có thể giải mã phần thân virus tại chỗ hoặc ở một nơi khác, sự lựa chọn này có thể được quyết định bởi các ràng buộc bên ngoài như là khả năng viết mã của chương trình bị lây nhiễm.

Hình 1.5. Mô tả quá trình virus được giải mã [20]

Một điểm yếu lớn nhất của phương pháp mã hóa ở trên là phần thân virus sẽ được mã hóa giống nhau ở một lần lây nhiễm kế tiếp, sự không thay đổi này làm cho virus dễ dàng bị phát hiện giống như là một virus không hề che dấu. Điều này sẽ được khắc phục với một khóa mã sinh ngẫu nhiên : Khóa được sử dụng cho việc mã hóa sẽ thay đổi ngẫu nhiên với mỗi một lây nhiễm mới. Rõ ràng, quá trình giải mã virus cần phải được cập nhật cho mỗi lây nhiễm để kết hợp chặt chẽ với khóa mới.

1.2.1.2.2. Virus tàng hình

Virus tàng hình là một virus nỗ lực tiến hành các bước để che dấu sự lây nhiễm của chính nó, không chỉ có phần thân virus, một virus tàng hình luôn cố gắng ẩn mọi thứ không chỉ với các phần mềm chống virus. Một vài kỹ thuật virus tàng hình thực hiện được trình bày bên dưới:

Một nhãn thời gian của tập tin gốc bị lây nhiễm có thể được khôi phục sau khi lây nhiễm do đó tập tin không thấy sự thay đổi mới.

Virus có thể được lưu dữ hoặc có khả năng tái sinh tất cả những thông tin về tập tin trước khi lây nhiễm bao gồm nhãn thời gian của nó, kích thước tập tin, và nội dung của tập tin, sau đó các cuộc gọi vào/ra hệ thống có thể bị ngăn chặn, và virus sẽ phát lại các thông tin ban đầu đề đáp ứng bất kì hoạt động vào/ra nào trên các tập tin bị lây nhiễm, làm cho nó xuất hiện như là không bị lây nhiễm. Kỹ thuật này cũng được ứng dụng đến các khối khởi động vào ra.

Phương pháp chính xác của việc ngăn chặn các cuộc gọi vào/ra thì phụ thuộc vào hệ điều hành. Với MS-DOS là một ví dụ những yêu cầu vào/ra được tạo ra từ các cuộc gọi ngắt, toàn bộ xử lý được đặt thông qua véc tơ ngắt, virus chỉ cần sửa véc tơ ngắt để chèn chính nó vào chuỗi xử lý ngắt. Trên các hệ thống khác thì biểu diễn vào/ra sử dụng các thư viện chia sẻ, vì vậy một virus có thể đặt chính nó thành một khóa vào trong thư viện chia sẻ những công việc thường lệ để ngăn các cuộc gọi vào ra với hầu hết các ứng dụng. Một số hệ thống lưu trữ các bộ nạp khởi động thứ cấp như các khối đĩa liên tiếp, để làm nhiệm vụ nạp các khởi động chính đơn giản hơn .Trên các hệ thống này, có 2 cách xem các bộ nạp khởi động thứ cấp như là một chuỗi các khối và như là một tập tin trong hệ thống tập tin. Một virus có thể chèn chính nó vào khối khởi động thứ cấp của khối và di chuyển các khối ban đầu đến nơi khác trong hệ thống tập tin. Kết quả là mọi thứ tỏ ra bình thường, hệ thống tập tin cho thấy không có sự thay đổi nào rõ ràng, nhưng trên thực tế virus đã bị ẩn và chạy một cách tự do của bộ nạp khởi động chính.

Kỹ thuật tàng hình chồng chéo với các kỹ thuật được sử dụng bởi rootkit, Rootkits là một bộ công cụ cho những người đã đột nhập vào máy tính, chúng sử dụng những bộ công cụ toolkit để ẩn và tránh bị phát hiện, mã độc ngày nay cũng sử dụng rootkit ví dụ như Ryknos Trojan horse cố gắng che dấu chính nó bằng việc sử dụng một rootkit dành cho quản lý bản quyền số.

1.2.1.2.3 Virus Oligomorphism

Giả sử một virus mã hóa với khóa mã hóa ngẫu nhiên được thay đổi với mỗi lần lây nhiễm mới, chỉ có những phần không thay đổi của virus là các mã nằm trong vòng lặp giải mã. Các phần mềm chống virus sẽ khai thác điều này để phát hiện, vì vậy sự phát triển tiếp theo hợp lý là thay đổi mã trong vòng lặp giải mã với mỗi lây nhiễm.

Một virus oligomorphic hay là virus bán đa hình là một virus được mã hóa trong đó có một hữu hạn nhỏ số lượng các vòng lặp giải mã khác nhau theo ý của nó. Vì vậy virus này nó có thể sinh ra được hữu hạn các biến thể khác nhau.

Virus này chọn một vòng lặp giải mã mới từ vùng chứa cho mỗi lần lây nhiễm mới ví dụ: Whale đã có 30 biến thể giải mã khác nhau và Memorial có 96 biến thể giải mã.

Trong khả năng phát hiện, oligomorphic chỉ tạo ra một virus hơi khó khăn để spot. Thay vì tìm kiếm một vòng lặp giải mã cho virus, phần mềm chống virus có thể đơn giản thực hiện liệt kê ra các vòng giải mã mà virut có khả năng và tìm ra kiếm tất cả biến thể của chúng.

1.2.1.2.4. Virus đa hình

Một virus đa hình bề ngoài thì giống như một virus Oligomorphism, cả hai cùng mã hóa virus, cả hai cùng thay đổi vòng lặp giải mã của chúng cho mỗi lần lây nhiễm, tuy nhiên một virus đa hình có một số lượng vô hạn các biến thể của vòng lặp giải mã. Tremor là một ví dụ nó có tới gần sáu tỷ khả năng lặp giải mã tức là có thể sinh ra 6 tỷ biến thể khác nhau. Có thể nói virus đa hình rõ ràng không thể bị phát hiện bởi liệt kê tất cả các kết hợp có thể của chúng.

Như vậy có thể thấy một virus đa hình là một virus thực hiện các phương pháp che dấu dựa vào việc mã hóa các thân mã chương trình của nó bằng việc thay đổi các cách thức mã hóa hay thay đổi phương thức giải mã nhiều lần, chúng có khả năng tinh vi lẩn trốn trước sự tìm kiếm của các phần mềm diệt virus bởi các biến thể sinh ra ngẫu nhiêntheo thời gian và các đối tượng lây nhiễm.

1.2.1.2.5. Virus siêu đa hình

Virus siêu đa hình là virus mà đa hình phần thân mã bên trong của nó. Chúng không thực hiện mã hóa do đó không cần các vòng giải mã tuy nhiên để tránh bị phát hiện bởi sự thay đổi, một phiên bản mới hay biến thể của phần thân virus được thực hiện cho mỗi lần lây nhiễm mới.

Tất cả các kỹ thuật sửa đổi mã được sử dụng bởi virus đa hình đều được áp dụng vào virus siêu đa hình, cả 2 đều sử dụng một “mutation engine” (“mutation engine” là một chương trình máy tính có thể được sử dụng để chuyển đổi một chương trình thành phiên bản tiếp theo bao gồm các mã lệnh khác nhau nhưng hoạt động với các chức năng tương tự nhau) , ngoại trừ việc một virus đa hình thì không cần thay đổi “engine” cho mỗi lần lây nhiễm bởi vì nó có thể cư trú bên trong phần mã hóa của

virus. Ngược lại "mutation engine" của một virus siêu đa hình sẽ tự biến đổi chính nó cho mỗi lần lây nhiễm.

Có thể nói virus siêu đa hình rất tinh vi với khả năng lai tạo và kết hợp đa hình chúng tao ra vô số các biến thể chỉ từ một mã virus ban đầu làm cho các phần mềm tìm và diệt virus vô cùng khó khăn trong việc phát hiện ra chúng và bị qua mặt dễ dàng.

1.2.2. Logic Bomb

Logic bomb là loại mã độc không tự nhân bản chúng có thể ký sinh trên các vật chủ, một mã độc logic bomb bao gồm 2 phần chính là :

Một payload là các mã nạp có thể là bất cứ thứ gì nhưng có ý đồ xấu hay là các mục đích của mã độc như là phá hoại hay phá hủy các chức năng an toàn của hệ thống.

Một trigger (kích hoạt) là một điều kiện đúng hoặc sai được đánh giá và điều khiển khi một payload được thực thi. Một điều kiện kích hoạt chính xác thì được giới hạn bởi khả năng sáng tạo và có thể dựa trên các điều kiện địa phương như là thời gian, ngày, tháng , năm để thực hiện kích hoạt, dựa vào một đăng nhập người dùng, hay một phiên bản hệ điều hành, hoặc bất kỳ một sự kiện nào đó mà kẻ viết mã độc nghĩ ra . Kích hoạt cũng có thể được thiết kế để kích "nổ" từ xa.

Đặc tính của mã độc logic bomb là chúng sẽ cư trú bằng cách ký sinh trên các phần mềm thông thường hoặc đứng một cách độc lập bằng cách nào đó ẩn nấp khỏi sự phát hiện của người dùng hay các phần mềm diệt virus giai đoạn này hầu như chúng không thực hiện hành vi gì cho đến khi điều kiện kích hoạt xảy ra hay một thông điệp tấn công được gửi các mã độc hại bên trong sẽ được thực thi.

1.2.3. Trojan Horse

Mã độc Trojan Horse là một cái tên được xuất phát từ một điển tích "Con ngựa thành Troy" trong thần thoại Hy Lạp. Trojan là một loại mã độc mặc dù không có khả năng tự nhân bản tuy nhiên nó thường lây nhiễm vào hệ thống với những thể hiện rất bình thường dưới hình thức đóng giả như là các chương trình phần mềm hữu ích thực hiện những nhiệm vụ thông thường và hợp pháp, nhưng thực chất bên trong lại bí mật thực hiện một số nhiệm vụ nguy hiểm, ác tính mà kẻ có ý đồ xấu đã cài đặt vào nó. Ví dụ như một mã độc Trojan có thể là một chương trình đánh cắp thông tin tài khoản và mật khẩu người dùng bằng cách khi người dùng thực hiện chức năng đăng nhập mặc dù các thông tin được gõ vào là chính xác tuy nhiên chúng sẽ tạo ra các nhắc nhở

thông báo rằng tên tài khoản và mật khẩu không đúng và thông báo yêu cầu người dùng nhập lại vào đó để xác thực trước khi chạy một chương trình đăng nhập thực sự. Người dùng không nghi ngờ rằng họ đã thực hiện một lỗi đánh máy và nhập lại các thông tin mà không hề hay biết. Đôi khi các Trojan cũng có thể được cài đặt các chức năng gián điệp được điều khiển và thực hiện các hành vi theo yêu cầu của máy chủ từ xa,

Chúng có thể ký sinh trên các vật chủ là các phần mềm thông thường và thực hiện các chức năng của vật chủ một cách bình thường nhưng lại thực hiện các chức năng độc hại một cách riêng biệt hoặc làm những công việc của chương trình chủ nhưng chỉnh sửa các chức năng để thực hiện ý đồ ác ý, núp dưới danh một phần mềm hữu ích để thực hiện các hành vi gây hại. Trên các máy tính chạy hệ điều hành Windows các Trojan thường được xuất hiện đính kèm thư điện tử bằng cách đặt tên dưới danh nghĩa là các chương trình lương thiện như các chương trình văn bản, tập tin .txt hay các tập tin đa phương tiện tuy nhiên ẩn bên trong lại là các tập tin thực thi như là .exe, .com, .scr, .bat vì một số phần mềm được cấu hình trong Windows mặc định không hiển thị thông tin này vì vậy chúng đánh lừa người dùng dụ dỗ họ mở các chương trình này và vô tình kích hoạt chúng. Trojan đôi khi cũng xuất hiện dưới dạng các biểu tượng thông thường của các phần mềm hữu ích thậm chí chúng có thể đóng giả chính các chương trình diệt virus và dễ dàng qua mặt người dùng thông thường.

1.2.4. Back Door

Back door là một mã độc không tự tái tạo và chúng thường kí sinh trên các vật chủ một back door là bất kì một cơ chế nào cái mà cho phép vượt qua các kiểm tra an ninh thông thường. Các lập trình viên đôi khi tạo ra back door là một chức năng bí mật trong một chương trình cho những lý do chính đáng như là bỏ qua một quá trình xác thực tốn thời gian khi gỡ lỗi một máy chủ mạng hay kiểm tra chương trình và khắc phục lỗi. Giống như là logic bomb, back door có thể được cài đặt trên các mã hay chương trình hợp pháp hoặc như một chương trình độc lập.

Một loại đặc biệt của back door là RAT được viết tắt của công cụ quản trị từ xa (Remote Administration Tool) hay là truy cập Trojan từ xa (Remote Access Trojan) điều này phụ thuộc vào người được hỏi. Các chương trình này cho phép một máy tính có thể giám sát và điều khiển từ xa, Người sử dụng có thể chủ ý cài đặt để truy cập vào một máy tính để làm việt ở nhà hoặc cho phép những nhân viên trợ giúp chẩn đoán và sửa chữa máy tính từ xa. Tuy nhiên nếu mã độc lén lút cài đặt một RAT trên một máy tính, sau đó nó sẽ mở ra một cửa hậu (back door) trở lại trong máy tính đó. Back door khi chạy trên các máy tính bị nhiễm sẽ lưu trú trong bộ nhớ đợi lệnh điều khiển

từ các cổng dịch vụ, chúng mở ra các cổng cho phép kẻ tấn công truy cập vào máy nạn nhân từ các cuộc tấn công mạng từ đó kẻ tấn công sẽ chiếm quyền giám soát và điều khiển máy nạn nhân. Back door tỏ ra rất nguy hiểm bởi khả năng lẩn trốn của nó đôi khi chúng được hẹn trước thời gian để kết nối ra ngoài vì thế trong thời gian lưu trú chúng không để lộ bất kỳ hành vi hay thông tin gì cho phép các phần mềm diệt mã độc phát hiện ra.

1.2.5. Sâu máy tính (Worm)

Sâu máy tính là loại mã độc có khả năng tự nhân bản tuy nhiên chúng thường xuất hiện như những chương trình độc lập mà không cần tập tin chủ để mang nó. Do đó nó không ký sinh vào các tệp tin hoặc một khu vực nào trên đĩa cứng bản thân nó đã là một chương trình hoàn chỉnh vì vậy không thể dùng các chương trình dò quét tệp tin để phát hiện và loại bỏ sâu máy tính. Môi trường hoạt động của sâu máy tính là mạng Internet chúng đặc biệt mạnh trong khả năng tự nhân bản nếu như virus cần có một sự kiện từ người dùng để kích hoạt tự sao chép và lây nhiễm như là : Mở tập tin, khởi động máy tính, sao chép tập tin….Sâu máy tính hoàn toàn có khả năng sao chép và nhân bản chính nó mà không cần bất kỳ thao tác nào của người sử dụng đây là một đặc điểm giúp chúng lan rộng với tốc độ chóng mặt. Mục đích của sâu máy tính là làm tiêu tốn tài nguyên băng thông của mạng máy tính và phá hoại các hoạt động của hệ thống như là phá hủy tệp tin, tạo ra cửa hậu cho phép tin tặc xâm nhập và điều khiển trái phép hệ thống. Chính nhờ khả năng về tốc độ lan truyền và khả năng lẫy nhiễm xuyên các hệ thống trong khoảng thời gian ngắn làm cho các tin tặc ngày càng ưu thích sâu máy tính hơn là các virus thông thường, ngày nay mục tiêu thường thấy của sâu máy tính là thực hiện các cuộc tấn công từ chối dịch vụ Ddos hoặc thực hiện xâm nhập cài đặt các back door và tiến hành các công việc phá hoại vô cùng nguy hiểm.

Sâu máy tính bao gồm 2 loại chính là sâu dịch vụ mạng và sâu gửi điện thư hàng loạt.

Sâu dịch vụ mạng (Network Service Worm): chúng phát tán và lây lan thông qua việc khai thác các lỗ hổng bảo mật của các dịch vụ mạng, các ứng dụng hay hệ điều hành. Sau khi lây nhiễm vào hệ thống chúng sẽ dò quét tất cả các ứng dụng hay hệ thống chạy cùng dịch vụ với các lỗ hổng đang bị khai thác và tiếp tục tấn công trên diện rộng với tốc độ chóng mặt chúng có thể làm tê liệt mạng máy tính và các hệ thống an ninh, một số sâu dịch vụ mạng như là Sasser được phát hiện lây lan trên mạng ngày1/5/2004 bằng việc khai thác lỗ hổng bảo mật của dịch vụ LSASS (Local

Security Authority Subsystem Service) tấn công vào các máy tính cài Windows 2000/ XP/ Server 2003. Hay một sâu dịch vụ nổi tiếng khác là Witty.

Sâu gửi điện thư hàng loạt (Mass Mailing Worm): Chúng là các sâu tấn công vào dịch vụ thư điện tử cách thức hoạt động của chúng là lây lan trên các hệ thống thư điện tử, một khi hệ thống bị lây nhiễm nó sẽ dò quét danh sách các địa chỉ email và tự động gửi bản sao của chính nó tới những địa chỉ mà nó tìm được bằng cách sử dụng các email máy khách thuộc hệ thống bị lây nhiễm hoặc email thuộc hệ thống do nó tạo ra, nó có thể gửi đồng loạt các thư này vào mạng lưới của hệ thống làm cho các máy chủ trở nên quá tải và làm cho hệ thống chịu một lượng lớn "bomb mail". Loại sâu này hiện nay chủ yếu được dùng với mục đích quảng cáo như phát tán các thư rác hay những chương trình mã độc nguy hiểm khác một số ví dụ về sâu máy tính gửi điện thư hàng loạt là: Beagle, Mydoom và Netsky.

1.3. Các kỹ thuật phát hiện mã độc

Mục đích của việc phát hiện hiện mã độc là đưa ra những cảnh báo sớm để có cơ chế ngăn chặn kịp thời trước khi các mã độc thực hiện các hành vi hay chức năng của chúng chính vì vậy vai trò của phát hiện mã độc là hết sức quan trọng, chúng ta luôn luôn phải tìm kiếm và phát hiện sự tồn tại của mã độc ngay cả khi chúng không thực thi hay làm bất cứ điều gì, thậm chí nếu một mã độc không hoạt động trên một hệ thống việc phát hiện ra nó vẫn rất cần thiết bởi vì điều đó đảm bảo rằng nó sẽ không ảnh hưởng đến hệ thống khác. Lấy hệ thống email là một ví dụ nơi mà các máy nhận có thể chạy trên hệ thống hay chứa những ứng dụng, dịch vụ hoàn toàn khác với máy chủ vì vậy phải đảm bảo rằng mã độc không thể được thực thi trên bất kỳ hệ thống nào.

Các kỹ thuật phát hiện mã độc là một quá trình tìm kiếm và thẩm định xem một chương trình phần mềm có thể đã bị lây nhiễm mã độc hay bên trong có chứa các đoạn mã được xem là mã độc hay không, thêm vào đó các hành vi của chúng cũng được phân tích và xem xét là nhóm các hành vi thông thường hay các hành vi thuộc về mã độc, dựa vào các kết quả đó để có thể chứng minh và phát hiện sự tồn tại của mã độc trên các hệ thống.

1.3.1. Các kỹ thuật phát hiện dựa trên 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 của nó gồm có 3 phương pháp chính là kỹ thuật dò quét, chẩn đoán dựa trên kinh nghiệm và kiểm tra tính toàn vẹn.

1.3.1.1. Kỹ thuật dò quét (scanner)

Thông thường mỗi một mã độc được biểu diễn bởi một hay nhiều mẫu, hoặc là các dấu hiệu (signatures), chuỗi tuần tự các byte là những cái được coi là đặc trưng duy nhất của mã độc. Các dấu hiệu này đôi khi còn được gọi là các chuỗi (scan strings) và chúng không cần bất kỳ một ràng buộc về chuỗi nào. Một vài phần mềm chống mã độc có thể hỗ trợ việc sử dụng các ký tự đại diện (wildcards) cho mỗi một byte tùy ý, một phần của một byte, hoặc không hay nhiều byte. Quá trình phát hiện mã độc bằng cách tìm kiếm thông qua một tập tin với các dấu hiệu của nó thì được gọi là scanning và các mã được tìm thấy được gọi là một scanner. Cụ thể hơn nữa, quá trình phát hiện được thực hiện thông qua một dòng các mã byte chúng có thể là toàn bộ nội dung của một khối khởi động, toàn bộ nội dung của tập tin, hoặc là một phần của tập tin được đọc hoặc ghi, hay cũng có thể là các gói tin mạng.

Với hàng trăm ngàn dấu hiệu để phát hiện việc tìm kiếm chúng tại một thời điểm thì không khả thi chút nào. Một trong thách thức lớn nhất của kỹ thuật này là tìm ra các thuật toán có khả năng tìm được nhiều mẫu một cách hiệu quả và đồng thời đánh giá được chúng.

1.3.1.2. Kỹ thuật Static Heuristics

Kỹ thuật này được áp dụng để nhân lên khả năng chuyên gia trong các phần mềm chống virus, chẩn đoán dựa trên kinh nghiệm trong phương pháp phân tích tĩnh có thể tìm thấy các mã độc đã biết hoặc chưa biết bằng cách tìm kiếm một mẩu mã mà có những đặc điểm chung giống như là một mã độc thay vì scanning các dấu hiệu đặc biệt của mã độc. Đây là một kỹ thuật phân tích tĩnh có nghĩa là các mã sẽ được phân tích mà không thực thi và không có gì đảm bảo về bất kỳ một mã nghi ngờ được tìm thấy sẽ thực thi khi nào. Kỹ thuật này được thực hiện thông qua 2 bước:

Thu thập dữ liệu: Dữ liệu thu thập có thể được sử dụng từ bất kỳ một kỹ thuật dựa trên kinh nghiệm nào, có hay không một kỹ thuật kinh nghiệm phân loại chính xác các đầu vào điều đó không thực sự quan trọng bởi vì các kết quả của nhiều kinh nghiệm sẽ được kết hợp và được phân tích sau đó. Một scanner có thể được sử dụng để xác định vị trí một dấu hiệu là các biểu hiện được chỉ ra của những mã đáng nghi được gọi là các booster. Sự hiện diện của các booster này làm tăng khả năng các mã sẽ được phân tích là các mã độc như là: các mã Junk, tự thay đổi mã, sử dụng các cuộc gọi hàm API không được cung cấp, điều khiển véc tơ ngắt, sử dụng các lệnh bất thường đặc biệt là cái không được sinh ra bởi trình biên dịch, chuỗi chứa những từ ngữ khiêu dâm, hoặc các tín hiệu rõ ràng như là một từ “virus”. Một việc cũng không kém phần

quan trọng là tìm kiếm những thứ xuất hiện trong một mã bình thường đây là những cái mà mã độc không thường làm. Ví dụ virus thì không thường tạo một hộp thoại pop-up cho người sử dụng. Điều này sẽ được xem xét là một heuristic phủ định hay một stopper.

Các kinh nghiệm heuristic khác có thể được tính toán mà không dựa trên scanning như là:

Sự khác nhau giữa một điểm vào và kết thúc của một tập tin có thể được tính toán, các giá trị cực nhỏ khi thực hiện so sánh với những giá trị giống như vậy ở một tập tin không bị lây nhiễm.

Phân tích phổ của mã lệnh có thể được thực hiện, tính toán các tần số của các byte hoặc là các lệnh được sử dụng trong mã. Các mã được mã hóa sẽ có dấu hiệu phổ khác với mã không được mã hóa.

Phân tích: phân tích dữ liệu tĩnh heuristic có thể xác định đơn giản như là đánh các trọng số cho mỗi giá trị heuristic và tính tổng các kết quả. Nếu tổng số vượt qua một giá trị ngưỡng được sử dụng như là một căn cứ phát hiện thì dữ liệu đầu vào có thể đã bị lây nhiễm.

Một số phương pháp phức tạp trong các phân tích dữ liệu có thể sử dụng trí tuệ nhân tạo như là mạng neural, hệ chuyên gia, hay các kỹ thuật khai phá dữ liệu.

Kỹ thuật Static heuristics có thể được xem như là một cách để giảm các yêu cầu tài nguyên của kỹ thuật scanner. Toàn bộ dấu hiệu của mã độc trong một cơ sở dữ liệu mã độc có thể được chắt lọc giảm xuống một tập nhỏ hơn, tổng quát hơn. Một kỹ thuật quét mã độc có thể tìm kiếm những dấu hiệu đặc trưng ngắn này và nạp vào trong bộ dữ liệu chứa toàn bộ các dấu hiệu nếu phù hợp với những gì đã tìm thấy. Điều này làm giảm bớt yêu cầu lưu dữ toàn bộ các dấu hiệu đặc trưng trong bộ nhớ.

1.3.1.3. Kỹ thuật kiểm tra sự toàn vẹn (Integrity Checkers)

Ngoại trừ một số virus đồng dạng các mã độc chủ yếu hoạt động bằng cách thay đổi các tập tin. Kỹ thuật kiểm tra tính toàn vẹn nhằm khai thác các hành vi này để tìm ra mã độc bằng cách xem các thay đổi trái phép vào các tập tin. Một kiểm tra toàn vẹn được khởi đầu bằng việc tính và lưu dữ một checksum cho mỗi tập tin trong hệ thống được xem xét. Sau đó một checksum của tập tin cần kiểm tra sẽ được tính lại và so sánh với giá trị checksum gốc của nó. Nếu checksum khác nhau có nghĩa là đã có một sự thay đổi diễn ra. Kỹ thuật này được sử dụng rất phổ biến trong các phần mềm chống mã độc ban đầu các mã độc được tìm thấy sẽ được sử dụng một hàm băm

như là MD5,SHA,CRC… để tính toán ra giá trị băm duy nhất của nó sau đó các giá trị này được lưu trong cơ sở dữ liệu như là các dữ liệu mẫu định danh mã độc đó. Khi muốn kiểm tra một tập tin có phải là mã độc hay không chúng ta chỉ cần tính toán lại hàm băm và so sánh giá trị kết quả với cơ sở dữ liệu đã có. Nếu tồn tại một mã băm như vậy trong cơ sở dữ liệu mã độc thì có thể biết chính xác nó là một mã độc.

1.3.2. Các kỹ thuật phát hiện dựa trên phân tích động

Kỹ thuật phát hiện mã độc dựa trên phân tích động là kỹ thuật quyết định một tập tin có bị lây nhiễm hay không thông qua việc thực thi các mã chương trình và quan sát các hành vi của nó. Để thực hiện được phương pháp này cần phải thiết lập môi trường để mã độc thực thi và các công cụ cho phép quan sát các hành động của chúng như: theo dõi các tiến trình, theo dõi các thông tin về registry, sự thay đổi của các tệp tin, thư mục, theo dõi sự thay đổi của lưu lượng mạng và các kết nối TCP/IP… Tất cả các hành vi của mã độc sẽ được lưu dưới dạng các nhật ký phục vụ cho công việc phân tích sau này.

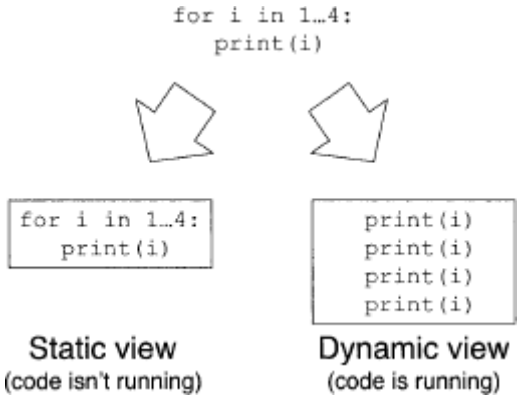
1.3.2.1. Kỹ thuật Behavior Monitors/Blockers

Một behavior blocker là một kỹ thuật chống mã độc giám sát các hành vi thực thi của một chương trình trong thời gian thực, theo dõi các hành động, các khối lệnh khả nghi của nó, nếu những hành động này được tìm thấy behavior blocker có thể ngăn chặn những hành động khả nghi thành công và chấm dứt các tiến trình của một chương trình hay là có thể hỏi người dùng cho cho các hành động phù hợp để xử lý. Một behavior blocker thì tìm kiếm điều gì? Về cơ bản một behavior blocker xem xét một chương trình thuộc vào các hành vi được xem xét là bị chặn hay là các hành vi thông thường. Chúng có thể là những hành vi liên quan đến tập tin như mở, sửa đổi hay xóa, hay những thao tác fomat ổ đĩa phân vùng, thay đổi registry, kết nối mạng…Một hành vi thông thường được mô hình hóa trong 3 cách được miêu tả như sau:

a. Một hành động được cho phép điều này được gọi là một phát hiện positive
b. Một hành động không được cho phép điều này được gọi là một phát hiện negative
c. Một cách kết hợp cả 2 giống như kỹ thuật static heuristics bao gồm có boosters và stoppers.

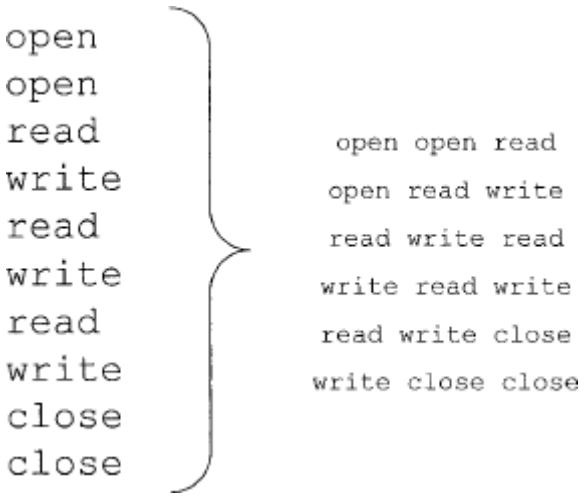
Một điều tương tự được rút ra từ các hệ thống miễn dịch trong tự nhiên, các behavior blocker cố gắng để nhận thức chính nó từ những thứ không phải là nó, hay

là các hành vi bình thường từ những hành vi bất bình thường. Cách tư duy này là cái mà hệ thống miễn dịch cần làm để phân biệt các tế bào bình thường xâm nhập từ bên ngoài. Tuy nhiên điều này cần phải được xem xét cẩn thận vì đôi khi các hành vi bất thường không có nghĩa chắc rằng nó là một hành vi thuộc về mã độc.



Hình 1.6 So sánh 2 kỹ thuật phân tích tĩnh và động [20]

Kỹ thuật behavior blocker có thể tìm kiếm những dấu hiệu theo phương pháp phân tích động mà những biểu hiện là một hành vi thuộc về mã độc. Ví dụ nhìn vào cách hoạt động vào/ra được thêm vào của một mã độc cũng thể hiện các dấu hiệu của nó.



Các dấu hiệu đôi khi là chuỗi các hành vi như hình minh họa ở trên là chuỗi các hành vi liên quan đến các thao tác xử lý tập tin với độ dài là 3. Dựa vào các dấu hiệu này chúng ta hoàn toàn có thể tìm ra được những bất thường trong các hành vi của một mã độc.

1.3.2.2. Kỹ thuật Emulation

Giám sát các hành vi kỹ thuật Behavior blocking cho phép mã chương trình chạy trên máy thực. Ngược lại kỹ thuật phát hiện mã độc sử dụng mô phỏng giả lập (emulation) cho phép các mã được chạy và phân tích trong một môi trường mô phỏng giả lập. Với kỳ vọng rằng dưới môi trường mô phỏng một mã độc sẽ tiết lộ chính nó. Bởi vì bất kỳ một mã độc nào được tìm thấy sẽ không được chạy trên máy tính thực sự vì vậy không có một hiểm họa nào xảy ra.

Mô phỏng có thể được áp dụng qua 2 cách mặc dù ranh giới phân biệt giữa chúng là mập mờ:

Dynamic heuristic: đây là kỹ thuật có độ chính xác tương tự như kỹ thuật static heuristic chỉ có một sự khác nhau là các dữ liệu được thu thập như thế nào: dynamic heuristic phân tích thu thập các dữ liệu của nó từ môi trường giả lập về các mã chương trình sẽ được phân tích. Các phân tích này được thực hiện giống như là cách mà kỹ thuật static heuristic đã thực hiện. Dynamic heuristic cũng có thể tìm kiếm các đặc trưng tương tự như là behavior blockers, như là các lời gọi hệ thống. Môi trường giả lập là một môi trường ảo an toàn trong đó các mã chạy được giám sát. Dynamic heuristics có thể được sử dụng hiệu quả để phát hiện các dấu hiệu hành vi của virus siêu đa hình.

Generic decryption: với những virus đa hình vòng giải mã có thể là một yếu tố gây ra khó khăn cho các phần mềm diệt mã độc để phát hiện được chúng. Kỹ thuật generic decryption có thể là những cứu cánh cho vấn đề này bởi vì dựa vào vòng lặp giải mã riêng của mã độc để giải mã phần thân của nó. Có một nhận xét là các mã độc cho dù mã hóa thế nào thì chúng đều có chung những hành vi để thực hiện cùng một mục đích. Một khi được giải mã, phần thân của mã độc có thể được phát hiện bằng cách sử dụng phương pháp dò quét thông thường, điều này xác định chính xác khả năng nhận biết các mã độc đa hình, kỹ thuật này sử dụng các kinh nghiệm để xác minh khi một mã độc mã hóa chính nó. Ví dụ như một virus đang cố gắng thực thi một lệnh nằm thường chú trong vùng nhớ ở lần sửa đổi trước (ví dụ giải mã). Một cái dấu hiệu khác là sự thay đổi về kích thước của một quá trình giải mã mặc dù khoảng thay đổi này sẽ tùy vào kiến trúc. Trên nền tảng Intel x86, 24 byte hoặc nhiều hơn bị sửa đổi trên bộ nhớ là một dấu hiệu tiềm năng của một quá trình giải mã đã điễn ra. Bên cạnh kinh nghiệm kỹ thuật dự trên mô phỏng có thể quét bộ nhớ định kỳ trong khoảng thời gian mô phỏng để phát hiện những dấu hiệu bất thường.

CHƯƠNG 2: PHẦN MỀM MÃ NGUỒN MỞ CLAMAV

2.1. Tổng quan

ClamAV là một bộ công cụ chống virus mã nguồn mở cho UNIX, được thiết kế một cách đặc biệt cho việc quét thư điện tử trên gateway. Nó cung cấp một vài tiên ích bao gồm một daemon đa luồng linh động, một bộ quét dòng lệnh và một công cụ tự động cập nhật cơ sở dữ liệu. ClamAV có chứa cơ sở dữ liệu lớn định nghĩa về các loại virus và được cập nhật hàng ngày thông qua kết nối Internet.

Cốt lõi của ClamAV là một thư viện chia sẻ các danh sách: mã nguồn virus, malware, toolkit, rootkit,… cho phép ta dễ dàng nhận biết và phân biệt các nguy hại đang tồn tại trên máy thông qua các danh sách trong cơ sở dữ liệu.

2.2. Tính năng

- Hỗ trợ quét theo dòng lệnh
- Quét theo lịch đặt sẵn.
- Quét nhanh, daemon đa luồng với sự hỗ trợ cho việc quét truy cập
- Sử dụng giao diện milter cho sendmail
- Nâng cao cơ sở dữ liệu cập nhật với sự hỗ trợ các bản cập và chữ ký số
- Truy cập quét (Linux ® và FreeBSD ®)
- Cơ sở dữ liệu virus được cập nhật nhiều lần trong ngày
- Tích hợp hỗ trợ cho các định dạng lưu trữ khác nhau, bao gồm cả Zip, RAR, TAR, GZIP, BZIP2, OLE2, nội các, CHM, BinHex, SIS và các định dạng khác
- Tích hợp hỗ trợ cho thực thi ELF và các tập tin thực thi Portable nén với UPX, FSG, NsPack, wwpack32, MEW, Upack và obfuscated với SUE, Y0da Cryptor và các định dạng khác
- Tích hợp hỗ trợ cho các định dạng tài liệu phổ biến bao gồm MS Office và MacOffice hình ảnh, HTML, RTF và PDF.

Bên cạnh những tính năng ưu điểm như trên, ClamAV còn có một hạn chế: ClamAV không hỗ trợ cơ chế bảo vệ thời gian thực nên khi nào chúng ta có nghi ngờ về sự xuất hiện của virus, malware, toolkit,… thì có thể khởi động ClamAV lên để quét. Hoặc có thể dựa vào cơ chế đặt lịch quét mà ClamAV đã hỗ trợ trên.

2.3. Cài đặt

2.3.1. Yêu cầu

Như đã trình bày ở trên, ClamAV được thiết kế đặc biệt cho các hệ thống UNIX, LINUX, vì vậy những thành phần sau được yêu cầu để biên dịch ClamAV dưới UNIX:

- Các gói zlib và zlib-devel
- Openssl phiên bản 0.9.8 hoặc cao hơn và gói libssl-devel
- Trình biên dịch gcc
- GNU make (gmake)

Những gói dưới đây là tùy chọn tuy nhiên được đề nghị sử dụng:

- bzip2 và bzip2-devel
- libxml2 và libxml2-dev
- check unit testing framework

Các gói được yêu cầu cho việc hỗ trợ bytecode JIT:

- Trình biên dịch GCC C và C++
- OSX Xcode phiên bản trước 5.0 sử dụng một trình biên dịch g++ frontend cái mà không tương thích với ClamAV JIT. Có thể biên dịch ClamAV JIT với clang++ hoặc biên dịch ClamAV không có JIT.
- Một CPU hỗ trợ cho JIT: X86, X86-64, PowerPC, PowerPC64
- Các gói cần cho quá trình JIT unit test
- GNU make
- Python phiên bản 2.5.4 hoặc mới hơn

2.3.2. Cài đặt trên shell account

Để cài đặt ClamAV trên một tài khoản *shell* không có đặc quyền bạn không cần tạo bất kỳ người dùng hoặc nhóm nào. Giả sử thư mục *home* của bạn là */home/gary* bạn nên xây dựng nó như sau:

```
$ ./configure --prefix=/home/gary/clamav --disable-clamav
$ make; make install
```

Để kiểm tra sự thực thi cài đặt:

```
$ ~/clamav/bin/freshclam
$ ~/clamav/bin/clamscan ~
```

2.3.3. Thêm nhóm và người dùng mới

Nếu đang cài đặt cho lần sử dụng đầu tiên, bạn phải thêm một nhóm và người dùng mới tới hệ thống:

```
# groupadd clamav
# useradd -g clamav -s /bin/false -c "Clam AntiVirus" clamav
```

Tham khảo tài liệu hướng dẫn sử dụng nếu hệ điều hành của bạn không có tiện ích *groupadd* và *useradd*. Đừng quên khóa truy cập tới tài khoản.

2.3.4. Biên dịch gói cơ sở

Ngay khi đã tạo nhóm và người dùng ClamAV, vui lòng trích rút tệp nén:

```
$ zcat clamav-x.yz.tar.gz | tar xvf -
$ cd clamav-x.yz
```

Giả sử bạn muốn cài đặt các file cấu hình trong */etc*, cấu hình và xây dựng phần mềm như sau:

```
$ ./configure --sysconfdir=/etc
$ make
$ su -c "make install"
```

Trong bước cuối cùng phần mềm được cài đặt vào thư mục */usr/local* và các file cấu hình trong thư mục */etc*. Chú ý đừng bao giờ cho phép các bit SUID và SGID trong chuỗi nhị phân của ClamAV

2.3.5. Biên dịch với clamav-milter

libmilter và các file phát triển của nó được yêu cầu, để cho phép clamav-milter, cấu hình ClamAV với

```
$ ./configure --enable-milter
```

2.3.6. Sử dụng hệ thống LLVM

Một vài vấn đề đã được báo cáo khi biên dịch ClamAV gắn liền LLVM với những bản phát hành trình biên dịch C++ gần đây. Những vấn đề này có thể được tránh bằng việc cài đặt và sử dụng một thư viện hệ thống LLVM bên ngoài. Để cấu hình ClamAV để sử dụng LLVM cái mà đã được cài đặt như một thư viện hệ thống thay vì LLVM JIT gắn liền nó, sử dụng như sau:

```
$ ./configure --with-system-llvm=/myllvm/bin/llvm-config
$ make
$ sudo make install
```

Tham số *--with-system-llvm* là tùy chọn, chỉ ra tên đường dẫn của tiện ích cấu hình (llvm-config). Khi không có tham số *--with-system-llvm*. *./configure* sẽ tìm kiếm LLVM trong thư mục */usr/local/* và sau đó là */usr*.

Các phiên bản được khuyến nghị của LLVM là 3.2, 3.3, 3.4 và 3.5. Một vài trường hợp cài đặt đã chỉ ra những vấn đề khi sử dụng các phiên bản LLVM cũ hơn. Các phiên bản LLVM mới hơn 3.5 hiện thời không được hỗ trợ trong ClamAV

2.3.7. Tiến hành các kiểm tra đơn vị

ClamAV bao gồm các kiểm tra đơn vị cái mà cho phép kiểm tra các tệp tin nhị phân đã biên dịch có làm việc đúng trên nền tảng của bạn.

Bước đầu tiên là sử dụng trình quản lý gói của hệ điều hành để cài đặt gói check. Nếu hệ điều hành không có gói đó, bạn có thể tải về từ *http://check.sourceforge.net/*, xây dựng nó và cài đặt.

Để trợ giúp kịch bản cấu hình của ClamAV xác định *check*, bạn nên cài đặt *pkg-config*, cách sử dụng ưa thích hơn là dùng trình quản lý gói hoặc từ *http://pkg-config.freedesktop.org*.

Cách tiến hành được khuyến nghị để chạy các kiểm thử đơn vị là như sau, cái này bảo đảm rằng bạn sẽ nhận về một lỗi nếu *unit-tests* không thể được xây dựng:

```
$ ./configure --enable-check
$ make
$ make check
```

Khi *make check được* hoàn thành, sẽ nhận một thông điệp như sau:

```
All 8 tests passed
```

Nếu một *unit test* thất bại, bạn sẽ nhận một thông điệp giống như sau:

```
1 of 8 tests failed
Please report to http://bugs.clamav.net/
```

Nếu các kiểm tra đơn vị bị vô hiệu hóa, sẽ nhận thông điệp như sau:

```
*** Unit tests disabled in this build
*** Use ./configure --enable-check to enable them
SKIP: check_clamav
PASS: check_clamd.sh
PASS: check_freshclam.sh
PASS: check_sigtool.sh
PASS: check_clamscan.sh
=====================
All 4 tests passed
```

(1 tests were not run)

Chạy *./configure --enable-check* sẽ biết nguyên nhân tại sao.

2.3.8. Tải về cơ sở dữ liệu mã độc mới nhất của ClamAV

Trước khi bạn có thể chạy ClamAV trong chế độ daemon (clamd), 'clamdscan', hoặc 'clamscan', đây là trình quét virus dòng lệnh của ClamAV, bạn phải có các file cơ sở dữ liệu virus của ClamAV (.cvd) được cài đặt ở vị trí thích hợp trên hệ thống của mình. Vị trí mặc định cho những file cơ sở dữ liệu này là */usr/local/share/clamav*.

Dưới đây là danh sách những file cơ sở dữ liệu virus sẵn có của ClamAV:

- bytecode.cvd (dấu hiệu để phát hiện bytecode)
- main.cvd (file cơ sở dữ liệu virus chính của ClamAV)
- daily.cvd (file cập nhật cơ sở dữ liệu hàng ngày)
- safebrowsing.cvd (dấu hiệu virus cho duyệt web an toàn)

Những file này có thể được tải qua HTTP từ website chính của ClamAV hoặc qua tiện ích *'freshclam'*. Sử dụng *'freshclam'* là phương pháp được ưa thích để giữ cho cơ sở dữ liệu được cập nhật không cần can thiệp thủ công.

2.4. Cách sử dụng

2.4.1. clamd

Clamd là một daemon đa luồng cái mà sử dụng libclamav để quét virus. Nó có thể làm việc trong cả hai chế độ:

- Unix socket
- TCP socket

Daemon này được cấu hình hoàn toàn trong file clamd.conf. clamd nhận ra những lệnh sau: PING, VERSION, RELOAD, SHUTDOWN, SCAN file/directory, RAWSCAN, CONTSCAN, MULTISCAN, ALLMATCHSCAN, INSTREAM…

2.4.2. clamdscan

Clamdscan là một client đơn giản. Trong nhiều trường hợp bạn có thể sử dụng nó như một sự thay thế clamscan tuy nhiên bạn phải nhớ rằng:

- Nó chỉ phụ thuộc vào clamd
- Mặc dầu nó chấp nhận những tùy chọn dòng lệnh giống nhau như clamscan, hầu hết chúng bị lờ đi bởi vì chúng phải được cho phép trực tiếp trong clamd, tức là clamd.conf

- Trong chế độ TCP các file được quét phải có thể được truy cập từ clamd, nếu bạn cho phép Local-Socket in clamd.conf sau đó clamdscan sẽ thử khắc phục giới hạn này bằng việc sử dụng FILDES

2.4.3. Quét lúc truy cập

Có một luồng đặc biệt trong *clamd* mà thực thi quét lúc truy cập dưới Linux và chia sẻ cơ sở dữ liệu virus với daemon này. Mặc định, luồng này sẽ chỉ thông báo cho bạn khi những mối đe dọa tiềm tàng được phát hiện. Nếu bạn bật chế độ phòng ngừa (prevention) qua *clamd.conf* sau đó bạn phải theo một vài quy tắc quan trọng khi sử dụng nó:

- Luôn luôn tạm dừng daemon một cách sạch sẽ, sử dụng lệnh SHUTDOWN hoặc tín hiệu SIGTERM
- Đừng bao giờ bảo vệ cái thư mục mà phần mềm quét mail sử dụng để mở tệp đính kèm. Truy cập tới tất cả những file nhiễm độc sẽ tự động bị khóa và trình quét sẽ không thể phát hiện bất kỳ virus nào nữa.
- Xem toàn bộ hệ thống file sử dụng tùy chọn OnAccessMountPath trong clamd.conf
- Sử dụng trình quét On-Access để xem một hệ thống file ảo sẽ dẫn đến những tác động không được xác định

2.4.4. clamdtop

clamdtop là một công cụ để giám sát một hoặc nhiều bản sao của clamd. Nó chỉ ra các công việc trong hàng đợi của clamd, sử dụng bộ nhớ và thông tin về cơ sở dữ liệu được tải. bạn có thể xác đinh trên dòng lệnh cái mà clamd sẽ kết nối tới. Mặc định nó sẽ thử kết nối tới clamd cục bộ như được định nghĩa trong clamd.conf.

2.4.5. clamscan

clamscan la trình quét virus dòng lệnh của ClamAV. Nó có thể được sử dụng để quét các file hoặc thư mục. Để clamscan làm việc đúng, các file cơ sở dữ liệu virus phải được cài đặt trên hệ thống mà bạn đang sử dụng clamscan.

Cách sử dụng chung của clamscan là: clamscan [option] [file/directory/-]

2.4.6. clamBC

clamBC là công cụ kiểm tra bytecode của ClamAV. Nó có thể sử dụng để test các file chứa đựng bytecode.

2.4.7. freshclam

freshclam là một công cụ cập nhật cơ sở dữ liệu virus của ClamAV và đọc thông tin cấu hình từ file 'freshclam.conf' (cái này có thể bị ghi đè bởi các tùy chọn dòng lệnh). Hành vi mặc định của freshclam là thử cập nhật cơ sở dữ liệu cái mà được ghép cặp với cdiffs đã được tải, những cơ sở dữ liệu tiềm ẩn khả năng hỏng không được cập nhật và tự động được thay thế sau một vài lần thử thất bại trừ khi được chỉ rõ.

Đây là một cách sử dụng đơn giản bao gồm cdiffs:

```
$ freshclam
ClamAV update process started at Mon Oct 7 08:15:10 2013
main.cld is up to date (version: 55, sigs: 2424225, f-level: 60, builder: neo)
Downloading daily-17945.cdiff [100%]
Downloading daily-17946.cdiff [100%]
Downloading daily-17947.cdiff [100%]
daily.cld updated (version: 17947, sigs: 406951, f-level: 63, builder: neo)
Downloading bytecode-227.cdiff [100%]
Downloading bytecode-228.cdiff [100%]
bytecode.cld updated (version: 228, sigs: 43, f-level: 63, builder: neo)
Database updated (2831219 signatures) from database.clamav.net (IP: 64.6.100.177)
```

2.4.8. clamconf

clamconf là tiện ích cấu hình ClamAV. Nó được sử dụng để hiển thị giá trị của các tùy chọn trong ClamAV, nó đưa ra nội dụng của clamd.conf (hoặc chỉ ra nếu nó không được cấu hình đúng), nội dung của freshcam.conf, và hiển thị thông tin về các thiết lập phần mềm, cơ sở dữ liệu, thông tin phiên bản.

```
$ clamconf
Checking configuration files in /etc/clamav
Config file: clamd.conf
-----------------------
ERROR: Please edit the example config file /etc/clamav/clamd.conf
Config file: freshclam.conf
---------------------------
ERROR: Please edit the example config file /etc/clamav/freshclam.conf
clamav-milter.conf not found
Software settings
-----------------
Version: 0.98.2
Optional features supported: MEMPOOL IPv6 AUTOIT_EA06 BZIP2 RAR JIT
Database information
--------------------
Database directory: /xclam/gcc/release/share/clamav
WARNING: freshclam.conf and clamd.conf point to different database directories
print_dbs: Can't open directory /xclam/gcc/release/share/clamav
```

```
Platform information
--------------------
uname: Linux 3.5.0-44-generic #67~precise1-Ubuntu SMP Wed Nov 13 16:20:03 UTC 2013 i686
OS: linux-gnu, ARCH: i386, CPU: i686
Full OS version: Ubuntu 12.04.3 LTS
zlib version: 1.2.3.4 (1.2.3.4), compile flags: 55
Triple: i386-pc-linux-gnu
CPU: i686, Little-endian
platform id: 0x0a114d4d0404060401040604
Build information
-----------------
GNU C: 4.6.4 (4.6.4)
GNU C++: 4.6.4 (4.6.4)
CPPFLAGS:
CFLAGS: -g -O0 -D_LARGEFILE_SOURCE -D_LARGEFILE64_SOURCE
CXXFLAGS:
LDFLAGS:
Configure: '--prefix=/xclam/gcc/release/' '--disable-clamav' '--enable-debug' 'CFLAGS=-g -O0'
sizeof(void*) = 4
Engine flevel: 77, dconf: 77
```

2.4.9. Định dạng đầu ra

2.4.9.1. clamscan

clamscan đưa tất cả các thông điệp chương trình bình thường tới stdout và các lỗi,cảnh báo tới stderr. Có thể sử dụng tùy chọn –stdout để chuyển hướng tất cả thông điệp tới stdout. Các thông điệp lỗi và cảnh báo từ libclamav luôn luôn được đưa ra stderr. Một đầu ra điển hình nhìn giống như sau:

```
/tmp/test/removal-tool.exe: Worm.Sober FOUND
/tmp/test/md5.o: OK
/tmp/test/blob.c: OK
/tmp/test/message.c: OK
/tmp/test/error.hta: VBS.Inor.D FOUND
```

Khi một virus được tìm thấy tên của nó được in giữa tên của file đó và chuỗi FOUND. Trong trường hợp các file nén, trình quét phụ thuộc vào libclamav và chỉ in virus đầu tiền được tìm thấy trong file đó

```
$ clamscan malware.zip
malware.zip: Worm.Mydoom.U FOUND
```

Khi sử dụng cờ --allmatch (-z), clamscan có thể in ra nhiều dòng 'virus FOUND'.

2.4.9.2. clamd

Định dạng đầu ra của clamd là rất giống với clamscan

```
$ telnet localhost 3310
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
SCAN /home/zolw/test
/home/zolw/test/clam.exe: ClamAV-Test-File FOUND
Connection closed by foreign host.
```

Trong chế độ SCAN nó đóng kết nối khi virus đầu tiên được phát hiện

```
SCAN /home/zolw/test/clam.zip
/home/zolw/test/clam.zip: ClamAV-Test-File FOUND
```

CONTSCAN và MULTISCAN không dừng quét khi một virus bị phát hiện. các thông điệp lỗi được in trong định dạng dưới đây

```
SCAN /no/such/file
/no/such/file: Can't stat() the file. ERROR
```

2.5. LibClamAV

Libclamav cung cấp một phương pháp dễ dàng và hiệu quả để thêm một sự ngăn chặn virus vào trong phần mềm. thư viện này là an toàn và trong suốt, có thể phát hiện và quét trong các tệp tin nén, thư điện tử, tệp tin MS Office, file thực thi và những đinh dạng đặc biệt khác.

2.5.1. Bản quyền

Libclamav đươc cấp bản quyền theo giấy phép GNU GPL v2, điều này có nghĩa là chúng ta không được phép xây dựng các phần mềm nguồn đóng, thương mại dựa vào nó. Tất cả các phần mềm sử dụng libclamav phải tuân theo GPL.

2.5.2. Các hàm API

2.5.2.1. Header file

Mỗi chương trình sử dụng libclamav phải thêm vào tệp tiêu đề (header file) clamav.h:

```
#include <clamav.h>
```

2.5.2.2. Bước khởi tạo

Trước khi sử dụng libclamav, chúng ta nên gọi *cl_init()* để khởi tạo nó, tiếp theo để tạo một engine quét mới gọi *cl_engine_new()*. Để giải phóng các tài nguyên được cấp phát bởi engine sử dụng *cl_engine_free()*. Nguyên mẫu các hàm này:

```
int cl_init(unsigned int options);
```

```
struct cl_engine *cl_engine_new(void);
int cl_engine_free(struct cl_engine *engine);
```

cl_init() và *cl_engine_free()* trả về *CL_SUCCESS* khi thành công hoặc một mã khác khi lỗi. cl_engine_new() trả về một con trỏ hoặc NULL nếu không đủ bộ nhớ.

2.5.2.3. Tải về cơ sở dữ liệu virus

Tập lệnh dưới đây cung cấp một giao diện để tải về cơ sở dữ liệu virus:

```
const char *cl_retdbdir(void);
int cl_load(const char *path, struct cl_engine *engine,
unsigned int *signo, unsigned int options);
```

cl_retdbdir(void) trả về đường dẫn mặc định tới thư mục chứa cơ sở dữ liệu ClamAV. *cl_load()* tải về một file cơ sở dữ liệu đơn hoặc tất cả từ một thư mục cho trước (khi *path* chỉ tới một thư mục). tham số thứ hai được sử dụng để chuyển con trỏ tới engine cái mà sẽ được cấp phát trước với *cl_engine_new()*. Một vài dấu hiệu đã được tải sẽ được thêm tới *signo*. Tham số cuối cùng có thể truyền những cờ sau:

- CL_DB_STDOPT: đây là một biệt hiêu cho một tập các tùy chọn quét được đề nghị
- CL_DB_PHISHING: tải về các dấu hiệu lừa đảo (phishing)
- CL_DB_PHISHING_URLS: khởi tạo môđun phát hiện phishing và tải các tệp .wdb và .pdb
- CL_DB_PUA: tải về dấu hiệu cho các ứng dụng không mong muốn tiềm ẩn (PUA)
- CL_DB_OFFICIAL_ONLY: chỉ tải về các dấu hiệu chính thức từ cơ sở dữ liệu được ký số.
- CL_DB_BYTECODE: tải về bytecode

2.5.2.4. Xử lý lỗi

Sử dụng *cl_strerror()* để chuyển đổi mã lỗi thành các thông điệp có thể đọc được, hàm này trả về một chuỗi được cấp phát tĩnh:

```
if(ret != CL_SUCCESS) {
printf("cl_load() error: %s\n", cl_strerror(ret));
cl_engine_free(engine);
return 1;
}
```

2.5.2.5. Cấu trúc engine

Khi tất cả cơ sở dữ liệu đã được tải, tiếp theo cần chuẩn bị engine phát hiện bằng cách *gọi cl_engine_compile()*. Trong trường hợp thất bại chúng ta vẫn cần giải phóng bộ nhớ được cấp cho engine với *cl_engine_free():*

```
int cl_engine_compile(struct cl_engine *engine);
```

2.5.2.6. Một vài lưu ý

Khi bạn tạo một engine mới với *cl_engine_new()*, nó sẽ có tất cả các giá trị mặc định được khuyến nghị. Chúng ta có thể kiểm tra và sửa đổi các giá trị này sử dụng các hàm dưới đây:

```
int cl_engine_set_num(struct cl_engine *engine,
enum cl_engine_field field, long long num);
long long cl_engine_get_num(const struct cl_engine *engine,
enum cl_engine_field field, int *err);
int cl_engine_set_str(struct cl_engine *engine,
enum cl_engine_field field, const char *str);
const char *cl_engine_get_str(const struct cl_engine *engine,
enum cl_engine_field field, int *err);
```

Chú ý là không nên sửa đổi các giá trị mặc định trừ phi chúng ta biết những gì chúng ta đang làm.

2.5.2.7. Kiểm tra cơ sở dữ liệu

Việc giữ cho cơ sở dữ liệu được cập nhật là rất quan trọng. Có thể theo dõi những thay đổi trong cơ sở dữ liệu với họ hàm *cl_stat..()*.

```
int cl_statinidir(const char *dirname, struct cl_stat *dbstat);
int cl_statchkdir(const struct cl_stat *dbstat);
int cl_statfree(struct cl_stat *dbstat);
```

2.5.2.8. Các hàm quét dữ liệu

Có thể quét một file hoặc bộ mô tả sử dụng:

```
int cl_scanfile(const char *filename, const char **virname,
unsigned long int *scanned, const struct cl_engine *engine,
unsigned int options);
int cl_scandesc(int desc, const char **virname, unsigned
long int *scanned, const struct cl_engine *engine,
unsigned int options);
```

2.5.2.9. clamav-config

Sử dụng *clamav-config* để kiểm tra thông tin biên dịch cho libclamav.

```
$ clamav-config --libs
-L/usr/local/lib -lz -lbz2 -lgmp -lpthread
```

```
$ clamav-config --cflags
-I/usr/local/include -g -O2
```

2.5.3. Định dạng CVD

CVD (ClamAV virus database) là một tệp tarball được ký số chứa đựng một hoặc nhiều cơ sở dữ liệu. phần đầu là một chuỗi dài 512 byte với các trường được phân tách bằng dấu hai chấm:

```
ClamAV-VDB:build time:version:number of signatures:functionality
level required:MD5 checksum:digital signature:builder name:build time (sec)
```

sigtool –info hiển thị thông tin chi tiết về file CVD:

```
$ sigtool -i daily.cvd
File: daily.cvd
Build time: 10 Mar 2008 10:45 +0000
Version: 6191
Signatures: 59084
Functionality level: 26
Builder: ccordes
MD5: 6e6e29dae36b4b7315932c921e568330
Digital signature: zz9irc9irupR3z7yX6J+OR6XdFPUat4HIM9ERn3kAcOWpcMFxq
Fs4toG5WJsHda0Jj92IUusZ7wAgYjpai1Nr+jFfXHsJxv0dBkS5/XWMntj0T1ctNgqmiF
+RLU6V0VeTl4Oej3Aya0cVpd9K4XXevEO2eTTvzWNCAq0ZzWNdjc
```

CHƯƠNG 3: XÂY DỰNG CƠ SỞ DỮ LIỆU MẪU MÃ ĐỘC

Trong chương này chúng tôi giới thiệu về việc tạo cơ sở dữ liệu mẫu mã độc cho các phần mềm chống virus. Hiện nay có rất ít thông tin liên quan đến việc tạo cơ sở dữ liệu mẫu mã độc, chỉ có một số ít tài liệu về cách tạo cơ sở dữ liệu với ClamAV hoặc sử dụng các công cụ khác. Vì vậy trong chương này chúng tôi sẽ giới thiệu ba kỹ thuật chính của việc phát hiện mã độc dựa trên dấu hiệu (Signature) và tập trung chủ yếu vào việc tạo cơ sở dữ liệu cho những tệp tin thực thi trên hệ điều hành Microsoft Windows. Các dấu hiệu phổ biến nhất là mã băm (Hash), mã byte (Byte-signature) và dựa vào kinh nghiệm (Heuristics)

3.1. Cơ sở dữ liệu mã độc dựa trên mã băm

Kiểu dấu hiệu đơn giản và cơ bản nhất là một giá trị băm. Giá trị băm được tạo ra bởi một hàm băm, là một thủ tục hoặc hàm toán học chuyển đổi một lượng lớn dữ liệu thành một giá trị đơn. Hàm băm được sử dụng phổ biến nhất là MD5 và SHA-1 [1,43]. Các hàm băm này cực kỳ chính xác, nếu có khối dữ liệu đã được băm và sau đó cùng một khối dữ liệu như vậy có một byte thay đổi và băm lại thì các giá trị băm sẽ khác nhau. Trong ví dụ bên dưới hai chuỗi “HelloWorld!” và “HelloWorld” chỉ khác nhau một ký tự “!” nhưng giá trị băm hoàn toàn khác nhau.

MD5 của chuỗi "HelloWorld!" = "06e0e6637d27b2622ab52022db713ce2"

MD5 của chuỗi "HelloWorld" = "68e109f0f40ca72a15e05cc22786f8e6"

Cơ sở dữ liệu mã độc dựa trên MD5 có thể được tạo ra bằng cách sử dụng ClamAV. Công cụ Yara không hỗ trợ băm tệp tin. ClamAV yêu cầu hai thuộc tính để tạo dấu hiệu mã băm MD5. Đầu tiên là kích thước tệp tin theo byte và thứ hai là mã băm MD5. ClamAV đi kèm với một công cụ được gọi là sigtool có thể được sử dụng để tạo dấu hiệu mã MD5. Sigtool có thể được tìm thấy trong thư mục "bin" trong thư mục cài đặt ClamAV. Bên dưới là ví dụ về cách sử dụng sigtool cho tệp tin one.txt

C:\EXP\clamwin\bin>sigtool.exe --md5 one.txt 08bc3de154cadb33affd7d4433ebc31e:16:C:\EXP\one.txt

Các mẫu dấu hiệu của ClamAV được phân cách bởi dấu hai chấm ':'. Phần thứ nhất là mã MD5 (08bc3de154cadb33affd7d4433ebc31e), thứ hai là kích thước và phần cuối cùng là vị trí tệp tin hoặc đầu ra. Đầu ra thường là tên mã độc hoặc một cái gì đó đặc trưng cho tệp tin. Dấu hai chấm không được sử dụng trong đầu ra bởi vì chúng được coi là các ký tự đặc biệt trong cơ sở dữ liệu của ClamAV. Cơ sở dữ liệu mẫu MD5 cần được lưu trong một tệp có đuôi .hdb. Đầu ra từ sigtool có thể được dẫn đến (>) một tệp tin gọi là shredder.hdb. Điều này làm cho nó dễ dàng tạo các dấu hiệu mã MD5 với tất cả các tệp trong một thư mục sử dụng một vòng lặp đơn giản trong

cửa sổ dòng lệnh [37]. Thêm nữa điều này cũng giúp tạo các dữ liệu mẫu mã độc ClamAV từ kết quả của Virustotal mà không cần có tệp tin đó vì mã MD5 và kích thước tệp đã được cung cấp trong trình phân tích của Virustotal.

Giá trị băm rất hữu ích nếu mẫu mã độc là tĩnh nhưng nếu một byte trong tệp thực thi bị thay đổi, mẫu mã băm sẽ bị phá vỡ. Trong một số trường hợp mã nguồn của tệp thực thi không bao giờ thay đổi nhưng dữ liệu mà tệp thực thi thực hiện lại thay đổi. Một ví dụ về mã không thay đổi là tệp tin thực thi được tạo ra bởi bộ GUI Remote Access Trojan. Phần mã sẽ giống nhau nhưng nếu một người dùng sử dụng địa chỉ 192.168.0.1 và một người khác đã sử dụng 192.160.2 thì các phần dữ liệu sẽ khác nhau. Trong ví dụ bên dưới, mã băm từng phần có thể được sử dụng để nhắm tới khối dữ liệu phần mã nguồn và tạo một dấu hiệu dựa trên mã băm.

PESectionSize:MD5:MalwareName

.code:0970e94b5ea5bbb91b9cf963c47b4895:MalwareName

Các mẫu dấu hiệu nhận biết này cần được lưu trong một tệp có đuôi .mdb để có định dạng phù hợp với ClamAV. Như đã nêu bên trên, dấu hai chấm được coi là ký tự đặc biệt để tách dữ liệu trong cơ sở dữ liệu mẫu của ClamAV. Phần đầu tiên là tên phần PE (.code), phần thứ hai là mã băm MD5 của phần PE này (0970e94b5ea5bbb91b9cf963c47b4895) và cuối cùng là đầu ra hoặc tên của mã độc (MalwareName) [37]). Mã băm cũng có thể được sử dụng để xác định các tệp tin đã bị sửa đổi. Thuật ngữ Fuzzy Hashing [24,44,17] là một kỹ thuật về cơ bản vẫn là nhận dạng chương trình mã độc qua mã băm nhưng đã được bổ sung thêm các phân tích và tính toán để từ một mã băm của mã độc có thể nhận ra các mã băm "họ hàng", nâng cao khả năng phát hiện chương trình mã độc. Kỹ thuật này có thể được sử dụng để xác định các tệp tin chứa các dữ liệu đã bị xóa, sửa đổi hoặc chứa dữ liệu mới đã được chèn thêm. Dưới đây là một ví dụ so sánh hai tệp tin thực thi với sự khác biệt một byte.

C:\EXP>md5sum * 0bcc4aab14639664f63865cc4959fb53 second-edit.exe

1d4720491fb7d551b38ab4299a2e6787 second.exe

C:\EXP>ssdeep *

ssdeep,1,1--blocksize:hash:hash,filename

384:cKxNLII2Fz67gMvzoH/WjCpu6JBnPLl7jz:cKnH26xCX0,"C:\EXP\second-edit.exe"

384:BKxNLII2Fz67gMvzoH/WjCpu6JBnPLl7jz:BKnH26xCX0,"C:\EXP\second.exe"

C:\EXP\second.exe matches C:\EXP\second-edit.exe(99)

Con số ở cuối dòng trên (99) là một độ đo sự tương đồng, con số này càng lớn những tệp tin này càng tương đồng nhau.

Fuzzy hashing yêu cầu tất cả các giá trị băm được lưu trữ trong một tệp tin văn bản ASCII và sau đó mỗi tệp tin đã băm cần phải được kết hợp lại với tệp tin văn bản mã băm ASCII. Kiểu quét này đòi hỏi nhiều tính toán hơn so với các dạng quét khác do phải băm và so sánh tất cả các giá trị băm. Một vấn đề với Fuzzy hashing (cũng như với tất cả các hàm băm) là những sửa đổi dường như rất nhỏ của tác giả hay lập trình viên cũng có thể thay đổi đáng kể giá trị đã được tính toán. Tệp tin thực thi second.exe bên trên đã được sửa đổi bằng cách thay đổi một số chuỗi đầu ra và biên dịch lại thành second2.exe, không đoạn mã nguồn nào bị thay đổi. Tiếp tục sử dụng công cụ ssdeep để so sánh hai tệp này như trong ví dụ dưới đây.

```
C:\EXP\ssdeep *
ssdeep,1.1--blocksize:hash:hash,filename
384:BKxNLII2Fz67gMvzoH/WjCpu6JBnPLl7jz:BKnH26xCX0,"C:\EXP\second.exe"
384:5KTNJIv2w05gDvzoHoRjCpP6JqnuLl7jz:5KOCXrQQ0,"C:EXP\second2.exe"
C:\EXP\string>ssdeep
ssdeep: No input files
C:\EXP\string>ssdeep -d *
C:\EXP\string>ssdeep *
ssdeep,1.1--blocksize:hash:hash,filename
384:BKxNLII2Fz67gMvzoH/WjCpu6JBnPLl7jz:BKnH26xCX0,"C:\EXP\string\second.exe"
384:5KTNJIv2w05gDvzoHoRjCpP6JqnuLl7jz:5KOCXrQQ0,"C:\EXP\string\second2.exe"
C:\EXP\>ssdeep second.exe > o.txt
C:\EXP\>ssdeep second2.exe > o2.txt
C:\EXP\>ssdeep -d out*
C:\EXP\o2.txt matches C:\EXP\o.txt (50)
```

Vì tất cả các mã băm cần được lưu trong một tệp tin văn bản sau đó được so sánh với nhau nên ssdeep có thể đòi hỏi chi phí tính toán cao [16].

3.2. Cơ sở dữ liệu mẫu dựa trên mã byte

Mã byte (Byte signature) là một dấu hiệu dựa trên một chuỗi các byte có mặt trong một tệp tin hoặc luồng dữ liệu. Mã byte là một kỹ thuật phát hiện rất phổ biến và đã được sử dụng trong chương trình quét virus đầu tiên. Tính hữu dụng của kỹ thuật này là do độ chính xác mà nó cung cấp cho việc phát hiện một chuỗi các byte.

Chuỗi byte được chọn bởi vì nó tồn tại trong nhiều biến thể của mã độc từ cùng một họ. Mã byte có thể là bất kỳ loại dữ liệu nào như một đoạn mã nguồn hoặc dữ liệu được chứa bên trong một luồng dữ liệu hoặc một tệp thực thi, tệp PDF (Portable Document File) hoặc tài liệu Word. Bên dưới là một đoạn mã byte trong định dạng của ClamAV cho tệp second.exe.

```
Seconddotexe:1:77CF2463CC0263AB0185E6
```

Trong đó "Seconddotexe" là đầu ra được hiển thị bởi chương trình quét ClamAV; đây thường là tên của một họ mã độc. Phần thứ hai là dành cho chương trình quét của ClamAV biết được loại tệp tin của tệp tin đã được quét. Giá trị '1' để chương trình quét nhận biết các tệp tin thực thi di động (PE files). Để quét bất kỳ loại tệp nào, cần phải sử dụng giá trị '0'. "77CF2463CC0263AB0185E6" là biểu diễn thập lục phân của đoạn chương trình hợp ngữ dưới đây.

```
start: 0x302C2F  length: 0xC
77     nop
CF 24     call    dword ptr [esi]
63 CC 02    add     esi, 4
63 AB 01    add     ebx, 1
85 E6     jnz     short loc_302F30
```

Trong định dạng của Yara đoạn mã byte được biểu diễn như sau.

```
rule example
{
  strings:
  signature = { 43 77 CF 24 63 CC 02 63 AB 01 85 E6 }
  condition:
  signature
}
```

Định dạng dấu hiệu nhận dạng mã độc của Yara thì khác nhiều so với ClamAV. Phong cách cú pháp tương tự như một cấu trúc của C. Chuỗi đầu tiên xác định rằng chúng ta đang tạo ra một luật của "example". Các "strings" và "condition" là các từ khóa được sử dụng bởi chương trình quét Yara. Các “strings” sẽ xác định các dấu hiệu nhận biết chương trình mã độc. Dấu hiệu nhận biết của Yara có thể là bất kỳ định dạng nào từ chuỗi (strings), hex-byte, regex và một số định dạng khác. Từ khóa “condition” xác định trong những trường hợp nào Yara sẽ cảnh báo về dấu hiệu nhận biết. Có rất nhiều điều kiện có thể xây dựng cho Yara để phát hiện một tệp tin [7].

Binary diffing

Các kỹ thuật phân tích thủ công và tự động có thể được sử dụng để tìm các khối mã có mặt trong các biến thể. Quá trình so sánh nhiều tệp tin thực thi cho các điểm tương đồng được gọi là Binary diffing [24,18,45]. Binary diffing theo cách thủ công bao gồm việc xem xét mã hợp ngữ của nhiều tệp tin và ghi lại các phần mã tồn tại trong các tệp tin. Khi các khối mã được xác định, một sự so sánh liên tiếp có thể được thực hiện để kiểm tra sự tương đồng. Nếu mã có độ tương đồng cao, nó có thể là một ứng cử viên tốt cho cách nhận biết byte-signature. Một cách tiếp cận bán tự động của kỹ thuật Binary diffing bao gồm việc nhóm các tệp tin thành các tệp tin tương đồng nhau, sử dụng một công cụ để dịch mã máy thành mã hợp ngữ (disassembler) và ghi vào một tệp văn bản, sau đó so sánh và phân tích các đoạn mã hợp ngữ tương đồng này. Chúng ta có thể sử dụng công cụ IDA (Interactive Disassembler)[24,40,21] để tạo mã hợp ngữ và sau đó sử dụng KDiff (công cụ mã nguồn mở để so sánh sự khác nhau của các tệp tin) [19] như ví dụ sau.

```
C:\EXP\>dir
    23,503 second.exe
    23,503 second2.exe
C:\EXP\>"C:\ProgramFiles\IDAFree\idag.exe" -B "second.exe"
C:\EXP\>"C:\ProgramFiles\IDAFree\idag.exe" -B "second2.exe"
C:\EXP\>dir
    51,262 second.asm
    23,503 second.exe
   204,956 second.idb
    51,274 second2.asm
    23,503 second2.exe
   204,956 second2.idb
```

Cờ "-B" là để tạo ra một cơ sở dữ liệu cho IDA (.idb) và một đầu ra là tệp mã hợp ngữ (.asm). KDiff là một công cụ tốt để so sánh các tệp tin bởi vì nó không thực hiện một sự so sánh trực tiếp giữa hai tệp tin, nó sẽ thử sắp xếp các đoạn mã bị tách biệt bởi một lượng lớn dữ liệu hoặc mã. Các phần này có thể khác nhau do một chức năng có thể không nằm trong một tệp thực thi hoặc có sự xuất hiện mã hoặc dữ liệu rác. KDiff cũng có thể so sánh đến ba tệp tin cùng một lúc. Các công cụ tự động như Bindiff [24,12], PatchDiff2 [24] và DarunGrim [24] có thể được sử dụng để tìm kiếm sự khác biệt giữa các tệp cơ sở dữ liệu IDA. VxClass [24] là một công cụ khác có thể

được sử dụng để tạo ra các bộ biến thể, so sánh và xác định các phần mã có thể được sử dụng cho sự phát hiện dựa trên dấu hiệu. Quá trình so sánh cũng rất hữu ích cho việc hiển thị các đoạn mã nơi các ký tự đại diện cần phải được sử dụng cho sự phát hiện dựa vào byte-signature. Các ký tự đại diện sẽ cho phép chương trình quét bỏ qua các byte mã không cố định trong nhiều biến thể. Các byte không cố định này có thể bị gây ra bởi việc chèn mã rác, bổ sung hoặc loại bỏ mã hoặc bổ sung hoặc loại bỏ dữ liệu. Một sự thay đổi nhỏ trong bố cục tệp tin có thể phá vỡ quá trình phát hiện dựa vào byte-signature. Các lệnh phổ biến sử dụng độ lệch (offset) dựa trên bố cục tệp là long jmps, lời gọi các hàm con (sub-routine), các API hoặc các lệnh khác như cmp, mov, sub, add…. Cách tốt nhất là khi tạo các signature luôn thêm các ký tự đại diện (wildcard) cho các offset mặc dù chúng vẫn giữ nguyên trong nhiều biến thể. Ví dụ dưới đây cho thấy bố cục tệp tin có thể phá vỡ quá trình phát hiện dựa vào byte-signature. Hai khối mã trong hai tệp three.exe và three2.exe, địa chỉ bắt đầu, chức năng và chiều dài đều giống hệt nhau. Sự khác biệt duy nhất giữa hai khối mã là độ lệch địa chỉ của dword được so sánh với ebx. Sự khác nhau trong bố cục tệp tin là do các chuỗi đầu ra có độ dài khác nhau.

Khối mã trong three.exe

start: 0x412368 length: 0x14

62 EC 6B 23 52 00 cmp ebx, offset dword_52236B

1E 63 2B FF FF FF jnb loc_411236

CB 22 EC 23 00 mov esi, 23EC22h

6E 8B F2 lea edi, [ebp+var_20]

62 EC 6B 23 52 00 1E 63 2B FF FF FF CB 22 EC 23 00 6E 8B F2

Khối mã trong three2.exe

start: 0x412368 length: 0x14

62 EC 73 23 52 00 cmp ebx, offset dword_522373

1E 63 2B FF FF FF jnb loc_411236

CB 22 EC 23 00 mov esi, 23EC22h

6E 8B F2 lea edi, [ebp+var_20]

62 EC 73 23 52 00 1E 63 2B FF FF FF CB 22 EC 23 00 6E 8B F2

Trong định dạng của ClamAV chúng ta có thể biểu diễn cho cả hai tệp này như sau.

Both three dot exe :1:62EC********1E632BFFFFFFCB22EC23006E8BF2

Trong định dạng của Yara chúng ta có thể biểu diễn như sau.

```
rule both_three_dot_exe
{
strings:
signature = { 62 EC ?? ?? ?? ?? 1E 63 2B FF FF FF CB 22 EC 23 00 6E 8B F2 }
condition:
signature
}
```

3.3. Cơ sở dữ liệu mẫu dựa vào kinh nghiệm

Kỹ thuật phát hiện cuối cùng là dựa vào kinh nghiệm (heuristics). Heuristics được sử dụng khi chương trình mã độc hại quá phức tạp đối với mã băm và byte-signature. Heuristics là thuật ngữ chung cho các kỹ thuật khác nhau được sử dụng để phát hiện chương trình độc hại dựa vào hành vi của chúng. Đây là một trong những dạng phát hiện phức tạp nhất. Một công cụ chống virus có thể sử dụng các kỹ thuật phân tích như giả lập môi trường, API hooking, sand-boxing, các bất thường trong tệp tin… để phát hiện mã độc. Mỗi công cụ chống virus sử dụng các thuật toán khác nhau và kỹ thuật độc quyền khác nhau. Một ví dụ đơn giản về việc tạo ra một dấu hiệu nhận biết dựa trên heuristics gồm có một API logger và các luật (rules) dựa trên các API. Hãy bắt đầu với chuỗi "Hello World" của mã độc Poison Ivy (một loại Trojan trên hệ điều hành Microsoft Windows). Chúng ta có thể sử dụng một máy chủ làm ví dụ để tạo một dấu hiệu nhận biết dựa trên luật API (API rule). Khi Poison Ivy được thực thi nó sẽ tạo ra một mutex, thêm một khóa (key) vào registry và sao chép chính nó vào thư mục System32. Mutex mặc định cho Poison Ivy là ")! VoqA.I4". Bằng cách sử dụng chương trình giám sát Kerberos API Monitor chúng ta có thể mô phỏng một API hook (có thể được sử dụng bởi một công cụ chống virus). Một luật API dựa trên heuristics cho Poison Ivy có thể như sau.

Rule A

An API call to RtlMoveMemory with a string of "SOFTWARE\Classes\http\shell\open\commandV"

Rule B

An API call to CreateMutexA with a string of ")!VoqA.I4"

Rule C

An API call to GetSystemDirectory

if (Rule A then Rule B then Rule C)

then

Process = PoisonIvy

Keribos Output

Rule A

second.exe | 00401447 | RtlMoveMemory(0012F458, 0040162F: "SOFTWARE\Classes\http\shell\open\commandV", 00000028) returns: 0012F458

.....

Rule B

second.exe | 0040155D | CreateMutexA(00000000, 00000000, 0012F43B: ")!VoqA.I4") returns: 0000003C

.....

Rule C

second.exe | 004018BF | GetSystemDirectoryA(0012F6F1, 000000FF) returns: 00000013

CHƯƠNG 4: ĐỀ XUẤT GIẢI PHÁP TRÍCH CHỌN ĐẶC TRƯNG CHO BÀI TOÁN PHÁT HIỆN MÃ ĐỘC

4.1. Đặt vấn đề

Chúng ta biết rằng giai đoạn quan trọng nhất của kỹ thuật phát hiện mã độc là xác định rõ việc biểu diễn các tập tin mã độc. Việc biến đổi một tập hợp dữ liệu đầu vào lớn thành tập các đặc trưng được gọi là trích rút đặc trưng. Khi chúng ta có dữ liệu vào dư thừa lúc này chúng ta cần sử dụng các phương pháp trích rút đặc trưng để giảm bớt sự dư thừa này. Điều quan trọng là chúng ta phải có được một phép đo chính xác các đặc trưng mà ảnh hưởng đến việc phân loại đầu vào là độc hay không độc. Quá trình trích rút biến đổi các đặc trưng thành một tập thông tin con có tổ chức và dễ quản lý hơn. Thêm nữa nó cũng làm giảm tập dữ liệu cần xử lý dẫn đến chi phí tính toán cũng giảm theo. Đầu ra của pha trích rút đặc trưng là một véc tơ chứa tần số của các đặc trưng được trích rút. Các đặc trưng được trích rút được chọn sao cho độ chính xác phân loại là cao nhất. Thời gian cần thiết để nhận được các đặc trưng từ tập dữ liệu đầu vào cũng phụ thuộc vào phương pháp trích rút.

Phương pháp trích rút ảnh hưởng tới sự thực thi của hệ thống phát hiện về hiệu suất và sự chính xác. Lần đầu tiên Schultz và các cộng sự [33] đã giới thiệu quan điểm ứng dụng các kỹ thuật học máy trong phát hiện mã độc dựa trên sự biểu diễn các tập tin tương ứng từ tập dữ liệu. Họ đã thực hiện ba phương pháp trích rút, trong khi đó nhiều nhà nghiên cứu đã mở rộng ý tưởng trích rút đặc trưng để cải thiện sự thực thi và độ chính xác của hệ thống. Theo các nghiên cứu đã nói ở trên phương pháp trích rút đặc trưng được miêu tả như sau:

4.2. Một số phương pháp trích rút đặc trưng phổ biến

4.2.1. Đặc trưng byte n-gram

Đặc trưng Byte n-gram là chuỗi n byte được trích xuất từ các mã độc, chúng được sử dụng như dấu hiệu để nhận dạng phần mềm độc hại. Mặc dù kiểu đặc trưng này không cung cấp thông tin có ý nghĩa nhưng nó mang lại độ chính xác cao trong việc phát hiện những mẫu mã độc mới. Tony Abou-Assaleh và các cộng sự [38] đã trích rút đặc trưng byte n-gram từ mã nhị phân của tập tin có n-gram xuất hiện nhiều nhất trong mỗi lớp là L trong tập huấn luyện được chọn để biểu thị đặc tính của lớp. Tony Abou-Assaleh đề xuất một bộ phân loại sử dụng thuật toán K láng giềng gần nhất (KNN). Thí nghiệm của họ đạt độ chính xác 98% trên bộ dữ liệu các tệp tin độc và không độc. R. Moskovitch và các cộng sự sử dụng byte n-grams kết hợp với

opcode n-grams như là các đặc trưng, họ tiến hành thí nghiệm trên hơn 30.000 tập tin và độ chính xác thu được là trên 99%.

4.2.2. Đặc trưng opcode n-gram

Các nghiên cứu trước đây chỉ ra rằng trích rút đặc trưng opcode là hiệu quả hơn cho sự phân loại. Chúng biểu thị những thống kê đa dạng giữa các phần mềm độc hại và hợp pháp. Một vài opcode hiếm khi là sự dự đoán tốt cho các hành vi độc hại. Đầu tiên trong tập dữ liệu tất cả các tập tin thực thi được dịch ngược và opcode được trích rút. Một opcode là một một lệnh trong hợp ngữ, nó miêu tả thao tác được thực thi. Nó là dạng ngắn của mã hoạt động. Một lệnh gồm có một opcode và nhiều toán hạng. Một số thao tác có các toán hạng mà nhờ đó opcode có thể hoạt động, tùy thuộc vào kiến trúc CPU, thanh ghi, giá trị lưu trữ trên bộ nhớ và ngăn xếp...

Moskovitch và các cộng sự tuyên bố rằng chuỗi opcode 2-gram là chuỗi N-gram tốt nhất về sự chính xác trong phân loại. Tuy nhiên, có nhiều chuỗi 2-gram opcode có độ chính xác giảm. Santos và các cộng sự đã sử dụng những chuỗi opcode cho việc phân loại mã độc với các thuật toán phân loại và chọn đặc trưng khác nhau, họ sử dụng những chuỗi opcode 1-gram và 2-gram để phát hiện những biến thể mới của các họ mã độc. Họ sử dụng biểu đồ tần số cho mỗi chuỗi n-gram tính toán tần số tỷ lệ tương tự cho mỗi mẫu mã độc.

4.2.3. Đặc trưng PE

Các đặc trưng này được trích rút từ một phần nào đó của tập tin EXE. Các đặc trưng PE được trích rút bằng phân tích tĩnh sử dụng thông tin cấu trúc của PE. Những đặc trưng đầy ý nghĩa này chỉ ra rằng tập tin này đã được sửa đổi hoặc bị nhiễm độc để thực hiện những hoạt động nguy hiểm. Shafiq và các cộng sự đã đề xuất một phương pháp thời gian thực để phát hiện mã độc dựa trên đặc trưng cấu trúc được khai thác từ PE. Họ đã tiến hành thử nghiệm trên hai tập dữ liệu Malfease and Vxheavens và nhận xét rằng đặc trưng PE có chi phí xử lý thấp.

4.2.4. Đặc trưng chuỗi

Những đặc trưng này dựa trên văn bản trơn đã được mã hóa trong tập tin thực thi như windows, getversion, getstartupinfo, getmodulefilename, messagebox, library, vv. Những chuỗi này là những ký tự có thể in liên tiếp được mã hóa trong các tập tin PE cũng như phi PE. Đặc trưng chuỗi được sử dụng trong thí nghiệm của Schultz và các cộng sự cho độ chính xác là 97.11% khi so với việc sử dụng đặc trưng PE và byte n-grams. Các đặc trưng chuỗi thường không mạnh vì chúng có thể được sửa đổi dễ dàng bất cứ lúc nào. Y. Ye đã đề xuất một hệ thống phát hiện mã độc,

SBMDS, hệ thống này phân loại mã độc sử dụng SVM (Support vector machine) dựa trên đặc trưng chuỗi, nó tốt hơn so với các phần mềm diệt virus đang tồn tại.

4.2.5. Đặc trưng dựa trên chức năng

Đặc trưng này được trích rút ra theo các hành vi tại thời điểm chạy của tập tin chương trình. Các chức năng này thuộc về một tập tin thực thi và sử dụng chúng để tạo ra các thuộc tính khác nhau đại diện cho tập tin này. Các lời gọi hàm phân tích động bao gồm lời gọi hệ thống, lời gọi đến các API,… Bất kỳ phần mềm độc hại triêu gọi một vài lời gọi hệ thống mức nhân để giao tiếp với hệ điều hành, nó là một dấu hiệu của mã độc.

Lee và các cộng sự đã đề xuất một kỹ thuật phân nhóm mã độc sử dụng kỹ thuật học máy giám sát kết hợp phân tích hành vi của các tập dữ liệu mẫu theo các lời gọi hệ thống và các tham số trong môi trường ảo và tạo ra một hồ sơ hành vi cho các hoạt động mạng. Thêm họ đã tính toán sự tương đồng giữa các hồ sơ đó và phân nhóm các mẫu khác nhau được thực hiện bằng cách áp dụng phâm cụm k- medoids.

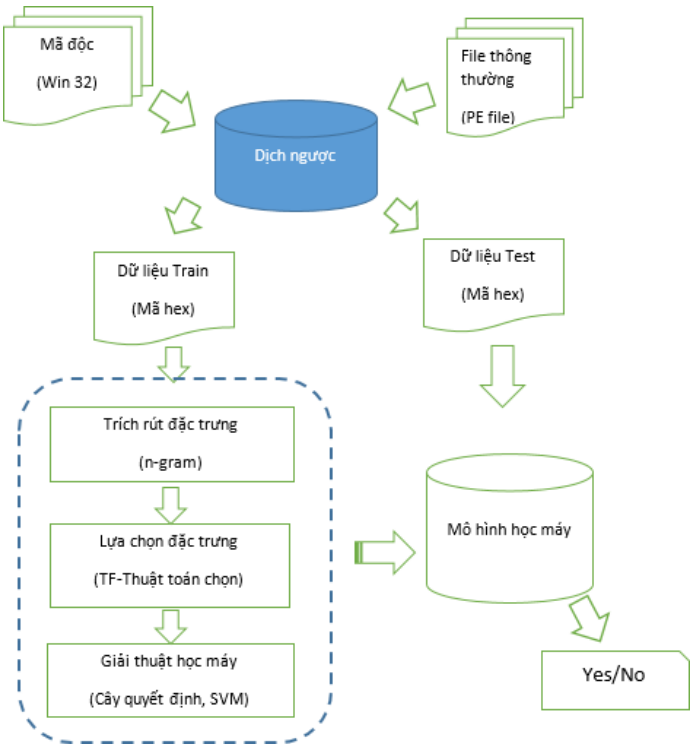
4.3. Tổng quan về phương pháp thực hiện

Hiện nay học máy đang là một hướng nghiên cứu được ứng dụng trong nhiều lĩnh vực đặc biệt là trong các bài toán phân lớp dữ liệu. Với mục tiêu xây dựng một phần mềm có khả năng tự động nhận diện mã độc, trong báo cáo này tôi trình bày một hướng nghiên cứu trong đó ứng dụng kỹ thuật học máy vào việc phân lớp và phát hiện mã độc. Báo cáo tập trung nghiên cứu và đưa ra một đề xuất cũng như xây dựng một giải pháp trích chọn đặc trưng nâng cao hiệu quả và phù hợp cho các bài toán phân lớp dữ liệu. Quá trình được thực nghiệm và phân tích trên các bộ dữ liệu mã độc chỉ ra rằng phương pháp đề xuất cho kết quả phân lớp chính xác và hiệu suất tương đối tốt.

Các nghiên cứu về hướng tiếp cận học máy trong phát hiện và nhận diện mã độc đã được nhiều tác giả tham gia và nghiên cứu [34] trong đó nổi bật lên là sử dụng các phương pháp phân tích mã độc và trích rút các đặc trưng là các mã byte hay các mã opcode [3] hay sử dụng các hàm gọi đến API [23]. Mỗi phương pháp tiếp cận có những ưu điểm và nhược điểm khác nhau, phương pháp dựa vào mã opcode cho kết quả khá tốt tuy nhiên việc dịch ngược ra các mã opcode này đôi khi là khó chính xác cũng như việc tiền xử lý dữ liệu trở nên phức tạp, các phương pháp dựa trên các hàm API lại không bao phủ được đầy đủ các thông tin, phương pháp dựa vào mã byte là một phương pháp hiệu quả dễ dàng trong tiền xử lý tuy nhiên khó khăn trong việc trích chọn các đặc trưng thực sự có hiệu quả. Nhìn chung các phương pháp ở trên đều dựa

vào các nguyên lý trong kỹ thuật học máy và khai phá dữ liệu mà đối tượng là các mã độc [33]. Các nghiên cứu chủ yếu sử dụng phương pháp n-gram [9, 13] và quá trình trích rút đặc trưng dựa vào việc tính tần số xuất hiện của các đặc trưng, tuy nhiên các tập đặc trưng này thường lớn và gây khó khăn cũng như giảm hiệu quả trong quá trình xây dựng mô hình, một số phương pháp chọn đặc trưng như nén các đặc trưng hay giảm số chiều các đặc trưng khá hiệu quả nhưng đôi khi không lọc được các đặc trưng gây nhiễu. Trong nghiên cứu này chúng tôi sẽ đề xuất một giải pháp cho phép lựa chọn các đặc trưng tốt nhất và loại bỏ các đặc trưng có thể gây nhiễu cho mô hình, các đặc trưng này được đánh giá từ chính dữ liệu đầu vào phù hợp với mô hình. Để đơn giản các thực nghiệm và so sánh tôi sử dụng phương pháp n-gram byte trong việc trích xuất các đặc trưng đầu vào.

Trong nghiên cứu này chúng tôi tập trung nghiên cứu các loại mã độc thực thi trên nền Windows các file mã độc được thu thập từ trang chủ "Vxheaven" [42] và các file mã thông thường là các file hệ thống thuộc kiểu file "PE" chủ yếu là các file ".exe" chạy trên hệ điều hành Windows. Tổng quan về phương pháp được mô tả qua 6 bước sau:



Hình 4.1. Tổng quan về phương pháp thực hiện

Bước 1: Thu thập dữ liệu các file mã độc và các file PE thông thường.

Bước 2: Các file này sẽ được dịch ngược về mã hex thông qua một chương trình được viết bằng ngôn ngữ Python.

Bước 3: Sau khi dịch ngược thì các dữ liệu huấn luyện sẽ được trích rút ra các đặc trưng là các mã hex dựa vào phương pháp n-gram. Trong khuôn khổ nghiên cứu này kích thước sử dụng là 2-gram.

Bước 4: Từ các đặc trưng là các mã n-gram byte thực hiện tính tần số xuất hiện (Term Frequency) của các mã n-gram này trên mỗi tập dữ liệu. Sau đó áp dụng thuật toán được tôi đề xuất (được trình bày ở phần sau) để chọn ra được một bộ đặc trưng tốt nhất.

Bước 5: Từ bộ đặc trưng có được ta đưa chúng vào xây dựng các mô hình học máy ở đây tôi sử dụng hai giải thuật để thực nghiệm và so sánh là cây quyết định và svm.

Bước 6: Sau khi xây dựng xong mô hình thì đưa các dữ liệu test vào để đánh giá kết quả.

4.3.1. Tiền xử lý dữ liệu

4.3.1.1. Sử dụng các kỹ thuật phân tích mã độc

Như đã trình bày ở chương 1 hiện nay phân tích mã độc chủ yếu dựa vào 2 phương pháp chính là phân tích động và phân tích tĩnh. Mỗi phương pháp lại đều có những ưu và nhược điểm khác nhau. Phân tích động xác định chính xác mã độc bằng việc quan sát và phân tích các hành vi của nó thông qua việc tạo ra các môi trường thực thi phù hợp cho mã độc, phương pháp này có ưu thế là quá trình phân tích nhanh, dễ dàng ngay cả với những mã độc bị mã hóa phức tạp tuy nhiên lại không hiệu quả khi gặp phải những mã độc chạy theo thời gian hoặc các mã độc có khả năng nhận ra sự theo dõi và sẽ không thực thi các chức năng của nó. Phương pháp phân tích tĩnh dựa vào việc thực hiện dịch ngược mã độc về dạng assembly hay dạng mã hex và phân tích các dấu hiệu của mã độc, phương pháp này tuy gặp khó khăn với những mã độc bị mã hóa hay đóng gói phức tạp nhưng có ưu điểm là có thể phát hiện ra mã độc ngay cả khi không cần thực thi nó và các đặc trưng của mã độc cũng như cấu trúc của nó cho phép phát hiện mã độc một cách chính xác.

Hiện nay đa số quá trình phân tích mã độc còn thủ công tốn nhiều thời gian vì vậy việc ứng dụng học máy để xây dựng các hệ thống có khả năng tự động phát hiện mã độc là điều hết sức cần thiết. Trong nội dung nghiên cứu và thử nghiệm này chúng tôi sử dụng phương pháp phân tích tĩnh, ở giai đoạn tiền xử lý này các file thông thường và các file mã độc là các file dạng PE sẽ được dịch ngược về các mã hex có rất nhiều công cụ cho phép thực hiện điều này có thể kể ra như: IDA, OLLYDBG… Tuy nhiên để đồng bộ và phát triển các hệ thống tự động sau này, chúng tôi đã xây

dựng một chương trình dựa trên cấu trúc của một file PE cho phép dịch ngược các file này về mã hex. Chương trình được viết bằng ngôn ngữ Python và sử dụng thư viện Pefile. Các file dữ liệu mẫu sau khi được dịch ngược sẽ được xử lý để lấy các mã hex quan trọng chủ yếu chúng nằm ở các phần PE header và Section nơi chứa các mã chương trình (Executable Code Section), các dữ liệu (Data Section), các tài nguyên (Resources Section), các thư viện (Import Data, Export Data) …

Các file thông thường và mã độc sau khi được dịch ngược sẽ lấy nội dung các mã hex của từng file này và lưu lại thành các file text tương đương phục vụ cho quá trình trích chọn đặc trưng và xây dựng mô hình dự đoán.

4.3.1.2. Phương pháp sử dụng đặc trưng n-gram

Trong nghiên cứu này chúng tôi sử dụng chuỗi các byte là các đặc trưng đầu vào trong đó ở giai đoạn tiền xử lý các file dữ liệu mẫu được trích xuất dựa vào việc tính tần số xuất hiện của các n-gram byte. N-gram là một dãy các byte liên tiếp có độ dài N được mô tả như sau:

Với một dãy các mã hex sau khi dịch ngược giả sử là "AB C0 EF 12" thì dãy các n-gram byte thu được là:

Bảng 4.1. Mô tả dãy các n-gram byte

1-gram	2-gram	3-gram	4-gram
AB	AB C0	AB C0 EF	AB C0 EF 12
C0	C0 EF	C0 EF 12	
EF	EF 12		
12			

Có thể nhận thấy rằng với độ dài n càng cao thì kích thước đặc trưng càng lớn. Đối với mã hex có 16 giá trị khác nhau như vậy không gian đặc trưng của 1-gram sẽ là 16^2=256 với 2-gram là 16^4=65536. Trong nội dung báo cáo này chúng tôi chủ yếu tập trung vào phương pháp chọn đặc trưng vì vậy các kết quả được thử nghiệm trên dãy 2-gram. Như vậy ở giai đoạn tiền xử lý các file dữ liệu mẫu được dịch ngược sang các mã hex và sau đó tiếp tục được trích rút ra các đặc trưng dựa vào phương pháp n-gram.

4.3.1.3. Tính tần số xuất hiện (Term Frequency)

Như chúng ta đã phân tích ở các phần trên, dữ liệu mẫu sau khi thực hiện tiền xử lý sẽ được lưu dưới dạng các file text và được trích rút các dãy n-gram, một trong những phương pháp được biết đến trong các bài toán khai phá và phân lớp dữ liệu text đó là phương pháp tính tần số xuất hiện (TF). Dựa vào phương pháp như vậy ở bước này chúng tôi thực hiện tính toán tần số xuất hiện của mỗi n-gram byte khác nhau trên từng file dữ liệu mẫu. Các kết quả này được lưu như mỗi véc tơ đặc trưng và trước khi được đưa vào mô hình học sẽ được xử lý để trích ra một bộ đặc trưng tốt nhất.

Công thức để tính các giá trị TF được cho như sau:

$$tf(t,d) = \frac{f(t,d)}{\max\{f(w,d): w \in d}$$

Tần số xuất hiện của một mã n-gram byte trong một tập mẫu (file dữ liệu mẫu đã được đưa về dạng text) được tính bằng thương của số lần xuất hiện n-gram byte đó trong tập mẫu và số lần xuất hiện nhiều nhất của một n-gram byte bất kỳ trong tập mẫu đó (giá trị sẽ thuộc khoảng (0, 1))

- f(t,d) - số lần xuất hiện của một n-gram byte t trong tập mẫu d.
- max{f(w,d):w∈d} - số lần xuất hiện nhiều nhất của một n-gram byte bất kỳ trong tập mẫu.

4.3.2. Đề xuất giải pháp chọn đặc trưng cho thuật toán phân lớp

Trong phương pháp học máy có thể thấy rằng khi số lượng đặc trưng lớn sẽ làm giảm hiệu suất và đôi khi là chất lượng của mô hình học. Lượng đặc trưng quá nhiều sẽ khiến cho quá trình huấn luyện và phân lớp dữ liệu tốn kém về mặt tài nguyên cũng như thời gian xử lý thậm chí nếu nhiều đặc trưng phổ biến sẽ dẫn đến dư thừa gây nhiễu và ảnh hưởng đến chất lượng khi xây dựng mô hình, chính vì vậy bài toán đặt ra và cần thiết là làm sao loại bỏ được các đặc trưng gây nhiễu và chọn được một tập đặc trưng đại diện tốt nhất mà vẫn đảm bảo độ chính xác hiệu quả của mô hình dự đoán. Trong phương pháp n-gram nêu trên ta thấy sẽ có các đặc trưng mà tần số xuất hiện của chúng tương tự nhau trên 2 lớp vì vậy khi đưa vào mô hình học máy sẽ không đạt kết quả cao.

Phương pháp trích chọn đặc trưng mà tôi đề xuất mục đích là tìm ra tập các đặc trưng mà giá trị tần số xuất hiện trung bình của chúng trên 2 lớp cần phân chia có độ lệch lớn nhất. Cụ thể là các mã n-gram byte mà có tần số xuất hiện trên các tập của lớp này khác nhất với chính nó trên các tập của lớp còn lại.

4.3.2.1 Mô tả giải pháp

Gọi D là tập các đặc trưng có độ dài "d" phần tử là các mã n-gram byte. 2 lớp cần phân chia lớp thứ 1 có độ dài là "n" tập mẫu. Lớp thứ 2 có độ dài là "m" tập mẫu.

Gọi TF1[i] là tập chứa các giá trị tần số xuất hiện của đặc trưng D[i] ∈ D; i ϵ [0,d] trên các mẫu dữ liệu thuộc lớp thứ 1. Mỗi TF1[i] với i ϵ [0,d] là một mảng chứa 'n' phần tử .

Gọi TF2[i] là tập chứa các giá trị tần số xuất hiện của đặc trưng D[i] ∈ D; i ϵ [0,d] trên các mẫu dữ liệu thuộc lớp thứ 2. Mỗi TF2[i] với i ϵ [0,d] là một mảng chứa 'm' phần tử.

Bước 1: Với mỗi giá trị i ϵ [0,d] sắp xếp các phần tử trong TF1[i] và TF2[i] theo chiều giảm hoặc tăng dần

Bước 2: Sau khi thực hiện sắp xếp ở bước 1:

Với mỗi i ∈ [0,d] ta thực hiện chia các phần tử trong TF1[i] tương ứng thành "k" đoạn liên tiếp bắt đầu từ phần tử đầu tiên, mỗi đoạn chứa C1 phần tử (số phần tử trong các đoạn có thể không bằng nhau).

Tương tự với mỗi i ∈ [0,d] ta cũng thực hiện chia các phần tử trong TF2[i] tương ứng thành "k" đoạn liên tiếp bắt đầu từ phần tử đầu tiên, mỗi đoạn chứa C2 phần tử (số phần tử trong các đoạn có thể không bằng nhau)

Bước 3: Tính trung bình cộng tần số xuất hiện trên từng khoảng đã chia trên mỗi TF1[i]; i ∈ [0,d] với mỗi TF1[i] ta thu được "k" giá trị TB1[i][j] với i ϵ [0,d]; j∈[0,k] là giá trị tần số xuất hiện trung bình của D[i] trên đoạn j của TF1[i]

Tương tự thực hiện tính trung bình cộng tần số xuất hiện trên từng khoảng đã chia trên mỗi TF2[i]; i ∈ [0,d] với mỗi TF2[i] ta thu được "k" giá trị TB2[i][j] với i ϵ [0,d]; j∈[0,k] là giá trị tần số xuất hiện trung bình của D[i] trên đoạn j của TF2[i]

Bước 4: Tính độ lệch tần số xuất hiện trung bình trên "k" đoạn đã chia của mỗi TF1[i] và TF2[i].

Gọi TB[i][j] là độ lệch trung bình của đặc trưng D[i] trên đoạn j của 2 tập TF1[i] và TF2[i] thì TB[i][j] được tính bằng trị tuyệt đối của phép trừ giữa TB1[i][j] và TB2[i][j] ta có:

$$TB[i][j] = | TB1[i][j] - TB2[i][j] |$$

$$(i \epsilon [0,d]; j \epsilon [0,k])$$

Với mỗi i trện đoạn [0,d] thực hiện tính các TB[i][j] của nó với mỗi j ∈ [0,k] trên 2 tập TF1[i] và TF2[i] tương ứng.

Bước 5: Với mỗi i ϵ [0,d] ta thực hiện tính độ lệch trung bình giá trị tần số của đặc trưng D[i] tương ứng trên toàn tập bằng cách tính sau:

$$DL[i] = \frac{\sum_{j=0}^{k} TB[i][j]}{k}$$

Trong đó DL[i] với i ∈[0,d] là độ lệch trung bình của đặc trưng D[i] trên 2 lớp cần phân chia.

Bước 6: Kết thúc bước 5 ta sẽ thu được kết quả độ lệch trung bình tần số xuất hiện của "d" đặc trưng ban đầu từ "d" đặc trưng này thực hiện chọn ra một bộ đặc trưng có giá trị độ lệch cao nhất để đưa vào xây dựng mô hình.

4.3.2.2 Ví dụ

Bài toán: Giả sử cho 2 lớp với các đặc trưng là các mã hex 2-gram có số tập mẫu khác nhau và tần số xuất hiện như sau:

Bảng 4.2. Ví dụ tập tần số xuất hiện ban đầu của các đặc trưng thuộc lớp 1

Lớp 1	Đặc trưng	AA AA	BB BB	CC CC	DD DD
	Tập 1	0.5	0.7	1.0	0.2
	Tập 2	0.5	0.9	0	0.1
	Tập 3	0.6	0.5	1.0	0.2
	Tập 4	0.6	0.6	0	0.1

Bảng 4.3. Ví dụ tập tần số xuất hiện ban đầu của các đặc trưng thuộc lớp 2

Lớp 2	Đặc trưng	AA AA	BB BB	CC CC	DD DD
	Tập 1	0.6	0.2	0.5	0.1
	Tập 2	0.5	0.3	0.5	0.2
	Tập 3	0.6	0.3	0.5	0.7
	Tập 4	0.5	0.2	0.5	0.8
	Tập 5	0.6	0.3	0.5	0.3
	Tập 6	0.5	0.2	0.5	0.2

Bước 1: sau khi đã tính tần số xuất hiện của các đặc trưng trên mỗi tập mẫu thuộc 2 lớp thực hiện sắp xếp các giá trị theo chiều giảm dần ở mỗi đặc trưng trong cả 2 lớp như sau:

Bảng 4.4. Kết quả tập dữ liệu lớp 1 sau khi thực hiện sắp xếp

Lớp 1	Đặc trưng	AA AA	BB BB	CC CC	DD DD
		0.6	0.9	1.0	0.2
		0.6	0.7	1.0	0.2
		0.5	0.6	0	0.1
		0.5	0.5	0	0.1

Bảng 4.5. Kết quả tập dữ liệu lớp 2 sau khi thực hiện sắp xếp

Lớp 2	Đặc trưng	AA AA	BB BB	CC CC	DD DD
		0.6	0.3	0.5	0.8
		0.6	0.3	0.5	0.7
		0.6	0.3	0.5	0.3
		0.5	0.2	0.5	0.2
		0.5	0.2	0.5	0.2
		0.5	0.2	0.5	0.1

Bước 2: Tiến hành chia lớp 1 và lớp 2 mỗi lớp thành k đoạn. Giả sử ta chọn k=2 như vậy với lớp 1 mỗi đoạn sẽ có 2 giá trị tần số liên tiếp cạnh nhau, với lớp 2 mỗi đoạn có 3 giá trị tần số liên tiếp cạnh nhau.

Bảng 4.6. Kết quả tập dữ liệu lớp 1 sau khi thực hiện phân đoạn

Lớp 1	Đặc trưng	AA AA	BB BB	CC CC	DD DD
	Đoạn 1	0.6	0.9	1.0	0.2
		0.6	0.7	1.0	0.2
	Đoạn 2	0.5	0.6	0	0.1
		0.5	0.5	0	0.1

Bảng 4.7. Kết quả tập dữ liệu lớp 2 sau khi thực hiện phân đoạn

Lớp 2	Đặc trưng	AA AA	BB BB	CC CC	DD DD
	Đoạn 1	0.6	0.3	0.5	0.8
		0.6	0.3	0.5	0.7
		0.6	0.3	0.5	0.3
	Đoạn 2	0.5	0.2	0.5	0.2
		0.5	0.2	0.5	0.2
		0.5	0.2	0.5	0.1

Minh họa với đặc trưng "BB BB" ta có:

Bảng 4.8. Minh họa với đặc trưng "BB BB" thuộc lớp 1 sau khi thực hiện phân đoạn

Lớp 1	Đặc Trưng	BB BB
	Đoạn 1 (chứa 2 phần tử)	0.9
		0.7
	Đoạn 2 (chứa 2 phần tử)	0.6
		0.5

Bảng 4.9. Minh họa với đặc trưng "BB BB" thuộc lớp 2 sau khi thực hiện phân đoạn

Lớp 2	Đặc Trưng	BB BB
	Đoạn 1 (chứa 3 phần tử)	0.3
		0.3
		0.3
	Đoạn 2 (chứa 3 phần tử)	0.2
		0.2
		0.2

Bước 3: Trên mỗi đoạn thực hiện tính ra giá trị trung bình cộng tần số xuất hiện của mỗi đặc trưng:

Minh họa với đặc trưng "BB BB":

- Giá trị tần số xuất hiện trung bình của nó trên đoạn 1 của lớp 1 sẽ là (0.9+0.7)/2=0.8
- Giá trị tần số xuất hiện trung bình của nó trên đoạn 2 của lớp 1 sẽ là : (0.6+0.5)/2 = 0.55
- Giá trị tần số xuất hiện trung bình của nó trên đoạn 1 của lớp 2 sẽ là: (0.3+0.3+0.3)/3=0.3
- Giá trị tần số xuất hiện trung bình của nó trên đoạn 2 của lớp 2 sẽ là: (0.2+0.2+0.2)/3=0.2

Tương tự cách tính như vậy với các đặc trưng còn lại thu được bảng sau:

Bảng 4.10. giá trị trung bình cộng tần số xuất hiện của mỗi đặc trưng trên từng đoạn thuộc lớp 1

Lớp 1	Đặc trưng	AA AA	BB BB	CC CC	DD DD
TF trung bình trên từng đoạn	Đoạn 1	0.6	0.8	1.0	0.2
	Đoạn 2	0.5	0.55	0	0.1

Bảng 4.11. giá trị trung bình cộng tần số xuất hiện của mỗi đặc trưng trên từng đoạn thuộc lớp 2

Lớp 2	Đặc trưng	AA AA	BB BB	CC CC	DD DD
TF trung bình trên từng đoạn	Đoạn 1	0.6	0.3	0.5	0.6
	Đoạn 2	0.5	0.2	0.5	0.16

Bước 4: Thực hiện tính độ lệch của tần số xuất hiện giữa các đoạn của từng đặc trưng trên 2 lớp, bằng cách trừ lấy trị tuyệt đối 2 tần số trên cùng một đoạn của cùng 1 đặc trưng trên 2 lớp dữ liệu.

Minh họa với đặc trưng " BB BB" thì:

- Độ lệch tần số giữa 2 lớp của đặc trưng " BB BB" trên đoạn 1 là: |0.8 – 0.3|=0.5

- Độ lệch tần số giữa 2 lớp của đặc trưng " BB BB" trên đoạn 2 là: |0.55-0.2|=0.35

Bảng 4.12. Kết quả độ lệch tần số xuất hiện trên từng đoạn

Độ lệch TF trên từng đoạn	Đặc trưng	AA AA	BB BB	CC CC	DD DD
	Đoạn 1	0	0.5	0.5	0.4
	Đoạn 2	0	0.35	0.5	0.06

Bước 5: Từ kết quả độ lệch tần số xuất hiện có được trên các đoạn ở bước 4 tiến hành tính trung bình độ lệch tần số xuất hiện của mỗi đặc trưng trên toàn tập, bằng cách tính trung bình cộng độ lệch tần số xuất hiện trên các đoạn của mỗi đặc trưng để ra kết quả cuối cùng.

Minh họa với đặc trưng "BB BB" thì độ lệch tần số trung bình trên cả toàn dữ liệu là: (0.5+0.35)/2=0.425

Bảng 4.13. Kết quả độ lệch tần số xuất hiện trên toàn tập dữ liệu

Độ lệch TF trên toàn dữ liệu	Đặc trưng	AA AA	BB BB	CC CC	DD DD
	Đoạn 1	0	0.425	0.5	0.23

Như vậy dễ thấy các đặc trưng có độ lệch cao nhất là "CC CC" và "BB BB" sau đó là "DD DD" ta có thể xác định được các đặc trưng này có sự sai khác lớn về tần số xuất hiện của 2 lớp đúng theo thứ tự.

Bước 6: Từ các kết quả có được thực hiện chọn ra T đặc trưng có độ lệch về giá trị tần số xuất hiện lớn nhất.

4.3.3. Xây dựng mô hình dự đoán dựa trên các thuật toán phân lớp

Sau khi chọn được tập đặc trưng dựa vào phương pháp đã trình bày phía trên, bước tiếp theo ở giai đoạn này là tiến hành đưa các giá trị tần số xuất hiện của các đặc trưng n-gram byte đã được chọn trên các tập mẫu ban đầu vào các thuật toán trong học máy để xây dựng mô hình dự đoán., các dữ liệu mẫu được gán nhãn thành 2 lớp mã độc và file bình thường. Giai đoạn này sử dụng các véc tơ đặc trưng như là dữ liệu huấn luyện đầu vào cho các thuật toán phân lớp dữ liệu [28,39] có thể áp dụng như: k láng giềng (KNN), Naive Bayes (NB), cây quyết định (DT), Rừng ngẫu nhiên (RF), máy véc tơ hỗ trợ (SVM)… Trong phương pháp học máy giám sát khó có thể

nói là thuật toán phân lớp nào tốt nhất vì mỗi thuật toán có những ưu, nhược điểm riêng phù hợp cho mỗi loại dữ liệu khác nhau. Riêng đối với các bài toán phân lớp dữ liệu văn bản thì cây quyết định và SVM là 2 thuật toán được cho là có độ chính xác cao và được sử dụng khá rộng rãi hiện nay. Trong đó thuật toán máy véc tơ hỗ trợ (SVM) được đánh giá là rất phù hợp và hiệu quả trong các bài toán phân lớp dữ liệu văn bản, trong khi đó cây quyết định (DT) là thuật toán dễ hiểu, dựa vào việc sinh luật khá gần gũi tư duy của con người .Trong nghiên cứu này tôi lựa chọn 2 thuật toán tiêu biểu là cây quyết định và máy véc tơ hỗ trợ để xây dựng mô hình đự đoán phát hiện mã độc đồng thời các kết quả của 2 phương pháp sẽ được so sánh và đánh giá thông qua quá trình thực nghiệm.

4.4. Thực nghiệm và đánh giá

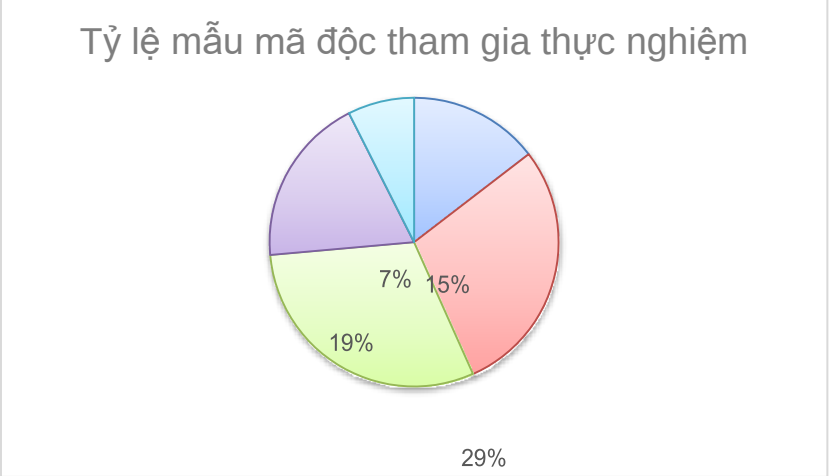
4.4.1. Dữ liệu thực nghiệm

Các kết quả của đề tài được thực nghiệm với đầu vào là 4698 file dữ liệu mẫu trong đó 2373 file mã thông thường và 2325 file mã độc, file mã độc được thu thập từ trang chủ “Vxheaven” [42] và các file mã thông thường là các file hệ thống thuộc kiểu file “PE” chủ yếu là các file “.exe” chạy trên trên hệ điều hành Windows số lượng các mẫu mã độc được mô tả như sau:

Bảng 4.14. Tỷ lệ số lượng từng mẫu mã độc tham gia thực nghiệm

Kiểu mã độc	Backdoor. win32	Worm. win32	Virus. win32	Trojan. win32	Khác
Số lượng mẫu	339	668	704	441	173

Trong đó tỷ lệ các mẫu mã độc được biểu diễn trực quan trong biểu đồ sau:



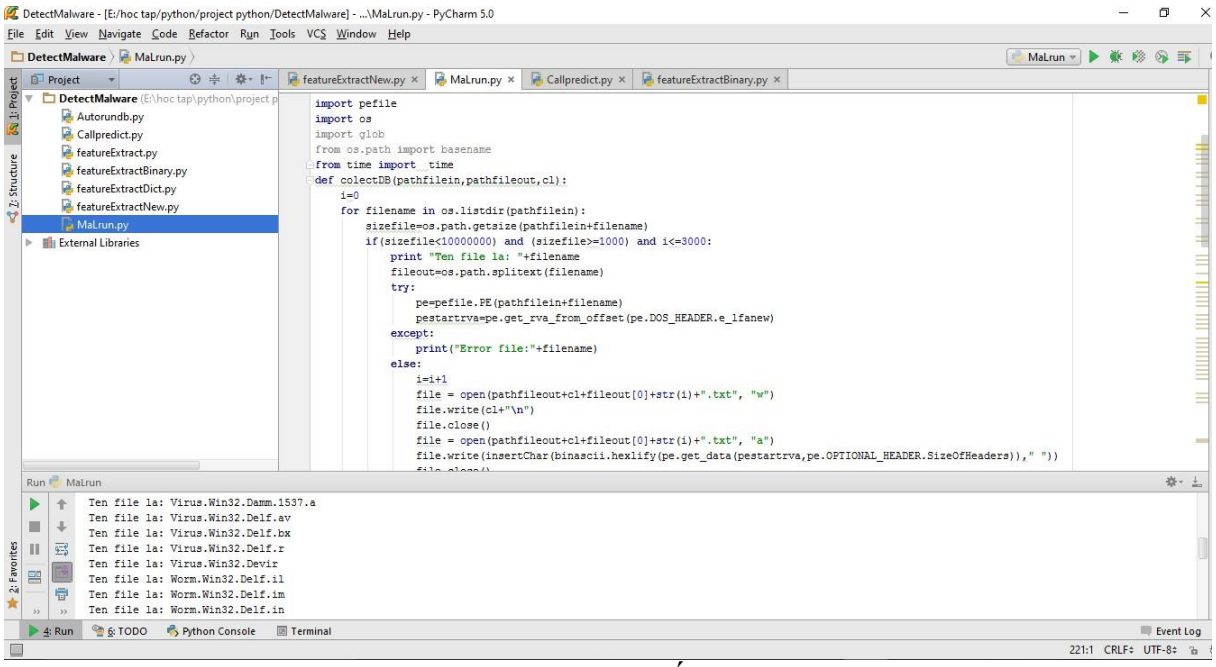
Hình 4.2. Biểu đồ mô tả tỷ lệ mẫu mã độc tham gia thực nghiệm

4.4.2. Chương trình thực nghiệm

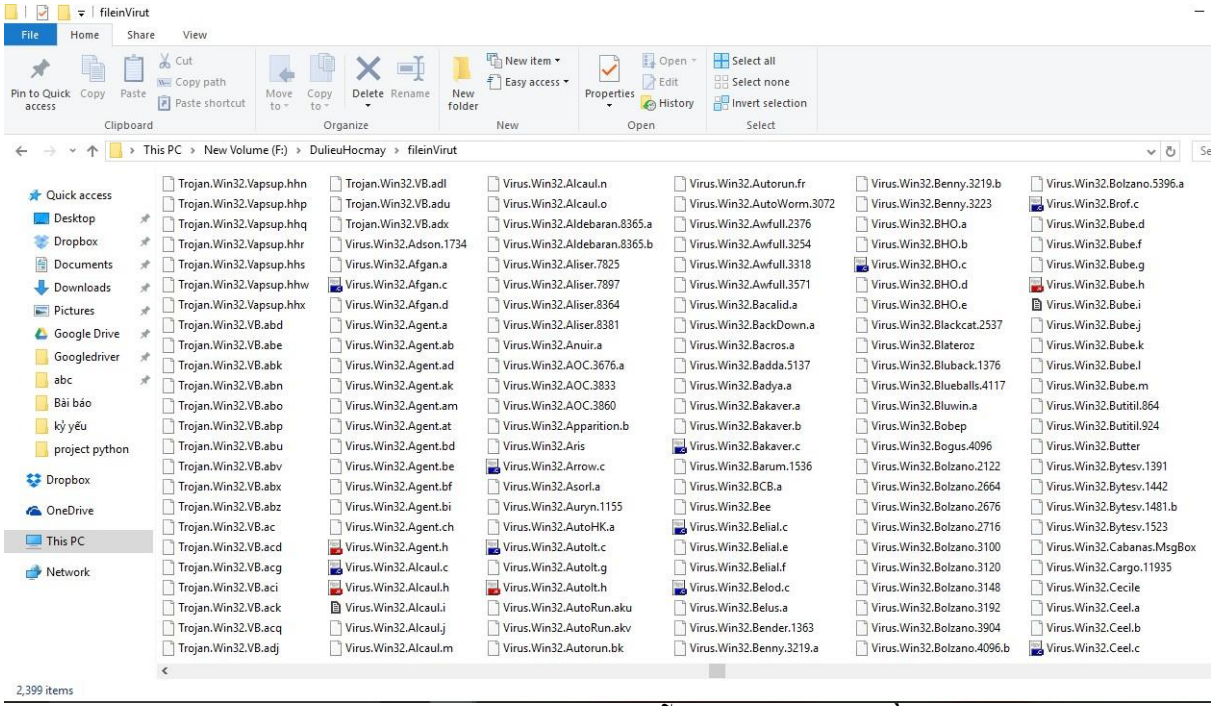
Bảng 4.15. Thông tin về chương trình thực nghiệm

Môi trường thực nghiệm	- Processor: Intel(R) Core i5 – 2450M CPU @ 2.50GHz - Memory (Ram): 8.00 GB - System type: 64-bit Operating System, x64-based processor - Windows 10 Pro -2016
Ngôn ngữ sử dụng	Python
Công cụ lập trình (IDE)	PyCharm – JetBrains 5.0 Pro
Thư viện và các gói cài đặt chính	Scikit-learn, scipy, numpy,matplotlib,ipython, Pefile

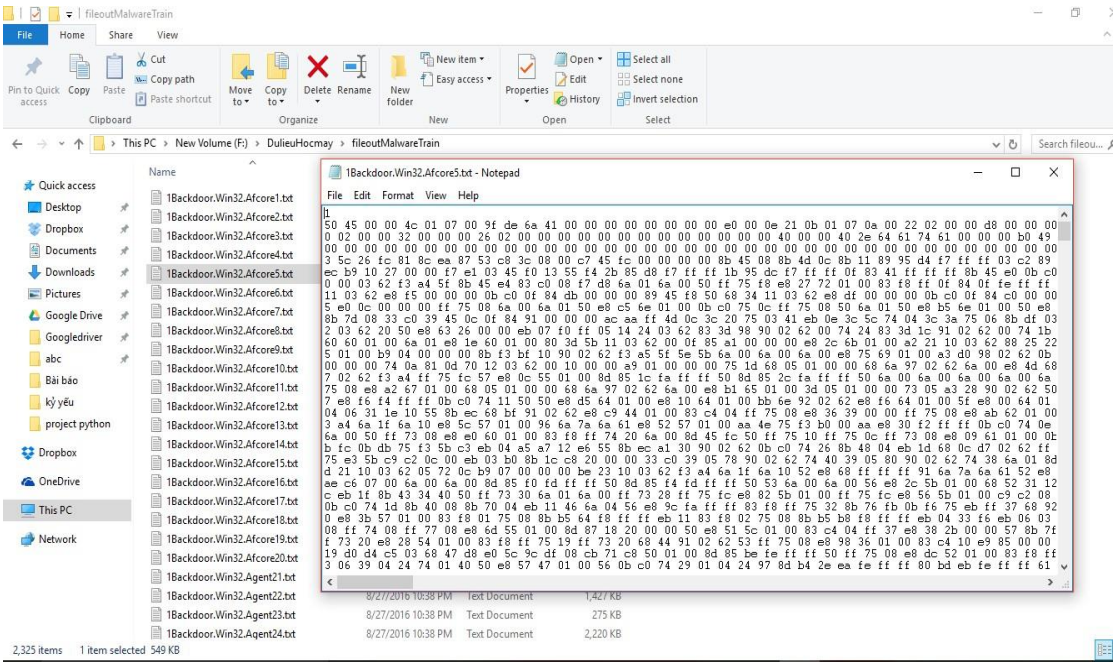
Chương trình thực nghiệm được lập trình trên ngôn ngữ Python trong đó 2 mã nguồn chính là chương trình thực hiện đọc, phân tích cấu trúc một tập tin PE và tiến hành dịch ngược về mã hex, chương trình thứ 2 là chương trình đọc các dữ liệu mẫu, chạy các thuật toán trích chọn đặc trưng và chạy các thuật toán học máy:



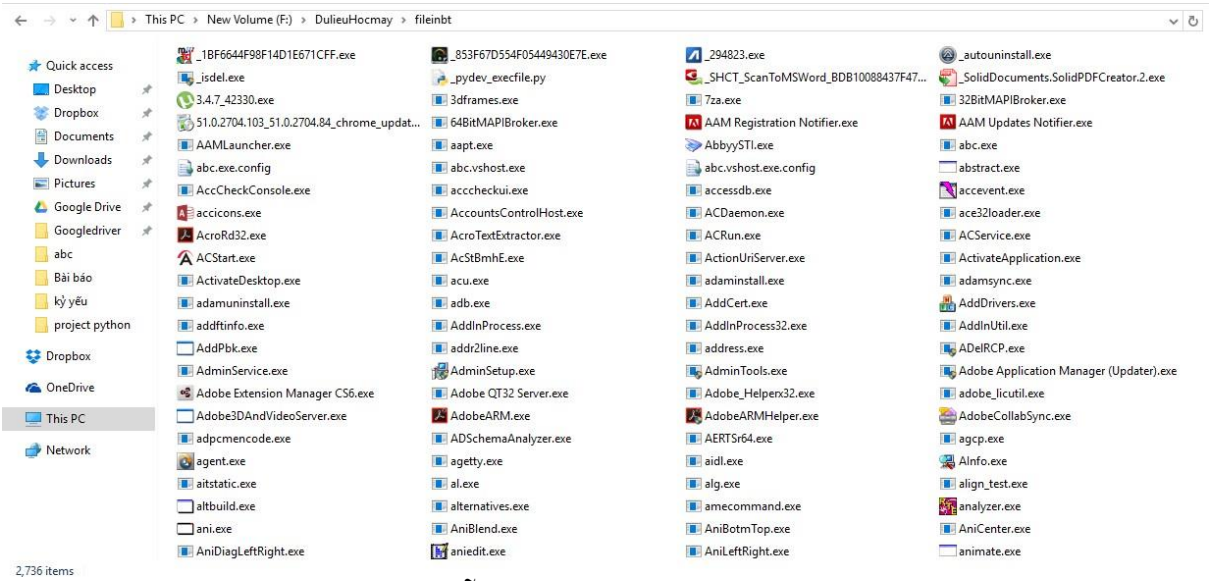
Hình 4.3. Chương trình đọc, phân tích cấu trúc các file PE và dịch ngược



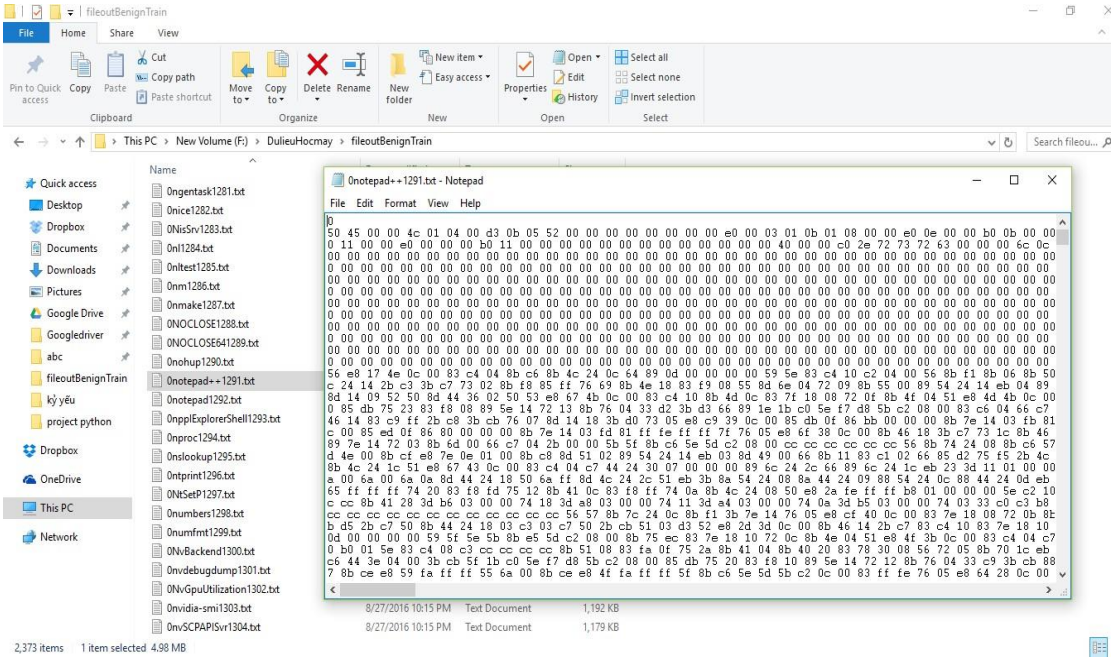
Hình 4.4. Tập dữ liệu mẫu mã độc ban đầu



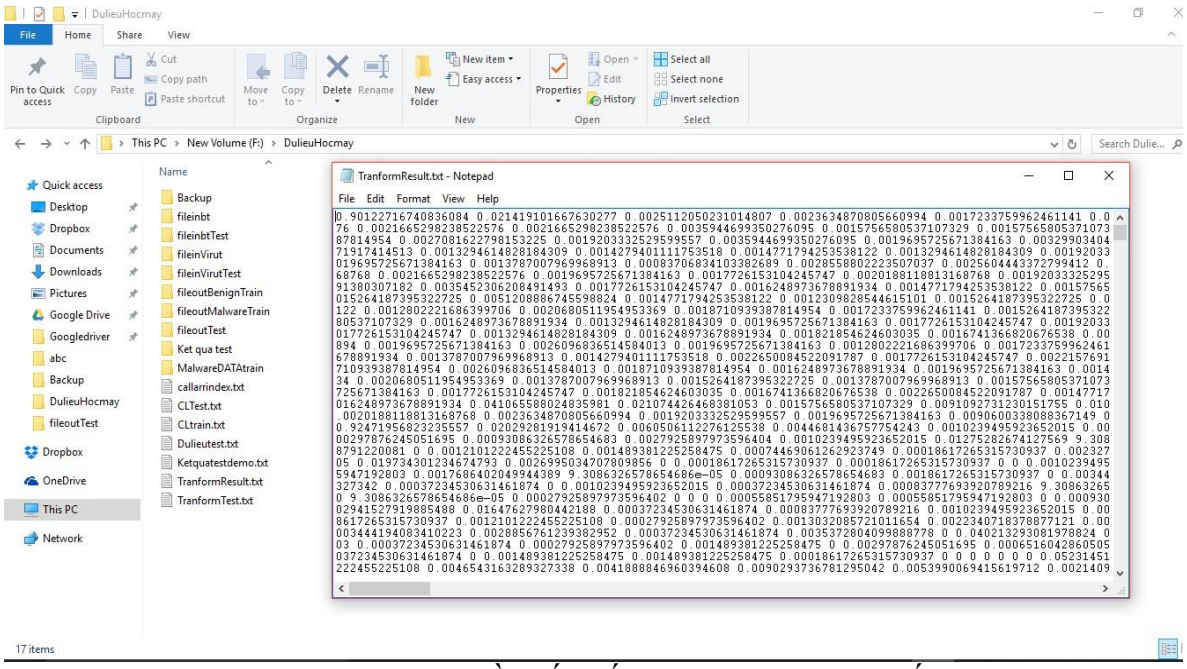
Hình 4.5. Tập dữ liệu mẫu mã độc sau khi được dịch ngược về mã hex và được gán nhãn phân lớp



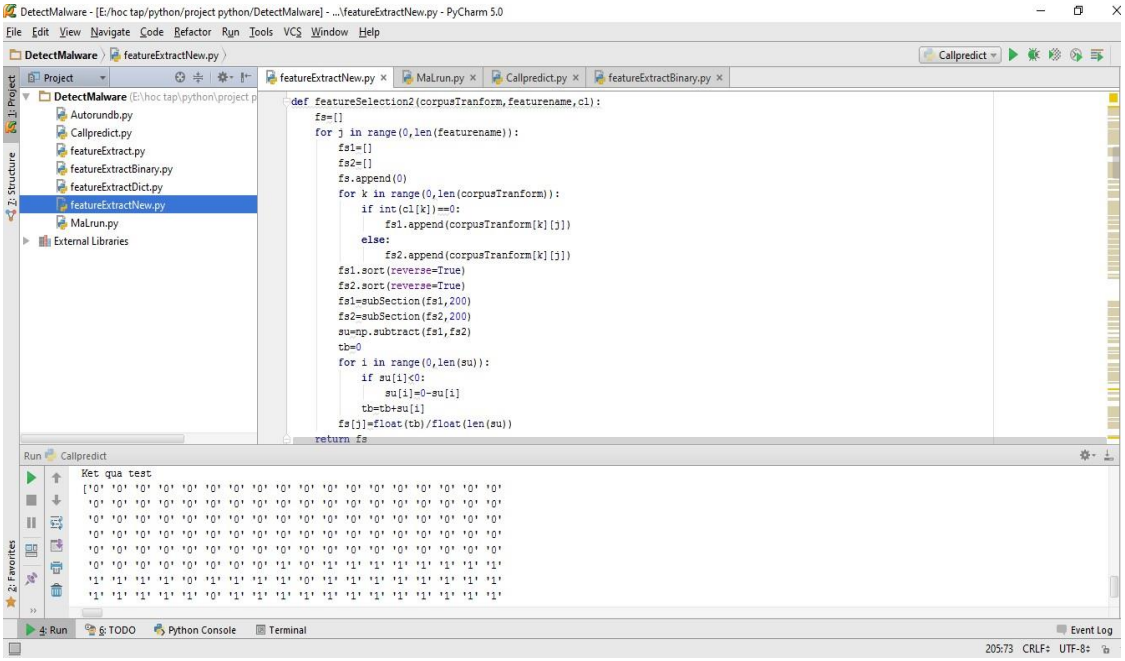
Hình 4.6. Tập dữ liệu mẫu các file thực thi thông thường trên Windows



Hình 4.7. Tập dữ liệu mẫu các file thực thi thông thường sau khi được dịch ngược về mã hex và được gán nhãn phân lớp



Hình 4.8. Tập các tần số xuất hiện của dữ liệu huấn luyện



Hình 4.9. Chương trình trích chọn đặc trưng và xây dựng mô hình dự đoán mã độc

4.4.3. Đánh giá dựa trên phương pháp ma trận nhầm lẫn

Một mô hình được đánh giá tốt nếu như tỉ lệ: TP (True Positive) và TN (True Negative) lớn đồng thời tỉ lệ FP (False Positive) và FN (False Negative) là nhỏ trong đó:

TP: Số lượng các mẫu thuộc lớp dương được phân loại chính xác vào lớp dương.

FP: Số lượng các mẫu không thuộc lớp dương bị phân loại nhầm vào lớp dương.

TN: Số lượng các mẫu không thuộc lớp dương được phân loại đúng.

FN: Số lượng các mẫu thuộc lớp dương bị phân loại nhầm vào các lớp không phải lớp dương.

		Lớp dự đoán (Predicted class) +	Lớp dự đoán (Predicted class) −
Lớp thực (True class)	+	True Positive-TP	False Negative-FN Type II error
	−	(False Positive-FP) Type I error	True Negative-TN

Độ chính xác của mô hình được tính bằng công thức sau:

$$Acc=(TP+TN)/(TP+TN+FP+FN)$$

TPR (True positive rate) được tính bằng công thức sau:

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

FPR (False positive rate) được tính bằng công thức sau:

$$\text{FPR} = \frac{FP}{FP+TN}$$

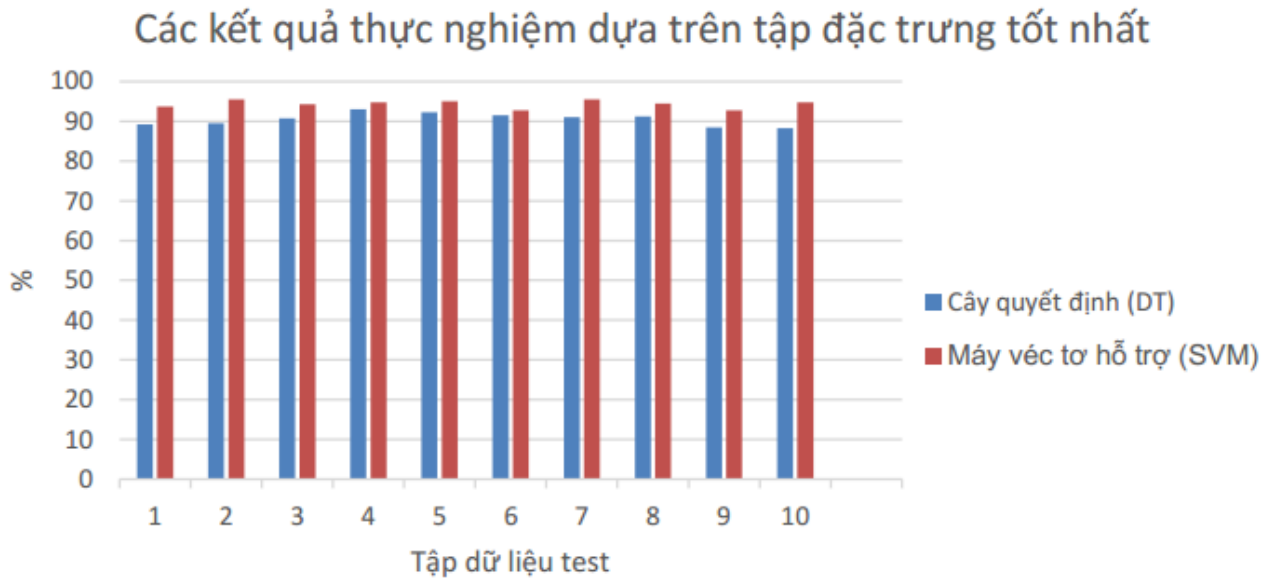
4.4.4. Kết quả thực nghiệm

Tiến hành trích rút các đặc trưng 2-gram byte như vậy toàn bộ không gian đặc trưng sẽ là 16^4=65536 đặc trưng, sau đó áp dụng thuật toán đã đề xuất tôi thực hiện thu gọn và trích chọn ra 800 đặc trưng tốt nhất. Trong tổng số 4698 file thực hiện chọn ra 400 file test (trong đó 200 file mã độc và 200 file thông thường) các file còn lại độc lập với dữ liệu test được dùng làm dữ liệu huấn luyện. Tiến hành kiểm tra chéo trên các tập dữ liệu bằng cách chọn ngẫu nhiên 400 file test khác nhau với các file huấn luyện độc lập còn lại và thực hiện kiểm tra nhiều lần, các kết quả được đáng giá dựa trên phương pháp ma trận nhầm lẫn trong đó coi lớp dương (Positive) là mã độc , lớp âm (Negative) là file thông thường, các kết quả thu được mô tả trong bảng sau:

Bảng 4.16. Kết quả thực nghiệm trên các mô hình sử dụng tập đặc trưng tốt nhất

n-gram	Số đặc trưng	Mô hình	Độ chính xác	TPR	FPR
2	800	DT	90.75%	0.90	0.085
2	800	SVM	94.50%	0.945	0.055

Trong đó các kết quả chi tiết thực nghiệm trên các tập dữ liệu test khác nhau như sau:



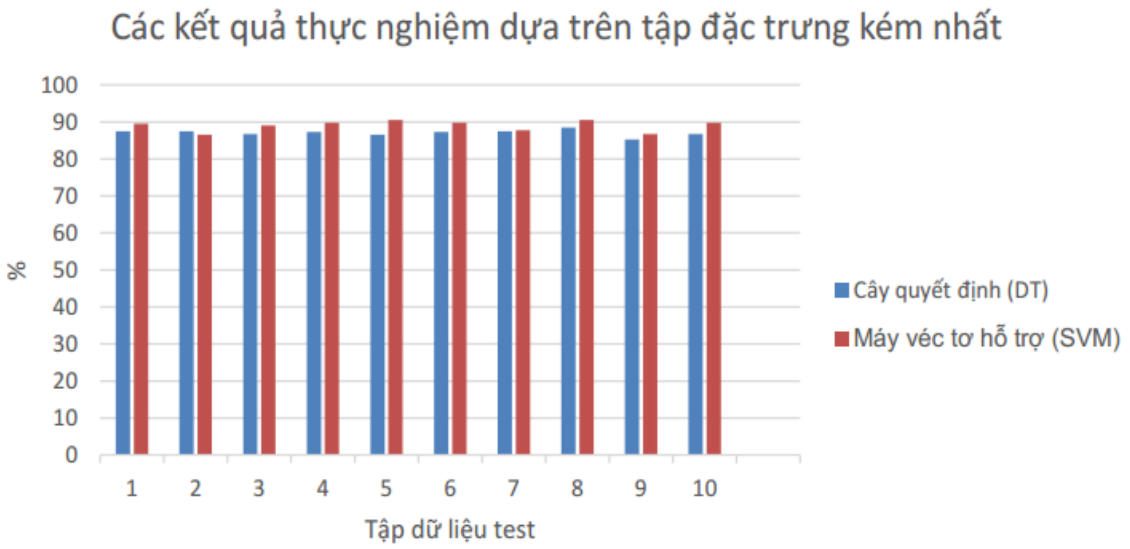
Hình 4.10. Kết quả độ chính xác thực nghiệm trên các tập dữ liệu test khác nhau sử dụng các đặc trưng được đánh giá tốt nhất

Để có thể so sánh được hiệu quả của giải pháp chọn đặc trưng đã đề xuất. Tôi tiến hành kiểm tra trên cùng tập dữ liệu và sử dụng 800 đặc trưng khác bị đánh giá kém nhất (có độ lệch tần số xuất hiện thấp nhất) để xây dựng mô hình học, các kết quả kiểm tra thu được như sau:

Bảng 4.17. Kết quả thực nghiệm trên các mô hình sử dụng tập đặc trưng kém nhất

n-gram	Số đặc trưng	Mô hình	Độ chính xác	TPR	FPR
2	800	DT	87.3%	0.855	0.11
2	800	SVM	88.5%	0.93	0.16

Trong đó các kết quả chi tiết thực nghiệm trên các tập dữ liệu test khác nhau như sau:



Hình 4.11. Kết quả độ chính xác thực nghiệm trên các tập dữ liệu test khác nhau sử dụng các đặc trưng bị đánh giá kém nhất

Nhận xét: Như vậy dựa vào các kết quả thực nghiệm có thể thấy tập đặc trưng được đánh giá cao mà thuật toán đã tìm ra cho kết quả tốt hơn rất nhiều so với tập đặc trưng bị đánh giá thấp.

KẾT LUẬN VÀ KIẾN NGHỊ

Đề tài đã xây dựng thành công phần mềm diệt virus ICTUAV, trong đề tài chúng tôi đã đề xuất sử dụng giải pháp trích chọn đặc trưng để tăng khả năng phát hiện mã độc của phần mềm, đây là một đóng góp quan trọng của đề tài. Tuy nhiên phần mềm vẫn còn một nhược điểm là chưa có cơ chế bảo vệ thời gian thực. Trong thời gian tới chúng tôi sẽ khắc phục nhược điểm này và tiếp tục tập trung nâng cấp phần mềm, cải thiện giải pháp trích chọn đặc trưng để nâng cao khả năng phát hiện mã độc.

Để kết quả của đề tài có thể tiếp tục được phát triển cần có một phòng thí nghiệm chuyên phân tích về mã độc, qua đó có thể xây dựng được cơ sở dữ liệu mã độc cho phần mềm.

TÀI LIỆU THAM KHẢO

[1] Alfred J. Menezes, Paul V. Oorschot, Scott Vanstone (1996), *Handbook of applied Cryptography*.

[2] Álvarez, V. M. (2014), *YARA User's Manual*.

[3] Asaf Shabtai, Robert Moskovitch, Clint Feher, Shlomi Dolev and Yuval Elovici (2012), "Detecting unknown malicious code by applying classification techniques on OpCode patterns", *Security Informatics*, 1 (1). https://doi.org/10.1186/2190-8532-1-1.

[4] Asaf Shabtai, R. Moskovitch, Y. Elovici, C.Glezer (2009), "Detection of malicious code by applying machine learning classifiers on static features: A state-of-the-art survey", *Information Security Technical Report*, 14, pp. 16-29.

[5] Cai DM, Gokhale M, Theiler J (2007), "Comparison of feature selection and classification algorithms in identifying malicious executables", *Computational Statistics and Data Analysis*, 51, pp. 3156-3172.

[6] Chih-Wei Hsu, Chih-Chung Chang, and Chih-Jen Lin (2016), *A Practical Guide to Support Vector Classification*, Department of Computer Science National Taiwan University- Taipei 106- Taiwan.

[7] Dien N.K., Hieu T.T., Thinh T.N. (2014), "Memory-Based Multi-pattern Signature Scanning for ClamAV Antivirus", *FDSE 2014 Springer Cham*, 8860, pp. 58-70.

[8] Digit Oktavianto, Iqbal Muhardianto (2013), *Cuckoo Malware Analysis*, Packt Publishing.

[9] D Krishna, Sandeep Reddy, Arun K Pujari (2006), "N-gram analysis for computer virus detection", *Journal in Computer Virology*, 2 (3), pp. 231-239.

[10] Eldad Eilam (2005), *Reversing-Secrets of Reverse Engineering*, Wiley, 1 Edition April15.

[11] Gareth James, Daniela Witten, Trevor Hastie, Robert Tibshirani (2013), *An Introduction to Statistical Learning with Applications in R (Springer Texts in Statistics)*". Springer Publishing.

[12] Google LLC (2017), *BinDiff Manual Version 4.3*.

[13] Igor Santos, Yoseba K. Penya, Jaime Devesa and Pablo G. Bringas (2009). *N-grams-based file signatures for malware detection*, International Conference on Enterprise Information Systems, 2 (9), pp. 317-320 .

[14] Islam R, Tian R, Batten L, Versteeg S. (2010), "Classification of malware based on string and function feature selection". *In Proceedings of Cybercrime and Trustworthy Computing Workshop (CTC),* pp. 213-223.

[15] Jesse Kornblum (2015), *Ssdeep'documentation*.

[16] Jesse Kornblum (2006), "Identifying Almost Identical Files Using Context Triggered Piecewise Hashing", *Elsevier Digital investigation 3S*, pp.91-97.

[17] Jeffery D. Dodson and Ambareen Siraj (2015), "Applying Fuzzy Hashing to Steganography", *International Journal of Future Computer and Communication*, 4(6), pp.421-425.

[18] Jiang Ming, Dongpeng Xu (2017), "BinSim:Trace-based Semantic Binary Diffing via System Call Sliced Segment Equivalence Checking", *In Proceeding of the 26th USENIX Security Symposium*, pp. 134-143.

[19] Joachim Eibl (2007), *The KDiff3 Handbook,* The Free Software Foundation.

[20] John Aycock (2006), *Computer viruses and Malware (Advances in Information Security,* Springer Publishing.

[21] Justin Ferguson, Dan Kaminsky, Jason Larsen, Luis Miras, Walter Pearce (2008), *Reverse Engineering Code with IDA Pro,* Andrew Williams, USA.

[22] Kong. D and Yan. G (2013), "Discriminant Malware Distance Learning on Structural Information for Automated Malware Classification". *Proceedings of the ACM SIGMETRICS/International Conference on Measurement and Modeling of Computer Systems*, pp. 347-348.

[23] Madhu K. Shankarapani, Subbu Ramamoorthy, Ram S. Movva, Srinivas Mukkamala (2010), "Malware detection using assembly and API call sequences", *Journal in Computer Virology*, 7 (2), pp. 107-119.

[24] Michael Hale Ligh, Steven Adair, Blake Hartstein, Matthew RichardMalware (2011), *Analyst's Cookbook and DVD: Tools and Techniques for Fighting Malicious Code*, Wiley Publishing, Inc.

[25] Michael Sikorski, Andrew Honig (2012), *Practical Malware Analysis*, No Starch Press, 1 edition March 3.

[26] Nataraj L., Karthikeyan S., Jacob G. and Manjunath B (2011), "Malware Images: Visualization and Automatic Classification", *Proceedings of the 8th International Symposium on Visualization for Cyber Security*, Article No. 4.

[27] Nataraj L., Yegneswaran V., Porras P. and Zhang (2011), "A Comparative Assessment of Malware Classification Using Binary Texture Analysis and Dynamic Analysis". *Proceedings of the 4th ACM Workshop on Security and Artificial Intelligence*, pp. 21-30.

[28] Peter Harrington (2012), *Machine Learning in Action,* Manning Publications, pp. 1–129.

[29] Peter Szor (2005), *The Art of Computer Virus Reseach and Defense*, Addison Wesley Professional.

[30] Ricardo Dias (2014), *Intelligence-Driven Incident Response with YARA*, The SANS Institute.

[31] R. Islam, R Tian, Lynn, M. Batten , S. Versteeg (2013), "Classification of malware based on integrated static and dynamic features", *Journal of Network and Computer Applications,* 36 (2), pp. 646-656.

[32] Santos I., Devesa J., Brezo F., Nieves J. and Bringas P.G (2013), "OPEM: A Static-Dynamic Approach for Machine Learning Based Malware Detection", *Proceedings of International Conference CISIS12-ICEUTE12, Special Sessions Advances in Intelligent Systems and Computing,* 189, pp. 271-280.

[33] Schultz M, Eskin E, Zadok E, Stolfo S (2001), "Data mining methods for detection of new malicious executables", *Proc of the IEEE Symposium on Security and Privacy, IEEE Computer Society*, pp. 116-126.

[34] Smita Ranveer, Swapnaja Hiray (June 2015), "Comparative Analysis of Feature Extraction Methods of Malware Detection", *International Journal of Computer Applications (0975 8887),* 120 (5), pp. 244-254.

[35] Steve R. Gunn (1998), *Support Vector Machines for Classification and Regression,* Technical Report, Faculty of Engineering, Science and Mathematics School of Electronics and Computer Science.

[36] Tom M. Mitchell(1997), *Machine Learning,* McGraw-Hill Science.

[37] Tomasz Kojm (2016), *Clamav User manual*, Cisco Systems, Inc.

[38] Tony Abou-Assaleh, Nick Cercone, Vlado Keselj, and Ray Sweidan(2004), "Detection of new malicious code using n-grams signatures". *In Proceedings of Second Annual Conference on Privacy, Security and Trust*, pp. 193-196.

[39] Trent Hauck (2014), "Classifying Data with scikit-learn", *scikit-learn Cookbook*, Packt Publishing, pp. 119-157.

[40] Vlad Pirogov (2006), *Disassembling Code: IDA Pro and SoftICE*, A-LIST Publishing.

[41] V. Vapnik (1995), *The Nature of Statistical Learning Theory*, Springer, New York.

[42] Vxheavens, http://vx.netlux.org.

[43] William Stallings (2005), *Cryptography and Network Security Principles and Practices*, Fourth Edition, Prentice Hall.

[44] Yuping Li, Sathya Chandran Sundaramurthy, Alexandru G. Bardas, Xinming Ou, and Doina Caragea, Xin Hu, Jiyong Jang (2015), "Experimental study of fuzzy hashing in malware clustering analysis", *In Proceeding CSET'15 Proceedings of the 8th USENIX Conference on Cyber Security Experimentation and Test*, August 10, pp. 145-154.

[45] Z. Wang, K. Pierce and S. McFarling (2000), "BMAT - A Binary Matching Tool for Stale Profile Propagation", *The Journal of Instruction-Level Parallelism (JILP)*, 2, pp. 87-100.

PHỤ LỤC: PHẦN MỀM DIỆT VIRUS ICTUAV

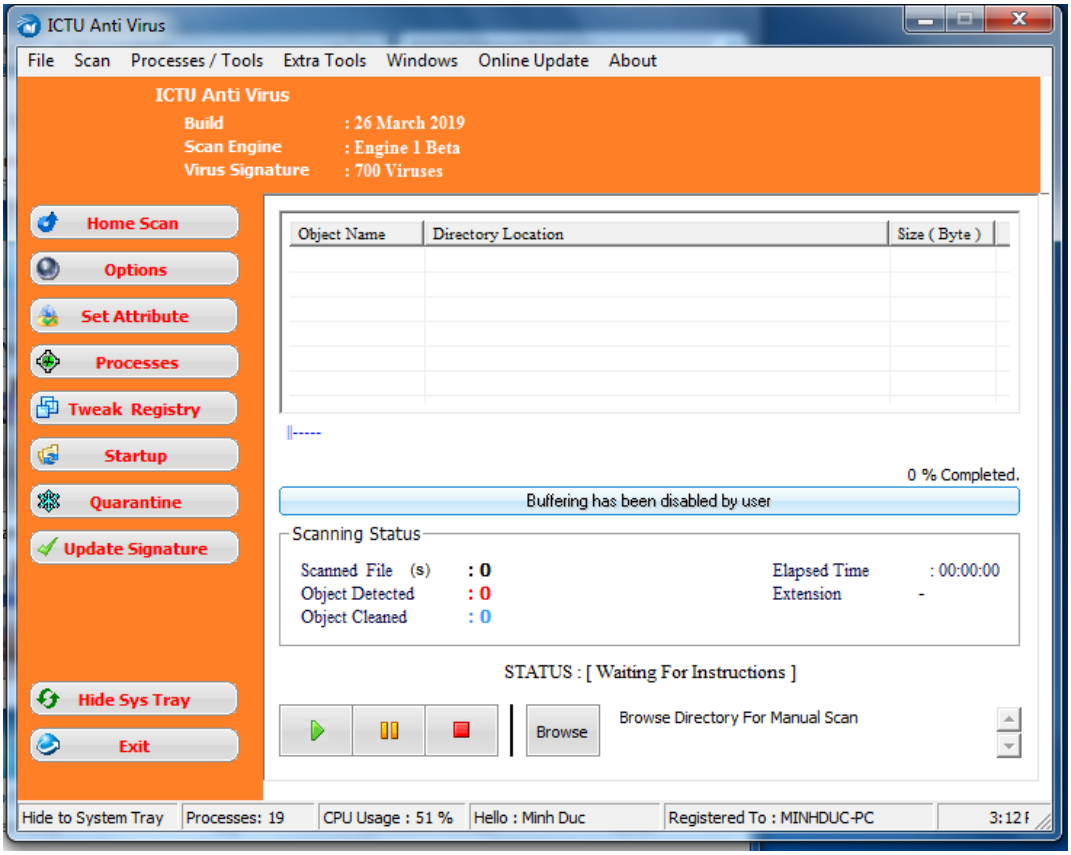
Tổng quan phần mềm

Phần mềm được xây dựng dựa trên bộ công cụ chống virus mã nguồn mở ClamAV. Chúng tôi kết hợp sử dụng engine của ClamAV, thuật toán trích chọn đặc trưng trình bày trong chương 4 và quá trình quét mã độc dưa vào kinh nghiệm (heuristics) trong mô đun quét mã độc của phần mềm.

Ngoài chức năng quét và diệt mã độc phần mềm còn cung cấp thêm nhiều chức năng tiện ích khác cho người dùng trên hệ điều hành Windows.

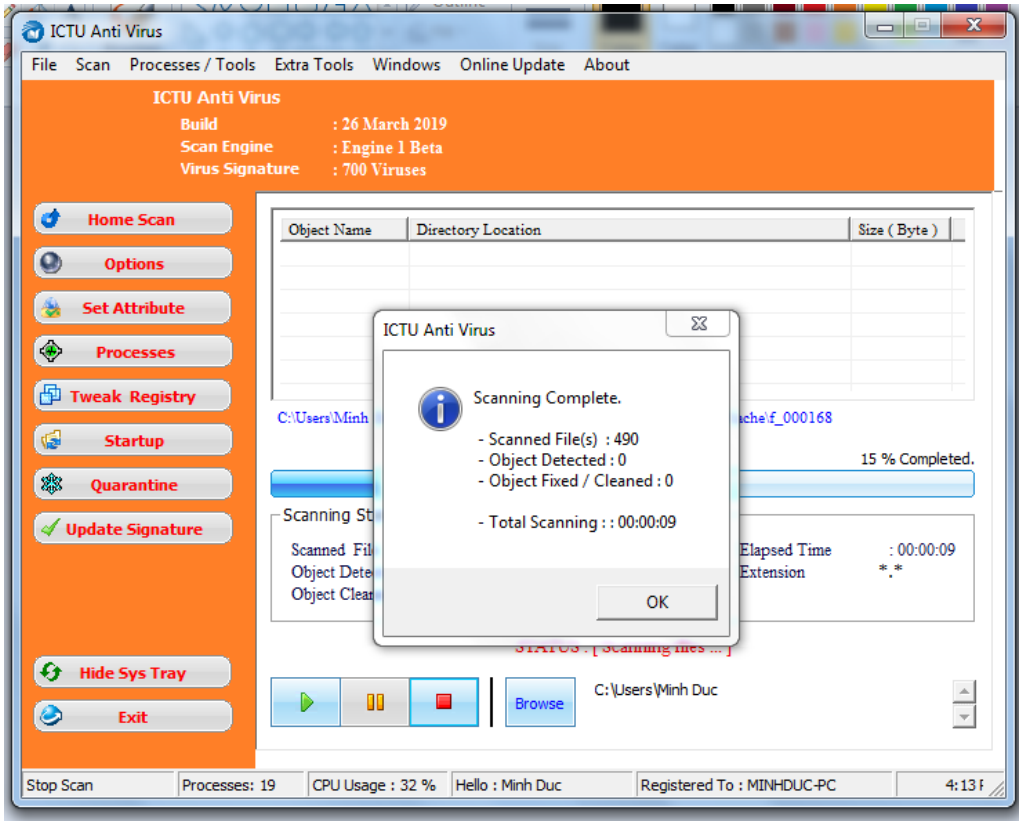
Các chức năng của phần mềm

Chi tiết cách sử dụng phần mềm chúng tôi gửi kèm theo báo cáo này, trong phần này chúng tôi chỉ giới thiệu một vài chức năng chính của phần mềm.



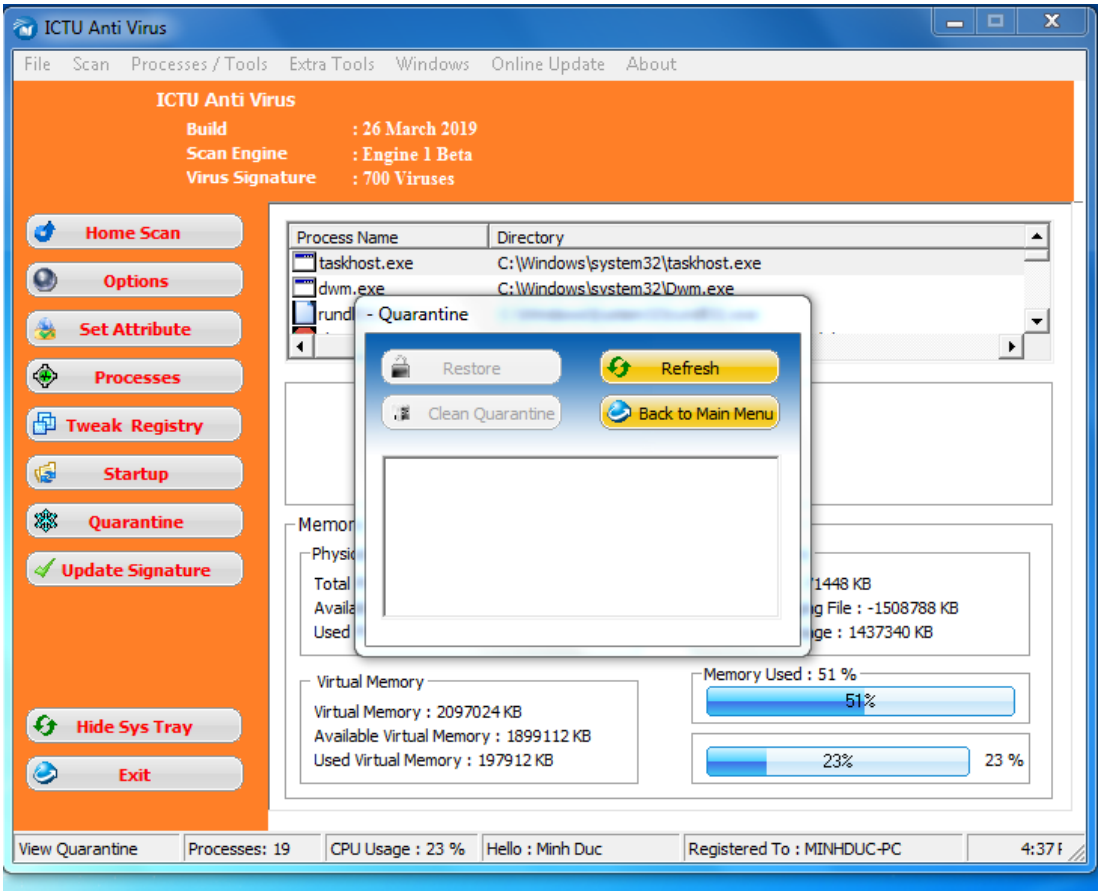
Giao diện chính của phần mềm

Để bắt đầu quá trình quét người dùng chọn vào tùy chọn Home scan, sau đó cần chọn thư mục hoặc tập tin cần quét.



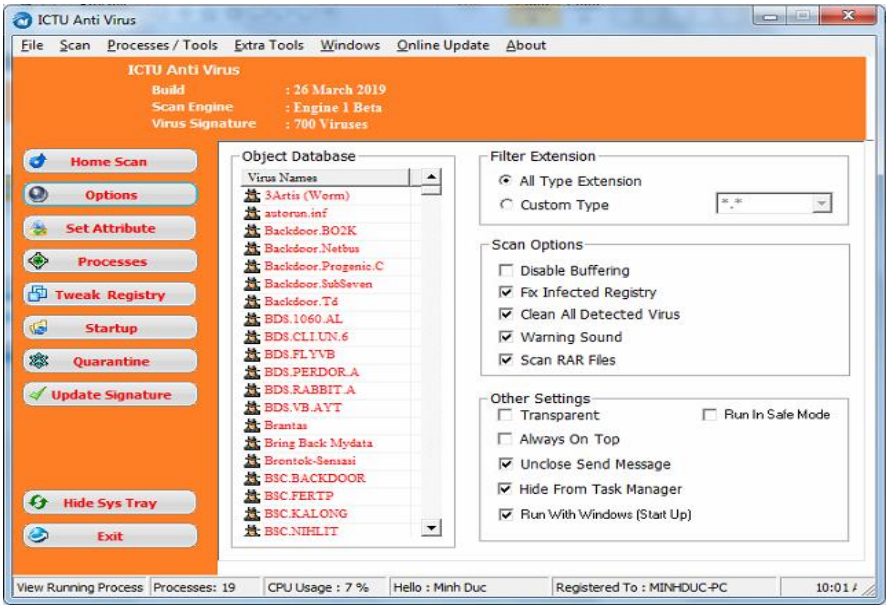
Kết quả quét

Nếu một tệp tin bị nhiệm mã độc, chúng sẽ được đưa vào thư mục cách ly Quarantine

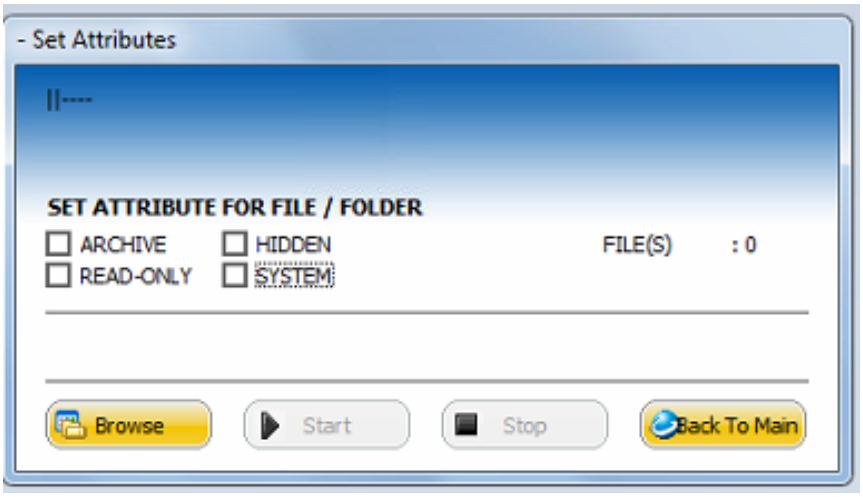


Các tệp tin bị cách ly

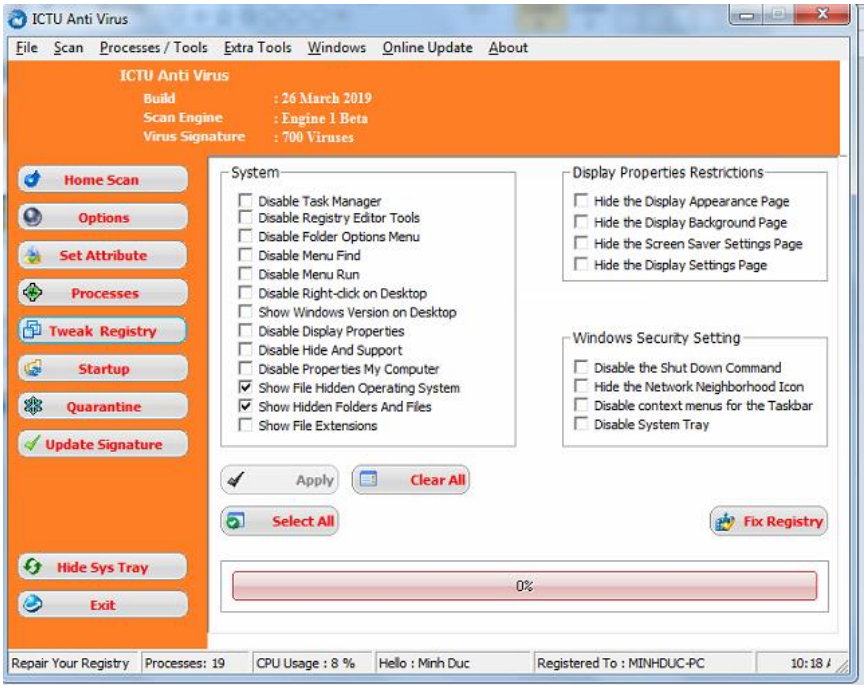
Các tệp bị cách ly có thể được khôi phục hoặc xóa bỏ. Một số hình ảnh về các chức năng khác của phần mềm.



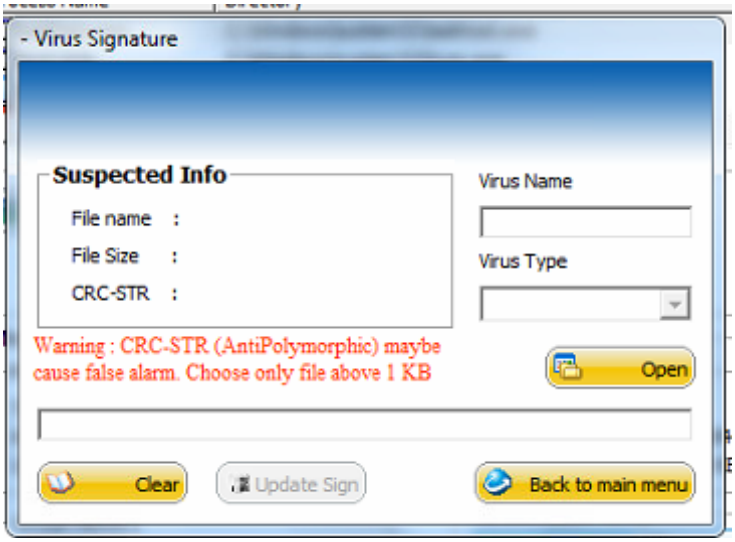
Một số tùy chọn



Thiết lập thuộc tính cho thư mục hoặc tập tin



Một số Tweak registry



Cập nhật cơ sở dữ liệu mã độc