HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

Trần Đình Tân

LUẬN VĂN THẠC SĨ KỸ THUẬT

(Theo định hướng ứng dụng)

HÀ NỘI - 2020

HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

Trần Đình Tân

ỨNG DỤNG THUẬT TOÁN ONE-CLASS SVM TRONG PHÁT HIỆN BOTNET TRÊN CÁC THIẾT BỊ IOT

CHUYÊN NGÀNH : HỆ THỐNG THÔNG TIN

MÃ SỐ: 8.48.01.04

LUẬN VĂN THẠC SĨ KỸ THUẬT

(Theo định hướng ứng dụng)

NGƯỜI HƯỚNG DẪN KHOA HỌC :

TS. NGÔ QUỐC DŨNG

HÀ NỘI - 2020

LỜI CAM ĐOAN

Tôi cam đoan đây là công trình nghiên cứu của riêng tôi.

Các số liệu, kết quả nêu trong luận văn là trung thực và chưa từng được ai công bố trong bất kỳ công trình nào khác.

Hà Nội, ngày tháng 11 năm 2019

TÁC GIẢ LUẬN VĂN

TRẦN ĐÌNH TÂN

LỜI CẢM ƠN

Tôi xin trân trọng cảm ơn các thầy cô trong Khoa công nghệ thông tin đã tạo điều kiện cho tôi một môi trường học tập tốt, đồng thời truyền đạt cho tôi một vốn kiến thức quý báu, một tư duy khoa học để phục vụ cho quá trình học tập và công tác của tôi.

Tôi xin gửi lời cảm ơn đến các bạn trong lớp Cao học Hệ thống thông tin M18CQIS01-B đã giúp đỡ tôi trong suốt thời gian học tập vừa qua.

Đặc biệt, tôi xin được bày tỏ lòng biết ơn sâu sắc đến thầy TS. NGÔ QUỐC DŨNG đã tận tình chỉ bảo cho tôi trong suốt quá trình học tập và nghiên cứu, giúp tôi có nhận thức đúng đắn về kiến thức khoa học, tác phong học tập và làm việc, tạo điều kiện thuận lợi để tôi hoàn thành luận văn này. Cuối cùng, tôi xin được gửi lời cảm ơn tới gia đình, đồng nghiệp, người thân đã động viên, giúp đỡ tôi trong quá trình hoàn thành luận văn.

Hà nội, tháng 11 năm 2019
Tác giả luận văn

MỤC LỤC

DANH MỤC CÁC KÝ HIỆU, CÁC CHỮ VIẾT TẮT

STT	Viết tắt	Tiếng anh	Tiếng việt
1	ARM	Advanced RISC Machine	Kiến trúc vi xử lý dạng RISC cho các môi trường khác nhau
2	IoT	Internet of things	Vạn vật kết nối
3	IRC	Internet Relay Chat	Chat chuyển tiếp Internet
4	MIPS	Microprocessor without Interlocked Pipeline Stages	Kiến trúc vi xử lý không có các giai đoạn đường ống lồng vào nhau
5	OSVM	One Class Support Vector Machine	
6	PPC	PowerPC	Một cấu trúc tập lệnh hướng dẫn máy tính rút gọn được tạo bởi liên minh Motorola Apple IBM
7	SCG	System call graph	Đồ thị lời gọi hàm hệ thống
8	SVM	Support Vector Machine	

DANH MỤC CÁC BẢNG

DANH MỤC CÁC HÌNH

MỞ ĐẦU

1. Lý do chọn đề tài

Trong cuộc cách mạng công nghiệp 4.0, Internet của vạn vật (Internet of Things - IoT) là một xu hướng công nghệ mới đang được phát triển rất mạnh mẽ làm thay đổi cách sống và cách làm việc của con người. Tuy nhiên, càng nhiều thiết bị được kết nối với nhau để chia sẻ thông tin thì đồng nghĩa với việc càng xuất hiện thêm nhiều lỗ hổng bảo mật đe dọa sự an toàn của chính các thiết bị IoT. Bên cạnh đó, nhiều chuyên gia an ninh mạng đánh giá các cuộc tấn công mạng vào các thiết bị IoT sẽ để lại hậu quả nghiêm trọng hơn so với các cuộc tấn công vào hệ thống máy tính thông thường. Theo số liệu tính đến đầu năm 2018 của Kaspersky Lab cho biết tổng số mẫu phần mềm độc hại nhắm đến các thiết bị IoT được họ phát hiện đã lên tới hơn 7.000, trong đó hơn một nửa xuất hiện chỉ trong năm 2017. Hầu hết các cuộc tấn công nhắm vào máy ghi hình kỹ thuật số hoặc máy quay IP (chiếm 63%), và 20% là vào các thiết bị mạng, gồm router, modem ... Khoảng 1% mục tiêu là các thiết bị quen thuộc nhất của người dùng như máy in và thiết bị gia đình thông minh khác[18].

Các mã độc nói chung và mã độc trên các thiết bị IoT nói riêng đều có rất nhiều biến thể vì vậy việc phát hiện rất khó khăn. Việc thu thập mã độc đã và đang được thực hiện thông qua các hệ thống HoneyPot cho các thiết bị IoT như IoTPot, Detux... Tuy nhiên, việc thu thập các tệp tin lành tính để từ đó áp dụng các thuật toán học máy nhằm phân biệt, phát hiện các tệp tin mã độc lại chưa có nhiều. Để thực hiện việc phân biệt giữa các tệp tin mã độc và lành tính trong điều kiện số lượng tệp tin giữa hai lớp mã độc/lành tính chênh lệch lớn thì việc sử dụng các thuật toán học máy 1 lớp trở nên cần thiết. Thuật toán One-class SVM đã được ứng dụng nhiều vào các bài toán phân lớp mã độc và cũng đã được chứng minh có hiệu quả trong việc phát hiện các mã độc thông thường. Từ lý đó và thực tiễn đảm bảo an ninh mạng cho các thiết bị IoT em đề xuất đề tài luận văn: "**Ứng dụng thuật toán One-class SVM trong phát hiện botnet trên các thiết bị IoT**".

2. Tổng quan về vấn đề nghiên cứu

Hiện nay, trên thế giới đã có nhiều công trình nghiên cứu về botnet trên các thiết bị IoT, trong đó điển hình là công trình nghiên cứu của nhóm tác giả Vitor Hugo Bezerra và các thành viên công bố vào năm 2018, với tiêu đề: One-class Classification to Detect Botnets in Iot a devices. Trong công trình nghiên cứu này nhóm tác giả đã xây dựng mô hình phát hiện botnet và chạy thử nghiệm trên thiết thị Rasperrypi, các bước tiến hành như sau: cài đặt công cụ thu thập dữ liệu trên thiết bị IoT; thu thập dữ liệu; chuẩn hóa dữ liệu thu thập; trích xuất đặc trưng; training model; vận hành thử nghiệm. Các kết quả đạt được rất khả quan, tuy nhiên tập dataset của nhóm tác giả chỉ có các mẫu mã độc, không có các mẫu sạch nên tập dataset bị lệch dẫn đến kết quả nhận diện không được cao. Bên cạnh đó tác giả chỉ mới thử nghiệm mô hình trên thiết bị Raspberry pi.

Tại Hội thảo quốc gia lần thứ XX: Một số vấn đề chọn lọc của Công nghệ thông tin và truyền thông năm 2017 diễn ra tại Quy Nhơn, nhóm tác giả Lê Hải Việt và các thành viên đã công bố bài báo: Xây dựng mô hình phát hiện mã độc trên thiết bị định tuyến bằng tác tử. Trong bài báo này, nhóm tác giả mới chỉ đề xuất giải pháp phát hiện botnet trong các thiết bị router mà chưa đề cập đến các thiết bị IoT khác.

3. Mục đích nghiên cứu

Xây dựng và thử nghiệm mô hình phát hiện botnet trên các thiết bị IoT bằng thuật toán One-class SVM.

4. Đối tượng và phạm vi nghiên cứu

Đối tượng nghiên cứu:

- Thuật toán one-class, SVM, one-class SVM;

- Các thiết bị IoT;

- Botnet trên các thiết bị IoT.

Phạm vi nghiên cứu:

- Hiện nay, có rất nhiều chủng loại thiết bị IoT, tuy nhiên, trong phạm vi nghiên cứu của đề tài này chỉ tập trung vào các thiết bị IoT dân dụng. Các thuật toán học máy sẽ sử dụng đặc trưng System-call Graph với bộ dữ liệu từ bộ IoTPot gồm 4000 mẫu IoT botnet và thu thập thêm từ các nguồn khác như: Virusshare,…

5. Phương pháp nghiên cứu

- Phương pháp nghiên cứu lý thuyết: Đọc và phân tích tài liệu về các thuật toán học máy;

- Phương pháp thực nghiệm: Xây dựng và thử nghiệm mô hình áp dụng thuật toán one-class SVM trong phát hiện botnet trên các thiết bị IoT.

6. Nội dung

Cấu trúc của luận văn sẽ bao gồm 3 chương, cụ thể như sau:

CHƯƠNG 1: TỔNG QUAN VỀ MÃ ĐỘC IOT BOTNET VÀ CÁC BIỆN PHÁP PHÁT HIỆN

Chương này sẽ trình bày kiến thức tổng quan về các thuật toán học máy: one-class; SVM; one-class SVM và trình bày về phát hiện botnet trong các thiết bị IoT.

CHƯƠNG 2: XÂY DỰNG MÔ HÌNH PHÁT HIỆN IOT BOTNET

Chương này trình bày về việc áp dụng thuật toán học máy one-class SVM vào trong việc xây dựng mô hình phát hiện botnet trong các thiết bị IoT.

CHƯƠNG 3: THỬ NGHIỆM VÀ ĐÁNH GIÁ

Chương này trình bày về các bước cài đặt mô hình, thử nghiệm, từ kết quả thu được đưa ra những nhận xét, đánh giá.

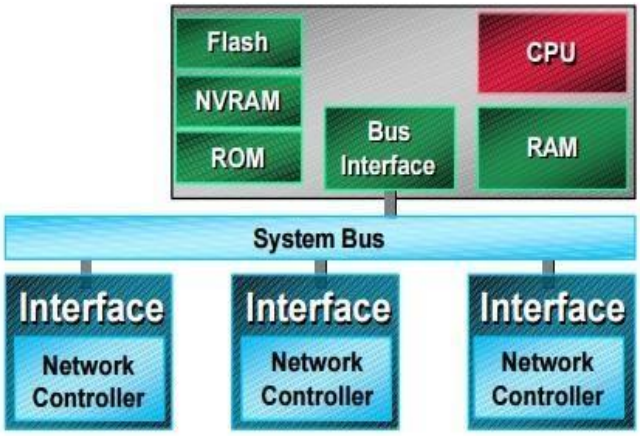
CHƯƠNG 1: TỔNG QUAN MÃ ĐỘC IOT BOTNET VÀ CÁC BIỆN PHÁP PHÁT HIỆN

1.1. Tổng quan về mã độc IoT Botnet

1.1.1. Tổng quan về thiết bị IoT dân dụng

Các thiết bị IoT dân dụng hiện nay phần lớn bao gồm các thiết bị định tuyến, IP Camera và các thiết bị Smartbox-TV. Phần lớn các thiết bị này có cấu trúc phần cứng và phần mềm tương tự nhau nên trong phần này, tác giả lựa chọn trình bày chi tiết về kiến trúc của thiết bị định tuyến.

Thiết bị định tuyến (router) là thiết bị mạng lớp 3 của mô hình OSI (Network Layer) được sử dụng trong việc liên kết giữa hai hoặc nhiều mạng máy tính lại với nhau nhằm chuyển các gói dữ liệu giữa các thiết bị mạng. Cấu trúc của một thiết bị định tuyến được mô tả qua hình 1.1 và gồm các thành phần chính như sau:



Hình 1.1. Cấu trúc của một thiết bị định tuyến

(Nguồn: Internet)

- CPU: Điều khiển mọi hoạt động của bộ định tuyến trên cơ sở các hệ thống chương trình thực thi của hệ điều hành.

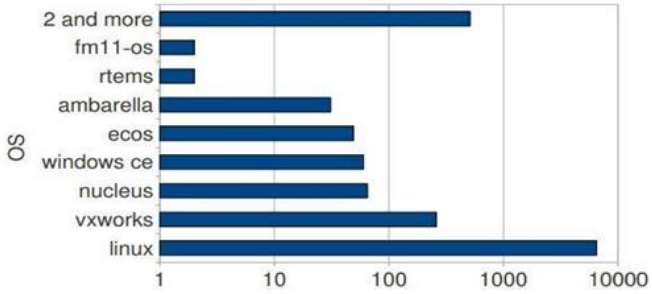
- ROM: Chứa các chương trình tự động kiểm tra và có thể có thành phần cơ bản nhất sao cho bộ định tuyến có thể thực thi được một số hoạt động tối thiểu ngay cả khi không có hệ điều hành hay hệ điều hành bị hỏng.

- RAM: Cấp phát vùng nhớ cho các quá trình như: lưu trữ các bảng định tuyến, các vùng đệm, tập tin cấu hình khi chạy, các thông số đảm bảo hoạt động của bộ định tuyến.

- FLASH: Là thiết bị nhớ có khả năng ghi và xóa, không mất dữ liệu khi mất nguồn. Thông thường, firmware của bộ định tuyến được lưu trữ ở đây. Tùy thuộc các thiết bị định tuyến khác nhau mà hệ điều hành sẽ được chạy trực tiếp từ Flash hay được tải lên RAM trước khi chạy. Tập tin cấu hình cũng có thể được lưu trữ trong Flash.

- NVRAM (None-Volatile RAM): Có chức năng tương tự như FLASH nhưng với khả năng lưu trữ ít hơn. NVRAM thường chứa tập tin cấu hình của thiết bị để đảm bảo khi khởi động, cấu hình mặc định của Thiết bị định tuyến sẽ được tự động nạp về đúng trạng thái đã lưu giữ.

Trên các thiết bị IoT dân dụng như thiết bị định tuyến, IP Camera thì firmware dựa trên nền hệ điều hành nhân Linux được sử dụng phổ biến và rộng rãi. Theo nghiên cứu của Andrei Costin và cộng sự trên 32.356 firmware thì tỉ lệ dựa trên nền tảng Linux lên tới 86%.



Hình 1.2. Nền tảng hệ điều hành phổ biến trên các thiết bị định tuyến

(Nguồn: Andrei Costin)

Cấu trúc của firmware rất đa dạng, phụ thuộc vào chức năng và thiết kế của từng nhà sản xuất. Các firmware có thể được chia thành các kiểu như sau:

- Integrated (apps + OS-as-a-lib): Đây là một bản firmware không đầy đủ, các chức năng và hệ điều hành được xây dựng như một thư viện chứ không có đầy đủ các thành phần cần thiết như trong bản Full-blown.

- Partial updates (apps or libs or resources or support): Loại firmware này chỉ chứa các tập tin dùng trong việc cập nhật cho bản firmware cần nâng cấp.

1.1.2. Tổng quan về mã độc Botnet trên thiết bị IoT dân dụng

Dựa trên các nghiên cứu của Kishore, Costin[11][12][13] và cộng sự các loại mã độc botnet trên các thiết bị IoT dân dụng có các loại sau đây:

- **Linux.Hydra**: Là mã độc đầu tiên lây nhiễm trên các thiết bị IoT (gọi tắt là mã độc IoT). Linux.Hydra xuất hiện vào năm 2008, ở dạng mã nguồn mở nhằm mục đích tấn công các thiết bị dựa trên nền tảng kiến trúc MIPS. Pha thực hiện khai thác của Linux.Hydra dựa trên tấn công từ điển vào các thiết bị định tuyến D-Link có lỗ hổng về xác thực. Khi lây nhiễm thiết bị thành công, mã độc Linux.Hydra sẽ biến thiết bị trở thành một phần trong mạng botnet dựa trên IRC, nhưng chỉ thực hiện tấn công SYN Flood. Mặc dù nhiều nghiên cứu đã chỉ ra rằng Linux.Hydra có khả năng tấn công UDP Flood, nhưng mã nguồn được công bố thì không cho thấy khả năng này.

- **Psyb0t**: Tương tự như mã độc Linux.Hydra, mã độc Psyb0t được phát hiện lây nhiễm trên các thiết bị định tuyến, modem DSL có vi xử lý MIPS litteendian chạy firmware Mipsel Linux vào năm 2009 bởi nhà nghiên cứu bảo mật Terry Baume người Úc. Psyb0t đã lây nhiễm hơn 100.000 thiết bị và hoạt động dựa trên cơ chế nhận lệnh từ máy chủ C&C qua giao thức IRC. Phương thức chính để lây nhiễm thiết bị IoT của Psyb0t là sử dụng truy cập Telnet và SSH bằng cách tấn công vét cạn các khả năng đăng nhập với danh sách tài khoản được định nghĩa trước gồm 6.000 tên đăng nhập và 13.000 mật khẩu. Sau khi lây nhiễm, Psy0t sẽ chặn truy cập thiết bi

định tuyến qua một số cổng TCP như 22, 23, 80. Mã độc Psyb0t và có khả năng thực hiện tấn công UDP Flood, ICMP Flood.

- **Chuck Norris**: Ngay khi mã độc botnet Psyb0t được tạo ra, một mẫu mã độc mới đã được phát triển và trở thành đối thủ cạnh tranh trong năm 2010, được gọi là mã độc Chuck Norris. Mã độc này có rất nhiều điểm tương đồng với mã độc Psyb0t, vì thế đây có thể là mã độc tiến hóa của Psyb0t. Khả năng tấn công từ chối dịch vụ bằng các kỹ thuật UDP Flood, ACK Flood, SYN Flood. Mã độc Chuck Noris là một loại mã độc IRC bot được phát hiện lây nhiễm thiết bị định tuyến và modem DLS.

- **Tsunami/Kaiten**: Tsunami còn có thể thực hiện tấn công bằng một số kỹ thuật phức tạp như HTTP Layer 7 Flood, TCP XMASS. Mã độc Tsunami là mã độc IRC bot, hỗ trợ việc thực hiện nhiều câu lệnh và chỉnh sửa thông tin cấu hình máy chủ DNS trên thiết bị đã lây nhiễm khiến cho lưu lượng từ thiết bị IoT được chuyển hướng tới máy chủ điều khiển của kẻ tấn công. Tùy thuộc vào các biến thể của mã độc mà nó có thể chỉnh sửa vị trí lưu trữ các tập tin /etc/init.d/rc.local nhằm tự động thực thi những tập tin mã độc mỗi khi người dùng đăng nhập hoặc tại thư mục /etc/rc.d/rc.local để đảm bảo các tập tin thực thi khi hệ thống khởi động. Một khi đã cài đặt thì mã độc Tsunami sẽ tham gia vào một kênh trao đổi thông tin IRC đã được nhúng trong mã nguồn của mã độc để nhận các chỉ thị của kẻ tấn công từ xa. Bên cạnh khả năng thực hiện tấn công từ chối dịch vụ, mã độc có thể ngắt tiến trình, tải và thực thi các tập tin, giả mạo địa chỉ IP của những thiết bị dễ bị tổn thương.

- **Aidra/LightAidra/Zendran**: Xuất hiện trong khoảng 2012, đây là 3 loại mã độc có nhiều phần mã nguồn tương tự nhau vì thế có thể ghép vào chung một loại mã độc. So sánh với những loại mã độc đã trình bày trước thì mã độc này phức tạp hơn vì chúng có thể biên dịch dựa trên nhiều kiến trúc khác nhau như MIPS, ARM, PPC. Mã độc lây nhiễm và đưa thiết bị vào mạng botnet dựa trên IRC, có khả năng thực hiện một số tấn công cơ bản SYN Flood và ACL Flood.

Theo phân tích của hãng bảo mật Symantec thì mã độc Lightaidra lây nhiễm trên các thiết bị IoT có nền tảng Linux, khai thác lỗ hổng CVE-2014-6271 và được

phân loại là sâu, trojan. Mã độc Lightaidra lây lan bằng cách dò quét địa chỉ IP công cộng để tìm kiếm các thiết bị sử dụng dịch vụ telnet, các tài khoản đăng nhập mặc định hoặc không đặt mật khẩu. Mã độc Lightaidra có thể nhận các lệnh điều khiển từ các địa chỉ: irc.pollo.org, 178.79.183.247, 192.3.205.154, 168.235.156.117. Sâu mã độc Lightaidra thực hiện truyền tin với máy chủ C&C đã được nhúng trực tiếp trong mã nguồn của Lightaidra như gửi thông tin đăng nhập thành công, nhận lệnh khởi tạo các cuộc tấn công từ chối dịch vụ sử dụng các kỹ thuật TCP flood, UDP flood, DNS flood… Quy trình hoạt động của mã độc LightAidra/Aidra như sau: Đầu tiên, mã độc thử kết nối tới cổng Telnet. Khi kết nối thành công, mã độc có thể sử dụng kết hợp một số tài khoản mặc định như root/root, root/admin… Sau khi đăng nhập thành công, mã độc LightAidra/Aidra sẽ tải về tập tin getbinaries.sh và thực thi tập tin đó. Chức năng của tập này đơn giản là:

- Xóa những tập tin nhị phân mã độc cũ để đảm bảo chỉ có duy nhất mã độc LightAidra/Aidra chạy trên thiết bị;
- Tải về các tập tin nhị phân của mã độc và thực thi chúng;
- Thay đổi mật khẩu;
- Chỉnh sửa cài đặt trong tường lửa sử dụng Iptable;
- Xóa tập tin getbinaries.sh.

Sau khi thực thi các tập tin nhị phân đã tải về, mã độc LightAidra/Aidra thực hiện kết nối thiết bị tới hệ thống mạng botnet và máy chủ IRC. Tất cả các tập tin nhị phân mã độc LightAidra/Aidra được tải về thư mục /var/run, /var/tmp trên thiết bị (nếu thiết bị sử dụng x86 thì sẽ lưu trong thư mục /tmp) , đây là những thư mục sẽ bị xóa khi thiết bị khởi động lại. Trong quá trình phân tích nếu phát hiện những tập tin thực thi lưu trong thư mục /var/run thì đó có khả năng cao là tập tin mã độc bởi các tập tin thực thi không được lưu trữ tại /var/run trên một hệ thống bình thường.

Thông thường, việc khởi động lại thiết bị đủ để dọn dẹp thiết bị bởi vì đối với các nền tảng hệ thống nhúng thì hệ thống tập tin (tệp tinsystem) được mount dạng chỉ

đọc (read-only), vì thế các thư mục /tmp và /run lưu trữ trong RAM (Random Access Memory) được sử dụng. Mọi thông tin, dữ liệu trên /var và /tmp sẽ bị xóa khi thiết bị khởi động lại. Tuy nhiên, các thiết bị IoT thường ít khi khởi động lại và thay đổi mật khẩu nên việc lây nhiễm lại của mã độc diễn ra nhanh chóng.

- **Spike/Dofloo/MrBlack/Wrkatk/Sotdas/AES.DdoS**: Sau sự xuất hiện mạnh mẽ của các loại mã độc tương tự Linux.Hydra thì vào năm 2014 một dòng mã độc mới đã xuất hiện với nhiều loại mã độc như Spike, Dofloo, nhưng rất khó có thể phân biệt giữa các mã độc đó. Tuy nhiên, điểm khác biệt trong kiến trúc mạng botnet so với những dòng mã độc trước đây là thường sử dụng IRC-based thì dòng mã độc này sử dụng Agent-handler. Hơn nữa, một cơ chế bảo đảm sự bền vững bằng cách giả mạo tập tin etc/rc.local, nhằm vẫn tồn tại khi thiết bị khởi động lại. Bên cạnh đó, đặc trưng nổi bật của mã độc này là sử dụng luồng SendInfo để tính toán hiệu năng của thiết bị và gửi về máy chủ CNC, khi đó hacker có thể triển khai mật độ thực hiện tấn công từ chối dịch vụ phân tán trên mỗi thiết bị bot một cách hiệu quả.

- **Bashlite/Lizkebab/Torlus/Gafgyt**: Xuất hiện vào năm 2014, có nhiều đặc điểm tương tự như dòng mã độc Spike. Cụ thể, giao thức truyền tin dựa trên IRC đã được chỉnh sửa vì thế kiến trúc mạng botnet hoàn toàn không phụ thuộc vào máy chủ IRC do đó có thể coi mạng botnet này dựa trên kiến trúc AgentHandler. Các hình thức tấn cống từ chối dịch vụ phân tán cũng dựa trên một số kỹ thuật đơn giản như SYN, UDP, ACK Flood.

- **Elknot/BillGates Botnet**: được phát hiện vào năm 2015, đây là mã độc được sử dụng khá phổ biến tại Trung quốc để thực hiện tấn công từ chối dịch vụ phân tán (DRDOS). Mục tiêu chính của mã độc này là các thiết bị SOHO có kiến trúc vi xử lý MIPS, ARM.

- **XOR.DdoS**: Trong năm 2015, một làn sóng mã độc khai thác lỗ hổng Shellshock có tên là XOR.DdoS đã âm thầm lây nhiễm nhiều thiết bị IoT. Mã độc này có khả năng thực hiện nhiều loại tấn công từ chối dịch vụ phân tán như SYN, UDP, DNS, TCP Flood. Theo ghi nhận lại trong báo cáo của Akamai thì vào 9/2015

mạng botnet này đã thực hiện tấn công với DNS Flood ở mức 30 triệu truy vấn/giây, kết hợp với SYN Flood ở mức 140 Gbps. Đặc biệt, đúng như tên gọi của mã độc này, các kết nối với máy chủ C&C và mã nguồn của mã độc đều sử dụng mã hóa XOR.

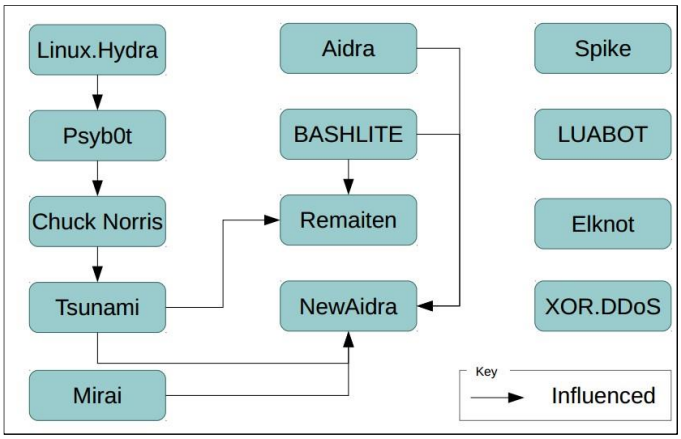
- **Remaiten/KTN-RM**: Xuất hiện trong năm 2015 và được biết đến khá rộng rãi như mã độc Mirai. Remaiten kết hợp các đặc điểm chính của hai loại mã độc là Tsunami và BASHLITE. Không như BASHLITE, mã độc Remaiten dựa trên kiến trúc IRC-based. Một điểm đặc trưng của mã độc IoT hiện nay là đều hỗ trợ dịch mã nguồn thành tệp khả chạy trên nhiều kiến trúc khác nhau. Theo các nhà nghiên cứu của hãng bảo mật ESET, mã độc Remaiten là mã độc IoT có những tính năng kết hợp giữa mã độc Tsunami và Bashlite. Remaiten thực hiện lây nhiễm thiết bị IoT chạy nền tảng Linux bằng phương pháp tấn công vét cạn dựa trên danh sách các tài khoản đăng nhập, mật khẩu thường xuyên được sử dụng. Các máy chủ C&C kết nối tới các thiết bị đã bị lây nhiễm bằng kênh truyền tin IRC. Remaiten có cấu trúc tinh vi, phức tạp hơn Tsunami và Bashlite ở điểm có thể tùy biến dựa trên kiến trúc thiết bị và phương thức tấn công mà mã độc thực hiện.

- **NewAidra/IRCTelnet**: được biết đến với tên gọi là Linux.IRCTelnet, mã độc này được kết hợp dựa trên mã nguồn gốc Aidra, giao thức của Kaiten IRC based, mã dò quét/mã lây nhiễm của BASHLITE và bộ từ điển tấn công của Mirai. Tất cả các thiết bị nhúng dựa trên các kiến trúc chuẩn đều có thể bị lây nhiễm bởi mã độc này và miền kỹ thuật tấn công lớn như TCP XMAS, TCP Flood. Hiện nay mã độc NewAidra được coi như là đối trọng lớn của Mirai trong hệ thống lây nhiễm thiết bị IoT.

- **Darlloz**: Hãng Symantec đã phát hiện ra một sâu mã độc có tên gọi là Darlloz, năm 2013, mã độc này khai thác lỗ hổng PHP có mã CVE-20121823. Tương tự như LightAidra, mã độc Darlloz hỗ trợ nhiều nền tảng kiến trúc như x86, ARM, MIPS, PPC… Nhằm chặn người dùng truy cập tới thiết bị IoT đã bị lây nhiễm thông qua Telnet, mã độc ngăn chặn các lưu lượng kết nối bằng telnet với cấu hình iptable và kết thúc tiến trình của dịch vụ telnet trên thiết bị. Một đặc điểm của sâu mã độc

Darlloz là sẽ nhắm tới ngăn cản sự lây nhiễm sâu mã độc LightAidra. Mã độc LightAidra lưu trữ các ID tiến trình thực thi của nó trong nhiều tập tin như /var/run/.lightpid, /var/run/.aidrapid và /var/run/lightpid. Khi đó, mã độc Darlloz sẽ tìm cách kết thúc các tiến trình có PID được lưu trữ trong các tập tin trên và xóa các tập tin của LightAidra khỏi thiết bị đã lây nhiễm hay như chặn các cổng kết nối mà mã độc LightAidra sử dụng.

- **Mirai**: Là mã độc khá nổi bật trong những năm qua, được sử dụng để thực hiện các vụ tấn công từ chối dịch vụ phân tán có tính quy mô rất lớn.



Hình 1.3. Sự tương quan giữa một số mã độc IoT Ddos

(Nguồn: DDoS-Capable IoT Malwares: Comparative Analysis and Mirai Investigation - Michele De Donno)

Quan sát mối quan hệ tương quan giữa các mã độc IoT thấy rằng Linux.Hydra là mã độc đầu tiên có khả năng thực hiện tấn công từ chối dịch vụ phân tán và mã nguồn của nó được tiến hóa thông qua 3 loại mã độc khác nhau. Điều này cho thấy Tsunami được tiến hóa khá nhiều từ Linux.Hydra nhưng một phần mã của nó được sử dụng để phát triển một nhánh mã độc mới là Remaiten và NewAidra – một trong những mã độc xuất hiện nhiều nhất hiện nay. Hình trên cũng chỉ ra được những mã độc trước đây khá lâu hầu hết không liên quan đến mã độc khác.

Bên cạnh đó, một đặc điểm có thể thấy các mã độc IoT trước đây chỉ tập trung khai thác các thiết bị kiến trúc MIPS nhưng hiện nay mã độc đã lây nhiễm lên các

thiết bị kiến trúc ARM, PowerPC. Hơn nữa, các hình thức tấn công từ chối dịch vụ phân tán của mã độc IoT cũng ngày càng phát triển, từ những hình thức tấn công giản đơn như SYN Flood cho đến các hình thức tấn công nâng cao như GRE IP Flood, GRE ETH Flood. Tuy nhiên, hầu hết kỹ thuật tấn công đó đều thuộc loại tấn công Flood bởi số lượng lớn các thiết bị IoT có lỗ hổng bị khai thác dễ dàng trở thành một bot trong mạng botnet và tấn công Flood yêu cầu kỹ năng lập trình cơ bản với những dòng mã giản đơn.

1.2. Tổng quan các phương pháp phát hiện mã độc

Các phương pháp phát hiện mã độc có thể chia thành hai nhóm phương pháp chính đó là phân tích tĩnh và phân tích động. Phân tích tĩnh (Static) là phương pháp phân tích, kiểm tra các phần mềm độc hại dựa trên các đặc trưng của chúng mà không cần thực thi. Phân tích động (Dynamic) là phương pháp mà phần mềm độc hại sẽ được thực thi nhằm giám sát và phát hiện các hành vi bất thường của chúng. Ngoài hai phương pháp phổ biến trên, một số các nghiên cứu gần đây cũng tập trung vào phương pháp lai (Hybrid-based). Với mục đích kết hợp các điểm mạnh của cả hai phương pháp tĩnh và động giúp việc phân tích chính xác và hiệu quả hơn.

1.2.1. Phân tích tĩnh

Phương pháp phân tích tĩnh mã độc dựa trên những đặc trưng của các tập tin mà không cần thực thi chúng để phát hiện mã độc. Những nghiên cứu gần đây trong hướng tiếp cận này thường tập trung vào phân tích chuỗi byte, sử dụng khai phá dữ liệu và học máy thay vì sử dụng những phương pháp thu thập đặc trưng truyền thống. Phương pháp phân tích mã trung gian (bytecode) dựa trên Entropy cũng được đề xuất để phát hiện các kỹ thuật gây rối, đóng gói, mã hóa những đoạn mã độc nhúng trên các tập tin. Những đặc trưng khác trong phân tích tĩnh thường được sử dụng như: tiêu đề tập tin (header), các lời gọi hàm (system calls) API, PSI (Printable Strings Information), FLF (Function Length Frequency), các thư viện liên kết,...

Thông tin luồng điều khiển trong tập tin (Control-Flow Information) cũng là hướng tiếp cận hiệu quả trong phát hiện mã độc. Ví dụ, ngữ nghĩa hình thức (formal

semantics) có thể được sử dụng để miêu tả những hành vi bất thường được cho là độc hại. Việc sử dụng các tiếp cận theo luồng điều khiển được sử dụng phổ biến trong các nghiên cứu phát hiện các đoạn mã có ngữ nghĩa tương đương trong các tập tin.

Một hướng tiếp cận tương tự sử dụng đồ thị luồng điều khiển (Control-Flow Graph) để phát hiện mã độc máy tính như worm, spyware trên các trình duyệt web, và mã độc siêu đa hình. Đồ thị luồng điều khiển được định nghĩa là đồ thị có hướng G = (V, E), trong đó các đỉnh $(u, v) \in V$ đại diện cho các khối và cạnh $e \in E$: $u \rightarrow v$ biểu thị khả năng luồng điều khiển từ u đến v. Để xây dựng được một đồ thị CFG thường tuân theo quy trình sau: gỡ rối, dịch ngược và diễn giải dữ liệu (interpretation). Kết thúc quy trình này, một đồ thị luồng điều khiển CFG hoặc đồ thị luồng dữ liệu (DFG - Data Flow Graph) được xây dựng. Các đồ thị này thể hiện quy trình thực thi và hành vi của mã độc. Tuy nhiên, thách thức chính của cách tiếp cận này là xây dựng và phân tích được quy trình thực thi, hành vi từ các mã nhị phân đối với các tập tin lớn và phức tạp. Để làm được điều này đòi hỏi một quy trình phức tạp và nhiều bước xử lý khác nhau.

- Ưu điểm: Các phương pháp tĩnh có ưu điểm lớn nhất là có thể phân tích một cách chi tiết các tập tin và đưa ra được cái nhìn tổng quát về tất cả các khả năng kích hoạt của chúng. Trong quá trình phân tích tĩnh vì không cần phải thực thi mã độc nên không cần phải thiết lập các thiết bị ngoại vi như trong môi trường hoạt động thực tế. Bên cạnh đó, vì không cần thực thi mã độc nên không thực hiện hành vi nào gây hại cho hệ thống. Đối với các tập tin nhỏ và tường minh thì các phương pháp tĩnh có tốc độ quét nhanh, ổn định và có tính lặp lại so với phân tích động. Điều này được minh chứng bằng việc khi áp dụng những thuật toán phân tích mới thì kết quả tương đồng và ít thay đổi giúp cho việc nghiên cứu và phát triển các thuật toán mới dễ dàng hơn.

- Hạn chế: Hạn chế lớn nhất của phân tích tĩnh là khi mã độc sử dụng các kỹ thuật gây rối phức tạp (obfuscations) như sắp xếp lại câu lệnh, chèn mã lệnh vô nghĩa, khi đó rất khó có thể thu thập được thông tin ngữ nghĩa chính xác cho việc phân tích. Khi một chương trình trở nên phức tạp hơn, bị gây rối nhiều hơn, thì CFG của nó sẽ

rất phức tạp. Nếu như không có những kỹ thuật gỡ rối đủ mạnh thì việc phân tích tĩnh trong trường hợp này gần như là bất khả thi. Hạn chế lớn thứ hai là công nghệ dịch ngược các bản mã nhị phân thành bản mã bậc cao còn nhiều hạn chế. Điều này là do hầu hết mã độc hiện nay được viết bằng ngôn ngữ bậc cao chứ không viết bằng các ngôn ngữ máy bậc thấp như trước. Điều này dẫn đến việc mã độc dễ dàng sử dụng các các kỹ thuật làm rối và siêu đa hình trong mã nguồn để gây ra những thay đổi lớn trong mã nhị phân. Theo Andreas Moser, nếu chỉ sử dụng phân tích tĩnh thì không đủ để phát hiện và phân lớp mã độc trong thực tế mà nên sử dụng phân tích tĩnh như một phần bổ xung cho phân tích động.

1.2.2. Phân tích động

Bên cạnh kỹ thuật phân tích tĩnh, ngày nay kỹ thuật phân tích động được áp dụng khá phổ biến để khắc phục một số hạn chế của phân tích tĩnh. Kỹ thuật phân tích động thông qua việc thực thi các mã chương trình và quan sát các hành vi của nó để phát hiện có hay không mã độc. Phân tích động hay còn gọi là phân tích dựa trên bất thường/hành vi sử dụng các tri thức của chuyên gia để tìm ra những hành vi của mã độc hay còn gọi những hành vi bất thường. Kỹ thuật này dựa trên nguyên lý làm việc sử dụng tập luật được coi là bình thường để quyết định một chương trình có cố ý vi phạm những tập luật đó không. Những hành vi vi phạm tập luật đó được coi là bất thường.

Tuy nhiên, đối với phân tích mã độc dựa trên hành vi thì việc xác định thế nào là một hành vi bất thường (abnormal) là không đơn giản. Ví dụ, để phân tích các luồng thông tin trên mạng là bất thường có thể dựa trên các tính chất như: Bất thường về giao thức, bất thường về thống kê, bất thường về ứng. Chính vì vậy, phương pháp này đem lại tỉ lệ dương tính giả khá cao (false positive). Thực tế, có nhiều chương trình lành tính nhưng khi thực thi lại có thể có nhiều hành vi bị liệt vào bất thường như mã độc. Yêu cầu quan trọng nhất trong phân tích động là xây dựng một môi trường an toàn có khả năng kiểm soát và quan sát hành vi của các tập tin khi thực thi

và tránh mọi lây nhiễm sang môi trường thực tế. Các môi trường an toàn được sử dụng phổ biến trong phát hiện mã độc hiện nay là:

- Máy ảo (virtual machine): Máy ảo giống như một máy tính (guess) bên trong một máy tính khác (host) và được chạy cách ly với hệ điều hành chủ từ đó tạo ra một môi trường an toàn để thực thi mã độc. Các máy ảo đều có chức năng lưu lại trạng thái hoạt động (snapshot) và có thể quan sát, thu thập các hành vi của tập tin thực thi. Tuy nhiên, hạn chế của phương pháp sử dụng máy ảo là nhiều loại mã độc có khả năng phát hiện được môi trường thực thi vì thế nhiều mã độc nếu chạy trên môi trường ảo sẽ không thể hiện hết đặc trưng hành vi của chúng.

- Sandbox: Là công cụ tạo ra môi trường an toàn khi thực thi mã độc mà không sợ ảnh hưởng tới hệ thống thực. Điểm khác biệt giữa Sandbox và Máy ảo là khả năng kiểm soát chặt chẽ các tài nguyên, cấp quyền hoạt động, các khả năng truy cập mạng, quan sát hệ thống, gỡ lỗi (debug) hay đọc/ghi tệp tin từ các thiết bị trong quá trình thực thi cho cả chương trình hoặc từng đoạn mã.

Dựa trên các công cụ trên mà nhà nghiên cứu T. Ronghua trích xuất các đặc trưng của lời gọi hệ thống (system-call) trong quá trình thực thi của các tập tin, sau đó áp dụng các thuật toán nhận dạng mẫu và phương pháp thống kê để chỉ ra sự khác biệt giữa các tập tin mã độc và lành tính. Cùng hướng nghiên cứu tương tự, P. V. Shijo và A. Salim đã đề xuất phương pháp phát hiện mã độc dựa trên lời gọi hệ thống (system-call) kết hợp với một số tham số của từng hàm truyền vào và sử dụng đó như một đặc trưng để phát hiện mã độc. Hay như hướng tiếp cận tìm sự khác biệt giữa các lời gọi API của mã độc và tập tin lành tính khi phân tích đặc trưng hành vi sử dụng log của nhiều lời gọi API. Mỗi lời gọi API trong danh sách log thu được sẽ coi như một đặc trưng và chuyển thành các đặc trưng vector. Phương pháp này lọc trực tiếp các hàm hệ thống mà mã độc sẽ sử dụng từ chính tập tin thực thi của mã độc.

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. Tuy nhiên, phương pháp này có một số ưu điểm, hạn chế như sau:

- Ưu điểm: Phương pháp phân tích động có hiệu quả và độ chính xác, cho phép xác định nhanh chóng và tổng quát về mã độc được phân tích thông qua các hành vị bộc lộ của chúng. So với phương pháp phân tích tĩnh trong việc dịch ngược, gỡ rối (deobfuscation) thì phương pháp động cho phép phân tích dễ dàng ngay cả với những mã độc có cấu trúc, mã nguồn phức tạp. Ngoài việc giám sát được hành vi cụ thể, thì phương pháp động có thể thu thập được những thông tin như: các giá trị thanh ghi sử dụng, các tập tin được viết, cấp phát bộ nhớ… và những thông tin đó có thể trực tiếp sử dụng trong việc đánh giá các nguy cơ lây nhiễm mã độc.

- Hạn chế: Mặc dù có những ưu điểm như đã nêu, phân tích động chỉ có thể giám sát đơn luồng thực thi. Điều này đã được T. Ronghua chứng minh trong công bố của mình rằng khi các điều kiện môi trường ảnh hưởng trực tiếp đến việc kích hoạt mã độc như time-bomb, bot,… thì phương pháp động không thể giám sát hết các hành vi tiềm tàng của chúng. Việc giám sát được tất cả các khả năng thực thi của mã độc trong phân tích động đòi hỏi nhiều thời gian với dữ liệu ghi nhận là rất lớn. Do đó, khả năng xây dựng sơ đồ quá trình xử lý sẽ gặp nhiều hạn chế, không như phân tích tĩnh vì mã nguồn chứa tất cả các khả năng thực thi có thể của mã độc nên khả năng đánh giá sẽ cao hơn phân tích động. Phân tích động có thể dẫn tới làm lộ quá trình phát hiện và phân tích mã độc. Mã độc có thể phát hiện được đang bị phân tích, sau đó gửi thông tin này về kẻ phát tán mã độc.

Ngoài ra phân tích động cũng có thể gây nguy cơ mất an toàn cho mạng và hệ thống, do chúng ta chưa biết hết được mã độc có những module nào và khả năng của nó đến đâu. Điều này đến từ việc khó khăn trong xây dựng, thiết kế một môi trường mô phỏng cho phân tích tất cả các loại mã độc. Với những hạn chế như vậy thì một số nghiên cứu gần đây cũng tìm các kết hợp các điểm mạnh của hai phương pháp tĩnh và động để có được một phương pháp lai hiệu quả hơn.

1.2.3. Phân tích lai

Cả phân tích tĩnh và phân tích động đều có những hạn chế nhất định. Vì thế phân tích tĩnh và phân tích động đều có thể bổ trợ lẫn nhau. Vì vậy, các nghiên cứu

phương pháp lai hiện nay tiếp cận theo hướng phân tích tĩnh trước sau đó tiến hành dò quét động để bổ sung các thông tin nhằm xác định mã độc hoặc sử dụng phân tích động để thực hiện bóc tách mã độc rồi sử dụng phân tích tĩnh. Cụ thể như sau:

- Kevin A. Roundy: Tiếp cận theo hướng lai giữa phân tích tĩnh và phân tích động để xây dựng và duy trì CFG và DFG, cụ thể là sử dụng phân tích động để thực hiện unpack và sau đó có thể chỉnh sửa mã nguồn trước khi thực thi.

- P.V.Shijo: Đề xuất hướng tiếp cận tích hợp phân tích tĩnh và phân tích động vào phát hiện mã độc cụ thể là sử dụng phân tích tĩnh để thu thập các chuỗi không mã hóa trong các tập tin nhị phân thực thi (PSI) và thu thập các chuỗi lời gọi hệ thống API, từ đó kết hợp 2 đặc trưng trên để tạo ra một vector đặc trưng tích hợp nhằm phát hiện mã độc hiệu quả hơn.

- Rafiqul Islam: Đề xuất phương pháp kết hợp các đặc trưng phân tích tĩnh và phân tích động thành một kiểm thử tích hợp thay vì chỉ sử dụng riêng các đặc trưng. Cụ thể là thu thập thông tin về các lời gọi API (thu thập được từ phân tích động) và 2 đặc trưng thu thập được từ phân tích tĩnh là hàm độ dài tần số (FLF - Function length frequency), PSI (Printable String Information). Sau đó kết hợp các thông tin trên thành 1 vector đặc trưng tích hợp.

- Theo Zheng Yan: Tiếp cận theo hướng xây dựng tập các mẫu độc hại và mẫu chương trình lành tính, trong khi đó đối với những mã độc chưa biết đến thì sử dụng phân tích động để thu thập dữ liệu lời gọi hệ thống. Sau đó tiến hành đối sánh những thông tin thu được từ phân tích động với tập mẫu gồm các đặc trưng của mã độc và tập tin lành tính để xác định đó có phải mã độc không. Tuy nhiên hạn chế ở nghiên cứu này là phải thường xuyên thu thập các mẫu mã độc và các chương trình lành tính, thứ hai là năng lực tính toán còn nhiều hành chế. Nhưng điểm quan trọng trong nghiên cứu này là tập mẫu các đặc trưng chứa cả những đặc trưng của mã độc và của cả những tập tin lành tính.

Ngoài ra, BitBlaze Binary Analysis Platform là một nghiên cứu có sự hợp nhất giữa phân tích tĩnh và phân tích động, giữa xử lý đặc trưng với mô phỏng toàn bộ hệ

thống và các instrumentation nhị phân. BitBlaze sử dụng framework VINE dùng cho phân tích tĩnh bằng cách chuyển các mã assembler thành dạng ngôn ngữ trung gian (intermediate language) và framework dùng cho phân tích động TUME có khả năng mô phỏng, thực thi toàn bộ hệ thống để giám sát và theo dõi các hành vi (instrumentation) của các tập tin nhị phân.

1.3. Tổng quan về học máy

1.3.1. Các khái niệm cơ bản

1.3.1.1. Khái niệm học máy

Trí tuệ nhân tạo là một trong những đốt phá công nghệ của cuộc cách mạng công nghiệp lần thứ 4. Trí tuệ nhân tạo đang dần len lỏi vào mọi mặt của cuộc sống của con người từ công sở, nơi ở cho đến các địa điểm giải trí. Trong đó học máy là một tập con của trí tuệ nhân tạo. Khái niệm học máy (machine learning) được đề cập đến lần đầu tiên vào năm 1959 bởi Arthur Samuel. Ông là người tiên phong trong lĩnh vực chơi game trên máy tính và trí tuệ nhân tạo. Từ đó, Tom M. Mitchell đã đưa ra một khái niệm chính thức về học máy và đã được trích dẫn rộng rãi trong lĩnh vực học máy: "Một chương trình máy tính được gọi là học từ kinh nghiệm E để hoàn thành nhiệm vụ T, với hiệu quả được đo bằng phép đánh giá P, nếu hiệu quả của nó khi thực hiện nhiệm vụ T, khi được đánh giá bởi P, cải thiện theo kinh nghiệm E".

Các nhiệm vụ T trong học máy thường được mô tả thông qua việc một hệ thống học máy xử lý một điểm như thế nào. Ví dụ: Trong bài toán phân loại ảnh thì mỗi bức ảnh là một điểm dữ liệu, trong bài toán phát hiện mã độc thì một tệp tin thực thi là một điểm dữ liệu,… Một điểm dữ liệu bao gồm nhiều đặc trưng khác nhau và mỗi đặc trưng được biểu diễn bằng một con số. Và trong học máy thường biểu diễn một điểm dữ liệu bằng một vector $x \in \mathbb{R}^d$ trong đó mỗi phần tử x_i là một đặc trưng, vector này được gọi là vector đặc trưng. Hiện nay, rất nhiều nhiệm vụ phức tạp có thể giải quyết được bằng học máy như: nhiệm vụ phân lớp, hồi quy, phân nhóm, ….

Để có thể kiểm tra được năng lực của một thuật toán học máy, chúng ta cần phải có các phép đánh giá có thể đo đạc được kết quả. Thông thường, khi thực hiện một thuật toán học máy, dữ liệu sẽ được chia thành hai thành phần: tập huấn luyện và tập kiểm thử. Tập huấn luyện sẽ được dùng để tìm các tham số mô hình. Tập kiểm thử được dùng để đánh giá năng lực của mô hình tìm được. Việc đánh giá chất lượng của mô hình cũng có thể sử dụng cả hai tập, tuy nhiên để đi tìm tham số của mô hình thì ta chỉ được sử dụng dữ liệu của tập huấn luyện. Muốn mô hình hoạt động tốt trên tập kiểm thử thì trước hết nó phải hoạt động tốt trên tập huấn luyện. Đôi khi, ranh giới giữa hai tập huấn luyện và kiểm thử không thật sự rõ ràng. Các thuật toán thực tế luôn liên tục được cập nhật dựa trên dữ liệu mới được đưa vào, các thuật toán này được gọi là online learning (học trực tuyến). Phần dữ liệu mới này lúc đầu không được hệ thống sử dụng để xây dựng mô hình, nhưng về sau có thể được mô hình sử dụng để cải tiến mô hình. Ngược lại với online learning đó là offline learning đó là các hệ thống được xây dựng một lần dựa trên tập huấn luyện.

1.3.1.2. Phân loại các thuật toán học máy

Việc huấn luyện các mô hình học máy có thể xem là việc cho chúng trải nghiệm trên các tập dữ liệu – tập huấn luyện. Các dữ liệu khác nhau thì sẽ cho các mô hình các trải nghiệm khác nhau. Chất lượng của các tập dữ liệu này cũng ảnh hưởng tới hiệu năng của mô hình. Dựa trên tín chất của các tập dữ liệu, các thuật toán học máy có thể được phân thành hai loại: học không giám sát và học có giám sát.

Học có giám sát là thuật toán dữ đoán đầu ra của một hoặc nhiều dữ liệu mới dựa trên các cặp (đầu ra, đầu vào) đã biết trước. Học có giám sát là nhóm thuật toán học máy phổ biến nhất. Biểu diễn một cách toán học, học có giám sát là khi chúng ta có một tập hợp biến đầu vào $X = \{x_1, x_2, \dots, x_n\}$ và một tập hợp đầu ra tương ứng $Y = \{y_1, y_2, \dots, y_n\}$, trong đó x_i, y_i là các vector. Các cặp dữ liệu biết trước $(x_i, y_i) \in X \times Y$ tạo nên tập huấn luyện. Từ tập huấn luyện này, cần phải tạo ra một hàm số ánh xạ mỗi phần tử từ tập X sang một phần tử (xấp xỉ) tương ứng của tập Y.

$$y_i \approx f(x_i), \forall i = 1,2, \dots, N$$

Mục đích là xấp xỉ hàm số f thật tốt để khi có một dữ liệu x mới, chúng ta có thể tính được nhãn tương ứng của nó $y = f(x)$.

Ngược lại, trong học không giám sát, ta không biết được kết quả đầu ra mà chỉ biết các vector đặc trưng của dữ liệu đầu vào. Các thuật toán học không giám sát sẽ dựa vào cấu trúc của dữ liệu để thực hiện một công việc nào đó, ví dụ như phân nhóm hoặc giảm số chiều của dữ liệu (dimentionality reduction). Một cách toán học, học không giám sát là khi chúng ta chỉ có dữ liệu đầu vào X mà không biết đầu ra Y tương ứng. Không giống như trong các thuật toán học có giám sát, ta không biết câu trả lời chính xác cho mỗi dữ liệu đầu vào trong học không giám sát. Từ góc độ xác suất thống kê, học không giám sát trải nghiệm qua rất nhiều ví dụ (các điểm dữ liệu) x và cố gắng học phân phối xác suất p(x), hoặc các tính chất của phân phối của dữ liệu một cách trực tiếp hoặc gián tiếp. Trong khi đó, học có giám sát quan sát các ví dụ x và các kết quả tương ứng y, sau đó cố gắng học cách dự đoán y từ x thông qua việc đánh giá xác suất có điều kiện p(y|x). Xác suất này có thể diễn đạt bằng lời là biết rằng một điểm dữ liệu có vector đặc trưng là x, xác suất để đầu ra của nó bằng y là bao nhiêu.

1.3.1.3. Hàm mất mát và tham số mô hình

Mỗi mô hình học máy được mô tả bởi các tham số mô hình. Công việc của một thuật toán học máy là đi tìm các tham số mô hình phù hợp với mỗi bài toán. Việc đi tìm các tham số mô hình có liên quan mật thiết đến các phép đánh giá. Mục đích là đi tìm các tham số mô hình sao cho các phép đánh giá cho kết quả tốt nhất. Trong bài toán phân lớp, kết quả tốt có thể được hiểu là ít điểm dữ liệu bị phân lớp sai nhất. Trong bài toán hồi quy, kết quả tốt là khi sự sai lệch giữa đầu ra dự đoán và đầu ra thực sự là ít nhất.

Quan hệ giữa một phép đánh giá và các tham số mô hình thường được mô tả thông qua một hàm số được gọi là hàm mất mát (loss function). Hàm mất mát này thường có giá trị nhỏ khi phép đánh giá cho kết quả tốt và ngược lại. Việc đi tìm các tham số mô hình sao cho phép đánh giá trả về kết quả tốt tương đương với việc tối

thiểu hàm mất mát. Như vậy, việc xây dựng một mô hình học máy chính là việc đi giải một bài toán tối ưu. Quá trình đó có thể được coi là quá trình học của máy.

Bài toán tối thiểu hàm mất mát để tìm tham số mô hình thường được viết dưới dạng:

$$\theta^* = agrmin_\theta \, J(\theta)$$

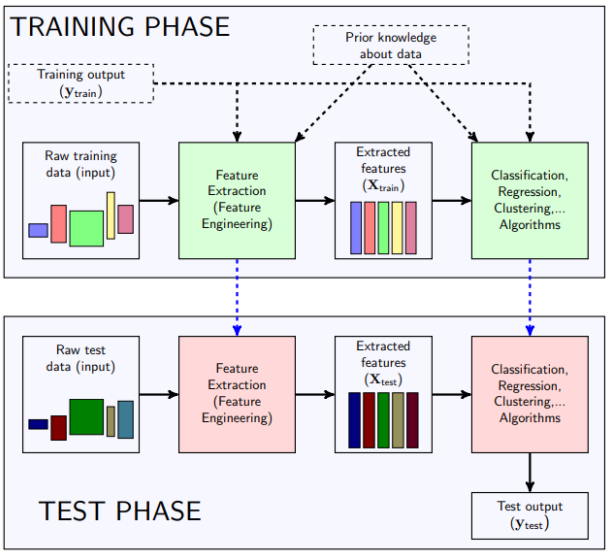
Trong đó:

θ: là tập hợp các tham số của mô hình;

$J(\theta)$: là hàm mất mát của mô hình.

1.3.1.4. Mô hình các thuật toán học máy

Trong mỗi bài toán của học máy sẽ có hai bước (phase) lớn là bước huấn luyện và bước kiểm thử. Dữ liệu cũng vậy sẽ được chia thành dữ liệu huấn luyện và dữ liệu kiểm thử. Bước huấn luyện chỉ sử dụng dữ liệu huấn luyện, bước kiểm thử chỉ sử dụng kiểm thử. Mô hình phần lớn của các thuật toán học máy có thể được mô phỏng như hình dưới.



Hình 1.4. Mô hình thường gặp trong các thuật toán học máy

(Nguồn: machinelearningcoban.com)

Từ dữ liệu thô ban đầu sẽ được Feature Extraction tạo ra một vector đặc trưng cho mỗi điểm dữ liệu đầu vào. Vector này thường sẽ có dữ liệu như nhau bất kể dữ liệu đầu vào có kích thước như thế nào. Dữ liệu thô là tất cả các thông tin mà ta có thể biết về dữ liệc. Các vector đặc trưng sau khi được khởi tạo sẽ được đưa vào các thuật toán học máy để huấn luyện trong pha huấn luyện, còn trong pha kiểm thử thì các vector đặc trưng này sẽ được dùng để đưa ra quyết định.

1.3.2. Support vector machines

Support vector machines (SVM) là một trong những thuật toán phân lớp phổ biến và hiệu quả. SVM hoạt động dựa trên 2 nguyên tắc[2]:

- SVM tìm cách phân lớp bằng một siêu phẳng sao cho khoảng cách từ siêu phẳng tới các lớp là lớn nhất. Nguyên tắc này được gọi là nguyên tắc lề cực đại (max margin);
- Trong trường hợp, không thể phân lớp bằng một siêu phẳng thì SVM sẽ ánh xạ không gian ban đầu sang một không gian khác (thường là có số chiều nhiều hơn), sau đó tìm lề cực đại trong không gian mới này.

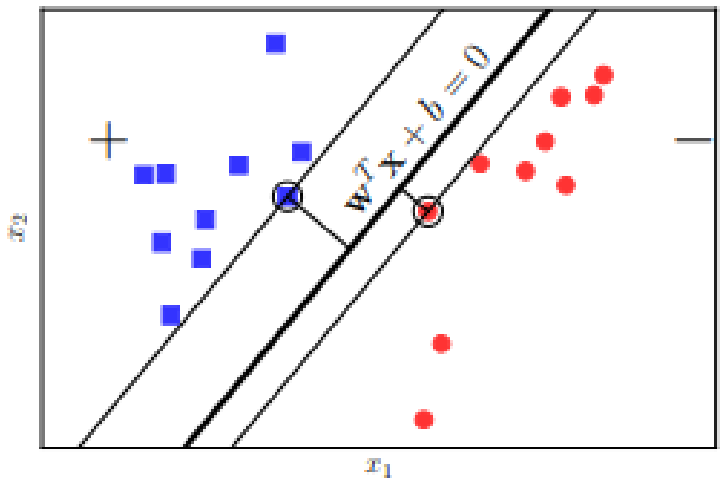
SVM được phát triển từ mô hình phân loại tuyến tính. Bên cạnh đó, ta có hàm phân loại trong trường hợp hồi quy tuyến tính như sau:

$$f(x) = \sum_{i=1}^{d} w_i x_i + b = \boldsymbol{w}^T \boldsymbol{x} + b$$

(Nguồn: Giáo trình nhập môn trí tuệ nhân tạo – Từ Minh Phương)

Trong đó:

- **w** là vector trọng số;
- b là độ lệch chuẩn.



Hình 1.5. Phân loại tuyến tính

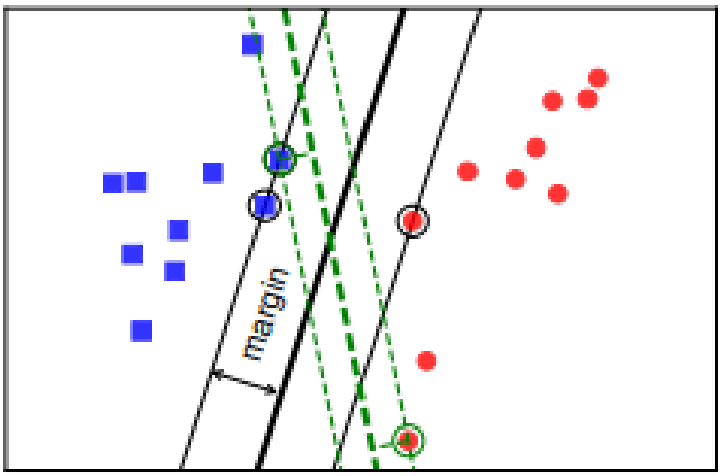
(Nguồn: Giáo trình nhập môn trí tuệ nhân tạo – Từ Minh Phương)

Siêu phẳng f(x) sẽ phân chia không gian thành hai phần, phần bên phải gồm những điểm x có f(x) >0 và phần bên trái gồm những điểm có f(x)<0. Như vậy, mỗi x sẽ có nhãn được phân loại y được xác định như sau:

$$y = sgn(f(x))$$

Trong đó sgn() là hàm dấu, nhận giá trị +1 nếu f(x)>0 và -1 nếu f(x)<0.

Tuy nhiên, dễ dàng nhận thấy rằng có thể tìm thấy được nhiều siêu phẳng có thể phân chia không gian thành hai phần và đảm bảo yêu cầu trên. Có một câu hỏi đặt ra là trong vô số siêu phẳng trên thì đâu là siêu phẳng tốt nhất. Đối với bài toán phân lớp thì siêu phẳng tốt nhất là siêu phẳng khi có điểm dữ liệu mới vào thì có tỉ lệ phân lớp chính xác cao nhất. Để làm được điều này, siêu phẳng này cần phải là siêu phẳng có biên lớn nhất. Biên của một lớp ở đây là khoảng cách điểm gần nhất của lớp đó tới siêu phẳng phân chia.



Hình 1.6. Biên của một lớp

(Nguồn: Giáo trình nhập môn trí tuệ nhân tạo – Từ Minh Phương)

Để xác định siêu phẳng với biên cực đại, ta cần tìm **w** và b tương ứng. Quá trình đó được thực hiện như sau. Trước hết, có thể yêu cầu hàm phân loại cho phép phân chia toàn bộ tập huấn luyện với một khoảng cách an toàn, chẳng hạn phải thoả mãn:

$$w^T x_i + b \geq +1 \text{ nếu } y_i = +1$$

$$w^T x_i + b \leq -1 \text{ nếu } y_i = -1$$

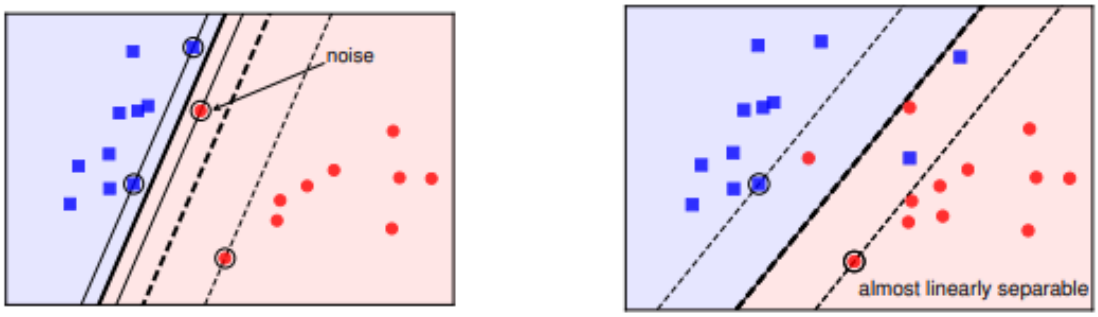
hay viết gọn lại thành $y_i(w^T x_i + b) - 1 \geq 0$ với mọi *i*. Điều này có thể thực hiện được, chẳng hạn bằng cách nhân w và b với hệ số phù hợp.

Các điểm x_i sao cho $\mathbf{w}^T x_i + b = +1$ nằm trên siêu phẳng song song với siêu phẳng phân cách và cách gốc toạ độ một khoảng $|1-b|/||w||$. Tương tự, các điểm x_i sao cho $\mathbf{w}^T x_i + b = -1$ nằm trên siêu phẳng song song với siêu phẳng phân cách và cách gốc toạ độ một khoảng $|-1-b|/||w||$. Khi đó, $m+ = m- = 1/||w||$, và giá trị của lề sẽ bằng $2/||w||$. Để giá trị lề là cực đại, ta cần xác định w sao cho $2/||w||$ là lớn nhất, hay $||w||^2/2$ là nhỏ nhất (cực tiểu hoá $||w||^2$ thuận lợi hơn do dễ tính đạo hàm hơn). Như vậy, siêu phẳng có lề cực đại được xác định bằng cách tìm **w** và b sao cho:

$\frac{1}{2}|| w ||^2$ là nhỏ nhất,

đồng thời thoả mãn ràng buộc: $y_i(\mathbf{w}^T x_i + b) - 1 \geq 0$ với mọi *i*.

Tuy nhiên, SVM ở trên sẽ không hoạt động tốt hoặc thậm chí không thể hoạt động trong trường hợp có một điểm nhiễu của lớp khác ở gần lớp còn lại, trong trường hợp này thì biên sẽ rất nhỏ, hoặc trong trường hợp tập dữ liệu không thể phân chia tuyến tính.



Hình 1.7. Hai trường hợp khi SVM thuần hoạt động không hiệu quả

(Nguồn: Giáo trình nhập môn trí tuệ nhân tạo – Từ Minh Phương)

Đối với hai trường hợp này, để SVM có thể hoạt động và hoạt động tốt ta phải chấp nhận hi sinh một số điểm dữ liệu để có biên tốt nhất. Với mỗi điểm x_n trong toàn bộ dữ liệu huấn luyện, ta giới thiệu thêm một biến đo sự hi sinh ξ_n tương ứng. Việc này được thể hiện bằng cách thay ràng buộc thành:

$$y_i(\mathbf{w}^T x_i + b) \geq 1 - \xi_i$$

trong đó $\xi_i \geq 0$ là các biến phụ, được thêm vào để cho phép các điểm nằm trong vùng của biên ($1 \geq \xi_i > 0$) hoặc thậm chí bị phân loại sai ($\xi_i \geq 1$). Tổng của các biến phụ này sẽ giới hạn mức độ sai khi phân loại, tổng càng lớn chứng tỏ càng sai nhiều, nếu tất cả biến phụ bằng 0 thì toàn bộ dữ liệu không lấn vào vùng biên và được phân loại đúng. Do vậy, ta có thể thêm tổng các biến phụ vào hàm mục tiêu như một thành phần phạt cho các trường hợp bị sai. Bài toán tìm cực trị ở trên sẽ trở thành:

Tìm $\mathbf{w}$ và b sao cho:

$\frac{1}{2}||\mathbf{w}||^2 + C\sum_i \xi_i$ là nhỏ nhất,

đồng thời thoả mãn ràng buộc:

$y_i(\mathbf{w}^T x_i + b) \geq 1 - \xi_i, \ \xi_i \geq 0$, với mọi *i*

Tham số C > 0 thể hiện sự ưu tiên giữa hai mục tiêu: 1) tìm được biên lớn; và 2) giảm số lỗi. C càng lớn thì mục tiêu thứ hai càng được ưu tiên và ngược lại. Mô hình phân loại như vậy gọi là phân loại với lề mềm (soft margin).

Có một phương pháp khác có thể phân loại tuyến tính cho các trường hợp không tuyến tính. Đó là phương pháp Kernel SVM, ý tưởng cơ bản của kernel SVM là tìm một phép biến đổi cho các tập dữ liệu không phân loại tuyến tính sang một không gian mới. Ở khôn gian mới này dữ liệu trở nên có thể phân loại tuyến tính hoặc có thể phân loại tuyến tính. Từ đây, bài toán phân lớp có thể giải quyết theo hướng hard hoặc soft-margin SVM.

Ví dụ về kernel SVM: ở hình a dữ liệu của hai lớp không phân biệt tuyến tính trong hai chiều. Nếu chúng ta coi chiều thứ ba là một hàm số của hai chiều còn lại $z = x^2+y^2$, thì các điểm dữ liệu sẽ được phân bố trên một mặt parabolic và đã trở nên có thể phân loại tuyến tính. Mặt phẳng phân loại hai lớp có thể tìm được bằng SVM thuần hoặc soft-margin SVM.

Tuy nhiên khi chiếu vào không gian mới đòi hỏi phải tính toán lại các đặc trưng mới. Số lượng đặc trưng như vậy có thể rất lớn vì thường chiều vào không gian mới thì số chiều của không gian mới sẽ nhiều hơn. Để tránh việc tính toán các đặc trưng trong không gian mới, một phương pháp được đưa ra đó là sử dụng hàm nhân (kernel function). Cụ thể, vector trong số của SVM sẽ được tính bởi:

$$\mathbf{w} = \sum_{i=1}^{n} y_i \alpha_i \phi(\mathbf{x}_i)$$

nên hàm phân loại sẽ trở thành

$$f(\mathbf{x}) = \mathbf{w}^T \phi(\mathbf{x}) + b = \sum_{i=1}^{n} y_i \alpha_i \phi(\mathbf{x}_i)^T \phi(\mathbf{x}) + b$$

(Nguồn: Giáo trình nhập môn trí tuệ nhân tạo – Từ Minh Phương)

Do các biểu thức trên chứa φ(x)Tφ(x') nên nếu ta có thể tìm được hàm K(x,x') = φ(x)Tφ(x') , ta có thể sử dụng K(x,x') trong các biểu thức trên và không cần tính cụ thể các ánh xạ φ(x), φ(x'). Hàm K(x,x') như vậy được gọi là hàm nhân (kernel function)

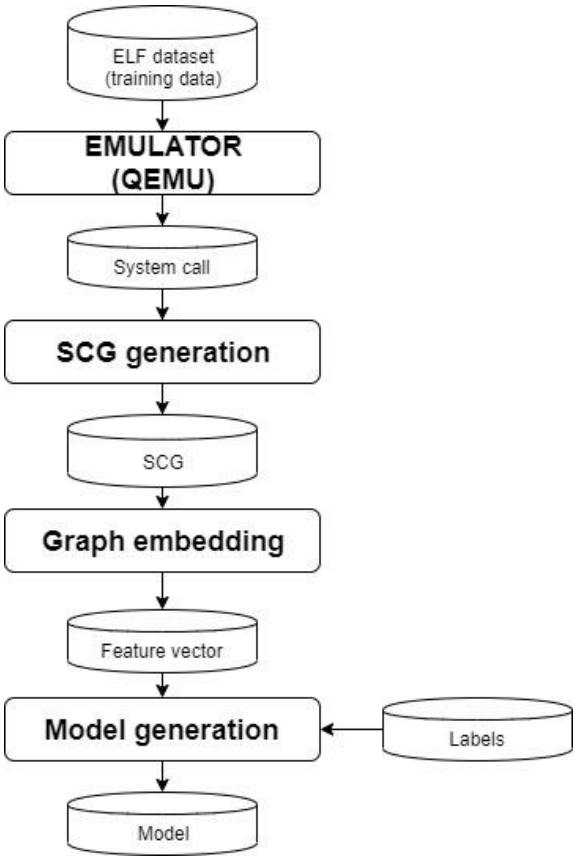
1.4. Kết luận chương

Trong nội dung chương 1 tác giả đã tập trung vào việc trình bày: kiến thức tổng quan về IoT; mã độc IoT botnet; tổng quan về các phương pháp phát hiện mã độc; tổng quan về học máy và thuật toán học SVM. Các kiến thức này sẽ là kiến thức nền tảng để tác giả vận dụng đề xuất nên mô hình phát hiện mã độc IoT botnet sẽ được trình bày trong chương 2.

CHƯƠNG 2: XÂY DỰNG MÔ HÌNH PHÁT HIỆN MÃ ĐỘC IOT BOTNET

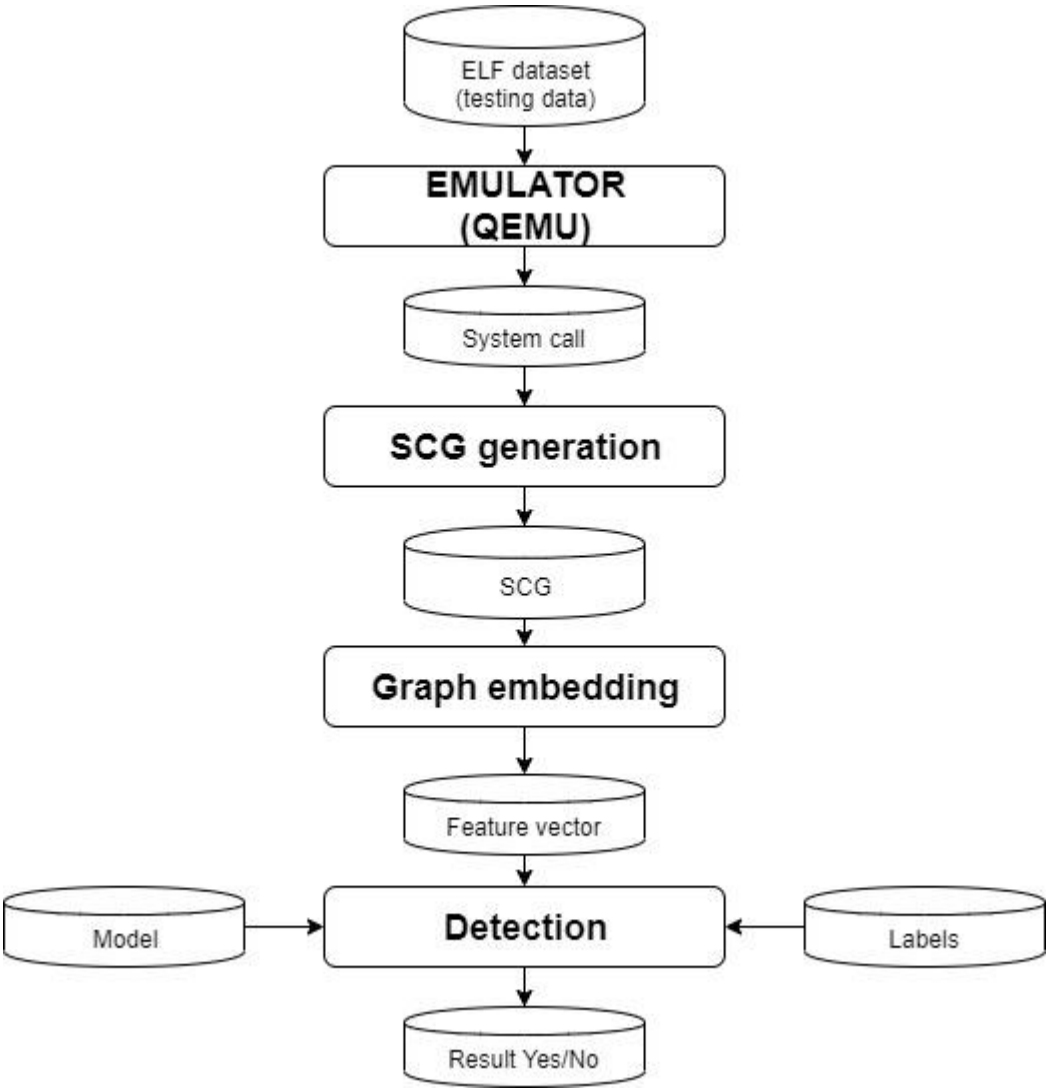
2.1. Mô hình tổng quan

Trong chương này, tác giả sẽ mô tả về mô hình tổng quan phát hiện mã độc IoT botnet. Mô hình tổng quan của bài toán áp dụng học máy trong phát hiện mã độc IoT botnet sẽ có hai pha: pha huấn luyện (trainning) và pha kiểm thử (testing). Pha huấn luyện là các bước để có thể xây dựng lên được mô hình, đầu vào là dữ liệu huấn luyện và đầu ra là mô hình. Pha kiểm thử là bước để đánh giá chất lượng của mô hình, đầu vào của pha này là dữ liệu kiểm thử, mô hình ở bước huấn luyện và đầu ra là kết quả phân loại dữ liệu kiểm thử. Pha huấn luyện của mô hình phát hiện mã độc IoT botnet được mô tả trong hình 2.1 như sau:



Hình 2.1. Pha huấn luyện trong hô hình phát hiện botnet trong các thiết bị IOT

Pha kiểm thử của mô hình phát hiện mã độc IoT botnet được mô tả trong hình 2.2 như sau:



Hình 2.2. Pha kiểm thử trong mô hình phát hiện mã độc IoT botnet

Hai pha huấn luyện và kiểm thử sẽ có các bước tiền xử lý dữ liệu giống nhau đó là:

- Thu thập dữ liệu bằng Emulator (QEMU);
- Xây dựng đồ thị SCG (SCG generation);
- Xây dựng đồ thị nhúng (graph embedding).

Sau khi có vector đặc trưng từ bước xây dựng đồ thị nhúng, trong pha huấn luyện sẽ sử dụng thêm dữ liệu nhãn đánh dấu mã độc/lành tính của tập dữ liệu huấn luyện để đưa vào kỹ thuật học máy để sinh model. Trong khi đó pha huấn luyện sẽ sử dụng model được xây dựng trong pha huấn luyện, cùng với dữ liệu kiểm thử để đưa ra kết quả phát hiện. Trong phần tiếp theo, tác giả sẽ mô tả chi tiết chức năng cũng như cấu trúc của từng thành phần cụ thể.

2.2. Thu thập dữ liệu

Để thực hiện phân tích động mã độc nói chung và mã độc trên IoT nói riêng, sử dụng Sandbox là kỹ thuật phổ biến. Sandbox là một kỹ thuật quan trọng trong lĩnh vực bảo mật có tác dụng cô lập các ứng dụng, ngăn chặn các mã độc để chúng không thể làm hỏng hệ thống máy tính, hay cài cắm các mã độc nhằm ăn cắp thông tin cá nhân của người dùng. Một Sandbox về cơ bản là một môi trường dùng để chạy phần mềm và môi trường đó được nằm trong sự kiểm soát chặt chẽ. Sandbox giúp hạn chế chức năng của một đoạn mã, cấp quyền cho một đoạn mã nào đó chỉ được thực hiện một số chức năng nhất định, từ đó nó không thể thực hiện những can thiệp khác có thể làm nguy hại cho máy tính người dùng. Tiêu chí quan trọng nhất của Sandbox phục vụ phân tích mã độc là lượng thông tin thu được sau khi thực thi mã độc trên Sandbox đó, hay làm mã độc bộc lộ tối đa hành vi. Sandbox có 2 loại là Sandbox vật lý và Sandbox ảo. Các Sandbox vật lý có khả năng để mã độc bộc lộ hành vi tốt hơn do nó dựa trên thiết bị thực tế, đầy đủ các thành phần và có thể kích hoạt đối với các mã độc có khả năng phát hiện hệ thống phân tích, theo dõi nó. Tất nhiên Sandbox vật lý có hạn chế lớn là khó tuỳ biến, khó khôi phục lại hiện trạng ban đầu và chi phí cao. Sandbox ảo dựa trên công nghệ mô phỏng, ảo hoá thường là sử dụng các máy ảo. Điều quan trọng nhất, để mã độc thực hiện được hành vi thì phải mô phỏng đầy đủ các môi trường như thiết bị ngoại vi, mạng, môi trường kết nối... Các Sandbox cho mã độc trên máy tính truyền thống thường hướng đến sử dụng Vmware, Virtual Box nhưng với các thiết bị IoT thì các máy ảo chạy Qemu là giải pháp được sử dụng phổ biến, do Vmware và Virtual Box không hỗ trợ các kiến trúc chíp nhúng phổ biến như MIPS, ARM...

Nhiều giải pháp Sandbox tương đối hoàn chỉnh cho phân tích mã độc đã được xây dựng như Cuckoo sandbox. Tuy nhiên, đối với các kiến trúc chíp ARM, MIPS Cuckoo sandbox cho kết quả nhật ký các lời gọi hệ thống không được đầy đủ và chi tiết. Vì vậy, trong luận văn này tác giả sử dụng bộ giả lập QEMU để mô phỏng kiến trúc chíp ARM, MIPS sử dụng mã nguồn mở Strace tool để thu thập log và điều quan trọng là sử dụng bản ảnh QEMU Debian VM dành cho kiến trúc ARM, MIPS do Aurel đăng tải lên.

QEMU là một phần mềm mã nguồn mở được viết bởi Fabrice Bellard dùng để giả lập và ảo hóa phần cứng. QEMU mô phỏng bộ vi xử lý của máy thông qua dịch nhị phân động và cung cấp một bộ các mô hình phân cứng và các thiết bị khác nhau điều đó cho phép QEMU có thể chạy với nhiều hệ điều hành khác nhau. Khi được sử dụng như một trình ảo hóa, QEMU có thể đạt được hiệu năng gần như máy thật bằng cách thực thi mã khách trực tiếp trên CPU của máy chủ. Bên cạnh đó, QEMU có thể lưu trữ và khôi phục lại trạng thái làm việc của hệ điều hành với tất cả các chương trình đang chạy[14]. Hiện nay, QEMU có thể mô phỏng các kiến trúc sau:

- IA-32 (x86) PCs
- x86-64 PCs
- MIPS64 Release 6 và các phiển bản trước
- Sun's SPARC sun4m
- Sun's SPARC sun4u
- ARM development boards (Integrator/CP and Versatile/PB)
- SH4 SHIX board
- PowerPC (PReP and Power Macintosh)
- ETRAX CRIS
- MicroBlaze
- RISC-V

Máy ảo trong QEMU có thể giao tiếp với nhiều loại phần cứng máy chủ vật lý, bao gồm cả đĩa cứng của người dùng, ổ đĩa CD-ROM, card mạng, giao diện âm thanh và thiết bị USB. Các thiết bị USB cũng có thể được mô phỏng hoàn toàn hoặc có thể sử dụng các thiết bị USB của máy chủ, mặc dù điều này đòi hỏi phải có đặc quyền của quản trị viên và có thể không hoạt động với tất cả các thiết bị. Ổ đĩa ảo có thể được lưu trữ ở định dạng đặc biệt (qcow hoặc qcow2) chỉ chiếm dung lượng băng với dung lượng mà hệ điều hành ảo sử dụng thực sự. Bằng cách này, một đĩa 120 GB giả lập có thể chỉ chiếm vài trăm megabyte trên máy chủ.Bên cạnh đó, định dạng QCOW2 cũng cho phép tạo ra các bản snapshot của máy ảo. Các định dạng ổ đĩa ảo QEMU có thể hỗ trợ như sau:

- macOS Universal Disk Image Format (.dmg) – Read-only;
- Bochs – Read-only;
- Linux cloop – Read-only;
- Parallels disk image (.hdd, .hds) – Read-only;
- QEMU copy-on-write (.qcow2, .qed, .qcow, .cow);
- VirtualBox Virtual Disk Image (.vdi);
- Virtual PC Virtual Hard Disk (.vhd);
- Virtual VFAT;
- VMware Virtual Machine Disk (.vmdk);
- Raw images (.img) that contain sector-by-sector contents of a disk;
- CD/DVD images (.iso) that contain sector-by-sector contents of an optical disk (e.g. booting live OSes).

Strace là một tiện ích được sử dụng để chẩn đoán, gỡ lỗi trong hệ điều hành linux. Strace được viết bởi Paul Kranenburg vào năm 1991 và dành cho SunOS. Nó được sử dụng để giám sát và giả mạo các tương tác giữa các quy trình và nhân Linux, bao gồm các lời gọi hệ thống, phân phối tín hiệu và thay đổi trạng thái của quy trình. Hoạt động của strace được thực hiện bởi tính năng kernel được gọi là ptrace[15]. Các tính năng cơ bản của Strace như sau:

- Cho phép xuất ra các lời gọi hệ thống của một chương trình;
- Lọc theo lời gọi hệ thống;
- Theo dõi các lời gọi hệ thống truy cập vào đường dẫn đã chỉ định;
- Trích xuất tất cả dữ liệu đọc/ghi ra tệp tin với định dạng hexadecimal và ASC II;
- Đếm số lần, thời gian và số lỗi của các lời gọi hệ thống;
- Lọc theo trang thái trả về của các lời gọi hệ thống;
- In ngăn xếp;
- Thay đổi mã trả về hoặc mã lỗi trả về trong các lời gọi hệ thống.

```
$ strace -c ls > /dev/null
% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- ----------------
 89.76    0.008016           4      1912           getdents
  8.71    0.000778           0     11778           lstat
  0.81    0.000072           0      8894           write
  0.60    0.000054           0       943           open
  0.11    0.000010           0       942           close
  0.00    0.000000           0         1           read
  0.00    0.000000           0       944           fstat
  0.00    0.000000           0         8           mmap
  0.00    0.000000           0         4           mprotect
  0.00    0.000000           0         1           munmap
  0.00    0.000000           0         7           brk
  0.00    0.000000           0         3         3 access
  0.00    0.000000           0         1           execve
  0.00    0.000000           0         1           sysinfo
  0.00    0.000000           0         1           arch_prctl
------ ----------- ----------- --------- --------- ----------------
100.00    0.008930                 25440         3 total
```

Hình 2.3. Kết quả trả về của một lệnh trong strace

Trong giai đoạn này, mô hình sẽ sử dụng QEMU để giả lập kiến trúc chíp ARM, MIPS kiến trúc được sử dụng phổ biến trên các thiết bị IOT dân dụng, sau đó chạy bản ảnh QEMU Debian VM dành riêng cho kiến trúc chíp ARM, MIPS. Công việc tiếp theo là sử dụng strace tool để có thể trích xuất ra được các lời gọi hệ thống của các tệp tin mã độc. Đầu vào của giai đoạn này là các tệp tin mã độc và kết quả đầu ra là tập các tệp tin log lưu trữ chi tiết các lời gọi hệ thống của tệp tin mã độc.

2.3. Xây dựng đồ thị SCG

Trong bước này, mô hình sẽ sử dụng nhật ký lời gọi hàm hệ thống để có thể xây dựng đồ thị lời gọi hệ thống. Mỗi tệp tin mẫu sẽ có một tệp tin nhật ký lời gọi và cũng sẽ có một đồ thị lời gọi hệ thống.

Một lời gọi hệ thống là việc một chương trình máy tính yêu cầu một dịch vụ từ nhân của hệ điều hành mà nó được thực thi. Điều này có thể bao gồm các dịch vụ liên quan đến phần cứng (ví dụ: truy cập ổ đĩa cứng), tạo và thực thi các tiến trình mới, giao tiếp với các dịch vụ nhân tích hợp như lập lịch xử lý[14]. Đồ thị lời gọi hệ thống là một đồ thị có hướng trong đó tập đỉnh là tập các lời gọi hệ thống, các cạnh biểu thị mối quan hệ lời gọi hàm liên tiếp giữa các đỉnh. Tuy nhiên các lời gọi hệ thống chúng ta có thể mã hóa thành các ký tự đơn giản hoặc thành các số vì lời gọi hệ thống là các hàm biểu diễn dữ liệu trong hệ thống sẽ tốn bộ nhớ và không cần thiết. Bên cạnh đó các hàm hệ thống của nhân linux có hạn nên có thể dễ dàng mã hóa. Trong mô hình này, tác giả sử dụng bảng mã hóa được miêu tả như sau:

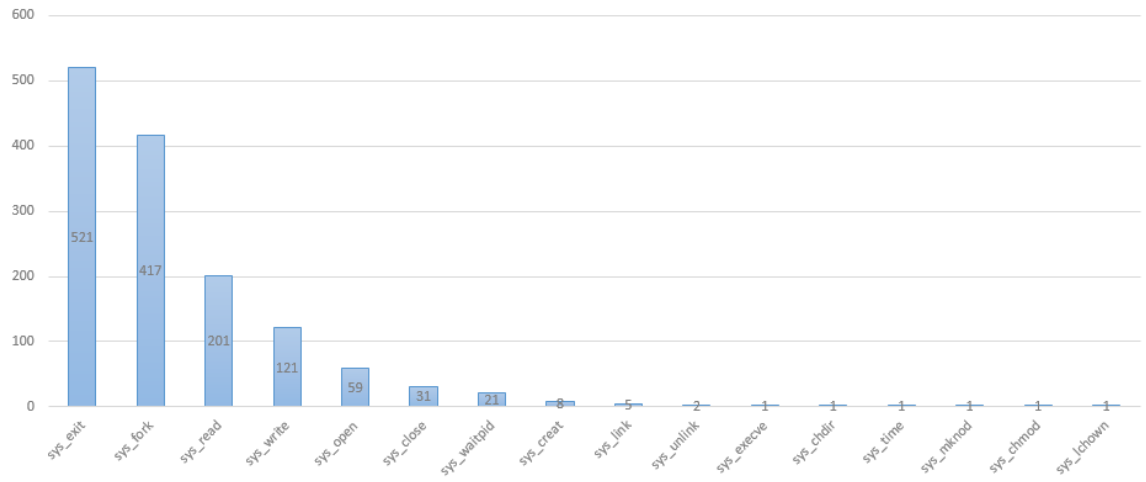
Bảng 2.1. Bảng mã hóa các hàm hệ thống

STT	**Hàm hệ thống**	**Mã**
1	restart_syscall	1
2	exit	2
…	…	…
346	ni_syscall	346
347	lchown16	347

Đối với một tệp tin mã độc hoặc lành tình thì số lượng lời gọi hàm hệ thống có thể rất lớn, con số có thể đạt lên vài nghìn lời gọi hàm hệ thống. Với số lượng lời gọi hàm hệ thống lớn như vậy, sẽ ảnh hưởng nhiều đến hiệu năng tính toán của mô hình. Thông thường số lượng dữ liệu chúng ta thu thập được càng nhiều thì chất lượng mô hình học máy sẽ tăng lên theo tỷ lệ thuận. Tuy nhiện, khi số lượng đạt đến một ngưỡng nhất định thì chất lượng mô hình sẽ cải thiện không đáng kể mà còn kéo theo chi phí tính toán lớn. Theo công bố của Canzanese thì đối với phân tích động hành vi độc hại số lượng 1000 lời gọi hàm hệ thống là đủ, với số lượng lời gọi lớn hơn 1000 thì tỷ lệ phát hiện tăng không đáng kể. Vì vậy, để có thể giảm số lượng lời gọi hàm hệ thống mà vẫn đảm bảo chất lượng mô hình học máy, có hai phương pháp đó là:

- Xây dựng và sử dụng bảng tần suất gọi hàm hệ thống được gọi sau đó chọn những hàm hệ thống được gọi nhiều để xây dựng đồ thị SCG;

- Đối với hàm hệ thống được gọi lặp lại và liên tiếp nhiều lần thì chỉ ghi nhận một lần duy nhất trong đồ thị SCG.

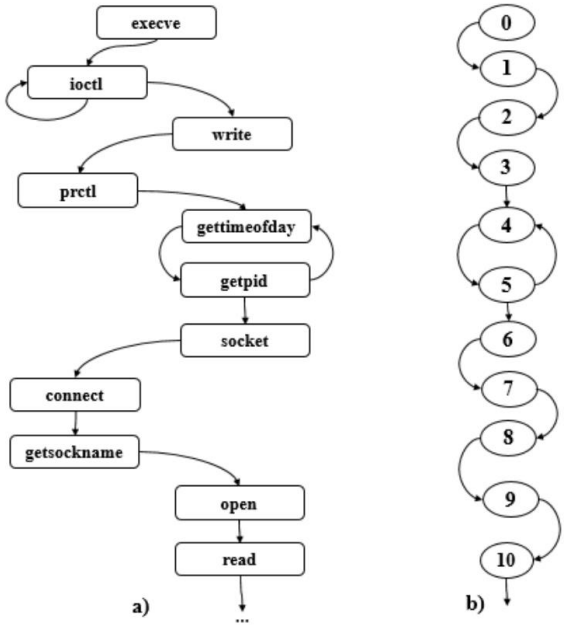


Hình 2.4. Biểu đồ tần suất gọi hàm hệ thống

Trong luận văn này, tác giả sử dụng phương pháp số hai để có thể giảm số lượng lời gọi hàm hệ thống đó là chỉ ghi nhận một lần đối với những hàm hệ thống được gọi lặp lại và liên tục. Đồ thị lời gọi hệ thống dựa trên kỹ thuật phân tích động sẽ lấy nhật ký lời gọi hệ thống của các tệp tin có mã độc và lành tính làm đầu vào, sau đó xây dựng nên đồ thị. Thuật toán xây dựng đồ thị lời gọi hệ thống có thể được biểu diễn như sau:

```
1: function scg_generation(S[], id)
   Input:
     - S[]: a list of system-calls log for each process
     - id: PID of the process
   Output: system call graph(SCG)
2: s_log = S[id]
3: V=[], E=[]
4: SCG = (V,E)
5: V = V U {s_log[0]}
6: for(i=1;i<|s_log|;i++) do
7:  if s_log[i]<>s_log[i-1] then
8:      if s_log[i] is "fork" then
9:          child_pid = Get PID of process which is opened by "fork"
```

```
10:         SCG = SCG ∪ scg_generation(S, child_pid)
11:     endif
12:     if s_log[i] not in V then
13:         V = V ∪ {s_log[i]}
14:     endif
15:     E = E ∪ {edge(s_log[i-1] connect to s_log[i])}
16: endif
17: endfor
18: return SCG
```



Hình 2.5. Ví dụ về đồ thị lời gọi hệ thống

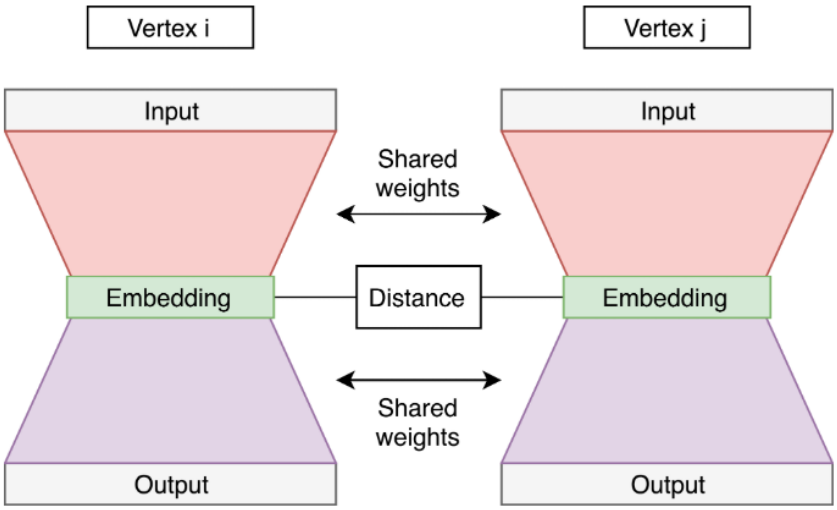
2.4. Đồ thị nhúng

Đến bước này, dữ liệu thô từ các tập tin mẫu đã được chuyển thành các đồ thị lời gọi hệ thống. Đồ thị là một phương pháp biểu diễn dữ liệu có ý nghĩa và dễ hiểu. Tuy nhiên các đồ thị này sẽ không có kích thước giống nhau và chính điều này sẽ gây khó khăn rất nhiều cho các thuật toán học máy, chi phí tính toán sẽ rất lớn, thâm chí không thể tính toán. Bên cạnh đó các kỹ thuật học máy với đồ thị còn nhiều hạn chế. Vì vậy, cần thêm một bước xử lý làm sao cho các đồ thị có thể chuyển về các dạng vector có kích thước giống nhau. Khi đã chuyển về vector thì hiệu suất tính toán của

thuật toán học máy sẽ tăng lên đáng kể. Một kỹ thuật tác giả sẽ áp dụng trong bước này là kỹ thuật đồ thị nhúng (graph embeddings).

Đồ thị nhúng là sự chuyển đổ các thuộc tính của đồ thị thành một vector hoặc một tập vector. Việc chuyển đổi này vẫn lưu giữ được các thông tin của đồ thị như: cấu trúc liên kết của đồ thị, mối liên quan giữa các đỉnh, sơ đồ con và các thông tin liên quan đến đồ thị[16]. Có 2 phương pháp chính trong đồ thị nhúng đó là: vertex embeddings và graph embeddings.

- Vertex embeddings: phương pháp sẽ chuyển đổi mỗi đỉnh trong đồ thị thành một vector. Phương pháp này thường được áp dụng trong các trường muốn trực quan hóa hoặc dự đoán cấp độ của đỉnh. Ví dụ như: trực quán hóa các đỉnh trong mặt phẳng 2D; dự đoán liên kết mới dựa trên sự tương đồng về đỉnh.
- Graph embeddings: phương pháp sẽ chuyển đổi đồ thị về một vector duy nhất. Phương pháp này thường được áp dụng vào việc muốn dự đoán về mức độ của đồ thị hoặc cần so sánh và trực quan hóa toán bộ đồ thị. Ví dụ như so sánh các cấu trúc hóa học.



Hình 2.6. Mô hình vertex embeddings

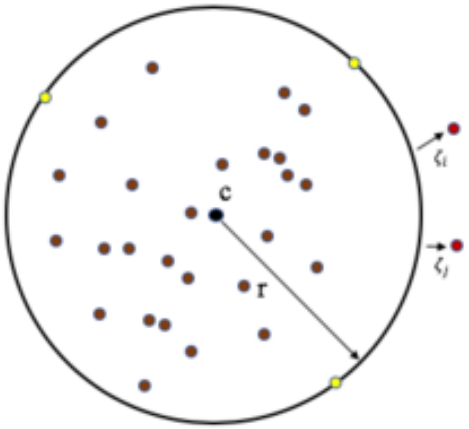
(Nguồn: Internet)

Một số hướng tiếp cận phổ biến trong phương pháp vertex embeddings đó là: DeepWalk, node2vec, SDNE,..và hướng tiếp cận phổ biến và tốt nhất hiện nay trong phương pháp graph embeddings đó là: graph2vec. Tuy nhiên, mọi hướng tiếp cận để xây dựng đồ thị nhúng đều dựa trên nền tảng phương pháp word2vec và mạng nơ-ron skip-gram.

Word2vec là một phương pháp nhúng để chuyển đổi các từ thành các vector nhúng. Các từ giống nhau thì có vector nhúng tương tự nhau. Word2vec sử dụng mạng nơ-ron skip-gram là một mạng nơ-ron với một lớp ẩn. Skip-gram được huấn luyện để có thể dự đoán từ tiếp theo trong câu.

2.5. Thiết lập mô hình học máy

Bài toán phân lớp truyền thống không thể hoạt động tốt trong trường hợp lực lượng giữa 2 lớp chênh lệch quá lớn hoặc thậm chí chỉ có dữ liệu của một tập dữ liệu. Ví dụ: bài toán phân loại trạng thái hoạt động bình thường của nhà máy hạt nhân, thì trong kịch bản này, có rất ít các trạng thái hệ thống bị lỗi mà chỉ có số liệu thống kê hoạt động bình thường của nhà máy. Trong trường hợp này, người ta đề xuất mô hình phân loại một lớp (One class classification – OCC). Thuật ngữ OCC được đưa ra bởi Moya và Hush vào năm 1996. Đối với OCC chỉ cần thu thập dữ liệu của một lớp chỉ định, không cần thu thập dữ liệu của lớp khác.



Hình 2.7. Siêu cầu bao lấy tất cả các điểm dữ liệu

(Nguồn: wikipedia.org)

Ý tưởng phân loại một lớp dựa trên SVM (OSVM) là việc xác định một siêu cầu nhỏ nhất (có tâm c và bán kính r) có thể bao lấy tất cả các điểm dữ liệu. OSVM có thể được xác định như sau:

$$\min_{r,c} r^2 \ sao\ cho\ ||\Phi(x_i) - c|| \leq \ r^2\ \forall i = 1,2,\dots,n$$

Tuy nhiên công thức trên sẽ rất hạn chế và nhạy cảm với nhiễu. Do đó một công thức linh hoạt hơn được xây dưng để có thể giảm nhiễu và có thể chấp nhận được như sau:

$$\min_{r,c,\zeta} r^2 + \frac{1}{\nu n}\sum_{i=1}^{n}\zeta_i$$

Sao cho

$$||\Phi(x_i) - c||^2 \leq r^2 + \zeta_i \quad \forall i = 1,2,\dots,n$$

Từ điều kiện tối ưu của Karush-Kuhn-Tucker (KKT), chúng ta có được

$$c = \sum_{i=1}^{n}\alpha_i\Phi(x_i),$$

khi đó $\alpha_i' s$là giải pháp cho vấn đề tối ưu hóa:

$$\max_{\alpha}\sum_{i=1}^{n}\alpha_i\kappa(x_i,x_i) - \sum_{i,j=1}^{n}\alpha_i\alpha_j\kappa(x_i,x_j)$$

Sao cho

$$\sum_{i=1}^{n}\alpha_i = 1 \text{ and } 0 \leq \alpha_i \leq \frac{1}{\nu n} \text{for all } i = 1,2,\dots,n.$$

Một số phương pháp đã được đề xuất để giải quyết vấn đề phân loại một lớp. Các cách tiếp cận có thể được phân loại thành ba loại chính: ước tính mật độ, phương pháp biên và phương pháp tái thiết.

2.6. Kết luận chương

Nội dung chương 2 căn cứ vào nghiên cứu lý thuyết tại chương 1 đưa ra mô hình đề xuất cho phương pháp phát hiện mã độc IoT botnet. Mô hình phát hiện bao gồm hai pha: Huấn luyện và kiểm thử. Các biến tiền xử lý của hai pha giống nhau bao gồm các bước:

- Thu thập dữ liệu bằng Emulator (QEMU);
- Xây dựng đồ thị SCG (SCG generation);
- Xây dựng đồ thị nhúng (graph embedding).

Pha huấn luyện sẽ sử dụng thêm dữ liệu nhãn đánh dấu mã độc/lành tính của tập dữ liệu huấn luyện để đưa vào kỹ thuật học máy để sinh model. Trong khi đó pha huấn luyện sẽ sử dụng model được xây dựng trong pha huấn luyện, cùng với dữ liệu kiểm thử để đưa ra kết quả phát hiện. Trong chương 3, tác giả sẽ mô tả các bước cài đặt môi trường, cũng như các công cụ sử dụng để xây dựng và kiểm thử mô hình học máy này.

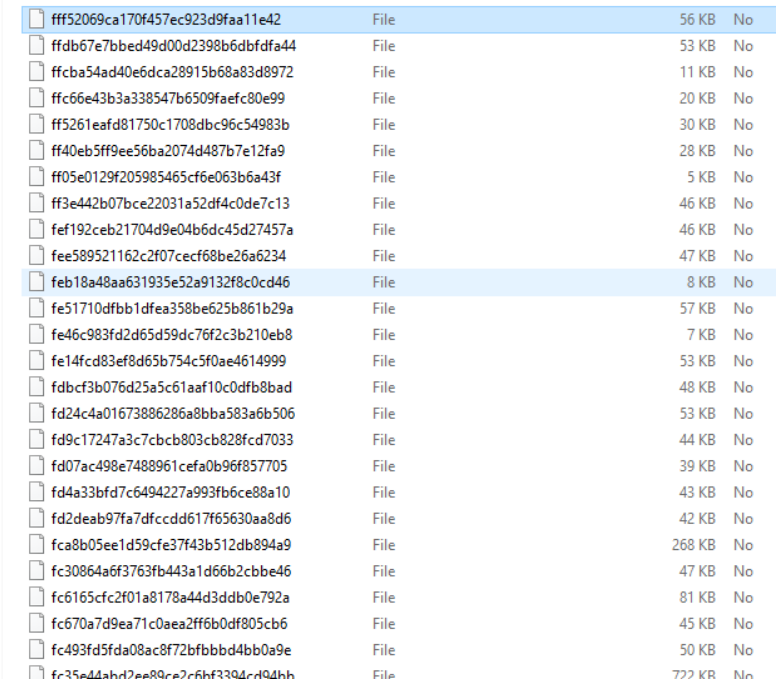
CHƯƠNG 3: THỬ NGHIỆM VÀ ĐÁNH GIÁ

Nội dung chương 3 sẽ tiến hành áp dụng mô hình phát hiện botnet trên các thiết bị IOT đã được đề xuất ở chương 2 vào tập dữ liệu mẫu. Sau đó kết quả sẽ được đưa ra nhật xét và đánh giá chất lượng của mô hình.

3.1. Thu thập và tiền xử lý dữ liệu

3.1.1. Dữ liệu mẫu

Luận văn đã thu thập được 1,326 tệp tin dữ liệu mẫu từ nguồn VirusShare, IoTPOT và các nguồn khác, trong đó có 1,248 tệp tin có chứa mã độc và 78 tệp tin lành tính không chưa mã độc. Tất cả các dữ liệu mẫu này đều là ứng dụng chạy trên hệ điều hành linux.



Hình 3.1. Các tệp tin mẫu chứa mã độc botnet trên các thiết bị IOT

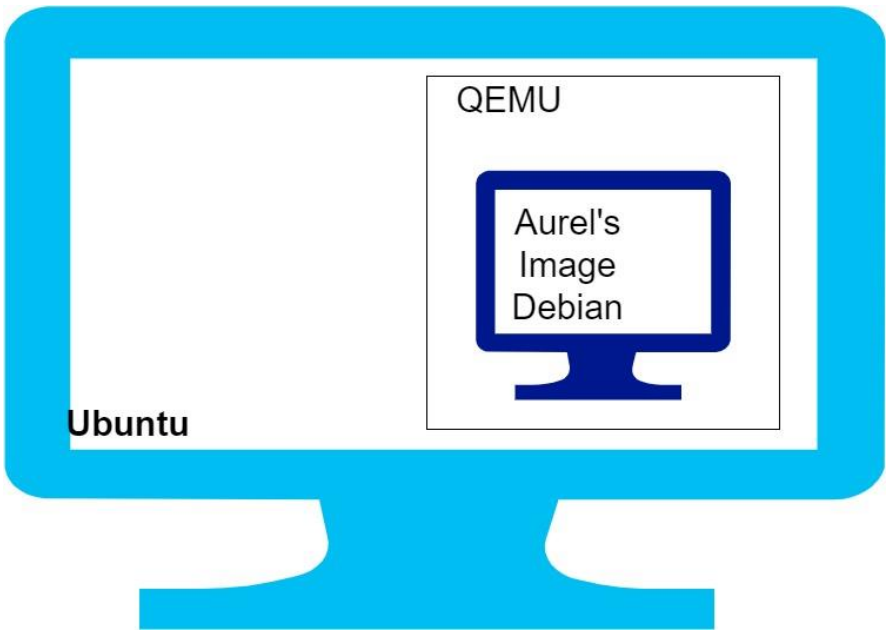
Tập dữ liệu mẫu sẽ được chia thành hai tập con: tập dữ liệu huấn luyện chiếm 75% và tập dữ liệu kiểm thử chiếm 25% của số tệp có mã độc, các tệp tin mẫu có chứa mã độc sẽ được chia ngẫu nhiên. Tuy nhiên, do mô hình học máy của luận văn áp dụng thuật toán One-class SVM nên tập huấn luyện sẽ chỉ chứa các tệp tin mẫu có

mã độc mà không chứa các tệp tin lành tính. Các tệp tin lành tính cùng với 25% tệp tin mã độc chỉ sử dụng để kiểm thử chất lượng mô hình.

3.1.2. QEMU

Để đảm bảo hành vi của mã độc được ghi lại đầy đủ, chính xác và nhanh chóng, máy chủ và máy khách được cấu hình như sau:

- Máy chủ:
  - CPU Intel Xeon CPU E3 – 1230 v6 3.50 GHz
  - Hệ điều hành Ubuntu 16.04 x64
  - Ram 8 Gb
  - Kết nối Internet
  - QEMU
- Máy khách:
  - Ram 4 GB
  - Hệ điều hành: Aurel’s Debian Image
  - Kết nối brige sang máy chủ



Hình 3.2. Mô hình cài đặt máy chủ - máy khách

Máy chủ Ubuntu có các chức năng:

- Cài đặt máy khách QEMU để thu thập các lời gọi hệ thống của tập dữ liệu mẫu;
- Cài đặt và chạy thuật toán để chuyển đổi log các lời gọi hệ thống sang đồ thị lời gọi hệ thống;
- Chạy chức năng Graph2vec để chuyển đổi đồ thị lời họi hệ thống sang vector;
- Cài đặt và huấn luyện thuật toán One-class SVM từ dữ liệu trích chọn được từ bước trên.

Máy khách QEMU có các chức năng:

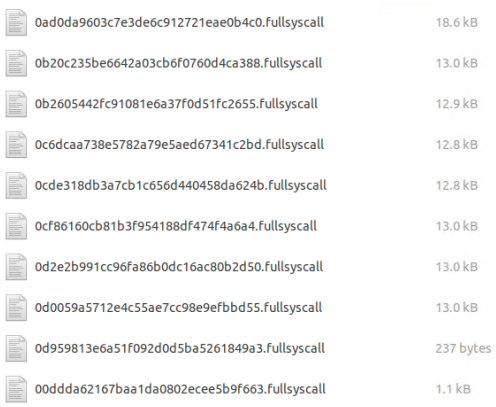
- Cài đặt công cụ Strace tool để thu thập lời gọi hệ thống của tập dữ liệu mẫu;
- Thực thi các tệp tin mã độc và ghi log lời gọi hệ thống ra các tập tin.

3.1.3. Thu thập dữ liệu

Để có thể thu thập được dữ liệu một cách tự động, tác giả đã sao chép các tệp tin mẫu mã độc vào máy khách sau đó chạy đoạn mã GetLog.sh để có thể lấy được log lời gọi hàm hệ thống. Với mỗi log của tệp tin mẫu mã độc sẽ được lưu trữ trong một tệp tin text.

```
execve("./sample/644829dbf4d2f3addffa6fb2a900ea63", ["./sample/644829dbf4d2f3addffa6fb"...], [/* 13 vars
*/]) = 0
ioctl(0, SNDCTL_TMR_TIMEBASE or SNDRV_TIMER_IOCTL_NEXT_DEVICE or TCGETS, {B38400
opost isig icanon echo ...}) = 0
ioctl(1, SNDCTL_TMR_TIMEBASE or SNDRV_TIMER_IOCTL_NEXT_DEVICE or TCGETS, {B38400
opost isig icanon echo ...}) = 0
write(1, "BUILD DONGS\n", 12)          = 12
prctl(PR_SET_NAME, 0x1a8f8, 0, 0, 0)    = 0
gettimeofday({1558425771, 760992}, NULL) = 0
getpid()                                 = 5218
gettimeofday({1558425771, 762627}, NULL) = 0
getpid()                                 = 5218
socket(PF_INET, SOCK_DGRAM, IPPROTO_IP) = 3
connect(3, {sa_family=AF_INET, sin_port=htons(53), sin_addr=inet_addr("8.8.8.8")}, 16) = 0
getsockname(3, {sa_family=AF_INET, sin_port=htons(45076), sin_addr=inet_addr("192.168.42.135")}, [16]) =
0
open("/proc/net/route", O_RDONLY)      = 4
read(4, "I", 1)                          = 1
```

Hình 3.3. Kết quả thu thập được khi thực thi mã độc trên máy khách



Hình 3.4. Các tệp tin nhật ký thu thập được trong môi trường sandbox

3.1.4. Xây dựng đồ thị SCG và đồ thị nhúng

Với kết quả thu thập được từ bước trên, tất cả các tệp tin log lời gọi hệ thống sẽ được sao chép về máy chủ, sau đó tiến hành chạy thuật toán để tiến hành xây dựng đồ thị lời gọi hệ thống và chuyển đồ thị đó thành vector. Với thuật toán xây dựng đồ thị lời gọi hệ thống được mô tả trong chương 2, tác giả đã cụ thể bằng ngôn ngữ python để có thể chạy được trong máy chủ ubuntu. Kết quả trả về với mỗi tệp tin mẫu là một tệp tin có định dạng xml sẽ lưu trữ thông tin các đỉnh, các cạnh của đồ thị lời gọi hàm hệ thống.

```
<?xml version='1.0' encoding='utf-8'?>
<gexf version="1.1" xmlns="http://www.gexf.net/1.1draft" xmlns:viz="http://
www.gexf.net/1.1draft/viz" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.w3.org/2001/XMLSchema-instance">
<graph defaultedgetype="directed" mode="static">
<attributes class="node" mode="static">
<attribute id="0" title="Label" type="string"/></attributes>
<nodes><node id = "11" label = "11">
<attvalues>
<attvalue for="0" value="1"/>
</attvalues>
</node>
<node id = "45" label = "45">
<attvalues>
<attvalue for="0" value="1"/>
</attvalues>
</node>
<node id = "28" label = "28">
<attvalues>
<attvalue for="0" value="1"/>
</attvalues>
</node>
```

Hình 3.5. Cấu trúc lưu trữ dữ liệu của đồ thị lời gọi hàm hệ thống

Bước tiếp theo tác giả sẽ sử dụng mã nguồn mở graph2vec để có thể chuyển đổi đồ thị lời gọi hàm hệ thống thành đồ thị nhúng. Đầu vào của bước này là các tệp

tin đồ thị *.gexf và đầu sẽ là một ma trận 2 chiều với mỗi hàng là một vector của một đồ thị. Ma trận vector này sẽ được lưu trữ trong một tệp tin *.txt có cấu trúc như hình dưới.


```
{
    "../../gexf/9bc488c8188ee79c2c99b32eb63cc55e.gexf.g2v3": [
        -0.17119821906089783,
        -0.0004615240322891623,
        -1.7827434539794922,
        0.3242648243904114,
        0.06580841541290283,
        -1.1540573835372925,
        0.5765106081962585,
        -0.5248183608055115,
        0.6449635028839111,
        -1.5051369667053223,
        1.1897075176239014,
        -0.9498820304870605,
        1.3193600177764893,
        0.14179106056690216,
        1.558129072189331,
        -1.8025147914886475,
        -0.9819702506065369,
        -0.2934807538986206,
        1.187371850013733,
        0.4684920310974121,
        0.9648829102516174,
        -1.5075292587280273,
        1.8924767971038818,
        0.00834315363317728,
        0.47199130058288574,
```


Hình 3.6. Cấu trúc của đồ thị nhúng

Kết quả cuối cùng của bước này chính là các vector đặc trưng sẽ được đưa vào thuật toán học máy trong bước tiếp theo.

3.2. Thử nghiệm

Trong bước này tác giả sẽ cài đặt thuật toán học máy one-class SVM bằng thư viện scikit-learn. Sklearn là một thư viện mã nguồn mở dành cho học máy được thiết kế bằng ngôn ngữ pyhton và trên nền NumPy và SciPy. Sklearn chứa hầu hết các thuật toán học máy hiện đại nhất. Đối với mô hình này tác giả sử dụng lớp **sklearn.svm** để chạy thuật toán one-class SVM. Các bước thử nghiệm mô hình như sau:

Chia dữ liệu thành 2 phần dữ liệu huấn luyện và dữ liệu kiểm thử:

- Dữ liệu huấn luyện: 936 mẫu (chiếm 75% số lượng mã độc);

- Dữ liệu kiểm thử: 312 mẫu (chiếm 25% số lượng mã độc) và 78 mẫu lành tính.

```
# Split data
X_mal = X[y == 0]
X_beg = X[y == 1]
X_train, X_mal_test = train_test_split(X_mal, test_size=.25, random_state=42)
```

Đưa dữ liệu vào huấn luyện và kiểm thử với các đối số nu=0.01 và gamma= 'auto'

```
# Apply
clf = OneClassSVM(nu=.01, gamma='auto')
# clf = LocalOutlierFactor(novelty=True, contamination=.2)
clf.fit(X_train)
y_true, y_pred = y_test, clf.predict(X_test)
```

3.3. Nhận xét đánh giá

3.3.1. Các tiêu chí đánh giá

Để có thể đánh giá được hiệu năng của hệ thống học máy nói chung và trong thuật toán One-class SVM mà luận văn đang sử dụng, cần phải có các tiêu chí đánh giá từ đó mới có thể nhận xét được thuật toán tốt hay xấu. Trong luận văn này tác giả sử dụng 4 tiêu chí đánh giá sau: Accuracy, Precision, Recall, F-measure.

Để có thể tính toán được các tiêu chí trên, chúng ta có Confusion matrix (ma trận nhầm lẫn) như sau:

Lớp C		Được phân loại bởi hệ thống	
		Thuộc	Không thuộc
Nhãn lớp thực sự (đúng)	Thuộc	TP	FN
	Không thuộc	FP	TN

Trong đó:

- TP (true positive): Số lượng dữ liệu thuộc lớp C được phân loại chính xác vào lớp C;
- FP (false positive): Số lượng dữ liệu bên ngoài bị phân loại nhầm vào lớp C;

- TN (true negative): Số lượng dữ liệu không thuộc lớp C được phân loại (chính xác);
- FN (false negative): Số lượng dữ liệu thuộc lớp C bị phân loại nhầm (vào cáclớp khác C).

Accuracy (Tính chính xác) là mức độ phân lớp chính xác của hệ thống đã được huấn luyện đối với tập dữ liệu được dùng để kiểm thử.

$$Accuracy = \frac{Số\ phán\ đoán\ chính\ xác}{Tổng\ số\ phán\ đoán}$$

Hoặc

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

Precision đối với lớp C bằng tổng số các ví dụ thuộc lớp C được phân loại chính xác chia cho tổng số các ví dụ được phân loại vào lớp C.

$$Precision\ (C) = \frac{TP}{TP + FP}$$

Recall đối với lớp C bằng tổng số các ví dụ thuộc lớp C được phân loại chính xác chia tổng số các ví dụ thuộc lớp C.

$$Recall(C) = \frac{TP}{TP + FN}$$

F-measure

$$F - measure = \frac{2.Precision.Recall}{Precision + Recall}$$

F-measure là một trung bình điều hòa (harmonic mean) của các tiêu chí Precision và Recall. F-measure có xu hướng lấy giá trị nào nhỏ hơn giữa 2 giá trị Precision và Recall. F-measure có giá trị lớn nếu cả 2 giá trị Precision và Recall đều lớn.

3.3.2. Đánh giá kết quả

Kết quả chạy được với bộ dữ liệu mẫu thu được như sau:

Confusion matrix:

Bảng 3.1. Kết quả Confusion matrix

	Được phân loại bởi hệ thống	
	Mã độc	Lành tính
Mã độc	300	12
Lành tính	8	70

Dựa vào Confusion matrix ta có thể thấy:

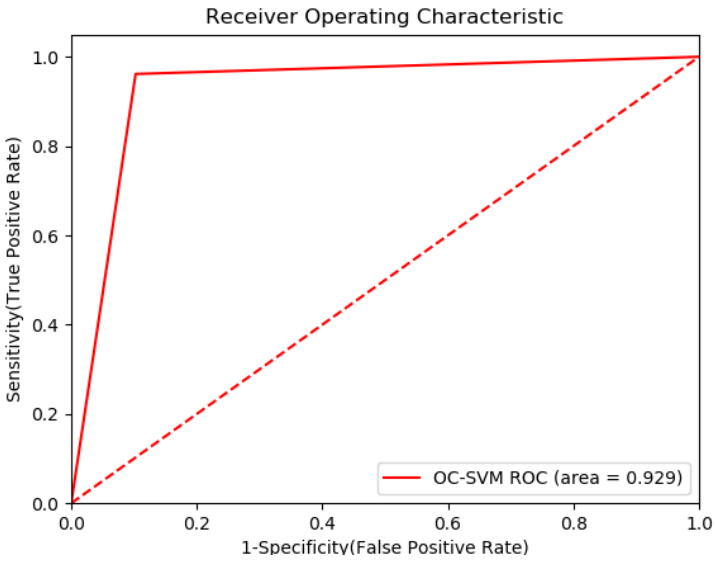
- Với 312 mẫu mã độc để kiểm thử thì mô hình phân lớp đúng 300 mẫu đúng vào lớp mã độc và 12 mẫu bị phân lớp sai vào lớp lành tính;
- Với 78 mẫu lành tính mô hình phân lớp đúng 70 mẫu vào lớp lành tình và 8 mẫu bị phân lớp sai vào lớp mã độc.

	Precision	Recall	F1- measure	support
Lành tính	0.85	0.90	0.88	78
Mã độc	0.97	0.96	0.97	312

Accuracy: 0.949

ROC AUC: 0.929

Falase Positive rate: 0.103



Hình 3.7. Đường ROC của mô hình đề xuất trong kiểm thử

Dựa vào kết quả kiểm thử mô hình đề xuất, có thể đưa ra một số nhận xét:

- Accuracy của mô hình đề xuất là: 94.9%, đối với một bài toán phân loại một lớp thì acc=94.9% là kết quả khá tốt;
- Đường ROC với giá trị AUC=0.929 có thể thấy hiệu quả của thuật toán với độ chính xác cao khi mà giá trị AUC tiến rất gần với giá trị lý tưởng là 1;

3.4. Kết luận chương

Chương 3 đã trình bày về quá trình thử nghiệm mô hình phát hiện mã độc IoT botnet, bao gồm các bước thực hiện, kết quả thực nghiệm, đánh giá và nhận xét về mô hình. Với 1,326 mẫu dữ liệu được chia thành 75% dành cho dữ liệu huấn luyện và 25% dành cho dữ liệu kiểm thử đưa vào mô hình huấn luyện và kiểm thử. Kết quả đạt được khá tốt đó là acc = 94.9%.

KẾT LUẬN

1. Những đóng góp của luận văn

Luận văn đã nghiên cứu về phương pháp phát hiện mã độc bonet trên các thiết bị IoT dân dụng vào thuật toán học máy one-lass SVM và đã đạt được một số kết quả như sau:

- Giới thiệu tổng quan về thiết bị IoT dân dụng;
- Giới thiệu tổng quan về mã độc IoT botnet và các phương pháp phát hiện mã độc;
- Trình bày cơ sở lý thuyết và đưa ra mô hình phát hiện mã độc IoT botnet bằng kỹ thuật học máy one-class SVM;
- Tiến hành thực nghiệm, đánh giá kết quả thu được.

Tổng kết lại, quá trình thử nghiệm mô hình được thực hiện như sau: Sử dụng QEMU để giả lập kiến trúc chíp MIPS, ARM và sử dụng công cụ Strace tool để thu thập nhật ký lời gọi hàm hệ thống của tệp dữ liệu mẫu, sau đó xây dựng đồ thị lời gọi hàm hệ thống, bước tiếp là sử dụng thư viện graph2vec để trích chọn đặc trưng từ đồ thị lời gọi hàm hệ thống và cuối cùng là đẩy các vector đặc trưng vào để huấn luyện thuật toán học máy one-class SVM dựa vào thư viện mã nguồn mở Sklearn.

2. Hướng phát triển tiếp theo

Hiện tại, luận văn chỉ mới sử dụng thuật toán học máy one-class SVM để áp dụng học máy và chỉ mới các thiết bị IoT dân dụng. Hướng tiếp theo sẽ là nghiên cứu các thuật toán học sâu để có thể áp dụng vào bài toán phát hiện mã độc botnet trên các thiết bị IoT. Bên cạnh đó có thể mở rộng phạm vi nghiên cứu thêm các loại thiết bị IoT khác từ đó có thể xây dựng được một hệ thống sandbox đa nền tảng cho các thiết bị IoT. Ngoài ra luận văn cũng sẽ thu thập thêm tập dữ liệu mẫu để có thể tăng độ chính xác cho kết quả phát hiện, bởi tập dữ liệu hiện tại chỉ mới có một số lượng nhỏ các tệp tin mã độc botnet.

DANH MỤC CÁC TÀI LIỆU THAM KHẢO

[1] Sử Ngọc Anh, Lê Hải Việt, Nguyễn Huy Trung, Ngô Quốc Dũng, Xây dựng mô hình thu thập phát hiện tấn công mạng sử dụng thiết bị IoT, Hội thảo lần thứ II: Một số vấn đề chọn lọc về an toàn an ninh thông tin – TP. Hồ Chí Minh, 2017.

[2] Từ Minh Phương, Giáo trình nhập môn trí tuệ nhân tạo, Học viện Công nghệ bưu chính viễn thông, 2014 [185-192].

[3] Lê Hải Việt, Ngô Quốc Dũng, Trần Nghi Phú, Nguyễn Ngọc Toàn, Xây dựng mô hình phát hiện mã độc trên thiết bị định tuyến bằng tác tử, Hội thảo quốc gia lần thứ XX: Một số vấn đề chọn lọc của Công nghệ thông tin và truyền thông – Quy Nhơn, 2017.

[4] Hernandez, Grant, Orlando Arias, Daniel Buentello, and Yier Jin, Smart nest thermostat: A smart spy in your home, Black Hat USA, 2014.

[5] Karim A, Salleh RB, Shiraz M, Shah SAA, Awan I, Anuar NB, Botnet detection techniques: review, future trends, and issues, J Zhejiang Univ Sci C 15(11):943–983, 2014.

[6] Kishore Angrishi, Turning Internet of Things(IoT) into Internet of Vulnerabilities (IoV) : IoT Botnets, arXiv preprint arXiv:1702.03681, 2017.

[7] Antonakakis M, Perdisci R, Dagon D, Lee W, Feamster N, Building a dynamic reputation system for DNS, USENIX security symposium, pp 273–290, 2010.

[8] Kheir N, Tran F, Caron P, Deschamps N, Mentor: positive DNS reputation to skim-off benign domains in botnet C&C blacklists, Cuppens-Boulahia N, Cuppens F, Jajodia S, El Kalam AA, Sans T (eds) ICT systems security and privacy protection. Springer, Berlin, Heidelberg, pp 1–14, 2014.

[9] Shin S, Xu Z, Gu G, EFFORT: efficient and effective bot malware detection, Proceedings IEEE INFOCOM. IEEE, pp 2846–2850, 2012.

[10] Vitor Hugo Bezerra , Victor G. Turrisi da Costa , Sylvio Barbon Junior , Rodrigo Sanches Miani , Bruno Bogaz Zarpelao, One-class Classification to Detect Botnets in IoT devices, 2018.

[11] Kishore Angrishi, Turning Internet of Things(IoT) into Internet of Vulnerabilities (IoV) : IoT Botnets, 2017.

[12] Andrei Costin, Large Scale Security Analysis of Embedded Devices' Firmware, 2016.

[13]Muhammad Junaid Bohio, "Analysis of a MIPS Malware", 2015.

[14] https://en.wikipedia.org/wiki/QEMU truy cập ngày 15/09/2019.

[15] https://en.wikipedia.org/wiki/Strace truy cập ngày 15/09/2019.

[16] https://towardsdatascience.com/graph-embeddings-the-summary-cc6075aba007 truy cập ngày 15/10/2019.

[17] https://en.wikipedia.org/wiki/System_call truy cập 02/11/2019.

[18] Kaspersky Lab: https://www.kaspersky.com