HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

Trần Hoàng Anh

NGHIÊN CỨU PHƯƠNG PHÁP LAI TRONG PHÁT HIỆN MÃ ĐỘC BOTNET TRÊN 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 - NĂM 2021

LỜI CAM ĐOAN

Tôi cam đoan đây là công trình nghiên cứu của tôi dưới sự hướng dẫn của thầy TS.Ngô Quốc Dũng.

Các số liệu kết quả trong luận văn là trung thực và chưa bao giờ được sử dụng để bảo vệ một học vị nào. Những số liệu được trích dẫn, sử dụng trong luận văn là trung thực được chỉ rõ nguồn trích dẫn và được phép sử dụng.

HỌC VIÊN THỰC HIỆN

Trần Hoàng Anh

LỜI CẢM ƠN

Trong quá trình học tập, nghiên cứu và hoàn thiện luận văn, tác giả đã nhận được sự động viên, khuyến khích và tạo điều kiện giúp đỡ nhiệt tình của các cấp lãnh đạo, anh chị em, bạn bè đồng nghiệp và gia đình. Đây là nguồn động viên quý giá giúp tác giả có điều kiện nghiên cứu và hoàn thành luận văn thạc sĩ.

Tác giả xin cảm ơn các thầy cô khoa Sau đại học, các thầy giáo, cô giáo Học viện Bưu chính Viễn thông trong quá trình giảng dạy đã truyền đạt để tác gải có được nền tảng kiến thức hỗ trợ rất lớn trong quá trình làm luận văn thạc sĩ.

Tác giả bày tỏ lòng biết ơn sâu sắc tới trưởng nhóm nghiên cứu Nguyễn Huy Trung và các thành viên nhóm nghiên cứu AISoft đã hỗ trợ, đóng góp ý kiến cho tác giả trong suốt quá trình nghiên cứu và hoàn thành luận văn thạc sĩ.

Đặc biệt, tác giả xin bày tỏ lòng biết ơn sâu sắc tới TS. Ngô Quốc Dũng, người đã trực tiếp hướng dẫn, tận tình chỉ bảo, giúp đỡ tác giả tiến hành các hoạt động nghiên cứu khoa học để hoàn thành luận văn này.

Với thời gian nghiên cứu còn hạn chế, thực tiễn cuộc sống lại vô cùng sinh động, luận văn khó tránh khỏi những sai sót, khuyết điểm. Tác giả rất mong nhận được các ý kiến đóng góp chân thành từ cá thầy giáo, cô giáo, đồng nghiệp, bạn bè.

Hà Nội, ngày 18 tháng 5 năm 2021

HỌC VIÊN

Trần Hoàng Anh

MỤC LỤC

DANH MỤC CÁC THUẬT NGỮ, CHỮ VIẾT TẮT

Viết tắt	Tiếng Anh	Tiếng Việt
API	Application Programming Interface	Giao diện lập trình ứng dụng
ARM	Advanced RISC Machine	Kiến trúc dạng RISC cho các vi xử lý máy tính
C&C	Command and control	Máy chủ điều kiển và kiểm soát
CFG	Control Flow Graph	Đồ thị luồng điều khiển
CNN	Convolutional Neural Network	Mạng nơ-ron tích chập
CPS	Cyber Physical Systems	Hệ thống tự động hóa có khả năng tự hành thông minh
CVE	Common Vulnerabilities and Exposures	Hệ thống Lỗ hổng và Phơi nhiễm phổ biến
DDoS	Distributed Denial of Service	Từ chối dịch vụ phân tán
DHT	Distributed hash table	Giao thức bảng băm phân tán
DNS	Domain Name Servers	Máy chủ phân giải tên miền
DSL	Digital Subcriber Line	Kênh thuê bao số
DT	Decision Tree	Cây quyết định
ELF	Executable and Linkable Format	Định dạng có thể thực thi và có thể liên kết
FCG	Function Call Graph	Đồ thị hàm gọi
FLF	Function Length Frequency	Tần suất độ dài hàm
FTP	File Transfer Protocol	Giao thức truyền tập tin
HTTP	HyperText Transfer Protocol	Giao thức truyền tải siêu văn bản
IoT	Internet of Things	Công nghệ vạn vật kết nối
IP	Internet Protocol	Giao thức Internet
IRC	Internet Relay Chat	Chat chuyển tiếp Internet
k-NN	K- nearest neighbor	k-hàng xóm gần nhất
LSTM	Long Short Term Memory	Mạng bộ nhớ dài - ngắn

Viết tắt	Tiếng Anh	Tiếng Việt
MIPS	Microprocessor without Interlocked Pipeline Stages	Kiến trúc dạng RISC rút gọn
OpCode	Operation Code	Mã thực thi
P2P	Peer-to-Peer	Mạng ngang hàng
PDoS	Permanent Denial of Service	Từ chối dịch vụ lâu dài
PSI	Printable String Infomation	Thông tin chuỗi in
RAM	Random Access Memory	Bộ nhớ truy cập ngẫu nhiên
RCE	Remote Code Execution	Thực thi mã từ xa
RF	Random Forest	Rừng quyết định
ROC	Receiver Operating Characteristic	Đặc trưng hoạt động của bộ thu nhận
SCG	System Call Graph	Đồ thị lời gọi hệ thống
SOHO	Small Office/Home Office	Thiết bị dùng cho văn phòng/nhà
SSH	Secure Shell	Giao thức mạng dùng để thiết lập kết nối mạng bảo mật
SVM	Support Vector Machine	Máy véc tơ tựa
TCP	Transmission Control Protocol	Giao thức điều khiển truyền vận
TFTP	Trivial File Transfer Protocol	Giao thức truyền tập tin đơn giản
UDP	User Datagram Protocol	Giao thức dữ liệu ngắn
URL	Uniform Resource Locator	Định vị tài nguyên thống nhất

DANH MỤC BẢNG

DANH MỤC HÌNH VẼ

MỞ ĐẦU

Trải qua gần 250 năm với ba cuộc cách mạng công nghiệp, những lợi ích mà cuộc cách mạng công nghiệp lần thứ 4 mang lại, không chỉ tạo ra biến đổi về lượng mà cả về chất đối với cuộc sống, xã hội loài người. Hiện nay, cuộc cách mạng công nghiệp 4.0 đang làm biến đổi nhanh chóng nền công nghiệp ở mọi quốc gia và diễn ra trên phạm vi toàn cầu. Với nhiều thành phần khác nhau, đặc điểm nổi bật nhất của cuộc cách mạng công nghiệp lần thứ 4 đó là việc dịch chuyển các hệ thống máy móc sản xuất truyền thống sang các hệ thống tự động hoá có khả năng tự hành một cách thông minh (Cyber Physical Systems - CPSs), xóa nhòa biên giới giữa thực và ảo. Các hệ thống này có thể sơ đồ hóa, giám sát và quản lý cả một ngành công nghiệp hoàn toàn thông qua không gian mạng, nâng tính hiệu quả của quá trình sản xuất lên một tầm cao mới.

Hiện nay, cuộc cách mạng công nghiệp 4.0 vẫn đang ở giai đoạn sơ khai. Xu hướng số hóa hay công cuộc chuyển đổi số đang xuất hiện ở mọi lĩnh vực, mọi ngành kinh tế, bao trùm cả công nghiệp, nông nghiệp, dịch vụ; cho tới các hoạt động thương mại, sản xuất, phân phối, lưu thông hàng hóa, các hạ tầng trong xã hội như giao thông vận tải, logistics, tài chính, ngân hàng... Trong những bước chuyển đổi đó, công nghệ Vạn vật kết nối (IoT – Internet of Things) phát triển một cách mạnh mẽ và là một trong những thành phần không thể thiếu của mọi hệ thống hiện nay. Các thiết bị IoT được ứng dụng không chỉ trong trong công nghiệp mà còn góp phần cải thiện đời sống của con người. Nghiên cứu của Gary David [8] đã dự đoán rằng, vào năm 2020, tổng số lượng các thiết bị được kết nối trên thế giới sẽ chạm mốc 50 tỷ thiết bị. Bên cạnh đó, theo một khảo sát của Satista [32], thị trường thiết bị IoT cũng đang có xu hướng tăng nhanh và dự đoán sẽ đạt tới giá trị 248 tỉ đô la vào cuối năm 2020.

Phần lớn các thiết bị IoT, vì mục đích triển khai một cách dễ dàng trên diện rộng, đã được sản xuất hàng loạt với khả năng bảo mật khá lỏng lẻo [14]. Trong một thời đại mà các thiết bị IoT vốn phân tán khắp nơi trên thế giới, sự kém bảo mật của những thiết bị này đã trở thành đích ngắm của vô vàn các cuộc tấn công bằng mã độc. Trong số những cuộc tấn công đó, loại tấn công lây lan với tốc độ chóc mặt nhất và

nguy hiểm nhất là các cuộc tấn công sử dụng mã độc IoT Botnet. Trên thực tế, trong thời gian gần đây, đã có sự gia tăng đáng kể về cả số lượng lẫn mức độ của các cuộc tấn công sử dụng mã độc IoT Botnet [6]. Có thể kể tới một cuộc tấn công gần đây nhất là cuộc tấn công từ chối dịch vụ phân tán (DDoS) năm 2016 vào hệ thống của Dyn, một trong những công ty cung cấp tên miền, có máy chủ được đặt tại Mỹ. Đây được coi là một trong những cuộc tấn công từ chối dịch vụ phân tán lớn nhất từ trước tới nay với tốc độ 1,2 Terabits trên giây [13]. Mã độc IoT Botnet Mirai - thủ phạm chính của cuộc tấn công, đã lây lan trên Internet thông qua các thiết bị như webcam, các thiết bị camera ghi hình và các thiết bị định tuyến có chạy một số phiên bản BusyBox. Hiện nay, các loại mã độc IoT Botnet phức tạp như Mirai đang dần xuất hiện với một tỷ lệ đáng báo động [5]. Tuy nhiên, do đặc điểm hạn chế về các tài nguyên trong hệ thống, việc phát hiện mã độc IoT Botnet đang ngày càng đa dạng và đổi mới là cực kỳ khó khăn. Số lượng các nghiên cứu về phát hiện IoT Botnet hiện nay vẫn còn khá ít [4]. Vì vậy, đây có thể coi là một hướng nghiên cứu ứng dụng khá cấp thiết trong tình hình hiện nay.

Với những lý do đã đề cập ở trên, tôi xin lựa chọn đề tài “Nghiên cứu phương pháp lai trong phát hiện mã độc Botnet trên thiết bị IoT”.

Chương 1: MÃ ĐỘC IOT BOTNET VÀ CÁC HƯỚNG PHÁT HIỆN

1.1. Tổng quan về mã độc IoT và IoT Botnet

1.1.1. Khái niệm mã độc IoT

Mã độc, gốc tiếng Anh là Malware, là sự kết hợp của từ Malicous và Software, thường dùng để chỉ tất cả những phần mềm không mong muốn. Theo G. McGraw và G. Morrisett, mã độc là bất kỳ đoạn mã nào được thêm, thay đổi hoặc xoá bỏ từ phần mềm của hệ thống nhằm lý do gây ảnh hưởng hoặc phá hoại đến chức năng của hệ thống đó.

Trong những năm gần đây, dưới sự phát triển vũ bão của công nghệ Internet of Things (gọi tắt là IoT), các loại mã độc nhắm tới các thiết bị IoT cũng tăng lên nhanh chóng. Về cấu tạo, thiết bị IoT có các đặc trưng cơ bản của một hệ thống nhúng và khác với các thiết bị máy vi tính ở các điểm sau:

- Về chức năng: Máy vi tính là vạn năng, còn hệ nhúng là chuyên dụng.

- Về tài nguyên: Hệ nhúng thường có tài nguyên hạn chế hơn nhiều so với máy vi tính hoặc chỉ mạnh về một tiêu chí nào đó phục vụ chức năng chuyên dụng của mình, ví dụ hoặc bộ nhớ rất lớn, hoặc bộ tính toán rất nhanh v.v. Nhưng thông thường dung lượng bộ nhớ thấp (RAM), bộ nhớ flash bị hạn chế về dung lượng thường được sử dụng để lưu trữ trong các firmware

- Về kiến trúc phần cứng: Hệ nhúng thường rất đa dạng về kiến trúc tùy theo chức năng mà hệ nhúng đó đảm nhiệm, kiến trúc bộ xử lý cũng khác nhiều so với máy vi tính. Không hỗ trợ kiến trúc x86, thường sử dụng kiến trúc ARM, MIPS, PowerPC, Sparc. Bộ vi xử lý dựa trên kiến trúc MIPS sử dụng phổ biến trong thiết bị định tuyến Cisco, Linksys, Mikrotk, modem DSL v.v. còn bộ vi xử lý dựa trên nền ARM (Advanced RISC Machines) được sử dụng phổ biến trong điện thoại thông tinh, TVs, thiết bị di động.

Như vậy, ta có thể hiểu, mã độc IoT là loại mã độc được tạo ra để nhắm tới các thiết bị IoT, lợi dụng các điểm yếu sẵn có trong lỗi kiến trúc và mã mềm kém chất

lượng của thiết bị để thực hiện các hành động như đánh cắp thông tin cá nhân của người sử dụng, xây dựng mạng botnet hoặc thậm chí phá hủy cả hạ tầng mạng.

Các hướng tấn công của mã độc IoT rất đa dạng, tuy nhiên chúng chủ yếu nhắm vào các tầng vật lý và tầng mạng của thiết bị IoT. Bảng 1.1 liệt kê 15 kiểu tấn công của mã độc IoT, được trình bày trong nghiên cứu của Dange và các cộng sự [6].

Bảng 1.1: Danh sách cách kiểu tấn công của mã độc IoT

STT	Kiểu tấn công	Tầng	Hoạt động	Ảnh hưởng
1	Jamming	Vật lý	Giảm tỷ lệ signal-to-noise khiến cho bên thu nhận bị gián đoạn	Tạo nên một cuộc tấn công từ chối dịch vụ
2	Tấn công sinkhole	Mạng	Kẻ tấn công cung cấp một con đường tối ưu tới các máy trạm. Tất cả thông tin đi qua đường này đều có thể bị phục hồi bởi kẻ tấn công	Gây mất tính bí mật của dữ liệu
3	Can thiệp vào nút mạng	Vật lý	Thay đổi, chỉnh sửa chức năng hoặc dữ liệu của các nút mạng	Gây mất tính toàn vẹn của dữ liệu
4	Tấn công blackhole	Mạng	Chèn vào một nút mạng mới hoặc điều khiển nút mạng đã có sẵn khiến cho tất cả các nút mạng lân cận thay đổi bảng định tuyến và chỉ truyền dữ liệu qua nút mạng này. Nút mạng nhận được gói tin sẽ ngừng chuyển tiếp gói tin.	Gây mất dữ liệu

5	Tấn công wormhole	Mạng	Cuộc tấn công có một hoặc nhiều nút mạng độc hại và một đường hầm giữa chúng. Nút mạng tấn công sẽ chặn bắt các gói tin từ một địa điểm và truyền chúng tới một nút ở rất xa.	Gây mất tính bí mật và toàn vẹn của dữ liệu
6	Tấn công sybil	Mạng	Một nút mạng có thể có nhiều định danh khác nhau. Các định danh này có thể được dùng để giả mạo hoặc phát tán mã độc.	Gây mất tính toàn vẹn
7	Tấn công Selective Forwarding	Mạng	Nút mạng độc hại đóng vai trò như một thiết bị định tuyến và lựa chọn gói tin để truyền đi hoặc loại bỏ.	Gây mất tính riêng tư và toàn vẹn của dữ liệu
8	Tấn công Hello flood	Mạng	Kẻ tấn công liên tục gửi đi thông điệp HELLO để xác định vị trí của các thiết bị lân cận. Thiết bị nhận được thông điệp này sẽ phải dành một phần tài nguyên để trả lời, dẫn đến việc tăng lưu lượng mạng và tiêu tốn năng lượng.	Tạo nên một cuộc tấn công từ chối dịch vụ
9	Tấn công man-in-the middle	Mạng	Kẻ tấn công phát lại thông điệp một cách bí mật và có thể thay đổi nội dung giao tiếp trong khi hai bên liên	Gây mất tính bí mật

			lạc vẫn cho rằng đang được kết nối trực tiếp với nhau.	
10	Tấn công từ chối dịch vụ	Mạng	Kẻ tấn công khiến cho thiết bị hoặc hệ thống mạng ngưng trệ hoạt động đối với người dùng hợp pháp.	Gây mất tính sẵn sàng của hệ thống
11	Tấn công gây nghẽn	Mạng	Một hoặc nhiều nút mạng độc hại liên tục gửi đi các thông điệp gây tắc nghẽn.	Gây mất tính sẵn sàng của hệ thống
12	Tấn công phát lại	Mạng	Kẻ tấn công gửi lại các thông tin đã bị chặn bắt trên đường truyền	Gây mất tính riêng tư của dữ liệu
13	Tấn công Sleep Deprivation	Vật lý	Mục tiêu chính của loại tấn công này là khiến cho thiết bị luôn hoạt động và không thể đi vào trạng thái tiết kiệm năng lượng.	Thiết bị tiêu tốn năng lượng nhanh hơn dẫn đến cạn nguồn và ngừng hoạt động. Hệ thống mất tính sẵn sàng.
14	Chèn nút mạng độc hại	Vật lý	Một nút mạng độc hại mới được chèn giữa hai hoặc nhiều nút mạng khác. Kẻ tấn công điều khiển nút mạng này và có thể thay đổi, chỉnh sửa dữ liệu đi qua nút mạng.	Gây mất tính toàn vẹn của dữ liệu
15	Xây dựng mạng lưới Botnet	Vật lý và mạng	Các thiết bị trong mạng bị biến thành các bot. Mỗi bot lại tìm đến các nút mạng yếu và biến chúng thành bot, từ	Hệ thống mất tính sẵn sàng, tính toàn vẹn và dẫn đến mất cả tính bí mật.

			đó xây dựng nên mạng lưới Botnet. Đây là một chu trình tuần hoàn, cho đến khi tất cả các thiết bị trong hệ thống mạng trở thành bot. Với mạng lưới Botnet, kẻ tấn công có thể chiếm quyền điều khiển hệ thống mạng, hack dữ liệu và thực hiện tấn công từ chối dịch phân tán.	

(Nguồn: IoT Botnet: "The Largest Threat to the IoT Network")

Như vậy, về tổng thể, mã độc IoT là loại mã độc có các hoạt động đặc trưng như: thực hiện tấn công từ chối dịch vụ phân tán; rà quét các cổng đang mở của các dịch vụ IoT như FTP, SSH hay Telnet; hay thực hiện tấn công vét cạn để chiếm quyền điều khiển thiết bị IoT.

1.1.2. Phân loại mã độc IoT

Dựa trên những đặc điểm của thiết bị IoT và các tính chất đặc trưng của mã độc, ta có thể phân loại độc IoT thành 5 loại như sau:

- Worm: Mã độc truy cập hệ thống mà không cần quyền người dùng và thực hiện các hành vi độc hại tiềm tàng với khả năng nhân bản lớn. Sâu mã độc có thể làm thay đổi các hành động bình thường của thiết bị. Có thể lấy ví dụ như sâu Stuxnet thực hiện tấn công vào các bộ điều khiển lập trình ở các nhà máy điện hạt nhân của Iran, hoặc mã độc Linux.Darlloz với khả năng lây nhiễm trên diện rộng nhiều thiết bị như thiết bị IoT dân dụng, hộp set-up.

- Trojan: Là các đoạn mã độc xuất hiện dưới hình thức các chương trình hợp lệ. Sau khi kích hoạt, mã độc Trojan sẽ tấn công thiết bị chủ bằng cách tạo ra các cửa hậu để mã độc có thể truy cập dễ dàng. Có thể kể đến như Trojan SoundMiner với khả năng trích xuất dữ liệu từ các thiết bị android.

- Rootkit: Được thiết kế để truy cập từ xa tới thiết bị thông qua việc chỉnh sửa nhân của hệ điều hành hoặc firmware của thiết bị. Một ví dụ về mã độc IoT Rootkit là Cloaker, với khả năng khai thác đặc trưng của vi xử lý ARM để tự ẩn giấu chính nó.

- Spyware: Mã độc ẩn mình trong thiết bị để thu thập thông tin mà người dùng không biết, đồng thời có thể giám sát các hoạt động web của thiết bị. Ví dụ về mã độc IoT Spyware như mã độc Flame (sKyWIper), một biến thể của spyware có thể tấn công diện rộng với mục đích không chỉ đánh cắp thông tin mà còn lắng nghe các tín hiệu microphone, bật Bluetooth trên thiết bị nếu có và gửi thông tin dò quét được tới thiết bị điều khiển gần nhất của hacker.

- Botnet: Mã độc này lây nhiễm và khai thác các thiết bị IoT nhằm thực hiện các cuộc tấn công diện rộng như tấn công từ chối dịch vụ phân tán. Các ví dụ tiêu biểu về mã độc IoT Botnet có thể kể tới như mã độc Linux.Mirai, Linux.Bashlite.

Ngoài ra, có thể phân loại mã độc IoT dựa trên hình thức tấn công của mã độc, bao gồm hai hình thức tấn công chính theo Aohui Wang [3] là: Tấn công vét cạn (brute-force) và Tấn công thực thi mã từ xa (Remote Code Execution – RCE). Trong đó, tấn công vét cạn là hình thức tấn công chủ yếu hiện nay. Nguyên nhân là do có rất nhiều thiết bị IoT thường sử dụng các tài khoản đăng nhập và mật khẩu mặc định yếu. Theo một báo cáo của ESET [34], có khoảng 15% các thiết bị định tuyến sử dụng mật khẩu yếu với tài khoản đăng nhập "admin". Hình thức tấn công này có thể dễ dàng bị xử lý thông qua việc người dùng tự thay đổi tên đăng nhập và mật khẩu phức tạp hơn, hoặc các hãng sản xuất thiết lập mật khẩu mặc định an toàn hơn. Bên cạnh đó, hình thức Tấn công thực thi mã từ xa lại dựa trên các lỗ hổng hệ thống khó được vá hơn như buffer-overflow, các dịch vụ không được mã hóa hoặc mã hóa kém... Do vậy, đây có thể là hình thức tấn công phổ biến trong tương lai gần.

1.1.3. Mã độc IoT Botnet và nguy cơ tấn công từ chối dịch vụ

IoT Botnet là một mạng lưới được tạo nên bởi các thiết bị đã lây nhiễm IoT bot. Về cơ bản, chức năng của IoT bot cũng tương tự như một bot truyền thống, tức là nó sẽ rà quét các thiết bị có chứa lỗ hổng bảo mật, lây nhiễm và biến thiết bị đó thành

một IoT bot mới. Mạng lưới IoT Botnet sẽ thường được điều khiển bởi một botmaster, cho phép thực hiện các hoạt động với sự hỗ trợ của các bot như tấn công từ chối dịch vụ, phát tán thư rác, tấn công lừa đảo, click ảo hoặc spyware. Nguyên nhân khiến các thiết bị IoT bị biến thành bot chủ yếu là do thiết chính sách bảo mật, bị lây nhiễm virus hoặc mở email có gắn kèm mã độc. Cách gọi mã độc IoT Botnet được đề cập trong luận văn là cách gọi chung cho toàn bộ các loại mã độc IoT được botmaster cài cắm vào các thiết bị IoT nạn nhân và điều khiển chúng thực hiện các hành động như tấn công từ chối dịch vụ...

Theo nghiên cứu của Allix [15] và cộng sự, hầu hết mã độc hiện nay được sinh ra thông qua việc sao chép mã nguồn hoặc biến thể của cùng mã độc gốc. Điều này cũng được thể hiện khá rõ đối với mã độc IoT Botnet thông qua các phân tích đánh giá một số họ mã độc IoT đã thực hiện các cuộc tấn công từ chối dịch vụ [13] dưới đây:

- Linux.Hydra: Mã độc đầu tiên lây nhiễm trên các thiết bị IoT. Linux.Hydra xuất hiện vào năm 2008, được công bố ở dạng mã nguồn. Chức năng của mã độc là nhắm tới tấn công các thiết bị dựa trên nền tảng kiến trúc MIPS.

- 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 litte-endian 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.

- 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 (một giá trị string xuất hiện trong tiêu đề của tập tin mã độc). 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.

- Tsunami/Kaiten: 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. Tsunami còn có thể thực hiện tấn công bằng một số kỹ thuật phức tạp như HTTP Layer 7 Flood, TCP XMASS.

- Aidra/LightAidra/Zendran: 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. Mã độc này 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. Chúng có thể biên dịch dựa trên nhiều kiến trúc khác nhau như MIPS, ARM, PowerPC và lây nhiễm trên các thiết bị IoT có nền tảng Linux, khai thác lỗ hổng CVE-2014-6271.

- Spike/Dofloo/MrBlack/Wrkatk/Sotdas/AES.DdoS: Mã độc này xuất hiện năm 2014, sử dụng Agent-handler thay cho IRC-based trong kiến trúc mạng botnet so với các mã độc trước. Mã độc này có 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.

- Bashlite/Lizkebab/Torlus/Gafgyt: Xuất hiện vào năm 2014, có nhiều đặc điểm tương tự như dòng mã độc Spike, mạng botnet mà chúng xây dựng dựa trên kiến trúc Agent-Handler. 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 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. Mục tiệu chính của mã độc này là các thiết bị SOHO có kiến trúc MIPS, ARM.

- 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.

- Mirai: Mirai là một phần mềm độc hại nền tảng Linux trên thiết bị IoT. Mã độc rà soát các thiết bị kém an toàn, biến chúng thành mạng botnet, rồi sử dụng trong tấn công DDoS, và phát tán thông qua Telnet. Năm 2015, các nhà nghiên cứu đến từ công ty bảo mật Nga Dr.Web đã phát hiện ra một loại Trojan mới trên Windows, được thiết kế nhằm mục đích giúp tin tặc phát tán mã độc Mirai sang nhiều thiết bị

khác nhau. Đây là một biểu hiện rõ nét của tính liên kết trong phát tán và đa nền tảng của mã độc trên IoT, đó cũng là lý do Mirai đã thực hiện các vụ tấn công từ chối dịch vụ phân tán có tính quy mô rất lớn.

- The Moon: Tháng 2/2014, tiến sĩ Johannes Ullrich của Học viện kỹ thuật SANS đã phát hiện ra một sâu mã độc mới có tên TheMoon. Mã độc này tấn công nhằm vào các thiết bị định tuyến Linksys E4200, WRT610N, E1000, E2000, E3000, E1550, E1500. TheMoon khai thác lỗ hổng qua câu lệnh thực thi khi chuyển đổi giá trị tham số "ttcp_ip" được gửi trong một yêu cầu POST. Sau khi khai thác lỗ hổng thành công, mã độc TheMoon sẽ tự tải mã nguồn về thiết bị thông qua câu lệnh wget.

Malware				DDoS	
Name	Year	Source Code	Agents CPU	Architecture Model	Feasible Attacks
Linux.Hydra	2008	Open Source	MIPS	IRC-Based	SYN Flood, UDP Flood
Psyb0t	2009	Reverse Eng.	MIPS	IRC-Based	SYN Flood, UDP Flood, ICMP Flood
Chuck Norris	2010	Reverse Eng.	MIPS	IRC-Based	SYN Flood, UDP Flood, ACK Flood
Tsunami, Kaiten	2010	Reverse Eng.	MIPS	IRC-Based	SYN Flood, UDP Flood, ACK-PUSH Flood, HTTP Layer 7 Flood, TCP XMAS
Aidra, LightAidra, Zendran	2012	Open Source	MIPS, MIPSEL, ARM, PPC, SuperH	IRC-Based	SYN Flood, ACK Flood
Spike, Dofloo, MrBlack, Wrkatk, Sotdas, AES.DdoS	2014	Reverse Eng.	MIPS, ARM	Agent-Handler	SYN Flood, UDP Flood, ICMP Flood, DNS Query Flood, HTTP Layer 7 Flood
BASHLITE, Lizkebab, Torlus, Gafgyt	2014	Open Source	MIPS, MIPSEL, ARM, PPC, SuperH, SPARC	Agent-Handler	SYN Flood, UDP Flood, ACK Flood
Elknot, BillGates Botnet	2015	Reverse Eng.	MIPS, ARM	Agent-Handler	SYN Flood, UDP Flood, ICMP Flood, DNS Query Flood, DNS Amplification, HTTP Layer 7 Flood, Other TCP Floods
XOR.DdoS	2015	Reverse Eng.	MIPS, ARM, PPC, SuperH	Agent-Handler	SYN Flood, ACK Flood, DNS Query Flood, DNS Amplification, Other TCP Floods
LUABOT	2016	Reverse Eng.	ARM	Agent-Handler	HTTP Layer 7 Flood
Remaiten, KTN-RM	2016	Reverse Eng.	ARM, MIPS, PPC, SuperH	IRC-Based	SYN Flood, UDP Flood, ACK Flood, HTTP Layer 7 Flood
NewAidra, Linux.IRCTelnet	2016	Reverse Eng.	MIPS, ARM, PPC	IRC-Based	SYN Flood, ACK Flood, ACK-PUSH Flood, TCP XMAS, Other TCP Floods
Mirai	2016	Open Source	MIPS, MIPSEL, ARM, PPC, SuperH, SPARC	Agent-Handler	SYN Flood, UDP Flood, ACK Flood, VSE Query Flood, DNS Water Torture, GRE IP Flood, GRE ETH Flood, HTTP Layer 7 Flood

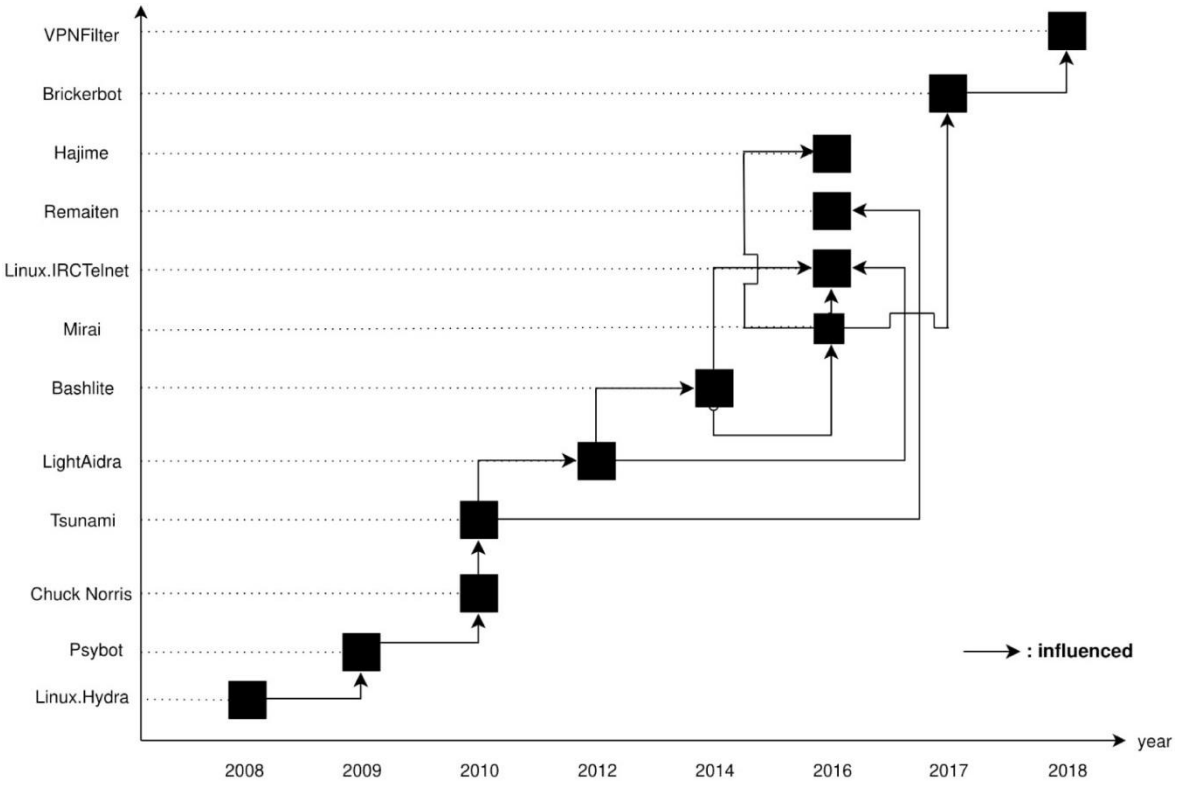
Hình 1.1: Danh sách mã độc IoT có khả năng thực hiện tấn công từ chối dịch vụ

(Nguồn: "Analysis of DDoS-capable IoT malwares")

- Bricker bot: Được phát hiện vào tháng 3/2017, mã độc Bricker bot sử dụng chung quy trình tấn công vét cạn qua cổng Telnet của Mirai [35]. Mục tiêu chính của mã độc này nhắm vào các thiết bị IoT sử dụng Busy box-linux. Sau khi đã thành công xâm nhập vào thiết bị, mã độc Briker bot sẽ thực hiện một chuỗi các câu lệnh Linux dẫn tới làm sập bộ nhớ lưu trữ, gián đoạn các hoạt động mạng và xóa sạch mọi thư mục trên thiết bị.

- VPNFilter: Được phát hiện vào tháng 5/2018, mạng lưới VPNFilter botnet có thể thực hiện chức năng như làm gián đoạn lưu lượng mạng, dò tìm các thiết bị SCADA trên hệ thống mạng và xóa sạch firmware để vô hiệu hóa tạm thời các thiết bị IoT. Các VPNFilter bot được kết nối tới một máy chủ có tên miền là toknowall.com để nhận lệnh điều khiển và các module hỗ trợ [30].

Kể từ khi mã độc Linux.Hydra công bố mã nguồn, đã có nhiều biến thể mã độc IoT được phát triển lên từ nó như Psybot, Chuck Norris và cuối cùng là Tsunami. Kế đến, một phần mã nguồn của Tsunami lại được sử dụng để phát triển các mã độc mới hơn là Remaiten và LightAidra. Từ mã độc Bashlite là thế hệ sau của LightAidra, mã độc Mirai đã kế thừa và phát triển nó trở thành một dòng mã độc mới với nhiều biến thể đa dạng. Hai mã độc gần đây là Bricker bot và VPNFilter cũng kế thừa một số tính chất của mã độc Mirai. Hình 1.2 thể hiện quá trình tiến hóa của các mã độc IoT Botnet [19].



Hình 1.2: Sự tiến hóa của mã độc IoT Botnet

(Nguồn: "A survey of IoT malware and detection methods based on static features")

1.2. Cấu trúc và nguyên lý hoạt động của mã độc IoT Botnet

1.2.1. Cấu trúc của mạng mã độc IoT Botnet

Mã độc IoT Botnet sử dụng cấu trúc mạng botnet tương tự với các mạng botnet thông thường, có thể chia ra thành hai dạng là mạng botnet tập trung và mạng botnet ngang hàng. Trong cả hai trường hợp, đầu tiên, mã độc IoT Botnet sẽ tiến hành tự động rà quét hệ thống mạng và lây nhiễm vào các thiết bị IoT có tồn tại lỗ hổng bảo mật hoặc mật khẩu yếu. Sau khi đã xâm nhập vào thiết bị, nó sẽ tải xuống một đoạn mã nguồn nhị phân từ máy chủ ra lệnh và kiểm soát (C&C) để biến thiết bị IoT thành bot.

Trong cấu trúc mạng botnet tập trung, các máy chủ C&C sẽ có địa chỉ cố định. Giao thức thường được sử dụng trong trường hợp này là IRC và HTTP. Ưu điểm của cấu trúc mạng botnet tập trung là việc dễ dàng triển khai cũng như quản lý mạng botnet của botmaster. Tuy nhiên, cấu trúc này có một nhược điểm là trong trường hợp địa chỉ máy chủ C&C bị phát hiện, dẫn tới việc bị đánh sập hoặc ngăn chặn thiết bị IoT truy cập đến đó thì mạng mã độc IoT Botnet sẽ bị vô hiệu hóa.

Bên cạnh đó, với cấu trúc mạng botnet ngang hàng, giao thức được sử dụng chủ yếu là giao thức P2P. Trong trường hợp này, việc xác định địa chỉ của máy chủ C&C sẽ khó khăn hơn và làm giảm khả năng mạng botnet bị vô hiệu hóa. Dù vậy, cấu trúc này cũng có nhược điểm là botmaster sẽ khó xác định vị trí và điều khiển các bot hơn với cấu trúc mạng botnet tập trung.

Ngoài hai thành phần cơ bản là bot và máy chủ C&C, các mạng mã độc IoT Botnet thường sẽ có thêm bốn thành phần phụ bao gồm:

- Máy Scanner được sử dụng để rà quét các lỗ hổng bảo mật trên thiết bị IoT.

- Máy chủ Report để thu thập các kết quả hoặc báo cáo rà quét từ các bot hoặc máy scanner.

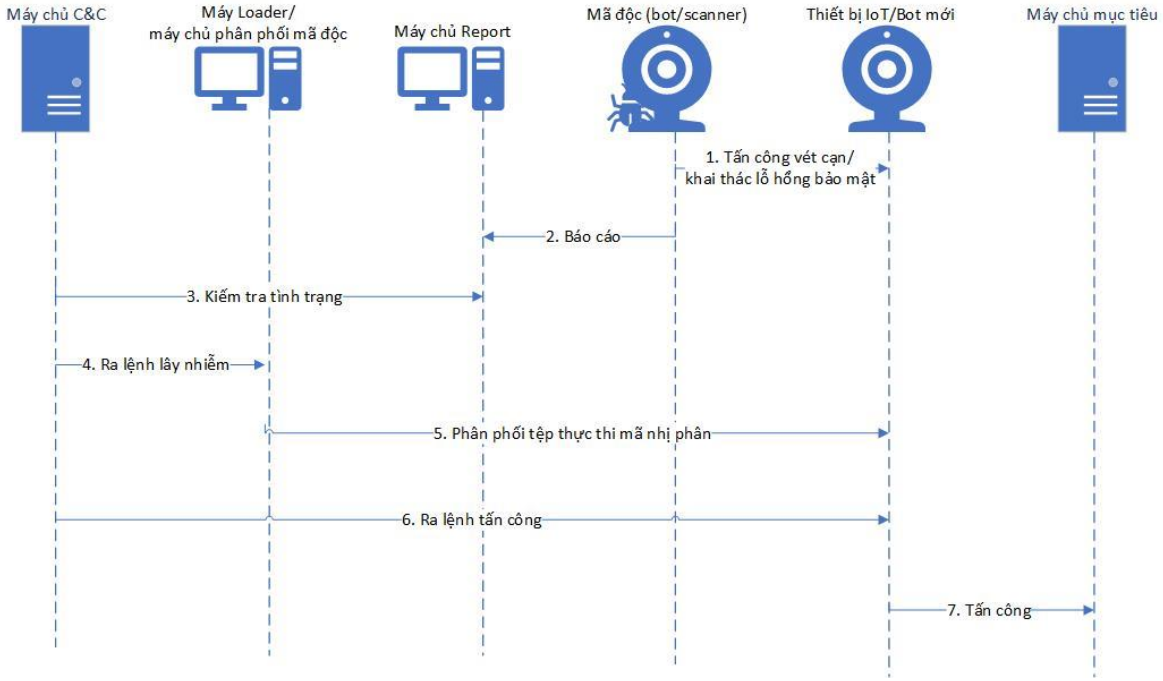
- Máy Loader được sử dụng để truy cập vào thiết bị IoT sau khi đã chiếm quyền điều khiển, chỉ dẫn thiết bị tải và thực thi mã độc.

- Máy chủ phân phối mã độc là nơi lưu trữ các tệp thực thi mã nguồn nhị phân của mã độc tương ứng với nền tảng của từng thiết bị IoT khác nhau như ARM, MIPS, hoặc x86... Các thiết bị IoT bị lây nhiễm sẽ tải mã độc từ đây.

Các chức năng của mỗi thành phần này có thể được tổ hợp và thực hiện bởi một thành phần khác. Ví dụ, với mã độc Mirai, các bot sẽ thực hiện luôn việc rà quét lỗ hổng bảo mật và tấn công từ chối dịch vụ đối với mục tiêu.

1.2.2. Nguyên lý hoạt động của mã độc IoT Botnet

Mục tiêu chủ yếu của các mã độc IoT Botnet là thực hiện tấn công từ chối dịch vụ. Trước khi bắt đầu, mã độc (bot, máy scanner hoặc máy chủ C&C) sẽ tiến hành rà quét các địa chỉ IP công khai thông qua một số cổng dịch vụ mở sẵn như các cổng Telnet, SSH.... Hình 1.3 mô tả nguyên lý hoạt động chung của mã độc IoT Botnet.



Hình 1.3: Hoạt động của mã độc IoT Botnet

Bước 1: Mã độc (bot hoặc máy scanner) thực hiện tấn công vét cạn để khám phá tên đăng nhập và mật khẩu được thiết lập thiếu an toàn của thiết bị IoT hoặc thực hiện tấn công khai thác các lỗ hổng bảo mật của thiết bị để chiếm quyền điều khiển thiết bị.

Bước 2: Sau khi đã chiếm được quyền điều khiển thiết bị, mã độc (bot hoặc máy scanner) sẽ gửi các thông tin của thiết bị IoT mới về máy chủ Report thông qua một cổng khác với cổng nó đã sử dụng để tấn công vét cạn hoặc khai thác lỗ hổng bảo mật.

Bước 3: Thông quá máy chủ C&C, botmaster sẽ thường xuyên kiểm tra tình trạng của các thiết bị IoT mục tiêu và tình trạng của mã độc thông qua việc giao tiếp với máy chủ Report thông qua một kênh truyền bí mật (ví dụ như Tor...).

Bước 4: Sau khi quyết định thiết bị IoT nào sẽ bị lây nhiễm, botmaster sẽ gửi đi một lệnh lây nhiễm tới máy Loader kèm theo các thông tin cần thiết về thiết bị IoT sẽ bị lây nhiễm như địa chỉ IP và kiến trúc phần cứng.

Bước 5: Máy loader sẽ truy cập vào thiết bị IoT mục tiêu, kế đến chỉ dẫn thiết bị tải mã nhị phân tương ứng của mã độc từ máy chủ phân phối mã độc (hoặc từ chính máy loader) và thực thi nó. Việc tải có thể thông qua các công cụ như GNU Wget hoặc Giao thức truyền tập tin đơn giản (TFTP)... Thông thường, sau khi mã độc được thực thi, nó sẽ ngay lập tức cố gắng tự bảo vệ khỏi các mã độc khác bằng cách đóng lại các lỗ hổng xâm nhập khác như dịch vụ Telnet hoặc SSH trên thiết bị. Trường hợp mã độc khác được phát hiện trong thiết bị IoT, mã độc mới sẽ tìm cách tiêu diệt đối thủ thông qua các cách thức khác nhau như xóa khỏi bộ nhớ hoặc ghi đè lên. Kế đến, mã độc cấu hình lại thiết bị để trở thành một phần của mạng botnet. Lúc này, thiết bị bot mới có thể giao tiếp với máy chủ C&C để nhận lệnh tấn công. Việc giao tiếp được thực hiện bằng cách phân giải địa chỉ tên miền được gắn kèm theo mã thực thi thay vì một địa chỉ IP cố định (ví dụ trong mã nguồn của Mirai thì giá trị của mục này là cnc.changeme.com...). Nhờ vào đó, botmaster sẽ thuận tiện hơn trong việc thay đổi địa chỉ IP của máy chủ C&C theo thời gian mà không cần phải thay đổi mã nhị phân hoặc thực hiện những giao tiếp không cần thiết.

Bước 6: Botmaster chỉ thị tất cả các bot thực hiện cuộc tấn công tới máy chủ mục tiêu thông qua việc gửi đi một câu lệnh đơn giản từ máy chủ C&C với các thông số cần thiết như kiểu tấn công, thời gian tấn công, địa chỉ IP của các thiết bị bot và máy chủ mục tiêu...

Bước 7: Các thiết bị bot bắt đầu tấn công máy chủ mục tiêu với một trong số những kiểu tấn công đã nêu trong Hình 1.1 như tấn công Generic Routing Encapsulation (GRE), TCP và HTTP flooding...

Bên cạnh nguyên lý hoạt động chung của IoT Botnet đã được mô tả trong Hình 1.3, các mã độc IoT Botnet cũng luôn có những đặc điểm riêng biệt trong nguyên lý hoạt động của chúng. Ví dụ trường hợp của mã độc Mirai, xuyên suốt quá trình lây nhiễm, nó luôn để lại những dấu vết có thể phát hiện được thông qua một phân tích mạng đơn giản [5]. Hoặc trường hợp của mã độc LuaBot [33] thì được tích hợp nhiều chức năng phức tạp như kênh giao tiếp C&C mã hóa và các tập luật iptables tùy biến để bảo vệ những thiết bị mà nó đã lây nhiễm. Mã độc Hajime [36] thì dựa trên giao tiếp phân tán hoàn toàn và tận dụng giao thức bảng băm phân tán (DHT) BitTorrent để phát hiện thiết bị ngang hàng và giao thức truyền tải uTorrent để trao đổi dữ liệu. Mỗi thông điệp của mã độc Hajime đều được mã hóa RC4 và tạo chữ ký bằng một cặp khóa công khai và bí mật. Trường hợp của mã độc Bricker bot [35], bằng cách tận dụng thông tin đăng nhập mặc định của dịch vụ SSH, lỗi cấu hình hoặc các lỗ hổng bảo mật đã biết, mã độc này sẽ thực hiện các cuộc tấn công từ chối dịch vụ lâu dài (PDoS) thông qua các cách thức bao gồm xóa firmware thiết bị, xóa các thư mục trong bộ nhớ thiết bị hoặc tái cấu hình các tham số mạng.

Ngoài ra, cuộc chiến tranh giành quyền kiểm soát thiết bị IoT cũng là một đặc điểm thường thấy ở các mã độc IoT Botnet. Có thể lấy ví dụ như các mã độc Carna, Darlloz và Wifatch sẽ xóa được mã độc LightAidra, hoặc họ mã độc Mirai có thể xóa được họ mã độc Bashlite...[13] Những đặc điểm này thể hiện rõ tính nguy hiểm của mã độc, tối ưu hóa khả năng tấn công và khả năng chống xóa bỏ của chúng đối với các mã độc khác.

1.3. Các phương pháp phát hiện mã độc IoT Botnet

Nghiên cứu phát hiện mã độc IoT Botnet là một vấn đề khá thách thức, khi mà mã độc hiện nay đang ngày càng phát triển để che dấu các hoạt động khả nghi trong hệ thống và vượt mặt các chương trình dò quét. Để giảm thiểu các thiệt hại do chúng gây ra, một số nghiên cứu về phát hiện mã độc IoT dựa trên học máy đã được triển

khai. Tổng quan lại, có thể chia ra 3 hướng tiếp cận chính trong phân tích phát hiện mã độc, bao gồm: phát hiện dựa trên phân tích tĩnh; phát hiện dựa trên phân tích động; và phát hiện dựa trên phương pháp lai.

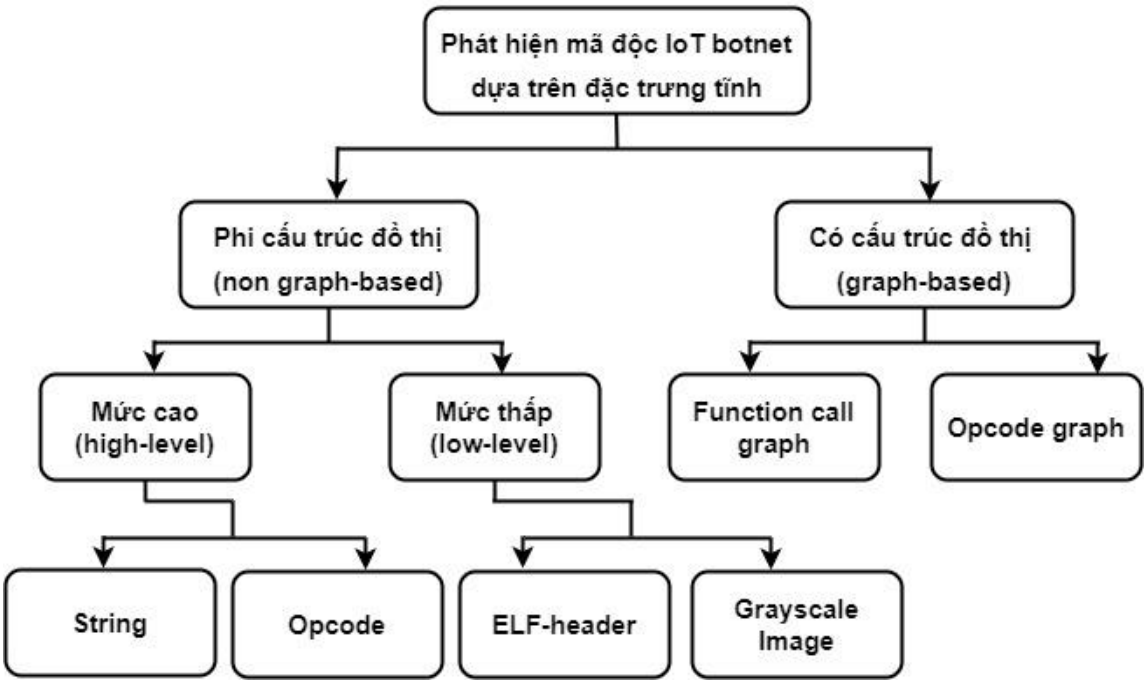
1.3.1. Phát hiện mã độc IoT Botnet dựa trên phân tích tĩnh

Phương pháp phân tích tĩnh là phương pháp phân tích, phát hiện mã độc, lỗ hổng bảo mật dựa trên những đặc trưng của các tập tin chương trình mà không cần thực thi chúng (trên thiết bị thực hoặc môi trường mô phỏng). Việc phân tích như vậy có thể thực hiện trên mã nguồn tường minh hoặc các tập tin nhị phân thực thi.

Trong hướng tiếp cận dựa trên phân tích tĩnh, các đặc trưng phổ biến của mã độc như: tiêu đề tập tin (header), các lời gọi hàm hệ thống, thông tin chuỗi in (PSI), FLF (Function Length Frequency), các thư viện liên kết, OpCode (trích xuất từ mã assembly)… sẽ được trích xuất từ các tập tin thực thi để đưa vào các thuật toán học máy phân loại [1], [10], [12], [18], [21]. Dịch ngược là hướng tiếp cận phổ biến để trích xuất những thông tin đặc trưng trên từ một tập tin thực thi. Cách thức trích xuất và xử lý những đặc trưng đó ảnh hưởng lớn đến độ chính xác và phức tạp của các phương pháp phát hiện mã độc IoT, những đặc trưng đó có thể được chia thành 02 nhóm: dựa trên các đặc trưng có cấu trúc đồ thị và dựa trên các đặc trưng không có cấu trúc đồ thị, như minh họa ở Hình 1.4.

Các phương pháp phát hiện mã độc sử dụng các đặc trưng không có cấu trúc đồ thị nhằm xây dựng các mô hình phát hiện chứa các thuộc tính của cấu trúc tập tin nhị phân để phân loại một tập tin nhị phân là mã độc hay lành tính. Những phương pháp này dựa trên trích xuất các đặc trưng gồm Opcode, Strings hoặc cấu trúc tập tin phân biệt các mẫu mã độc. Những đặc trưng này có thể được chia thành 2 nhóm: đặc trưng mức cao và đặc trưng mức thấp. Cụ thể, các đặc trưng mức thấp có thể được thu thập trực tiếp từ trong cấu trúc của tập tin, trong khi đó các đặc trưng mức cao cần sử dụng các công cụ hỗ trợ phân tách (disassembler) như IDA Pro hoặc Radare2. Những nghiên cứu biểu diễn các tập tin thực thi bằng các đặc trưng không có cấu trúc đồ thị sẽ phụ thuộc nhiều vào giá trị của các đặc trưng (ví dụ lời gọi hàm inet_toa) và sẽ không thể mô tả thông tin ngữ nghĩa phức tạp giữa các đặc trưng (ví dụ như dữ liệu

phụ thuộc trong vòng đời mã độc IoT botnet). Bên cạnh đó các nghiên cứu sử dụng đặc trưng không có cấu trúc đồ thị thường khá yếu với mã độc sử dụng kỹ thuật gây rối như mã hóa, chèn dữ liệu rác...



Hình 1.4: Phân loại các đặc trưng tĩnh trong phát hiện mã độc IoT Botnet

(Nguồn: "A survey of IoT malware and detection methods based on static features")

Một vài nghiên cứu theo hướng phát hiện mã độc IoT dựa trên phân tích tĩnh gần đây có thể kể đến như:

- Nghiên cứu của HaddadPajouh và các cộng sự [10] đề xuất một hướng tiếp cận sử dụng học sâu với mạng nơ-ron hồi quy (RNN) và các đặc trưng tĩnh trích xuất từ mã OpCode thực thi của ứng dụng IoT. Mô hình đề xuất được huấn luyện với tập dữ liệu gồm 281 mẫu mã độc và 270 mẫu lành tính, sau đó được kiểm tra với 100 mẫu mã độc IoT mới thông qua ba cấu hình mạng bộ nhớ dài - ngắn (LSTM) khác nhau. Độ chính xác cao nhất của mô hình đề xuất là 98,18%.

- Nghiên cứu của Su và các cộng sự [12] đưa ra một mô hình phát hiện mã độc IoT Botnet thời gian thực thông qua việc chuyển đổi mã nhị phân của mã độc sang dạng ảnh đa mức xám, sau đó sử dụng mạng nơ-ron tích chập (CNN) trọng số nhẹ hai lớp để tự động trích xuất các đặc trưng tĩnh các bức ảnh mã độc đó và đưa vào

huấn luyện. Mô hình được thực nghiệm với tập dữ liệu bao gồm 500 mẫu mã độc và 365 mẫu lành tính, đạt được độ chính xác cao nhất là 94,0%.

- Nghiên cứu của Ngô Quốc Dũng và các cộng sự [18] đề xuất hướng tiếp cận sử dụng các đặc trưng tĩnh từ đồ thị thông tin chuỗi in (PSI) cho phát hiện mã độc IoT Botnet. Các đặc trưng từ đồ thị thông tin chuỗi in được xây dựng dựa trên đồ thị hàm gọi (Function Call Graph) rút gọn, tượng trưng cho chu trình hoạt động của mã độc IoT Botnet. Mô hình được thử nghiệm trên tập dữ liệu gồm 12000 mẫu với 7000 mẫu mã độc IoT Botnet và đạt được độ chính xác vào khoảng 98,7%.

1.3.2. Phát hiện mã độc IoT Botnet dựa trên phân tích động

Phân tích động là phương pháp phân tích cách hoạt động của mã độc khi mã độc được thực thi. Bằng cách giám sát các hoạt động của mã độc, cách thức thực thi lây lan như thế nào, nó kết nối đến đâu, cài đặt những gì vào hệ thống, thay đổi thành phần nào nhằm mục đích ngăn chặn việc lây nhiễm, tạo ra các dấu hiệu nhận dạng hiệu quả. Việc phân tích mã độc đòi hỏi quá trình theo dõi liên tục và lặp đi lặp lại. Nếu thực hiện trên hệ thống thiết bị thật sẽ mất rất nhiều công sức và thời gian. Nó cũng có thể gây thiệt hại lớn cho cả hệ thống. Vì thế, cần phải thiết lập một môi trường an toàn cho việc chạy mã độc để có thể thu thập thông tin về mã độc một cách tốt nhất. 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 được định trước hay không.

Trong hướng tiếp cận phát hiện mã độc IoT Botnet dựa trên phân tích động, tập tin thực thi mã độc sẽ được cho chạy trực tiếp trong một môi trường thời gian thực. Các công cụ giám sát sẽ được sử dụng để trích xuất các hành vi đặc trưng như các lời gọi API (Application Programming Interfaces), các hành vi mạng, các thay đổi trên registry hoặc việc sử dụng bộ nhớ… và sử dụng để phân loại mã độc [11], [20], [27]. Các môi trường thời gian thực có giám sát thường được sử dụng trong phân tích động như các công cụ xây dựng môi trường máy ảo (virtual machine) hoặc sandbox.

Quy trình phân tích động thông thường sẽ gồm các bước sau:

- Bước 1: Thực thi tập tin mã độc trong môi trường mô phỏng.

Các tập tin thực thi mã độc trên nền tảng thiết bị IoT thông thường đều ở định dạng ELF (Executable and Linkable Format) sẽ được đưa vào môi trường mô phỏng thiết bị IoT để tiến hành thực thi thông qua nhiều kênh khác nhau như SSH, Telnet, sao chép vào bản ảnh (image firmware),… Sau đó, các tập tin ELF này được kích hoạt theo nhiều cách khác nhau, dựa trên kinh nghiệm của người phân tích hoặc kịch bản có sẵn.

- Bước 2: Giám sát các hành vi của tập tin mã độc.

Thực hiện giám sát các hành vi của tập tin ELF khi thực thi trong môi trường mô phỏng sau khi được kích hoạt chạy. Các hành vi được giám sát cơ bản bao gồm: hành vi liên quan tới giao thức mạng (mở cổng, tạo kết nối, truyền và nhận tệp tin, chuyển hướng kết nối), hành vi liên quan tới tiến trình (tạo tiến trình con, ẩn tiến trình, tắt tiến trình khác), hành vi gọi thực thi các chương trình khác. Để giám sát các hành vi này thường sử dụng các công cụ được cung cấp, tích hợp sẵn trong các môi trường mô phỏng hoặc tự phát triển các công cụ phù hợp với môi trường này. Các công cụ thường được sử dụng như: WireShark, TShark, Strace, TCPdump, ProcessMon, Process Explorer, ProcDOT, Noriben...

- Bước 3: Phát hiện các tác động tiêu cực của tập tin mã độc với môi trường mô phỏng.

Bên cạnh việc giám sát toàn bộ các hành vi của tập tin ELF, việc giám sát các thay đổi của môi trường mô phỏng cũng giúp ích cho việc phát hiện các mã độc. Đôi khi các biến thể mã độc che giấu hành vi đủ tốt để không bị các công cụ giám sát hành vi ghi nhận. Tuy nhiên, mục đích cuối cùng của mã độc là ảnh hưởng tiêu cực tới hệ thống nên việc phát hiện các tác động tiêu cực là một phần quan trọng để phân biệt tệp ELF là mã độc hay mã sạch. Các thay đổi này thông thường sẽ nằm ở các nguồn tài nguyên hệ thống (băng thông mạng, bộ nhớ, năng lực xử lý tính toán), các tập tin và thư mục (tạo tập tin mới, tạo thư mục mới để tải tệp tin lây nhiễm), thay đổi cấu hình mặc định của hệ thống, tạo lỗ hổng bảo mật, cổng hậu (backdoor). Các công cụ thường được sử dụng như: RegShot, Autoruns, Fiddler.

- Bước 4: Lựa chọn các đặc trưng hành vi độc hại của tập tin mã độc.

Sau quá trình giám sát toàn bộ các hành vi và tác hại của tập tin ELF đối với môi trường mô phỏng, người phân tích phải xác định các hành vi đặc trưng của tập tin ELF để đưa ra quyết định đó là mã độc hay lành tính. Các đặc trưng thông thường sẽ được lựa chọn dựa trên đặc điểm của các họ mã độc đã biết hoặc kinh nghiệm của người phân tích. Ví dụ: đặc trưng về lưu lượng mạng, tạo cổng kết nối mới, tạo tiến trình con, thay đổi cấu hình hệ thống, tạo cổng hậu.

- Bước 5: Lưu trữ các đặc trưng hành vi độc hại phục vụ phát hiện mã độc.

Cuối cùng, sau khi đã lựa chọn các đặc trưng cụ thể về hành vi độc hại của mã độc, người phân tích sẽ tiến hành xác định các giá trị của đặc trưng thông qua thực nghiệm để sinh ra mẫu chữ ký động hoặc báo cáo về đặc trưng hành vi, qua đó sử dụng để phát hiện mã độc.

Một vài nghiên cứu theo hướng phát hiện mã độc IoT dựa trên phân tích động gần đây có thể kể đến như:

- Nghiên cứu của Jeon và các cộng sự [11] đề xuất mô hình sử dụng mạng nơ-ron tích chập trong môi trường ảo hóa lồng nhau dựa trên đám mây để phát hiện mã độc IoT. Các đặc trưng động theo hành vi của tập tin thực thi mã độc thu được trong môi trường ảo sẽ được chuyển đổi thành dạng ảnh màu RGB, từ đó thực hiện trích chọn đặc trưng và phân loại mã độc. Mô hình được thực nghiệm với tập dữ liệu 1402 mẫu bao gồm cả mã độc IoT và lành tính. Độ chính xác của mô hình phát hiện đạt được là 99,28%.

1.3.3. Phát hiện mã độc IoT Botnet dựa trên phương pháp lai

Trong mỗi phương pháp phân tích tĩnh và phân tích động đều có những ưu điểm và hạn chế nhất định. Bảng 1.2 tổng hợp các ưu nhược điểm của từng phương pháp đã nêu.

Ta có thể thấy, hướng phát hiện dựa trên phân tích tĩnh sẽ có lợi thế hơn trong việc hiểu rõ cấu trúc của mã độc. Trong khi đó, hướng phân tích động lại có thể giải quyết các kỹ thuật làm rối mã độc. Từ đó, sản sinh ra một hướng tiếp cận nữa, với mục tiêu vận dụng được ưu điểm của cả hai phương pháp trên, đó chính là phương pháp lai [7], [20], [21], [27]. Đây cũng là hướng nghiên cứu chính của luận văn.

Bảng 1.2: So sánh các phương pháp phân tích tĩnh và động trong phát hiện mã độc IoT Botnet

	Phân tích động	Phân tích tĩnh
Ưu điểm	- Quan sát thực thi cụ thể của một chương trình để quyết định tập tin là mã độc dễ dàng - Phân tích động hiệu quả hơn đối với mã độc gây rối	- 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 - Không cần phải thực thi mã độc nên không bị ảnh hưởng bởi tính đa kiến trúc khi xây dựng môi trường thực thi
Hạn chế	- Chỉ có thể giám sát đơn luồng thực thi - Làm lộ quá trình phát hiện và phân tích mã độc - Có thể gây nguy cơ mất an toàn cho mạng và hệ thống - Khó mô phỏng đầy đủ thiết bị IoT (tính đa kiến trúc)	- Phân tích tĩnh phụ thuộc nhiều vào công nghệ dịch ngược - Khó khăn khi xử lý mã độc sử dụng kỹ thuật gây rối

Có một số cách thức thực hiện phương pháp lai khác nhau như thực hiện phân tích tĩnh dựa trên kết quả phân tích động, thực hiện phân tích động dựa trên kết quả phân tích tĩnh, hoặc thực hiện song song cả hai quy trình phân tích tĩnh và động. Để đảm tính khách quan trong hai quy trình phân tích tĩnh và động, luận văn sẽ lựa chọn cách thực hiện cả hai quy trình phân tích tĩnh và động song song, sau đó dựa trên kết quả của hai quy trình là tập các đặc trưng tĩnh và động để tích hợp chúng lại và sử dụng thuật toán học máy để huấn luyện và phân loại mã độc.

Một vài nghiên cứu theo hướng phát hiện mã độc IoT dựa trên phương pháp lai thông qua việc tích hợp các đặc trưng động và đặc trưng tĩnh có thể kể đến như:

- Nghiên cứu của Shijo và các cộng sự [20] tích hợp các đặc trưng tĩnh lấy từ PSI và các đặc trưng động lấy từ chuỗi lời gọi API và chuyển chúng sang dạng vector để huấn luyện và phân loại mã độc. Mô hình phân loại sử dụng tập dữ liệu gồm 997

mẫu mã độc và 490 mẫu lành tính. Độ chính xác cao nhất trong quá trình thực nghiệm đạt được là 98,71%.

- Nghiên cứu của Chen và các cộng sự [27] sử dụng mạng học sâu để trích xuất đặc trưng tĩnh từ đồ thị hàm gọi FCG và đặc trưng động trong các lời gọi API để phát hiện mã độc. Quá trình thực nghiệm sử dụng tập dữ liệu gồm 80 họ mã độc và 4519 mẫu mã độc. Độ chính xác đạt được của nghiên cứu trong phát hiện mã độc này là 83,17%.

- Nghiên cứu của Islam và các cộng sự [21] tích hợp các vector đặc trưng tĩnh trích xuất từ FLF (Function length frequency) và PSI cũng các vector đặc trưng động trích xuất từ các tham số lời gọi API thành một vector duy nhất tương ứng với mỗi 2939 tệp thực thi để phân loại mã độc. Kết quả thực nghiệm cho ra độ chính xác cao nhất là 97,05%.

- Nghiên cứu của Gandora và các cộng sự [7] để xuất một framework tích hợp các đặc trưng tĩnh và động phục vụ cho việc phân loại mã độc. Trong mô hình, các đặc trưng tĩnh được trích xuất từ số phân vùng khả nghi trong tệp thực thi và tần suất hàm gọi, còn các đặc trưng động được trích xuất từ các hoạt động mạng và hoạt động của tệp thực thi. Tập dữ liệu được sử dụng để thực nghiệm bao gồm 998 mẫu mã độc và 428 mẫu lành tính, thu được độ chính xác cao nhất là 99,58% với thuật toán phân loại Random Forest.

Kết luận chương 1

Nội dung chương 1 đã trình bày một cách khái quát về mã độc IoT nói chung và IoT Botnet nói riêng. Cùng với đó là tổng quan về các hướng nghiên cứu đã được vận dụng để phân tích, phát hiện mã độc IoT Botnet.

Trong chương 1, luận văn đã đưa khái niệm, phân loại của mã độc IoT nói chung và cụ thể về IoT Botnet nói riêng. Từ đó, làm rõ cấu trúc và các nguyên lý hoạt động, lây lan của mã độc IoT Botnet. Bên cạnh đó, luận văn cũng tìm hiểu về một số hướng nghiên cứu và các phương pháp đã được vận dụng để phân tích, phát hiện mã độc IoT Botnet cùng các ưu điểm, nhược điểm của từng phương pháp phân tích động, phân tích và phân tích lai.

Kết quả nghiên cứu của chương 1 sẽ là cơ sở để luận văn lựa chọn xây dựng thử nghiệm phương pháp lai trong phát hiện mã độc IoT Botnet ở chương 2.

Chương 2: PHƯƠNG PHÁP LAI TRONG PHÁT HIỆN MÃ ĐỘC IOT BOTNET

2.1. Xây dựng các đặc trưng tĩnh

2.1.1. Một số đặc trưng tĩnh trong phát hiện mã độc IoT Botnet

Trong phân tích tĩnh, tùy thuộc vào việc trích chọn và xử lý các đặc trưng sẽ ảnh hưởng đến độ chính xác và độ phức tạp của phương pháp phát hiện mã độc IoT Botnet. Dựa trên các nghiên cứu về phân tích tĩnh trong phát hiện mã độc IoT, ta có thể liệt kê một số đặc trưng tĩnh [19] có thể được sử dụng và mức độ ảnh hưởng của chúng trong phát hiện mã độc IoT Botnet, bao gồm:

- Mã thực thi (Opcode): là một trong những đặc trưng thường được sử dụng nhất trong phát hiện mã độc. Một Opcode là một chỉ lệnh đơn có thể được thực thi bởi bộ xử lý (CPU) mô tả các hành vi của một tệp thực thi. Trong hợp ngữ assembly, Opcode thường là các lệnh đơn giản như CALL, ADD, MOV... Đây là đặc trưng có ảnh hưởng khá lớn, thường phải trải qua các bước tiền xử lý như tạo thành các chuỗi Opcode tuần tự, chuyển hóa các Opcode của tệp thực thi thành dạng vector, hay so sánh tần suất lặp của một Opcode trong tệp thực thi,... để sử dụng trong phát hiện mã độc.

- Các chuỗi (String): Một chuỗi trong tệp thực thi là một dãy các ký tự như “gayfgt” thường được lưu trữ dưới định dạng mã ASCII (1 byte/ký tự) hoặc mã Unicode (2 byte/ký tự). Mỗi một chuỗi in (printable string) trong một tệp thực thi đều có thể được trích xuất ra các thông tin hữu dụng như địa chỉ IP, địa chỉ URL kết nối,... từ đó xác định tệp thực thi có phải mã độc hay không. Thông tin chuỗi in PSI là đặc trưng mức cao và ảnh hưởng lớn, cần phải thông qua các bước tiền xử lý như xây dựng đồ thị, tạo mẫu chữ ký,... để sử dụng trong phát hiện mã độc.

- Tiêu đề tập tin ELF (Executable and Linkable format): Định dạng tập tin ELF chứa trong đó rất nhiều thông tin hữu ích có thể sử dụng trong phát hiện mã độc, có thể kể đến như:

+ Entropy được coi là thang đo hiệu quả khả năng lưu trữ thông tin của một tập tin. Một tập tin nén có entropy càng cao thì hiệu quả lưu trữ thông tin của tập tin đó càng lớn. Đây là một trong những thông tin khá hữu ích để kiểm tra hành vi độc hại

của tệp tin, khi mà entropy của tệp tin càng lớn thì khả năng có chứa mã độc của nó càng cao. Tuy nhiên, mã độc có thể tránh bị phát hiện bằng cách chèn thêm dữ liệu không byte.

+ Thời điểm tập tin được tạo hoặc thời điểm biên dịch cũng là một thông tin có thể hỗ trợ xác định tập tin là mã độc hay lành tính. Các tập tin chứa mã độc thường có thời điểm khá chung chung như trước năm 2007 hoặc một năm lạ nào đó trong khi thời điểm của tập tin lành tính là khá rõ ràng. Mã độc IoT Botnet bắt đầu lây nhiễm trên thiết bị IoT vào năm 2008. Dù vậy, đặc điểm này chỉ có thể hỗ trợ chứ không góp phần xác định chính xác tập tin có phải mã độc hay không.

+ Các phân vùng có tên khả nghi cũng có thể là một thông tin để phục vụ cho việc xác định mã độc. Những trình biên dịch thông thường sẽ đặt tên thông dụng cho từng phân vùng như .data, .text,... Tuy nhiên, với các tập tin chứa mã độc thì tên một số phân vùng thường sẽ được đặt ngẫu nhiên. Do đó, đặc điểm này có thể xác định tập tin được liệt kê vào danh sách khả nghi hay không.

+ Thông tin nén cũng được coi là một thông tin hỗ trợ xác định tập tin mã độc hay lành tính. Các tập tin mã độc thường được nén với mục đích vượt qua các chương trình phát hiện mã độc thông qua mẫu chữ ký hoặc tránh bị dịch ngược.

+ Dung lượng tập tin cũng có thể được sử dụng như một đặc trưng, dựa trên sự khác nhau về dung lượng có thể của tập tin mã độc và lành tính. Trong khi mã độc được lập trình sao cho dung lượng càng nhỏ càng tốt để có thể truyền đi nhanh và hoạt động trong bộ nhớ giới hạn của thiết bị IoT, thì các tập tin lành tính thường có dung lượng cố định và phải đủ để đảm bảo hoạt động của thiết bị.

+ Các thông tin về tập tin như phiên bản thiết bị, tên thiết bị, tên nhà sản xuất cũng có thể sử dụng như một đặc điểm để phát hiện mã độc. Nguyên nhân bởi các tập tin lành tính sẽ thường cố gắng thêm đầy đủ những thông tin trên, còn tập tin mã độc thì sẽ tránh việc thêm vào những thông tin chiếm dung lượng này.

Tuy có nhiều đặc điểm có thể sử dụng, nhưng những thông tin lấy từ tiêu đề tập tin ELF được coi là đặc trưng mức thấp, và việc sử dụng những đặc trưng này để phân loại mã độc có thể thiếu chính xác nếu không được xử lý cẩn thận.

- Ảnh đa mức xám: Là dạng ảnh mà mỗi điểm ảnh có giá trị từ 0 đến 255. Trong bài toán phát hiện mã độc, các tập tin thực thi có thể được phân tích và chuyển đổi sang dạng chuỗi nhị phân 0 và 1, sau đó kết hợp các giá trị nhị phân đó thành cá vector 8-bit thể biểu diễn giá trị thập lục phân từ 00 đến FF. Các vector này cuối cùng sẽ được chuyển đổi thành dữ liệu ảnh với các điểm ảnh có giá trị từ 0 đến 255, với 0 là điểm ảnh đen và 255 là điểm ảnh trắng. Các bức ảnh chuyển đổi từ tập tin thực thi này có thể được đưa vào mạng học sâu để xử lý, trích chọn đặc trưng và phân loại. Tuy được phân loại vào đặc trưng mức thấp, nhưng loại đặc trưng này lại khá hiệu quả trong phát hiện mã độc.

- Đồ thị hàm gọi (FCG) hoặc đồ thị luồng điều khiển (CFG) cũng là một trong những đặc trưng khá phổ biến trong phân tích tĩnh. Đây là dạng đồ thị có hướng, biểu diễn tất cả các hướng thực thi có thể có trong chương trình, với mỗi đỉnh (nút) được thể hiện dưới dạng một khối cơ bản và mỗi cạnh có hướng thể hiện luồng điều khiển khả thi giữa các khối cơ bản. Thông tin luồng điều khiển có hai dạng chính là luồng điều khiển ngoại liên tiến trình và luồng điều khiển nội liên tiến trình. Đồ thị luồng điều khiển ngoại liên tiến trình biểu diễn các mối liên kết giữa các hàm và các tiến trình trong tập tin thực thi trong một đồ thị luồng điều khiển duy nhất. Mặt khác, đồ thị luồng điều khiển nội liên tiến trình sẽ được biểu diễn dưới dạng một tập hợp các đồ thị luồng điều khiển, với mỗi đồ thị tương ứng với một tiến trình của một hàm. Các đặc trưng được trích xuất từ đồ thị luồng điều khiển thường cho ra độ chính xác khá cao trong phân loại mã độc.

- Đồ thị thông tin chuỗi in (PSI) [18] là một hướng tiếp cận mới trong việc trích xuất các đặc trưng từ đồ thị. Đồ thị thể hiện được chu trình hoạt động tiêu biểu của các mã độc IoT Botnet. Các đặc trưng rút ra từ đồ thị có thể sử dụng cho quá trình phân loại mã độc và cho ra độ chính xác khá cao cùng thời gian phân loại nhanh.

2.1.2. Đặc trưng tĩnh được chọn cho phương pháp lai

Dựa trên nguyên lý hoạt động của mã độc IoT Botnet, ta có thể thấy, cho dù các mã độc IoT Botnet ngày càng tiến hóa về cả số lượng biến thể lẫn chủng loại thì hoạt

động của chúng vẫn tuân theo một quy luật nhất định. Ta có thể tóm tắt các quy luật hoạt động đó thành một chu trình gồm 5 giai đoạn như sau:

- Giai đoạn thứ 1: Các mã độc IoT Botnet rà quét các thiết bị IoT khác có chứa lỗ hổng bảo mật và có khả năng bị lấy nhiễm. Công việc này có thể được thực hiện một cách ngẫu nhiên bằng cách rà quét các địa chỉ IP với các cổng dịch vụ quen thuộc như Telnet (cổng 23 hoặc 2323) và SSH (cổng 22).

- Giai đoạn thứ 2: Chiếm quyền truy nhập vào các thiết bị thông qua lỗ hổng bảo mật. Bot sẽ tiến hành tấn công vét cạn hoặc tấn công khai thác để tìm ra tài khoản và mật khẩu đăng nhập mặc định yếu thường được thiết lập trên thiết bị IoT như root/root, admin/123,...

- Giai đoạn thứ 3: Lây nhiễm thiết bị IoT. Sau khi đã truy cập vào thiết bị, bot sẽ lây nhiễm thông qua một trình loader để tải và thực thi mã nhị phân tương ứng với phiên bản botnet. Việc tải thường thông qua các giao thức như FTP, HTTP hay TFTP.

- Giai đoạn thứ 4: Giao tiếp với máy chủ C&C thông qua địa chỉ IP hoặc URL. Mã độc IoT Botnet sẽ cố gắng kết nối và truyền tải các thông tin về thiết bị như địa chỉ IP và kiến trúc phần cứng tới máy chủ Report thông qua một cổng khác.

- Giai đoạn thứ 5: Chờ lệnh từ máy chủ C&C. Bot vừa lặp lại chu trình lây nhiễm như trên, vừa ẩn mình trong thiết bị và đợi lệnh tấn công từ máy chủ C&C.

So sánh với các mã độc botnet khác [5], [24], hầu như tất cả các giai đoạn trong chu trình hoạt động của mã độc IoT Botnet đều để lại dấu vết có thể bị phát hiện. Những dấu hiệu này bao gồm các địa chỉ IP, các mẫu tên đăng nhập/mật khẩu, các lệnh tấn công,... Đây có thể coi là một đặc trưng của riêng mã độc IoT Botnet và là một hướng khá hiệu quả trong phát hiện mã độc IoT Botnet.

Vì lý do đó, luận văn sẽ lựa chọn việc trích xuất các đặc trưng từ đồ thị thông tin chuỗi in PSI để làm các đặc trưng tĩnh phục vụ cho phương pháp lai. Đây là đồ thị được xây dựng thể hiện cho một chu trình hoạt động tiêu biểu của mã độc IoT Botnet bao gồm 5 giai đoạn nêu trên. Mỗi thông tin chuỗi in sẽ là một hoạt động ở từng giai đoạn của bot. Các đặc trưng trích xuất từ đồ thị có thể được áp dụng cho các thuật

toán học máy để tiến hành phân loại mã độc và lành tính. Định nghĩa của đồ thị được trình bày như sau [18]:

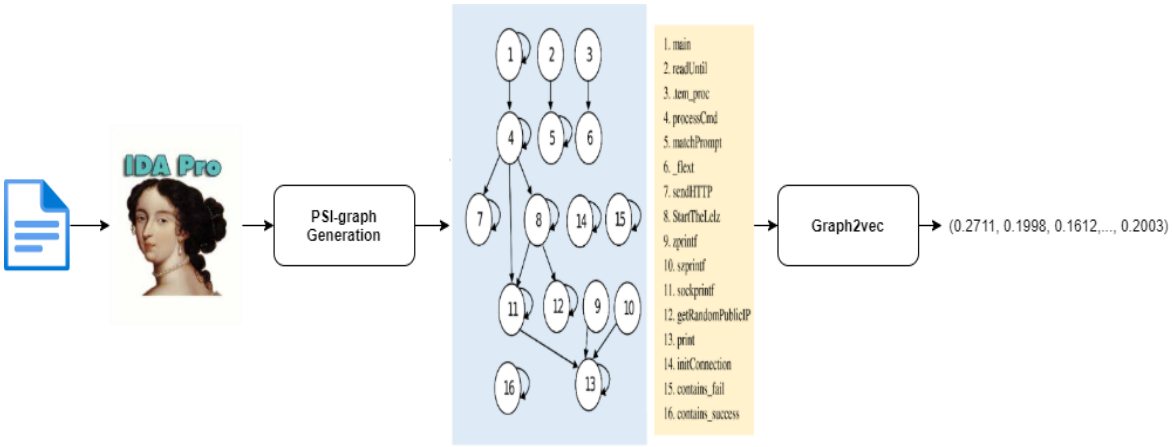
Đồ thị PSI là đồ thị có hướng được định nghĩa bởi $G_{PSI} = (V, E)$*, với:*

- V là tập các các đỉnh được tạo thành từ các hàm trong đồ thị hàm gọi và có chứa các thông tin chuỗi in PSI,

- E là tập các cạnh có hướng $\{(V_i, V_j), (V_k, V_h), \dots\}$ *thể hiện mối quan hệ gọi và được gọi giữa các hàm.*

2.1.3. Xây dựng tập đặc trưng tĩnh

Sau khi đã xác định lựa chọn việc trích xuất các đặc trưng tĩnh từ đồ thị thông tin chuỗi in PSI, việc xây dựng tập đặc trưng tĩnh cho phương pháp lai sẽ được thực hiện thông qua các bước được thể hiện trong Hình 2.1.



Hình 2.1: Quy trình xây dựng tập đặc trưng tĩnh

Bước 1: Tiến hành dịch ngược các tập tin ELF sử dụng công cụ giải nén UPX và công cụ dịch ngược IDA Pro để xây dựng đồ thị hàm gọi FCG.

Đồ thị hàm gọi FCG được định nghĩa là đồ thị có hướng $G = (V, E)$, với V là tập các đỉnh tương ứng với các hàm và E là tập các cạnh tương ứng với mối quan hệ gọi và được gọi giữa các hàm [16]. Thông thường, trong một chương trình, sẽ có hai loại hàm được sử dụng là các hàm cục bộ và các hàm ngoại vi. Các hàm cục bộ được thực thi để xử lý các tác vụ trong chương trình, còn các hàm ngoại vi thường là các hàm hệ thống hoặc thư viện. Các hàm cục bộ có thể có rất nhiều tên gọi khác nhau cho có dù cùng một chức năng. Trong khi đó, các hàm ngoại vi thường sẽ được đặt

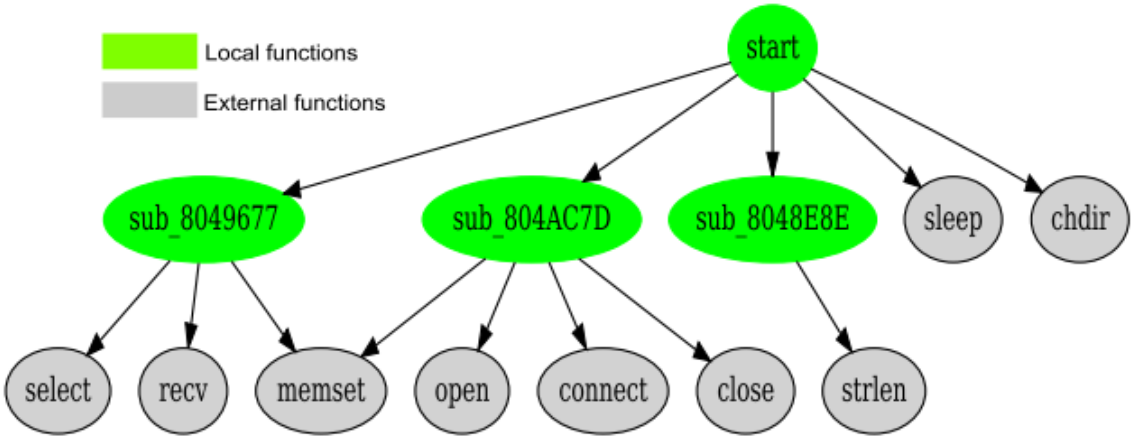
tên thống nhất trong cả chương trình. Bên cạnh đó, các hàm ngoại vi thông thường sẽ không gọi đến các hàm cục bộ vì chúng được lấy từ hệ thống hoặc các thư viện hàm.

- Với mỗi tập tin thực thi, trước khi tiến hành dịch ngược, cần kiểm tra xem tập tin đó có bị nén hay không và nén bởi công cụ nào. Công đoạn này có thể thực hiện với công cụ phát hiện tập tin nén Detect It Easy [37].

- Kế đến, nếu tập tin được xác định bị nén thì sẽ phải sử dụng một số công cụ để hỗ trợ giải nén như UPX [38].

- Đối với những tập tin đã được giải nén, có thể sử dụng công cụ dịch ngược IDA Pro [29] để lấy được mã assembly từ mã nhị phân và một tập hợp các hàm xác định của chương trình.

- Đồ thị hàm gọi FCG sẽ được xây dựng dựa trên những thông tin thu được bằng thuật toán tìm kiếm theo chiều rộng [23]. Bắt đầu từ hàm entry point, quan hệ gọi và được gọi giữa các hàm sẽ được xác định và lần lượt được thêm vào đồ thị. Ví dụ về một đồ thị có chứa các hàm cục bộ và ngoại vi được biểu diễn trong Hình 2.2.



Hình 2.2: Một phần đồ thị hàm gọi của mẫu mã độc Linux.Mirai

(Nguồn: A novel graph-based approach for IoT botnet detection)

Bước 2: Xây dựng đồ thị thông tin chuỗi in PSI.

Đồ thị thông tin chuỗi in PSI được xây dựng bằng cách lược bớt các hàm và các quan hệ trong đồ thị hàm gọi FCG, chỉ giữ lại những hàm và quan hệ tượng trưng cho các quy trình hoạt động tiêu biểu của mã độc IoT Botnet. Do đó, đồ thị thông tin

chuỗi in PSI sẽ có số đỉnh và cạnh ít hơn đồ thị hàm gọi FCG nhiều lần, từ đó giảm bớt độ phức tạp cần tính toán.

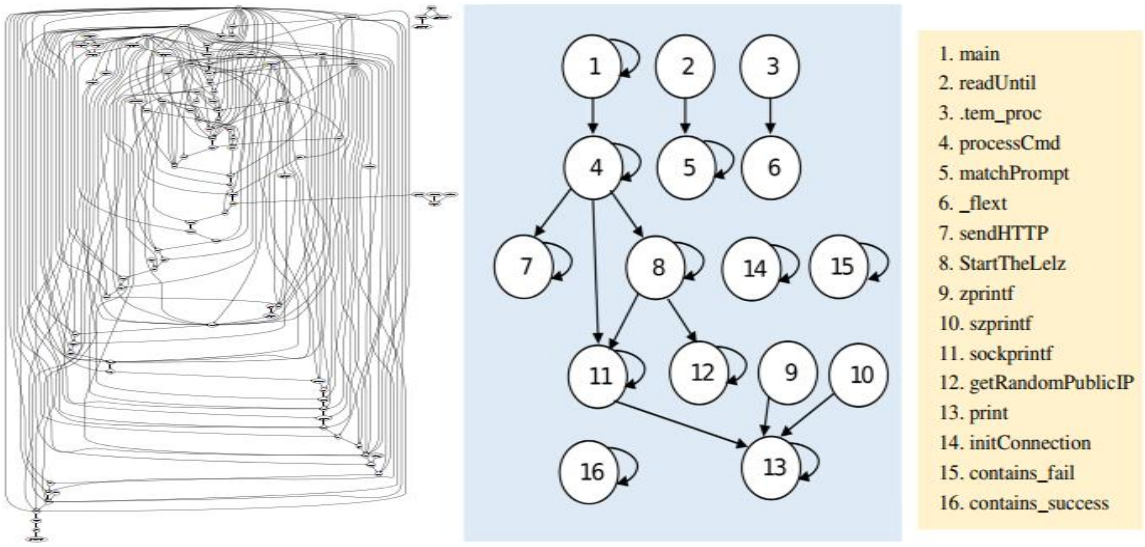
- Trước khi xây dựng đồ thị PSI, cần phải trích xuất tất cả các thông tin chuỗi in từ mã nhị phân của tập tin bằng cách sử dụng một plug-in của công cụ IDA Pro. Các thông tin chuỗi in này thường chứa những thông tin ngữ nghĩa quan trọng có thể xác định được mục đích của kẻ tấn công. Ví dụ các chuỗi như "/dev/watchdog; /dev/misc/watchdog" luôn luôn xuất hiện trong mã độc Linux.Mirai thể hiện việc Botnet ngăn không cho thiết IoT khởi động lại [22]. Trong một vài trường hợp, chuỗi được trích xuất ra có thể bị làm rối hoặc bị mã hóa.

- Sau khi lấy được những thông tin chuỗi in từ các mẫu tập tin thực thi, đồ thị PSI sẽ được xây dựng tương ứng với từng mẫu theo thuật toán [18].

Thuật toán: Xây dựng đồ thị PSI (FCG)

1: Khởi tạo V = [], E = []
2: **For** mỗi đỉnh v_i trong FCG **do:**
3: **If exist** psi trong v_i và $v_i \notin V$ **do:**
4: $V = V \cup v_i$
5: **End if**
6: **For** mỗi cạnh $e_j(v_j, v_k)$ **do:**
7: **If exist** psi trong v_k và $v_k \notin V$ và $e_j(v_j, v_k) \notin E$ **do:**
8: $V = V \cup v_k$
9: E= $E \cup e_j(v_j, v_k)$
10: **End If**
11: **End for**
12: **End for**
13: **Return** V, E

Để đảm bảo sự cân bằng giữa độ chính xác của thuật toán phân loại và độ phức tạp tính toán, các hàm được chọn phải chứa các thông tin chuỗi in với ít nhất 3 ký tự. Hình 2.3 cho thấy một ví dụ của đồ thị hàm gọi FCG và đồ thị thông tin chuỗi in PSI được trích xuất từ mã nguồn của một mẫu mã độc Linux.Bashlite.



Hình 2.3: Đồ thị FCG và đồ thị PSI của một mẫu mã độc Linux.Bashlite [18]

Bước 3: Chuyển hóa đồ thị thông tin chuỗi in PSI thành vector đặc trưng tĩnh.

Trong đồ thị PSI, mỗi chuỗi liên kết các hoạt động của mã độc IoT Botnet sẽ được thể hiện dưới dạng một đồ thị con. Như vậy, tất cả các đồ thị con trong đồ thị sẽ cho thấy toàn bộ hành vi của mã độc botnet. Và việc xem xét tất cả các đồ thị con có thể có để đại diện cho toàn bộ đồ thị là điều cần thiết.

Có thể thấy, việc xử lý mối liên hệ giữa toàn bộ đồ thị và các đồ thị con khá tương tự việc với xử lý một văn bản với các từ trong văn bản đó. Như vậy, một kỹ thuật xử lý khá phù hợp có thể dùng trong trường hợp này là graph2vec [2]. Graph2vec được xây dựng dựa trên ý tưởng của doc2vec, sử dụng mạng skip-gram, coi toàn bộ đồ thị như một văn bản và các đồ thị con như các từ trong văn bản.

Graph2vec sẽ thực hiện xử lý đồ thị PSI thông qua 3 bước sau:

- Lấy mẫu và dán nhãn lại toàn bộ các đồ thị con. Mỗi đồ thị con là một tập hợp các đỉnh tồn tại xung quanh đỉnh được chọn. Các đỉnh trong đồ thị con không được ở xa hơn số cạnh đã chọn.

- Huấn luyện mô hình skipgram. Tương tự văn bản là một tập các từ, toàn bộ đồ thị sẽ là một tập các đồ thị con. Mô hình skipgram được huấn luyện để tối đa hóa xác suất dự đoán đồ thị con tồn tại trong đồ thị đầu vào. Đồ thị đầu vào được cung cấp dưới dạng one-hot vector.

- Tính toán các embedding vector để cung cấp định danh cho mỗi đồ thị như một one-hot vector ở đầu vào. Các embedding vector là kết quả của lớp hidden layer.

Điểm mạnh của graph2vec nắm ở việc thuật toán xem xét và bảo toàn cấu trúc đỉnh và cạnh của toàn bộ đồ thị thông qua các đồ thị con. Do đó, các đồ thị càng có cấu trúc giống nhau thì các embedding vector tương ứng trong không gian đặc trưng càng gần.

Kết quả của quy trình này là tập các one-hot vector với chiều dài tùy ý tượng trưng cho tập các đồ thị PSI của các mẫu tệp thực thi ELF.

2.2. Xây dựng các đặc trưng động

2.2.1. Một số đặc trưng động trong phát hiện mã độc IoT Botnet

Khác với phân tích tĩnh, tính hiệu quả của phân tích động chủ yếu xoay quanh việc thực thi mã độc trong môi trường thời gian thực và giám sát hành vi của chúng. Quá trình phân tích động càng thu thập được nhiều thông tin đặc trưng về hành vi của mã độc thì quá trình việc phân loại mã độc về sau sẽ càng hiệu quả. Một số loại đặc trưng động chủ yếu thường được thu thập bao gồm: các lời gọi hệ thống (System call), thông tin mở rộng của các lời gọi hệ thống (EXINFO), lưu lượng truy cập mạng và các thông tin về hiệu năng máy chủ [17]. Thông tin tóm tắt về các đặc trưng này được trình bày trong Bảng 2.1.

Bảng 2.1: Các đặc trưng động thường được sử dụng trong phát hiện mã độc IoT Botnet

STT	Loại đặc trưng	Đặc trưng	Mô tả đặc trưng
1	System call	Syscall_time_stamp	Timestamp cho mỗi system-call
2		Syscall_name	Tên của system-call
3		Syscall_list_arg	Các đối số của system-call
4		Syscall_ret	Giá trị trả về của system-call
5		Syscall_pid	ID của tiến trình được gọi bởi tệp tin ELF
6		PID_num	Số tiến trình được tạo bởi tệp tin ELF
7	EXINFO	exinfo_name	Tên exinfo
8		exinfo_type	Loại exinfo

STT	Loại đặc trưng	Đặc trưng	Mô tả đặc trưng
9		exinfo_path	Đường dẫn exinfo
10	Lưu lượng truy cập mạng	Net_service	Loại dịch vụ (HTTP, FTP, SMTP,...)
11		Net_sbytes	Số byte gửi
12		Net_sttl	Giá trị thời gian sống
13		Net_smean	Kích thước gói tin trung bình gửi từ địa chỉ IP nguồn
14		Net_ ct_dst_sport_ltm	Số lượng kết nối của cùng một địa chỉ đích và cổng nguồn trong 100 kết nối theo thời gian cuối cùng.
15	Hiệu năng CPU	Num_User_login	Số phiên hoạt động của người dùng
16		Total_process_active	Tổng số tiến trình được kích hoạt trong hệ thống
17		Num_process_running	Số tiến trình đang chạy
18		Num_process_sleeping	Số tiến trình đang đợi
19		Num_process_stop	Số tiến trình đã ngừng
20		Num_process_zombie	Số tiến trình đang chờ để ngừng từ tiến trình mẹ
21		CPU_percent_user	Phần trăm CPU cho các tiến trình người dùng
22		CPU_percent_sys	Phần trăm CPU cho các tiến trình hệ thống
23		CPU_percent_ni	Phần trăm các tiến trình CPU với ưu tiên nâng cấp NICE
24		CPU_percent_ide	Phần trăm CPU không sử dụng
25		CPU_percent_wait	Phần trăm các tiến trình CPU đang đợi các hoạt động I/O
26		CPU_percent_hd_ir	Phần trăm CPU phục vụ cho ngắt từ phần cứng
27		CPU_percent_st_ir	Phần trăm CPU phục vụ cho ngắt từ phần mềm
28		CPU_percent_st	Lượng CPU bị trình giả lập lấy từ máy ảo này cho các tác vụ khác (chạy máy ảo khác)
29	Hiệu năng các tiến trình	Process_PID	ID của tiến trình
30		Process_User	Người dùng tiến trình
31		Process_PR	Mức ưu tiên của tiến trình
32		Process_NI	Giá trị NICE của tiến trình
33		Process_Virt	Bộ nhớ ảo được sử dụng bởi tiến trình

STT	Loại đặc trưng	Đặc trưng	Mô tả đặc trưng
34		Process_RES	Bộ nhớ thực được sử dụng bởi tiến trình
35		Process_SHR	Bộ nhớ chia sẻ của tiến trình
36		Process_S	Trạng thái của tiến trình: S=sleep R=running Z=zombie
37		Process_percent_cpu	Phần trăm CPU được dùng bởi tiến trình
38		Process_percent_mem	Phần trăm RAM được dùng bởi tiến trình
39		Process_time	Tổng thời gian hoạt động của tiến trình
40		Process_comm	Tên của tiến trình
41	Hiệu năng RAM	Mem_total	Tổng dung lượng nhớ của hệ thống
42		Mem_used	Dung lượng nhớ đang được hệ thống sử dụng
43		Mem_free	Dụng lượng nhớ còn trống
44		Mem_buff	Tổng dung lượng nhớ được bộ đệm sử dụng
45	Bộ nhớ thay thế	Swap_total	Tổng dung lượng bộ nhớ thay thế trong hệ thống
46		Swap_free	Dụng lượng nhớ thay thế còn trống
47		Swap_used	Dụng lượng nhớ thay thế hệ thống đang sử dụng
48		Swap_cached	Tổng dung lượng cache hệ thống sử dụng

(Nguồn: V-Sandbox For Dynamic Analysis IoT Botnet)

2.2.2. Lựa chọn môi trường giám sát thời gian thực

Lựa chọn môi trường thời gian thực để giám sát và thu thập các hành vi đặc trưng của mã độc cũng là một yếu tố hết sức quan trọng trong quy trình phân tích động. Môi trường hoạt động chủ yếu các mã độc IoT Botnet là trên các thiết bị IoT với những giới hạn về tài nguyên và bộ nhớ. Bên cạnh đó, các mã độc IoT Botnet còn có khả năng hoạt động trên nhiều nền tảng kiến trúc khác nhau. Do vậy, môi trường máy ảo hoặc sandbox được sử dụng để mô phỏng giám sát cần phải đáp ứng được một số tiêu chí như:

- Hỗ trợ nhiều nền tảng kiến trúc CPU;

- Tự động cấu hình môi trường mô phỏng để chạy các tập tin thực thi;

- Không gây ảnh hưởng đến môi trường bên ngoài môi trường mô phỏng;

- Cung cấp được các thư viện mà tập tin thực thi yêu cầu để hoạt động bình thường;

- Cung cấp môi trường mạng mô phỏng đầy đủ cho hoạt động của mã độc IoT Botnet như kết nối với máy chủ C&C, truyền và nhận lệnh từ máy chủ C&C;

- Giám sát được các hành vi của mã độc IoT Botnet gây ảnh hưởng tới môi trường giả lập (các lời gọi hệ thống, thay đổi của các tập tin, thư mục, lưu lượng truy cập mạng, hiệu năng phần cứng,...)

Hiện nay, đã có một số sandbox có thể tạo môi trường mô phỏng chạy các tập tin thực thi cho thiết bị IoT như IoTBOX [28], LiSA [31], V-Sandbox [17]... Trong đó, công cụ IoTBOX được xây dựng để phân tích các mã độc IoT tấn công dựa trên Telnet và hỗ trợ 8 kiến trúc CPU bao gồm MIPS, MIPSEL, PPC, SPARC, ARM, MIPS64, sh4, and X86. Tuy nhiên, IoTBOX chỉ tập trung giám sát các hành vi mạng mà không đề cập đến các hành vi khác của mã độc. Công cụ Linux Sandbox LiSa thì hỗ trợ phân tích chủ yếu các mã độc Linux chạy trên các kiến trúc MIPS, ARM, Intel 80386, x86-64, Aarch64. Tuy nhiên, công cụ này lại không hỗ trợ vai máy chủ C&C, thứ đóng vai trò quan trọng trong việc cho phép các mã độc IoT Botnet thể hiện hành vi. Bên cạnh đó, các template liên kết động thường không thể chạy trên LiSA do thiếu thư viện hỗ trợ.

Vì những lý do trên, luận văn đã lựa chọn công cụ V-sanbox để mô phỏng môi trường thời gian thực giám sát các hành vi của mã độc IoT Botnet và trích xuất các đặc trưng hành vi. Điểm mạnh của V-sandbox là có thể hỗ trợ bổ sung các thư viện liên kết động còn thiếu, đồng thời cung cấp mô phỏng máy chủ C&C và một danh sách các lệnh điều khiển thường gặp được gửi từ máy chủ C&C. Thông qua đó, môi trường mô phỏng của V-sandbox có thể kích hoạt các hành vi độc hại của mã độc IoT Botnet như thực hiện tấn công từ chối dịch vụ.

Do điểm mạnh của V-sandbox nằm ở khả năng kích hoạt bổ sung các hành vi độc hại của mã độc IoT, nên dữ liệu của các lời gọi hệ thống sẽ bao gồm nhiều thông

tin hơn như các lời gọi kết nối, gửi, nhận thông tin,... Vì vậy, luận văn sẽ tập trung khai thác dữ liệu của các lời gọi hệ thống để xây dựng tập đặc trưng động.

2.2.3. Xây dựng tập đặc trưng động

Như vậy, quy trình xây dựng tập đặc trưng động sẽ được thực hiện thông qua ba bước như được thể hiện trong Hình 2.4.

Hình 2.4: Quy trình xây dựng tập đặc trưng động

Bước 1: Thực thi các tập tin trong môi trường mô trường có giám sát. Các tệp ELF sẽ được đưa vào môi trường mô phỏng của V-sandbox để giám sát và trích xuất các hành vi đặc trưng. Kết quả đầu ra của V-sanbox là một tập tin báo cáo các lời gọi hệ thống mà tệp ELF đã thể hiện trong môi trường sandbox.

Bước 2: Dựa trên dữ liệu báo cáo các lời gọi hệ thống để xây dựng đồ thị lời gọi hệ thống SCG. Cách thức xây dựng đồ thị này cũng tương tự xây dựng đồ thị hàm gọi FCG thông qua việc duyệt thứ tự các lời gọi hệ thống.

Bước 3: Chuyển hóa đồ thị lời gọi hệ thống SCG thành vector đặng trưng động. Tương tự việc chuyển hóa các vector đặc trưng tĩnh, các đồ thị lời gọi hệ thống SCG cũng được chuyển hóa thành tập các vector đặc trưng động thông qua thuật toán graph2vec.

Kết quả của quy trình này cũng là một tập các one-hot vector với chiều dài tùy ý tượng trưng cho tập các đồ thị SCG của từng mẫu tệp thực thi ELF.

2.3. Phương pháp tích hợp đặc trưng

2.3.1. Lựa chọn phương pháp tích hợp đặc trưng tĩnh và động

Như đã đề cập trong phần 1.3.3, có rất nhiều hướng tiếp cận cho phương pháp lai trong phát hiện mã độc dựa trên việc tích hợp rất nhiều các đặc trưng được trích xuất từ đặc điểm của tệp tin như Opcodes, chuỗi các ký tự, v.v... Tuy nhiên, cũng cần phải tính đến ảnh hưởng của những đặc trưng đó tới độ chính xác trong phát hiện mã độc. Rất nhiều loại đặc trưng được xây dựng dựa trên chính bản chất của mã độc để chúng có thể được đánh giá một cách toàn diện và hạn chế phát hiện sai. Dù vậy, khi tổng hợp lại tất cả các đặc trưng đó để phân loại thì chúng chưa hẳn đã có thể phát huy được hết mức. Trên thực tế, việc kết nhiều mức độ nhiễu khác nhau và sự xung đột dữ liệu khiến cho độ chính xác của phương pháp giảm xuống là điều hoàn toàn có thể xảy ra. Vì lý do đó, luận văn đã lựa chọn việc giới hạn chỉ kết hợp hai đặc trưng của tập tin thực thi ELF là đồ thị thông tin chuỗi in PSI và đồ thị lời gọi hệ thống SCG. Các đồ thị đặc trưng này đã được chuyển hóa thành hai tập vector đặc trưng tiêu biểu cho hai phương pháp tĩnh và động. Hai tập này có sự tương đồng về không gian vector do đều được chuyển hóa thông qua thuật toán graph2vec.

Như vậy, phương pháp tích hợp hai tập đặc trưng này cần đảm bảo sự đơn giản về dữ liệu, tránh nhiễu và tối ưu thời gian tính toán.

Trong các hướng tích hợp vector đặc trưng, có ba hướng tiếp cận có thể kết hợp các loại vector đặc trưng là kết quả của những quá trình phân tích khác nhau, bao gồm: Early fusion, Late fusion và Intermediate fusion [9].

- Early fusion là phương pháp kết hợp được thực hiện ở mức đặc trưng. Các vector từ nhiều tập đặc trưng khác nhau được nối lần lượt với nhau thành một vector đặc trưng lớn, và sau đó được sử dụng làm đầu vào cho thuật toán phân loại.

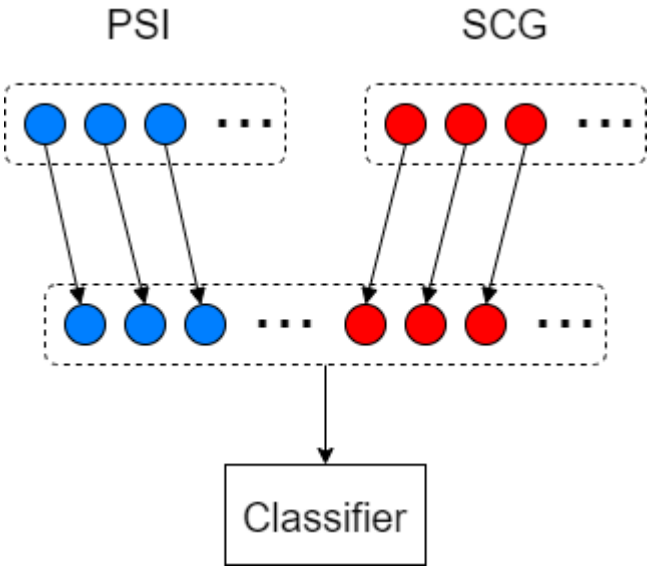
- Late fusion là dạng kết hợp được thực hiện sau đối với đầu ra của thuật toán phân loại đối với từng tập vector đặc trưng là đầu vào. Phương pháp này huấn luyện từng tập vector đặc trưng riêng lẻ với từng mô hình phân loại riêng, sau đó hợp nhất các quyết định đầu ra thông qua các cơ chế hợp nhất như: bỏ phiếu, lấy giá trị cao nhất, lấy giá trị trung bình cao nhất,… Ưu điểm của phương pháp late fusion là lựa chọn được mô hình phân loại phù hợp cho từng tập vector đặc trưng riêng.

- Intermediate fusion là phương pháp xây dựng tập vector đặc trưng bằng cách hợp nhất các vector đặc trưng thông qua một mạng học sâu đơn giản và thực hiện phân loại.

Trong ba hướng tiếp cận, phương pháp Intermediate fusion sử dụng mạng học sâu để phân loại, phương pháp Late fusion thì lại phù hợp hơn trong trường hợp các vector đặc trưng các sự khác biệt rõ ràng và không tương đồng về không gian vector. Do đó hai phương pháp này không phù hợp cho tích hợp các đặc trưng tĩnh và động để cải thiện độ chính xác trong phân loại của từng phương pháp tĩnh và động riêng lẻ. Vì vậy, luận văn đã lựa chọn sử dụng phương pháp Early fusion để kết hợp hai tập vector đặc trưng tĩnh và động. Do vector đặc trưng kết hợp chỉ cấu thành từ hai loại đặc trưng nên vẫn sẽ tránh được nhiễu và đảm bảo thời gian huấn luyện và phân loại ngắn. Đồng thời, với việc vận dụng thuật toán phân loại hợp lý vẫn sẽ có thể cho ra kết quả phát hiện với độ chính xác cao [25].

2.3.2. Xây dựng tập các đặc trưng lai giữa tĩnh và động

Tập các đặc trưng lai giữa tĩnh và động được xây dựng sử dụng phương pháp early fusion để kết hợp tập vector đặc trưng tĩnh trích xuất từ đồ thị thông tin chuỗi in PSI và tập vector đặc trưng động trích xuất từ đồ thị lời gọi hệ thống SCG. Quy trình thực hiện được mô tả như trong Hình 2.5



Hình 2.5: Xây dựng tập vector đặc trưng lai giữa tĩnh và động

Vector đặc trưng thu được sau quá trình kết hợp sẽ có dạng:

$$v_{fused} = [v_{psi}, v_{scg}]$$

Trước khi đưa tập vector đặc trưng lai vào các thuật toán phân loại, cần phải thực hiện kỹ thuật lựa chọn đặc trưng. Kỹ thuật này không chỉ hỗ trợ trong việc giảm kích thước dữ liệu để giảm thời gian tính toán, mà còn giúp ích trong việc loại bỏ các đặc trưng dư thừa, gây nhiễu. Thêm vào đó, kỹ thuật này còn cho phép tạo ra các mô hình học máy có độ chính xác cao và ngăn ngừa vấn đề quá vừa dữ liệu.

Kỹ thuật lựa chọn đặc trưng được sử dụng cho tập vector đặc trưng lai là kỹ thuật chọn đặc trưng bằng mô hình bọc với SVM (Support Vector Machine) tuyến tính. Do tập đặc trưng của phương pháp lai không nhiều nên việc sử dụng kỹ thuật chọn đặc trưng với mô hình bọc sẽ cho ra tập đặc trưng phù hợp trong thời gian ngắn.

Tiếp theo, tập vector đặc trưng thu được đại diện cho mỗi tập tin ELF sẽ được chuẩn hóa để tập dữ liệu có giá trị kỳ vọng bằng không và độ lệch chuẩn bằng 1. Như vậy, với mỗi vector x:

$$x_{standardized} = \frac{x - \mu}{\sigma}$$

Với μ và σ là lần lượt giá trị kỳ vọng và độ lệch chuẩn của tập dữ liệu ban đầu. Cuối cùng, tập dữ liệu sau khi chuẩn hóa sẽ được đưa vào các mô hình học máy phân loại để huấn luyện và kiểm tra.

2.4. Các thuật toán phân loại mã độc

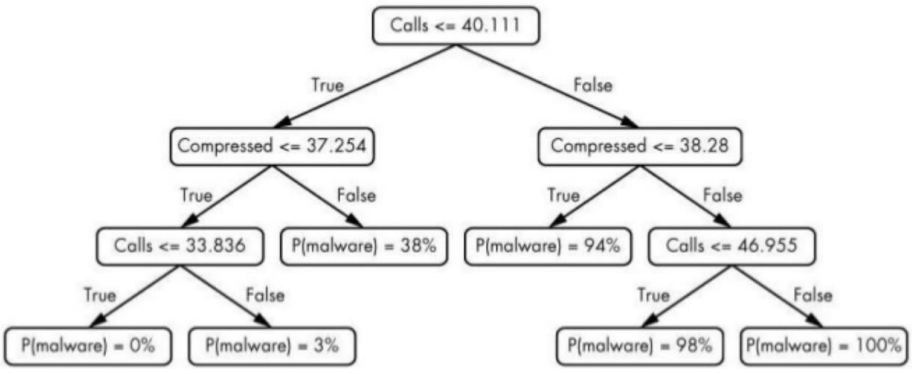
Hiện nay, có rất nhiều nghiên cứu sử dụng các kỹ thuật học máy trong phân loại mã độc [1], [7], [20], [21], [26]. Trong đó, các thuật toán phân loại như thuật toán Cây quyết định (Decision Tree), thuật toán k-láng giềng gần nhất, Support Vector Machines và Rừng ngẫu nhiên (Random Forest) là những thuật toán thường sẽ cho ra độ chính xác cao. Nguyên nhân là bởi bài toán phát hiện mã độc là bài toán phân loại hai lớp. Mục tiêu của bài toán là phân loại được một tệp tin thực thi có nhãn là mã độc hoặc lành tính. Do đó, các thuật toán với nguyên tắc phân chia các ví dụ có nhãn khác nhau thường sẽ có độ chính xác cao.

Vì những lý do trên, luận văn đã lựa chọn sử dụng bốn thuật toán này cho bài toán phân loại mã độc với tập vector đặc trưng lai giữa tĩnh và động. Dưới đây sẽ trình bày sơ lược về bốn thuật toán này.

2.4.1. Cây quyết định (DT)

Cây quyết định là một cấu trúc ra quyết định có dạng cây. Cây quyết định nhận đầu vào là một bộ giá trị đặc trưng mô tả một đối tượng hay một tình huống và trả về một giá trị rời rạc. Mỗi bộ đặc trưng vào được gọi là một mẫu hay một ví dụ, đầu ra gọi là loại hay nhãn phân loại.

Cây quyết định được biểu diễn dưới dạng một cấu trúc cây. Mỗi nút trung gian, tức là nút không phải nút lá, tương ứng với phép kiểm tra một đặc trưng. Mỗi nhánh phía dưới của nút đó tương ứng với một giá trị của đặc trưng hay một kết quả của phép thử. Khác với nút trung gian, nút lá không chứa đặc trưng mà chứa nhãn phân loại. Để xác định nhãn phân loại cho một mẫu nào đó, ta cho mẫu chuyển động từ gốc cây về phía nút lá. Tại mỗi nút, đặc trưng tương ứng với nút được kiểm tra, tùy theo giá trị của thuộc tính đó mà mẫu được chuyển xuống nhánh tương ứng bên dưới. Quá trình này lặp lại cho đến khi mẫu tới được nút lá và được nhận nhãn phân loại là nhãn của nút lá tương ứng.



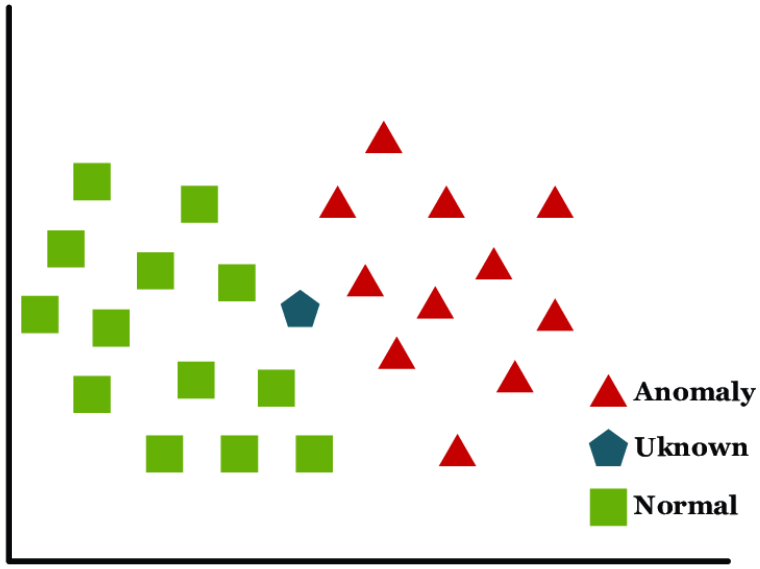
Hình 2.6: Ví dụ đơn giản về phân loại cây quyết định (DT)

Nhiệm vụ của thuật toán học là xây dựng cây quyết định phù hợp với tập dữ liệu huấn luyện, tức là cây quyết định có đầu ra giống (nhiều nhất) với nhãn phân loại cho trong tập mẫu. Trong trường hợp số đặc trưng ít, việc xây dựng cây quyết định như

vậy có thể thực hiện bằng cách liệt kê tất các cây quyết định hợp lệ và kiểm tra để chọn ra cây phù hợp với dữ liệu.

2.4.2. K-láng giềng gần nhất (k-NN)

Các bộ phân loại k-láng giềng gần nhất dựa trên việc học bằng sự giống nhau. Các mẫu huấn luyện được mô tả bởi các thuộc tính số n-chiều. Mỗi số đại diện cho một điểm trong không gian n-chiều. Vì vậy tất cả các mẫu huấn luyện được lưu trữ trong không gian n-chiều. Khi có một mẫu chưa biết cho trước thì bộ phân loại k-láng giềng gần sẽ tìm kiếm trong không gian mẫu k mẫu huấn luyện gần mẫu chưa biết đó nhất; k mẫu huấn luyện này là k "láng giềng gần nhất" của mẫu chưa biết. Mẫu chưa biết được phân vào lớp phổ biến nhất trong số k láng giềng gần nhất của nó.



Hình 2.7: Ví dụ đơn giản về phân loại k-láng giềng gần nhất (k-NN)

Với phương pháp này, với mỗi một vector đặc trưng có một nhãn, 2 vector có cùng nhãn sẽ giống nhau còn lại sẽ khác nhau. Với một vector mới nó sẽ được gán nhãn theo các vector có đặc điểm giống nó nhất.

2.4.3. Support Vector Machines (SVM)

Đây là một trong những thuật toán học máy phổ biến nhất. Nguyên tắc của SVM là tìm một siêu phẳng phù hợp với lề cực đại, tức là khoảng cách từ siêu phẳng tới

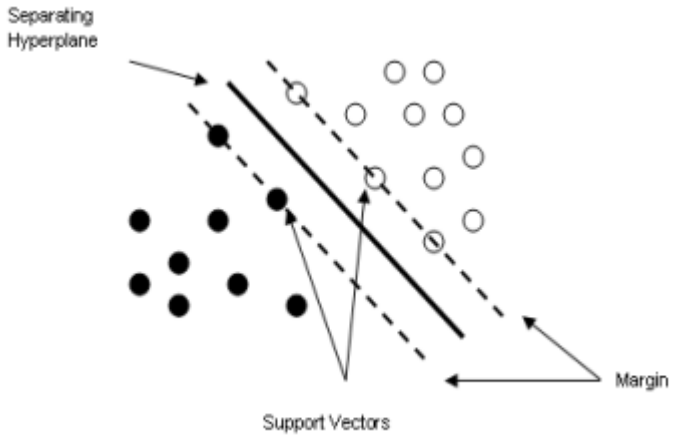
những ví dụ có nhãn khác nhau là lớn nhất. Giả sử ta có một hàm tuyến tính phân biệt và hai nhãn có thể phân tách tuyến tính với các giá trị mục tiêu là +1 và -1. Một siêu phẳng phân biệt phải thỏa mãn:

$$w'x_i + w_0 \geq 0 \; nếu \; t_i = +1$$

$$w'x_i + w_0 < 0 \; nếu \; t_i = -1$$

Khoảng cách từ vector x bất kỳ tới siêu phẳng là $|w'x_i + w_0| \; / \; \|w\|$ và khoảng cách tới gốc tọa độ là $|w_0| \; / \; \|w\|$. Các vector xác định ranh giới của lề được gọi là các vector tựa (support vector).

SVM là một thuật toán khá mạnh và thường được dùng trong các bài toán phân loại. Tuy nhiên, thuật toán này yêu cầu khả năng tính toán khá cao để huấn luyện với tập dữ liệu. Ngoài ra, thuật toán cũng khá nhạy với các dữ liệu nhiễu và thường gặp phải vấn đề quá vừa dữ liệu.



Hình 2.8: Phân loại với Support Vector Machines (SVM)

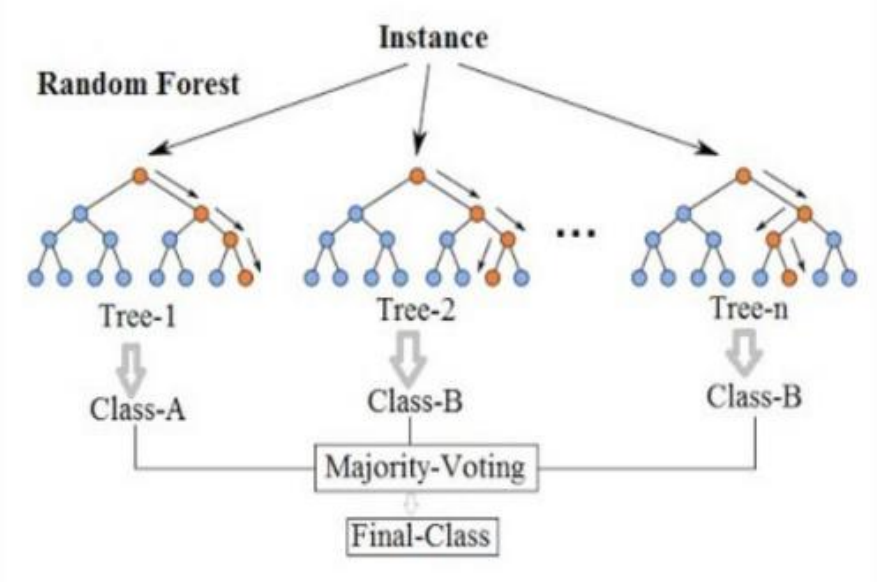
2.4.4. Random Forest (RF)

Random Forest (RF) là thuật toán phân loại bằng cách xây dựng nhiều cây quyết định từ tập dữ liệu. Mỗi cây sẽ được xây dựng độc lập dựa trên giá trị của một vector ngẫu nhiên. Tất cả các cây quyết định đều được phân bổ tương tự như nhau.

Để xây dựng cây, giả sử n là số các training observation, và p là số các đặc trưng trong tập huấn luyện. Để xác định các nút quyết định của cây, lấy k << p là số các đặc trưng được chọn. Sau đó, chọn các mẫu bootstrap từ n các observation trong tập huấn luyện và dùng các observation còn lại để ước tính lỗi của cây trong giai đoạn

kiểm tra. Tiếp theo, ta chọn ngẫu nhiên k đặc trưng ở một số nút quyết định và tính toán cách phân tách tốt nhất dựa trên k đặc trưng trong tập huấn luyện. So với các thuật toán cây quyết định khác, trong thuật toán này, cây sẽ luôn phát triển và không bị tỉa.

Thuật toán Ramdom Forest có thể xử lý một lượng lớn các đặc trưng trong tập dữ liệu. Ngoài ra, trong quá trình xây dựng các cây, chúng cũng tạo ra sự ước lượng không mang tính khuynh hướng của các lỗi tổng quát. Thêm vào đó, thuật toán còn có thể ước tính khá tốt những dữ liệu bị thiếu. Hạn chế của Ramdom Forest là việc không có khả năng tái lập lại, do quá trình xây dựng cây là ngẫu nhiên. Ngoài ra, việc diễn giải mô hình cuối cùng và các kết quả sau đó cũng khá khó khăn bởi số lượng lớn cây quyết định.



Hình 2.9: Phân loại với Random Forest (RF)

Kết luận chương 2

Nội dung chương 2 của luận văn đã trình bày kết quả của quá trình nghiên cứu, tìm hiểu hướng tiếp cận dựa trên phương pháp lai trong phát hiện mã độc IoT Botnet. Hướng tiếp cận của phương pháp lai mà luận văn đã chọn là tiến hành phân tích tĩnh và động song song, sau đó kết hợp các đặc trưng thu được thành một tập đặc trưng lai dùng cho phát hiện mã độc IoT Botnet.

Trong giai đoạn phân tích tĩnh, luận văn đã lựa chọn sử dụng các đặc trưng tĩnh là đồ thị thông tin chuỗi in (PSI) và sử dụng công cụ graph2vec để trích xuất thành các vector đặc trưng tĩnh.

Trong giai đoạn phân tích động, luận văn đã lựa chọn sử dụng công cụ V-Sandbox làm môi trường giám sát thời gian thực và khai thác dữ liệu của các lời gọi hệ thống để xây dựng tập đặc trưng động. Đồ thị lời gọi hệ thống SCG được xây dựng dựa trên các dữ liệu lời gọi hệ thống thu thập được từ sandbox và thông qua công cụ graph2vec để trích xuất thành các vector đặc trưng động.

Để tích hợp tập các vector đặc trưng tĩnh và động thành tập các vector đặc trưng lai, luận văn đã sử dụng phương pháp tích hợp đặc trưng Early fusion. Phương pháp kết hợp đặc trưng này sẽ đảm bảo sự đơn giản về dữ liệu, tránh nhiễu và tối ưu thời gian tính toán đối với tập vector đặc trưng lai.

Trong chương này, luận văn cũng lựa chọn sử dụng bốn thuật toán phân loại phù hợp cho phương pháp lai thử nghiệm, bao gồm các thuật toán: Cây quyết định, K-láng giềng gần nhất, Support Vector Machines và Random Forest.

Mô hình phương pháp lai thử nghiệm được xây dựng trong chương 2 sẽ là cơ sở để tiến hành thực nghiệm và đánh giá hiệu quả của phương pháp này được tiến hành trong chương 3.

Chương 3: THỬ NGHIỆM VÀ ĐÁNH GIÁ

3.1. Xây dựng tập dữ liệu

3.1.1. Phương pháp thu thập các mẫu mã độc IoT Botnet và lành tính

Để đảm bảo việc đánh giá mô hình phân loại một cách chính xác và khách quan thì vai trò của tập dữ liệu là rất quan trọng. Do đó, cần thu thập các mẫu mã độc IoT Botnet và lành tính với số lượng đủ nhiều và có phương pháp gắn nhãn chính xác.

Dựa trên tiêu chí đó, tập dữ liệu được thu thập từ các nguồn như sau:

- Đối với các mẫu mã độc IoT, có hai nguồn thu thập chính mà luận văn đã lựa chọn. Nguồn thứ nhất là từ nhóm dự án IoTPOT [28]. Dự án này đã thu thập được rất nhiều các mẫu mã độc trong vòng một năm từ tháng 10/2016 đến 10/2017. Và nguồn thứ hai là từ VirusShare [39], kho lưu trữ các mẫu mã độc online phục vụ cho mục đích nghiên cứu.

- Đối với các mẫu lành tính, luận văn thu thập từ ba nguồn, bao gồm: trích xuất từ các thiết bị IoT SOHO sử dụng công cụ binwalk [40], tải về từ trang web của các hãng sản xuất thiết bị IoT như TP-Link, D-Link, Asus..., đồng thời thu thập các mẫu tệp firmware của một số hãng như Buffalo, Netgear, Tenda... từ OpenWRT [41].

Sau khi thu thập, các tệp tin sẽ được kiểm tra một lần nữa sử dụng dịch vụ VirusTotal [42] để đảm bảo độ chính xác của các nhãn mã độc và lành tính. Tiếp theo, các mẫu bị trùng lặp, hư hại hoặc không phải tập tin ELF sẽ bị loại bỏ để đảm bảo tính khách quan cho tập dữ liệu. Bên cạnh đó, để có thể so sánh được chính xác nhất kết quả thực nghiệm của phương pháp lai với hai phương pháp phân tích tĩnh và động, các mẫu không thể dịch ngược trong quá trình phân tích tĩnh và không được hỗ trợ hoặc không thể thực thi trong sandbox cũng sẽ bị loại bỏ. Do đó, tập dữ liệu đạt được sẽ đảm bảo tính thống nhất trong thực nghiệm với từng phương pháp.

3.1.2. Mô tả tập dữ liệu

Như vậy, sau khi thực hiện các phương pháp trên, tập dữ liệu thu được bao gồm 6520 mẫu, trong đó có 4644 mẫu mã độc IoT Botnet và 1876 mẫu tập tin IoT lành tính. Các mẫu mã độc IoT Botnet chủ yếu bao gồm các họ mã độc như

Bashlite/Gafgyt, Mirai, Tsunami/Kaiten, Spike/Dofloo/MrBlack,... Phân bố kiến trúc của các mẫu mã độc và lành tính được thể hiện trong Bảng 3.1.

Bảng 3.1: Phân bố kiến trúc của tập dữ liệu

Kiến trúc	Mã độc	Lành tính
ARM	1180	594
MIPS	1063	1261
Intel 80386	1522	11
x86-64	426	3
PowerPC	453	7
Tổng	**4644**	**1876**

Tập dữ liệu sẽ được chia ra thành hai tập con phục vụ cho việc huấn luyện và kiểm tra với tỷ lệ phân chia lần lượt là 70% và 30%.

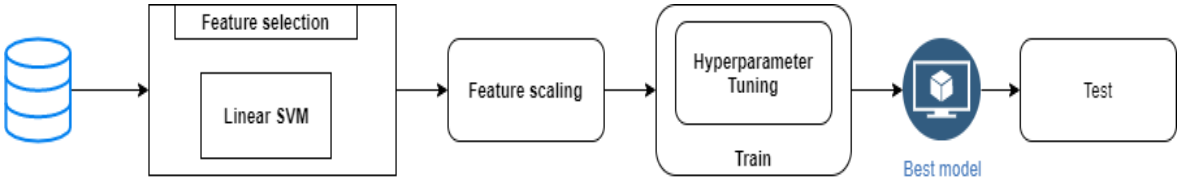
3.2. Phương pháp đánh giá và các độ đo sử dụng

3.2.1. Phương pháp đánh giá

Để đánh giá độ chính xác của phương pháp lai trong phát hiện mã độc IoT Botnet, luận văn sẽ thực nghiệm tập vector đặc trưng lai với các bộ phân loại Cây quyết định, K-láng giềng gần nhất, Support Vector Machines và Random Forest. Bên cạnh đó, để so sánh sự nổi trội của phương pháp lai, luận văn cũng sẽ thực hiện các thực nghiệm tương tự với hai tập vector đặc trưng của đồ thị thông tin chuỗi in PSI và đồ thị lời gọi hệ thống SCG riêng biệt.

Tất cả các thực nghiệm được thực hiện trên cùng một tập dữ liệu đã trình bày trong phần 3.1 với mã nguồn được viết bằng ngôn ngũ Python và chạy trên máy tính với cấu hình Intel Xeon CPU E5-2650 v2 @ 2.60GHz × 4 with 8 GB RAM, hệ điều hành Ubuntu 16.04 LTS.

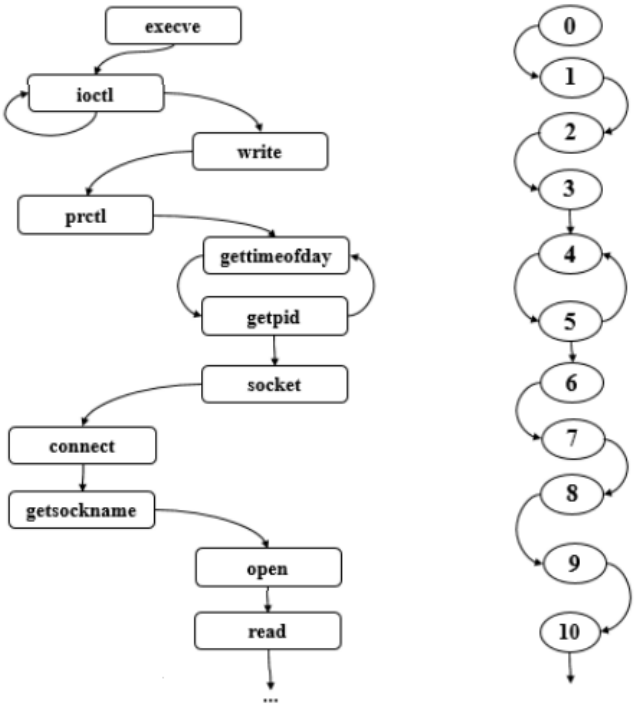
Để xử lý vấn đề thiếu cân bằng giữa các mẫu mã độc và lành tính trong tập dữ liệu, các bộ phân loại Cây quyết định, Support Vector Machines và Random Forest sẽ được đánh trọng số cho nhãn. Tập huấn luyện và tập kiểm thử được phân chia theo tỷ lệ 7:3. Cách thức chia này sẽ đảm bảo tỷ lệ số lượng mẫu dữ liệu của từng nhãn trong hai tập là như nhau. Quá trình thực nghiệm được thể hiện trong Hình 3.1



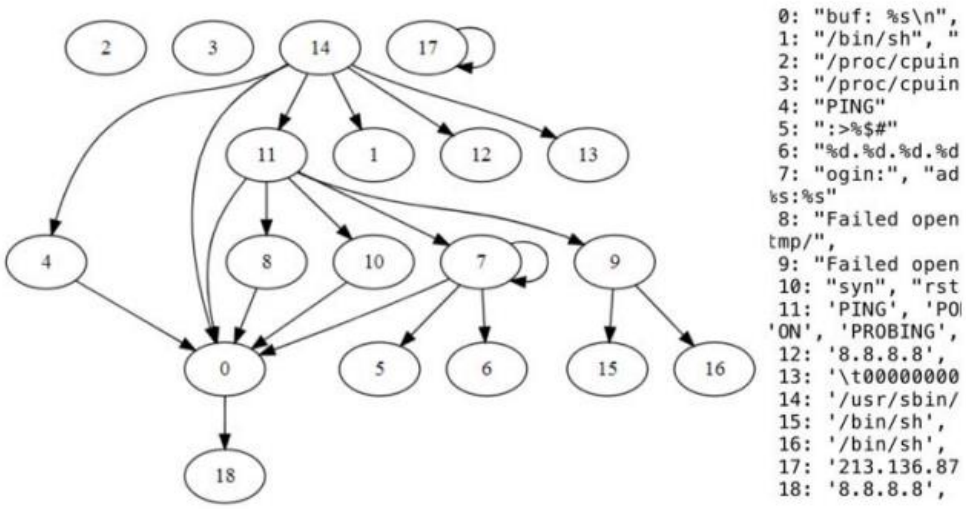
Hình 3.1: Thực nghiệm phân loại mã độc

Đầu tiên, các mẫu được đưa qua xử lý sinh dữ liệu đồ thị PSI và SCG theo quy trình như đã trình bày ở chương 2. Mỗi mẫu tương ứng sẽ lần lượt được thu thập dữ liệu đồ thị PSI và SCG. Hình 3.2 và 3.3 là ví dụ đồ thị SCG và PSI của một mẫu mã độc Mirai.

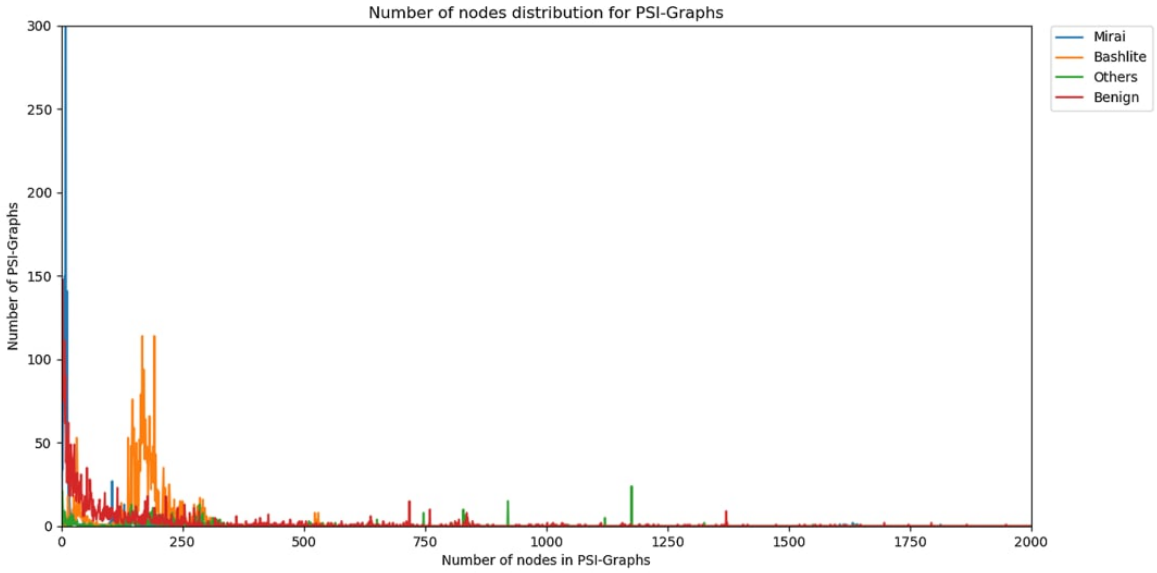
Số lượng đỉnh và cạnh của đồ thị SCG và PSI tùy thuộc vào từng tệp tin ELF mà sẽ khác nhau. Với đồ thị PSI thì đỉnh của nó sẽ là các hàm có chứa chuỗi in, còn ở đồ thị SCG thì đỉnh sẽ là các syscall. Trong giới hạn các mẫu của tập dữ liệu thì số lượng đỉnh đồ thị PSI của một mẫu sẽ nằm trong khoảng từ 1 đến 300 đỉnh. Tương tự, đồ thị SCG cũng được lựa chọn số đỉnh trung bình vào khoảng 300 syscall graph.



Hình 3.2: Ví dụ đồ thị SCG của một mẫu mã độc Mirai

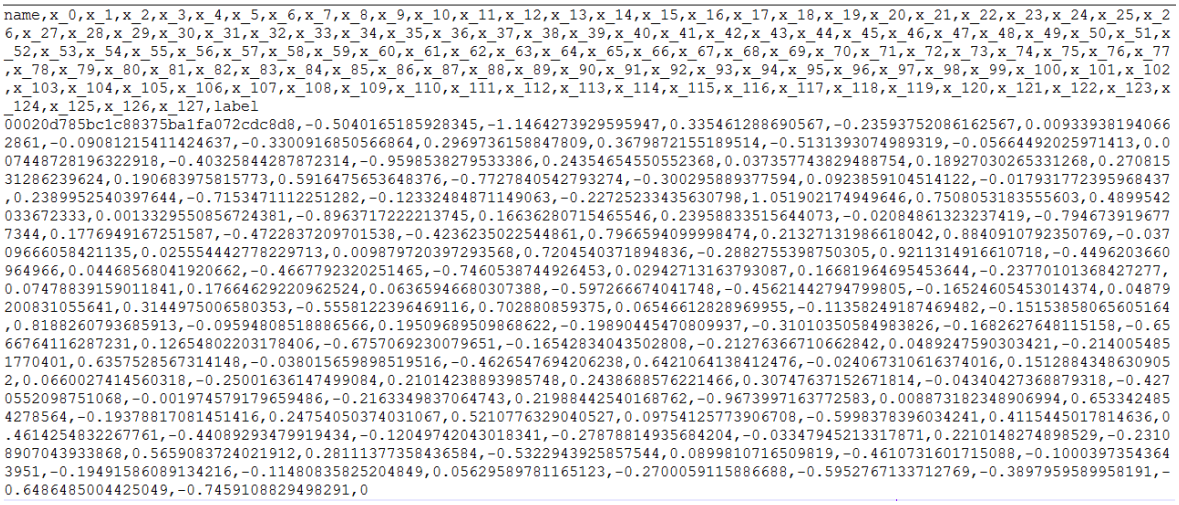


Hình 3.3: Ví dụ đồ thị PSI của một mẫu mã độc Mirai



Hình 3.4: Phân bố số lượng đồ thị PSI của các tập mẫu

Dữ liệu đồ thị PSI và SCG sau đó sẽ được vector hóa theo phương pháp graph2vec. Các vector đặc trưng đồ thị PSI và SCG của từng mẫu sẽ được lưu trữ với số chiều là 128 cho mỗi loại vector. Số chiều 128 của vector là kết quả của quá trình xử lý với thuật toán graph2vec.



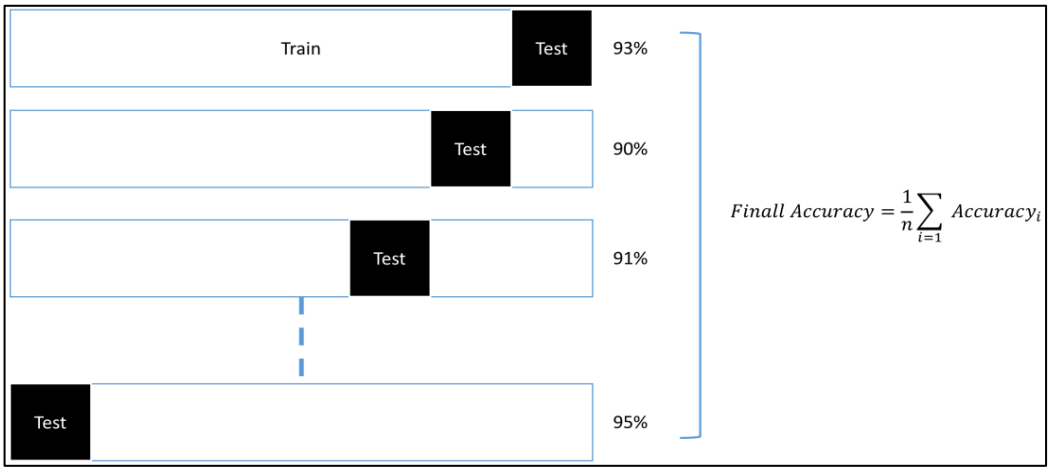
Hình 3.5: Ví dụ dữ liệu lưu trữ dạng vector của đồ thị PSI và đồ thị SCG

Tuy nhiên dữ liệu vector này được xem là một mẫu đa biến số và dữ liệu này có thể được biểu diễn ở dạng ma trận với hàng và cột, luận văn thấy rằng ma trận thu được có các đặc trưng với nhiều khoảng giá trị khác nhau nên cần chuẩn hóa để đảm bảo kết quả phân loại, quá trình chuẩn hóa theo công thức:

$$x_{standardized} = \frac{x - \mu}{\sigma}$$

Với μ và σ là kỳ vọng và độ lệch chuẩn của tập dữ liệu ban đầu một cách tương ứng.

Sau đó, luận văn đưa dữ liệu vector trên vào các mô hình học máy đã lựa chọn. Để đa dạng bộ dữ liệu, luận văn sử dụng kỹ thuật đánh giá chéo 5-fold. Trong đó, tham số được đánh giá bằng cách huấn luyện mô hình trên 4 fold và kiểm thử trên fold còn lại. Cuối cùng, mô hình có tham số cho kết quả đánh giá trung bình tốt nhất được huấn luyện trên toàn bộ tập huấn luyện và kiểm thử trên bộ kiểm thử. Toàn bộ quá trình lựa chọn mô hình với tham số được tiến hành tự động theo code có sẵn với kết quả đầu ra là mô hình với tham số cho kết quả tốt nhất. Nguyên lý của kỹ thuật đánh giá chéo k-fold được mô tả ở Hình 3.6.



Hình 3.6: Kỹ thuật đánh giá chéo k-fold

Cuối cùng, luận văn sử dụng các mô hình đã tối ưu tương ứng với từng thuật toán để tiến hành phân loại. Để giải quyết vấn đề mất cân bằng giữa số lượng mẫu mã độc và số lượng mẫu lành tính, luận văn đã sử dụng thêm kỹ thuật đánh trọng số cho nhãn. Theo đó, nhãn thiểu số sẽ được đánh trọng số cao hơn, và thuật toán bị phạt nặng hơn khi dự đoán sai nhãn thiểu số.

Quá trình điều chỉnh siêu tham số cho các thuật toán học máy bằng cách tìm kiếm siêu tham số bằng lưới tham số được trình bày trong Bảng 3.2.

Bảng 3.2: Bảng giá trị tham số tương ứng với mỗi thuật toán

STT	**Thuật toán**	**Tham số**	**Giá trị**
1	Decision Tree	Tiêu chí chia node	- Sử dụng Entropy - Sử dụng Gini
2	kNN	Số hàng xóm	[5,100,500]
		Trọng số	- Uniform: hàng xóm là như nhau - Distance: hàng xóm gần hơn có ảnh hưởng lớn hơn
3	SVM (RBF kernel)	C	[0.001,0.001, 0.1,1,10,100,1000]
		Gamma	[0.001,0.001, 0.1,1,10,100,1000]

4	Random Forest	Tiêu chí chia node	- Sử dụng Entropy - Sử dụng Gini
		Số cây	[10,100,1000]
		Boostrap	[Có, không]

3.2.2. Các độ đo sử dụng để đánh giá

Tương tự như các nghiên cứu về phân loại mã độc, ta quy ước các khái niệm sẽ được sử dụng để tính hiệu quả phân loại như sau:

- Dương đúng (TP): thể hiện việc một mẫu mã độc được phân loại đúng và được gắn nhãn là mã độc.

- Âm đúng (TN): thể hiện việc một mẫu lành tính được phân loại đúng và được gắn nhãn là lành tính.

- Dương sai (FP): thể hiện việc một mẫu lành tính được phân loại sai và được gắn nhãn là mã độc.

- Âm sai (FN): thể hiện việc một mẫu mã độc được phân loại sai và được gắn nhãn là lành tính.

Dựa trên các quy ước khái niệm TP, TN, FP, FN như trên, ta có thể tính ra các độ đo được dùng để đánh giá hiệu quả phân loại như sau:

- Độ chính xác $Accuracy = \frac{TP+TN}{TP+FP+TN+FN}$

- Tỷ lệ dương đúng $TPR = \frac{TP}{TP+FN}$

- Tỷ lệ dương sai $FPR = \frac{FP}{FP+TN}$

- Độ chính xác $Precision = \frac{TP}{TP+FP}$

- Độ đo $F1 = 2\frac{Precision*TPR}{Precision+TPR}$

Bên cạnh đó, luận văn cũng sử dụng thêm độ đo diện tích phía dưới đường cong ROC để đánh giá. Đây là đường cong biểu diễn khả năng phân loại của một mô hình phân loại tại các ngưỡng dựa trên hai độ đo dương đúng và dương sai. Phần diện tích phía dưới đường cong ROC và trên trục hoành có giá trị nằm trong khoảng [0, 1]. Khi

diện tích này càng lớn thì đường cong ROC có xu hướng tiệm cận đường thẳng y = 1 và khả năng phân loại của mô hình càng tốt.

Trong thực nghiệm của luận văn, đường cong ROC và diện tích phía dưới đường cong ROC được tính toán và vẽ sử dụng các hàm *roc_curve* và *roc_auc_score* có sẵn trong thư viện sklearn.

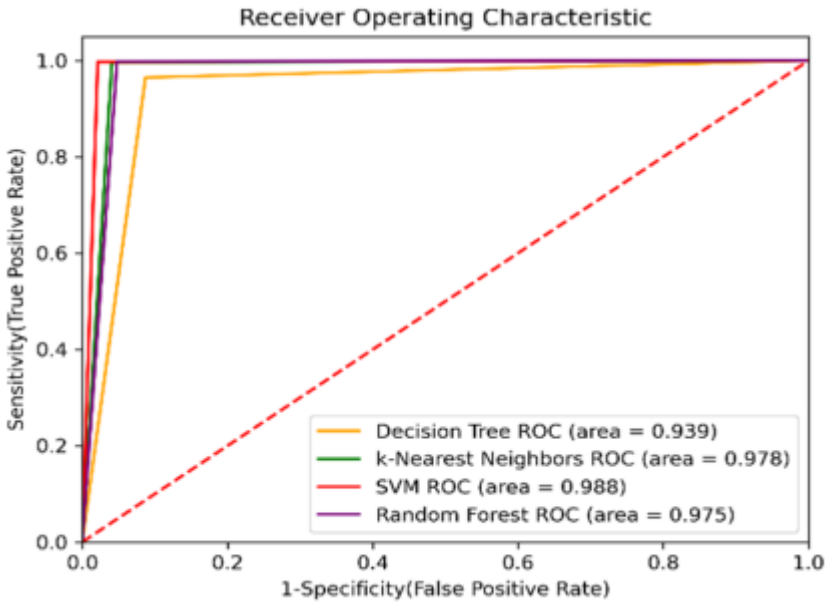
3.3. Kết quả thực nghiệm

Kết quả thực nghiệm của phương pháp lai với 4 bộ phân loại Cây quyết định, K-láng giềng gần nhất, Support Vector Machines và Random Forest được thể hiện trong Bảng 3.3. Các độ đo đánh giá lần lượt là độ chính xác Accuracy, diện tích phía dưới đường cong ROC, tỷ lệ dương sai, độ chính xác Precision, tỷ lệ dương đúng và độ đo F1.

Bảng 3.3: Kết quả thực nghiệm với phương pháp lai

(%)	DT	k-NN	SVM	RF
ACC	94.99	98.57	99.18	98.52
ROC AUC	93.89	97.78	98.79	97.53
FPR	8.70	4.09	2.13	4.80
Precision	96.48	98.37	99.14	98.10
TPR	96.48	99.64	99.71	99.86
F1	96.48	99.00	99.43	98.97

Mối quan hệ giữa tỷ lệ dương đúng và dương sai được biểu diễn thông qua đường cong ROC tương ứng với từng bộ phân loại như trong Hình 3.7



Hình 3.7: Đường cong ROC của các bộ phân loại

Bên cạnh thực nghiệm đánh giá hiệu quả của phương pháp lai, các thực nghiệm đánh giá đối với hai tập vector đặc trưng của đồ thị thông tin chuỗi in PSI và đồ thị lời gọi hệ thống SCG cũng được thực hiện để so sánh. Kết quả so sánh của phương pháp lai với hai phương pháp phân tích tĩnh và động được thể hiện trong Bảng 3.4.

Bảng 3.4: So sánh phương pháp phân tích động, phân tích tĩnh và phương pháp lai

(%)	PSI	SCG	Lai
ACC	96.83	98.41	99.18
ROC AUC	96.66	97.75	98.79
FPR	3.73	3.71	2.13
Precision	98.47	98.61	99.14
TPR	97.06	99.2	99.71
F1	97.76	98.91	99.43

3.4. Đánh giá và so sánh

Đánh giá kết quả thực nghiệm đối với phương pháp lai trong Bảng 3.3, ta có thể thấy phương pháp lai tích hợp các đặc trưng giữa tĩnh và động do luận văn thực hiện có thể phát hiện mã độc IoT Botnet một cách khá hiệu quả. Cả bốn bộ phân loại đều cho tỷ lệ chính xác cao với tỷ lệ dương sai thấp. Trong đó, mô hình cho ra kết quả

cao nhất là Support Vector Machines với độ chính là 99,18% cùng tỷ lệ dương sai thấp vào khoảng 2,13%. Diện tích phía dưới đường cong của bộ phân loại Support Vector Machines cũng là cao nhất với giá trị 98,79%, cho thấy bộ phân loại không bị bias do số lượng lớn các mẫu mã độc trong tập dữ liệu. Các bộ phân loại khác cũng cho ra kết quả tương tự với độ chính xác cao. Diện tích phía dưới đường cong ROC lớn, thể hiện khả năng phân loại hai nhãn mã độc và lành tính khá tốt.

Đối với thực nghiệm riêng lẻ với giữa hai phương pháp phân tích tĩnh và động, kết quả cũng cho thấy bộ phân loại Support Vector Machines cho ra kết quả tốt nhất đối với cả phương pháp tĩnh và động. Bên cạnh đó, dựa trên dữ liệu từ Bảng 3.4, ta có thể thấy phương pháp lai cho ra kết quả tốt hơn so với việc sử dụng riêng lẻ từng tập đặc trưng tĩnh và động cho thực nghiệm. Đồng thời, ảnh hưởng của hai đặc trưng tĩnh và động đối với đặc trưng lai cũng gần như ngang bằng nhau, do độ chính xác khi sử dụng riêng lẻ từng đặc trưng không có sự chênh lệch quá lớn. Như vậy, việc kết hợp các đặc trưng tĩnh và động riêng lẻ với ảnh hưởng ngang bằng nhau trong trường hợp này là lựa chọn phù hợp.

Kế đến, luận văn tiến hành so sánh mô hình phát hiện mã độc IoT Botnet sử dụng phương pháp lai đã thực hiện với các mô hình phát hiện mã độc IoT Botnet trong một số nghiên cứu khác như sử dụng đặc trưng Opcode kết hợp với mạng nơ-ron hồi quy (RNN) của HaddadPajouh và các cộng sự [10]; hay phương pháp sử dụng đặc trưng ảnh đa mức xám kết hợp với mạng nơ-ron tích chập (CNN) của Su và các cộng sự [12]. Kết quả so sánh được trình bày trong Bảng 3.5.

Bảng 3.5: So sánh với một số phương pháp khác

Phương pháp	ACC (%)	AUC (%)	Precision (%)	TPR (%)	F1 (%)
HaddadPajouh [10]	96,35	96,28	97,74	96,53	97,13
Su [12]	95,13	95,23	96,9	94,67	95,77
Luận văn	99,18	98,79	99,14	99,71	99,43

Có thể thấy phương pháp lai do luận văn thực hiện cũng cho ra kết quả cao hơn so với một số phương pháp phát hiện mã độc IoT Botnet khác. Kết quả này cũng thể

hiện tính hiệu quả và cải tiến của phương pháp lai so với các phương pháp tĩnh và động đơn thuần.

Kết luận chương 3

Trong chương 3, luận văn đã trình bày về quá trình thử nghiệm và đánh giá mô hình phương pháp lai đã trình bày ở chương 2, bao gồm xây dựng tập dữ liệu, lựa chọn phương pháp đánh giá, tiến hành thử nghiệm, và cuối cùng là so sánh và đưa ra kết luận đánh giá.

Tập dữ liệu mà luận văn đã thu thập được bao gồm 6520 mẫu, trong đó có 4644 mẫu mã độc IoT Botnet và 1876 mẫu tập tin IoT lành tính. Tập vector đặc trưng lai được thực nghiệm với bốn bộ phân loại đã chọn và đánh giá dựa trên các độ đo như: độ chính xác Accuracy, diện tích phía dưới đường cong ROC, tỷ lệ dương sai, độ chính xác Precision, tỷ lệ dương đúng và độ đo F1. Bên cạnh đó, để thể hiện sự nổi trội của phương pháp lai, các thực nghiệm tương tự đối với hai tập vector tĩnh và động riêng lẻ cũng được thực hiện để so sánh. Đồng thời, phương pháp lai thử nghiệm cũng được so sánh với một số phương pháp khác trong phát hiện mã độc IoT Botnet.

Kết quả đánh giá cho thấy phương pháp lai thử nghiệm có độ chính xác khá cao và tỷ lệ dương sai thấp, có khả năng phân loại mã độc và lành tính khá tốt. Đồng thời, kết quả này cũng thể hiện được tính hiệu quả và cải tiến của phương pháp lai thử nghiệm trong luận văn.

KẾT LUẬN VÀ KIẾN NGHỊ

Sự phát triển mạnh mẽ của mạng Internet và đặc biệt thời gian gần đây là Internet vạn vật (IoT - Internet of Things) đã mang lại các tiện ích phong phú cho người dùng như trong truyền nhận dữ liệu, liên lạc, quản lý, theo dõi sức khoẻ, thiết bị tự hành hay cơ sở hạ tầng thông minh. Phân tích và phát hiện mã độc IoT Botnet trên các thiết bị IoT là một hướng quan trọng về mặt lý thuyết cũng như ứng dụng, là cơ sở nền tảng để mở rộng để nghiên cứu cho tất cả các loại mã độc trên thiết bị IoT, và là một trong những nội dung quan trọng để phòng chống nguy cơ tấn công mạng trong hoạt động đảm bảo an ninh mạng, góp phần bảo vệ an ninh quốc gia, trật tự an toàn xã hội.

Để xác định phạm vi nghiên cứu, luận văn đã tiến hành nghiên cứu các phương pháp phân tích, phát hiện mã độc Botnet trên thiết bị IoT, đồng thời tìm hiểu hướng tiếp cận sử dụng phương pháp lai trong phát hiện mã độc IoT Botnet. Mô hình của phương pháp lai đã được tiến hành thực nghiệm với tập dữ liệu các mẫu IoT Botnet được thu thập trên thực tế. Kết quả đánh giá đã cho thấy rõ ưu thế của phương pháp lai so với các phương pháp tĩnh và động đơn thuần.

Kết quả của luận văn là bước đầu cho việc mở rộng hướng phát hiện mã độc IoT Botnet dựa trên phương pháp lai, kết hợp các ưu điểm tốt nhất của các phương pháp phân tích tĩnh và phân tích động. Nội dung nghiên cứu của luận văn được trình bày và công bố tại Kỷ yếu hội nghị quốc tế, cụ thể là *"IoT Botnet Detection Based on the Integration of Static and Dynamic Vector Features"* tại Hội nghị quốc tế lần thứ tám về Truyền thông và Điện tử (ICCE 2020) và *"Toward an approach using graph-theoretic for IoT botnet detection"* tại Hội nghị quốc tế về máy tính, mạng và Internet vạn vật (CNIOT 2021)

Tuy nhiên, luận văn vẫn còn một số hạn chế trong xây dựng tập đặc trưng lai. Cụ thể là những khó khăn trong việc dịch ngược một số mẫu mã độc IoT Botnet để trích xuất đồ thị PSI. Bên cạnh đó, do công cụ V-sandbox mới chỉ hỗ trợ 5 loại nền tảng kiến trúc và danh sách các chỉ thị của máy chủ C&C mô phỏng vẫn còn hạn chế nên chưa thể mô phỏng được hết các mẫu mã độc. Ngoài ra, phương pháp tích hợp

các đặc trưng tĩnh và động còn khá đơn giản và chỉ phù hợp khi tích hợp số ít các loại vector đặc trưng. Trong trường hợp cần tích hợp nhiều loại đặc trưng cả tĩnh và động thì phương pháp tích hợp này có thể sẽ không hiệu quả.

Dựa trên kết quả nghiên cứu, luận văn đưa ra một số kiến nghị cho các hướng phát triển trong tương lai như sau:

- Tiếp tục cải thiện phương pháp tích hợp các đặc trưng tĩnh và động, để có thể tích hợp được nhiều đặc trưng mạnh hơn, góp phần cải thiện độ chính xác của phương pháp phát hiện mã độc IoT Botnet nói riêng và mã độc IoT nói chung.

- Nghiên cứu các phương pháp phòng thủ, bảo vệ các thiết bị IoT khỏi bị lây nhiễm bởi mã độc IoT Botnet nói chung và mã độc IoT nói chung. Từ đó xây dựng các bộ công cụ kiểm định, đánh giá độ an ninh, an toàn của thiết bị IoT trước khi đưa vào sử dụng trong các hệ thống thông tin, nhất là các hệ thống thông tin quan trọng về an ninh quốc gia.

- Nghiên cứu các phương pháp phòng, chống hoạt động tấn công mạng sử dụng mã độc IoT Botnet của các cá nhân, tổ chức. Xây dựng quy trình ứng phó với phương thức tấn công mạng này để xác định nguồn gốc, loại trừ hành vi tấn công mạng và thu thập các tài liệu chứng cứ phục vụ quá trình điều tra, truy tố.

DANH MỤC TÀI LIỆU THAM KHẢO

[1] Ajit Kumar , K. S. Kuppusamy, and G. Aghila, (2019), "A learning model to detect maliciousness of portable executable using integrated feature set", *Journal of King Saud University-Computer and Information Sciences*, 31(2), 252-265.

[2] Annamalai Narayanan, M. Chandramohan, R. Venkatesan, L. Chen, Y. Liu and S. Jaiswal, (2017), "graph2vec: Learning distributed representations of graphs", *arXiv preprint*, arXiv:1707.05005.

[3] Aohui Wang, R. Liang, X. Liu, Y. Zhang, K. Chen, and J. Li, (2017), "An inside look at IoT malware", In *International Conference on Industrial IoT Technologies and Applications*, Springer, Cham, pp. 176-186

[4] Bertino, Elisa, and Nayeem Islam, (2017), "Botnets and internet of things security", *Computer*, 50(2), 76-79.

[5] Constantinos Kolias, (2017), "DDoS in the IoT: Mirai and other botnets", *Computer*, 50(7), 80-84.

[6] Dange, Smita, and Madhumita Chatterjee, (2020), "IoT Botnet: The Largest Threat to the IoT Network", *In Data Communication and Networks*, Springer, Singapore, pp. 137-157.

[7] Gandotra, Ekta, Divya Bansal, and Sanjeev Sofat, (2014), "Integrated framework for classification of malwares". In *Proceedings of the 7th International Conference on Security of Information and Networks*, pp. 417-422.

[8] Gary Davis, (2020), "Life with 50 Billion Connected Devices", 2018 IEEE Int. *In Conf. Consum. Electron* , p. 1.

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

[10] Hamed HaddadPajouh, A. Dehghantanha, R. Khayami and K. K. R. Choo, (2018), “A deep recurrent neural network based approach for internet of things malware threat hunting”, *Future Generation Computer Systems*, vol. 85, pp. 88-96.

[11] Jeon, Jueun, Jong Hyuk Park, and Young-Sik Jeong, (2020), “Dynamic Analysis for IoT Malware Detection with Convolution Neural Network model”, *IEEE Access*.

[12] Jiawei Su, D. V. Vasconcellos, S. Prasad, D. Sgandurra, Y. Feng, and K. Sakurai, (2018), “Lightweight classification of IoT malware based on image recognition”, *In 2018 IEEE 42Nd annual computer software and applications conference (COMPSAC)*, IEEE, Vol. 2, pp. 664-669.

[13] Kishore Angrishi, (2017), “Turning internet of things (iot) into internet of vulnerabilities (iov): Iot botnets”, *arXiv preprint arXiv:1702.03681*.

[14] Kumar, Sathish Alampalayam, Tyler Vealey, and Harshit Srivastava, (2016), “Security in internet of things: Challenges, solutions and future directions”, *In 2016 49th Hawaii International Conference on System Sciences (HICSS)*, IEEE, pp. 5772-5781.

[15] Michele De Donno, N. Dragoni, A. Giaretta and A. Spognardi, (2017), “Analysis of DDoS-capable IoT malwares”, In *2017 Federated Conference on Computer Science and Information Systems (FedCSIS)*, IEEE, pp. 807-816.

[16] Ming Xu, L. Wu, S. Qi, J. Xu, H. Zhang, Y. Ren and N. Zheng, (2013), “A similarity metric method of obfuscated malware using function-call graph”, *Journal of Computer Virology and Hacking Techniques*, 9(1), 35-47.

[17] Ngo Quoc Dung, Le Hai Viet, (2020), “V-Sandbox for Dynamic Analysis IoT Botnet”, *IEEE Access*, 8, 145768-145786.

[18] Ngo Quoc Dung, Le Van Hoang and Nguyen Huy Trung (2020), “A novel graph-based approach for IoT botnet detection”, *International Journal of Information Security*, 19(5), 567-577.

[19] Ngo Quoc Dung, Le Van Hoang, Nguyen Doan Hieu and Nguyen Huy Trung (2020), “A survey of IoT malware and detection methods based on static features”, *ICT Express*.

[20] P. V. Shijo and A. Salim, (2015), “Integrated static and dynamic analysis for malware detection”, *Procedia Computer Science*, 46, 804-811.

[21] Rafiqul Islam, R. Tian, L. M. Batten and S. Versteeg, (2013), “Classification of malware based on integrated static and dynamic features”, *Journal of Network and Computer Applications*, 36(2), 646-656.

[22] Roger Hallman, J. Bryan, G. Palavicini, J. Divita, and J. Romero-Mariona, (2017), “IoDDoS-the internet of distributed denial of sevice attacks”, In *2nd international conference on internet of things, big data and security. SCITEPRESS*, pp. 47-58.

[23] Shanhu Shang, N. Zheng, J. Xu, M. Xu, and H. Zhang, (2010), “Detecting malware variants via function-call graph similarity”. In *2010 5th International Conference on Malicious and Unwanted Software*, IEEE, pp. 113-120.

[24] Silva Sérgio SC, R. M. Silva, R. C. Pinto, and R. M. Salles, (2013), “Botnets: A survey”, *Computer Networks*, 57(2), 378-403.

[25] Ngo Quoc Dung, Nguyen Huy Trung, Tran Hoang Anh and Nguyen Doan Hieu, (2021), “IoT Botnet detection based on the integration of static and dynamic vector features”, In *2020 IEEE Eighth International Conference on Communications and Electronics (ICCE)*, IEEE, pp. 540-545.

[26] Ucci, Daniele, Leonardo Aniello, and Roberto Baldoni, (2019), “Survey of machine learning techniques for malware analysis”, *Computers & Security*, 81, 123-147.

[27] Yao Saint Yen, Z. W. Chen, Y. R. Guo and M. C. Chen, (2019), “Integration of Static and Dynamic Analysis for Malware Family Classification with Composite Neural Network”, *arXiv*, arXiv-1912.

[28] Yin Minn Pa Pa, S. Suzuki, K. Yoshioka, T. Matsumoto, T. Kasama and C. Rossow, (2016), "IoTPOT: A novel honeypot for revealing current IoT threats", *Journal of Information Processing*, 24(3), 522-533.

[29] A. Darki, M. Faloutsos, N. Abu-Ghazaleh and M. Sridharan, (2019), *IDAPro for IoT Malware analysis?*, In 12th {USENIX} Workshop on Cyber Security Experimentation and Test ({CSET} 19). [Online]. Available: https://www.usenix.org/system/files/cset19-paper_g.pdf

[30] C. Cimpanu, (2018), *FBI takes control of APT28's VPNFilter botnet*. [Online]. Available: https://www.bleepingcomputer.com/news/security/fbi-takes-control-of-apt28s-vpnfilter-botnet

[31] D. Uhrıcek, (2020), *LiSa–Multiplatform Linux Sandbox for Analyzing IoT Malware*. [Online]. Available: http://excel.fit.vutbr.cz/submissions/2019/058/58.pdf

[32] *Forecast end-user spending on IoT solutions worldwide from 2017 to 2025*. [Online]. Available: https://www.statista.com/statistics/976313/global-iot-market-size. Truy cập ngày 25/07/2020.

[33] MalwareMustDie, (2016), *MMD-0057-2016 - Linux/LuaBot - IoT botnet as service*, MalwareMustDie Blog. [Online]. Available: https://blog.malwaremustdie.org/2016/09/mmd-0057-2016-new-elf-botnet-linuxluabot.html

[34] Peter Stančík, (2016), *ESET Analysis: At Least 15% of Home Routers Unsecured*. [Online]. Available: https://www.eset.com/int/about/newsroom/press-releases/products/eset-analysis-at-least-15-of-home-routers-unsecured-1

[35] Radaware, (2017), *BrickerBot results in PDoS attack*. [Online]. Available: https://www.radware.com/security/ddos-threats-attacks/brickerbot-pdos-permanent-denial-of-service

[36] S. Edwards, I. Profetis, (2016), *Hajime: Analysis of a Decentralized Internet Worm for IoT Devices*, Rapidity Networks. [Online]. Available: http://security.rapiditynetworks.com/publications/2016-10-16/hajime.pdf

[37] https://github.com/horsicq/Detect-It-Easy. Truy cập ngày 17/12/2020.

[38] https://github.com/upx . Truy cập ngày 17/12/2020

[39] https://virusshare.com. Truy cập ngày 03/02/2021.

[40] https://github.com/ReFirmLabs/binwalk . Truy cập ngày 03/02/2021.

[41] https://openwrt.org/ . Truy cập ngày 03/02/2021.

[42] https://www.virustotal.com . Truy cập ngày 05/02/2021.

PHỤ LỤC

1. Module xây dựng đồ thị PSI:

```
import os
import sys
import json
import networkx as nx
import pandas as pd
from tqdm import tqdm
from joblib import Parallel, delayed
import multiprocessing

def get_node_list():
    vertices = set()
    mal_list = pd.read_csv('input/list_malware.csv').values.flatten()
    beg_list = pd.read_csv('input/list_benign.csv').values.flatten()
    mal = [name[:name.find('_')] for name in mal_list]
    beg = [name[:name.find('_')] for name in beg_list]

    for folder in ['bashlite/', 'mirai/', 'others/', 'benign/']:
        path = 'input/psi_graph/' + folder
        for _, _, files in os.walk(path):
            for file in tqdm(files, desc=folder[:-1]):
                if file.replace('.txt', '') in mal or
file.replace('.txt', '') in beg:
                    fpath = path + file
                    with open(fpath, 'r') as f:
                        data = f.read().split('\n')
                    for line in data[2:-1]:
                        e = line.split()
                        if len(e) == 2:
                            vertices.add(e[0])
                            vertices.add(e[1])
    with open('temp/list_node_psi.txt', 'w') as f:
        f.writelines('\n'.join(list(vertices)))


def psig():
    mal_list = pd.read_csv('input/list_malware.csv').values.flatten()
    beg_list = pd.read_csv('input/list_benign.csv').values.flatten()
    mal = [name[:name.find('_')] for name in mal_list]
    beg = [name[:name.find('_')] for name in beg_list]
    with open('temp/list_node_psi.txt') as f:
        vertices = f.read().split('\n')

    def run(folder):
        path = 'input/psi_graph/' + folder
        for _, _, files in os.walk(path):
            for file in tqdm(files, desc=folder[:-1]):
                if file.replace('.txt', '') in mal or
file.replace('.txt', '') in beg:
                    G = {'edges': list()}
                    fpath = path + file
                    with open(fpath, 'r') as f:
                        data = f.read().split('\n')
                    for line in data[2:-1]:
```

```
                        e = line.split()
                        if len(e) == 2:
                            G['edges'].append(
                                [vertices.index(e[0]),
vertices.index(e[1])])
                    with open('input/psig/' + file.replace('.txt',
'.json'), 'w') as f:
                        json.dump(G, f)

    num_cores = multiprocessing.cpu_count()
    folders = ['bashlite/', 'mirai/', 'others/', 'benign/']
    results = Parallel(n_jobs=num_cores)(delayed(run)(i) for i in
folders)


if __name__ == "__main__":
    # get_node_list()
    psig()
```

2. Module xây dựng đồ thị SCG

```
import os
import sys
import json
import networkx as nx
import pandas as pd
from tqdm import tqdm

def get_node_list():
    vertices = set()
    for label in ['malware', 'benign']:
        path = 'input/list_' + label + '.csv'
        file_list = pd.read_csv(path).values.flatten()
        root = '/media/ais/data/final_report_' + label + '/'

        for folder in tqdm(file_list, desc='Get nodes of ' + label):
            rp_path = root + folder
            for _, _, files in os.walk(rp_path):
                for file_name in files:
                    if file_name.startswith('strace'):
                        file_path = rp_path + '/' + file_name
                        with open(file_path, 'r') as f:
                            data = json.load(f)
                        for syscall in data:
                            vertices.add(syscall['name'])
    with open('temp/list_node.txt', 'w') as f:
        f.writelines('\n'.join(list(vertices)))


def scg():
    with open('temp/list_node.txt') as f:
        vertices = f.read().split('\n')

    for label in ['malware', 'benign']:
        path = 'input/list_' + label + '.csv'
        file_list = pd.read_csv(path).values.flatten()
        root = '/media/ais/data/final_report_' + label + '/'
```

```
        for folder in tqdm(file_list, desc='Generate SCGs of ' +
label):
            rp_path = root + folder
            G = {'edges': list()}
            for _, _, files in os.walk(rp_path):
                for file_name in files:
                    if file_name.startswith('strace'):
                        file_path = rp_path + '/' + file_name
                        with open(file_path, 'r') as f:
                            data = json.load(f)

                        u = -1
                        for syscall in data:
                            if syscall['name'] != 'fork':
                                v = vertices.index(syscall['name'])
                                if u >= 0 and u != v and [u, v] not in
G['edges']:
                                    G['edges'].append([u, v])
                                u = v
                            else:
                                u = -1
            with open('input/scg/' + folder + '.json', 'w') as f:
                json.dump(G, f)


if __name__ == "__main__":
    # get_node_list()
    scg()
```

3. Module graph2vec

```
"""Graph2Vec module."""

import json
import glob
import hashlib
import pandas as pd
import networkx as nx
from tqdm import tqdm
from joblib import Parallel, delayed
from param_parser import parameter_parser
from gensim.models.doc2vec import Doc2Vec, TaggedDocument

class WeisfeilerLehmanMachine:
    """
    Weisfeiler Lehman feature extractor class.
    """
    def __init__(self, graph, features, iterations):
        """
        Initialization method which also executes feature extraction.
        :param graph: The Nx graph object.
        :param features: Feature hash table.
        :param iterations: Number of WL iterations.
        """
        self.iterations = iterations
        self.graph = graph
```

```
        self.features = features
        self.nodes = self.graph.nodes()
        self.extracted_features = [str(v) for k, v in features.items()]
        self.do_recursions()

    def do_a_recursion(self):
        """
        The method does a single WL recursion.
        :return new_features: The hash table with extracted WL
features.
        """
        new_features = {}
        for node in self.nodes:
            nebs = self.graph.neighbors(node)
            degs = [self.features[neb] for neb in nebs]
            features = [str(self.features[node])]+sorted([str(deg) for
deg in degs])
            features = "_".join(features)
            hash_object = hashlib.md5(features.encode())
            hashing = hash_object.hexdigest()
            new_features[node] = hashing
        self.extracted_features = self.extracted_features +
list(new_features.values())
        return new_features

    def do_recursions(self):
        """
        The method does a series of WL recursions.
        """
        for _ in range(self.iterations):
            self.features = self.do_a_recursion()

def dataset_reader(path):
    """
    Function to read the graph and features from a json file.
    :param path: The path to the graph json.
    :return graph: The graph object.
    :return features: Features hash table.
    :return name: Name of the graph.
    """
    name = path.replace(".json", "").split("/")[-1]
    data = json.load(open(path))
    graph = nx.from_edgelist(data["edges"])

    if "features" in data.keys():
        features = data["features"]
    else:
        features = nx.degree(graph)

    features = {int(k): v for k, v in features.items()}
    return graph, features, name

def feature_extractor(path, rounds):
    """
    Function to extract WL features from a graph.
    :param path: The path to the graph json.
    :param rounds: Number of WL iterations.
    :return doc: Document collection object.
```

```
    """
    graph, features, name = dataset_reader(path)
    machine = WeisfeilerLehmanMachine(graph, features, rounds)
    doc = TaggedDocument(words=machine.extracted_features, tags=["g_" +
name])
    return doc

def save_embedding(output_path, model, files, dimensions):
    """
    Function to save the embedding.
    :param output_path: Path to the embedding csv.
    :param model: The embedding model object.
    :param files: The list of files.
    :param dimensions: The embedding dimension parameter.
    """
    out = []
    beg_list = pd.read_csv('input/list_benign.csv').values.flatten()
    beg = [name[:name.find('_')] for name in beg_list]
    for f in files:
        identifier = f.split("/")[-1].replace(".json", "")
        # identifier = identifier[:identifier.find('_')]
        label = 0 if identifier in beg else 1
        out.append([identifier] + list(model.docvecs["g_"+identifier])
+ [label])
    column_names = ["name"]+["x_"+str(dim) for dim in
range(dimensions)] + ["label"]
    out = pd.DataFrame(out, columns=column_names)
    out = out.sort_values(["name"])
    out.to_csv(output_path, index=None)

def main(args):
    """
    Main function to read the graph list, extract features.
    Learn the embedding and save it.
    :param args: Object with the arguments.
    """
    graphs = glob.glob(args.input_path + "*.json")
    print("\nFeature extraction started.\n")
    document_collections =
Parallel(n_jobs=args.workers)(delayed(feature_extractor)(g,
args.wl_iterations) for g in tqdm(graphs))
    print("\nOptimization started.\n")

    model = Doc2Vec(document_collections,
                    vector_size=args.dimensions,
                    window=0,
                    min_count=args.min_count,
                    dm=0,
                    sample=args.down_sampling,
                    workers=args.workers,
                    epochs=args.epochs,
                    alpha=args.learning_rate)

    save_embedding(args.output_path, model, graphs, args.dimensions)

if __name__ == "__main__":
    args = parameter_parser()
    main(args)
```

4. Module kết hợp vector đặc trưng động và tĩnh

```
import pandas as pd

psi = pd.read_csv('output/psig.csv')
scg = pd.read_csv('output/scg.csv')
fusion = list()
for row in psi.values:
    name = row[0]
    scg_row = scg.loc[scg['name']==name, 'x_0':].values.flatten()
    fusion_row = list(row[:-1])
    fusion_row.extend(scg_row)
    fusion.append(fusion_row)
header = ['name'] + [f'x_{i}' for i in range(256)] + ['label']
pd.DataFrame(fusion, columns=header).to_csv('output/fusion.csv',
index=None)
```

5. Module thiết lập tham số cho các bộ phân loại

```
"""Parameter parser to set the model hyperparameters."""

import argparse

def parameter_parser():
    """
    A method to parse up command line parameters.
    By default it gives an embedding of the partial NCI1 graph dataset.
    The default hyperparameters give a good quality representation
without grid search.
    Representations are sorted by ID.
    """
    parser = argparse.ArgumentParser(description="Run Graph2Vec.")

    parser.add_argument("--input-path",
                        nargs="?",
                        default="./dataset/",
                    help="Input folder with jsons.")

    parser.add_argument("--output-path",
                        nargs="?",
                        default="./features/nci1.csv",
                    help="Embeddings path.")

    parser.add_argument("--dimensions",
                        type=int,
                        default=128,
                    help="Number of dimensions. Default is 128.")

    parser.add_argument("--workers",
                        type=int,
                        default=4,
                    help="Number of workers. Default is 4.")

    parser.add_argument("--epochs",
                        type=int,
```

```
                        default=10,
                    help="Number of epochs. Default is 10.")

    parser.add_argument("--min-count",
                        type=int,
                        default=5,
                    help="Minimal structural feature count. Default is
5.")

    parser.add_argument("--wl-iterations",
                        type=int,
                        default=2,
                    help="Number of Weisfeiler-Lehman iterations.
Default is 2.")

    parser.add_argument("--learning-rate",
                        type=float,
                        default=0.025,
                    help="Initial learning rate. Default is 0.025.")

    parser.add_argument("--down-sampling",
                        type=float,
                        default=0.0001,
                    help="Down sampling rate of features. Default is
0.0001.")

    return parser.parse_args()
```

6. Module phân loại huấn luyện và đánh giá

```
from __future__ import print_function
import os
import argparse
import pickle
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from time import time
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.feature_selection import SelectFromModel
from sklearn.preprocessing import StandardScaler
from sklearn import metrics
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import LinearSVC, SVC


def param_parser():
    parser = argparse.ArgumentParser(
        description='Machine Learning based Classifiers')
    parser.add_argument('--input-path',
                        nargs='?',
                        default='./output/scg.csv',
                        help='Input folder with CSV.')
    parser.add_argument('--output-path',
```

```
                        nargs='?',
                        default='./result/',
                        help='Result path.')
    parser.add_argument('--seed',
                        type=int,
                        default=2020,
                        help='Random seed. Default is 2020.')
    args = parser.parse_args()
    return args


def report(args, names, y_true, y_pred):
    with open(f'{args.output_path}/result.txt', 'w') as f:
        for name in names:
            clf_report = metrics.classification_report(
                y_true[name], y_pred[name], digits=4)

            cnf_matrix = metrics.confusion_matrix(y_true[name],
y_pred[name])
            TN, FP, FN, TP = cnf_matrix.ravel()
            TPR = TP / (TP + FN)
            FPR = FP / (FP + TN)

            f.write(str(name) + ':\n')
            f.write('Accuracy: %0.4f\n' %
                    metrics.accuracy_score(y_true[name], y_pred[name]))
            f.write('ROC AUC: %0.4f\n' %
                    metrics.roc_auc_score(y_true[name], y_pred[name]))
            f.write('TPR: %0.4f\nFPR: %0.4f\n' % (TPR, FPR))
            f.write('Classification report:\n' + str(clf_report) +
'\n')
            f.write('Confusion matrix:\n' + str(cnf_matrix) + '\n')
            f.write(64*'-'+'\n')


def draw_roc(args, names, colors, y_true, y_pred):
    plt.figure()
    for name, color in zip(names, colors):
        fpr, tpr, _ = metrics.roc_curve(y_true[name], y_pred[name])
        auc = metrics.roc_auc_score(y_true[name], y_pred[name])
        plt.plot(fpr, tpr, color=color,
                 label='%s ROC (area = %0.3f)' % (name, auc))
    plt.plot([0, 1], [0, 1], 'r--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('1-Specificity(False Positive Rate)')
    plt.ylabel('Sensitivity(True Positive Rate)')
    plt.title('Receiver Operating Characteristic')
    plt.legend(loc="lower right")
    plt.savefig(f'{args.output_path}/roc.png', dpi=300)


def load_data(args):
    data = pd.read_csv(args.input_path)
    X = data.values[:, 1:-1]
    y = data.values[:, -1].astype('int')

    X_train, X_test, y_train, y_test = train_test_split(
```

```
        X, y, random_state=args.seed, test_size=0.3, stratify=y)

    fs = SelectFromModel(
        LinearSVC(penalty="l1", dual=False,
random_state=args.seed).fit(X_train, y_train), prefit=True)
    X = fs.transform(X)
    pickle.dump(fs, open(f'{args.output_path}model/fs.sav', 'wb'))

    scaler = StandardScaler()
    X_train = scaler.fit_transform(X_train)
    X_test = scaler.transform(X_test)
    pickle.dump(scaler, open(f'{args.output_path}model/scaler.sav',
'wb'))

    return X_train, y_train, X_test, y_test


def main(args):
    y_true = dict()
    y_pred = dict()

    names = ['Naive Bayes', 'Decision Tree',
             'k-Nearest Neighbors', 'SVM', 'Random Forest']
    fnames = ['NB', 'DT', 'kNN',  'SVM', 'RF']

    classifiers = [
        GaussianNB(),
        DecisionTreeClassifier(random_state=args.seed,
                               class_weight='balanced'),
        KNeighborsClassifier(n_jobs=-1),
        SVC(random_state=args.seed, class_weight='balanced'),
        RandomForestClassifier(random_state=args.seed,
                               class_weight='balanced', n_jobs=-1),
    ]

    hyperparam = [
        {},
        {'criterion': ['gini', 'entropy']},
        {'n_neighbors': [5, 100, 500], 'weights': ['uniform',
'distance']},
        {'C': np.logspace(-3, 3, 7), 'gamma': np.logspace(-3, 3, 7)},
        {'criterion': ['gini', 'entropy'], 'n_estimators': [
            10, 100, 1000], 'bootstrap': [True, False]},
    ]

    colors = ['blue', 'orange', 'green', 'red',
              'purple', 'brown', 'pink', 'gray']

    print('Preprocessing data...', end=' ', flush=True)
    t = time()
    X_train, y_train, X_test, y_test = load_data(args)
    print('Done in %0.2f' % (time()-t))

    for name, fname, est, hyper in zip(names, fnames, classifiers,
hyperparam):
        print(f'{name}...')
        clf = GridSearchCV(est, hyper, cv=5, n_jobs=-1)
```

```
            t = time()
            clf.fit(X_train, y_train)
            print('\tTraining done in %0.2f' % (time()-t))

            t = time()
            y_true[name],  y_pred[name] = y_test, clf.predict(X_test)
            print('\tTesting done in %0.2f' % (time()-t))

            acc = 100 * metrics.accuracy_score(y_true[name], y_pred[name])
            print('\tAccuracy: %0.2f' % acc)

            pickle.dump(clf, open(f'{args.output_path}model/{fname}.sav',
'wb'))

    report(args, names, y_true, y_pred)
    draw_roc(args, names, colors, y_true, y_pred)


if __name__ == '__main__':
    args = param_parser()
    os.mkdir(f'{args.output_path}/model/')
    main(args)
```