HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

NGUYỄN VIỆT DŨNG

PHÁT HIỆN SỚM MÃ ĐỘC IOT BOTNET TRÊN CÁC THIẾT BỊ IOT

LUẬN VĂN THẠC SỸ KỸ THUẬT

(Theo định hướng ứng dụng)

HÀ NỘI – 2021

HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

NGUYỄN VIỆT DŨNG

PHÁT HIỆN SỚM MÃ ĐỘC IOT BOTNET TRÊN CÁC THIẾT BỊ IOT

CHUYÊN NGÀNH: HỆ THỐNG THÔNG TIN

MÃ SỐ: 8.48.01.04

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

NGƯỜI HƯỚNG DẪN KHOA HỌC: PGS TS PHẠM VĂN CƯỜNG

HÀ NỘI – 2021

BẢN CAM ĐOAN

Tôi cam đoan đã thực hiện việc kiểm tra mức độ tương đồng nội dung luận văn qua phần mềm DoIT một cách trung thực và đạt kết quả mức độ tương đồng 5% toàn bộ nội dung luận văn. Bản luận văn kiểm tra qua phần mềm là bản cứng luận văn đã nộp để bảo vệ trước hội đồng. Nếu sai tôi xin chịu các hình thức kỷ luật theo quy định hiện hành của Học viện.

Hà Nội, ngày tháng năm 2021

HỌC VIÊN CAO HỌC

(Ký và ghi rõ họ tên)

Nguyễn Việt Dũng

LỜI CẢM ƠN

Để hoàn thành được quyển luận văn thạc sĩ này, đầu tiên, em xin gửi lời cảm ơn sâu sắc nhất đến thầy Phó Giáo sư, Tiến sỹ Phạm Văn Cường, người đã định hướng, trực tiếp dẫn dắt và hướng dẫn cho em trong suốt thời gian thực hiện đề tài nghiên cứu khoa học. Đồng thời, em cũng cảm ơn các thầy cô Khoa Sau đại học, Khoa Công nghệ Thông tin – Học viện Công nghệ Bưu chính Viễn thông đã giúp đỡ, tạo mọi điều kiện thuận lợi cho em trong quá trình học tập tại Học viện cũng như quá trình làm luận văn thạc sỹ.

Trong nội dung luận văn không thể tránh khỏi những hạn chế và thiếu sót, em mong muốn sẽ nhận được nhiều đóng góp quý báu đến từ các quý thầy cô, bạn bè, đồng nghiệp để nội dung nghiên cứu được hoàn thiện hơn nữa và có ý nghĩa thiết thực áp dụng trong thực tế.

Sau cùng, em xin gửi lời cảm ơn đến gia đình, người thân và bạn bè đã luôn bên cạnh ủng hộ, động viên em trong cuộc sống cũng như trong thời gian hoàn thành luận văn thạc sĩ.

Xin chân thành cảm ơn tất cả mọi người!

Hà Nội, ngày tháng năm 2021

Người thực hiện

Nguyễn Việt Dũng

MỤC LỤC

DANH MỤC TỪ VIẾT TẮT

Từ viết tắt	Viết đầy đủ (Tiếng Anh)	Viết đầy đủ (Tiếng Việt)
C&C	Command and Control server	Máy chủ ra lệnh và điều khiển
CFG	Control Flow Graph	Đồ thị luồng điều khiển
CPU	Central Processing Unit	Bộ xử lý trung tâm
DDoS	Distributed Denial of Service attack	Tấn công từ chối dịch vụ phân tán
DL	Deep Learning	Học sâu
DNN	Deep Neural Networ	Mạng nơ-ron học sâu
DNS	Domain Name System	Hệ thống tên miền
DT	Decision Tree	Thuật toán cây quyết định
ELF	Linux Executable and Linkable Format	Định dạng tệp tin thực thi và liên kết động trong Linux
IoT	Internet of Things	Vạn vật kết nối Internet
ITU	International Telecommunication Union	Cơ quan chuyên trách về công nghệ thông tin và truyền thông của Liên hiệp quốc
KNN	K-nearest neighbors	Thuật toán K điểm gần nhất
ML	Machine Learning	Học máy
P2P	Peer to Peer network	Mạng ngang hàng
PSI	Printable String Information	Thông tin có ý nghĩa
RF	Random Forest	Thuật toán rừng cây ngẫu nhiên
RNN	Recurrent Neural Network	Mạng nơ-ron hồi quy
SVM	Support Vector Machine	Thuật toán máy hỗ trợ vector

DANH MỤC HÌNH ẢNH

DANH MỤC BẢNG BIỂU

MỞ ĐẦU

Với sự phát triển nhanh chóng của các thiết bị IoT trên thế giới cả về số lượng lẫn chức năng, môi trường hoạt động, các loại mã độc IoT Botnet cũng được tiến hoá mạnh mẽ và khó bị phát hiện và phân tích hơn giúp duy trì hoạt động lây nhiễm, tấn công mạng. Nhiều loại mã độc IoT Botnet gần đây đã được thiết kế để tránh bị phát hiện bởi các giải pháp bảo mật truyền thống hiện có hạn như phần mềm phát hiện và xử lý mã độc (Anti-virus), hệ thống phát hiện và xử lý xâm nhập mạng (IDS/IPS), các bộ lọc gói tin bằng tường lửa thông thường (Firewall). Các giải pháp bảo mật truyền thống này chỉ thực sự được phát hiện được mạng lưới IoT Botnet khi chúng đã ở trong giai đoạn thực thi tấn công từ chối dịch vụ phân tán (DDoS) và gây ra hậu quả được thấy rõ như gần 400.000 thiết bị IoT bị lây nhiễm mã độc Mirai và vụ tấn công DDoS đã được ghi nhận với quy mô lớn nhất được thực hiện bởi mã độc Mirai có lưu lượng lên đến 1.2 Tbps vào năm 2016.

Về cơ bản, các phương pháp phát hiện mã độc đều dựa trên hai phương pháp chính là phân tích tĩnh và động. Tuy nhiên, hạn chế lớn của phân tích tĩnh là khó có thể phát hiện sớm được mã độc IoT botnet, chính vì vậy luận văn lựa chọn sử dụng phương pháp phân tích động để có thể phát hiện sớm mã độc IoT Botnet. Do đó, luận văn thực hiện đề tài ***"Phát hiện sớm mã độc IoT botnet trên các thiết bị IoT"*** nhằm tập trung tìm hiểu, ứng dụng và thực nghiệm phương pháp hiệu quả trong phát hiện sớm mã độc IoT botnet, góp phần đảm bảo an ninh, an toàn hệ thống mạng nói chung và hệ thống mạng thiết bị IoT nói riêng.

CHƯƠNG 1: TỔNG QUAN VỀ PHÁT HIỆN SỚM MÃ ĐỘC TRÊN CÁC THIẾT BỊ IOT

1.1. Tổng quan về các thiết bị IoT và IoT Botnet

1.1.1. Tổng quan về thiết bị IoT

1.1.1.1. Khái niệm thiết bị IoT

Cụm từ IoT (Internet of Things - Vạn vật kết nối Internet) lần đầu được sử dụng bởi Kevin Ashton - nhà khoa học đã sáng lập ra Trung tâm Auto-ID tại Viện công nghệ Massachusets (MIT - Massachusetts Institute of Technology) vào năm 1999. Theo định nghĩa của Kevin Ashton [1], "*Internet of Things*" là "*tập hợp các thiết bị cảm biến và bộ điều khiển nhúng được kết nối thông qua môi trường mạng (có dây và không dây)*". Với định nghĩa của Kevin Ashton, thuật ngữ thiết bị "*IoT*" được sử dụng để chỉ các thiết bị cảm biến và bộ điều khiển nhúng điện tử.

Cũng theo cách tiếp cận này, từ điển Oxford có nêu "*Internet of Things (danh từ): là sự kết nối thông qua Internet của các thiết bị điện toán nhúng trong các đối tượng hàng ngày cho phép chúng có thể gửi và nhận dữ liệu*". Tổ chức IREC (European Research Cluster on the IoT) cũng đã đưa ra khái niệm về IoT như sau "*IoT là một kiến trúc toàn cầu động, có khả năng tự cấu hình dựa trên giao thức truyền thông tương tác tiêu chuẩn, ở đó các đồ vật (gồm cả vật lý và ảo) có khả năng định danh và các tính chất vật lý và ảo hóa, có giao diện thông minh và kết hợp khéo léo với nhau để hòa vào hệ thống thông tin mạng".*

Liên minh Viễn thông thế giới (ITU – International Telecommunication Union) [2] cũng đã đưa ra khái niêm về "IoT", khái niệm này đã góp phần giúp làm sáng tỏ hơn về IoT. Theo ITU thì: "*Internet of Things là một cơ sở hạ tầng toàn cầu trong xã hội thông tin, nó cho phép các dịch vụ thông minh hoạt động bằng cách kết nối các vật thể bao gồm cả vật lý và ảo dựa trên các công nghệ thông tin truyền thông phù hợp hiện có và đang phát triển"* [3]. Với khái niệm được nêu ở trên, IoT có thể được nhìn nhận trong một viễn cảnh rộng như là một tầm nhìn với những

hàm ý về công nghệ và xã hội. Thông qua việc khai thác nhận dạng, thu thập, xử lý dữ liệu và khả năng truyền thông, IoT tận dụng mọi thứ để hỗ trợ cho các loại ứng dụng, trong khi duy trì sự riêng tư cần thiết.

Theo khái niệm của ITU, vạn vật (Things) là đối tượng của thế giới thực (vật chất tồn tại) hoặc của thế giới thông tin (thực thể ảo), có thể được xác định và tích hợp vào mạng thông tin, truyền thông hiện có và đang phát triển. Vạn vật (Things) được định nghĩa phân thành hai loại chính:

- Physical things: Vật thể tồn tại trong thế giới vật chất và có khả năng cảm nhận, hoạt động tương tác trở lại môi trường và kết nối với các thực thể khác. Các thiết bị đại diện cho loại này có thể kể đến như: cảm biến nhiệt độ, các rô-bốt công nghiệp, phần cứng thiết bị nhúng gia dụng.

- Virtual things: Những thực thể ảo (không cảm nhận vật lý được) tồn tại trong thế giới thông tin và có khả năng lưu trữ, xử lý và truy cập dữ liệu. Một số ví dụ về Virtual things có thể kể đến như: nội dung đa phương tiện, phần mềm ứng dụng và các đại diện dịch vụ của các vật thể vật lý (tài khoản ảo).

Theo đó, ITU cũng đưa ra khái niệm thiết bị IoT là "*các thiết bị có khả năng kết nối và có thể cảm nhận thay đổi của môi trường, tương tác qua cơ cấu truyền động, thu thập, lưu trữ và xử lý dữ liệu*". Từ khái niệm thiết bị IoT này cho thấy rằng trong môi trường IoT, có rất nhiều loại thiết bị IoT khác nhau như: điện thoại di động thông minh, máy tính cá nhân, đồng hồ thông minh, smart TV, máy in, máy quét, IP Camera, thiết bị định tuyến, thiết bị gia dụng thông minh có kết nối Internet,... Các thiết bị IoT có mặt ở mọi nơi, hầu hết các ngành nghề, các mặt của đời sống con người như y tế, quản lý dây uyền sản xuất, quản lý năng lượng, hệ thống giao thông thông minh… Ngoài các tiện ích đem lại và sự có mặt trong nhiều mặt của cuộc sống, ngành nghề thì các thiết bị IoT cũng được dự báo sẽ đóng góp lớn vào nền kinh tế toàn cầu. Theo báo cáo của công ty IoT Analytics [4] (nhà cung cấp hàng đầu về tìm hiểu thị trường cho IoT có trụ sở ở Đức) thì giá trị kinh tế toàn cầu do IoT mang lại sẽ từ 2.700 tỷ USD cho đến 6.700 tỷ USD trước năm 2025.

1.1.1.2. Phân loại thiết bị IoT

Hiện nay đã có nhiều cách giúp phân loại các thiết bị IoT được đưa ra, dựa trên nhiều tiêu chí khác nhau như chức năng, nhiệm vụ, vị trí trong lớp ứng dụng,... Tuy nhiên, trong luận văn này sử dụng tiêu chí tài đặc điểm tài nguyên phần cứng được cung cấp cho thiết bị IoT để phân loại theo đề xuất của Bencheton [5]. Theo đó, các thiết bị IoT được phân chia thành thiết bị có phần cứng hiệu năng cao (high-capacity resource) và phần cứng hạn chế tài nguyên (constrained resource). Các thiết bị có phần cứng hiệu năng cao được trang bị tài nguyên phần cứng có năng lực xử lý, lưu trữ, truyền tải dữ liệu lớn. Các thiết bị này thường được biết đến như máy tính cá nhân, máy chủ, laptop, máy tính bảng, điện thoại thông minh,... Đối lập với những thiết bị được trang bị phần cứng hiệu năng cao là các thiết bị có phần cứng hạn chế tài nguyên bao gồm năng lực xử lý, lưu trữ, truyền tải dữ liệu và năng lượng dữ trữ,... Các thiết bị này thường được biết đến như cảm biến môi trường, thiết bị gia dụng thông minh, thiết bị mạng dân dụng,...

Các thiết bị IoT có phần cứng hạn chế tài nguyên được chia thành ba lớp dựa trên dung lượng bộ nhớ truy xuất ngẫu nhiên (RAM), dung lượng bộ nhớ lưu trữ (Flash, ROM) và khả năng xử lý của CPU (được minh họa trong bảng 1.1). Đặc trưng của từng lớp thiết bị IoT có phần cứng hạn chế tài nguyên này có tác động đến khả năng bảo mật của nó và do đó, nó đưa ra các giới hạn cho việc áp dụng một số giải pháp bảo mật.

Bảng 1.1. Phân loại thiết bị IoT và khả năng tích hợp giải pháp bảo mật [5]

Loại thiết bị	**Lớp**	**Dung lượng RAM**	**Dung lượng bộ nhớ lưu trữ**	**Thiết bị trong thực tế**	**Khả năng tích hợp giải pháp bảo mật**
Thiết bị hạn chế tài	Lớp 0	<< 10KB	<< 100 KB	Cảm biến môi trường thông thường (low-end	Hầu như không thể tích hợp giải pháp bảo mật

nguyên (Cons-trained device)				sensors)	
	Lớp 1	~ 10KB	~ 100KB	Đèn thông minh, Khóa cửa thông minh, …	Có thể sử dụng các giao thức chuyên dụng được thiết kế cho các nút bị ràng buộc (như The Constrained Application Protocol - CoAP) nhưng chúng không thể sử dụng các giao thức bảo mật tiêu chuẩn mạnh hơn.
	Lớp 2	~ 50KB	~ 250 KB	Thiết bị thông minh (smart appliances), cảm biến thông minh cao cấp (high-end smart sensors) như bộ điều khiển nhiệt độ phòng thông minh, robot hút bụi, điều hòa thông minh, máy in điều khiển qua Internet, thiết bị mạng dân dụng (router, switch, repeater SOHO), …	Có khả năng thực hiện các giao thức bảo mật tiêu chuẩn cơ bản, tuy nhiên có khả năng bị hạn chế hiệu quả thực hiện, ví dụ như hạn chế băng thông mạng, hiệu quả thực thi mệnh lệnh của thiết bị,…)
Thiết bị hiệu năng cao (High-		>> 50KB	>> 250KB	TV thông minh, điện thoại thông minh, máy tính bảng, máy tính có	Có thể tích hợp các giải pháp bảo mật chuyên dụng, nâng cao cả về phần cứng và phần mềm.

capacity device)				kết nối mạng, trung tâm điều khiển nhà thông minh, trung tâm kết nối thiết bị thông minh,…	

Do hạn chế về tài nguyên phần cứng, các thiết bị IoT này khó sử dụng các phương án đảm bảo an toàn thông tin thông thường cho thiết bị phần cứng hiệu năng cao đã được phát triển. Chính vì vậy đã dẫn tới sự thiếu hụt các giải pháp bảo mật cho những thiết bị này và trở thành mục tiêu tấn công ưa thích của những kẻ tấn công trong thời gian gần đây [6]–[8]. Với lý do nêu trên, phạm vi nghiên cứu luận văn lựa chọn thiết bị IoT có phần cứng hạn chế tài nguyên (gọi tắt là thiết bị IoT cỡ nhỏ) để nghiên cứu phát hiện các hành vi của mã độc.

1.1.1.3. Đặc điểm của thiết bị IoT cỡ nhỏ

Để có thể đưa ra giải pháp phát hiện mã độc botnet phù hợp cho thiết bị mà luận văn này hướng tới, luận văn đi sâu làm rõ những đặc điểm của thiết bị IoT cỡ nhỏ bao gồm:

- *Môi trường hoạt động ít chịu sự điều khiển trực tiếp của con người*: Các thiết bị IoT có tính di động và tự hành cao theo các kịch bản hoạt động được cài đặt sẵn, hiếm khi cần sự điều khiển trực tiếp của con người (ví dụ: Thiết bị định tuyến, IP Camera, rô-bốt hút bụi trong gia đình,…). Những thiết bị này có thể truy cập vật lý vào nhiều vị trí và tuỳ phụ thuộc vào điều kiện môi trường và thực hiện các nhiệm vụ được lập trình sẵn một cách tự động. Với khả năng tự hành, thu thập và trao đổi dữ liệu giữa các thiết bị IoT dựa trên cơ sở hạ tầng mạng nên các vấn đề an toàn bảo mật có thể kể đến như mất an toàn thông tin trong giao thức và ứng dụng trên các thiết bị IoT là có thể xảy ra.

- *Tính đa nền tảng phần cứng và phần mềm*: Khác với các thiết bị điện tử truyền thống như máy tính đa phần sử dụng vi xử lý kiến trúc i386 thì các thiết bị

IoT cỡ nhỏ thường sử dụng nhiều loại các kiến trúc vi xử lý tiêu thụ năng lượng thấp như: MIPS, ARM, PowerPC, SPARC, MIPSEL,... Ngoài ra, các nhà sản xuất các thiết bị, cụ thể hơn là các nhà sản xuất đến từ Trung Quốc cũng có những thay đổi riêng trong kiến trúc vi xử lý của riêng mình nhằm giảm giá thành và thực hiện những mục đích khác như tiết kiệm năng lượng, kích thước sản phẩm. Ngoài ra, để tối ưu cho các chức năng phục vụ cho nhu cầu của người sử dụng, các nhà sản xuất cũng tự mình thay đổi các phần sụn (firmware) và phần mềm lớp ứng dụng cho phù hợp với thiết bị của mình. Điều này dẫn tới sự đa dạng trong nền tảng phần mềm của các thiết bị IoT cỡ nhỏ.

- *Tài nguyên phần cứng hạn chế*: Như đã trình bày, các thiết bị IoT cỡ nhỏ thường được trang bị các phần cứng hạn chế tài nguyên nhằm để các thiết bị được bán ra với giá cả thấp, giảm kích thước và giảm sự tiêu hao năng lượng. Điều này vô hình chung đã dẫn đến các thiết bị IoT cỡ nhỏ có đặc điểm chung như dung lượng bộ nhớ ít, năng lực tính toán nhỏ, năng lượng pin dự trữ cho thời gian hoạt động ngắn.

- *Trạng thái động:* Trạng thái của các thiết bị IoT thay đổi linh hoạt, ví dụ như lúc hoạt động và ngủ chờ, lúc kết nối và ngắt kết nối... phụ thuộc vào hoàn cảnh của các thiết bị gồm vị trí, chức năng và tốc độ di chuyển. Hơn nữa, số lượng các thiết bị IoT cũng có thể thay đổi do khả năng di chuyển của vật thể mang kèm thiết bị này.

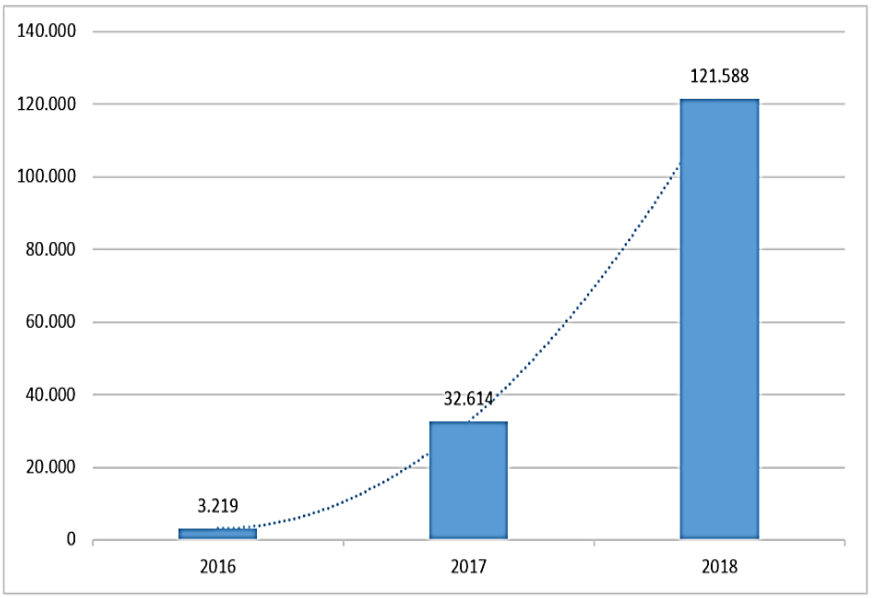
- *Khả năng kết nối đa kênh:* Các thiết bị IoT có khả năng kết nối với các thiết bị và hạ tầng truyền dẫn theo nhiều giao thức khác nhau như Wifi, Bluetooth, Zigbee, Z-wave, LoRa, Lifi,...

1.1.2. Tổng quan về mã độc IoT Botnet

1.1.2.1. Khái niệm mã độc IoT Botnet

Mặc dù có rất nhiều loại mã độc tấn công, lây nhiễm các thiết IoT, nhưng xu hướng mã độc botnet được xem là phổ biến nhất, gây hậu quả lớn nhất đối với các

thiết bị IoT [9]. Các hoạt động của mã độc IoT botnet gần đây cho thấy tội phạm mạng đang chuyển hướng lợi dụng thiết bị IoT để thực hiện các cuộc tấn công mạng với số lượng lớn thiết bị với băng thông cực lớn gây gián đoạn mạng Internet. Hiện nay, số lượng mã độc được ra đời với mục tiêu lây nhiễm, tấn công các thiết bị IoT cỡ nhỏ ngày càng tăng, cụ thể là theo báo cáo của hãng Kaspersky thì số lượng mã độc trên thiết bị IoT năm 2018 đã tăng gấp hơn 37 lần so với năm 2016 [10], minh họa ở hình 1.3.



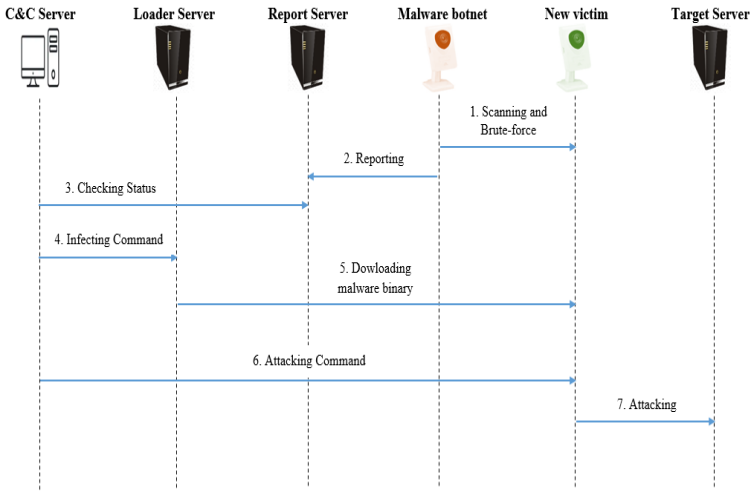
Hình 1.1. Số lượng mã độc botnet trên các thiết bị IoT giai đoạn 2016 – 2018

Trước khi đi vào thuật ngữ mã độc IoT botnet thì cần tìm hiểu thuật ngữ mã độc IoT. Thuật ngữ mã độc có rất nhiều cách hiểu khác nhau, theo Helenius [11], mã độc được là các chương trình được thiết kế với mục đích người dùng không mong muốn. Ed Skoudis và cộng sự thì cho rằng [12] mã độc là tập các câu lệnh được lây nhiễm vào máy tính người dùng để kiểm soát máy tính thực hiện các hành vi độc hại theo mục đích của kẻ tấn công. Một cách tổng quan, mã độc (tiếng anh là Malicious Software/Code) là một chương trình hoặc một đoạn mã được bí mật chèn vào hệ thống người dùng nhằm gây hại hệ thống máy tính, hệ thống mạng, thông tin dữ liệu, … như tính tin cậy, tính bảo mật, tính toàn vẹn hoặc tính sẵn sàng.

Trong các loại mã độc xuất hiện trên thiết bị IoT cỡ nhỏ, phổ biến nhất là mã độc IoT Botnet (theo thống kê của Kaspersky [13]). Tác giả Pamela và cộng sự [14] đã đưa ra khái niệm IoT Botnet là "*một mạng lưới các thiết bị IoT cỡ nhỏ bị xâm nhập và lây nhiễm mã độc phục vụ xây dựng Botnet*". Theo khái niệm này kết hợp với phạm vi nghiên cứu của luận văn, mã độc IoT Botnet có thể hiểu là loại mã độc cho phép xây dựng mạng lưới Botnet dựa trên các thiết bị IoT cỡ nhỏ.

1.1.2.2. Cấu trúc và nguyên lý hoạt động của IoT Botnet

Thông qua việc khảo sát các loại mã độc IoT Botnet đã xuất hiện đến hiện nay, luận văn tổng quát lại mã độc IoT Botnet chứa hầu hết 2 thành phần cơ bản và 4 thành phần hỗ trợ, minh họa ở hình 1.4.



Hình 1.2. Các bước trong vòng đời của mã độc IoT Botnet

Các thành phần gồm: Mã độc botnet thực hiện tấn công DDoS khi nhận lệnh; Máy chủ C&C để điều khiển mã độc botnet; Bộ Scanner để dò quét các thiết bị IoT mới có thể bị khai thác; Máy chủ Reporting có chức năng thu thập các dữ liệu dò quét của mã độc botnet hoặc bộ Scanner; Bộ Loaders được sử dụng để đăng nhập vào các thiết bị IoT có thể bị khai thác, và ra chỉ thị cho chúng tải về các tập tin thực thi mã độc có kiến trúc phù hợp; Máy chủ phân phối mã độc xác định vị trí lưu trữ mà độc sẽ được tải về bởi các thiết bị IoT đã bị lây nhiễm. Trên thực tế một số mạng Botnet có thể sẽ có thêm hoặc giảm bớt một số thành phần cấu trúc, xong các thành phần đó vẫn thực hiện đầy đủ các chức năng được luận văn trình bày bên trên.

Với những thành phần đó, cơ chế hoạt động chung cho hầu hết mã độc IoT botnet như sau [15]:

Bước 1: Mã độc dò quét dải địa chỉ IP ngẫu nhiên thông qua TCP cổng 23/2323 để tìm kiếm các thiết bị IoT có lỗ hổng bảo mật để xâm nhập, lây nhiễm mở rộng mạng lưới máy tính botnet. Sau khi dò quét được thiết bị có khả năng dễ bị xâm nhập, mã độc IoT tiến hành tấn công vét cạn vào thiết bị để leo thang đặc quyền. Việc tấn công vét cạn của mã độc IoT Botnet thường được thực hiện dựa trên tổ hợp các tài khoản, mật khẩu mặc định có sẵn được nhúng trong mã nguồn mã độc. Ví dụ admin/admin, root/root, admin/123…

Bước 2: Sau khi dò quét được thiết bị có khả năng xâm nhập và đã thu thập được thông tin để xác thực và leo thang đặc quyền trên thiết bị (thông qua giao diện dòng lệnh hoặc giao diện đồ họa), thì mã độc sẽ gửi những thông tin đặc trưng của thiết bị về máy chủ Report thông qua các cổng dịch vụ khác (không sử dụng cổng đã dò quét ban đầu), những thông tin được gửi về như địa chỉ IP, giá trị cổng giao thức, kiến trúc thiết bị và tài khoản xác thực.

Bước 3: Mã độc nhận lệnh từ C&C để kiểm tra thông tin đặc tả của thiết bị như địa chỉ IP, kiến trúc phần cứng (MIPS, ARM, PowerPC, …)

Bước 4: Sau khi máy chủ C&C tiếp nhận thông tin đặc tả về thiết bị thì sẽ ra lệnh cho máy chủ Loader lựa chọn tập tin thực thi mã độc phù hợp.

Bước 5: Máy chủ Loader gửi tới thiết bị muốn xâm nhập tập tin mã độc phù hợp. Ngay sau khi tập tin mã độc được tải về và thực thi trên thiết bị thì mã độc sẽ xóa tập tin thực thi và chỉ chạy trong bộ nhớ RAM để tránh bị phát hiện, đồng thời mã độc sẽ tắt các dịch vụ cho phép truy cập từ xa như Telnet, SSH, vô hiệu hóa các chức năng của tường lửa… Thậm chí một số mã độc IoT Botnet còn tìm kiếm các mã độc khác nhằm diệt chúng để tránh bị ảnh hưởng đến tài nguyên thiết bị. Việc khởi động lại thiết bị có thể xóa được mã độc botnet, nhưng việc này chỉ được thực hiện bởi quản trị hệ thống hoặc quản trị mạng. Ví dụ, nếu thiết bị lây nhiễm mã độc là các thiết bị định tuyến thì hệ thống mạng sẽ bị gián đoạn khi các thiết bị định

tuyến khởi động lại. Hơn nữa, hành động này có thể vi phạm chính sách thỏa thuận mức độ phục vụ (SLA – Service Level Agreement) của các dịch vụ có độ sẵn sàng cao.

Bước 6 và 7: Thông qua C&C kẻ tấn công có thể ra lệnh cho mã độc thực hiện các tấn công từ chối dịch vụ phân tán bằng nhiều kỹ thuật như UDP flood, SYN flood, GRE IP flood… tới một mục tiêu cụ thể.

1.1.2.3. Sự tiến hoá của mã độc IoT Botnet

Cũng giống như các loại mã độc truyền thống, sự tiến hóa của mã độc IoT Botnet cũng gia tăng theo sự thay đổi của công nghệ điện toán, các phương thức trao đổi dữ liệu Internet... Sự phức tạp và khả năng phá hoại của mã độc đã tăng lên kể từ khi công nghệ Internet of Things (IoT) ra đời với hàng ngàn, hàng tỷ thiết bị IoT được kết nối tới mạng Internet. Allix [16] đã đưa ra ý kiến cho rằng, hầu như các họ mã độc hiện tại được tạo ra thông qua sao chép mã nguồn hoặc là bản sửa đổi của mã độc đã có kết hợp với sự sáng tạo của người viết mã độc. Ý kiến này được thể hiện lại khá rõ với mã độc IoT bằng việc khảo sát, phân tích, đánh giá và tổng hợp một số nghiên cứu như [15], [8], [17], [18], [19], [20] thì luận văn đưa ra một tóm tắt ngắn gọn về sự phát triển, tiến hóa của mã độc IoT botnet sử dụng trong những cuộc tấn công DDoS được ghi lại gần đây.

Linux.Hydra [18] được coi là một trong những mã độc phát tán trên thiết bị IoT sớm nhất. Lần đầu ra mắt vào năm 2008, với mục đích tấn công vào các thiết bị định tuyến sử dụng vi xử lý MIPS. Mã độc này được thiết kế để lây nhiễm trên các thiết bị IoT và ra lệnh cho các thiết bị này thực hiện tấn công DDoS bằng kỹ thuật SYN Flood dựa trên IRC (Internet Relay Chat).

Psyb0t [19] xuất hiện lần đầu vào năm 2009 với nhiều điểm tương tự với mã độc Linux.Hydra như có mục tiêu là các thiết bị Router, modem DSL chạy trên nền tảng kiến trúc MIPS. Các bot của mạng Psyb0t được điều khiển bởi máy chủ C&C thông qua giao thức IRC để thực hiện tấn công UDP Flood, ICMP Flood. Sau khi

thiết bị bị lây nhiễm, botnet Psy0t sẽ chặn truy cập mạng của thiết bi định tuyến qua một số cổng TCP như 22, 23, 80.

Tsunami/Kaiten [15] [18] là mã độc botnet sử dụng giao thức IRC được tiến hóa từ mã độc Linux.Hydra với các tính năng tấn công tinh vi như HTTP Layer 7 Flood, TCP XMASS, sửa đổi cấu hình DNS Server, ngắt tiến trình, tải và thực thi các tập tin, giả mạo địa chỉ IP của những thiết bị dễ bị tổn thương. Thậm chí biến thể của loại mã độc này có thể chỉnh sửa vị trí lưu trữ các tập tin */etc/init.d/rc.local* nhằm tự động thực thi những tập tin mã độc mỗi khi người dùng đăng nhập hoặc tại thư mục */etc/rc.d/rc.local* để đảm bảo mã độc được chạy cùng hệ thống khi khởi động [6].

Aidra/LightAidra [18] là 2 loại mã độc botnet dựa trên IRC xuất hiện lần đầu vào năm 2012. LightAidra/Aidra tương tự nhau về mã nguồn nên chúng được xét vào cùng họ mã độc. Đặc biệt, LightAidra/Aidra có thể chạy trên nhiều kiến trúc vi xử lý có thể kể đến là: MIPS, MIPSEL ARM, PowerPC, x86/86-64 và SuperH. Mã độc này có thể khởi tạo các cuộc tấn công từ chối dịch vụ sử dụng các kỹ thuật TCP flood, UDP flood, DNS flood, SYN Flood và ACL Flood. Sau khi đã lây nhiễm thành công các thiết bị IoT, mã độc LightAidra/Aidra được tải về thư mục /var/run, /var/tmp trên thiết bị (nếu thiết bị sử dụng kiến trúc vi xử lý x86 thì mã độc sẽ lưu trong thư mục /tmp) [21], đây là là vị trí không lưu trữ các tập tin thực thi trên các hệ thống bình thường bởi dữ liệu sẽ bị xóa khi thiết bị khởi động lại.

Linux.Darlloz là mã độc được hãng Symantec phát hiện vào năm 2013, mã độc này khai thác lỗ hổng PHP có mã CVE-2012-1823. Tương tự như LightAidra, mã độc Linux.Darlloz hỗ trợ nhiều nền tảng kiến trúc như x86, ARM, MIPS, PPC… Nhằm chặn người dùng truy cập tới thiết bị IoT đã lây nhiễm, mã độc này chặn các lưu lượng kết nối bằng telnet với cấu hình iptable và kết thúc tiến trình của dịch vụ telnet trên thiết bị. Ngoài ra, mã độc Linux.Darlloz sẽ tìm cách xóa các tập tin của LightAidra khỏi thiết bị đã lây nhiễm và kết thúc các tiến trình có ID tiến trình

(PID) được lưu trữ trong các tập tin như */var/run/.lightpid*, */var/run/.aidrapid* và */var/run/lightpid*.

Linux.Wifatch là mã độc lây nhiễm trên các thiết bị IoT có khả năng xác thực yếu hoặc sử dụng tài khoản và mật khẩu mặc định. Khi lây nhiễm thành công vào thiết bị, mã độc Wifatch sẽ tìm cách loại bỏ những mã độc khác và ngăn chặn truy cập telnet với thông báo "Telnet đã được đóng nhằm tránh lây nhiễm thiết bị trong tương lai. Vui lòng ngắt telnet, thay đổi mật khẩu kết nối telnet hoặc cập nhật firmware" trong nhật ký của thiết bị. Linux.Wifatch sử dụng mạng ngang hàng (P2P) để cập nhật mã độc và xóa các dấu vết mã độc lưu lại trên thiết bị IoT. Theo phân tích của hãng bảo mật Symantec [22] thì sau khi lây nhiễm lên thiết bị, mã độc Linux.Wifatch có thể thực hiện những hành vi mở cổng hậu hoặc kết nối tới máy chủ C&C thông qua giao thức TCP.

Spike/Dofloo/MrBlack/Wrkatk/Sotdas/AES.DdoS [18] được phát hiện vào giữa năm 2014 có khả năng lây nhiễm trên các thiết bị sử dụng vi xử lý ARM và MIPS. Thay vì sử dụng IRC-based để giao tiếp như các loại botnet trước đây, dòng mã độc này sử dụng Agent-handler để giao tiếp với các bot. Khác với các botnet trước đây, nhằm duy trì sự lây nhiễm khi thiết bị khởi động lại, mã độc này đã sử dụng cách giả mạo tập tin *etc/rc.local*. Hơn nữa, cải tiến nổi bật của mã độc này là sử dụng luồng *SendInfo* để tính toán hiệu năng của thiết bị và gửi về máy chủ C&C, khi đó kẻ tấn công có thể triển khai mật độ thực hiện tấn công từ chối dịch vụ phân tán trên mỗi thiết bị bot hiệu quả với hiệu năng của chúng.

BASHLITE/Gafgyt/Q-Bot/Torlus/Lizkebab/LizardStresser) là loại mã độc xuất hiện vào năm 2014. Loại mã độc này có có đặc tính tương đồng với Spike như sử dụng Agent-Handler, có thể lây nhiễm đa kiến trúc phần cứng, thực hiện tấn công DDoS bằng các phương pháp phổ biến như SYN, UDP, TCP ACK Flood.

Remaiten/KTN-RM là mã độc IoT có những tính năng kết hợp giữa mã độc Tsunami và Bashlite. Remaiten thực hiện lây nhiễm thiết bị IoT chạy trên nền tảng hệ điều hành Linux bằng phương pháp tấn công vét cạn dựa trên danh sách các tài

khoản đăng nhập, mật khẩu mặc định hoặc thương xuyên được sử dụng. Các máy chủ lệnh và điều khiển (C&C) kết nối, và ra lệnh cho các bot bằng cách sử dụng kênh truyền tin IRC. Mã độc Remaiten cải tiến hơn so với hai mã độc Tsunami và Bashlite ở đặc điểm có thể tùy biến dựa trên kiến trúc thiết bị và phương thức tấn công mà mã độc thực hiện.

Mirai là loại mã độc botnet được viết bằng ngôn ngữ C (5732 dòng mã) và Google Go (1197 dòng mã). Mã độc Mirai sử dụng những kỹ thuật tấn công DDoS đơn giản như TCP SYN Flood, UDP Flood, DNS… Tuy nhiên, hậu quả tấn công của mạng botnet Mirai đem lại vô cùng lớn khi có thời điểm có tới 400.000 thiết bị IoT bị lây nhiễm mã độc Mirai và vụ tấn công đã được ghi nhận với quy mô lớn nhất được thực hiện bởi mã độc Mirai có lưu lượng lên đến 1.2 Tbps [18] vào năm 2016. Mã nguồn của botnet Mirai được phát triển để chạy cho nhiều loại vi xử lý trên hệ điều hành Linux vì đây là nền tảng phổ biến trên các thiết bị IoT và hệ thống nhúng.

Linux.IRCTelnet là mã độc IoT được phát triển bằng cách kết hợp mã nguồn của LightAidra/Aidra, giao thức IRC của mã độc Tsunami, các kỹ thuật lây nhiễm của mã độc BASHLITE và danh sách xác thực của mã độc Mirai. Mã độc được thiết kế để dễ dàng lây nhiễm cho các thiết bị IoT với các kiến trúc vi xử lý phổ biến. Mã độc Linux.IRCTelnet sử dụng nhiều kỹ thuật tấn công như TCP Flood, TCP XMAS, UDP Flood trong cả IPv4 và IPv6.

Hajime là một mã độc IoT botnet được khám phá vào 10/2016 bởi công ty mạng Rapidity [23]. Hajime sử dụng phương thức lây nhiễm tương tự mã độc Mirai. Tuy nhiên, kiến trúc của mã độc Hajime sử dụng hoàn toàn dựa trên mô hình giao tiếp phân tán và sử dụng giao thức bảng mã băm phân tán BitTorrent (DHT) cho phát hiện ngang hàng và giao thức giao vận uTorrent cho trao đổi dữ liệu. Mỗi thông điệp được mã hóa bằng RC4 và được ký sử dụng các khóa công khai và riêng tư. Chính vì vậy, việc phát hiện các hành vi độc hại của mã độc Hajime hiện còn gặp nhiều khó khăn.

BrickerBot là một mã độc IoT botnet dựa trên busybox được phát hiện bởi các nhà nghiên cứu hãng Radware [24] vào 4/2017. Mã độc Brickerbot có khả năng lây nhiễm và triển khai tấn công DDoS trên các thiết bị IoT tương tự như Mirai. Nguy hiểm hơn, mã độc này có khả năng phá hủy (bricks) các thiết bị sau khi đã lây nhiễm, khiến thiết bị đó không còn sử dụng được nữa. Mã độc Brickerbot cố gắng thực hiện tấn công từ chối dịch vụ các thiết bị IoT bằng nhiều phương pháp khác nhau như xóa các tập tin khỏi bộ nhớ, cấu hình lại thông số mạng, thay đổi firmware thiết bị…

VNPFilter xuất hiện vào năm 2018, là mã độc được thiết kế để gây hại đến các thiết bị định tuyến và các thiết bị lưu trữ trên mạng như thiết bị NAS (Network Attached Storage) [25] nhằm đánh cắp dữ liệu. Không giống như các mã độc IoT khác, VPNFilter là mã độc có khả năng tồn tại ngay cả khi thiết bị IoT được khởi động lại.

Qua quá trình đánh giá sự tiến hoá của các loại mã độc IoT Botnet trên cơ sở phương thức truyền thông, vòng đời, chức năng cũng như mã nguồn. Luận văn đã cho thấy mã độc Linux.Hydra là mã độc IoT botnet có khả năng tấn công từ chối dịch vụ phân tán đầu tiên xuất hiện từ năm 2008 với kiến trúc tương đối đơn giản. Kể từ khi mã nguồn của mã độc Linux.Hydra được công bố thì các nhà viết mã độc IoT đã kế thừa những ưu điểm và cho ra nhiều biến thể mã độc IoT Botnet như Psybot, Chuck Norris, Tsunami… Từ mã độc Tsunami các nhà viết mã độc đã phát triển thành hại loại mã độc xuất hiện gần đây nhất là Remaiten và LightAidra. Hơn nữa, từ mã độc Tsunami các nhà viết mã độc đã phát triển ra Bashlite và từ mã độc Bashlite thì mã độc Mirai đã kế thừa và tiến hòa với nhiều chức năng phức tạp hơn với các cuộc tấn công DDoS quy mô lớn tới 1.2 Tbps [18] vào năm 2016. Kể từ đó, mã độc Mirai đã tiếp tục được phát triển và tiến hóa thành nhiều biến thể khác nhau như BrickerBot, VPNFilter… Gần đây ngoài việc tận dụng các lợi thế của các loại mã độc IoT Botnet trước đó, các nhà viết mã độc đã tích hợp nhiều tính năng nguy hiểm và phức tạp hơn trên các loại mã độc như: mã hóa, proxy ẩn danh, rootkit và khả năng tự hủy, xáo trộn mã… khiến việc phát hiện, phân tích chúng khó khăn hơn

rất nhiều. Có thể kể đến VPNFilter là đại diện cho loại mã độc IoT botnet có cấu trúc phức tạp, tinh vi, sử dụng các kỹ thuật tự vệ được tìm thấy trong mã độc IoT botnet. Khảo sát trên của luận văn cho thấy, các loại mã độc IoT Botnet vẫn đang được phát triển, tiến hoá thường xuyên với cấu trúc tinh vi và khả năng bảo vệ phức tạp hơn. Tuy nhiên, phần khảo sát các loại mã độc bên trên vẫn chưa đầy đủ bởi tội phạm mạng thường xuyên sửa đổi và cập nhật các loại mã độc đã biết để tạo ra các loại mã độc mới, khai thác nhiều loại thiết bị IoT hơn.

1.2. Các nghiên cứu liên quan trong phát hiện sớm mã độc

Các phương pháp phát hiện mã độc IoT botnet có thể chia thành 2 hướng tiếp cận chính như: (1) các phương pháp dựa trên chữ ký (signature-based) và (2) các phương pháp dựa trên hành vi (behavior-based). Các phương pháp dựa trên chữ ký sử dụng một chuỗi các byte có thể là mã băm, được trích xuất từ mã độc IoT botnet đã biết như một chữ ký đại diện đặc trưng duy nhất cho mỗi tập tin nhận dạng mã độc. Ngoài ra, việc phát hiện mã độc dựa trên chữ ký đem lại khả năng phát hiện đúng cao với các loại mã độc IoT botnet đã được biết và phân tích. Tuy nhiên, việc gia tăng liên tục về số lượng cũng như sự phức tạp của mã độc như mã độc "zero-day" và mã độc "unknown" thì phương pháp dựa trên chữ ký khó có thể phát hiện tốt mã độc khi mà số lượng chữ ký mã độc ngày càng tăng, thậm chí bùng nổ nên yêu cầu không gian lưu trữ phải tỷ lệ thuận với số lượng chữ ký, điều này khiến chi phí lưu trữ chữ ký của mã độc rất lớn. Do đó, phương pháp phát hiện mã độc dựa trên hành vi được các nhà nghiên cứu áp dụng và phát triển nhằm bỏ qua những hạn chế mà phương pháp dựa trên chữ ký để lại. Phương pháp dựa trên hành vi yêu cầu thực thi các tập tin trong một môi trường có giám sát và các hành vi sẽ được ghi nhận và kiểm tra làm căn cứ để phát hiện mã độc. Phương pháp này giúp nhanh chóng hiểu được bản chất hành vi của tập tin. Đồng thời nhiều mã độc hiện nay được xây dựng từ cùng mã nguồn thành các biến thể vì vậy cấu trúc tập tin có thể khác nhau nhưng hành vi cơ bản không thay đổi. Tuy nhiên hạn chế của phương pháp này nằm ở tốn kém thời gian thực thi và lưu trữ đặc trưng mẫu hành vi.

Ngày nay với xu hướng mã độc trên thiết bị IoT đặc biệt mà mã độc IoT botnet đang tăng trưởng không ngừng cả về số lượng và biến thể, kéo theo đó các dữ liệu về chữ ký và hành vi mã độc cũng tăng với số lượng lớn khiến việc xử lý, phân tích được thực hiện bằng con người trở nên rất khó khăn. Nhằm khắc phục các vấn đề trên, hiện nay các nhà nghiên cứu tiếp cận các phương pháp phát hiện mã độc dựa trên học máy. Phương pháp dựa trên học máy không sử dụng chữ ký hay hành vi mã độc cụ thể mà nó sử dụng các đặc trưng và các đặc trưng này được xem là thành phần lõi của phát hiện dựa trên học máy. Thông qua việc khảo sát có thể thấy, tất cả các hướng tiếp cận trên được nhóm thành 2 phương pháp chính là phân tích tĩnh (static) và phân tích động (dynamic).

1.2.1. Phân tích tĩnh

Phân tích tĩnh là phương pháp phân tích nội dung của mã nguồn mà không cần thực thi các tệp tin để phát hiện các hành vi nghi vấn. Phương pháp phân tích tĩnh cho phép chi tiết hóa toàn bộ luồng điều khiển (Control Flow Graph – CFG) và luồng dữ liệu (Data Flow Graph – DFG) thông qua các công cụ dịch ngược mã nguồn như IDA Pro, BinDiff… để phát hiện mã độc bằng phân tích đặc trưng như mã thực thi (Opcode), lời gọi hàm hệ thống (API calls) hay các chuỗi ký tự có nghĩa trong mã nguồn (Printable Strings Information – PSI). Phương pháp này cho phép phân tích chi tiết các tệp tin và đưa ra các khả năng kích hoạt của mã độc [26].

Theo hướng tiếp cận này có thể kể đến phương pháp của Costin [27] đã đề xuất một framework để thu thập, lọc, unpack và phân tích tĩnh firmware quy mô rộng từ đó giúp phát hiện lỗ hổng bảo mật, mã độc. Tuy nhiên, nghiên cứu trên chỉ sử dụng các đặc trưng rời rạc mà không đi vào sự tương tác, liên quan giữa các đặc trưng…Trong khi đó, mã độc IoT botnet luôn có quy trình hoạt động khá tương đồng nhau và có sự tương tác với nhau [15], [28].

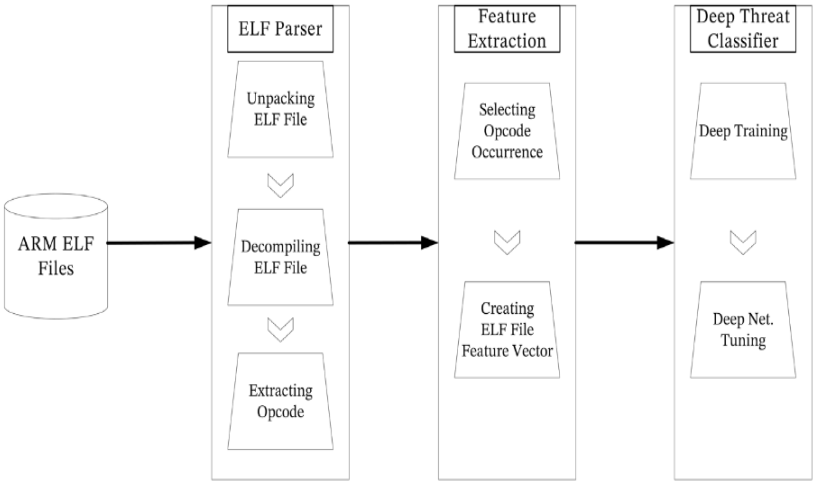
Yan Shoshitaishvili và cộng sự [29] cũng đã đề xuất framework Angr trong bảo mật IoT. Angr có khả năng xử lý, phân tích các firmware thành các đồ thị luồng điều khiển và đồ thị luồng dữ liệu và sử dụng biểu đồ này để tạo ra các lát cắt xác

thực bắt đầu từ entry-point đến các điểm xác thực của chương trình giúp phát hiện mã độc được nhúng hoặc các lỗ hổng trên firmware. Phương pháp này cho phép chi tiết hóa toàn bộ luồng điều khiển CFG và luồng dữ liệu DFG cho từng tập tin hệ thống trong firmware để từ đó phát hiện mã độc bằng kỹ thuật phân tích đặc trưng (symbolic execution). Tuy nhiên, phương pháp sử dụng Symbolic Execution chỉ phù hợp với việc phân tích các tập tin hệ thống nhỏ, và không khả thi với các tập tin lớn do độ phức tạp cao khi xây dựng các đồ thị liên quan và thực thi Symbolic Execution [27].

Azmoodhe [30] đã đề xuất một phương pháp phát hiện mã độc trên thiết bị IoT cho quân sự bằng học sâu (Internet of battlefield things - IoBT) dựa trên đồ thị mã Opcode. Đầu tiên, họ sử dụng Objdump để trích xuất chuỗi opcode và lựa chọn opcode 2 gram làm đặc trưng đầu vào. Sau đó, Azmoodhe xây dựng đồ thị mã Opcode từ các mã Opcodes đã chọn dựa trên biểu đồ luồng kiểm soát dữ liệu (Control Flow Graph - CFG) cho từng mẫu. Tác giả đã đánh giá phương pháp được đề xuất với tập dữ liệu bao gồm 128 mẫu mã độc và 1078 mẫu lành tính (tất cả các tệp thực thi chạy trên kiến trúc ARM) và có kết quả phát hiện đúng là 99,68%. Điểm hạn chế của đề xuất này là dataset bị giới hạn về số lượng mẫu và vấn đề đa kiến trúc không được xem xét vì kết quả thử nghiệm chỉ đưa ra đánh giá cho tập dữ liệu cho vi xử lý ARM. Bên cạnh đó, các chuỗi opcode có thể bị gián đoạn bởi các biến thể mã đơn giản do các tùy chọn biên dịch khác nhau cho đa kiến trúc vi xử lý trên thiết bị IoT.

Tiếp đó, Haddad Pajouh [31] cũng đề xuất phương pháp phát hiện mã độc IoT Botnet dựa trên học sâu RNN khi phân tích các OpCodes của mã nguồn (tổng quan về mô hình đề xuất như hình 1.5). Trong nghiên cứu này, tác giả chủ yếu phân tích các mẫu mã độc được thực thi trên phần cứng sử dụng bộ vi xử lý ARM. Sau đó, tác giả đánh giá hiệu suất của phương pháp đưa ra với các bộ phân loại học máy thông thường như SVM, KNN, Nave Bayes, Decision Tree, Random Forest, Ada-Boost và phương pháp đề xuất đã cho kết quả phát hiện tốt. Tuy nhiên, phương pháp của Haddad Pajouh phụ thuộc rất nhiều vào khả năng dịch ngược mã nguồn và

trích xuất chuỗi Opcode từ tệp tin ELF. Hơn nữa, phương pháp đề xuất chỉ thử nghiệm đánh giá chủ yếu trên các thiết bị sử dụng vi xử lý ARM, do đó không đáp ứng được yêu cầu thực nghiệm trên đa nền tảng kiến trúc vi xử lý.



Hình 1.3. Mô hình phát hiện IoT Botnet của Haddad Pajouh [31]

Với những nghiên cứu được khảo sát ở trên đã cho thấy, phân tích tĩnh đem lại kết quả khả quan trong bảo mật IoT nói chung và phát hiện mã độc IoT nói riêng. Tuy nhiên, phân tích tĩnh vẫn tồn tại nhiều hạn chế cho việc phân tích, phát hiện mã độc IoT Botnet như: Khó áp dụng đối với mã độc sử dụng kỹ thuật gây rối (obfuscation) hoặc đóng gói (pack) phức tạp do hạn chế của công cụ Unpack và Debug; Khó thu thập mẫu mã độc do mã độc chỉ lưu trữ trên RAM thiết bị, và biến mất khi khởi động lại thiết bị; Kết quả dịch ngược có thể không chính xác do các tùy chọn biên dịch khác nhau của công cụ dịch ngược đối với các nền tảng CPU đa dạng của các thiết bị IoT. Vì vậy, với bài toán phát hiện mã độc IoT Botnet trên các thiết bị IoT hiện nay, hướng tiếp cận dựa trên phân tích tĩnh trở nên khó thực hiện.

1.2.2. Phân tích động

Phân tích động là phương pháp giám sát các hành vi trong khi các tập tin đó đang chạy, từ đó phát hiện có hay không các hành vi độc hại, bất thường. Môi trường thực thi các tập tin thường là một môi trường mô phỏng (như sandbox) hoặc các thiết bị IoT thực tế (như cài đặt các tác tử). Những thông tin được thu thập như

các hành vi mức hệ thống (syscall, giá trị thanh ghi, dữ liệu bộ nhớ), các hành vi mức mạng (dữ liệu luồng mạng pcap). Phân tích động sẽ loại bỏ được các kỹ thuật gây rối mã nguồn, không dịch ngược được mã nguồn tệp tin thường gặp trong phân tích động. Tuy nhiên, khó khăn khi thực hiện phân tích động là việc xây dựng môi trường cho phép mã độc bộc lộ hoàn toàn các hành vi và có khả năng giám sát đầy đủ các hành vi đó.

Đối với việc sử dụng phân tích động để phát hiện IoT Botnet có thể được phân loại theo hai phương pháp chính là phát hiện xâm nhập dựa trên dữ liệu mạng (Network-based Intrusion Detection System – NIDS) và phát hiện xâm nhập dựa trên dữ liệu của máy (Host-based Intrusion Detection System – HIDS).

Theo hướng phát hiện xâm nhập dựa trên dữ liệu mạng, Doshi [32] đã đề xuất thuật toán phân lớp học máy sử dụng lưu lượng mạng nhằm phát hiện tấn công DDoS vào hệ sinh thái IoT. Mục tiêu của phương pháp đề xuất là có thể phát hiện lưu lượng tấn công DDoS trên các nút thiết bị IoT cục bộ bằng cách sử dụng thuật toán học máy với chi phí sử dụng tài nguyên thấp và có độ chính xác lên tới 99%. Để kiểm chứng, các mô hình thuật toán học máy khác nhau đã được kiểm tra và so sánh bao gồm thuật toán KNN, SVM, Decision Tree, Random Forest và ANN. Do các thiết bị IoT có nhược điểm hạn chế về tài nguyên phần cứng, vì vậy chúng không thể chạy trực tiếp các giải pháp phát hiện yêu cầu hiệu năng cao. Do đó cơ chế phát hiện được đề xuất thực hiện chính sách phát hiện dựa trên phân tích gói tin mạng với các đặc trưng được lựa chọn để giảm chi phí tính toán. Thuật toán phát hiện đã có thể thực hiện phát hiện trong thời gian thực trên các thiết bị IoT.

Alrashdi [33] đã đề xuất sử dụng bộ dữ liệu UNSW-NB15 [34] để xây dựng NIDS phát hiện IoT Botnet trong thành phố thông minh (smart city) hiệu quả hơn, thay thế bộ dữ liệu KDD99 đã lỗi thời. Alrashdi sử dụng giải thuật Random Forest với 12 đặc trưng được chọn từ bộ dữ liệu UNSW-NB15 để đạt được độ chính xác 99,34%.

Cardoso [35] đã đề xuất hệ thống phát hiện xâm nhập xử lý sự kiện phức tạp (CEPIDS) cho phép phân tích lưu lượng truy cập theo thời gian thực và được thiết kế đặt ở rìa của hệ thống mạng. CEPIDS thông qua bộ lọc sự kiện sẽ theo dõi và thu thập lưu lượng mạng. Bộ xử lý sự kiện bao gồm hai mô-đun, bộ phân tích gói tin và phát hiện tấn công, sẽ phân tích các gói lưu lượng dựa trên các thuộc tính của chúng và xác định hành vi tấn công trong mạng. CEP sẽ gửi các quy tắc tới Action Engine để gửi thêm cảnh báo về hoạt động độc hại đang diễn ra và chặn quyền truy cập vào các dịch vụ liên quan. Mô hình đề xuất đã được thử nghiệm trên thiết bị Raspberry Pi để đảm bảo rằng nó hoạt động tốt trên các thiết bị có sức mạnh tính toán thấp như thiết bị IoT. Mô phỏng cuộc tấn công được thực hiện trên thiết bị Raspberry Pi 3 Model B, nơi sử dụng CPU, sử dụng Ram và tốc độ gói tin đến và số lượng các cuộc tấn công được xác định. Các IDS mã nguồn mở hiện tại như SNORT và BRO tiêu thụ nhiều tài nguyên RAM và CPU trong khi cơ chế CEP được đề xuất sử dụng xử lý sự kiện tiêu tốn ít tài nguyên hơn. Điều này cho phép nó được triển khai trên các thiết bị IoT.

Ngoài hướng tiếp cận dựa trên NIDS được trình bày phía trên, cũng đã có nhiều nhà nghiên tiếp cận phát hiện mã độc theo HIDS.

Massimo Ficco [36] đã giới thiệu mô hình học máy phát hiện mã độc sử dụng chuỗi lời gọi API được gọi bởi các ứng dụng trong khi chúng đang chạy, sau đó mô hình hóa chúng bởi chuỗi Markov. Mô hình này áp dụng các đặc trưng theo thời gian để phân loại IoT Botnet. Kết quả thực nghiệm của phương pháp cho thấy tỷ lệ F1 đạt đến 89% với thuật toán Naive Bayes trên bộ dữ liệu gồm 22000 phần mềm lành tính và 24000 mã độc. Trong mô hình này, Ficco thu thập các lời gọi API từ các ứng dụng đã thực hiện để xây dựng "Call Dependency Graph and Calls tree". Từ đó, Ficco đã trích xuất đặc trưng phù hợp vào mô hình hành vi Markov Chain để phân loại IoT Botnet. Tuy nhiên, việc sử dụng công cụ mobSF [37] để trích chọn các đặc trưng dẫn đến giới hạn của nghiên cứu này chỉ phù hợp với các thiết bị Android, do đó sẽ gặp khó khăn khi mở rộng sang các hệ điều hành nhúng khác của thiết bị IoT, môi trường phổ biến với hệ điều hành Linux.

Breitenbacher [38] đề xuất HIDS với tên Hades-IoT IDS, yêu cầu năng lượng thấp cho các thiết bị IoT. Hades-IoT IDS sử dụng đặc trưng lời gọi hệ thống (system calls) để phát hiện hành vi độc hại của mã độc VP-Nfilter và IoTReaper trên 7 thiết bị IoT, bao gồm các bộ định tuyến và IP camera. Khả năng phát hiện mã độc của Hades-IoT IDS là 100% theo công bố tác giả. Tuy nhiên, kết quả thử nghiệm trên chỉ được áp dụng trên không gian thử nghiệm hạn chế (7 thiết bị IoT và hai mẫu mã độc) nên chưa thể kết luận được hiệu quả của phương pháp.

Qua việc khảo sát các nghiên cứu trên về việc phát hiện mã độc Botnet sử dụng phương pháp phân tích động cho thấy, phần lớn các nhà nghiên cứu chú tâm vào việc phát hiện mã độc dựa trên luồng mạng NIDS. Quá trình phát hiện xâm nhập dựa trên luồng mạng thường chỉ phát hiện hành vi độc hại khi mà thiết bị đã bị lây nhiễm thành công, trở thành một phần của IoT Botnet và bắt đầu truyền thông đến các máy chủ lệnh và điều khiển, các bot khác hoặc chúng thực hiện tấn công. Hình thức hiện mã độc dựa trên dữ liệu máy chủ HIDS có thể khắc phục được nhược điểm này. Tuy nhiên các thiết bị IoT khác với các thiết bị điện toán truyền thống vì chúng hạn chế về tài nguyên xử lý cũng như năng lượng. Hơn nữa với sự phát triển thần tốc về số lượng cũng như sự đa dạng về chức năng khiến cho các thiết bị IoT trở nên bất đồng nhất cả về kiến trúc phần cứng, giao tiếp truyền thông cũng như trạng thái hoạt động. Do đó, hướng tiếp cận phát hiện mã độc chỉ bằng HIDS là khó khăn và chưa đầy đủ.

1.3. Mô tả bài toán

Với sự phát triển nhanh chóng của các thiết bị IoT trên thế giới cả về số lượng lẫn chức năng, môi trường hoạt động. Do đó, các loại mã độc IoT Botnet cũng được tiến hoá thích ứng với môi trường của nạn nhân và khó bị phát hiện và phân tích hơn giúp duy trì hoạt động. Nhiều loại mã độc IoT Botnet gần đây đã được thiết kế để tránh bị phát hiện bởi các giải pháp bảo mật truyền thống hiện có hạn như phần mềm phát hiện và xử lý mã độc (Anti-virus), hệ thống phát hiện và xử lý xâm nhập mạng (IDS/IPS), các bộ lọc gói tin bằng tường lửa thông thường (Firewall). Các giải pháp bảo mật truyền thống này chỉ thực sự được phát hiện được

mạng lưới IoT Botnet khi chúng đã ở trong giai đoạn thực thi tấn công từ chối dịch vụ phân tán (DDoS) và gây ra hậu quả được thấy rõ. Hiện nay cũng đã có nhiều phương pháp để phát hiện, chống lại các cuộc tấn công DDoS như vậy. Tuy nhiên, việc phát hiện và chống lại mạng Botnet khi các cuộc tấn công đã diễn ra là khó khăn vì khi đó mạng Botnet đã tích trữ một lượng lớn các bot cho mình. Minh chứng là có thời điểm có tới 400.000 thiết bị IoT bị lây nhiễm mã độc Mirai và vụ tấn công DDoS đã được ghi nhận với quy mô lớn nhất được thực hiện bởi mã độc Mirai có lưu lượng lên đến 1.2 Tbps [18] vào năm 2016. Với số lượng lớn bot như vậy kèm theo lưu lượng tấn công lớn sẽ dễ dàng đánh bại mọi hệ thống phần cứng hiện nay. Hơn thế nữa, giai đoạn rà quét và xâm nhập chiếm dụng quyền quản trị thiết bị mục tiêu (Bước 1 và 2) của mạng lưới IoT Botnet diễn ra trong thời gian dài. Nếu như có thể phát hiện các thiết bị IoT cỡ nhỏ bị lây nhiễm trước khi chúng thực thi lệnh tấn công từ chối dịch vụ (Bước 6 và 7) thì sẽ hạn chế được hậu quả phá hoại của mã độc IoT Botnet.

Về cơ bản, các phương pháp phát hiện mã độc đều dựa trên hai phương pháp chính là phân tích tĩnh và động như đã trình bày ở trên. Với những nội dung khảo sát và đánh giá được luận văn trình bày bên trên, luận văn lựa chọn sử dụng phương pháp phân tích động để có thể phát hiện sớm mã độc IoT Botnet. Phương pháp phát hiện được luận văn lựa chọn sẽ kết hợp được ưu điểm của các phương pháp phát hiện trước đây và hướng tới mục tiêu có thể phát hiện sớm mã độc IoT Botnet ngay từ giai đoạn đầu.

Kết luận chương

Trong chương 1, luận văn đã chỉ ra tổng quan về các thiết bị IoT cỡ nhỏ, đặc điểm mã độc IoT Botnet lây nhiễm trên loại thiết bị này và các giải pháp phát hiện mã độc IoT Botnet đã được công bố. Từ đó, nội dung chương đưa ra những vấn đề còn tồn tại và đề xuất phương án giải quyết các vấn đề này. Nội dung chi tiết phương pháp giải quyết vấn đề được nêu ra tại Chương 2 của luận văn.

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

Những nghiên cứu liên quan về vấn đề phát hiện mã độc IoT Botnet đa phần sử dụng các dữ liệu được thu thập theo thời gian gọi là dữ liệu chuỗi thời gian. Với số lượng lớn các thiết bị IoT thì lượng dữ liệu thu thập được là rất lớn và có thể gây khó khăn trong việc xây dựng các bộ phân loại. Phương pháp phát hiện mã độc IoT thường được xây dựng bằng cách xây dựng một bộ phân loại đơn lẻ trên tập dữ liệu thu thập đầy đủ theo chuỗi thời gian. Các phương pháp xây dựng bộ phân loại khác nhau thường có những ưu, nhược điểm khác nhau, nhưng điểm chung của các mô hình này chính là khi mã độc đã thực hiện toàn bộ hành vi trong hệ thống thì mới có thể phát hiện được, trong thực tế có thể gây hậu quả nghiêm trọng cho hệ thống nếu không cảnh báo trước nguy cơ. Để đáp ứng được yêu cầu phân loại chính xác trong việc sử dụng các bộ phân loại học máy, mô hình học máy cộng tác đã được luận văn đưa vào sử dụng để tăng hiệu suất dự đoán của mô hình. Mô hình học máy cộng tác đưa ra dự đoán từ việc sử dụng các bộ phân loại con và tổng hợp dự đoán từ những bộ phân loại này để đưa ra quyết định. Từ những điểm đã nêu trên, trong phần này luận văn sẽ xây dựng mô hình học máy phát hiện sớm mã độc IoT Botnet sử dụng các bộ dữ liệu có đặc trưng về thời gian của thiết bị, thực hiện các phương pháp chọn lọc và chuẩn hóa dữ liệu, áp dụng mô hình học máy cộng tác với các bộ phân loại con thích hợp cho bài toán phát hiện sớm mã độc IoT Botnet.

2.1. Tổng quan mô hình học máy cộng tác

Học máy cộng tác là quá trình sử dụng các bộ phân loại và kết hợp kết quả dự đoán của các bộ phân loại này để tạo nên một mô hình đưa ra quyết định phức tạp nhưng cải thiện hiệu năng dự đoán hơn so với các bộ phân loại con. Phương pháp kết hợp có thể linh hoạt dựa trên đặc trưng, hoặc kết quả phân loại. Có một vài lý do khiến cho mô hình học máy cộng tác có thể cải thiện khả năng dự đoán [39]:

- Tránh việc mô hình quá vừa dữ liệu: Khi chỉ có một lượng nhỏ dữ liệu, thuật toán học có xu hướng tìm ra nhiều giả thuyết khác nhau dự đoán tất cả dữ liệu

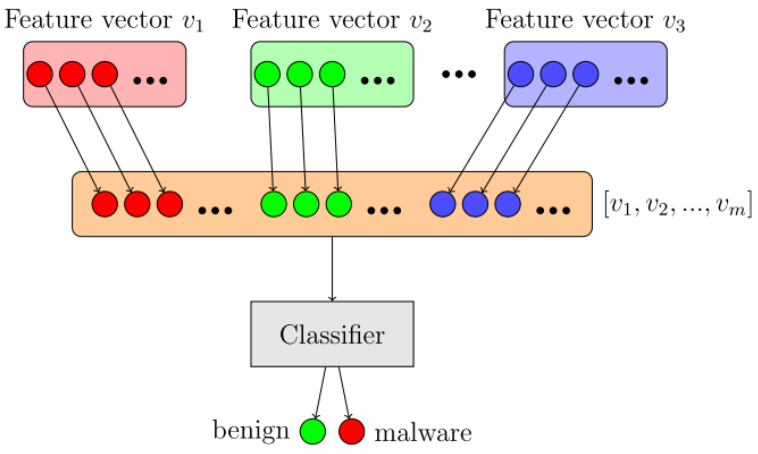
huấn luyện một cách hoàn hảo trong khi đưa ra dự đoán kém cho các trường hợp chưa nhìn thấy. Tính trung bình các giả thuyết khác nhau làm giảm nguy cơ chọn một giả thuyết không chính xác và do đó, cải thiện hiệu suất dự đoán tổng thể.

- Lợi thế về việc tính toán: Những bộ phân loại đơn lẻ thực hiện các tìm kiếm cục bộ có thể gặp khó khăn trong tối ưu cục bộ. Bằng cách kết hợp nhiều bộ phân loại học, các phương pháp tổng hợp làm giảm nguy cơ đạt được cực tiểu cục bộ.

- Biểu diễn dữ liệu: Giả thuyết tối ưu có thể nằm ngoài không gian của bất kỳ mô hình đơn lẻ nào. Bằng cách kết hợp các mô hình khác nhau, không gian tìm kiếm có thể được mở rộng và do đó, đạt được sự phù hợp hơn với không gian dữ liệu.

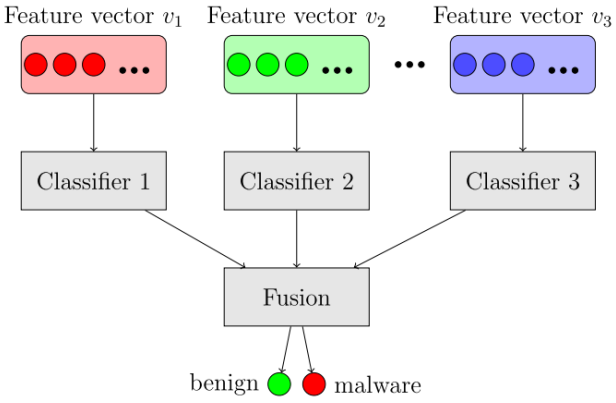
Dựa vào phương thức kết hợp dữ liệu có thể chia các mô hình học cộng tác thành 3 nhóm chính:

- Hợp nhất sớm: là phương pháp hợp nhất các dữ liệu đầu vào bằng cách tạo ra một tập dữ liệu đại diện cho các tập dữ liệu con đơn lẻ. Tập dữ liệu đại diện này được sinh ra bằng cách nối các đặc trưng của các tập dữ liệu con vào với nhau để tạo thành tập dữ liệu đại diện có chứa tất cả các đặc trưng của các tập dữ liệu con. Sau khi đã có được tập đại diện thì mô hình phân loại sử dụng một thuật toán học máy duy nhất để thực hiện quá trình phân loại dữ liệu đại diện.



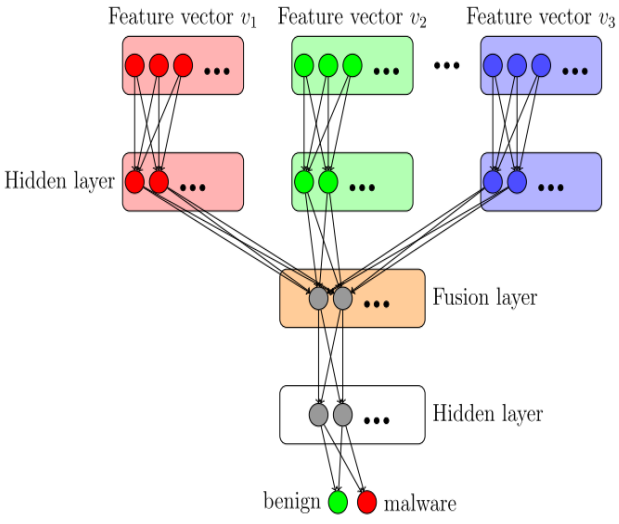
Hình 2.1. Mô hình hợp nhất sớm

- Hợp nhất muộn: là phương pháp cho phép các tập hợp các kết quả phân loại của các bộ học máy phân loại (Classifier) đơn lẻ thông qua hàm hợp nhất (Fusion). Mỗi một bộ dữ liệu đặc trưng đầu vào sẽ được huấn luyện và phân loại dựa trên các thuật toán học máy riêng biệt. Kết quả phân loại sẽ được tổng hợp để đưa ra quyết định cuối cùng.



Hình 2.2. Mô hình hợp nhất muộn

- Hợp nhất trung gian (Intermediate fusion): là cách hợp nhất các đặc trưng qua việc sử dụng các lớp ẩn (Hidden layer). Các đặc trưng đầu vào sẽ được đưa vào các thuật toán học máy có lớp ẩn để tìm ra các đặc trưng có liên quan tới mục tiêu phân loại. Kết quả phân loại từng bộ dữ liêu riêng lẻ này sẽ được đưa qua lớp hợp nhất và quyết định cuối cùng cũng sử dụng một lớp ẩn để tổng hợp kết quả.



Hình 2.3. Mô hình hợp nhất trung gian

Các mô hình học máy cộng tác kể trên đều có ưu điểm và nhược điểm, bằng việc nghiên cứu lý thuyết và qua quá trình thực nghiệm, luận văn đã lựa chọn sử dụng phương pháp hợp nhất muộn cho bài toán phát hiện sớm mã độc IoT Botnet và đạt được hiệu quả khả quan.

2.2. Mô hình ứng dụng

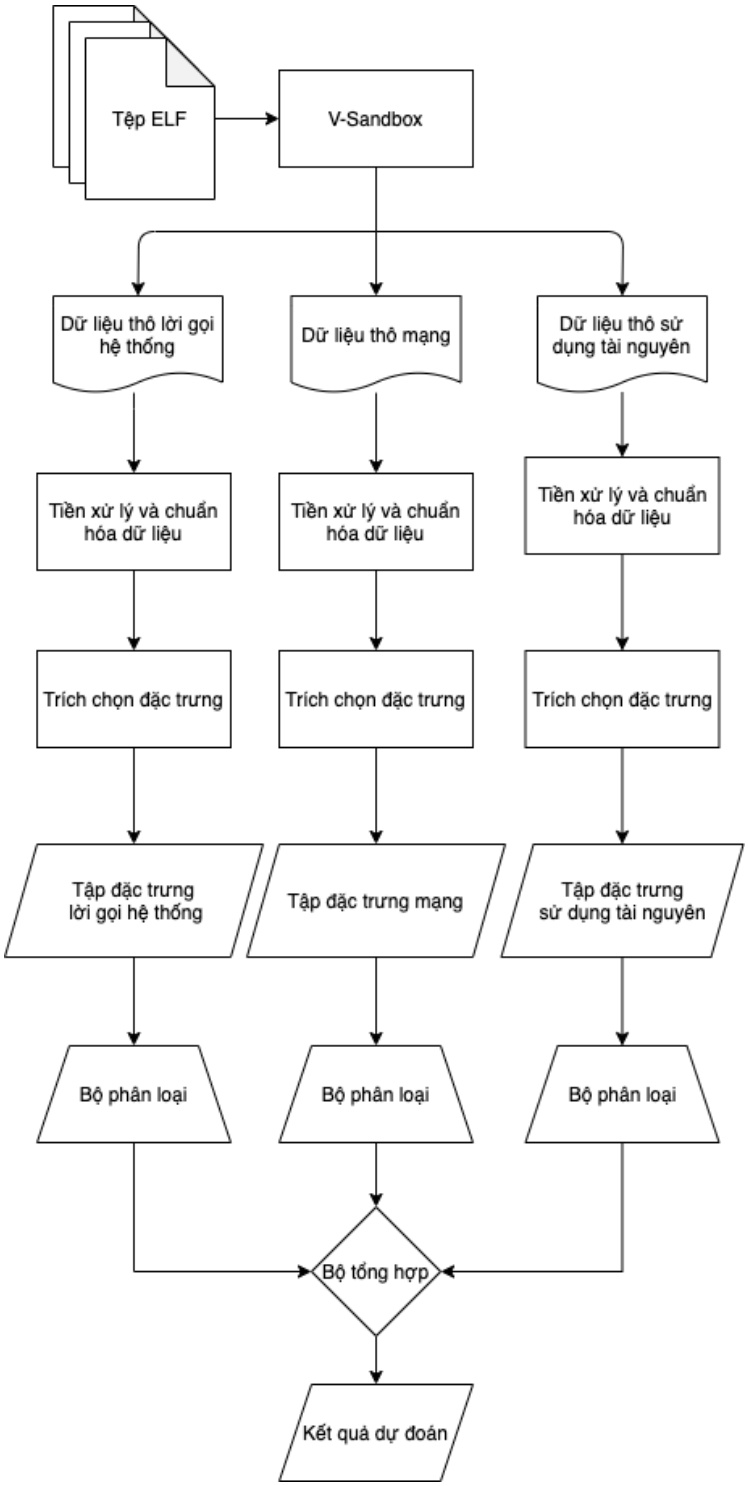
Trong phần này, luận văn sẽ ứng dụng mô hình học máy cộng tác phát hiện sớm IoT Botnet. Kiến trúc tổng quát của mô hình được biểu diễn cụ thể trong hình 2.4. Tổng quát của mô hình có 5 thành phần chính để trích xuất và xử lý dữ liệu giúp đưa ra quyết định:

- Bộ phận thu thập dữ liệu;
- Bộ phận tiền xử lý và chuẩn hóa dữ liệu;
- Bộ phận trích chọn đặc trưng;
- Các bộ phát hiện dựa trên thuật toán học máy khác nhau;
- Bộ tổng hợp kết quả phát hiện.

Sự khác biệt để giải quyết vấn đề phát hiện mã độc IoT Botnet trong mô hình thử nhiệm so với các mô hình truyền thống là mô hình thử nghiệm sử dụng môi trường sandbox (trong trường hợp này là V-Sandbox [40]) để lấy một phần nhỏ lượng dữ liệu đặc trưng cho các hành vi đầu tiên của tệp đầu vào đang được xử lý để thực hiện phân tích và phát hiện mã độc thay vì phải đợi mã độc thực hiện đầy đủ hành vi để thu thập và xử lý với toàn bộ dữ liệu.

Tổng quan quy trình xử lý và đưa ra quyết định của hệ thống được mô tả như sau. Các dữ liệu hành vi (bao gồm luồng mạng, lời gọi hệ thống và sử dụng tài nguyên của thiết bị) của tệp ELF đầu vào được thu thập trong môi trường sandbox. Các dữ liệu thu thập được này sẽ được tiền xử lý, chuẩn hóa và lựa chọn các đặc trưng phù hợp để đưa vào các mô hình bộ phân loại sử dụng thuật toán học máy. Những dự đoán của các mô hình học máy đơn lẻ này sau đó sẽ được đưa vào bộ tổng hợp để đưa ra kết luận phân loại cuối cùng.

Các thành phần và hoạt động cụ thể của chúng sẽ được trình bày trong phần tiếp theo và quá trình thực hiện cụ thể sẽ được trình bày trong chương 3 qua phần thực nghiệm và kết quả.



Hình 2.4. Kiến trúc mô hình ứng dụng

2.2.1. Bộ phận thu thập dữ liệu

Qua tìm hiểu, luận văn đã tìm được một số môi trường được xây dựng cho việc thu thập dữ liệu hành vi trong bài toán phát hiện mã độc trên các thiết bị IoT cỡ nhỏ như IOTBOX [41], Cuckoo [42], REMNUX [43], Limon [44] và V-Sandbox [40]. Mỗi một loại sandbox đều có nhưng ưu nhược điểm khác nhau và cụ thể về sự khác biệt trong việc hỗ trợ thu thập dữ liệu của các loại sandbox được mô tả trong Bảng 2.1.

Bảng 2.1. Các tính năng của các mô trường Sandbox

Tính năng / **Sandbox**	**Khả năng hỗ trợ**			**Thu thập dữ liệu**		
	Đa kiến trúc CPU	**C&C Server**	**Thư viện liên kết động**	**Luồng mạng**	**Lời gọi hệ thống**	**Chiếm dụng tài nguyên hệ thống**
IOTBOX [41]	Có	Không	Không	Có	Không	Không
Cuckoo [42]	Có	Không	Không	Có	Không đầy đủ	Không
REMNUX [43]	Không	Không	Không	Không	Có	Không
Limon [44]	Không	Không	Không	Có	Có	Không
V-Sandbox [40]	**Có**	**Có**	**Có**	**Có**	**Có**	**Có**

Qua việc so sánh các tính năng của các môi trường Sandbox trong bảng trên, dễ dàng nhận thấy V-Sandbox có đầy đủ tính năng hơn so với các môi trường sandbox khác. Kiến trúc tổng quan của môi trường V-Sandbox được tác giả minh họa như trong Hình 2.5. Với đầu vào là các tệp định dạng ELF (được sử dụng phổ biến trong Linux), môi trường V-Sandbox tự động tạo môi trường phù hợp cho phép tệp này thực thi và giám sát các hành vi tương tác với hệ điều hành.

Cụ thể, để thực hiện được nhiệm vụ này, quy trình thực hiện trong V-Sandbox bao gồm các bước sau:

Bước 1, tệp ELF đầu vào được đưa vào khối "ELF Metadata Extraction component (EME)" để phân tích tiêu đề tệp (file header) nhằm trích xuất các thông tin yêu cầu cơ bản để có thể thực thi được bao gồm: định dạng tệp thực thi, kiến trúc vi xử lý, loại hệ điều hành, có cần thư viện liên kết động hay không,…

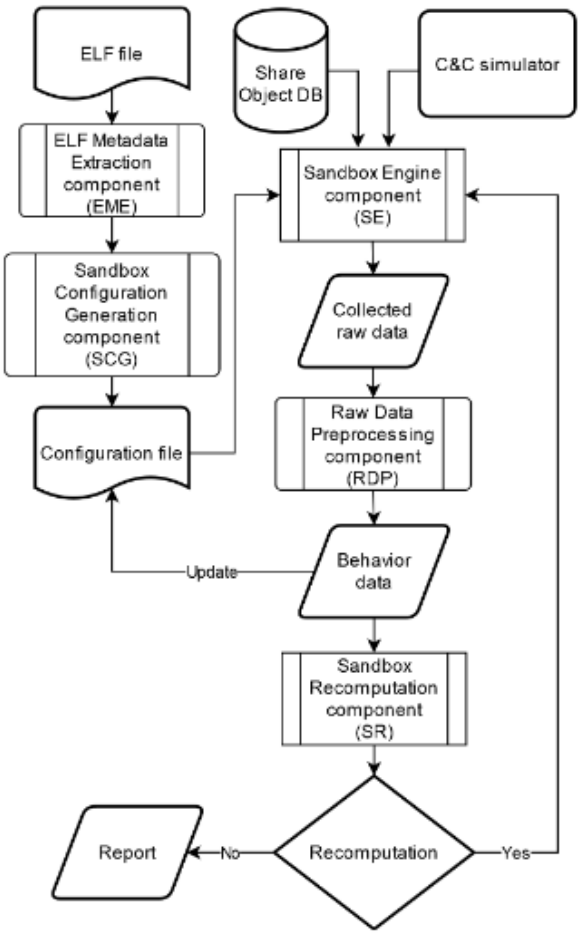
Bước 2, các thông tin yêu cầu cơ bản của tệp thực thi do khối EME trích xuất được chuyển cho khối "Sandbox Configuration Generation component (SCG)" để tự động sinh cấu hình khởi tạo ban đầu cho môi trường ảo hóa thực thi tệp đầu vào. Các thông tin cấu hình này được lưu trữ vào tệp cấu hình (Configuration file).

Bước 3, môi trường ảo hóa "Sandbox Engine component (SE)" được tự động khởi tạo dựa trên các cấu hình được lưu trữ trong "Configuration file". Tệp thực thi ELF được đưa vào môi trường này để kích hoạt thực thi và thể hiện các hành vi của mình. Môi trường SE được xây dựng dựa trên nền tảng ảo hóa QEMU [45] và tích hợp các tác tử giám sát hành vi cơ bản như: luồng mạng, lời gọi hệ thống, chiếm dụng tài nguyên hệ thống. Các yêu cầu về kết nối với C&C server hay thư viện liên kết động được đáp ứng thông qua các khối "C&C simulator" và "Share Object DB".

Bước 4, các thông tin giám sát do tác tử ghi nhận được trong môi trường SE được tiền xử lý bởi khối "Raw Data Preprocessing component (RDP)" để xác định nội dung, yêu cầu cũng như thống kê các hành vi của tệp đầu vào. Các yêu cầu bổ sung của tệp thực thi đối với môi trường ảo hóa (như thư viện liên kết động, kết nối với C&C server, …) được ghi nhận và cập nhật tự động vào tệp "Configuration file".

Bước 5, dựa trên các dữ liệu hành vi và yêu cầu của tệp thực thi đối với môi trường SE, khối "Sandbox Recomputation component (SR)" tiến hành tính toán và ra quyết định xem có nên chạy lại môi trường SE để thu thập thêm dữ liệu hành vi hay không. Nếu quyết định đưa ra là có thì sẽ chuyển sang bước 3, nếu không thì chuyển sang bước tiếp theo (bước 6).

Bước 6, báo cáo chi tiết về các hành vi được giám sát và thu thập từ tệp thực thi được sinh ra bởi khối "Report". Các dữ liệu thu thập được minh họa như trong các hình dưới đây.

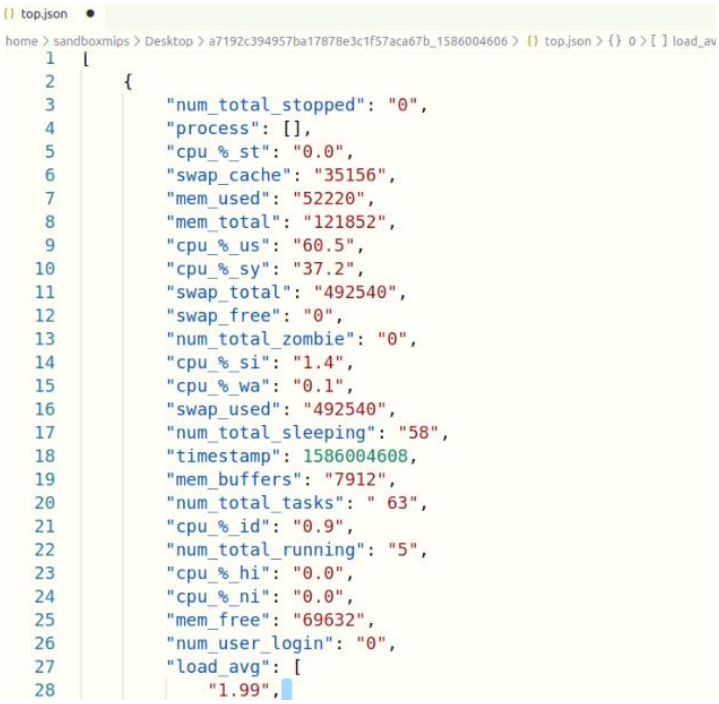


Hình 2.5. Kiến trúc của V-Sandbox [40]

Hình 2.6. Dữ liệu lời gọi hệ thống được thu thập bởi V-Sandbox

```
sandboxmips:a7192c394957ba17878e3c1f57aca67b_1586004606$ tcpdump -c 10 -r tcpdump.pcap
reading from file tcpdump.pcap, link-type EN10MB (Ethernet)
19:50:07.858808 IP 192.168.122.101.59487 > 192.168.122.1.12345: Flags [S], seq 3181798
659, win 14600, options [mss 1460,sackOK,TS val 4294920805 ecr 0,nop,wscale 2], length
 0
19:50:07.860616 IP 192.168.122.1.12345 > 192.168.122.101.59487: Flags [S.], seq 356555
1994, ack 3181798660, win 65160, options [mss 1460,sackOK,TS val 2998667613 ecr 429492
0805,nop,wscale 7], length 0
19:50:07.864555 IP 192.168.122.101.59487 > 192.168.122.1.12345: Flags [.], ack 1, win
3650, options [nop,nop,TS val 4294920807 ecr 2998667613], length 0
19:50:08.048936 IP 192.168.122.101.59487 > 192.168.122.1.12345: Flags [P.], seq 1:5, a
ck 1, win 3650, options [nop,nop,TS val 4294920853 ecr 2998667613], length 4
19:50:08.049312 IP 192.168.122.1.12345 > 192.168.122.101.59487: Flags [.], ack 5, win
510, options [nop,nop,TS val 2998667802 ecr 4294920853], length 0
19:50:08.051783 IP 192.168.122.101.59487 > 192.168.122.1.12345: Flags [P.], seq 5:6, a
ck 1, win 3650, options [nop,nop,TS val 4294920854 ecr 2998667802], length 1
19:50:08.051953 IP 192.168.122.1.12345 > 192.168.122.101.59487: Flags [.], ack 6, win
510, options [nop,nop,TS val 2998667805 ecr 4294920854], length 0
19:50:08.052096 IP 192.168.122.1.12345 > 192.168.122.101.59487: Flags [P.], seq 1:15,
ack 6, win 510, options [nop,nop,TS val 2998667805 ecr 4294920854], length 14
19:50:08.053260 IP 192.168.122.101.59487 > 192.168.122.1.12345: Flags [.], ack 15, win
 3650, options [nop,nop,TS val 4294920855 ecr 2998667805], length 0
19:50:08.230657 IP 192.168.122.101.35419 > 192.168.0.1.46718: UDP, length 512
sandboxmips:a7192c394957ba17878e3c1f57aca67b_1586004606$
```

Hình 2.7. Dữ liệu luồng mạng được thu thập bởi V-Sandbox



Hình 2.8. Dữ liệu sử dụng tài nguyên hệ thống được thu thập bởi V-Sandbox

Từ nhưng ưu điểm và tính năng nêu trên, luận văn nhận thấy môi trường V-Sandbox rất phù hợp cho quá trình xây dựng môi trường mô phỏng các thiết bị IoT để thu dữ liệu hành vi nhằm giải quyết bài toán phát hiện sớm mã độc. Quá trình mô phỏng trên V-Sandbox sẽ sử dụng bộ dữ liệu (Dataset) các tệp thực thi ELF được tác giả Lê Hải Việt thu thập và chia sẻ [40].

2.2.2. Bộ phận tiền xử lý và chuẩn hóa dữ liệu

Trong bài toán phát hiện mã độc IoT Botnet, các hành vi thường được những nhà nghiên cứu sử dụng để nhận biết dấu hiệu của chúng thường là: các lời gọi hệ thống, các gói tin trao đổi trong mạng và các thay đổi về tài nguyên của hệ thống. Do đó, luận văn sử dụng ba loại hành vi phổ biến kể trên để thu thập dữ liệu phục vụ cho đánh giá khả năng phát hiện mã độc IoT Botnet của mô hình ứng dụng.

Như đã đề cập trong mô tả tổng quan về mô hình ứng dụng, dữ liệu đưa vào mô hình học máy để phát hiện chỉ là các hành vi đầu tiên của tệp thực thi, không phải là dữ liệu toàn bộ hành vi mã độc cần thực hiện. Vì vậy, cần xác định lượng dữ liệu cần thiết là bao nhiêu để có khả năng phát hiện chính xác mã độc IoT Botnet.

Để trả lời câu hỏi này, luận văn đã tiến hành khảo sát, so sánh sự khác nhau trong dữ liệu thu thập được của mã độc IoT Botnet và tệp lành tính. Sau khi thu được dữ liệu tương ứng với lượng dữ liệu được xác định kể trên, luận văn tiến hành trích xuất đặc trưng để thu được bộ dữ liệu vector làm đầu vào cho các thuật toán học máy. Quá trình xử lý riêng biệt của mỗi loại dữ liệu được trình bày chi tiết sau đây.

2.2.2.1. Dữ liệu lời gọi hệ thống

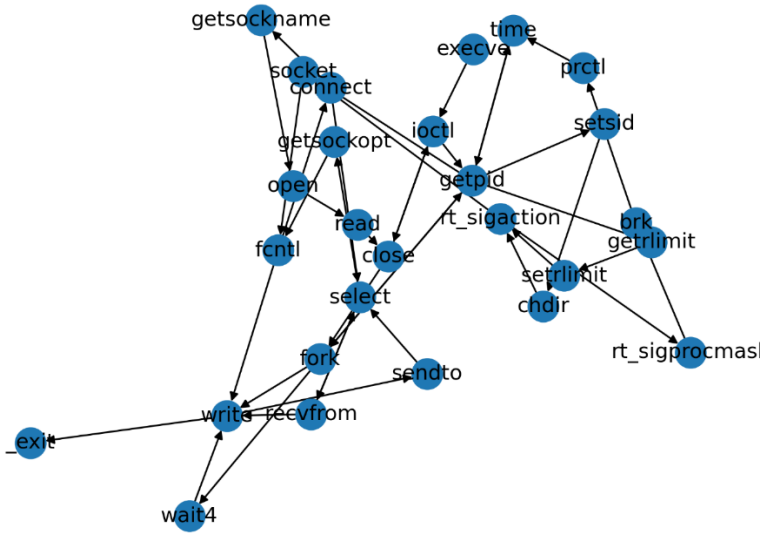
Từ quá trình thống kê dữ liệu có trong bộ dữ liệu lời gọi hệ thống, luận văn nhận thấy số lượng lời gọi hệ thống của các tệp thực thi dữ liệu lành tính thường ít hơn số lượng lời gọi hệ thống của các tệp thực thi có chứa mã độc, các tệp lành tính thường chỉ có khoảng dưới 100 lời gọi hệ thống mỗi lần thực thi trong khi con số này đối với các tệp có chứa mã độc là trên 300. Do vậy luận văn chọn ngưỡng 300 lời gọi hệ thống làm ngưỡng ngắt giám sát cho V-Sandbox và sử dụng 300 lời gọi hệ thống đầu tiên này làm dữ liệu cho các bộ học máy huấn luyện và kiểm thử.

Để trích xuất đặc trưng từ dữ liệu lời gọi hệ thống, luận văn sử dụng đồ thị lời gọi hệ thống (System Call Graph) được đề xuất bởi tác giả Lê Hải Việt và các cộng sự [46]. Theo đó, đồ thị lời gọi hệ thống được định nghĩa như sau:

Định nghĩa đồ thị lời gọi hệ thống (SCG): Một đồ thị System Call Graph được định nghĩa là $SCG(V, E)$*, trong đó:*

- $V = \{v_i\}$ *là tập các đỉnh* v_i *, mỗi đỉnh thể hiện một lời gọi hàm có cùng tên.*
- $E = \{e_i = (v_i, v_j)\}$ *là tập các cạnh* e_i *liên kết đỉnh* v_i *với đỉnh* v_j *của đồ thị.*

Hình 2.8 minh họa một đồ thị lời gọi hàm mà luận văn thu được.



Hình 2.9. Minh họa một đồ thị lời gọi hàm

Tuy nhiên, dữ liệu đồ thị không thể được huấn luyện trực tiếp bằng các bộ phân loại thông dụng. Do đó, dữ liệu này phải được chuyển về dưới dạng véc-tơ. Quá trình này được thực hiện thông qua công cụ Graph2vec [47]. Graph2vec là một kỹ thuật học không giám sát để chuyển đổi một đồ thị thành dạng véc-tơ số dựa trên ý tưởng hướng tiếp cận của thuật toán nhúng văn bản Doc2vec [48]. Theo đó, Graph2vec học cách biểu diễn các đồ thị bằng cách xem toàn bộ một đồ thị như một văn bản và các đồ thị con như các từ tạo nên văn bản đó. Kết quả thu được là một không gian véc-tơ nhúng mà các đồ thị có cấu trúc giống nhau thì có véc-tơ đặc trưng gần nhau. Đặc điểm này giúp cho các thuật toán học máy có thể hoạt động hiệu quả hơn trong việc phân loại đồ thị.

2.2.2.2. Dữ liệu luồng mạng

Quá trình thống kê dữ liệu có trong bộ dữ liệu luồng mạng cũng cho thấy số lượng gói tin sử dụng để giao tiếp giữa các tệp lành tính cũng ít hơn các tệp có chứa mã độc. Điều này có thể lý giải bằng lý thuyết của IoT Botnet vì chung thường xuyên phải kết nối và nhận lệnh từ máy chủ C&C, dẫn đến lưu lượng truy cập cao hơn bình thường. Kết quả thống kê cho thấy hầu hết các tệp cho số lượng gói tin

được sử dụng nằm ở ngưỡng 50 gói tin. Do đó, quá trình phát hiện sớm sử dụng ngưỡng 50 gói tin đầu tiên để ngắt giám sát và tiến hành phát hiện mã độc.

Dữ liệu luồng mạng sau khi thu được từ môi trường V-Sandbox là các tệp tin định dạng PCAP. Luận văn sử dụng công cụ CICFlowMeter để trích xuất đặc trưng dữ liệu từ các file PCAP này. Công cụ đã được sử dụng để xây dựng bộ dữ liệu luồng mạng CSE-CIC IDS2018 [49]. Luận văn đã sử dụng 24 đặc trưng được mô tả ở bảng dưới. Tuy nhiên, một tập tin PCAP chứa nhiều luồng mạng (flow), vì vậy để sinh được 1 véc-tơ duy nhất đại diện cho tập tin PCAP, luận văn tiến hành kết hợp thông tin của các luồng có trong tệp tin PCAP bằng cách thống kê các đại lượng tổng và giá trị lớn nhất. Bộ đặc trưng bao gồm 48 đặc trưng thống kê trên và 1 đặc trưng số lượng luồng mạng trong tập tin PCAP đó. Như vậy, có tất cả 49 đặc trưng được sử dụng.

Bảng 2.2. Mô tả đặc trưng CSE-CIC được sử dụng

Đặc trưng	**Mô tả**
fl_dur	Độ dài luồng
tot_fw_pk	Tổng số gói tin theo chiều đi
tot_bw_pk	Tổng số gói tin theo chiều về
tot_l_fw_pkt	Tổng kích thước gói tin theo chiều đi
tot_l_bw_pkt	Tổng kích thước gói tin theo chiều về
fw_iat_tot	Tổng thời gian giữa 2 gói tin được gửi theo chiều đi
bw_iat_tot	Tổng thời gian giữa 2 gói tin được gửi theo chiều về
fw_psh_flag	Số lần cờ PSH được bật trong gói tin di chuyển theo hướng đi (0 cho UDP)
bw_psh_flag	Số lần cờ PSH được bật trong gói tin di chuyển theo hướng về (0 cho UDP)
fw_urg_flag	Số lần cờ URG được bật trong gói tin di chuyển theo hướng đi (0 cho UDP)
bw_urg_flag	Số lần cờ URG được bật trong gói tin di chuyển theo hướng về (0 cho UDP)
fw_hdr_len	Kích thước được sử dụng cho header theo chiều đi
bw_hdr_len	Kích thước được sử dụng cho header theo chiều về

fin_cnt	Số gói tin với FIN
syn_cnt	Số gói tin với SYN
rst_cnt	Số gói tin với RST
pst_cnt	Số gói tin với PUSH
ack_cnt	Số gói tin với ACK
urg_cnt	Số gói tin với URG
cwe_cnt	Số gói tin với CWE
ece_cnt	Số gói tin với ECE
fw_win_byt	Kích thước các gói tin được gửi trong một cửa sổ ban đầu theo chiều đi
bw_win_byt	Kích thước các gói tin được gửi trong một cửa sổ ban đầu theo chiều về
Fw_act_pkt	Số các gói tin với ít nhất 1 byte của TCP data payload theo chiều đi

2.2.2.3. Dữ liệu sử dụng tài nguyên hệ thống

Thống kê bộ dữ liệu cho thấy các tệp thực thi mã độc thường yêu cầu sử dụng tài nguyên nhiều hơn so với các tệp thực thi lành tính và biểu hiện rõ nhất ở 20 trạng thái tài nguyên hệ thống đầu tiên. Do đó, luận văn chọn ngưỡng 20 trạng thái đầu tiên làm ngưỡng phát hiện sớm.

Các đặc trưng thu được từ việc chạy môi trường V-Sandbox đã được trình bày trong bài báo của tác giả Lê Hải Việt về V-Sandbox [40]. Tuy nhiên, dữ liệu sử dụng tài nguyên thiết bị bao gồm chuỗi nhiều trạng thái hiệu năng liên tiếp của hệ thống. Để có thể mô tả lại dữ liệu này, luận văn sử dụng phương pháp thống kê để định nghĩa các đặc trưng được sử dụng. Theo đó thông tin liên quan đến CPU, bộ nhớ của các trạng thái sẽ được tính toán các đại lượng trung bình cộng, độ lệch chuẩn, số lớn nhất, số nhỏ nhất. Với 20 đặc trưng của V-Sandbox (bảng 2), luận văn tiến hành thống kê và thu được 80 đặc trưng theo các 4 đại lượng trên.

Bảng 1.3. Đặc trưng hiệu năng hệ thống của V-Sandbox

Đặc trưng	**Mô tả**
num_total_running	Số tiến trình đang chạy

num_total_sleeping	Số tiến trình đang ngủ
num_total_zombie	Số tiến trình đang ở trạng thái zombie
num_total_stopped	Số tiến trình đã dừng lại
cpu_%_us	%CPU cho các tiến trình ở user space
cpu_%_sy	%CPU cho các tiến trình ở kernel
cpu_%_ni	%CPU cho các tiến trình "niced" ở user space
cpu_%_id	%CPU cho các tiến trình nhàn rỗi
cpu_%_wa	%CPU cho các tiến trình nhàn rỗi khi chờ I/O
cpu_%_hi	%CPU cho các tiến trình đang dùng ngắt phần cứng
cpu_%_si	%CPU cho các tiến trình đang dùng ngắt phần mềm
cpu_%_st	%CPU cho các tiến trình đang chờ phục vụ CPU ảo khác
mem_total	Tổng bộ nhớ RAM vật lý
mem_used	Lượng bộ nhớ đã được sử dụng
mem_free	Lượng bộ nhớ còn trống
mem_buffers	Lượng bộ nhớ được dành cho bộ đệm file
swap_total	Lượng bộ nhớ swap sẵn có
swap_used	Lượng bộ nhớ swap đã được sử dụng
swap_free	Lượng bộ nhớ swap còn trống
swap_cache	Lượng bộ nhớ swap được sử dụng như bộ nhớ cache

2.2.3. Bộ phận trích chọn đặc trưng

Sau khi trích xuất đặc trưng và véc-tơ hóa, dữ liệu đã có thể đưa vào các bộ phân loại để huấn luyện/ kiểm thử. Tuy nhiên để nâng cao độ hiệu quả của mô hình, luận văn tiến hành trích chọn đặc trưng.

Trích chọn đặc trưng giúp nâng cao độ chính xác, độ hiệu quả cho mô hình vì 2 lý do chính:

- Trích chọn đặc trưng giúp lược bỏ các đặc trưng thừa, không đóng góp nhiều vào quá trình phân loại của mô hình, cũng như các đặc trưng gây nhiễu, gây ảnh hưởng đến hiệu quả của bộ phân loại. Do đó, trích chọn đặc trưng vừa giúp nâng cao độ chính xác cho mô hình, vừa giảm thiểu tình trạng quá khớp (overfitting).

- Trích chọn đặc trưng làm giảm chiều véc-tơ đặc trưng đầu vào, qua đó đẩy nhanh tốc độ tính toán, giúp mô hình hội tụ nhanh hơn.

Có nhiều phương pháp trích chọn đặc trưng, luận văn đã sử dụng phương pháp đơn giản Information gain và áp dụng lên 2 loại đặc trưng là dữ liệu luồng mạng và đặc trưng dữ liệu hiệu năng hệ thống.

Information Gain là đại lượng đo độ hỗn tạp hay hỗn loạn của đặc trưng. Trong cây quyết định, information gain đươc sử dụng làm tiêu chí phân chia một node. Một thuộc tính có information gain hay độ giảm entropy lớn sẽ được chọn để chia node, bởi vì tính bất định của thông tin được giảm xuống nhiều nhất. Trong bài toán trích chọn đặc trưng, information gain được sử dụng để đo độ liên quan của đặc trưng A đối với lớp C. Giá trị của thông tin chung (mutual information) giữa thuộc tính A và lớp C càng cao thì độ liên quan giữa chúng càng lớn.

$$I(C, A) = H(C) - H(C|A),$$

Trong đó $H(C) = -\sum p(C)\log p(C)$, entropy của lớp C, và $H(C|A)$ là entropy của lớp C với điều kiện thuộc tính A, $H(C|A) = \sum p(C|A).\log p(C|A)$.

Luận văn tiến hành tính thông tin chung giữa tất cả các đặc trưng và nhãn và thu được kết quả sau:

- Đối với đặc trưng mạng:

Kết quả lựa chọn đặc trưng được trình bày ở bảng 3. Một số đặc trưng cho giá trị thông tin chung tính được bằng 0 là do những đặc trưng này chỉ có 1 giá trị

duy nhất trong cả bộ dữ liệu huấn luyện. Do đó, những đặc trưng này chắc chắn sẽ bị loại bỏ, vì không đóng góp gì vào quá trình phân loại của các thuật toán học máy.

Bảng 2.4. Kết quả lựa chọn đặc trưng đối với dữ liệu luồng mạng

STT	Đặc trưng	Giá trị thông tin chung
1	Sum Flow Duration	0.372012577
2	Max Flow Duration	0.369644832
3	Sum Fwd Header Len	0.346346025
4	Sum TotLen Bwd Pkts	0.345087273
5	Sum Bwd Header Len	0.337106148
6	Max Fwd Header Len	0.32391613
7	Sum TotLen Fwd Pkts	0.309791295
8	Max Tot Fwd Pkts	0.285359729
9	Max Bwd Header Len	0.281949122
10	Sum Tot Fwd Pkts	0.273975118
11	Sum Tot Bwd Pkts	0.272993573
12	Sum Init Fwd Win Byts	0.260047838
13	Sum SYN Flag Cnt	0.253579661
14	Max Init Fwd Win Byts	0.253139496
15	Max Tot Bwd Pkts	0.251793929
16	Max SYN Flag Cnt	0.250833604
17	Num of flow	0.237716803
18	Sum Fwd IAT Tot	0.224904239
19	Sum ACK Flag Cnt	0.222165099
20	Max Init Bwd Win Byts	0.217576474
21	Sum Init Bwd Win Byts	0.216794286
22	Max Fwd IAT Tot	0.216773193
23	Max ACK Flag Cnt	0.213157528
24	Max TotLen Bwd Pkts	0.207378135
25	Max PSH Flag Cnt	0.207333195

26	Max Bwd IAT Tot	0.200159166
27	Sum PSH Flag Cnt	0.199239229
28	Sum Bwd IAT Tot	0.195212525
29	Max Fwd Act Data Pkts	0.184307592
30	Sum Fwd Act Data Pkts	0.171449815
31	Max TotLen Fwd Pkts	0.160311281
32	Sum RST Flag Cnt	0.010274675
33	Max RST Flag Cnt	0.007822328
34	Sum Bwd URG Flags	0.006036142
35	Max Bwd URG Flags	0.005970236
36	Max FIN Flag Cnt	0.005551485
37	Max Bwd PSH Flags	0.004969663
38	Sum CWE Flag Count	0.004009674
39	Sum URG Flag Cnt	0.003913788
40	Max ECE Flag Cnt	0.00386521
41	Sum Fwd URG Flags	0.001662337
42	Max Fwd PSH Flags	0.001566316
43	Sum Bwd PSH Flags	0.001318452
44	Max CWE Flag Count	0.0000960228779700589
45	Sum Fwd PSH Flags	0
46	Max Fwd URG Flags	0
47	Sum FIN Flag Cnt	0
48	Max URG Flag Cnt	0
49	Sum ECE Flag Cnt	0

Thông qua thực nghiệm, luận văn đã loại bỏ 35 đặc trưng và trích chọn được14 đặc trưng có độ ảnh hưởng lớn nhất là đầu vào cho các bộ phân loại ở bước tiếp theo.

Bảng 2.5. Kết quả thực nghiệm trích chọn đặc trưng luồng mạng

Số lượng thuộc	**Bagging**	**ADABoost**	**GradientBoosting**	**Random Forest**

tính loại bỏ				
5	0.8997	0.8992	0.8997	0.9001
6	0.8997	0.8992	0.8997	0.9001
7	0.8997	0.8992	0.8997	0.9001
…	…	…	…	…
26	0.8997	0.8992	0.8997	0.8998
27	0.8997	0.8992	0.8997	0.8999
28	0.8997	0.8992	0.8997	0.9007
29	0.8997	0.8992	0.8997	0.9002
30	0.8997	0.8992	0.8997	0.9002
31	0.8997	0.8992	0.8997	0.9002
32	0.8997	0.8992	0.8997	0.9002
33	0.8997	0.8992	0.8997	0.9007
34	0.8997	0.8992	0.8997	0.9007
35	0.8997	0.8992	0.8997	0.9007
36	0.8997	0.8992	0.8579	0.9000
37	0.8997	0.8992	0.8579	0.8996
38	0.8997	0.8412	0.8579	0.8996
39	0.8579	0.8412	0.8579	0.8756

- Đối với đặc trưng hiệu năng hệ thống:

Kết quả lựa chọn đặc trưng được trình bày ở bảng 2.6. Tương tự như trên, vẫn có một số đặc trưng chỉ có 1 giá trị duy nhất trong cả bộ dữ liệu huấn luyện và chắc chắn sẽ bị loại bỏ.

Bảng 2.6. Kết quả lựa chọn đặc trưng đối với dữ liệu sử dụng tài nguyên hệ thống

STT	Đặc trưng	Giá trị thông tin chung
1	swap_cache_min	0.510173671
2	swap_cache_mean	0.5050600104371805
3	swap_cache_max	0.498094775282484
4	mem_used_max	0.4682006921112478
5	mem_free_min	0.44960384835297584
6	mem_buffers_mean	0.4116367593785524
7	mem_used_mean	0.39160181365695945
8	mem_free_mean	0.37941985079589613

9	mem_buffers_max	0.36702565545971266
10	mem_buffers_min	0.3566475702951071
11	mem_used_min	0.3513761994707427
12	mem_free_max	0.3457422988426153
13	swap_cache_std	0.25775071501529534
14	cpu_%_id_mean	0.243891296447309
15	cpu_%_us_std	0.2428158822101545
16	num_total_sleeping_mean	0.22561910873929514
17	cpu_%_sy_std	0.22061873437034207
18	num_total_sleeping_std	0.21578443811444448
19	num_total_running_std	0.1966289112520896
20	cpu_%_si_mean	0.18947732179435373
21	mem_used_std	0.1839097844563511
22	mem_buffers_std	0.18360125334328004
23	mem_free_std	0.18239427896344584
24	mem_total_max	0.17863128578005938
25	cpu_%_us_mean	0.17328371073594506
26	cpu_%_id_std	0.17071013972288296
27	cpu_%_id_min	0.1671489873452583
28	cpu_%_sy_mean	0.1669520663549513
29	mem_total_mean	0.16643424937832774
30	mem_total_min	0.16584211482365552
31	swap_total_mean	0.16211295718250573
32	swap_used_min	0.1615541425073912
33	swap_used_max	0.16114490776217094
34	swap_total_max	0.15774251489085955
35	swap_used_mean	0.15698069426121397
36	swap_total_min	0.1562099656595004
37	num_total_running_mean	0.15597401722722304

38	num_total_sleeping_max	0.15263247480499942
39	cpu_%_id_max	0.1474418118513614
40	num_total_sleeping_min	0.1427537584707077
41	cpu_%_si_std	0.1270362115990633
42	cpu_%_us_max	0.10295535267397304
43	cpu_%_si_max	0.09414795844826629
44	cpu_%_us_min	0.08981474666952849
45	cpu_%_sy_max	0.08624582096029121
46	cpu_%_si_min	0.07506520008779982
47	num_total_zombie_mean	0.06874427881613965
48	num_total_zombie_std	0.06433460383002698
49	cpu_%_sy_min	0.060269431721807765
50	cpu_%_wa_mean	0.02212752687258357
51	cpu_%_wa_max	0.018779986
52	cpu_%_wa_std	0.017766649
53	num_total_zombie_max	0.017054818
54	num_total_running_max	0.012111471
55	cpu_%_ni_mean	0.008975781
56	num_total_zombie_min	0.005855758
57	swap_free_mean	0.004319841
58	cpu_%_hi_std	0.004089593
59	num_total_stopped_min	0.003677032
60	cpu_%_st_mean	0.003309134
61	cpu_%_st_std	0.002752953
62	cpu_%_ni_max	0.002724151
63	swap_free_max	0.002162291
64	num_total_stopped_max	0.00181426
65	swap_free_min	0.000719225
66	cpu_%_wa_min	0.000313534

67	num_total_running_min	0
68	num_total_stopped_mean	0
69	num_total_stopped_std	0
70	cpu_%_ni_std	0
71	cpu_%_ni_min	0
72	cpu_%_hi_mean	0
73	cpu_%_hi_max	0
74	cpu_%_hi_min	0
75	cpu_%_st_max	0
76	cpu_%_st_min	0
77	mem_total_std	0
78	swap_total_std	0
79	swap_used_std	0
80	swap_free_std	0

Thông qua thực nghiệm, luận văn đã loại bỏ 62 đặc trưng và chọn 18 đặc trưng có độ ảnh hưởng lớn nhất là đầu vào cho các bộ phân loại ở bước tiếp theo.

Bảng 2.7. Kết quả thực nghiệm trích chọn đặc trưng dữ liệu sử dụng tài nguyên thiết bị

Số lượng thuộc tính loại bỏ	**Bagging**	**ADABoost**	**GradientBoosting**	**Random Forest**
14	0.9884	0.9865	0.9865	0.9881
15	0.9884	0.9865	0.9865	0.9881
16	0.9884	0.9865	0.9865	0.9881
17	0.9884	0.9865	0.9884	0.9881
18	0.9884	0.9865	0.9884	0.9881
19	0.9884	0.9865	0.9884	0.9881
20	0.9884	0.9865	0.9884	0.9881
21	0.9884	0.9865	0.9884	0.9881
22	0.9884	0.9865	0.9884	0.9881
23	0.9884	0.9865	0.9884	0.9881
24	0.9884	0.9865	0.9884	0.9881
25	0.9884	0.9865	0.9884	0.9881
…	…	…	…	…
57	0.9884	0.9865	0.9884	0.9908
58	0.9884	0.9865	0.9884	0.9865

59	0.9884	0.9865	0.9884	0.9908
60	0.9884	0.9865	0.9884	0.9865
61	0.9884	0.9865	0.9884	0.9908
62	0.9884	0.9865	0.9884	0.9908
63	0.9884	0.9865	0.9884	0.9884
64	0.9884	0.9865	0.9884	0.9884
65	0.9865	0.9865	0.9865	0.9884
66	0.9865	0.9865	0.9865	0.9884
67	0.9865	0.9865	0.9865	0.9908
68	0.9865	0.9865	0.9865	0.9865
69	0.9865	0.9865	0.9865	0.9884
70	0.9865	0.9865	0.9865	0.9908
71	0.9764	0.9764	0.9865	0.9908
72	0.9764	0.9764	0.9855	0.9878
73	0.9763	0.9762	0.9855	0.9904
74	0.9763	0.9762	0.9855	0.9877

2.2.4. Bộ tổng hợp dự đoán

Quá trình kết hợp kết quả phát hiện các bộ phân loại học máy khác nhau cần đến sự tham gia của một hàm tổng hợp. Trong học máy, các hàm tổng hợp kết quả phổ biến được sử dụng như Voting, Stacking, Bagging và Boosting. Luận văn lựa chọn phương pháp bầu chọn (Voting) để áp dụng cho bài toán phát hiện sớm của mình dựa trên kết quả thực nghiệm. Kết quả cuối cùng của một dự đoán được thực hiện bởi đa số "phiếu bầu" theo hai chiến lược khác nhau là biểu quyết cứng (hard voting) và biểu quyết mềm (soft voting).

- Biểu quyết cứng: Trong biểu quyết cứng (còn được gọi là biểu quyết đa số), mỗi bộ phân loại riêng lẻ đưa ra dự đoán dữ liệu đầu vào thuộc nhãn nào và nhãn chiếm đa số dự đoán sẽ chiến thắng.

$$\hat{y} = mode\{C_1(x), C_2(x), \dots, C_m(x)\}$$

Trong đó: $C_i(x)$ là kết quả dự đoán (nhãn) của bộ phân lớp thứ j

- Biểu quyết mềm: Trong biểu quyết mềm, mỗi bộ phân loại riêng lẻ đưa ra xác suất dự đoán dữ liệu đầu vào thuộc về từng nhãn tương ứng. Các giá trị dự đoán được đánh trọng số theo mức độ quan trọng của bộ phân loại và được tổng hợp lại. Sau đó, nhãn có tổng xác suất có trọng số lớn nhất sẽ được chọn.

$$\hat{y} = arg \max_{i} \sum_{j=1}^{m} w_j p_{ij}$$

Trong đó: w_i là trọng số của kết quả dự đoán (giá trị p) thuộc bộ phân lớp thứ j.

Trong phần thực nghiệm đánh giá được trình bày tại chương 3, luận văn sử dụng biểu quyết mềm vì các bộ phân loại đơn lẻ được thực nghiệm đều có đầu ra là xác suất dự đoán. Ngoài ra biểu quyết mềm xem xét xác suất, mức độ chắc chắn của từng bộ phân loại đơn lẻ rồi mới quyết định nên sẽ đem lại kết quả chính xác hơn so với với việc chỉ xem xét nhãn dự đoán của từng bộ phân loại như biểu quyết cứng.

Kết luận chương

Trong chương này, luận văn đã ứng dụng mô hình tổng quan cho bài toán phát hiện sớm mã độc IoT Botnet, mô tả cụ thể về chức năng, kỹ thuật sử dụng và mục đích của các thành phần chính. Mô hình sử dụng môi trường sandbox để thu thập 3 loại dữ liệu đặc trưng dựa trên những nền tảng nghiên cứu của các bên nghiên cứu liên quan đến chủ đề được đề cập. Những dữ liệu này sau đó được phân tích và xử lý phù hợp để đưa vào các bộ phân loại. Mô hình cộng tác được ứng dụng để giải quyết bài toán phát hiện với sự kết hợp để tăng hiệu năng cho các bộ phân loại đơn lẻ. Kết quả triển khai thực nghiệm mô hình với tập dữ liệu thực tế và đánh giá so sánh với các nghiên cứu đã công bố được trình bày trong chương tiếp theo của luận văn.

CHƯƠNG 3: THỰC NGHIỆM VÀ ĐÁNH GIÁ

Trong chương này sẽ trình bày về tập dữ liệu sử dụng cho quá trình thực nghiệm, các bước thực thi cụ thể từ mô hình tổng quan để đưa ra kết quả từ tập dữ liệu đầu vào. Sau khi thu được kết quả thực nghiệm sẽ thực hiện đánh giá độ hiệu quả của mô hình, các so sánh kết quả với các nghiên cứu đã có và từ đó đề ra phương hướng nghiên cứu phát triển sau này cho bài toán.

3.1. Bộ dữ liệu

Để đánh giá kết quả hoạt động của mô hình ứng dụng, luận văn sử dụng tập dữ liệu do tác giả Lê Hải Việt thu thập và chia sẻ [40]. Tập dữ liệu này chứa 8911 mẫu bao gồm 5023 IoT Botnet và 3888 mẫu lành tính đã được thu thập và sử dụng cho thực nghiệm. Chia tập dữ liệu theo tỉ lệ 70% bộ dữ liệu dùng để huấn luyện các mô hình và 30% bộ dữ liệu để kiểm nghiệm độ hiệu quả. Mô tả về các mẫu trong tập dữ liệu được mô tả trong bảng 3.1 và minh họa thống kê dựa trên 3 loại dữ liệu trong hình 3.1-3.3.

Bảng 3.1. Mô tả bộ dữ liệu

Loại	**Số lượng**
Mã độc Bashlite	2786
Mã độc Mirai	1510
Mã độc IoT Botnet khác (MrBlack, Spike, Dofloo,…)	727
Tệp lành tính	3888

name	net
e030450160d68521df95dd2f29e824361e42ae29861ebf93c3c92fdd0885d3c1_1590922273	0
ab48c1756b57ae2ed003a7034cc10033_1590893149	10677
561c08abf0461fff854aea267defb48f_1590665806	104
d6faf2d4f4e853fabf182114eb098a767eda829617973de3596e376d74b5a315_1590837872	7488
5a579399bcf7a1c61d851e26a768bc14d423ee6b9ce98d0e89829c92bfd2da64_1590701213	28
4879c5039f56d58c58d1d0ace460af8129a643fd6cbf239493620a2d54fec6e6_1590527576	352
b50979ccb8cc6ecaf6b765620950726f_1590973969	23685
7ed8de4d70ad33f419a505dc62d11ba1225c98b3a59d1bb9b360aeeb6e5cfc6e_1590829459	154
c3cf3d4f6a47fff7f9618ebf04be4407fb1eb4383673a723c21727b8ee95a15e_1590961890	1

Hình 3.1. Minh họa thống kê tập dữ liệu về luồng mạng

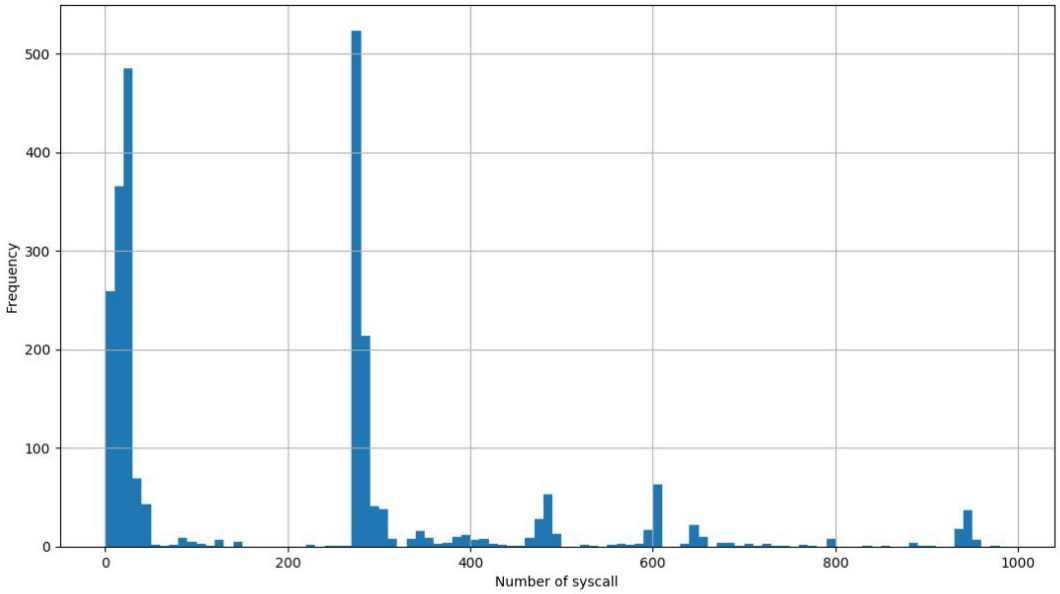
name	syscall
e030450160d68521df95dd2f29e824361e42ae29861ebf93c3c92fdd0885d3c1_1590922273	20
ab48c1756b57ae2ed003a7034cc10033_1590893149	271
561c08abf0461fff854aea267defb48f_1590665806	5322
d6faf2d4f4e853fabf182114eb098a767eda829617973de3596e376d74b5a315_1590837872	10864
5a579399bcf7a1c61d851e26a768bc14d423ee6b9ce98d0e89829c92bfd2da64_1590701213	886
4879c5039f56d58c58d1d0ace460af8129a643fd6cbf239493620a2d54fec6e6_1590527576	494
b50979ccb8cc6ecaf6b765620950726f_1590973969	274
7ed8de4d70ad33f419a505dc62d11ba1225c98b3a59d1bb9b360aeeb6e5cfc6e_1590829459	283

Hình 3.2. Minh họa thống kê về tập dữ liệu lời gọi hệ thống

name	per
e030450160d68521df95dd2f29e824361e42ae29861ebf93c3c92fdd0885d3c1_1590922273	2
ab48c1756b57ae2ed003a7034cc10033_1590893149	84
561c08abf0461fff854aea267defb48f_1590665806	76
d6faf2d4f4e853fabf182114eb098a767eda829617973de3596e376d74b5a315_1590837872	177
5a579399bcf7a1c61d851e26a768bc14d423ee6b9ce98d0e89829c92bfd2da64_1590701213	17
4879c5039f56d58c58d1d0ace460af8129a643fd6cbf239493620a2d54fec6e6_1590527576	125
b50979ccb8cc6ecaf6b765620950726f_1590973969	86
7ed8de4d70ad33f419a505dc62d11ba1225c98b3a59d1bb9b360aeeb6e5cfc6e_1590829459	60
c3cf3d4f6a47fff7f9618ebf04be4407fb1eb4383673a723c21727b8ee95a15e_1590961890	2

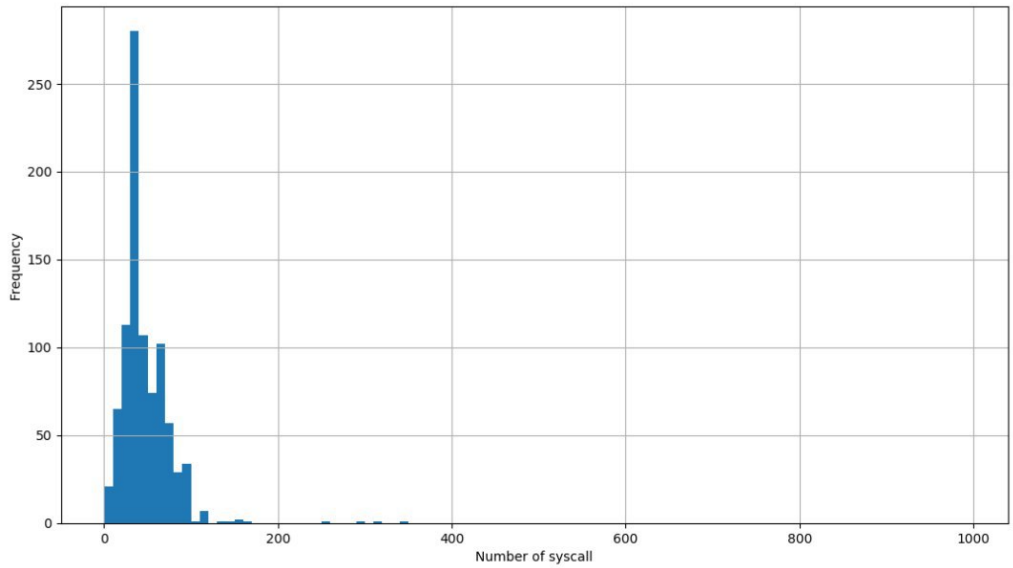
Hình 3.3. Minh họa thống kê về tập dữ liệu sử dụng hiệu năng của hệ thống

Luận văn tiến hành phân tích, thống kê chi tiết và vẽ biểu đồ thống kê của các dữ liệu mã độc cũng như lành tính trong bộ dữ liệu để tạo cơ sở cho việc xác định ngưỡng phát hiện sớm như đã trình bày ở chương 2: 300 lời gọi hệ thống, 20 trạng thái tài nguyên hệ thống và 50 gói tin luồng mạng đầu tiên. Các kết quả thống kê như hình dưới.



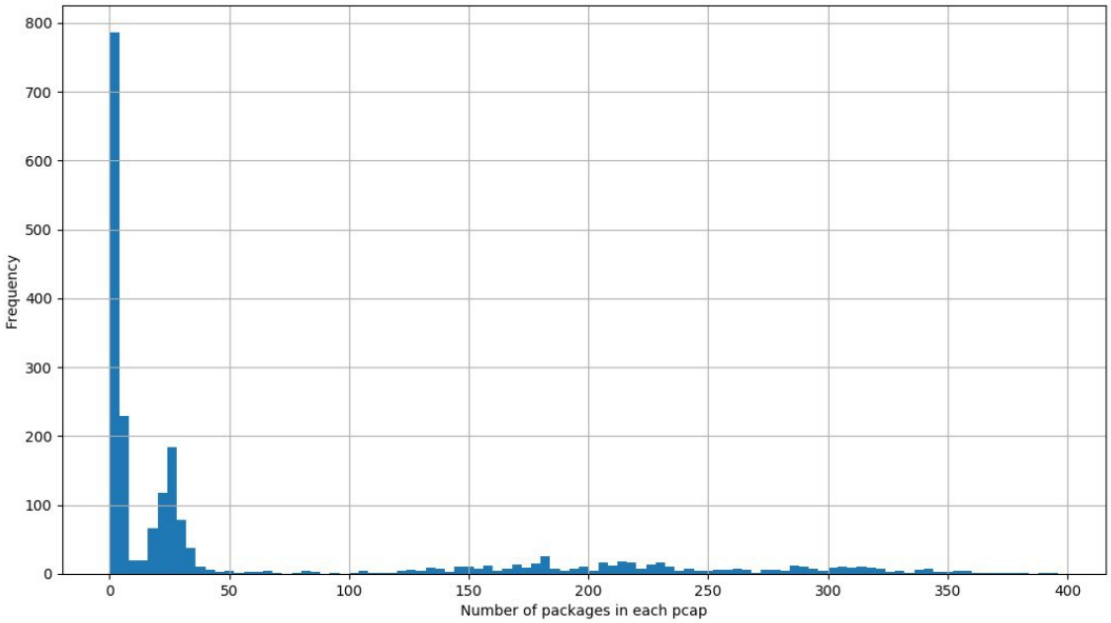
Hình 3.4. Thống kê số lượng lời gọi hệ thống của mã độc IoT Botnet

Tiến hành phân tích số lượng lời gọi hệ thống của các mẫu trong Dataset trong hình 3.4 và 3.5, luận văn nhận thấy các mã độc IoT Botnet chủ yếu thường thể hiện hành vi hoạt động dưới ngưỡng 300 lời gọi hệ thống, trong khi số lượng lời gọi hệ thống của các tệp lành tính thể hiện ở hình 3.5 tập trung chủ yếu dưới 100 lời gọi hệ thống.

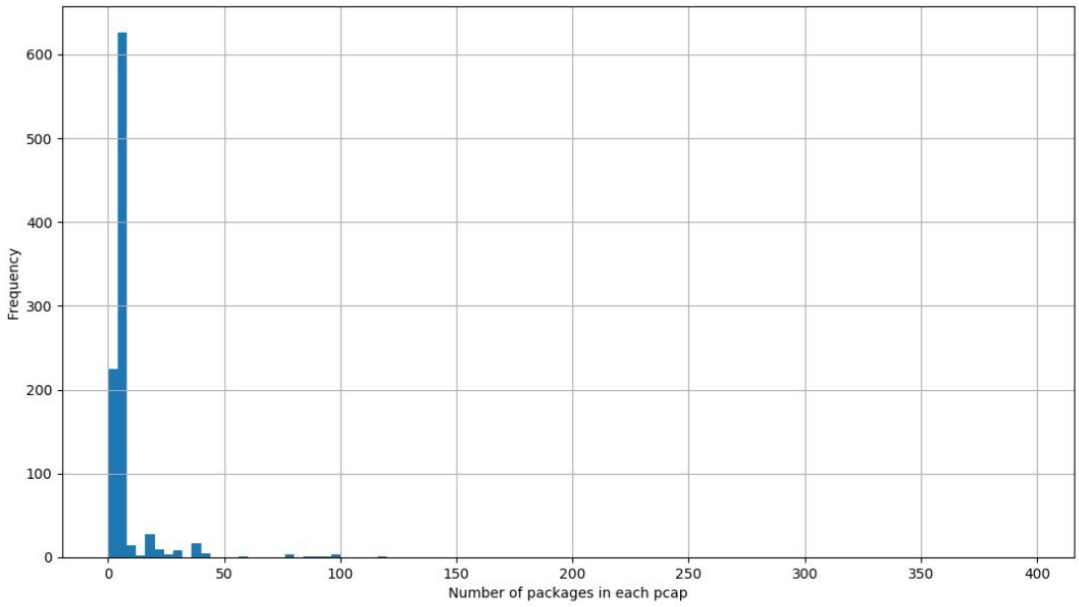


Hình 3.5. Thống kê số lượng lời gọi hệ thống của tệp lành tính

Hình 3.6 cho thấy lượng gói tin giao tiếp trong luồng mạng của IoT Botnet tập trung chủ yếu dưới 50 gói tin và có một số lượng mẫu IoT Botnet có thể lên đến 400 gói tin giao tiếp luồng mạng, trong khi đó lượng gói tin của các tệp lành tính thể hiện trong hình 3.7 gần như chỉ dưới ngưỡng 50 gói tin.

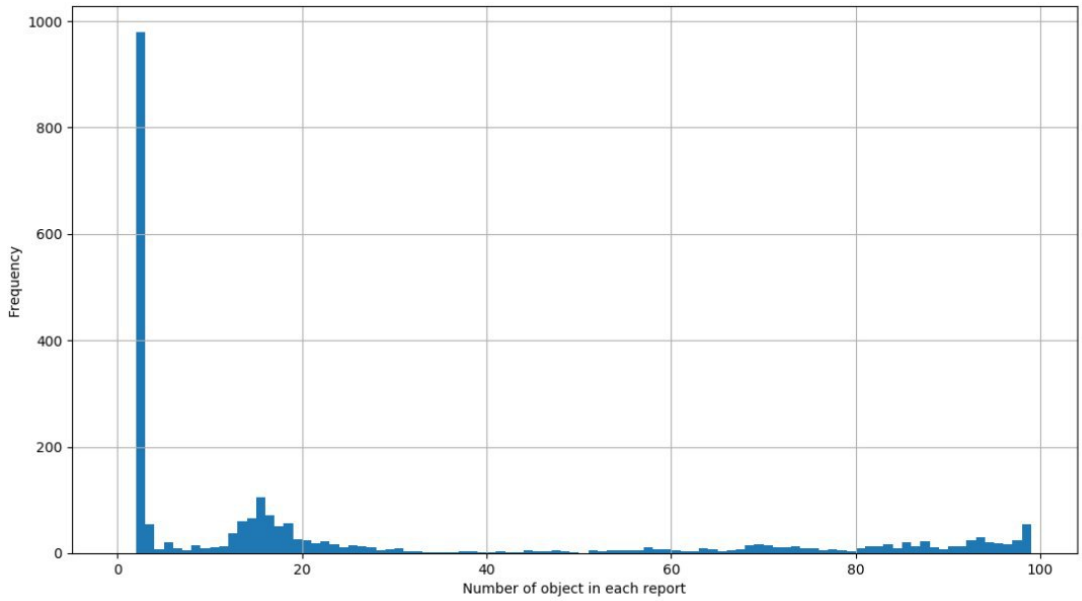


Hình 3.6. Thống kê số lượng gói tin mạng của IoT Botnet

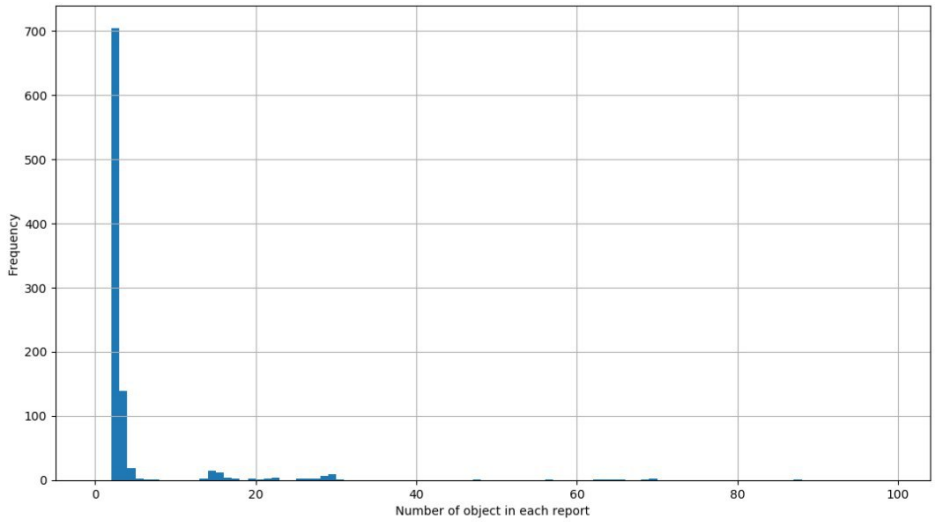


Hình 3.7. Thống kê số lượng gói tin mạng của tệp lành tính

Về thông tin yêu cầu sử dụng tài nguyên của hệ thống, hình 3.8 cho thấy các mẫu IoT Botnet thường có lượng yêu cầu sử dụng tài nguyên hệ thống cao, có thể lên đến 100 yêu cầu, nhưng chủ yếu lượng lớn vẫn tập trung ở ngưỡng 20 yêu cầu sử dụng tài nguyên. Hình 3.9 thể hiện các tệp lành tính thường không yêu cầu sử dụng tài nguyên nhiều như các mẫu IoT Botnet, các mẫu này đều có dưới 20 yêu cầu sử dụng tài nguyên hệ thống.



Hình 3.8. Thống kê yêu cầu chiếm dụng tài nguyên hệ thống của IoT Botnet



Hình 3.9. Thống kê yêu cầu chiếm dụng tài nguyên hệ thống của tệp lành tính

Tập dữ liệu sau khi được đưa vào bộ phận tiền xử lý và chuẩn hóa sẽ có các đặc trưng được biểu diễn như trong hình 3.10 - 3.12.

name	Num of flow	Sum Flow Duratio	Max Flow Duratic	Sum Tot Fwd Pkts	Max Tot Fwd Pkts	Sum Tot Bwd Pkts	Max Tot Bwd Pkts	Sum TotLen Fwd I	Max TotLen Fwd I	Sum TotLen Bwd	Max TotLen Bwd
06916bd06712bc6652ee54782508da93_1593707152	6	3642	1498	6	1	6	1	270	46	270	46
ca90ad8fefe28c724c2736984d6e95ec_1590180585	2	1844	1508	2	1	2	1	90	46	90	46
017662f941ebe50c7e0d3fd9a826144a_1589905104	2	2254	1644	2	1	2	1	90	46	90	46
9a983419ddc7ab33507b41e3685f33bf_1590173095	2	2017	1704	2	1	2	1	90	46	90	46
888ccce705a9450caef48e6cf7901781_1590141182	0	0	0	0	0	0	0	0	0	0	0
740cea9b9bdbb761c49270e72f4b77db_1589981287	6	7683	1806	6	1	6	1	270	46	270	46
f1085b90bdd41b23bb18e9a457e3a266_1593698468	2	1839	990	2	1	2	1	90	46	90	46
ade13c6d0182c53febc9b8486296e483_1590143440	0	0	0	0	0	0	0	0	0	0	0
7535ff0d990b6cfc8d20dc06f3a7201a_1593511352	2	1724	1478	2	1	2	1	90	46	90	46
e4c17224fd574e62ad0e8ae2a8306a26_1590216459	2	1876	1716	2	1	2	1	90	46	90	46
c5ce05ebee11ea7c06dc3e4bdd853931_1590172542	2	2345	1955	2	1	2	1	90	46	90	46
8ca7ec7b22b8bc8fc436672a62c1f7e9_1593691094	2	1755	1526	2	1	2	1	90	46	90	46
bf07a7a21d7ffa99bb8ab4035db5cdf6_1590162833	2	11074	9801	2	1	2	1	90	46	90	46
a553e97d88f7b7934276856f1a528716_1590132757	2	1727	1443	2	1	2	1	90	46	90	46
91d9701615b7b179a1b7eb666a490374_1590158787	2	10959	10624	2	1	2	1	90	46	139	95
318c43200fc8c71c24ca592e2eda9c9c_1589946963	2	2633	1357	2	1	2	1	90	46	90	46
9ab79e3cf393ae7506653cf2e485268a_1593688609	2	1488	1252	2	1	2	1	90	46	90	46
462f4c4aebb2a55141264a4a52d54144_1593685252	2	3258	2349	2	1	2	1	90	46	90	46
0056d6b61f45430102109b1e241dbdc8_1589902117	2	1907	1577	2	1	2	1	90	46	90	46
783692b89d729bd25d268e063028a93b_1593656486	0	0	0	0	0	0	0	0	0	0	0
456c81820c4bbbd795d238c28a7b5709_1589974130	2	12730	10172	2	1	2	1	90	46	90	46
9484c862cb0e7915fd2a1c0e36d7a917_1590163775	0	0	0	0	0	0	0	0	0	0	0

Hình 3.10. Bộ đặc trưng dữ liệu luồng mạng chuẩn hóa theo mô tả đặc trưng của bộ dữ liệu mạng CSE-CIC-IDS2018

name	num_total	num_total	num_total	num_total	num_total	num_total	num_total	num_total	num_total	num_total	num_total	num_total	num_total	num_total	num_total	cpu_%_us	cpu_%_us	cpu_%_us	cpu_%_us
06916bd06712bc6652ee54782	2.444444	1.342561	5	1	82.88889	1.099944	86	82	0	0	0	0	0	0	0	26.18889	1.099439	27.9	24.3
ca90ad8fefe28c724c2736984d	2.5	1.5	4	1	56.5	0.5	57	56	0.5	0.5	1	0	0	0	0	59.65	0.15	59.8	59.5
017662f941ebe50c7e0d3fd9a8	2.5	1.5	4	1	56.5	0.5	57	56	0	0	0	0	0	0	0	59.2	0.1	59.3	59.1
9a983419ddc7ab33507b41e36	3	2.160247	6	1	56.66667	0.942809	58	56	0	0	0	0	0	0	0	59.06667	0.20548	59.3	58.8
888ccce705a9450caef48e6cf7	3	2	5	1	50	0	50	50	0.5	0.5	1	0	0	0	0	59.2	0.1	59.3	59.1
740cea9b9bdbb761c49270e72f	2.533333	1.203698	5	1	52.93333	1.236482	54	50	0	0	0	0	0	0	0	59.96	0.54626	60.7	59
f1085b90bdd41b23bb18e9a457	2.5	1.5	4	1	50.5	0.5	51	50	0.5	0.5	1	0	0	0	0	58.4	0.1	58.5	58.3
ade13c6d0182c53febc9b84862	3	2	5	1	51.5	1.5	53	50	0	0	0	0	0	0	0	59.4	0.1	59.5	59.3
7535ff0d990b6cfc8d20dc06f3a	2.666667	0.942809	4	2	57	1.414214	59	56	0	0	0	0	0	0	0	60	0.244949	60.3	59.7
e4c17224fd574e62ad0e8ae2a8	3.5	0.5	4	3	56	1	57	55	0.5	0.5	1	0	0	0	0	59.75	0.15	59.9	59.6
c5ce05ebee11ea7c06dc3e4bdd	2	1	3	1	50.5	0.5	51	50	0.5	0.5	1	0	0	0	0	59.05	0.05	59.1	59
8ca7ec7b22b8bc8fc436672a62	2.75	0.829156	4	2	80.75	0.829156	82	80	0	0	0	0	0	0	0	24.675	0.258602	25	24.3
bf07a7a21d7ffa99bb8ab4035d	2	1	3	1	56.5	0.5	57	56	0	0	0	0	0	0	0	59.7	0.1	59.8	59.6
a553e97d88f7b7934276856f1a	2.5	0.5	3	2	56	1	57	55	0	0	0	0	0	0	0	59.7	0.1	59.8	59.6
91d9701615b7b179a1b7eb666	4	1	5	3	50.5	1.5	52	49	0	0	0	0	0	0	0	58.15	0.05	58.2	58.1
318c43200fc8c71c24ca592e2e	2.5	0.5	3	2	56.5	0.5	57	56	0	0	0	0	0	0	0	59.9	0.1	60	59.8
9ab79e3cf393ae7506653cf2e4	2.75	1.47902	5	1	57	1.224745	59	56	0	0	0	0	0	0	0	60.125	0.192029	60.4	59.9
462f4c4aebb2a55141264a4a52	2.666667	1.699673	5	1	50.66667	0.942809	52	50	0	0	0	0	0	0	0	58.06667	0.094281	58.2	58
0056d6b61f45430102109b1e24	3	2	5	1	57	1	58	56	0	0	0	0	0	0	0	59.85	0.15	60	59.7
783692b89d729bd25d268e063(	2	1	3	1	51	1	52	50	0.5	0.5	1	0	0	0	0	58.55	0.05	58.6	58.5
456c81820c4bbbd795d238c28a	2	1	3	1	51.5	1.5	53	50	0	0	0	0	0	0	0	58.8	0.1	58.9	58.7
9484c862cb0e7915fd2a1c0e36	3	2	5	1	57.5	1.5	59	56	0	0	0	0	0	0	0	52.75	0.05	52.8	52.7
a4d8e93df4f3b9aeff6c61593aa	2.5	1.5	4	1	50	0	50	50	0	0	0	0	0	0	0	58.8	0.1	58.9	58.7
52ee76e2bb81ea0dbac6ca2278	3	2	5	1	56.5	0.5	57	56	0.5	0.5	1	0	0	0	0	59.95	0.15	60.1	59.8
f308903a85b05334d5aefb7a64	2.5	1.5	4	1	50.5	0.5	51	50	0.5	0.5	1	0	0	0	0	58.75	0.05	58.8	58.7

Hình 3.11. Bộ đặc trưng dữ liệu sử dụng tài nguyên chuẩn hóa theo đầu ra của V-Sandbox

name	#0	#1	#2	#3	#4	#5	#6	#7	#8	#9	#10	#11	#12	#13	#14	#15	#16	#17	#18	#19
ca422e82a753cc77fd39359	0.417392	-0.03792	0.614784	-0.10077	0.274377	0.03604	0.775788	0.163963	-0.07387	0.201733	0.231688	-0.02996	-0.19174	0.452063	0.362744	-0.33363	0.206744	0.406058	0.407741	0.028207
75b425d4c17e38c482adbb9	-0.14718	0.299331	0.463711	-0.30361	0.04424	0.075229	-0.02438	0.200255	-0.38953	-0.12514	0.010705	0.209225	-0.42184	-0.03995	0.353882	-0.07386	-0.0258	0.256283	-0.17373	-0.20575
f31857095e18b0ac8a7a92d	-0.00176	-0.00063	-0.00314	-0.00252	-0.00297	-0.00145	0.000528	0.003748	-0.00043	-0.00285	-0.00064	-0.00112	-0.00237	-0.00289	-0.0005	0.003219	-0.00168	0.00141	-0.00225	0.000605
b10365c2ed158f42e0d3042	0.022033	-0.01266	0.446192	-0.4163	0.635272	0.437435	0.385394	0.080333	-0.56905	-0.00684	0.172779	-0.27382	-0.06186	-0.28334	0.176178	0.029936	-0.16532	0.089836	-0.05862	-0.15821
245c9cc38b412e8f44d3b92f	-0.0962	0.433907	1.011162	-0.06523	-0.13495	0.241593	-0.15654	0.367647	-0.10447	-0.03408	0.226747	0.08468	-0.53062	-0.46443	0.395556	-0.30085	-0.12144	0.122074	0.04191	0.204901
d5728c795024c3b222e47fa	-0.28907	0.154219	0.415879	-0.30724	0.029628	0.362739	-0.0734	-0.29989	-0.47756	-0.025	0.09332	0.12423	-0.1446	0.112502	0.43597	0.005839	-0.31836	0.18118	0.051933	-0.04126
5d9dbea9e5282a888efc534	-0.01318	0.353012	0.607745	-0.17214	-0.56273	-0.56886	-0.05968	-0.27012	-0.08558	-0.42347	-0.41621	0.161724	-0.23135	-0.18187	0.657714	-0.20444	-0.12053	-0.03243	-0.14843	-0.54647
b71664df5f5e98b55bb9456	0.081088	1.337151	0.361997	-1.40365	0.075783	0.212316	-0.36495	0.543099	-0.41076	-0.08299	-0.18544	-0.45737	0.458539	0.219077	1.182909	0.083035	-0.07689	0.784917	0.143437	-0.58269
d03fb405ac5bbdd7e9408b2	-0.25357	0.193825	0.375461	-0.30972	-0.01456	0.353501	-0.08779	-0.28084	-0.41776	0.012673	0.140692	0.159382	-0.20381	0.097105	0.406815	-0.01846	-0.28706	0.191409	-0.01246	-0.069
9ad8153341fd7ec5ceff4e66	0.225042	0.519073	0.769218	0.32508	0.162861	0.437172	0.027757	-0.11979	-0.50686	0.203949	-0.11293	0.18567	-0.37332	0.106478	0.635276	-0.24322	-0.16033	0.328741	-0.22825	-0.16233
f673d8017239bc44665dd34	0.255746	0.237762	0.613032	-0.1865	0.587081	0.205936	0.183963	-0.28986	-0.09969	0.029589	-0.03962	0.271528	-0.05857	-0.12939	0.369428	-0.27796	-0.04009	0.460018	-0.35985	0.224411
ea9eb02384e7c682959baa2	-0.14334	0.351151	0.474507	-0.35802	0.055298	0.110998	-0.06229	0.12844	-0.42126	-0.10658	0.048058	0.204316	-0.42517	-0.07929	0.349108	-0.14733	-0.02902	0.300404	-0.16249	-0.19764
9607542fb3a1c70d5687378	0.034741	0.028888	0.526335	-0.21137	-0.0243	-0.23027	0.145494	0.162151	0.007353	0.249399	0.347631	0.261706	-0.28469	-0.05473	0.662704	-0.15386	-0.17961	0.205242	0.06738	-0.25302
5b7b5ff335d4886f2dec647a	-0.10835	0.090265	0.854913	-0.68474	0.007111	-0.02896	-0.19689	-0.05032	-0.20575	-0.58587	-0.21166	-0.01845	-0.04184	-0.00576	0.800238	-0.12428	-0.17744	0.230784	-0.10066	-0.30897
38f4b61c190969bd54a7b47	0.03743	0.233955	0.697677	-0.29646	-0.33348	0.125354	-0.34608	0.20325	-0.37535	-0.24818	-0.36015	0.169096	-0.13553	-0.06391	0.58914	0.104786	-0.29931	0.634703	-0.33271	-0.40689
e5727dec18a8c103a5738f2	-0.04429	0.022743	0.416712	-0.46799	0.625582	0.416693	0.347489	0.111736	-0.54602	0.00772	0.178264	-0.34786	-0.08633	-0.26232	0.171659	0.012985	-0.18913	0.057902	-0.043	-0.10319
c208f7d28574298f31a60c53	0.278494	0.091782	0.434803	-0.22934	-0.12764	-0.5359	0.756806	0.038701	-0.33608	0.017152	0.32207	-0.08773	-0.29326	-0.07098	0.349737	-0.13453	0.015086	0.42394	0.206366	-0.28545
74d9494e23ca1a69e0128a2	0.329144	0.2971	0.207542	-0.51156	0.212242	0.089371	0.061098	0.337489	-0.24993	0.063206	0.365313	0.169767	-0.28018	0.006353	0.665	0.092555	-0.10749	0.052168	0.158571	-0.08997
0c59360fca2ad71ff3ae71dc	0.042912	0.302755	0.973488	-0.20227	-0.02839	0.154709	-0.36625	0.14501	-0.06943	-0.28075	-0.19212	-0.03649	-0.34802	0.461677	0.786784	-0.36884	-0.06095	0.320535	0.030281	0.003158
e673d98a5b9267b2b71a521	-0.26521	0.017313	0.630079	-0.35302	0.179435	0.290078	-0.24522	0.321574	-0.39175	0.048162	-0.22297	0.322941	-0.01256	0.094246	0.859197	-0.29977	-0.12879	0.228327	0.33009	-0.02469
698221d41cc41273e535a15	-0.17145	0.351307	0.398695	-0.3529	0.081661	0.092789	-0.07096	0.195571	-0.44406	-0.12535	-0.0356	0.158482	-0.36695	-0.08849	0.371836	-0.09209	-0.09042	0.290106	-0.11213	-0.18462
6991b61d6b8ac639530e432	-0.3309	0.406627	0.854847	-0.34289	-0.21119	0.175671	-0.18708	-0.03287	-0.10974	-0.43092	-0.34616	0.048189	-0.15165	-0.1064	0.990737	-0.01803	-0.11722	0.277402	-0.0959	-0.54681
04ca6eb03cd704266f5518b	-0.00161	-0.00353	-0.00072	0.001656	0.000171	-0.00065	-0.00378	0.001249	-0.00025	-0.00066	-0.00094	-0.00011	0.002951	0.002875	0.003075	0.002832	-0.0006	0.002958	0.003743	-0.00051
d4e146cac920b23578c1d16	-0.21588	0.210603	0.494665	-0.29457	0.155683	0.207889	0.054948	0.048882	-0.46316	-0.10894	0.014314	0.142726	-0.28739	-0.19347	0.254745	-0.08465	-0.06045	0.141387	0.037188	-0.09083
6f77993cbe5a2f762d458d5	0.289063	0.305564	0.268342	-0.04281	0.349053	0.092081	0.381969	0.263415	-0.19793	0.24906	-0.16273	0.947429	-0.28368	0.269315	0.61912	-0.10736	0.192682	0.37383	-0.38285	0.04861
147af70b815093d9247e22f	0.014795	0.553409	0.16702	-0.30323	0.415719	-0.04275	0.178464	-0.04967	-0.09156	0.086259	-0.01112	-0.13627	0.266683	-0.17136	0.696277	-0.06352	0.046174	0.185606	-0.24118	-0.12011

Hình 3.12. Bộ đặc trưng dữ liệu lời gọi hệ thống trích xuất các đặc trưng từ đồ thị SCG thành vector đặc trưng

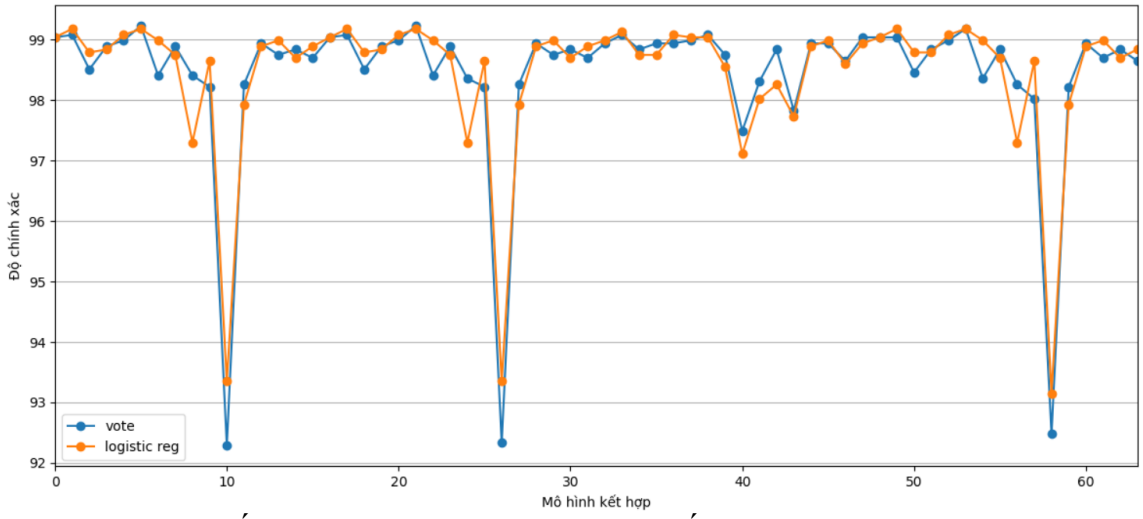
3.2. Môi trường triển khai thực nghiệm

Quá trình triển khai thực nghiệm để đánh giá mô hình đề xuất được tiến hành trên máy chủ với thông số kỹ thuật cơ bản gồm: Bộ vi xử lý AMD Ryzen 3.6 GHz, ổ cứng lưu trữ HDD dung lượng 1TB, bộ nhớ DDR3 32 GB. Luận văn đã tìm hiểu và sử dụng các thuật toán học máy thường được sử dụng hiện nay cho bài toán phát hiện mã độc như Bagging, ADABoost, Random Forest, GradientBoosting và các hàm hợp nhất khác nhau như biểu quyết hay hồi quy tuyến tính để chọn được phương pháp tối ưu nhất. Sử dụng các bộ phân loại hợp nhất tăng khả năng tổng hợp kết quả do các bộ phân loại này cũng xây dựng trên các thành phần học yếu hơn để đưa ra kết quả. Các thuật toán học máy đơn lẻ thử nghiệm Bagging, Random Forest, ADABoost, GradientBoosting,… được cài đặt thông qua ngôn ngữ Python với thư viện Scikit-learn (Sklearn) [50].

3.3. Kết quả thực nghiệm

Luận văn đã sử dụng các bộ phân loại tổng hợp như Bagging, Random Forest, ADABoost, GradientBoosting tạo được tổ hợp 64 cách kết hợp 4 thuật toán này và hàm hợp nhất bầu chọn, hồi quy tuyến tính để đánh giá hiệu quả của mô hình học máy cộng tác. Từ các cách kết hợp các bộ phân loại phổ biến cùng với sử dụng 2 phương pháp hợp nhất thu được 128 kết quả thử nghiệm đánh giá độ chính xác của việc phát hiện mã độc IoT Botnet cho mô hình ứng dụng được mô tả trong Hình 3.13 và Bảng 3.2.

Dựa vào kết quả thu được mô tả trong hình, ta có thể nhận thấy hàm hợp nhất bầu chọn có hiệu suất tốt tương đương với hàm hồi quy tuyến tính và bộ ba thuật toán phân loại học máy đơn lẻ sử dụng random forest trên 3 tập dữ liệu thuộc tính khác nhau cho kết quả tốt nhất với độ chính xác ACC = 99.23% sử dụng phương pháp bầu chọn. Các mô hình học máy đã được luận văn điều chỉnh các tham số trên toàn bộ dữ liệu và tiến hành đánh giá mô hình ứng dụng dựa trên các chỉ số thu được. Kết quả đánh giá mô hình ứng dụng trên bộ dữ liệu được trình bày trong bảng 3.2.



Hình 3.13. Kết quả đánh giá các phương án kết hợp thuật toán học máy

Bảng 3.2. Độ chính xác các bộ phân loại học máy đơn lẻ huấn luyện trên bộ dữ liệu và kết quả tổng hợp dự đoán

STT	Thuật toán	Độ chính xác phân loại dữ liệu của mô hình (ACC - %)				
		Dữ liệu luồng mạng	Dữ liệu sử dụng tài nguyên thiết bị	Dữ liệu lời gọi hệ thống	Bầu chọn	Hồi quy tuyến tính
1.	Bagging + Bagging + Bagging	89.97	98.84	97.2	99.04	99.04
2.	Bagging + Bagging + Random Forest	89.97	98.84	97.78	99.08	99.18
3.	Bagging + Bagging + AdaBoost	89.97	98.84	97.2	98.51	98.79
4.	Bagging + Bagging + Gradient Tree Boosting	89.97	98.84	97.69	98.89	98.84
5.	Bagging + Random Forest + Bagging	89.97	99.08	97.2	98.99	99.08
6.	Bagging + Random Forest + Random Forest	89.97	99.08	97.78	99.23	99.18
7.	Bagging + Random Forest + AdaBoost	89.97	99.08	97.2	98.41	98.99
8.	Bagging + Random Forest + Gradient Tree Boosting	89.97	99.08	97.69	98.89	98.75
9.	Bagging + AdaBoost + Bagging	89.97	98.65	97.2	98.41	97.3

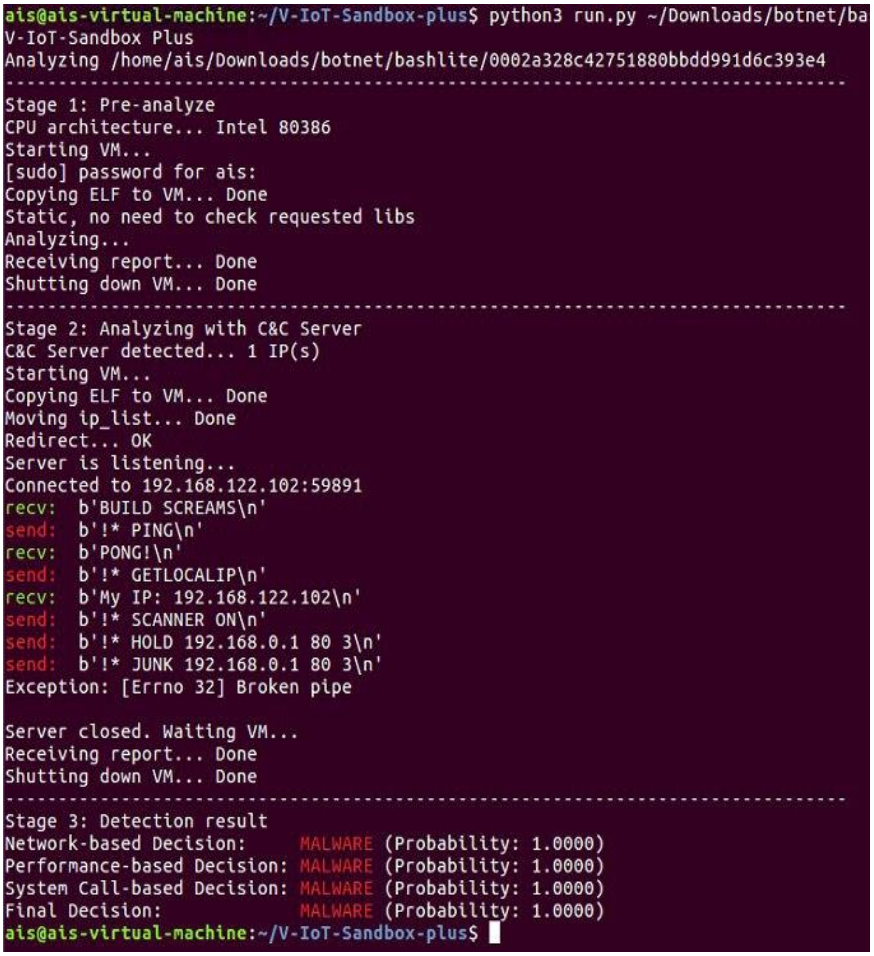
STT	Thuật toán	Độ chính xác phân loại dữ liệu của mô hình (ACC - %)				
		Dữ liệu luồng mạng	Dữ liệu sử dụng tài nguyên thiết bị	Dữ liệu lời gọi hệ thống	Bầu chọn	Hồi quy tuyến tính
10.	Bagging + AdaBoost + Random Forest	89.97	98.65	97.78	98.22	98.65
11.	Bagging + AdaBoost + AdaBoost	89.97	98.65	97.2	92.29	93.35
12.	Bagging + AdaBoost + Gradient Tree Boosting	89.97	98.65	97.69	98.26	97.93
13.	Bagging + Gradient Tree Boosting + Bagging	89.97	98.65	97.2	98.94	98.89
14.	Bagging + Gradient Tree Boosting + Random Forest	89.97	98.65	97.78	98.75	98.99
15.	Bagging + Gradient Tree Boosting + AdaBoost	89.97	98.65	97.2	98.84	98.7
16.	Bagging + Gradient Tree Boosting + Gradient Tree Boosting	89.97	98.65	97.69	98.7	98.89
17.	Random Forest + Bagging + Bagging	90.07	98.84	97.2	99.04	99.04
18.	Random Forest + Bagging + Random Forest	90.07	98.84	97.78	99.08	99.18
19.	Random Forest + Bagging + AdaBoost	90.07	98.84	97.2	98.51	98.79
20.	Random Forest + Bagging + Gradient Tree Boosting	90.07	98.84	97.69	98.89	98.84
21.	Random Forest + Random Forest + Bagging	90.07	99.08	97.2	98.99	99.08
22.	**Random Forest + Random Forest + Random Forest**	**90.07**	**99.08**	**97.78**	**99.23**	**99.18**
23.	Random Forest + Random Forest + AdaBoost	90.07	99.08	97.2	98.41	98.99
24.	Random Forest + Random Forest + Gradient Tree Boosting	90.07	99.08	97.69	98.89	98.75
25.	Random Forest + AdaBoost + Bagging	90.07	98.65	97.2	98.36	97.3

STT	Thuật toán	Độ chính xác phân loại dữ liệu của mô hình (ACC - %)				
		Dữ liệu luồng mạng	Dữ liệu sử dụng tài nguyên thiết bị	Dữ liệu lời gọi hệ thống	Bầu chọn	Hồi quy tuyến tính
26.	Random Forest + AdaBoost + Random Forest	90.07	98.65	97.78	98.22	98.65
27.	Random Forest + AdaBoost + AdaBoost	90.07	98.65	97.2	92.33	93.35
28.	Random Forest + AdaBoost + Gradient Tree Boosting	90.07	98.65	97.69	98.26	97.93
29.	Random Forest + Gradient Tree Boosting + Bagging	90.07	98.65	97.2	98.94	98.89
30.	Random Forest + Gradient Tree Boosting + Random Forest	90.07	98.65	97.78	98.75	98.99
31.	Random Forest + Gradient Tree Boosting + AdaBoost	90.07	98.65	97.2	98.84	98.7
32.	Random Forest + Gradient Tree Boosting + Gradient Tree Boosting	90.07	98.65	97.69	98.7	98.89
33.	AdaBoost + Bagging + Bagging	89.92	98.84	97.2	98.94	98.99
34.	AdaBoost + Bagging + Random Forest	89.92	98.84	97.78	99.08	99.13
35.	AdaBoost + Bagging + AdaBoost	89.92	98.84	97.2	98.84	98.75
36.	AdaBoost + Bagging + Gradient Tree Boosting	89.92	98.84	97.69	98.94	98.75
37.	AdaBoost + Random Forest + Bagging	89.92	99.08	97.2	98.94	99.08
38.	AdaBoost + Random Forest + Random Forest	89.92	99.08	97.78	98.99	99.04
39.	AdaBoost + Random Forest + AdaBoost	89.92	99.08	97.2	99.08	99.04
40.	AdaBoost + Random Forest + Gradient Tree Boosting	89.92	99.08	97.69	98.75	98.55
41.	AdaBoost + AdaBoost + Bagging	89.92	98.65	97.2	97.49	97.11
42.	AdaBoost + AdaBoost + Random Forest	89.92	98.65	97.78	98.31	98.02
43.	AdaBoost + AdaBoost + AdaBoost	89.92	98.65	97.2	98.84	98.26

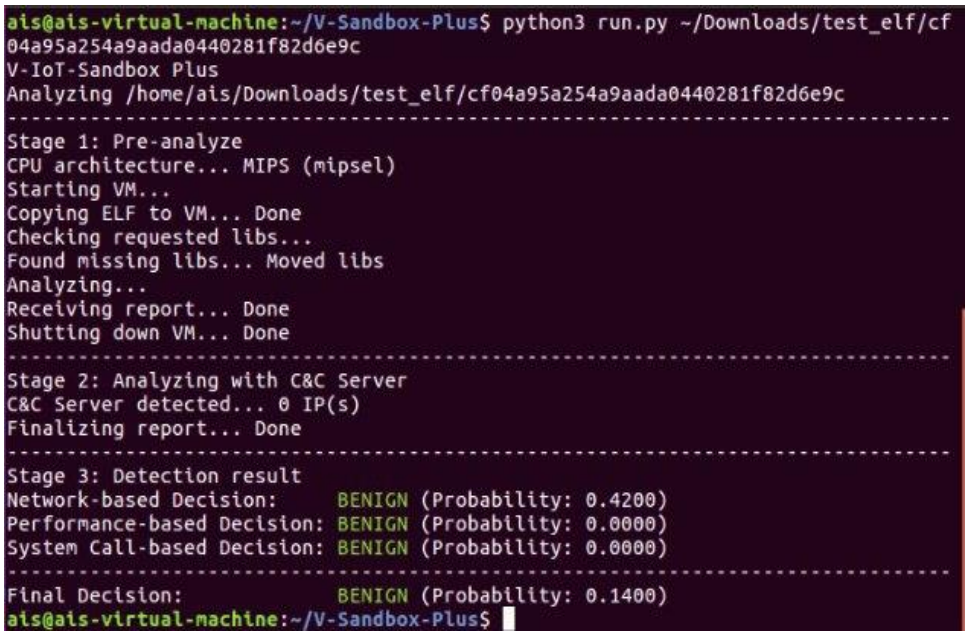
STT	Thuật toán	Độ chính xác phân loại dữ liệu của mô hình (ACC - %)				
		Dữ liệu luồng mạng	Dữ liệu sử dụng tài nguyên thiết bị	Dữ liệu lời gọi hệ thống	Bầu chọn	Hồi quy tuyến tính
44.	AdaBoost + AdaBoost + Gradient Tree Boosting	89.92	98.65	97.69	97.83	97.73
45.	AdaBoost + Gradient Tree Boosting + Bagging	89.92	98.65	97.2	98.94	98.89
46.	AdaBoost + Gradient Tree Boosting + Random Forest	89.92	98.65	97.78	98.94	98.99
47.	AdaBoost + Gradient Tree Boosting + AdaBoost	89.92	98.65	97.2	98.65	98.6
48.	AdaBoost + Gradient Tree Boosting + Gradient Tree Boosting	89.92	98.65	97.69	99.04	98.94
49.	Gradient Tree Boosting + Bagging + Bagging	89.97	98.84	97.2	99.04	99.04
50.	Gradient Tree Boosting + Bagging + Random Forest	89.97	98.84	97.78	99.04	99.18
51.	Gradient Tree Boosting + Bagging + AdaBoost	89.97	98.84	97.2	98.46	98.79
52.	Gradient Tree Boosting + Bagging + Gradient Tree Boosting	89.97	98.84	97.69	98.84	98.79
53.	Gradient Tree Boosting + Random Forest + Bagging	89.97	99.08	97.2	98.99	99.08
54.	Gradient Tree Boosting + Random Forest + Random Forest	89.97	99.08	97.78	99.18	99.18
55.	Gradient Tree Boosting + Random Forest + AdaBoost	89.97	99.08	97.2	98.36	98.99
56.	Gradient Tree Boosting + Random Forest + Gradient Tree Boosting	89.97	99.08	97.69	98.84	98.7
57.	Gradient Tree Boosting + AdaBoost +	89.97	98.65	97.2	98.26	97.3

STT	Thuật toán	Độ chính xác phân loại dữ liệu của mô hình (ACC - %)				
		Dữ liệu luồng mạng	Dữ liệu sử dụng tài nguyên thiết bị	Dữ liệu lời gọi hệ thống	Bầu chọn	Hồi quy tuyến tính
	Bagging					
58.	Gradient Tree Boosting + AdaBoost + Random Forest	89.97	98.65	97.78	98.02	98.65
59.	Gradient Tree Boosting + AdaBoost + AdaBoost	89.97	98.65	97.2	92.48	93.15
60.	Gradient Tree Boosting + AdaBoost + Gradient Tree Boosting	89.97	98.65	97.69	98.22	97.93
61.	Gradient Tree Boosting + Gradient Tree Boosting + Bagging	89.97	98.65	97.2	98.94	98.89
62.	Gradient Tree Boosting + Gradient Tree Boosting + Random Forest	89.97	98.65	97.78	98.7	98.99
63.	Gradient Tree Boosting + Gradient Tree Boosting + AdaBoost	89.97	98.65	97.2	98.84	98.7
64.	Gradient Tree Boosting + Gradient Tree Boosting + Gradient Tree Boosting	89.97	98.65	97.69	98.65	98.84

Mô hình sau khi được huấn luyện xong sẽ được đưa vào vận hành thử trong hệ thống; luận văn lựa chọn thời gian thu thập dữ liệu theo thời gian thực là 03 giây (đảm bảo để có thể thu được 300 lời gọi hệ thống, 20 trạng thái tài nguyên thiết bị và 50 gói tin luồng mạng như đã phân tích) cho quá trình thực thi tệp đầu vào và đưa ra kết quả dự đoán phân loại tệp tin. Kết quả chạy thực tế được minh họa như trong các Hình 3.14, 3.15.



Hình 3.14. Quá trình phân tích tệp chứa mã độc



Hình 3.15. Quá trình phân tích tệp lành tính

3.4. Đánh giá kết quả thực nghiệm

Với kết quả được nêu ra ở phần trên có thể đánh giá mô hình ứng dụng có khả năng phát hiện chính xác với *ACC = 99.23%*. Kết quả thực nghiệm này cho thấy hiệu quả của việc sử dụng mô hình học máy cộng tác cho 3 loại dữ liệu hành vi phổ biến trong phát hiện IoT Botnet. Sử dụng Information Gain để trích chọn đặc trưng, cách kết hợp các bộ phân lớp sử dụng thuật toán học máy với nhau trong một mô hình cộng tác đã góp phần làm tăng hiệu quả phát hiện của mô hình. Mô hình kết hợp đã cho kết quả phát hiện vượt trội hơn các mô hình học máy đơn lẻ. Khả năng phát hiện sớm của mô hình cũng được thể hiện ở đặc điểm chỉ lấy một phần nhỏ lượng dữ liệu đặc trưng cho các hành vi đầu tiên của tệp đầu vào đang được xử lý để thực hiện phân tích và phát hiện mã độc thay vì phải đợi mã độc thực hiện đầy đủ hành vi để thu thập và xử lý với toàn bộ dữ liệu.

Kết luận chương

Trong chương này, luận văn đã trình bày kết quả thực nghiệm triển khai mô hình ứng dụng được mô tả trong chương 2 với bộ dữ liệu và nền tảng phần cứng, phần mềm đi kèm. Kết quả thử nghiệm đã cho thấy hiệu quả vượt trội của mô hình ứng dụng và khả năng ứng dụng vào giải quyết bài toán phát hiện mã độc IoT Botnet trong thực tế.

KẾT LUẬN VÀ KIẾN NGHỊ

Cuộc cách mạng 4.0 và sự bùng nổ phát triển về công nghệ, khoa học kỹ thuật những năm vừa qua đã mang lại rất nhiều thay đổi lớn với cuộc sống nhân loại. Trong cuộc phát triển đó, xu hướng Vạn vật kết nối IoT nổi một cách mạnh mẽ và trở thành một phần không thể thiếu trong không gian số của mỗi cá nhân, tổ chức. Các thiết bị IoT đã, đang và sẽ tiếp tục được sử dụng phổ biến tại các tổ chức, doanh nghiệp ở nhiều quốc gia trên thế giới, trong đó có cả Việt Nam. Số lượng thiết bị IoT ngày càng tăng, tỉ lệ thuận với đó là số lượng mã độc, các cuộc tấn công khai thác đem lại thách thức rất lớn đối với việc bảo đảm an ninh, an toàn thông tin. Song song với việc phát triển và mở rộng rất nhanh về số lượng thì các nhà phát triển thiết bị IoT lại không có sự quan tâm đến vấn đề bảo mật khiến những thiết bị này trở thành mục tiêu dễ dàng cho các hành vi tấn công, khai thác. Do đó việc nghiên cứu, phát triển các hình thức bảo vệ các thiết bị IoT là hoàn toàn cần thiết, góp phần đảm bảo an ninh, an toàn thông tin trên môi trường mạng.

Trong phạm vi nghiên cứu về phân tích, phát hiện mã độc Botnet trên các thiết bị IoT, luận văn đã tiến hành tìm hiểu các phương pháp phân tích, phát hiện mã độc Botnet trên thiết bị IoT phổ biến hiện nay và sau đó xây dựng, ứng dụng thực nghiệm mô hình học máy cộng tác trong phân tích, phát hiện mã độc IoT Botnet. Cụ thể, luận văn đã đạt được một số kết quả như sau:

- Nghiên cứu, lựa chọn, ứng dụng và xây dựng thử nghiệm thành công mô hình học máy cộng tác trong phân tích và phát hiện mã độc IoT Botnet.

- Thử nghiệm phát hiện mã độc với các mô hình học máy đơn lẻ và so sánh đánh giá với mô hình học máy đã xây dựng. Kết quả cho thấy hiệu suất phát hiện được cải thiện vượt trội so với việc sử dụng các bộ học máy đơn lẻ.

- Đem lại khả năng ứng dụng trong thực tế khi mô hình cho ra kết quả phát hiện trong thời gian ngắn và chỉ yêu cầu lượng dữ liệu đầu vào nhỏ ngay khi mã độc bắt đầu thực hiện hành vi. Do đó giảm thiểu được hậu quả của mã độc gây ra với thiết bị và hệ thống thông tin.

Kết quả của luận văn góp phần bổ sung vào các nghiên cứu phát hiện mã độc IoT Botnet dựa trên phương pháp phân tích động và tiềm năng ứng dụng cao. Một số nội dung nghiên cứu trong luận văn cũng được chấp nhận công bố trên Kỷ yếu hội nghị quốc tế lần thứ 3 về Điện tử, truyền thông và khoa học máy tính (ICECCE 2021) với bài báo *"Adversarial Attack and Defense on Graph-based IoT Botnet Detection Approach"*.

Tuy nhiên, luận văn vẫn còn một số hạn chế, vướng mắc ở phần xử lý sandbox vì không gian tài nguyên yêu cầu lớn, thời gian khởi động chậm. Ngoài ra khi chạy V-Sandbox để thu thập hành vi dữ liệu của một số mẫu thì có thể xảy ra việc thực hiện vòng lặp để thu thập thêm dữ liệu cho mỗi lần chạy khiến cho thời gian xử lý 1 mẫu lên đến 3 phút.

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 nghiên cứu, thử nghiệm và cải thiện phương pháp kết hợp các đặc trưng và bộ phân loại nhằm cải thiện độ chính xác và thời gian xử lý của phương pháp phát hiện sớm mã độc IoT Botnet.

- Nghiên cứu, cải tiến để rút ngắn thời gian hoạt động của V-Sandbox và lượng tài nguyên yêu cầu để làm cho mô hình phát hiện sớm IoT Botnet hoạt động hiệu quả hơn.

DANH MỤC TÀI LIỆU THAM KHẢO

[1] K. Ashton, "That 'internet of things' thing," *RFID J.*, vol. 22, no. 7, pp. 97–114, 2009.

[2] International Telecommunication Union, "Overview of the Internet of things, Recommendation ITU-T Y.20602013." [Online]. Available: [Online]. Available: https://www.itu.int/rec/T-REC-Y.2060-201206-I

[3] "Overview of Internet of Things." ITU-T Y.2060, Jun. 2012.

[4] Knud Lasse Lueth, "IoT Market – Forecasts at a glance," 2014. [Online]. Available: https://iot-analytics.com/iot-market-forecasts-overview/. Visited on: 15/4/2018

[5] C. Lévy-Bencheton, E. Darra, G. Tétu, G. Dufay, and M. Alattar, "Security and resilience of smart home environments good practices and recommendations," *Eur. Union Agency Netw. Inf. Secur. ENISA Heraklion Greece*, 2015.

[6] A L Johnson, Symantec Security Response, "IoT devices being increasingly used for DDoS attacks," 2016. [Online]. Available: https://www.symantec.com/connect/blogs/iot-devices-being-increasingly-used-ddos-attacks. Visited on: 17/9/2017

[7] W. Zhou, Y. Jia, A. Peng, Y. Zhang, and P. Liu, "The effect of iot new features on security and privacy: New threats, existing solutions, and challenges yet to be solved," *IEEE Internet Things J.*, vol. 6, no. 2, pp. 1606–1616, 2018, doi: https://doi.org/10.1109/JIOT.2018.2847733.

[8] Andrei Costin, "Large Scale Security Analysis of Embedded Devices' Firmware," Thesis of Doctor, Paris Institute of Technology, France, 2016.

[9] Costin, Andrei, and Jonas Zaddach, "IoT malware: Comprehensive survey, analysis framework and case studies," presented at the BlackHat USA, 2018.

[10] Mikhail Kuzin, Yaroslav Shmelev, Vladimir Kuskov, "New trends in the world of IoT threats," 2018. [Online]. Available: https://securelist.com/new-trends-in-the-world-of-iot-threats/87991/. Visited on: 15/2/2019

[11] Helenius Marko, "A system to support the analysis of antivirus products' virus detection capabilities." Tampere University Press, 2002.

[12] Ed Skoudis, Lenny Zeltser, "Malware: fighting malicious code." Prentice Hall, 2004.

[13] Kaspersky Lab report, "IoT: a malware story," *Securelist - Kaspersky Lab's cyberthreat research and reports*. https://securelist.com/iot-a-malware-story/94451/ (accessed Dec. 19, 2019).

[14] P. Beltrán-García, E. Aguirre-Anaya, P. J. Escamilla-Ambrosio, and R. Acosta-Bermejo, "IoT Botnets," in *Telematics and Computing*, Cham, 2019, pp. 247–257.

[15] Angrishi Kishore, "Turning internet of things (iot) into internet of vulnerabilities (iov): Iot botnets." arXiv preprint arXiv:1702.03681, 2017.

[16] Allix Kevin, et al., "A Forensic Analysis of Android Malware--How is Malware Written and How it Could Be Detected?," 2014, pp. 384–393. doi: https://doi.org/10.1109/COMPSAC.2014.61.

[17] Muhammad Junaid Bohio, "Analysis of a MIPS Malware." SANS Institute, 2015. [Online]. Available: [Online]. Available: https://www.sans.org/reading-room/whitepapers/malicious/analyzing-backdoor-bot-mips-platform-35902. Visited on: 16/7/2017

[18] De Donno Michele, et al, “DDoS-capable IoT malwares: Comparative analysis and Mirai investigation,” *Secur. Commun. Netw. Hindawi*, pp. 1–30, 2018, doi: https://doi.org/10.1155/2018/7178164.
[19] Celeda, P., Krejci, R., & Krmicek, V., “Revealing and analysing modem malware,” 2012, pp. 971–975. doi: https://doi.org/10.1109/ICC.2012.6364598.
[20] Celeda, P., Krejcí, R., Vykopal, J., & Drasar, M., “Embedded malware-an analysis of the Chuck Norris botnet,” 2010, pp. 3–10. doi: https://doi.org/10.1109/EC2ND.2010.15.
[21] Домът на Виерко, “lightaidra 0x2012 (aidra),” 2013. [Online]. Available: https://vierko.org/tech/lightaidra-0x2012/. Visited on: 19/9/2017
[22] Symantec Official Blog, “Linux.Wifatch,” 2015. [Online]. Available: https://www.symantec.com/security_response/writeup.jsp?docid=2015-011216-2314-99&tabid=2. Visited on: 19/9/2017
[23] Kolias Constantinos, et al., “DDoS in the IoT: Mirai and other botnets,” *IEEE Comput.*, vol. 50, no. 7, pp. 80–84, 2017, doi: https://doi.org/10.1109/MC.2017.201.
[24] “Radware. Brickerbot results in PDOS attack.” [Online]. Available: https://security.radware.com/ddos-threatsattacks/brickerbot-pdos-permanent-denial-of-service/. Visited on: 18/5/2018
[25] Symantec Security Response Security Response Team, “Vpnfilter: New router malware with destructive capabilities.” [Online]. Available: https://www.symantec.com/blogs/threat-intelligence/vpnfilteriot-malware, 2018. Visited on: 18/8/2018
[26] T. Ronghua, “An Integrated Malware Detection and Classification System,” *MEng Chongqing Univ BEngChangchun Univ Sci Technol*, 2011.
[27] Costin, A., Zaddach, J., Francillon, A. and Balzarotti, D., “A large-scale analysis of the security of embedded firmwares,” 2014, pp. 95–110.
[28] McDermott, Christopher D., Farzan Majdani, and Andrei V. Petrovski, “Botnet detection in the internet of things using deep learning approaches,” 2018, pp. 1–8. doi: https://doi.org/10.1109/IJCNN.2018.8489489.
[29] Shoshitaishvili Yan, et al., “Firmalice - Automatic Detection of Authentication Bypass Vulnerabilities in Binary Firmware,” 2015, pp. 1–15. [Online]. Available: http://dx.doi.org/10.14722/ndss.2015.23294
[30] A. Azmoodeh, A. Dehghantanha, and K.-K. R. Choo, “Robust Malware Detection for Internet of (Battlefield) Things Devices Using Deep Eigenspace Learning,” *IEEE Trans. Sustain. Comput.*, vol. 4, no. 1, pp. 88–95, Jan. 2019, doi: 10.1109/TSUSC.2018.2809665.
[31] H. HaddadPajouh, A. Dehghantanha, R. Khayami, and K.-K. R. Choo, “A deep Recurrent Neural Network based approach for Internet of Things malware threat hunting,” *Future Gener. Comput. Syst.*, vol. 85, pp. 88–96, 2018.
[32] R. Doshi, N. Apthorpe, and N. Feamster, “Machine learning ddos detection for consumer internet of things devices,” 2018, pp. 29–35.
[33] I. Alrashdi, A. Alqazzaz, E. Aloufi, R. Alharthi, M. Zohdy, and H. Ming, “AD-IoT: anomaly detection of IoT cyberattacks in smart city using machine learning,” 2019, pp. 0305–0310.
[34] N. Moustafa and J. Slay, “The evaluation of Network Anomaly Detection Systems: Statistical analysis of the UNSW-NB15 data set and the comparison with the KDD99 data set,” *Inf. Secur. J. Glob. Perspect.*, vol. 25, no. 1–3, pp. 18–31, 2016.

[35] A. M. da Silva Cardoso, R. F. Lopes, A. S. Teles, and F. B. V. Magalhães, “Real-time DDoS detection based on complex event processing for IoT,” 2018, pp. 273–274.
[36] M. Ficco, “Detecting IoT Malware by Markov Chain Behavioral Models,” 2019, pp. 229–234.
[37] S. Sachdeva, R. Jolivot, and W. Choensawat, “Android Malware Classification based on Mobile Security Framework.,” *IAENG Int. J. Comput. Sci.*, vol. 45, no. 4, 2018.
[38] D. Breitenbacher, I. Homoliak, Y. L. Aung, N. O. Tippenhauer, and Y. Elovici, “HADES-IoT: A Practical Host-Based Anomaly Detection System for IoT Devices,” 2019, pp. 479–484.
[39] T. G. Dietterich, “Ensemble learning,” *Handb. Brain Theory Neural Netw.*, vol. 2, pp. 110–125, 2002.
[40] H.-V. Le and Q.-D. Ngo, “V-Sandbox for Dynamic Analysis IoT Botnet,” *IEEE Access*, vol. 8, pp. 145768–145786, 2020.
[41] Y. M. P. Pa, S. Suzuki, K. Yoshioka, T. Matsumoto, T. Kasama, and C. Rossow, “Iotpot: A novel honeypot for revealing current iot threats,” *J. Inf. Process.*, vol. 24, no. 3, pp. 522–533, 2016.
[42] C. Guarnieri, A. Tanasi, J. Bremer, and M. Schloesser, “The cuckoo sandbox,” 2012.
[43] “REMnux: A free Linux Toolkit for Reverse-Engineering and Analyzing Malware.” https://remnux.org/ (accessed Mar. 10, 2020).
[44] K. Monnappa, “Automating linux malware analysis using limon sandbox,” *Black Hat Eur. 2015*, 2015.
[45] F. Bellard, “QEMU, a fast and portable dynamic translator,” *ATEC ’05 Proc. Annu. Conf. USENIX Annu. Tech- Nical Conf.*, pp. 41–44, 2005.
[46] H. V. Le, Q. D. Ngo, and V. H. Le, “Iot Botnet Detection Using System Call Graphs and One-Class CNN Classification,” *Int. J. Innov. Technol. Explor. Eng.*, vol. 8, no. 10, pp. 937–942, Aug. 2019, doi: 10.35940/ijitee.J9091.0881019.
[47] A. Narayanan, M. Chandramohan, R. Venkatesan, L. Chen, Y. Liu, and S. Jaiswal, “graph2vec: Learning distributed representations of graphs,” *ArXiv Prepr. ArXiv170705005*, 2017.
[48] J. H. Lau and T. Baldwin, “An empirical evaluation of doc2vec with practical insights into document embedding generation,” *ArXiv Prepr. ArXiv160705368*, 2016.
[49] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, “Toward generating a new intrusion detection dataset and intrusion traffic characterization.,” in *ICISSP*, 2018, pp. 108–116.
[50] “scikit-learn: machine learning in Python — scikit-learn 0.20.2 documentation.” https://scikit-learn.org/stable/ (accessed Jan. 16, 2019).