**Phát hiện mã độc trên các thiết bị IoT dựa trên lời gọi Syscall và phân loại một lớp SVM**

|  |  |  |
| --- | --- | --- |
| Trần Nghi Phú\*, Ngô Quốc Dũng  Học viện An ninh nhân dân Hà Nội, Việt Nam Email: tnphvan@gmail.com  \* KX.T31.01 |  | Hoàng Đăng Kiên, Nguyễn Ngọc Bình, Nguyễn Đại Thọ  *Đại học Công nghệ, Đại học quốc gia Hà Nội Email: binhnn, nguyendaitho@vnu.edu.vn* |

*Abstract* - **Mã độc trên các thiết bị Vạn vật kết nối Internet - Internet of Things (IoT), thành phần cốt lõi trong cuộc cách mạng 4.0, đang ngày càng tăng nhanh. Các thiết bị IoT có kiến trúc MIPS chiếm tỉ lệ lớn, song các nghiên cứu phát hiện mã độc dựa trên hành vi của các thiết bị này chưa được đề cập. Chúng tôi đề xuất quy trình phân tích phát hiện mã độc trong các thiết bị IoT sử dụng kiến trúc MIPS dựa trên lời gọi hệ thống syscall (hay system call) bằng kỹ thuật phân loại một lớp SVM. Các chương trình được chạy trong C500-Sandbox để thu thập các syscall được gọi, các syscall được biểu diễn dưới dạng đặc trưng n-gram, sau đó sử dụng phương pháp phân tích thành phần chính (Principal Component Analysis - PCA) để giảm số chiều trước khi được huấn luyện/nhận dạng bằng mô hình phân loại một lớp SVM (One class SVM). Quy trình cho kết quả phát hiện tốt với chỉ số F-Score (F1) = 0,976, Average precision (AP) = 0,992, Accuracy (AC) = 0,956.**

Keywords: **IoT; system call; one class svm; malware**;

# I. GIỚI THIỆU

Sự phát triển mạnh mẽ của vạn vật kết nối Internet (IoT - Internet of Things) đã mang lại các tiện ích phong phú cho người dùng như trong quản lý năng lượng thông minh, theo dõi sức khoẻ, thiết bị tự hành hay cơ sở hạ tầng thông minh... Năm 2017 đã có khoảng 7 tỉ thiết bị IoT, đến năm 2020 ước tính sẽ có 50 tỷ thiết bị IoT sẽ được kết nối vào mạng Internet [1] và có mặt ở mọi nơi mà đặc biệt là các thiết bị như CameraIP, VoIp, IpTV, thiết bị định tuyến... Những thiết bị IoT có thể sử dụng để giám sát, theo dõi người dùng mọi lúc, mọi nơi hay tranh thủ số lượng lớn các thiết bị này để tạo các cuộc tấn công mạng quy mô lớn. Điều này đã làm dấy lên những lo ngại về việc lộ lọt thông tin của các tổ chức, cá nhân khi các thiết bị này bị tin tặc tấn công và chiếm quyền quản trị. Cho đến thời điểm hiện nay, những nghiên cứu, chính sách về bảo mật, an ninh an toàn thông tin chưa được quan tâm đúng mức so với sự phát triển nhanh và rộng của IoT.

Ngày nay, số lượng mã độc tấn công các thiết bị IoT ngày càng tăng nhanh, gồm nhiều loại khác nhau như mã độc tấn công từ chối dịch vụ (DDos), cổng hậu, thư rác... Mạng IoT botnet đã gửi hơn 750.000 thư rác mỗi ngày [2], Mirai một trong những mã độc nổi tiếng trên các thiết bị IoT đã tạo ra những cuộc tấn công từ chối dịch vụ lớn nhất trong lịch sử với dung lượng vượt ngưỡng 1,5 Terabit/s (Tbps)[3]. Kaspersky Lab cũng đã theo dõi kết quả các mẫu mã độc IoT xuất hiện trong thời gian gần đây và phát hiện có sự tăng đột biến về số lượng, với 46 mẫu được phát hiện vào năm 2013, 696 mẫu vào năm 2015 và đến năm 2017 đã lên đến 7.242 mẫu. Nguy cơ lộ lọt thông tin và cài cắm các mã độc từ các thiết bị IoT đã được H.Grant [4] phân tích, chứng minh là hiện hữu. Daming D. Chen [5] đã tiến hành phân tích 23.035 firmware từ 42 nhà cung cấp và phát hiện ra 887 firmware có lỗ hổng bảo mật, trong đó có 14 lỗ hổng bảo mật chưa được biết đến trước đây. Tại Việt Nam, tháng 6/2016, tập đoàn BKAV đã công bố kết quả khảo sát 21 triệu thiết bị trên Internet, trong đó có tới 5,6 triệu thiết bị trên thế giới bị nhiễm lỗ hổng PetHole [6], dẫn đến nguy cơ mất quyền điều khiển thiết bị.

Kiến trúc chip trên máy tính với đa số là i386 còn kiến trúc chíp sử dụng phổ biến trên các thiết bị IoT là ARM (ARM7, ARM9, Cortex), Intel ATOM, MIPS, Motorola, Axis CRIS và hệ điều hành sử dụng phổ biến là Embeded Linux như [7] đề cập, trong đó phiên bản Linux sử dụng phổ biến là Linux 2.6 và 3.0 [8]. Kiến trúc chíp MIPS [10] xuất hiện nhiều trong các thiết bị mạng như bộ định tuyến, bộ phát không dây, camera... Chương trình chạy trên mỗi kiến trúc chíp và hệ điều hành khác nhau các hành vi thể hiện sẽ khác nhau.

Để thực hiện phân tích động mã độc nói chung và mã độc trên IoT nói riêng, sử dụng sandbox là giải pháp phổ biến. Nhiều giải pháp sandbox tương đối hoàn chỉnh cho phân tích mã độc trên máy tính thông thường đã được xây dựng như Cuckoo [9], Anubis [11], CwSandbox [12]. Tuy nhiên các giải pháp này không áp dụng cho việc phân tích mã độc trên các thiết bị IoT do khác về kiến trúc chíp và hệ điều hành. Các nghiên cứu gần đây như [13], [14] tập trung giải quyết cho các thiết bị IoT nhưng vẫn giới hạn chức năng. Môi trường giả lập đa nền tảng cho Linux - Detux [13] công bố mã nguồn, số lượng mẫu thu thập được và thông tin thu được từ sandbox Detux khá đầy đủ nhưng không có khả năng thu thập hành vi hệ thống, chỉ tập trung thu thập thông tin mạng, song lại chưa đề cập vấn đề giả lập mạng và các dịch vụ mạng. IoTPot [14] xây dựng sandbox nhằm phân tích các hành vi của mã độc tấn công dựa trên giao thức Telnet dựa trên một số hành vi mạng và hệ thống, không sử dụng syscall để phân tích mã độc này. Nếu không giả lập mạng đầy đủ thì khi thực thi mã độc sẽ ảnh hưởng đến môi trường xung quanh và nhiều hành vi của mã độc sẽ không được thực hiện do không có đủ môi trường cần thiết. Như vậy, chưa có sandbox nào hỗ trợ các thiết bị IoT sử dụng chíp MIPS có chức năng thu thập syscall và giả lập Internet phục vụ phân tích mã độc.

Nhiều nghiên cứu thực nghiệm và công bố các dữ liệu về tập mã độc là các tệp tin thực thi trên hệ điều hành Embedded Linux của thiết bị sử dụng chíp có kiến trúc MIPS (gọi tắt là MIPS ELF) còn ít về số lượng và chưa được chuẩn hoá, gán nhãn. YMP. Pa và cộng sự [14] thu thập 4.000 mẫu mã độc trên IoT, trong đó có 938 là MIPS ELF, tiến hành đánh giá bằng VirusTotal [27], kết quả cho thấy nhiều mẫu có tỉ lệ phát hiện là mã độc dưới 3/67 phần mềm, trong đó không có các phần mềm uy tín. K.A. Asmitha và P. Vinod [15] đề xuất phương pháp phát hiện mã độc trên Linux, bộ dữ liệu thu thập được chỉ 668 mẫu. [13] thu thập được hơn 9.000 mẫu, trong đó có hơn 3.200 mẫu MIPS ELF, trong đó có một số mẫu không được phát hiện là mã độc.

Nhiều nghiên cứu đã tập trung phát hiện mã độc qua hành vi syscall của hệ thống cho kết quả phát hiện tốt, song chưa có nghiên cứu nào phát hiện mã độc qua syscall với các mã độc trong MIPS ELF. Các nghiên cứu tập trung vào phát hiện mã độc trên hệ điều hành Windows với kiến trúc chip i386, như R.J. Canzanese [16] đánh giá bằng nhiều phương pháp học máy hay [17] áp dụng học sâu cho kết quả khả quan. Mã độc trên hệ điều hành android cũng có thể phát hiện qua syscall như [18] đề cập. Asmitha và cộng sự [15] thử nghiệm phương pháp phát hiện mã độc trên Linux thông qua syscall với nhiều phương pháp học máy khác nhau, nhưng bộ dữ liệu ít chỉ với 668 mẫu và chỉ nghiên cứu trên kiến trúc i386.

Phân loại một lớp được đề xuất bởi J.Tax [19], là phương pháp cho phép huấn luyện mô hình phân lớp với dữ liệu huấn luyện chỉ có một nhãn. Phân loại một lớp dựa trên máy hỗ trợ véc tơ (Support Vector Machine - SVM) [20] ứng dụng dụng trong phát hiện thâm nhập cho khả năng phát hiện nhanh chóng các gói tin tấn công với tỷ lệ dương tính thật cao trong khi duy trì tỷ lệ dương tính giả ở mức thấp. Tập dữ liệu chúng tôi thu thập được có đặc thù là rất ít mẫu sạch, đa số là mẫu mã độc, khác với cách tiếp cận thông thường là học dựa trên tập các hành vi bình thường để phát hiện hành vi bất thường. Như vậy, phương pháp phân loại một lớp SVM đã chứng minh khả năng phân loại tốt trên tập dữ liệu chỉ có một nhãn, song vẫn chưa có nghiên cứu nào ứng dụng phân loại một lớp SVM để phát hiện mã độc và đặc biệt với tập dữ liệu huấn luyện rất đặc biệt là mẫu mã độc nhiều hơn mẫu sạch, hay học các hành vi mã độc để phát hiện các hành vi bình thường.

Trong bài báo này, chúng tôi đề xuất quy trình phát hiện mã độc trong MIPS ELF trong các thiết bị IoT dựa sự kết hợp của C500-Sandbox là cải tiến từ Detux sandbox và phân loại một lớp SVM. Quy trình của chúng tôi đề xuất gồm 6 bước, trong đó tập trung xây dựng C500-Sandbox, thu thập tập dữ liệu C500-IoT và áp dụng phương pháp trích chọn đặc trưng và học máy phù hợp để có mô hình phát hiện cho các độ đo tốt. C500-Sandbox được phát triển trên cơ sở kế thừa Detux-Sandbox, bổ sung khả năng thu thập hành vi syscall của hệ thống và mô phỏng mạng Internet để các chương trình bộc lộ tối đa các hành vi. Tập dữ liệu C500-IoT là tập dữ liệu chuyên về mã độc IoT trên chip MIPS tập hợp từ 2 nguồn [13,14], có số lượng lớn nhất công bố thời điểm hiện tại với 3.773 mẫu MIPS ELF được đánh giá và gán nhãn phục vụ cho nhiều nghiên cứu sau này. Phân loại một lớp SVM được huấn luyện để phát hiện các chương trình sạch, mã độc dựa trên cơ sở huấn luyện bởi các hành vi của mã độc. Quy trình cũng lựa chọn các phương pháp trích chọn đặc trưng và học máy phù hợp, thực nghiệm lựa chọn tìm các ngưỡng về độ dài syscall cần theo dõi của chương trình để có khả năng phát hiện mã độc tốt nhất. Đây là quy trình đầu tiên thử nghiệm phát hiện mã độc trong MIPS ELF trên các thiết bị IoT dựa trên đặc trưng hành vi và phân loại một lớp SVM, và là nghiên cứu đầu tiên thử nghiệm phân loại một lớp SVM cho nhận diện chương trình sạch, mã độc trên cơ sở học hành vi mã độc. Kết quả thử nghiệm cho thấy quy trình có khả năng phát hiện tốt chương trình sạch, mã độc trong MIPS ELF trên các thiết bị IoT với kiến trúc chip MIPS với chỉ số F1 = 0,976, AP = 0,992.

Phần tiếp theo của bài báo được tổ chức như sau: Mục 2 trình bày các thành phần xây dựng sandbox phân tích mã độc trong các thiết bị IoT; Mục 3 mô tả quy trình phân tích phát hiện mã độc trong các thiết bị IoT mà chúng tôi đề xuất; Mục 4 là thử nghiệm quy trình; Cuối cùng là kết luận và phương hướng phát triển tiếp theo.

# II. CÁC THÀNH PHẦN XÂY DỰNG SANDBOX PHÂN TÍCH ĐỘNG MÃ ĐỘC TRONG CÁC THIẾT BỊ IoT

Để thực hiện phân tích mã độc nói chung và mã độc trên IoT nói riêng, sandbox có vai trò hết sức quan trọng. Sandbox có 2 loại là sanbox vật lý và sandbox ảo. Tiêu chí quan trọng nhất của sandbox phục vụ phân tích mã độc là lượng thông tin thu được sau khi thực thi mã độc trên sandbox đó, hay làm mã độc bộc lộ tối đa hành vi. Các sandbox vật lý có khả năng để mã độc bộc lộ hành vi tốt hơn do nó dựa trên thiết bị thực tế, đầy đủ các thành phần và có thể kích hoạt đối với các mã độc có khả năng phát hiện hệ thống phân tích, theo dõi nó. Tất nhiên sandbox vật lý có hạn chế lớn là khó tuỳ biến, khó khôi phục lại hiện trạng ban đầu và chi phí cao. Sandbox ảo dựa trên công nghệ mô phỏng, ảo hoá thường là sử dụng các máy ảo. Điều quan trọng nhất, để mã độc thực hiện được hành vi thì phải mô phỏng đầy đủ các môi trường như thiết bị ngoại vi, mạng, môi trường kết nối... Các sandbox cho mã độc trên máy tính đa năng truyền thông thường hướng đề sử dụng Vmware, Virtual Box nhưng với các thiết bị IoT thì các Qemu là giải pháp được sử dụng phổ biến, do Vmware và Virtual Box không hỗ trợ các kiến trúc chíp nhúng phổ biến như MIPS, ARM...

## A. Hệ thống mô phỏng và ảo hoá

Qemu [32] là phần mềm mô phỏng và ảo hoá máy tính nổi tiếng, nguồn mở, có khả năng hỗ trợ mô phỏng 26 kiến trúc CPU khác nhau, đặc biệt các kiến trúc chíp IoT như MIPS, ARM... và hỗ trợ các hệ điều hành Windows, Linux. Detux [13] được xây dựng dựa trên Qemu, là sandbox hỗ trợ thực thi các kiểu tệp tin ELF, bash... trên đa kiến kiến trúc trong đó có MIPS, ARM, MIPSel, Intel... Các thông tin mà Detux sandbox thu thập được bao gồm các dữ liệu mạng (tệp tin pcap), thông tin tĩnh tuy nhiên chưa có thông tin về hành vi hệ thống, cụ thể là các syscall được gọi khi thực thi tệp tin.



Hình 1: Quy trình phát hiện mã độc trên thiết bị IoT dựa trên phân tích động

*B. Công cụ thu thập lời gọi hệ thống Syscall*

Strace [21] là công cụ trên hệ điều hành Linux cho phép theo dõi các chương trình đang chạy, thu thập các syscall của chương trình đó, gồm tên, tham số truyền vào và kết quả gọi syscall đó. Strace có sẵn nhiều chức năng, cho phép theo dõi các tiến trình được tạo ra từ tiến trình cho ban đầu đang được strace giám sát hoặc theo dõi các tiến trình đang chạy. Strace đã được sử dụng trong nhiều nghiên cứu để thu thập các hành vi của mã độc trong môi trường Android [18], Linux [15].

*C. Giả lập mạng*

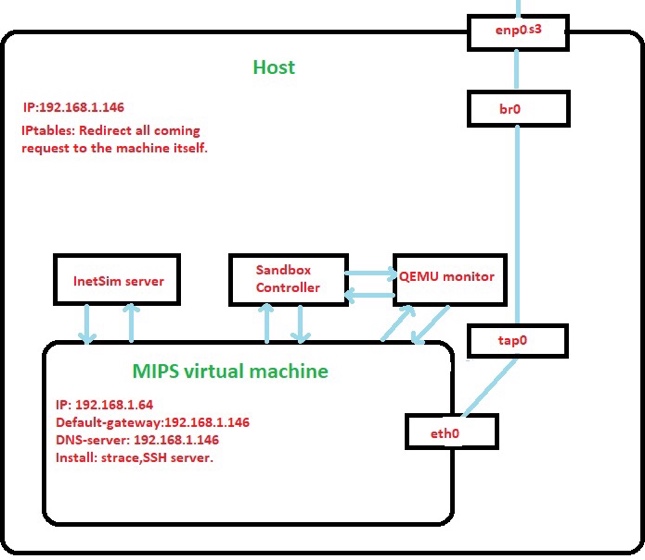
InetSim [22] là một gói phần mềm chạy trên nền Linux, chứa các kịch bản Perl được sử dụng để mô phỏng nhiều dịch vụ mạng như DNS, HTTP, FTP... Gần đây, PynetSim [26] ra đời, được đánh giá là bản nâng cấp phát triển hoàn thiện InetSim cho các thiết bị IoT. PynetSim phát triển trên Python3, cho phép phát hiện động giao thức của mã độc, hỗ trợ sẵn các kịch bản tương tác với các mã độc IoT như DDos bot, Mirai, LizardStresser.

# III. QUY TRÌNH PHÂN TÍCH PHÁT HIỆN MÃ ĐỘC TRÊN THIẾT BỊ IoT DỰA TRÊN PHÂN TÍCH ĐỘNG

Chúng tôi đề xuất quy trình gồm 6 bước để phát hiện mã độc như trong Hình 1.

*A. Thiết lập C500-IoT Sandbox*

C500-IoT Sandbox của chúng tôi tích hợp thêm khả năng thu thập hành vi là các lời gọi hệ thống syscall dựa trên công cụ Strace và kết hợp với bộ mô phỏng INetSim/pyNetsim để cung cấp mô phỏng dịch vụ mạng giúp cho mã độc tương tác bộc lộ các hành vi. Kết quả thu thập được của C500-IoT Sandbox sẽ là các thông tin phân tích tĩnh của mẫu thử, dữ liệu mạng do mẫu sinh ra dưới dạng tệp tin .pcap, log iNetsim/pyNetsim về các thông tin tương tác với Internet và các hành vi hệ thống syscall dưới dạng tệp tin syscall log. Cấu trúc C500-Sandbox có 3 thành phần chính là Sandbox Controller, QEMU Monitor và INetSim server được thể hiện như trong Hình 2.



Hình 2: Cấu trúc C500-IoTSandbox

- Sandbox Controller tương tác với QEMU mon- itor thông qua việc gọi các lệnh hiển thị cấu hình mạng, restore snapshot. Và sanbox controller tương tác với máy ảo thông qua việc gọi các thủ tục SSH tới máy ảo, truyền tệp tin thực thi từ máy thật vào trong, cấp quyền thực thi, yêu cầu thực thi file, tải tệp tin dữ liệu thu được sau khi chạy mẫu từ máy ảo.

- Monitor thực hiện việc tương tác máy ảo thông qua việc restore snapshot.

- Máy ảo tương tác với INetSim server thông qua việc gửi các request (http,ftp,dns...) và nhận lại các fake respond từ INetSim.

- enp0s3, br0, tap0 là các Network Interface.

Sandbox controller gọi thủ tục khởi động máy ảo và QEMU monitor, khởi tạo máy ảo từ trạng thái đã có bằng cách sử dụng chức năng khôi phục bản ảnh hoặc cấu hình mạng, cài đặt các gói yêu cầu và tạo một snapshot để restore lại sau khi chạy mẫu.

*B. Chạy mẫu trong môi trường C500-Sandbox*

Bước này nhằm thu thập các syscall của chương trình được gọi trong quá trình thực thi mẫu, kết quả nhận được là syscall log. Sau khi C500-Sandbox được khởi tạo, Sandbox controller kết nối SSH tới máy ảo, truyền mẫu cần chạy vào máy ảo thông qua SFTP, cấp quyền thực thi cho file, khởi động các chức năng giám sát và thực thi mẫu trong khoảng thời gian định trước, trong bài này chọn ngưỡng thời gian là 30s. Strace được tích hợp vào trong sandbox sẽ thu thập các syscall được tạo ra từ chương trình được gọi lưu vào các tệp tin. Một chương trình mẫu ban đầu có thể tạo nhiều tiến trình con khác nhau, Strace có khả năng theo dõi các tiến trình con được tạo ra này để lưu vào các tệp tin khác nhau. Nếu malware gửi một request mạng ra bên ngoài, nó sẽ bị chuyển hướng tới INetSim thông qua Iptables và kỹ thuật DNS-poisoning được tích hợp sẵn trong InetSim và INetSim server sẽ trả lại fake respond cùng kiểu với đối tượng mà mẫu yêu cầu.

*C. Chuẩn hóa dữ liệu*

Mẫu cần được thu trong cùng cầu hình sandbox và cấu hình mô phỏng mạng trong một khoảng thời gian như sau và chọn ngưỡng độ đài tối thiểu của các syscall log để xây dựng các tập mẫu. Sau khi thu được syscall log của các mẫu cần loại bỏ tệp tin syscall log của các mẫu bị lỗi không thực thi thành công hoặc quá ngắn. Một mẫu khi thực thi có thể tạo nhiều tiến trình, một tiến trình sẽ tạo một syscall log do đó cần ghép các syscall log được tạo từ 1 mẫu. Phân các syscall log của các mẫu theo các ngưỡng độ dài tối thiểu thành các tập, sau đó để tiến hành lấy đặc trưng trên các tập này.

*D. Trích rút đặc trưng*

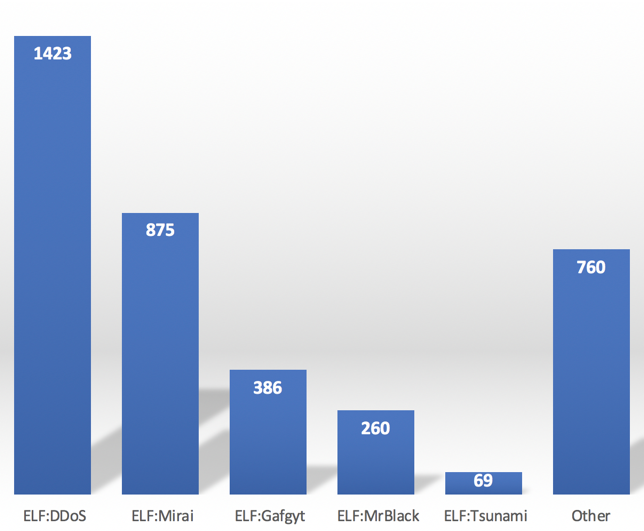
Phương pháp n-gram là phương pháp đếm số lần xuất hiện của n phần tử đứng gần nhau trong 1 chuỗi, số lần xuất hiện này được lưu trong 1 véc tơ để làm đặc trưng cho chuỗi đó. Phương pháp này đã được chứng minh hiệu quả trong phát hiện mã độc dựa trên syscall [23]. Trong bài này, chúng tôi sử dụng 1-gram và 2-gram để lấy đặc trưng dãy syscall được tạo ra khi thực thi chương trình sạch và các mã độc. Để chuẩn hoá véc tơ đặc trưng V sao cho các giá trị nằm trong đoạn [0,1] đảm bảo chuẩn đầu vào của phân loại một lớp SVM, các giá trị được chia cho n-gram có số lượng lớn nhất. Qua phân tích mã nguồn, hệ điều hành Linux trên kiến trúc MIPS có 345 syscall, nên véc tơ đặc trưng của 2-gram sẽ có 345x345 = 119.025 chiều. 1-gram là trường hợp đặc biệt, trong trường hợp này chính là số lượng xuất hiện các các syscall đơn lẻ hay tần suất xuất hiện các syscall, khi này véc tơ đặc trưng chỉ có 345 chiều. Với trường hợp 2-gram, số chiều rất lớn, yêu cầu phải áp dụng phương pháp giảm chiều, chúng tôi sử dụng phương pháp phân tích thành phần chính (Principal Component Analysis - PCA) [33] là một trong các phương pháp giảm chiều dữ liệu hiệu quả dựa trên một mô hình tuyến tính. Phương pháp này dựa trên quan sát rằng dữ liệu thường không phân bố ngẫu nhiên trong không gian mà thường phân bố gần các đường/mặt đặc biệt nào đó. Để giảm số chiều từ D về K với K < D là chỉ giữ lại K phần tử quan trọng nhất, chứa tối đa các thông tin của D chiều đó. Các phương pháp giảm chiều dữ liệu của PCA không yêu cầu phải có nhãn của dữ liệu, như vậy rất phù hợp với giảm chiều cho tập dữ liệu C500-IoT nói riêng và các tập dữ liệu lệch phải dùng đến phân lớp một lớp. Thêm vào đó, PCA có khả năng loại bỏ các thành phần ngoại lai, nên góp phần giảm nhiễu dữ liệu [25, 34].

*E. Huấn luyện và đánh giá*

Phát hiện mã độc là một bài toán phân lớp với hai nhãn mục tiêu là chương trình sạch và mã độc. Khi số lượng dữ liệu không tương đương hay lệch nhau rất lớn, xem như chỉ có 1 nhãn thì các thuật toán phân lớp bình thường không hoạt động hiệu quả, phân loại một lớp được đề cập trong [20] giải quyết tốt trong trường hợp này. Phương pháp phân lớp một lớp SVM (Support Vector Machine) được lựa chọn sử dụng trong hệ phân loại văn bản [31], hệ thống phát hiện xâm nhập [24]. Do đặc thù tập dữ liệu C500-IoT có số lượng dữ liệu giữa tập sạch và tập mã độc lệch nhau nên sử dụng phân loại một lớp SVM là giải pháp khả thi về mặt lý thuyết.

# IV. THỬ NGHIỆM QUY TRÌNH

*A. Thu thập dữ liệu*



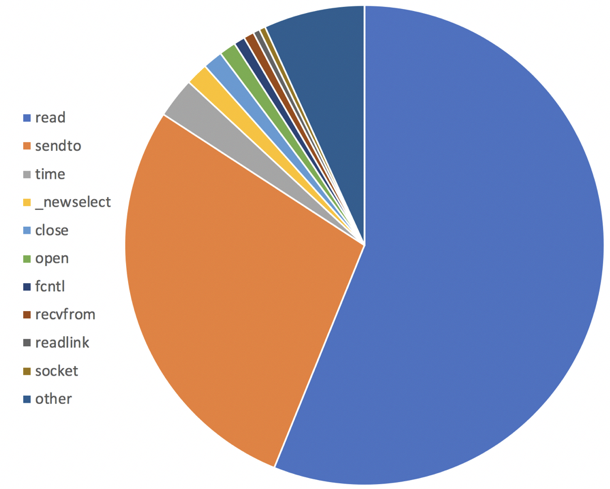
Hình 3: Các loại mã độc trong tập C500-IoT

Detux đã thu thập được hơn 9000 mẫu, trong đó có hơn 3.200 mẫu MIPS ELF, thêm vào đó [14] cũng đã tiến hành thu thập được 4.000 mẫu mã độc IoT, trong đó có 938 mẫu MIPS ELF, 38 mẫu trùng với tập 3.300 mẫu MIPS ELF của Detux. Các mẫu mã độc này được quét trên Virustotal [27], có 3.773 mẫu được phát hiện với tỉ lệ cao (tức tối thiểu 19/67 phần mềm, trong đó 4 cả phần mềm uy tín là Kaspersky, Avast, Avg, Symantec đều nhận diện đây là mã độc) sẽ được đưa vào tập dữ liệu C500-IoT để thử nghiệm. Kaspersky là engine có khả năng phát hiện tốt và cách đặt tên khá tường minh, do đó chúng tôi gán nhãn mã độc dựa trên cách phân loại của Kaspersky, kết quả như trong Hình 3. Các thiết bị sử dụng chíp MIPS đa phần là các thiết bị chuyên dụng nên các kho ứng dụng rất hạn chế, do đó số lượng các chương trình sạch thu được ít. Tập mã sạch được thu thâp từ các chương trình cơ bản có sẵn trên Embded Linux, được tích hợp sẵn trong busybox và một số ứng dụng cơ bản trên nền MIPS. Số lượng mẫu sạch thu được là 258 mẫu.

Bảng I: Đánh giá số mẫu theo độ dài syscall

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | **50** | **100** | **200** | **300** | **400** | **500** | **1500** |
| **Sạch** | 187 | 171 | 79 | 63 | 59 | 54 | 4 |
| **Mal** | 2.370 | 2.288 | 2.165 | 2.142 | 756 | 756 | 67 |

Theo nghiên cứu [23], việc lấy syscall phải trong khoảng thời gian tối thiểu 5s hoặc độ dài tối thiểu là 1.500 sẽ cho kết quả phát hiện tốt nhất với mã độc trên Windows với kiến trúc i386, song thực tế dữ liệu thu được trên MIPS ELF cho kết quả khác. Số lượng syscall thu được trong quá trình thực thi mẫu được phân thành các tập khác nhau theo độ dài tối thiểu của các syscall log để đưa vào huấn luyện nhằm xác định ngưỡng nào là phù hợp nhất cho việc phát hiện mã độc trong MIPS ELF, số liệu cụ thể được thể hiện trong Bảng I.



Hình 4: Tỉ lệ các syscall được gọi

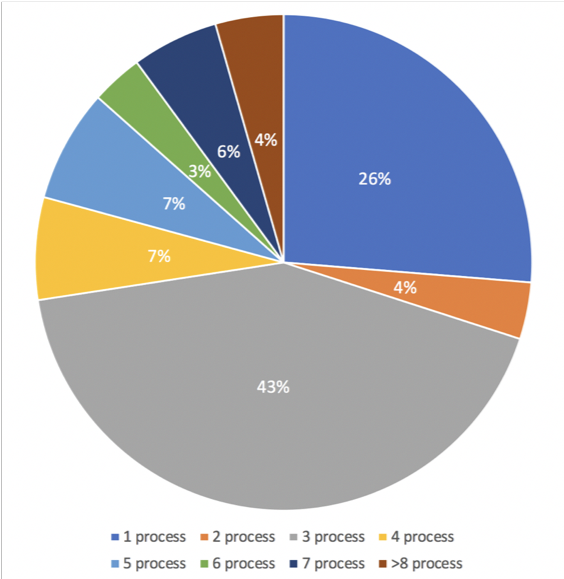
Phân các syscall log của các mẫu theo các ngưỡng độ dài tối thiểu thành các tập, sau đó để tiến hành lấy đặc trưng trên các tập này. Qua quá trình phân tích, đánh giá dữ liệu thu được chúng tôi có nhứng đánh giá sau:

- Những tệp tin thu thập strace có độ dài dưới 30 là được tạo ra bởi những chương trình bị lỗi do không đủ tham số để kích hoạt động. Một chương trình được hoạt động bình thường có độ dài syscall log trên 50, số lượng các chương trình có lượng syscall quá 1.500 như [23] không nhiều, đa số chương trình gọi số lượng syscall trong khoảng từ 50 đến 300. Miền giá trị độ dài syscall log của các chương trình MIPS ELF ngắn hơn cũng có thể lý giải do các chương trình viết trên IoT thường đơn giản hơn trên máy tính đa năng do hạ chế về tài nguyên, chức năng.

- Từ tiến trình ban đâu tạo ra khi chạy mẫu, mã độc tạo rất nhiều tiến trình con, có mã độc tạo 1000 tiến trình con. Số lượng các tiến trình con tạo ra thể hiện trong Hình 5.

- Tập mã độc chỉ sử dụng 136 syscall, tập chương trình sạch sử dụng 127 syscall, cả 2 tập này sử dụng tổng cổng 161 syscall trên tổng số 345 syscall. Phân bố syscall trong các mẫu mã độc không đều, 2 syscall read và sendto chiếm 86 % các syscall của hệ thống, chi tiết số lượng các syscall được gọi thống kê trong Hình 4.

- Qua phân tích, mã độc trên thiết bị IoT cũng có những tính năng ẩn dấu như phát hiện chương trình phân tích. Có một số mẫu mã độc có khả



Hình 5: Thống kê số tiến trình mà mã độc trong tập C500-IoT tạo ra

*B. Các độ đo hiệu năng*

Có nhiều cách để đánh giá một mô hình học máy, tuỳ thuộc vào từng mô hình và tập dữ liệu khác nhau. Với tập dữ liệu C500-IoT là tập có số lượng của 2 lớp có độ lệch lớn nên chúng tôi lựa chon phương pháp dùng Precision và Recall để đánh giá.

năng phát hiện bị strace theo dõi nên không kích hoạt. Một số nhận định trước đây cho rằng mã độc trên IoT đơn giản nhưng thực sự vẫn sử dụng các kỹ thuật ẩn dấu phổ biến của mã độc.

Trong đó:

TP: số mẫu mã độc được đoán là mã độc.

FP: số mẫu sạch được đoán là mã độc.

FN: số mẫu mã độc được đoán là sạch.

TN: số mẫu sạch được đoán là sạch.

Để đánh giá tổng thể của mô hình, đường biểu diễn quan hệ giữa Precison và Recall là Precision- Recall Curve được sử dụng. Ta đánh giá mô hình này tốt hay không bằng diện tích của Precision- Recall Curve tạo ra, người ta gọi là AUC hay Average precision (AP). Ngoài ra, để đánh giá quan hệ giữa Precision và Recall, người ta còn thường dùng đại lượng F1-score, được định nghĩa như sau:

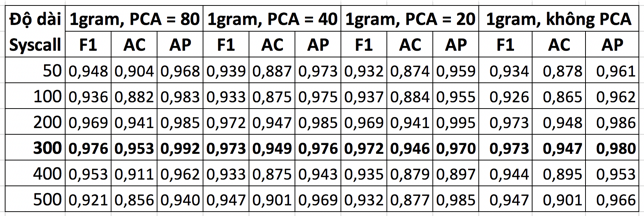
Theo cách đánh giá truyền thống, người ta dùng độ chính xác để đánh giá mô hình, trong trường hợp tập dữ liệu này độ chính xác chỉ là một kênh tham khảo.

*C. Kịch bản và quy trình thử nghiệm*

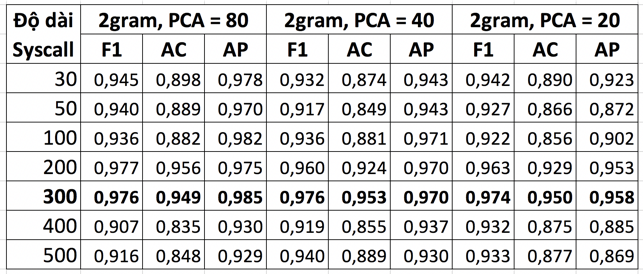
Chúng tôi cài đặt các thử nghiệm dựa trên thư viện Python Scikit-learn 0.19.2 trên máy Macbook Pro Core I5, Ram 16GB. Thử nghiệm với 5 tập dữ liệu sinh ra từ tập C500-IoT do chọn 5 ngưỡng của độ dài syscall log khác nhau là 50, 100, 200, 300, 400, 500. Với đặc trưng 1-gram, 2-gram thu được, chúng tôi thử nghiệm giảm chiều PCA với K lần lượt là 20, 40, 80. Riêng với 1-gram, do số chiều là 345 nên chúng tôi tiến hành thêm thử nghiệm không áp dụng giảm chiều bằng PCA.

*D. Kết quả thử nghiệm*

Bảng II: Kết quả thực nghiệm với 1-gram



Bảng III: Kết quả thực nghiệm với 2-gram



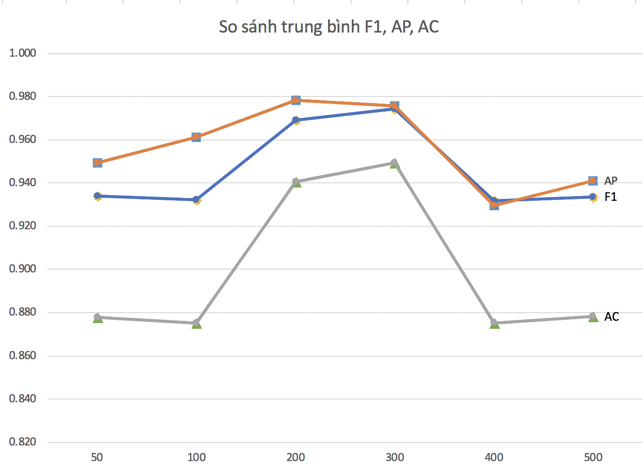
Kết quả thử nghiệm được thể hiện trong Bảng II, III và Biểu đồ 7, 6. Bảng III (II) là kết quả thử nghiệm mô hình với phương pháp trích chọn đặc trưng 1-gram (2-gram). Biểu đồ 6 biểu diễn giá trị trung bình của các độ đo F1, AP, AC với các cách lấy đặc trưng khác nhau với các tập mẫu. Giá trị F1 của các cách trích chọn đặc trưng được thể hiện trong Biểu đồ 7. Từ kết quả thực nghiệm nhận thấy:

• Kết quả cho thấy, các chỉ số mô hình tốt với giá trị F1 cao nhất đạt ngưỡng 0,977; AP cao nhất bằng 0,992; AC cao nhất bằng 0,956.

• So sánh giá trị trung bình của F1, AP, AC trong Hình 6 cho thấy, trích xuất các syscall log có độ dài 200-300 cho khả năng phát hiện tốt nhất.

• Phương pháp trích chọn đăc trưng bằng 2-gram cho giá trị F1 cao hơn phương pháp tần suất, tất nhiên độ chênh lệch không lớn. Nhưng phương pháp trích rút bằng tần suất cho giá trị AP cao hơn 2-gram, hay độ ổn định của mô hình cao hơn. Như vậy, sử dụng phương pháp 1-gram đơn giản, không gian đặc trưng ít nhưng vẫn đạt hiệu quả rất cao trong trường hợp này.

• Với phương pháp trích chọn đặc trưng bằng tần suất, áp dụng PCA cho kết quả tốt hơn về các chỉ số F1, AC và AP như trong Bảng II. Điều này có thể lý giải khi áp dụng PCA, nhiều ngoại lai được loại làm tăng độ chính xác của các đặc trưng.



Hình 6: So sánh trung bình của F1, AP, AC



Hình 7: So sánh chỉ số F1 giữa

# V. KẾT LUẬN

Trong bài báo này, chúng tôi đề xuất quy trình phát hiện mã độc trong MIPS ELF trong các thiết bị IoT dựa trên hành vi lời gọi hệ thống và phương pháp học máy phân loại một lớp SVM, trong đó tập trung đóng góp chính là xây dựng C500-IoT sandbox dựa trên cơ sở cải tiến Detux sandbox và xây dựng tập dữ liệu C500-IoT. Quá trình thực nghiệm cũng chỉ ra nhiều đặc trưng của kiểu mã độc trong MIPS ELF và ngưỡng độ dài syscall log phù hợp để phát hiện mã độc. C500-IoT sandbox không chỉ áp dụng cho kiến trúc MIPS mà có thể hỗ trợ 26 loại kiến trúc phổ biến khác. Bài báo thực nghiệm đánh giá tập dữ liệu, khả năng phát hiện mã độc bằng phương pháp trích chọn đặc trưng 1-gram và 2-gram, với việc kết hợp PCA để giảm chiều trước khi sử dụng dụng phân loại một lớp SVM để nhận dạng với kết quả tốt, chứng minh tập dữ liệu C500- IoT là đáng tin cậy và C500-IoT sandbox hoạt động hiệu quả.

Chúng tôi tiếp tục phát triển C500-IoT sandbox để mã độc bộc lộ nhiều hành vi hơn, đặc biệt không để phát hiện mã độc phát hiện đang bị theo dõi bởi strace. Trong quy trình này, chúng tôi chỉ mới khai thác các thông tin về syscall thu được từ C500-IoT Sandbox để phát hiện mã độc, các thông tin khác sẽ được tiếp tục nghiên cứu sử dụng.

CẢM ƠN

Nhóm tác giả xin chân thành cảm ơn sự hỗ trợ của đề tài K01.T31.2018 đã hỗ trợ nghiên cứu công trình. Cảm ơn em Phạm Ngọc Sơn lớp K60CD đã hỗ trợ nhóm nghiên cứu thu thập và phân tích mẫu!

# TÀI LIỆU THAM KHẢO

[1] Sebastian Muniz, Killing the myth of Cisco IOS rootkits, DIK, 2008. In EUSecWest.

[2] Hackers Use Refrigerator, Other Devices to Send 750,000 Spam Emails, <http://www.dailytech.com/>

[3] Roger Hallman, Josiah Bryan, Geancarlo Palavicini, and Joseph Divita and Jose Romero-Mariona, IoD- DoS - The Internet of Distributed Denial of Sevice Attacks, 2nd International Conference on Internet of Things, Big Data and Security. SCITEPRESS, p. 47- 58, 2017.

[4] H.Grant, O.Arias, D.Buentello, and Y.Jin, Smart nest thermostat: A smart spy in your home, Black Hat USA, 2014.

[5] Daming D.Chen\*, Manuel Egele, Maverick Woo and David Brumley, Towards Automated Dynamic Anal- ysis for Linux-based Embedded Firmware, Carnegie Mellon University, 2015.

[6] Pethole [Online]. Available http://pethole.net/

[7] A.Costin, J.Zaddach, A.Francillon and D. Balazarotti, A large-scale analysis of the security of embedded firmwares, in Proceedings of the 23rd USENIX Security Symposium, 2014, pp.95-110

[8] Shodan [Online]. http://shodan.io

[9] Shodan [Online]. https://cuckoosandbox.org/

[10] MIPS Wikipedia [Online].

https://vi.wikipedia.org/wiki/MIPS

[11] U. Bayer, A. Moser, C. Kruegel, and E. Kirda. Dynamic analysis of malicious code. Journal in Com- puter Virology, 2(1):67–77, 2006b

[12] C. Willems, T. Holz, and F. Freiling. CWSandbox: Towards automated dynamic binary analysis. IEEE Security and Privacy, 5(2), March 2007

[13] Detux Sandbox [Online].

https://github.com/detuxsandbox/detux

[14] Pa YMP, Suzuki, S Yoshioka, K Matsumoto, T Kasama, T Rossow, C 2016, ’IoTPOT: A novel hon- eypot for revealing current IoT threats’ Journal of Information Processing, vol 24, no. 3, pp. 522-533. DOI: 10.2197/ipsjjip.24.522

[15] Asmitha, K. A., and P. Vinod. A Machine Learning Approach for Linux Malware Detection. In 2014 International Conference on Issues and Challenges in Intelligent Computing Techniques (ICICT), 825–30, 2014. https://doi.org/10.1109/ICICICT.2014.6781387.

[16] 33 Canzanese, Raymond, Spiros Mancoridis, and Moshe Kam. Run-Time Classification of Malicious Processes Using System Call Analysis, 21–28. IEEE, 2015. https://doi.org/10.1109/MALWARE.2015.7413681.

[17] Bojan Kolosnjaji, Apostolis Zarras, George Webster, and Claudia Eckert. Deep Learning for Classification of Malware System Call Sequences, SpringerLink. Accessed October 21, 2018.

[18] Chaba, Sanya, Rahul Kumar, Rohan Pant, and Mayank Dave. Malware Detection Approach for An- droid Systems Using System Call Logs,” n.d., 5.

[19] J. Tax, One-Class Classification, Concept Learning in the Absence of Counter Examples, PhD thesis, Delft University of Technology, 2001.

[20] B. Scholkopf, J. Platt, J. Shawe-Taylor, A. J. Smola, and RC Williamson. Estimating the support of a high-dimensional distribution. Neural Computation, 13:1443–1471, 2001.

[21] Strace tool [Online]. http://sourceforge.net/projects/strace/

[22] InetSim [Online]. https://www.inetsim.org/

[23] Raymond J. Canzanese, Detection and Classifi- cation of Malicious Processes Using System Call Analysis, A Thesis In Drexel University, 2015.

[24] Perdisci, Roberto, Davide Ariu, Prahlad Fogla, Giorgio Giacinto, and Wenke Lee. McPAD: A Multiple Classifier System for Accurate Payload-Based Anomaly Detection. Comput. Netw. 53, no. 6 (April 2009): 864–881. https://doi.org/10.1016/j.comnet.2008.11.011.

[25] Ye, Qiang, and Weifeng Zhi. Outlier Detection in the Framework of Dimensionality Reduction. Interna- tional Journal of Pattern Recognition and Artificial Intelligence 29, no. 04 (March 12, 2015): 1550017.

https://doi.org/10.1142/S0218001415500172.

[26] PyNetsim [Online].

https://github.com/jjo-sec/pynetsim

[27] Virus Total [Online]. http://virustotal.com

[28] Rieck, Konrad, Philipp Trinius, Carsten Willems, and Thorsten Holz aff2n3. Automatic Analysis of Malware Behavior Using Machine Learning. J. Com- put. Secur. 19, no. 4 (December 2011): 639–668.

[29] Hui Suo, Jiafu Wan, Caifeng Zou, Jianqi Liu, Security in the Internet of Things: A Review LATEX, Guangzhou, China, 2012.

[30] Kai-Chi Chang, Raylin Tso, Min-Chun Tsai, IoT sandbox: to analysis IoT malware Zollard, Chang2017IoTST,2017.

[31] E. Leopold and j. Kindermann, "Text categoriza- tion with support vector machines," How to represent texts in input space? Machine Learning, p. 423–444, 2002.

[32] Qemu, http://wiki.qemu.org

[33] https://en.wikipedia.org/wiki/Principal component analysis

[34] Pritam Sahaa, Nabanita Roya, Deotima Mukher- jeea, Ashoke Kumar Sarkarb, "Application of Princi- pal Component Analysis for Outlier Detection in Het- erogeneous Traffic Data", The 7th International Con- ference on Ambient Systems, Networks and Tech- nologies (ANT 2016)