A Novel Framework to Classify Malware in MIPS Architecture-based IoT Devices

Tran Nghi Phu[1,2], Hoang Dang Kien[1], Ngo Quoc Dung[3], Nguyen Dai Tho[1,4], Nguyen Ngoc Binh[5]
[1] VNU, University of Engineering and Technology (VNU-UET), Hanoi, Vietnam
[2] People's Security Academy (PSA), Hanoi, Vietnam
[3] Posts and Telecommunications Institute of Technology (PTIT), Hanoi, Vietnam
[4] UMI UMMISCO 209 (IRD/UPMC), Hanoi, Vietnam
[5] The Kyoto College of Graduate Studies for Informatics (KCGI), Kyoto, Japan
Correspondence should be addressed to Tran Nghi Phu: tnphvan@gmail.com

Abstract: Malware on devices connected to the Internet via the Internet of Things (IoT)) is evolving and is a core component of the fourth industrial revolution. IoT devices use the MIPS architecture with a large proportion running on embedded Linux operating systems, but the automatic analysis of IoT malware has not resolved. We proposed a framework to classify malware in IoT devices by using MIPS-based system behavior (system call - syscall) got from our F-Sandbox passive process and machine learning techniques. The F-Sandbox is a new type for IoT sandbox, automatically created from the real firmware of the specialized IoT devices, inheriting the specialized environment in the real firmware, therefore creating a diverse environment for sandboxing as an important characteristic of IoT sandbox. This framework classifies five families of IoT malware with F1-Weight = 97.44%

Keywords: IoT devices; system call; MIPS; F-Sandbox ; malware classification.

1 Introduction

The rapid growth of the fourth industry, develop the Internet of Things (IoT) leads to an unprecedented revolution in the cyber-physical systems and provides rich utilities to users. It envisaged the number of interconnected devices to exceed 50 billion by 2020, with an estimate of about 8 devices per person [1]. Such an enormous amount will impact our digital lives in many application domains, transportation, healthcare, smarthome, smartcity, medical and health equipment, energy management, etc. However, in parallel with the developing of IoT technology, there is a security issue of information leakage when anything can become a spy device anytime, anywhere. Since the last decade, the number of malware on IoT devices has exploded. In the first half of 2018, there were more than 120,000 IoT malware instances detected by the Kaspersky IoT Lab [2]. That's more than triple the amount of IoT malware seen throughout 2017. Kaspersky Lab warns that the snowballing growth of malware families for smart devices is a continuation of a dangerous trend: 2017 also saw the number of smart device malware modifications rose to 10 times the amount seen in 2016.

Recorded attacks showed that target IoT devices have become critical. In September 2016, IoT malware built from Linux/Mirai malware responsible for 1.1 Tbps DDoS attacks directed at the Dyn

Domain Name System (DNS) provider [3]. In 2017, Linux/Brickerbot, a botnet similar to Mirai, infected more than 10 million IoT devices around the world [4]. There are many vulnerabilities that attackers can use to obtain privileges for IoT devices. OWASP has identified the ten main issues [5] in which insecure firmware, insecure web interface, and insufficient authentication mentioned.

These problems have caused research [6, 7, 8, 9, 10, 11, 12], the widespread attention of researchers such as IoT malware. These methods can be divided into two main categories: Static methods are expressed by analyzing and detecting malicious files without executing them. In static malware analysis, analysts reverse an executable file into assembly code to deepen their understanding of malware activity. The static analysis [8, 9] relies on extracting varieties characteristics from the executable such as Printable String Information (PSI), Function Length Frequency (FLF), Operational codes (Opcodes), n-gram of byte sequences, header section, and so on. Novom *et al.* [11] used fuzzy pattern tree for the opode of the executable to detect and classify in the IoT nodes. One of the major advantages of static analysis is the ability to observe the structure of malware. All possible execution paths in the malware sample without considering the processor architecture. Although static analysis has its advantages, it also has some limitations. The key disadvantage of this approach is that it is unable to detect complex and polymorphic malware. Therefore, the static analysis approach alone might not be sufficient to identify malware and should be complemented by the dynamic analysis [13]. The dynamic approach consists of monitoring executable files during its run-time to detect abnormal behaviors. This approach performed by collecting information such as API calls, network behavior, instruction traces, registry changes, memory writes, and so on during the running process. Based on these captured data, researchers have built machine learning (ML) or deep learning (DL) models to detect if a target file is a malware or not [11, 17, 22].

Almost all malware research is focusing on computing devices with the Intel architecture (x86-64) [14, 15] and recently has switched to develop frameworks to detect IoT malware such as [4, 16, 17, 18], especially with the ARM architecture. There have been a lot of frameworks to collect system calls of malware on the computer and mobile devices and automatically analyzed by machine learning. Deep4MalDroid [17] extracted the Linux kernel system calls from the executing apps on Android, generates a weighted directed graph and then applies a deep learning framework resting on the graph-based features. Martin *et al.* [18] proposed a framework for dynamic and static analysis with a large dataset extracted from Android applications and detected malware using a fusion of features and voting between classifiers. However, no framework can collect system calls of IoT samples, automatically analyze their logs, then measure classification model in a holistic of a full IoT dataset.

In general, the above frameworks have three main components: collection of IoT executable samples (including malware and benign executables), behavior extraction, and detection/classification. Concerning the first component, IoTPOT developed by Pa *et al.* [16] was the first honeypot to mimic IoT devices, allowing the authors to capture more than 4,000 IoT malware samples (according to [9]). Another IoT malware database, we can mention contains with more than 9,000 samples [20]. Besides IoT malware samples, it is crucial to collect benign files to be able to implement detection algorithms. Brash [21] has created 1,078 benign and 128 malware samples for ARM-based IoT applications. Similarly, Nguyen *et al.* [22] have collected 10,033 ELF files including 4,002 IoT malware samples and 6,031 benign files from different sources. Samples were then labeled as malware/benign with VirusTotal [23] before inputting for classifiers.

The second component comprises analyzing and logging executable files during its run-time to detect abnormal behaviors. To perform this approach, the most important part is to build a sandbox, called an emulator, so that executable files reveal all their behaviors. In such an environment, malware can only affect the virtual environments and not physical devices. Researchers use QEMU [24], a very popular open-source system emulator, to deal with this problem. QEMU supports many types of processors such as ARM, MIPS, PowerPC, x64, x86, etc that are popularly used in embedded devices [25]. Some works are focusing on developing sandbox such as [20, 19, 26, 27]. The sandboxes are used as a core of IoT malware analysis and detection frameworks such as [4, 16]. However, these frameworks exist drawbacks that limit the detection capabilities of malware. IoTBox [16] has built a sandbox to capture and analyze Telnet behaviors of IoT malware used for DDoS attacks. This approach could be useful for detecting network abnormal behaviors, but can not detect malware that behaves mostly inside the operating system of the device such as Linux/TheMoon [28]. Andrei *et al.* [4] proposed a framework to collect collect dynamic malware features based on the open-source Cuckoo sandbox [19] to determine whether a Linux/Elf file is

a malware or not. However, this framework is not presented on how the collected data was analyzed and the precision, accuracy of the obtained results. Rare [29] focused on how to activate malware on Router by discovering static and dynamic information to build a suitable environment for malware. But Rare only use OpenWrt to build emulated router, did not emulate NVRAM and did not mention detecting malware on the router. In the same approach, Chang *et al.* [30] proposed IoT sandbox which can support 9 kinds of CPU architectures including ARM (EL, HF), MIPS, MIPSEL, PowerPC, SPARC, x86, x64 and sh4. Then, machine learning or deep learning based classifiers used for classifying malware and benign files.

Current sandboxes have been built on basic environments, including a common Linux-based operating system and several additional monitoring tools. This strongly impacts on capture behaviors of a target executable file and the whole detection process. Firmwares are built and packaged for specialized devices, with specialized functions in many environments that have little in common. Many firmware programs cannot run in traditional malware analysis sandboxes based on basic environments. It is also hard to install required environments like firmware on the basic environment because it packages them in unpublished firmware. Therefore, we need to generate multi-vendor environments like firmwares. That means, IoT sandbox can set up not only an basic environment but also many environments like firmware of physical devices. To our knowledge, there are currently no sandboxes that can build environments based on an actual device firmware, can emulate NVRAM of devices and collect enough syscalls to classify malware.

The MIPS processor architecture [31] appears in many network devices such as routers, wireless transmitters, cameras [6, 25]. The program's behaviors are different when running on each different processor architecture and operating system. Therefore, studying malware on devices using the Embedded Linux operating system and MIPS processor (MIPS ELF) is necessary. Many studies have focused on detecting malware through the syscall behavior with good detection results [32, 33, 34], but to our knowledge, research has not been carryout, to detect malware via syscall in MIPS ELF. Previous studies focused on detecting malware on Windows operating system for the i386 processor architecture. Canzanese *et al.* [32] evaluated with many machine learning methods or deep learning applied with positive results. It can also detect malware on the Android operating system via syscall like [33]. Asmitha *et al.* [34] experimented with the method of detecting malware on Linux through syscall with many machine learning methods, but the dataset is small with only 668 samples and also done on the i386 architecture.

In this paper, we propose a novel framework that consists of analyzing MIPS ELF files based on syscall behaviors. We have collected and created a database of 3,773 MIPS ELF malware samples from Detux [20], IoTPOT [16], and VirusShare [35]. We have built a specific QEMU-based sandbox, which was inherited from Firmadyne [25] and Detux [20], aimed at monitoring system-call of the executable file. This sandbox, named F-sandbox, can self-configure the suitable requirement for a target MIPS executable file to reveal all behaviors. Within this framework, we also implement popular machine learning classifiers such as SVM, Random Forest, Naive Bayes to evaluate obtained data. The experiments show that our proposed framework can classify malware on MIPS with high accuracy by F1-Weight up to 97.44%. Our main contributions in this paper are:

- Proposing a novel sandbox which automatically set up the adaptive environment for activating MIPS ELF files.
- Comparing and selecting a suitable method of extracting features and machine learning classifiers for MIPS-based IoT malware detection purposes.
- Combining feature selection, feature reduction and machine learning methods with suitable parameters to build a novel framework for detecting MIPS-based IoT malware with high precision.
- Supporting the open source activities, we make our system available to the research community under an open-source license (GPL) to encourage further research into IoT. For more information about source code, please see *https://gitlab.com/Nghiphu/c500-sandbox*. The F-sandbox system and malicious code detection module are provided at address *http://firmware.vn*, in which the IoT dataset and instructional materials updated and provided to the community.

The remaining sections of the paper are organized as follows: Section 2 shows related works; Section

3 presents the F-Sandbox; Section 4 describes the IDMD framework; Section 5 is about experiments; Finally, the conclusions and directions of the next development are with Section 6.

2 Related works

To perform the dynamic analysis, the sandbox has a very important role. There are 2 types of sandboxes that are physical and virtual. Physical sandboxes are based on real hardware components such as RAM, CPU, network peripheral that give a better environment for the malware to reveal all their behaviors. However, physical sandboxes are difficult to customize, to store/restore their states. The type that is not suitable for IoT-based malware. Because of the diversity of hardware in IoT devices. Differs from PC, IoT devices have various processor architectures such as MIPS, ARM, PowerPC, SPARC. To deal with this problem, Virtual sandbox, based on the simulation and virtualization technology, is often used for monitoring executable files. In general, a virtual sandbox is built based on QEMU [24], a generic and open source machine emulator and virtualizer. QEMU supports 26 different CPU architectures, especially IoT processor architectures such as MIPS, ARM and supports Windows and Linux operating systems. The main challenge of the virtual sandbox is the difficulty of setting up the same function configuration, especially network peripherals. There are two main virtual sandboxes that we can mention, which are IoTPOT [16] and Detux [20].

Detux is an open-source sandbox based on QEMU that supports traffic analysis of the Linux malware in five different CPU architectures: MIPS, MIPSEL, ARM, x86, and x64. The first problem of Detux is that it does not virtualize network peripheral so malware can infect other devices through external connections. Second, the interaction with the operating system was not considered, so the information such as file creation, file deletion or read file are missing. Finally, the operating system to execute files in Debian Linux and it is not a common environment as used by IoT devices, therefore it could be not suitable for some malware.

IoTBox [16] is a sandbox aiming at analyzing network behaviors of IoT malware. It supports 8 different processor architecture as MIPS, ARM, MIPSEL, PowerPC, etc. IoTPOT can detect common types of DOS attacks such as SYN flood, UDP flood, ACK flood and scan (Telnet, UDP, TCP scan in some special port). DOS traffic will be blocked or allowed with low packet frequency to avoid harming the real system. However, limited to emulating only IoT devices from a few specific vendors, IoTBOX is not suitable for analyzing all kinds of malware that can infect IoT devices. When executing malware and monitoring, DOS attack behavior may not be immediately executed, therefore we have to wait for a long time to observe it.

Sandboxes capture two main data sources during the execution of target files that are system-calls and network behaviors. Syscalls can be collected with different tools such as *Strace* or *Kernel probes*.

- *Strace* [36] is a tool on Linux operating system that allows monitoring running programs, collecting the system-calls including the name, parameters, and results of calling the system-calls. *Strace* allows tracking processes generated from the process being tracked by *Strace*. *Strace* has been used in many works to collect the interaction between executable files in the Android environment [33] or Linux [34].

- *Kernel probes* abbreviated as *Kprobes* is a tool that allows dynamically breaking into kernel routine and collecting debugging information, one of the information kinds is a syscall. Our sandbox used *Kprobes* built-in Kernel to monitor syscalls of all processes running on the firmware of an IoT device. Sometimes using *strace* is not suitable for a tracking system called. First, many processes could run before starting *Strace*, therefore we can miss system calls. Second, *Strace* can be easily resisted by some anti-debug technical as check *ptrace* status in source code. During the process of collecting syscalls, in our experiments, we have encountered such malware samples.

Concerning network behaviors, the popular tool that could be used is InetSim [37]. InetSim is a Linux-based software package, containing Perl scripts used to simulate many network services such as DNS, HTTP, FTP, etc. PynetSim [38] is considered being an upgrade to improve InetSim for IoT devices. PyNetSim is developed in Python3, which allows detecting malicious code protocols, supporting scripts to interact with IoT malicious codes like bot DDoS, Mirai and LizardStresser.

It is important to note that these previous works did not focus on virtualizing network peripheral. This point limits considerably monitoring the IoT malware behaviors, especially IoT botnet malware. This issue is indirect solving by Chen *et al.* [25]. They proposed a framework, named Firmadyne, aiming at emulating router firmware's web-interface using QEMU. Firmadyne can auto-configure a suitable emulation environment for a wide range of the router, enabling a dynamic analysis of 23,035 firmware images gathered from 42 devices vendors. This method does not rely on physical hardware to perform the analysis like Avatar [39] but Firmadyne emulates the firmware non-volatile memory to execute the firmware web-interface. Once the router web-interface is emulated, the popular scanning framework Metasploit is used for exploring vulnerabilities and its corresponding exploits. However, Firmadyne analyzes only the web interfaces and ignore the firmware operating system execution. Hence, these methods can not trace abnormal behaviors to detect malware as shown in the analysis of Linux/TheMoon and Linux/Mirai malware experimentation. Based on Firmadyne, we have built a novel sandbox that can set up the network configuration for most IoT executable files including malware and benign. This sandbox, named F-Sandbox, is presented in the next section and it is a part of the IDMD framework.

3 F-Sandbox architecture

Our proposed sandbox uses instrumentated kernel that was built with *Kprobes* allow tracing Linux kernel function calls and any instruction inside the kernel and inspecting the registers. Using INetSim/PyNetSim to simulate the Internet, the F-Sandbox provides a full environment that reveals behaviors of an ELF file, including both syscalls and network interaction [40] during its runtime. The collected results from F-Sandbox will be static analysis information of the sample, network data generated in the pcap's form file, log INetsim/PyNetSim about information interacting with the Internet and other *syscall* system behavior in the form of a file *syscall* log. F-Sandbox structure has 4 main components: Sandbox Controller, Virtual Machine, QEMU Monitor and INetSim/PyNetSim server shown in Figure 1.

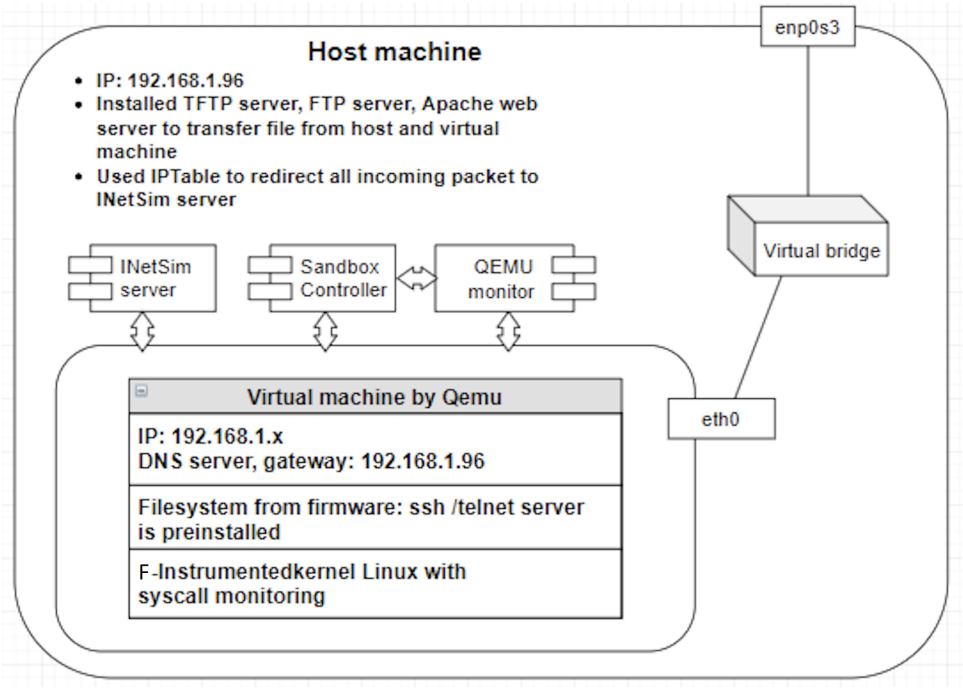

Figure 1: F-Sandbox structure

- *The Sandbox Controller* interacts with the QEMU monitor through calling commands that display network configuration and restore snapshots. The sandbox controller interacts with the virtual machine by calling SSH/Telnet procedures to the virtual machine, transmitting executable files

from the real machine inside, granting execution rights, requesting file execution and getting log files. It supports two methods, SSH and Telnet, to connect and interact virtual machine while Detux only supports SSH, this is our improvement over Detux. The sandbox controller operates the virtual machine by Telnet protocol, the file will be transferred to a virtual machine by *wget*, *ftp* or *tftp* instead of *scp* depending on the specific environment of the firmware.

- *The virtual machine* based on QEMU includes 2 components: QEMU image and Linux kernel. The image of QEMU used in F-Sandbox containing the file-system is extracted from the firmware by the Firmadyne Extract tool. It is added more tool and modified some configuration info to emulate in QEMU but still fully inheriting the packaged environment of the firmware. The Linux kernel used in the F-Sandbox is the F-Kernel which improved from the Firmadyne instrumented kernel. The instrumented kernel of Firmadyne can only listen to 15 syscalls of the system, therefore we extend to listen to all syscalls of the system aiming to collect all malicious code behavior. Collecting all syscalls can affect the speed of the F-Sandbox, in many situations it is only necessary to collect syscalls with a high frequency, therefore the F-Kernel allows flexible configuration of syscall quantities to be collected with different thresholds by setting *syscall* parameter when starting the F-Sandbox. The virtual machine interacts with the INetSim server through sending requests (HTTP, FTP, DNS, etc) and recovers fake responses from INetSim. *enp0s3* is an Ethernet Network Interface of the host machine, *eth0* is the Ethernet Network Interface of the virtual machine. A virtual bridge connected host machines and virtual machines.
- *The monitor* performs virtual machine interaction through snapshot restore; it is the same function of Detux.
- *InetSim server* performs a simulate of the Internet environment, it will redirect all network traffic to the InetSim server via IPtables and DNS poisoning. This is one of the best tools for simulation of network service, it helps us to emulate the network environment to make malware reveal behaviors as connecting to a server, scanning port without harming the real Internet environment. The INetSim log will record the interactions with the Internet.

We use QEMU as shown in Figure 2 to emulate firmware with loaded ELF files. A good deal of firmware had an SSH server, other ones had only Telnet server. Hence, the F-Sandbox controller provides both Telnet and SSH connectors to interact with the two types of firmware. Our method of transferring files to the virtual machines is also very diverse, we created an FTP server, an Apache web server and a TFTP server used by the virtual machine to download malware file and execute it.

The sandbox controller calls a procedure to start the virtual machine and QEMU monitor, initializes the virtual machine from the existing state by using the image recovery function or configuring the network, installing required packages and creating a snapshot to restore after running a sample.

4 IoT dynamic malware detection framework

We propose an IoT dynamic malware detection (IDMD) framework to learn and classify IoT malware, all steps as shown in Figure 4. The IDMD framework includes two phases: classification model generation and classification. In the classification model generation phase, labeled it to extracts ELF samples feature by feature vectors generation function, then these feature vectors with their labels are used to train a model. The feature vectors generation function includes three steps: sandbox initialization, sandbox execution, and feature selection. In the classification phase, unlabeled samples will be extracted feature by the feature vector generation function with generated parameters from the classification model generation phase, then the extracted feature vector will be classified by the generated model from the classification model generation phase.

4.1 Sandbox initialization

As mentioned above, our sandbox can build adaptive environments with samples by using suitable firmware. To select the suitable firmware for an ELF file to be executed, this step extracts neces-

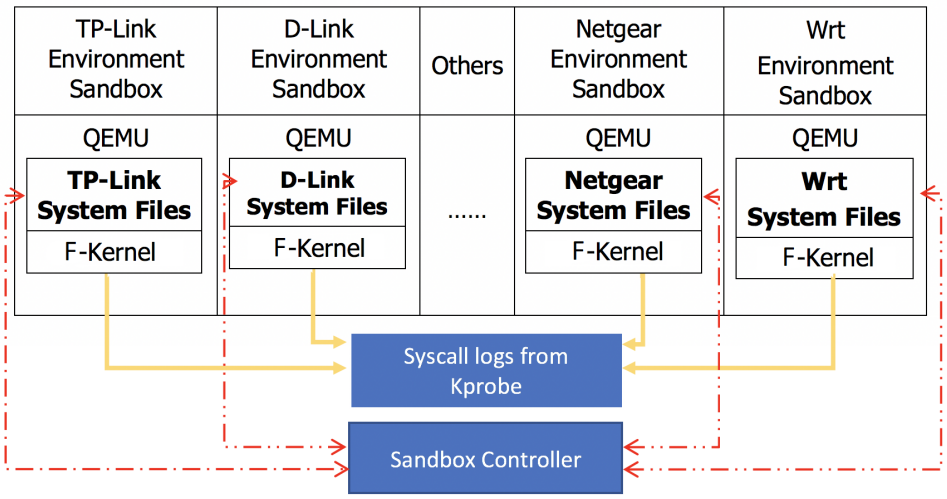

Figure 2: F-Sandbox with Multi-vendor environments

```
[ 1820.448000] fm: open[PID: 1121 (modprobe)]: file:/lib/libm.so.0
[ 1820.448000] fm: read[PID: 1121 (modprobe)]: fd:0, size:4096
[ 1820.448000] fm: close[PID: 1121 (modprobe)]: fd:0
[ 1820.448000] fm: open[PID: 1121 (modprobe)]: file:/lib/libc.so.0
[ 1820.448000] fm: read[PID: 1121 (modprobe)]: fd:0, size:4096
[ 1820.448000] fm: close[PID: 1121 (modprobe)]: fd:0
[ 1820.448000] fm: open[PID: 1121 (modprobe)]: file:/lib/libc.so.0
[ 1820.448000] fm: close[PID: 1121 (modprobe)]: fd:0
[ 1820.448000] fm: open[PID: 1121 (modprobe)]: file:/lib/modules/2.6.32.70/modules.dep
[ 1820.448000] fm: open[PID: 1121 (modprobe)]: file:/lib/modules/modules.dep
[ 1820.452000] fm: write[PID: 1121 (modprobe)]: fd:2, size:35
[ 1820.452000] fm: do_exit[PID: 1121 (modprobe)]: code:256
[ 1820.452000] fm: open[PID: 1123 (modprobe)]: file:/lib/libm.so.0
[ 1820.452000] fm: read[PID: 1123 (modprobe)]: fd:0, size:4096
[ 1820.452000] fm: close[PID: 1123 (modprobe)]: fd:0
[ 1820.452000] fm: open[PID: 1123 (modprobe)]: file:/lib/libc.so.0
[ 1820.452000] fm: read[PID: 1123 (modprobe)]: fd:0, size:4096
[ 1820.452000] fm: close[PID: 1123 (modprobe)]: fd:0
[ 1820.452000] fm: open[PID: 1123 (modprobe)]: file:/lib/libc.so.0
[ 1820.452000] fm: close[PID: 1123 (modprobe)]: fd:0
[ 1820.456000] fm: open[PID: 1123 (modprobe)]: file:/lib/modules/2.6.32.70/modules.dep
```

Figure 3: Captured system-calls log of a malware sample

sary information such as: Architecture (MIPS, ARM, etc), the endianness (Big-endian/Little-endian), bit-length (16/32/64). This static information is extracted by using *file* utility in Linux. It helps our framework to select suitable firmware, among our database of 253 real MIPS-based firmware of D-Link, TP-Link, Asus, Belkin, Linksys, etc. The selected firmware is used to initialize our Sandboxes. The selected firmware has many parameters such as what types of transfer protocol they supported (FTP, TFTP, SCP), what control protocol they supported (SSH, Telnet). This information will configure the sandbox controller, start and control the sandbox. It is not trivial to determine which firmware is the most suitable to an ELF file, therefore, for each ELF file, our sandbox loads selected firmware one by one to perform with, and the file with the most captured system-call will be kept for the next steps.

4.2 Sandbox execution

After initializing the sandbox with suitable required parameters, the sandbox controller transfer the input ELF file into the sandbox. Then, by granting all permissions to the ELF file, F-Sandbox starts the monitoring function for 30 seconds. *Kprobe* will collect the *syscall* generated from the running program

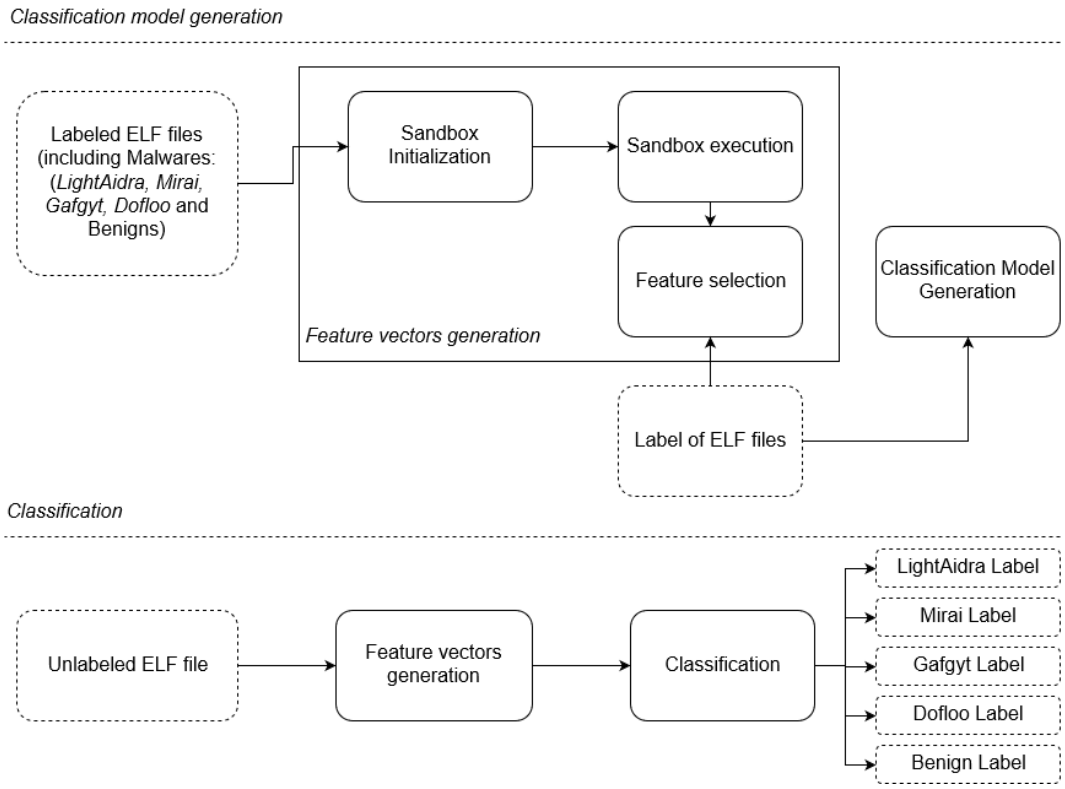

Figure 4: Proposed process for IoT malware classification

and its child processes. Therefore, it will be analysed and filtered to capture all log by the input ELF file. If the malware sends a network request outside, it will be automatically redirected to INetSim via Iptables and the built-in DNS-poisoning technique in InetSim and INetSim server will return the fake response of the same type as the object that the template requires. It presents a sample of captured syscall log in Figure 3.

4.3 Feature selection

To avoid the training time-consuming and reduce the risk of overfitting, we try to extract the most relevant features related to the label and then apply machine learning algorithms to improve the performance of the model. In this paper, there are two main selection steps:

- The selection of quantity and name of *syscall* among log files.
- The selection of feature vectors among n-gram features generated by selected *syscall*

The MIPS architecture Linux operating system has 345 different *syscalls*. However, they are not used by performed ELF files, especially ELF malware. The average used syscalls by 3,223 ELF files is about 146. The syscalls that are *read* and *sentto* are the most used ELF file which represents 86% of all captured log files. 30 most frequent *syscalls* represent over 99% of the whole captured ones. In our experiments, the more syscalls monitoring, the slower the performance system. When we try to implement F-Kernel monitoring of more than 45 syscalls by *Kprobe*, F-Sandbox slowdowns, leads to programs not working because of a timeout and the calculation time. F-Sandbox used *Kprobe* which is dynamic instrumentation by trap instructions and then looking up handlers will incur an enormous overhead. It is because of this reason that when we try to implement too many syscalls, our system will slowdown. Therefore, instead of taking into account all captured, we remove every *syscalls* except for the 30 most frequent ones and then apply the n-gram technique to construct feature vectors afterward.

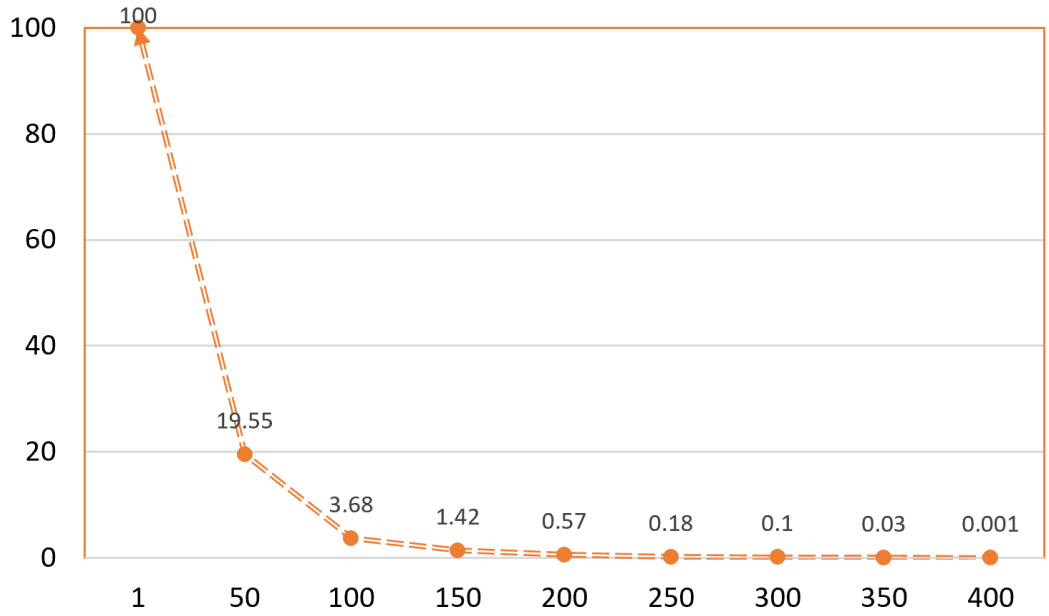

Figure 5: The ratio between the total features scores and the sub-features scores ranking order by descending

The n-gram is the method of counting the occurrences of n elements standing close to each other in a sequence, it stores the number of these occurrences in a vector to characterize that string. It has proven this method effective in detecting malware based on syscall [14]. In our research, we use n-gram with $n = 1, 2, 3$ to get the feature vectors of the *syscalls* sequence. With the selection of the syscall set as above, the vector is characterized by 2-gram and 3-gram directions with how to choose 30 features are $30 \times 30 = 900$ features and 27,000 features.

To select n-gram features that have the strongest relationship with the label, it can use efficient feature reduction techniques based on scores such as Chi-square or Information Gain [41]. But Chandra *et al.* [42] said that most of their cases, Chi-square was ahead or at part with Information Gain. In our experiments, Chi-square is more efficient than Information Gain. Therefore, our framework used Chi-square as a feature reduction method. The Chi-square score is computed between each feature and the target, afterward we select the desired number of features with the best Chi-square scores. It determines if the association between two categorical variables of the sample, would reflect their real association in the population and it is the technique we chose for selecting the most relevant features in this paper. We will rank features according to the following formula:

$$X^2(D,t,c) = \sum_{e_t \in \{0,1\}} \sum_{e_c \in \{0,1\}} \frac{(N_{e_t e_c} - E_{e_t e_c})^2}{E_{e_t e_c}} \tag{1}$$

Where $e_t = 1$ when the content of D contains a characteristic, otherwise $e_t = 0$; $e_c = 1$ when D belongs to class c, otherwise $e_c = 0$. N is the observed frequency of D and E as the expected frequency. We choose the number of feature vectors based on the ratio between the total features scores and the sub-features scores ranking order by descending. We determine that 350 features having the highest score represent 99.9% of the total score of 900 features (n=2) as shown in figure 5. These selected features will then input to the classification step.

4.4 Classification

The machine learning methods are well studied in detecting malware, each method will give the result for different datasets. Souri *et al.* [43] have reported many recent studies and evaluated three popular

Table 1: Number sample of each class

Label	Lightaidra	Mirai	Gafgyt	Dofloo	Benign
Number of samples	796	477	132	241	280
Average length of syscall log	354	3.012	300	691	286

research methods with malware detection and the best result is Support Vector Machine (SVM), Random Forest (RF) and Naive Bayes (NB). The SVM classification method has performed well on traditional classification tasks, and we selected it for the text classification systems, intrusion detection systems and promotes the ability with the n-gram feature extraction method. The advantage of this method is that its general capability can be improved by using the principle of structural risk minimization. SVM uses a kernel function to map training data into a higher-dimension space so that the problem is separable. Decision Tree is a structured hierarchical tree used to classify objects based on a series of rules. Random Forest uses many trees and forecast by averaging the predictions of each component tree. Naive Bayes is a probability-based classification method, which gives good results in detecting malicious code [44].

5 Experiments

5.1 MIPS IoT samples

An IoT sample dataset used for testing is a set of 3,403 MIPS ELF samples, including 3,223 malware samples and 228 benign files. Malware datasets collected from 2 different sources: Detux [20] and IoTPOT team [16]. Detux has collected more than 9,000 samples, including more than 3,200 MIPS ELF samples. IoTPOT has collected 4,000 samples, of which 938 MIPS ELF samples and only 38 samples coincide with the MIPS ELF pattern of Detux. To label the experimental samples, there are 2 main options which are an evaluation based on a reputable antivirus like Kaspersky [45] or antivirus software combination [15]. We labeled the templates based on Virustotal [23], first identify malicious designs that are Virustotal samples discovered by over 10 software, including reputable software Kaspersky, Avast, Avg, Symantec, we got 3,223 malicious samples. We then labeled this 3,223 malware based on the antivirus combination, which uses Symantec's main label, the antivirus engine has a good detection capability and a clever naming scheme. In the malicious dataset, there are 37 different families, many popular and popular malicious codes like LightAidra / Aidra / Zendran, Dofloo / Spike / MrBlack / Wrkatk / Sotdas / AES.DDoS / DnsAmp, Gafgyt / BASHLITE / Lizkebab / Torlus, Tsunami / Kaiten, SecurityRisk, Moose, Hajime, Trojan.Gen etc, but to ensure the classification accuracy, avoid the imbalance in the number of samples of the classes in the training process, articles select only 4 typical malicious family names and have a sample number greater than 100. We show the distribution of the malicious codes in the IoT dataset is shown in Figure 6.

We collect the benign dataset from the basic programs available on Embedded Linux, BusyBox built-in programs, Embedded Linux kernel, and some MIPS-based basic applications. The number of clean samples collected is 280 samples. Devices using MIPS chips are specialized devices, there are very limited application stores, the number of clean programs collected is therefore also limited. Thus, the input dataset has five labels, four malware labels including LightAidra, Mirai, Gafgyf, Dofloo, and one Benign label.

5.2 Collecting syscall log

According to [14], the syscall log must be collected within a certain time, the minimum length of the syscall log is 1,500, which will show the result in the best detection with malware on Windows i386 architecture. The numbers generated during the first 30s-focused sandbox implementation, when increasing this time to 60s, 90s, 120s, the syscall number obtained does not change much, so we continue to collect the log for each sample in 30 seconds. The average syscall log length of the sample set is lower than the threshold of 1,500 syscalls and for each type of malware, it gives the average length of the different logs, which is the average length of malicious code families shown in Table 1.

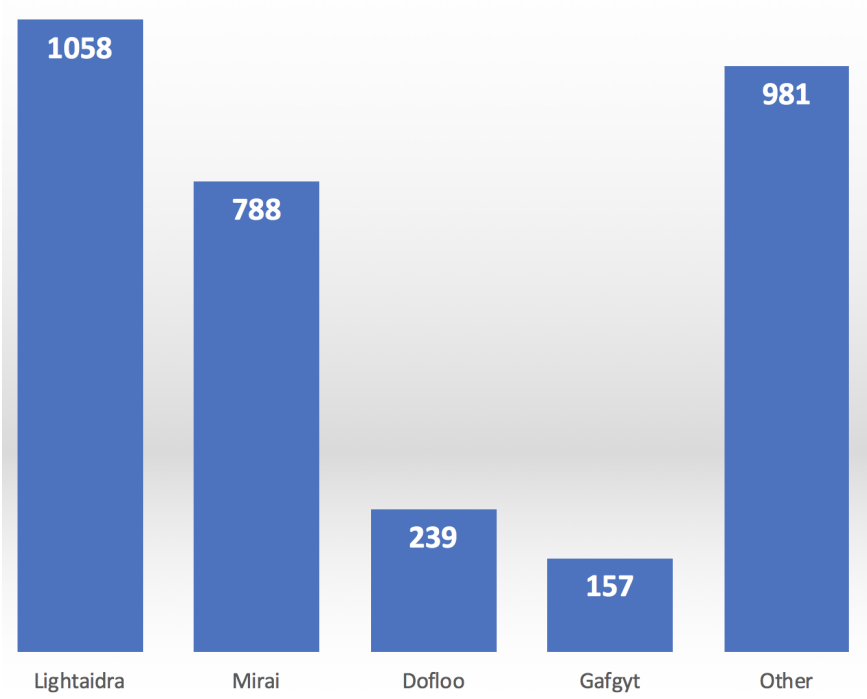

Figure 6: Types of malware in the IoT dataset

Table 2: Evaluating the number of syscall samples

Label	**50**	**100**	**200**	**300**	**400**	**500**	**1,500**
Number of benign samples	280	259	145	123	104	94	6
Number of malware samples	2,370	2,288	2,165	2,142	765	756	67

By analyzing the collected logs when executing the templates in F-Sandbox, the log files of length less than 50 are generated by samples with errors such as insufficient parameters to operate, lack of libraries, errors initialize and so on. A sample when is normally active has a length of syscall log more than 50, the number of programs with several syscalls over 1,500 such as [14] is not large, most programs call the number of syscalls in the range from 50 to 500 as shown in Table 2. The range of syscall's length MIPS ELF files is shorter which can also be explained because programs written on IoT are often simpler on multi-purpose computers due to constrained resources and functions.

From the initial process created when running the sample, the malicious code creates a lot of child processes, in which malicious code creating 1,000 child processes, and most malicious code creates 3 child processes. It shows the number of generated child processes is shown in Figure 8. Therefore, when collecting the syscall log of the sample, it is a syscall log of all processes generated when executing that pattern.

Analyzing the collected log file, the malware only uses 136 syscalls, the benign set uses 127, most of the syscall of 2 episodes overlap, there are 160 syscalls appear in both sets. Distributing irregular patterns, 2 syscalls read and send to occupy 86% of the system. It shows detailed statistics of syscalls in Figure 7. The data feature syscall log is also consistent with the published behavior of malicious code is sending information to the destination.

Through analysis, malware on IoT devices also has hidden features such as detection of analysis programs. Several malware samples can detect *Strace* tracking, so it does not activate them. Some previous observations suggest that malicious code on IoT is simple but still uses common hidden techniques of malware.

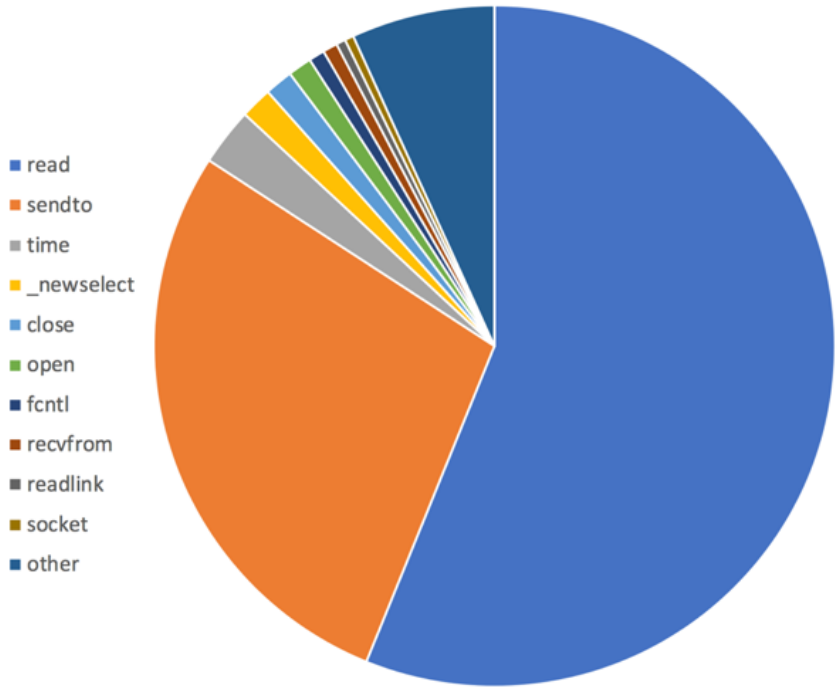

Figure 7: The rate of system calls

5.3 Performance measures

We measure the effectiveness of the classifiers in terms of precision, recall, and F_1 score. It computes these three quantities on a per-class basis. The precision for a class C_k is the fraction of processes classified as C_k,

$$PrecisionC_k = \frac{TPC_k}{TPC_k + FPC_k} \tag{2}$$

Where TPC_k and FPC_k are the numbers of true positives and false positives predicted by the classifier, i.e., the number of processes correctly and incorrectly classified as belonging to C_k. The precision measures the relevance of the positive classifications. A precision of 1 indicates that the classifier is always correct when it classifies a process as belonging to class C_k, whereas a precision of 0 shows it is never correct when it does so. The recall of a class C_k is the fraction of the processes belonging to C_k in the ground truth that are correctly classified,

$$RecallC_k = \frac{TPC_k}{TPC_k + FNC_k} \tag{3}$$

Where FNC_k is the number of false negatives, i.e., the number of instances of C_k misclassified as belonging to another class. The recall measures the sensitivity of the classifier. A recall of 1 shows that a classifier correctly identifies every instance of class C_k, whereas a recall of 0 indicates that a classifier never identifies instances of C_k. The F_1 score of a class C_k is the harmonic mean of the precision and recall of that class. An F_1 score of 0 indicates 0 recall or 0 precision, whereas an F_1 score of 1 indicates perfect recall and precision. In this study, per-class F_1 scores are averaged over all the classes to provide an overall characterization of a classifier. We consider three averaging techniques accounting for the unbalanced representation of the malware classes used in this study.

- Micro-averaged F_1 score: The F_1 score computed from the aggregate set, characterizing classifier performance on large classes.
- Macro-averaged F_1 score: The average of the F_1 scores of the classes, characterizing classifier performance on small classes.

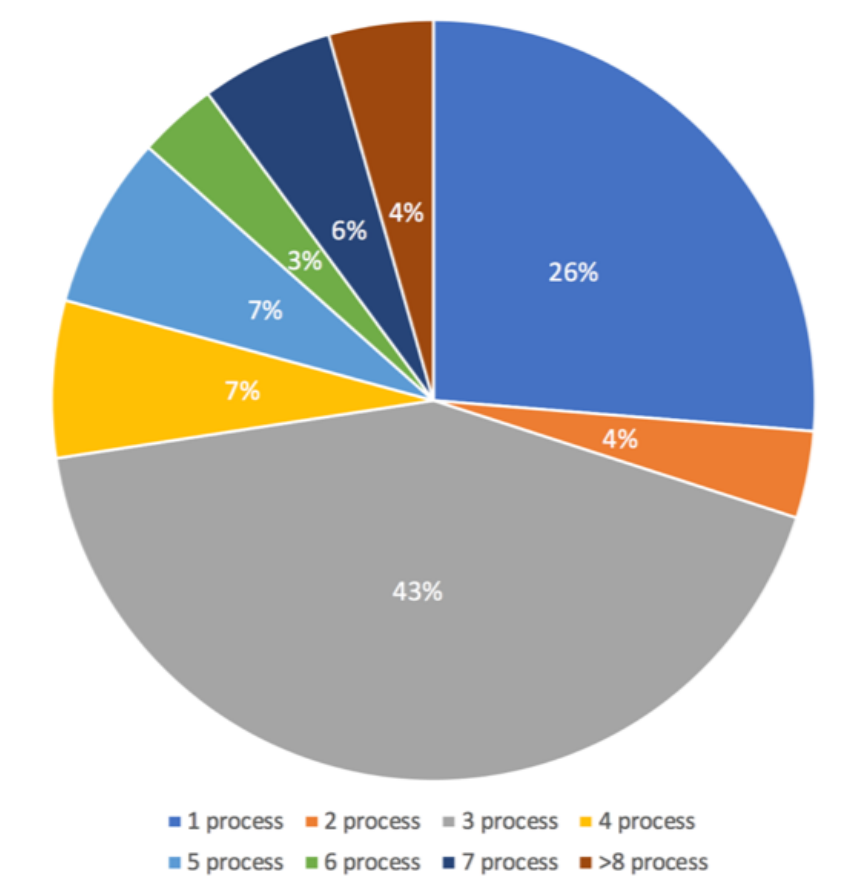

Figure 8: Statistics of the number of processes created by the IoT dataset

- Weighted F_1 score: The weighted average of the F_1 scores of the classes, with weights proportional to their support in the ground truth.

5.4 Training and evaluation

We installed the test based on the Scikit-learn Python library 0.19.2 on laptop Core i5 MacBook Pro, Ram 16 GB. Build 10 datasets generated from the IoT dataset according to the minimum length of syscall log n, with $n = 50, 100, 150, 200, 250, 300, 350, 400, 450, 500$ to determine which threshold is the most suitable for detecting MIPS ELF malware classification. If the length of the selected log file is too small, there will not be enough information to determine the program characteristics. On the contrary, if the length of the log file is too large, many programs that simply click are not evaluated and the ability to detect malicious code will be delayed. The goal is to determine the threshold as small as possible and measurement is most suitable, so that means the ability to classify and detect results very early when implementing the sample.

With the method of extracting n-gram characteristics, if n is too small (1-gram), the information got shows only the frequency of single syscall occurrences. If n is too large, the number of the characteristic collections is very large and the characteristics obtained are sensitive to the transformation techniques of malware. Recent studied experiments [32] [45] gave the best results with 2-gram and 3-gram, so in this model test with 2-gram and 3-gram.

With the n-gram characteristic got to reduce the dimension with Chi-Square with the number of dimensions reduced to K. Based on the experiments in [45, 46], we set $K = 350$.

We experimented with the dataset to select the best parameters for SVM, NB, and RF using the grid search method. For SVM, we classify using OVO method, use the *Sigmoid* kernel, and find the optimal

parameter C in the range [1-100,000], the *gamma* value in the range [0.0000001-1]. For RF we find two pros-parametric parameters, the feature function, which is one of three functions [sqrt, log2, sqr] and the number of *n_estimators* in the range [2-700].

Cross-validation is a popular method in machine learning to assess a detection/classification method's outcomes gained from experiments that can be generalized into an independent sample. It consists of many techniques such as Repeated Random Sub-sampling, K-fold, and Leave-Out. K-fold involves randomly dividing the set of observations into K groups, or folds, of equal size. The first fold is treated as a validation set, and the method is fit on the remaining K-1 folds [47]. The K-fold validation is suitable for limited size datasets [48]. In the experiments, we use a 5-fold cross-validation method with the selected parameter set in training and evaluating step. It divides data into different five sections, four of them are used as training data and the remained section is used as testing data for each experiment. Measures are calculated as the average of five times in these experiments.

5.5 Experimental results

The table in Figure 9 represents the overall results of F1-Macro, F1-Micro and F1-weight corresponding to different machine learning methods and threshold syscall lengths. The results show that the models for the measurements are approximately the same, altogether about one value and there are no many deviations that demonstrate a good model.

Len	F1 - Macro			F1 - Micro			F1 - Weight		
	RF	SVM	NB	RF	SVM	NB	RF	SVM	NB
50	85.53	80.31	67.71	92.89	91.69	68.87	91.98	89.63	67.06
100	86.10	85.15	69.51	93.03	92.02	69.46	92.23	91.32	67.85
150	85.45	83.17	70.69	93.9	92.45	70.49	92.79	91.33	69.26
200	85.98	83.04	78.51	93.95	92.71	84.96	92.95	91.52	83.88
250	85.58	82.83	78.29	94.00	92.63	84.90	92.88	91.44	83.78
300	85.53	81.28	78.21	94.15	92.83	84.95	93.01	91.13	83.91
350	85.05	81.30	73.86	93.73	92.49	79.79	92.65	90.93	78.50
400	**92.55**	88.49	79.24	**97.56**	96.13	93.67	**97.44**	96.01	93.40
450	92.22	90.48	**80.09**	97.53	96.08	**94.17**	97.42	96.04	**94.00**
500	92.41	**90.94**	79.79	97.51	**96.63**	93.97	97.39	**96.64**	93.76

Figure 9: F1-Macro, F1-Micro and F1-weight with 2-gram feature

The charts in Figures 10 11 12 compare the results of identification methods RF, SVM and NB with 2-gram and 3-gram feature. Experimental results show that 2-gram gives the best value for the classifier with all three algorithms RF, SVM, and NB.

The charts in Figures 13 and 14 compare the F1-Weighted value of 3 sets classifier with different minimum syscall log thresholds. The experimental results show that RF gives the best classification ability, then to SVM and NB. From both charts above it can be seen that the minimum syscall log threshold for classifying malicious code on the MIPS ELF set is 400.

From the experimental results, we can see:

- With MIPS ELF samples, our framework gets the highest results with the 2-gram feature extraction method with the Random Forest classification method, the minimum syscall length threshold is 400. Compared to the minimum length of the syscall log evaluated by Canzee *et al.* [32] with malware dataset on Windows is 1,500, the MIPS ELF file is smaller, the reason why it can be explained is because of the simple embedded program sets. There are fewer functions than Windows, which is also the feature of embedded software. That may also be the reason for the 2-gram explanation for the ability to classify better than 3-gram.

- The results in the IoT dataset are good and gather classification from objective sources, thus proving that F-Sandbox works correctly and efficiently.

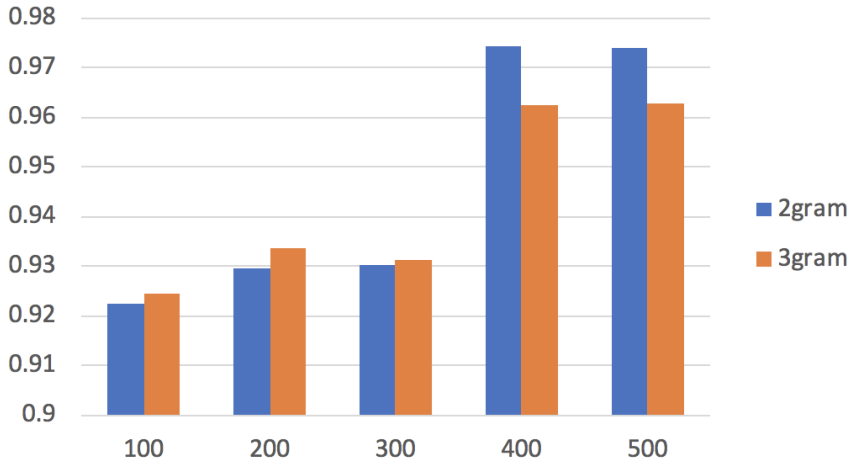

Figure 10: Random Forest Classification with 2-gram and 3-gram

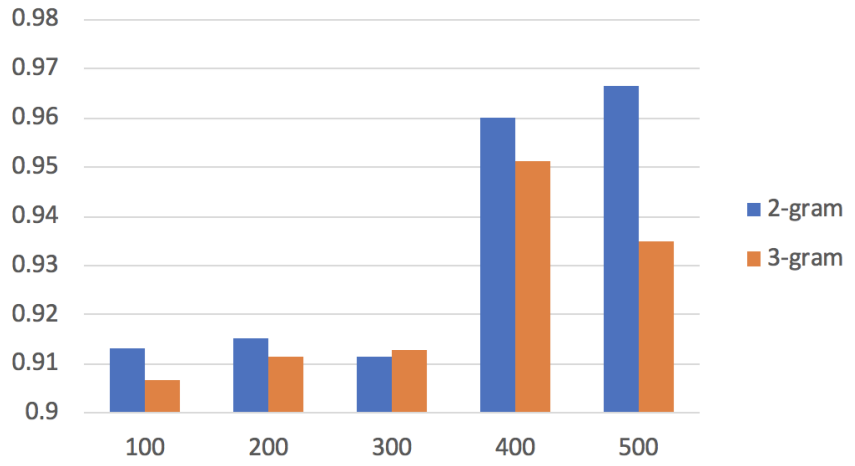

Figure 11: SVM Classification with 2-gram and 3-gram

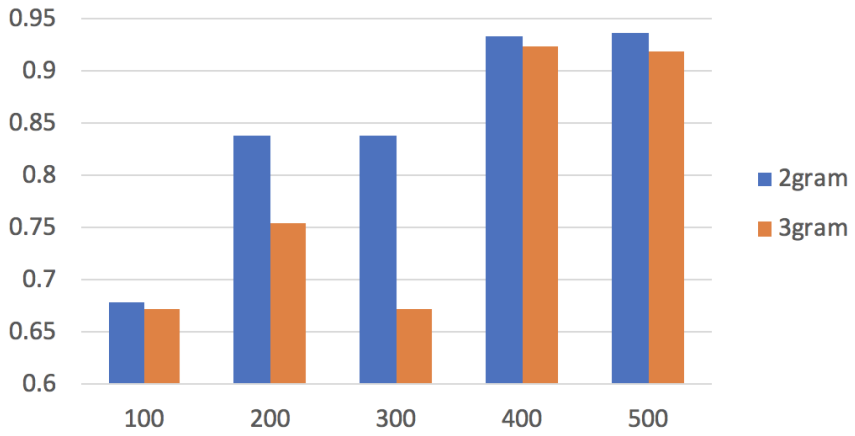

Figure 12: Naive Bayes Classification with 2-gram and 3-gram

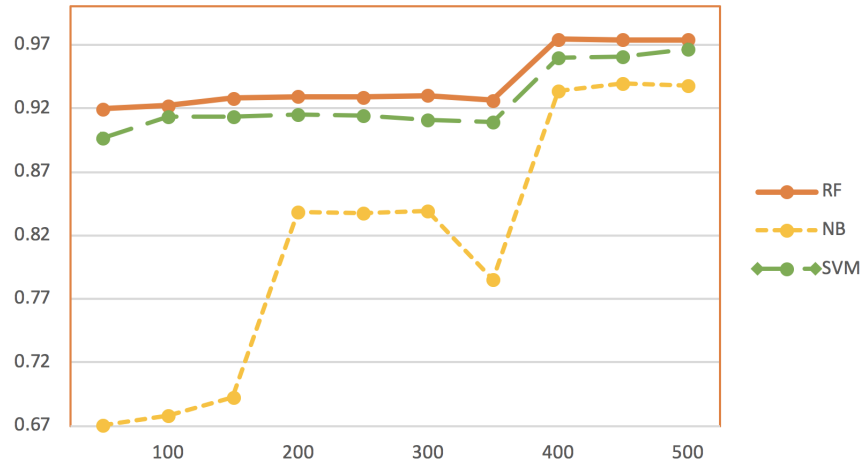

Figure 13: F1-Weighted with 2-gram

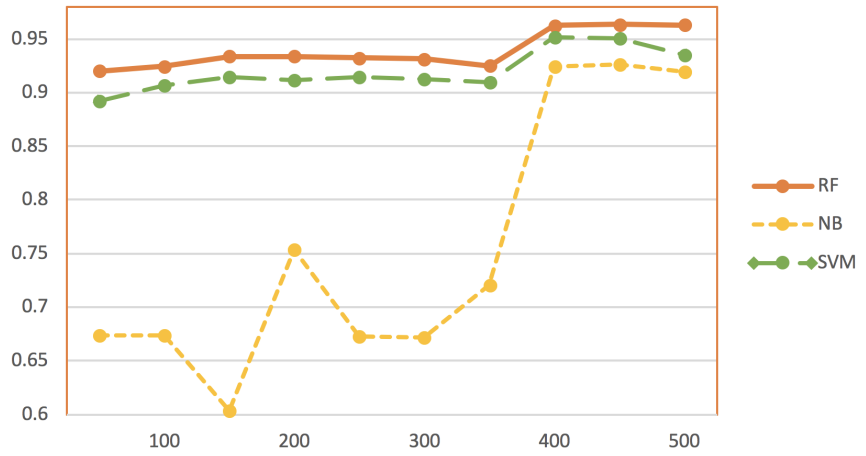

Figure 14: F1-Weighted with 3-gram

6 Conclusions and future work

In this paper, we proposed a framework for detecting and classifying MIPS ELF malware based on syscall and machine learning methods. This is the first study specializing in malware in MIPS ELF. In particular, the main contribution is MIPS IoT dataset, a novel type for IoT sandbox, based on the improvement and integration of Detux sandbox and Firmadyne emulation. We also collected, standardized the MIPS IoT dataset. In theory, the study has shown many characteristics of the malware type in MIPS ELF, finding the most suitable methods and parameters for detecting MIPS ELF malware based on machine learning methods. To the best of our knowledge, this paper is the first to provide a comprehensive framework, including not only an overall methodology but also a sandbox environment and related tools, for collecting MIPS-based malware syscalls, classifying them to their respective families, and evaluating the classification performances. We provided the generated IoT dataset used in our experiments and the source code of our sandbox tool to researchers at sites *https://gitlab.com/Nghiphu/c500-sandbox* and *http://firmware.vn* for academic purposes.

In future work, we continue to develop F-Sandbox so that malware can execute at high rates, exposing more behavior. In this framework, we have only exploited the information about syscall obtained from F-Sandbox to detect malware, other information such as network behaviors, process states will be further considered and researched. Static information not only uses to initialize F-Sandbox but also combines

with dynamic behaviors to detect/classify malware. Therefore, our works in near time integrate static information, static feature, system behaviors and network behaviors that extracted from F-Sandbox to detect/classify malware. F-Sandbox supports and experiments for the MIPS architecture, therefore next time, we will deploy for another architecture such as ARM, PowerPC.

References

[1] *The internet of things: How the next evolution of the internet is changing everything.* URL http://www.cisco.com/ [Accessed: 02-May-2019].

[2] Kaspersky IoT Lab Report. *New IoT malware grew three fold in H1 2018.* [Online]. Available: https://www.kaspersky.com/about/press-releases/2018_new-iot-malware-grew-three-fold-in-h1-2018. [Accessed: 02-May-2019].

[3] Kishore Angrishi, *Turning Internet of Things into Internet of Vulnerabilities: IoT Botnets*, ArXiv170203681v1 CsNI, February 2017, p. 13-19.

[4] Andrei Costin and Jonas Zaddach. *IoT Malware: Comprehensive Survey, Analysis Framework and Case Studies.* BlackHat USA, 2018.

[5] Internet of Things Top 10 Project. www.owasp.org/ [Accessed: 05-May-2019].

[6] Andrei Costin, Jonas Zaddach, Aurélien Francillon, and Davide Balzarotti, *A large-scale analysis of the security of embedded firmwares*, in Proceedings of the 23rd USENIX Security Symposium, 2014, p.95-110 [Online].Available: https://www.usenix.org/conference/usenixsecurity14/ techincal-sessions/presentation/costin.

[7] Drew Davidson, Benjamin Moench, Thomas Ristenpart, and Somesh Jha. *FIE on Firmware: Finding Vulnerabilities in Embedded Systems Using Symbolic Execution.* 22nd Security Symposium (USENIX), 2013, p.463-478.

[8] Akshay Kapoor and Sunita Dhavale. *Control Flow Graph Based Multiclass Malware detection using Bi-normal Separation, Defence Science Journal*, DESIDOC, vol.66, no.2, p.138-145, 2016.

[9] Mohannad Alhanahnah, Qicheng Lin, and Qiben Yan, *Efficient Signature Generation for Classifying Cross-Architecture IoT Malware*, IEEE Conference on Communications and Network Security (CNS), 2018, p.1-9.

[10] Yan Shoshitaishvili, Wang Ruoyu, Hauser Christophe, Kruegel Christopher and Vigna Giovanni, *Firmalice-Automatic Detection of Authentication Bypass Vulnerabilities in Binary Firmware*, NDSS, 2015.

[11] Ensieh Modiri Dovom, Amin Azmoodeh, Ali Dehghantanha, David Ellis Newton, Reza M. Parizi, and Hadis Karimipour. *Fuzzy Pattern Tree for Edge Malware Detection and Categorization in IoT.* Journal of Systems Architecture, 2019. https://doi.org/10.1016/j.sysarc.2019.01.017

[12] Pavel Celeda, Radek Krejci, and Vojtech Krmicek, *Revealing and analysing modem malware*, 2012 IEEE International Conference on Communications (ICC), Ottawa, ON, 2012, p. 971-975.

[13] Colin Tankard, *The security issues of the internet of things*, Computer Fraud&Security (9), 2015, p. 11-14.

[14] Raymond Canzanese. *Detection and Classification of Malicious Processes Using System Call Analysis.* Doctor of Philosophy, Drexel University, 2015. https://pdfs.semanticscholar.org/8060/eae74c98a66cfcc736f4fca61d46f4dbc1d4.pdf. [Accessed: 02-May-2019].

[15] Rieck, Konrad, Philipp Trinius, Carsten Willems, and Thorsten Holz aff2n3. *Automatic Analysis of Malware Behavior Using Machine Learning.* Journal of Computer Security (JCS), 2011, 19 (4), p.639–668.

[16] Pa Yin Min Pa, Shogo Suzuki, Katsunari Yoshioka, Tsutomu Matsumoto, Takahiro Kasama, and Christian Rossow. *IoTPOT: A Novel Honeypot for Revealing Current IoT Threats.* Journal of Information Processing, vol 24, no. 3 (2016), p.522-33. Doi:10.2197/ipsjjip.24.522.

[17] Shifu Hou, Aaron Saas, Lifei Chen, and Yanfang Ye. *Deep4MalDroid: A Deep Learning Framework for Android Malware Detection Based on Linux Kernel System Call Graphs.* 2016 IEEE/WIC/ACM International Conference on Web Intelligence Workshops (WIW). Doi:10.1109/wiw.2016.040

[18] Alejadro Martin, Rausl Lara-Cabrera, and David Camacho. *Android malware detection through hybrid features fusion and ensemble classifiers: the AndroPyTool framework and the OmniDroid dataset.* Information Fusion, 2018, p.128-142. Doi:10.1016/j.inffus.2018.12.006

[19] Cuckoo Sandbox - Automated Malware Analysis [Online]. Available *https://www.cuckoosandbox.org/* [Accessed: 02-May-2019].

[20] Detux [Online]. Available *https://github.com/detuxsandbox/detux* [Accessed: 02-May-2019].

[21] David Brash. *Recent Additions to the ARMv7-A Architecture.* In 2010 IEEE International Conference on Computer Design, 2010. Doi:10.1109/ICCD.2010.5647549.

[22] Huy Trung Nguyen, Quoc Dung Ngo, and Van Hoang Le. *IoT Botnet Detection Approach Based on PSI Graph and DGCNN Classifier.* In 2018 IEEE International Conference on Information Communication and Signal Processing (ICICSP), Singapore, 2018, p. 18-122. Doi:10.1109/ICICSP.2018.8549713.

[23] Virus Total [Online]. Available *http://virustotal.com* [Accessed: 02-May-2019].

[24] QEMU [Online]. Available *http://wiki.qemu.org* [Accessed: 02-May-2019].

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

[26] Carsten Willems, Thorsten Holz, and Felix Freiling. *Toward Automated Dynamic Malware Analysis Using CWSandbox.* IEEE Security Privacy 5, no. 2 (March 2007): p.32-39. Doi:10.1109/MSP.2007.45.

[27] Ulrich Bayer, Andreas Moser, Christopher Kruegel, and Engin Kirda. *Dynamic analysis of malicious code.* Journal in Computer Virology, 2006, vol. 2(no.1), p.67-77.

[28] *Suspected Mass Exploit Against Linksys E1000 / E1200 Routers*, Available at: https://isc.sans.edu/forums/diary/Suspected+Mass+Exploit+Against+Linksys+E1000+E1200+Routers/17621/. [Accessed: 02-May-2019]

[29] Ahmad Darki, Chun-Yu Chuang, Michalis Faloutsos, Zhiyun Qian and Heng Yin. *RARE: A Systematic Augmented Router Emulation for Malware Analysis.* In Passive and Active Measurement, edited by Robert Beverly, Georgios Smaragdakis, and Anja Feldmann, p.60-72. Lecture Notes in Computer Science. Springer International Publishing, 2018.

[30] Kai-Chi Chang, Raylin Tso, Min-Chun Tsai, *IoT sandbox: to analysis IoT malware Zollard*, International Conference on Internet of things and Cloud Computing, 2017, p.4-12.

[31] MIPS Wikipedia [Online]. Available *https://vi.wikipedia.org/wiki/MIPS.* [Accessed: 02-May-2019].

[32] Canzanese, Raymond, Spiros Mancoridis, and Moshe Kam. *Run-Time Classification of Malicious Processes Using System Call Analysis.* 10th International Conference on Malicious and Unwanted Software (MALWARE), 2015, p.21-28. Doi:10.1109/MALWARE.2015.7413681.

[33] Sanya Chaba, Rahul Kumar, Rohan Pant and Mayank Dave. *Malware Detection Approach for Android Systems Using System Call Logs.* ArXiv,1709.08805, 2017.

[34] K A Asmitha and P Vinod. *A Machine Learning Approach for Linux Malware Detection.* In 2014 International Conference on Issues and Challenges in Intelligent Computing Techniques (ICICT), 2014, p.825-830. Doi: 10.1109/ICICICT.2014.6781387.

[35] Virus share [Online]. Available *https://virusshare.com/* [Accessed: 02-May-2019].

[36] Strace tool [Online]. Available *http://sourceforge.net/projects/strace/* [Accessed: 02-May-2019].

[37] InetSim [Online]. Available *https://www.inetsim.org/* [Accessed: 02-May-2019].

[38] PyNetsim [Online]. Available *https://github.com/jjo-sec/pynetsim* [Accessed: 02-May-2019].

[39] J. Zaddach, L. Bruno, A. Francillon, and D. Balzarotti. *Avatar: A framework to support dynamic security analysis of embedded systems firmwares.* In Proceedings of the 2014 Network and Distributed System Security Symposium. The Internet Society, 2014, p. 23–26.

[40] Sikorski, Michael, and Andrew Honig. *Practical Malware Analysis: The Hands-On Guide to Dissecting Malicious Software.* 1st ed. San Francisco, CA, USA: No Starch Press, 2012.

[41] Yiming Yang, and Jan O. Pedersen. *A Comparative Study on Feature Selection in Text Categorization.* In Proceedings of the Fourteenth International Conference on Machine Learning, 412-420. ICML, San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1997.

[42] B. Chandra, Manish Gupta. *An efficient statistical feature selection approach for classification of gene expression data.* Journal of Biomedical Informatics. Volume 44, Issue 4, August 2011, p.529-535.

[43] Alireza Souri and Rahil Hosseini. *A State-of-the-Art Survey of Malware Detection Approaches Using Data Mining Techniques* Human-Centric Computing and Information Sciences 8, no. 1 (January 12, 2018): 3. Doi: 10.1186/s13673-018-0125-x.

[44] Shang, Fengjun, Yalin Li, Xiaolin Deng, and Dexiang He. *Android Malware Detection Method Based on Naive Bayes and Permission Correlation Algorithm.* Cluster Computing, June 17, 2017. Doi: 10.1007/s10586-017-0981-6.

[45] Ding Yuxin, Wei Dai, Shengli Yan, and Yumei Zhang. *Control Flow-Based Opcode Behavior Analysis for Malware Detection.* Computers & Security 44 (July 1, 2014): p.65-74. Doi: 10.1016/j.cose.2014.04.003.

[46] 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): p. 864-881. Doi: 10.1016/j.comnet.2008.11.011.

[47] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. An Introduction to Statistical Learning: with Applications in R. Springer, 1st edition. 2013, page: 181.

[48] Y. Bengio and Y. Grandvalet. *No unbiased estimator of the variance of k-fold cross-validation.* Journal of machine learning research 5 (Sep) (2004) p.1089– 1105