ISSN 1330-3651 (Print), ISSN 1848-6339 (Online) https://doi.org/10.17559/TV-20210202131602
Original scientific paper

Design of Automation Environment for Analyzing Various IoT Malware

Sungwon LEE, Hyeonkyu JEON, Gihyun PARK, Jonghee YOUN*

Abstract: With the increasing proliferation of IoT systems, the security of IoT systems has become very important to individuals and businesses. IoT malware has been increasing exponentially since the emergence of Mirai in 2016. Because the IoT system environment is diverse, IoT malware also has various environments. In the case of existing analysis systems, there is no environment for dynamic analysis by running IoT malware of various architectures. It is inefficient in terms of time and cost to build an environment that analyzes malware one by one for analysis. The purpose of this paper is to improve the problems and limitations of the existing analysis system and provide an environment to analyze a large amount of IoT malware. Using existing open source analysis tools suitable for various IoT malicious codes and QEMU, a virtualization software, the environment in which the actual malicious code will run is built, and the library or system call that is actually called is statically and dynamically analyzed. In the text, the analysis system is applied to the actual collected malicious code to check whether it is analyzed and derive statistics. Information on the architecture of malicious code, attack method, command used, and access path can be checked, and this information can be used as a basis for malicious code detection research or classification research. The advantages are described of the system designed compared to the most commonly used automated analysis tools and improvements to existing limitations.

Keywords: automated analysis; IoT malware; IoT malware classification; massive malware

1 INTRODUCTION

The Internet of Things (IoT) means states where humans, things, and services are connected with each other through embedded wireless communication systems, and it is often called Things Internet in South Korea. In the Internet of things, things refer to diverse embedded systems such as wearable devices, mobile devices, and electronic home appliances. The IoT was extended from Machine to Machine (M2M) in the past. Whereas M2M means only information exchanges between different devices, the IoT can be explained to be a term that collectively refers to all electronic devices with which M2M is possible, that is, all M2M devices connected through wireless networks. As can be seen from the term Internet of Things, the IoT can be said to be different from M2M because it has a strong tendency to actively introduce the "Internet" into the networks that connect devices in existing M2M. As ICTs are continuously developing and the fourth industrial revolution has recently come, overall IoT related fields such as IoT devices and technologies are also growing rapidly. The IoT technologies growing rapidly as such are generally penetrating into our lives too. Basically, the examples of those technologies include wearable devices, which are mounted on human bodies when they are used, and smart homes. Now, associated technologies have been developed further and are applied to various industries such as manufacturing, energy, construction, and agriculture. In addition, technologies such as big data and artificial intelligence, which are core technologies of the fourth industry, are grafted onto the IoT to become media for further development of IoT technologies. Not only the developing IoT-related technologies but also many IoT devices applied with those technologies have also been diversely and unceasingly released so that they can be easily seen in our everyday lives.

The IoT technologies and related devices that have been developed as such are being actively and widely used in industries beyond our daily lives and are becoming essential elements in the fourth industrial society. Therefore, many researchers and IoT device developers have been conducting many studies in relation to the security of IoT technologies and devices recently. After Mirai malware, which occurred in 2016 and attacked IoT devices around the world to cause damage, interest in the security has been continuously increasing. However, although the awareness and concern that IoT devices can pose serious threats to the corporate network have risen thanks to Mirai malware, IoT security remains at the conceptual level instead of existing in reality even now. There are many reasons for the foregoing including the fact that standards for security cannot be easily established because IoT devices are so massive and the fact that many unknown small and medium sized enterprises that produce IoT devices do not pay much attention to security so that those IoT devices can be connected to vulnerable networks. In fact, backdoor malware is sometimes found in the firmware inside those devices. Among many security threats of the IoT, the biggest problem is threats due to malware. After the emergence of Mirai malware, those pieces of malware that target IoT devices have been increasing exponentially. Not only new pieces of malware but also those pieces of IoT based malware that have become more powerful by complementing vulnerabilities in previous malware pose many difficulties to many analysts and researchers because previous analyses or studies were conducted focusing only on the behaviors of those pieces of malware that operate on the IoT such as botnet malware and DDoS malware.

This study is intended to propose methods to efficiently analyze those pieces of IoT related malware that appear in exponential quantities as such. Malware security can be effectively detected or secured after the malicious codes of the corresponding classification are analyzed first. It targets malicious codes that have already been collected in this study, and the target of attack, attack behavior, access path, and usage command can be identified. It utilized the relevant results to contribute to the IoT security industry with an ultimate goal of effectively responding to IoT malware.

Since the IoT system environment is diverse, IoT malware also has various environments. In the case of the existing analysis system, there is no environment for dynamic analysis by running IoT malware of various architectures. The analysis system designed in this paper supports various architectural environments to analyze IoT

malware, and automatically analyzes IoT malware through dynamic analysis after static analysis.

To this end, a new automated malware analysis framework was designed and implemented, and using the foregoing framework, an environment for analysis of massive Linux malware was constructed (refer to Chapter 4).

2 RELATED WORKS

Security research related to IoT malware has been variously conducted before [1, 2]. Representative ones are studies of Botnets and DDoS [3, 4]. The reason why the relevant behavioral analysis studies were actively conducted is the advent of Mirai malware in 2016, which had great effects on the IoT industry [5]. The situation where variant pieces of malware that emulated Mirai malware were fabricated and distributed contributed to the study trend. Various methods are emerging to study and detect IoT malware [6, 7]. However, studies focusing on the analysis of such malicious behavior to defend against the relevant behaviors clearly have limitations because if the malware evolves into massification, it is virtually impossible to analyze the malicious behaviors individually. Therefore, analysis of massive malware and automatic analysis of malware have already become essential elements in malware analysis studies. In the case of malware studies in the ICT field except for the IoT industry, automatic analyses of massive malware as such have already been conducted actively. [8] derives malware graph sets by automatically analyzing network behaviors such as the IP addresses, port numbers, protocols, and network dependent activities of malware. [9] collects and processes malicious behavior artifacts on file systems, memory, networks, and registries to label and analyze the features of malware. The relevant study can be an interesting study in that it derives 98% accuracy, despite its automatically analyzing massive malware exceeding 100 000 pieces based on unsupervised learning. [10] proposed an efficient environment configuration method for automatic analysis of malware based on dynamic analysis. Compared with static analysis, dynamic analysis has an advantage in that malicious behaviors can be identified more clearly. However, there is a problem in stability because malware should be firsthand executed and analyzed in the virtual environment called sandbox and the environment should be configured to fit the architecture or environment where malware operates. [10] is attempting to solve the problems of automatic dynamic analysis environments by proposing stable sandbox fabrication environments by citing and mixing machine learning, static analysis, and dynamic analysis that had been conducted previously.

[8, 9] and [10] propose efficient methods for automatic analysis of massive malware. In particular, [8] and [9] simultaneously classify malicious behaviors based on their characteristics. The family classification as such is very useful for detection of variant malware [11-13]. Variant pieces of malware are those pieces of malware that emulated the behaviors of existing malware and are one of malware pieces that are fabricated the most frequently in recent years [14]. Since variant pieces of malware conduct malicious behaviors that are very similar to those of the malware pieces they emulated, if data on the characteristics of existing malware pieces are possessed, the efficiency of analysis can be maximized and infringements can be easily responded. To that end, the analysis of massive malware is also important.

However, the above-mentioned studies have a limitation that they target Windows malware, that is, those pieces of malware that are based on PE files. Since most IoT devices are based on the Linux operating system following recent development of the IoT industry, the risk of those pieces of Linux malware that target those IoT devices for attacks is coming to the fore.

[15] proposes an automated analysis tool for Linux malware pieces. [15] fabricated the tool to enable the user to identify the results of various analyses on one report using Python scripts for many analysis tools used in Linux. It enables the analyses of various script files in addition to Elf file types and enables various analyses ranging from basic static analysis to dynamic analysis such as network analysis and memory analysis. However, some of the analysis tools used in [14] focus only on x86 elf. Therefore, they have limitations in light of the fact that IoT devices use embedded system-based architectures.

[16] also statistically and dynamically analyzed Linux malware pieces to compile the characteristics of the results. It analyzed not only X86-64 but also diverse CPU architectures such as MIPS and ARM through QEMU and compiled the statistics of the results of analyses of more than 10 500 pieces of malware thereby enabling users to identify the statistics of the malicious behaviors of Linux malware. The method in [16] has very similar directivity to that of the purpose pursued by the present study, but the present study has differences in that it focuses further on the features of malware pieces running on IoT devices beyond the statistics of Linux malware.

In our previous study, we proposed static and dynamic analysis methods to analyze IoT malware [17]. Based on this, this study confirms meaningful results and statistics through the analysis environment of various IoT malware.

3 IOT CLASSIFICATION DEFINITION OF CRITERIA

In this study, the criteria for IoT Classification Definition was proposed based on the data related to the IoT provided by the OWASP and the contents of analysis of actual IoT malware. All the IoT attack surfaces and vulnerabilities that can be identified with the data provided by the OWASP are related to network behaviors except for the aspect of hardware. IoT malware is analyzed in detail through the framework proposed in Chapter 5 and the judgment criteria for IoT malware determined in this Chapter 3 and the detailed content of analysis of IoT malware will be addressed in Chapter 5.

3.1 Attack Surface & Vulnerability

The Open Web Application Security Project (OWASP) provides various pieces of information through projects related to the IoT. The IoT projects are currently classified according to attack surfaces and vulnerabilities that may occur, and we analyzed IoT malware related characteristics based on the relevant research data. The OWASP IoT attack surfaces are classified into 18 items and the vulnerabilities that may occur in individual items

are described. The methods of approaches to the 18 attack surfaces can be largely divided into networking, hardware, and firmware. Network behaviors account for the most overwhelming part of the attack surfaces because ten out of 18 OWASP IoT attack surfaces include network behaviors. In fact, other attacks (Weak Passwords, Lack of encrypted, Backdoors, Internet Exposure, etc.) than the hardware approach include minimum network behaviors. Tab. 1 shows the list of OWASP IoT attack surfaces except for hardware and firmware and the characteristic that IoT related malicious behaviors of malware are based on network behaviors can be identified from Tab. 1.

Table 1 OWASP IoT Attack Surface Areas Related Networking

Attack Surface	Vulnerability
Device Web interface	Username enumeration, Weak passwords, etc.
Device Firmware	Vulnerability service (web, ssh, tftp, etc.)
Device Network Services	User CLI, Administrator CLI, Injection, etc.
Administrative Interface	Standard set of web application vulnerabilities
Cloud Web Interface	Username enumeration, Weak passwords, etc.
Update Mechanism	Malicious update, Update not signed, etc.
Mobile Application	Username enumeration, Weak passwords, etc.
Vendor Backend APIs	Injection attacks, Hidden services, etc.
Ecosystem Communication	Health checks, Pushing Update
Network Traffic	LAN to Internet, Wireless, Protocol Fuzzing
Authentication/ Authorization	Web application to cloud system authentication

3.2 Networking and Attack Surface based Classification

Since IoT malware is based on network behaviors, well known IoT malware (Mirai, hydra, aidra, bashlite, iotreaper, etc.) samples were analyzed to classify network behaviors in detail. In general, in the case of malicious behaviors conducted through web browsers, HTTP message related values were found and values related to other network behaviors such as Linux Network Commands, browser information, and socket errors were also found. In the results of the analysis, network behaviors were set forth in order of frequencies and accordingly, a network behavior list as shown in Tab. 2 was prepared.

Table 2 Networking Classification List

Networking Information
Web Access
Network Command
Network Information
Network Related Strings
Browser information
SSL

IoT equipment can be managed even without direct access because it is managed through network behaviors and user authentication and this is shown in the characteristics of malware too. IoT equipment is mainly controlled using a web interface or app rather than being directly controlled using hardware. These characteristics are connected to attacks using web vulnerabilities. In general, vulnerabilities that may occur in web/app interfaces include Injection, XXE, etc. Ways of bypassing IoT user authentication are largely divided into two types. In relation to user authentication, brute forcing and user enumeration are attack methods already well known and they are ill-used in actual IoT malware too. Tab. 3 below shows the classification criteria designed based on those vulnerabilities that may occur in the IoT.

Networking and Attack Surface are defined based on the content provided by OWASP IoT Project and the analysis of actual malware. Based on these criteria, IoT Malware is detected and classified.

Table 3 IoT Attack Surface Classification List

IoT Attack Surface
Injection
Xml eXternal Entity
Username enumeration
Brute Forcing
System Information
Busybox, Specific String

4 MASSIVE MALWARE ANALYSIS ENVIRONMENT

In this study, to analyze massive malware and check the results, a new automated malware analysis framework was designed and implemented, and using the framework, a massive Linux malware analysis environment was constructed. The malware analysis framework proposed in the present paper is an automated analysis tool that can use both static analysis methods and dynamic analysis methods in an integrated manner. Unlike the existing known tools that could obtain only the reports of analysis of single malware, this tool is characterized by the fact that it can analyze files in units of sets, process analysis result data in many stages, and summarize the results at the level of file sets such as the classification of the entire files by characteristic and the summation of numbers.

4.1 Analysis Environment

The overall structure of the analysis environment is as shown in Fig. 1. It consists of an analysis framework in which actual analysis tasks are managed, a front-end for user designation to determine detailed settings of the framework and task contents, a VM manager, agents, and analyzer scripts. The front-end is an execution file that refers to the framework, and is a component that specifies various environment settings for analysis and starts analysis tasks. It can be implemented in diverse forms as needed by the user, such as CLI and GUI applications or web applications.

The analysis framework is an assembly that includes a static analyzer that uses the existing analysis tools for Linux, a newly designed dynamic analyzer, and diverse task objects for handling and processing of data at various points before and after analysis tasks and was implemented as a DLL that operates on .NET Core Runtime. The dynamic analyzer, where malware is actually analyzed, conducts primary analysis using the tools basically provided by Linux and several open source based tools, and classifies the results of analysis into five categories; trace, state monitoring, network, process, and others. A list of the tools used internally can be found in Tab. 4.

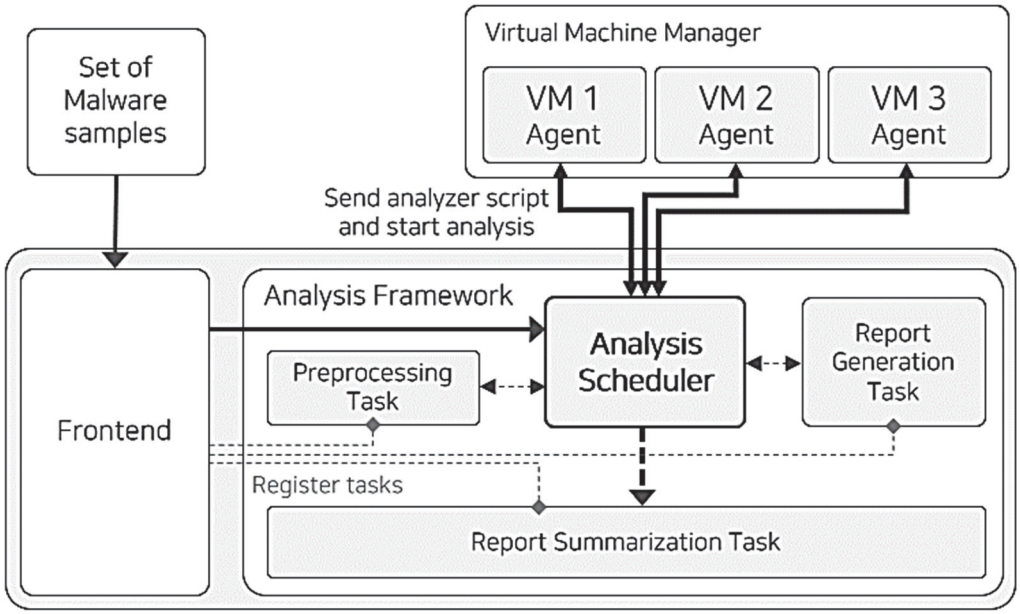

Figure 1 Structure of analysis environment

Table 4 List of tools used in analysis

Category	Tool
Trace	strace, ltrace
State monitoring	pstree, lsof, service, netstat, lastlog, procfs, etc.
Network	tcpdump, tshark
Memory	volatility
Static analysis	MinGW(ldd, file, readelf, strings), upx

4.2 Entire Analysis Pipeline

For analysis of massive malware, the entire analysis pipeline was constructed using the analysis environment mentioned in 4.1. The analysis pipeline contains various elements separately implemented for high-level data analysis for our analysis tasks, which consist of sets of report generation tasks and report summarization tasks and a set of Yara rules newly prepared for classification of Linux malware. Our analysis pipeline is as shown in Fig. 2.

The submitted malware samples underwent a static analysis stage, dynamic analysis stage, and analysis result interpretation stage and the malware was finally classified thereafter. In the static analysis stage, static analysis was conducted using the static analysis tools mentioned in Tab. 4 of 4.1 to recognize file types, identify ELF headers, check whether packed or not, and extract character strings.

When dynamic analysis had been judged possible through the static analysis, dynamic analysis was attempted using the prepared VM. In the dynamic analysis, changes in the process were identified using pstree and service commands, and the lost of loaded libraries was extracted using the lsof command. Network behaviors were identified using netstat and tcpdump. In addition, additional information such as the records of the shell commands, login records, and mount information was identified.

In the result interpretation stage, the information extracted in the static analysis and dynamic analysis stages was used to synthetically judge through which paths the malware actually approaches, what kind of network behaviors it conducted, what kind of vulnerabilities it utilizes, and which attack surfaces it aims at. Yara is utilized to check whether the relevant malware coincides with the predefined rules. However, among the Yara rule data disclosed in the open source repository as results of previous studies, an extremely small number of rule data for malware or malicious behaviors targeting Linux or Unix-like systems existed and even the existing rule data were not for detection of certain behaviors but were for identification of fixed character strings indicating malware in most cases.

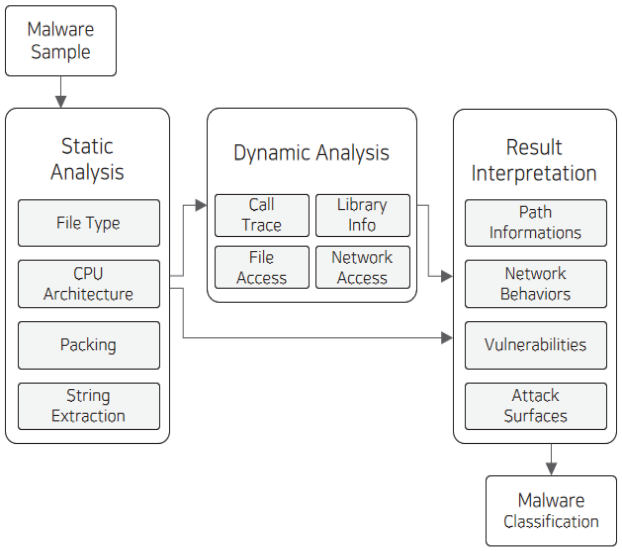

Figure 2 Entire analysis pipeline

Therefore, there were many problems in applying those rule data to the result interpretation stage. In addition, in the case of those pieces of malware that were targeting Linux, there were difficulties in obtaining the information necessary for malware classification because previous studies were insufficient compared to Windows malware. Consequently, the task to construct the entire analysis pipeline was carried out through repetitive trials and errors, and the framework was gradually improved so that the tasks to reprocess and reinterpret the generated reports could be carried out independently.

4.3 Comparison with Existing Analysis System

Compatibility, function, and structure were compared with existing open source Linux malware automation analysis tools. Cuckoo Sandbox, HaboMalHunter, and Limon Sandbox were selected for comparison. Cuckoo Sandbox is a widely known open source sandbox tool, and HaboMalHunter is a Linux malware analysis tool developed by Tencent. Limon Sandbox is a sandbox tool for Linux malware analysis introduced at Black Hat 2015.

Tab. 5 compares the compatibility of the above-mentioned comparison target tools for the three categories of host, guest, and virtualization software.

The host of this analysis environment was implemented cross-platform through the .NET core platform and MinGW. There are no separate restrictions on guests, but the results from this paper were limitedly tested on Ubuntu 10.04, 14.04, 18.04 and Debian 6. Virtualization software basically supports VirtualBox, VMware, and QEMU, and other virtualization software can be used by implementing adapter interfaces according to user needs. Compared to other tools, this analysis environment focuses on not limiting the host and guest analysis environments. For user convenience, the .NET Core was used to operate without difficulty in various host environments. Since the IoT product has multiple execution environments, the guest environment is also designed to be universal because there is no limit to analyze various malicious codes for the target. Virtualization software can also be selected by the user, and QEMU is used in this paper.

Table 5 Comparison of compatibility with existing analysis environment

	Designed environment	Cuckoo Sandbox	HaboMal Hunter	Limon Sandbox
Host	Windows, Linux, Mac OS X	Windows, Linux, Mac OS X	-	Linux
Guest	No limit	OS: Ubuntu 18.04	OS: Ubuntu 14.04 Kernel: No limit	No limit
VM	VirtualBox, VMware, QEMU	VirtualBox, VMware, XenServer, QEMU/KVM	-	VMware

Cuckoo Sandbox supports tracking in kernel mode, and uses systemtap to compile and use kernel modules that perform system call tracking functions. This analysis environment, HaboMalHunter, and Limon Sandbox all support tracing in user mode only, and use strace and ltrace. In system status monitoring, this analysis environment is mainly used for basic commands of Unix and Linux, whereas Cuckoo Sandbox uses the Volatility Framework. The system state is obtained from the contents of the memory dump, and HaboMalHunter and Limon Sandbox use the sysdig tool. For network analysis, pcap files were created through tcpdump for all the mentioned tools, and pcap was parsed through tshark, dpkt, or a script written by hand. HaboMalHunter and Limon Sandbox use inetsim to simulate and respond to malicious code requests for inactive servers as normal traffic. All of the memory analysis used the Volatility Framework, and most of the static analysis tools used similar ones.

5 IoT MALWARE ANALYSIS

The analysis automation framework designed in this paper performs static and dynamic analysis through several steps and tools, outputs various information, and verifies statistics and meaningful data for the entire as well as individual processing of the output information. This chapter classifies and analyzes the files suspected of being IoT malicious code based on the criteria determined by using the analysis automation framework designed earlier, OWASP, and existing real IoT malicious code for collected malicious code files. Discuss the efficiency, limitations, and results of the analysis system through analysis contents and statistics.

5.1 General Purpose IoT System Range

In 2018, an "IoT developer survey" was conducted with IoT developers and the results indicated that at least about 71.8% of the IoT developers were using Linux operating systems. In addition, other operating systems than Windows operating systems were also based on the Linux kernel to the extent that it can be safely judged that Linux operating systems were being used in most IoT devices [13].

Malware files analyzed by the analysis automation framework are analyzed mainly for executable files that have a Linux-based ELF format. The meaning of the suspicious IoT malicious code file determined in this study refers to the malicious code that can operate in the IoT system. Therefore, even if the original intention among the Linux-based ELF-type executable files is not the target of the IoT system, if the condition is satisfied, it is judged to operate in the IoT system and classified as an IoT malicious code.

5.2 IoT Malware Detection & Classification

Existing IoT malware files and randomly collected Linux-based files were applied to the analysis framework, and information on the target Linux-based ELF-type executable files could be verified. Tab. 6 shows the contents of the architecture classification of the classified ELF format executable files. 14 976 malicious codes could be classified into 12 architectures (even if they were different as little endian or big endian, they were classified as the same architecture), and 3 unknown architectures that were not classified were identified. Various types of files, such as text files, source codes, scripts, libraries, documents, and compressed files, were also identified, but excluded, not subject to analysis.

The ideal situation for the goal is to analyze all 14 976 malware with 12 architectures identified through the designed analysis automation system, but the malware actually analyzed 13 897 files with 5 architectures Intel 80386, AMD64, ARM, MIPS, PowerPC By analysis, 92.80% of malicious code files were analyzed.

Because the dynamic analysis environment for the remaining 7 architectures was not established, it was excluded from the analysis. It is also possible to analyze IoT malicious codes by constructing an execution environment for other minor IoT architectures identified in the future direction of further research.

Subsequent analyses analyzed 13897 files of 5 architectures of Intel 80386, AMD64, ARM, MIPS, and PowerPC in the designed analysis automation framework, and the results of static analysis and dynamic analysis were processed through the analysis pipeline.

Table 6 Classification of the architectures of collected samples

	Samples	Percentage
Intel 80386	5112	34.14
AMD 64	726	4.85
ARM	4367	29.16
AArch64	29	0.2
MIPS	2786	18.60
SuperH	49	0.33
PowerPC	906	6.05
Sparc	912	6.09
MC68000	33	0.22
ARCompact	3	-
Tilera TILE-Gx	49	0.32
Intel MCU	1	-
<unknown>	3	-
Total	**14976**	**100**

During dynamic analysis in the virtual machine, check the analysis result of the strace tool to check the contents of abnormal termination or error occurrence of malicious code. Tab. 7 summarizes the causes of abnormal termination or errors checked through strace.

The most identified error is an ENOENT (No such file or directory) error, which can be checked when the file to be executed does not exist or the required library does not exist. The version of the dynamically linked library is different or absent, and there is a problem with the dependency. If you improve it in the future, you will get better results.

Table 7 The cause of abnormal termination as determined by the strace tool

Error Description	Count	Error Description	Count
No such file or directory	5985	Function not implemented	244
Not a typewriter	2561	Try again	217
Segmentation fault	1454	Interrupted system call should be restarted	135
Illegal seek	748	Bad address	131
Bad file number	372	Invalid argument	114

The number of those pieces of malware that were judged to conduct network related behaviors was estimated to be 10 313 in total, which corresponds to 74.21% of the collected malware samples. That is, this means that 74.21% of the collected malware samples satisfied the minimum condition for application to IoT systems.

Table 8 Attack Surface based statistics

	CI	SQL	XXE
Brute Forcing	114	12	1
Using Busybox	6504	3	175
UsernameEnumeration	2870	1	2
System Info	6608	21	44
Specific Strings	2,420	-	21
Flooding	145	1	1
Download the File	8641	3	139
Change File Authority	1252	1	181
Server Remote Access	8173	19	90
Process Kill	2071	1	1

Overlapping number counting was allowed for the detailed classification items in Tab. 8. The percentages were prepared based on the entire number of the malware samples, which is 13 897. The IoT vulnerabilities of the 10313 samples that satisfied the minimum conditions were classified using the Attack Surface Classification explained in Chapter 3. The IoT vulnerabilities can be classified into three types; Command Injection, SQL injection and XXE and the values of the classified IoT malware can be identified in Tab. 8.

Tab. 9 shows the files judged to be CPU Architecture. "percentage" is the percentage filtered by IoT malware in each architecture sample.

Table 9 Network behavior statistics for IoT malware

	samples	percentage
Network Info	9183	89.04
Network Strings	7666	74.33
Network Command	9961	96.59
Browser Info	6355	61.62
Web Access	6114	59.28
SSL	93	0.90

5.3 Detailed Analysis of IoT Malware

Through the analysis of massive IoT malware, common characteristics and diverse meaningful data could be extracted. The 10 313 of IoT malware identified through the IoT malware detection classification set forth in Section 5.2 become new standards for the detailed analysis set forth in this Section. The frequencies of discovery were calculated by the command used in IoT malware or characteristic for access to files etc. and the samples were classified according to network related commands, and access to paths such as /bin, /proc, /sys and /etc.

Commands were used in almost all of the 10 313 IoT malware samples and among the commands, the top 10 commands with the highest frequencies of discovery are as shown in Tab. 10.

Table 10 Important shell commands TOP10

	samples	percentage
pkill	1477	14.32
chmod	952	9.23
killall	607	5.89
kill	591	5.73
ps	167	1.62
sh	159	1.54
cp	155	1.50
grep	104	1.01
cat	37	0.36
rm	37	0.36

Whether network related commands were used in IoT malware or not was analyzed. Tab. 11 shows the numbers of IoT malware samples in which certain network commands were used.

Table 11 Network commands TOP10

	samples	percentage
wget	8599	83.38
tftp	7428	72.03
telnet	6496	62.99
ftpget	6038	58.55
ssh	3055	29.62
dropbear	2999	29.08
netstat	1827	17.72
sshd	1737	16.84
iptable	1701	16.49
apt-get	1621	15.72

In Linux systems, the directory /proc is a virtual file system with no physical capacity. It enables identifying information on hardware such as CPU, RAM, and partitions and information of the process currently being executed. Among the IoT malware samples classified by us, 9106 samples, which were about 88.29%, were shown to access /proc and its sub-paths. Tab. 12 shows the frequencies of discovery from representative /proc sub-paths.

Table 12 Access to the /proc path

	samples	percentage
/proc/net/route	8565	83.05
/proc/cpuinfo	6383	61.89
/proc/stat	1804	17.49
/proc/self/exe	408	3.94
/proc/sys/kernel/osrelease	316	3.06
/proc/meminfo	266	2.58
/proc/self/maps	259	2.51
/proc/net/dev	255	2.47
/proc/sys/kernel/rtsig-max	253	2.45
/proc	253	2.45

The directory /sys is a point where sysfs virtual file systems are mounted and is a space where kernels record hardware information. The number of those IoT malware samples that accessed the /sys directory was 1651 out of the total number 10 313, corresponding to about 16.01%. The frequencies of discovery of individual detailed paths are set forth in Tab. 13.

Table 13 Access to /sys paths

	samples	percentage
/sys/devices/system/cpu	1527	14.81
/sys/devices/system/cpu/online	91	0.88
/sys/class/block	6	-
/sys/class	6	-
/sys/block	6	-
/sys/devices/system/cpu/cpu/cpufreq/scaling_cur_freq	3	-
/sys/bus/usb/devices	3	-
/sys/class/net	2	-
/sys/class/tty/console/active	2	-
/sys/power/state	2	-

The path /ETC is a directory where the overall system management and setting files exist. Diverse files exist here such as mount information files, user account information files, and executable program setting files. The number of those IoT malware samples that accessed the /etc directory was 7971 out of the total number 10 066, corresponding to about 77.29%. The frequencies of discovery of individual detailed paths are set forth in Tab. 14.

Table 14 Access to /etc paths

	samples	percentage
/etc/resolv.conf	7414	71.89
/etc/hosts	7394	71.70
/etc/config/resolv.conf	6075	58.91
/etc/config/hosts	6075	58.91
/etc/rc.conf	2887	27.94
/etc/rc.d/rc.local	2866	27.79
/etc/localtime	383	3.71
/etc/ld.so.cache	355	3.44
/etc/suid-debug	352	3.41
/etc/fstab	307	2.98

Among the IoT malware samples, those that accessed the user home directory and paths /usr/share and /var were inquired and such operations were found in 7120 out of the entire 10313 IoT malware samples. Tab. 15 below shows the frequencies of discovery of access to certain paths.

Table 15 Access to other paths

	samples	percentage
/tmp/	4372	43.43
/var/run/	4222	41.94
/var/	4199	41.71
/var/tmp/*	641	6.37
/var/run/*	641	6.37
/var/*	641	6.37
/tmp/*	641	6.37
/usr/share/locale	379	3.77
/usr/share/zoneinfo	365	3.63
/var/tmp	356	3.54

In addition to being mentioned in this paper, various commands, access paths, and other information can be found. We identified the traces of the interpreter languages Python and the perl language, and 3144 Python and 1686 Perl samples. A total of 152 556 IP addresses used as C&C servers or target servers contain an average of 15 IP addresses per sample.

Based on these data, meaningful data can be extracted or used as an important resource for the next goal.

5.4 Result

The collected malicious code files were analyzed to check the function and performance of the designed IoT malicious code analysis automation framework. In the analysis process, the limitations of not being able to analyze various architectures that could not be built in the design process, and the limitation of not showing the expected efficiency due to various reasons in the process of classifying IoT malicious code suspicious files were confirmed.

It is not appropriate to compare the performance of analysis and results using the analysis framework with other analysis systems with a simple advantage. However, when compared with the goal of providing various analysis environments, which was the biggest design goal when building the analysis system in this paper, and obtaining and processing various data, overall compared to other analysis systems, although there are some regrets in the analysis environment. It shows good performance.

In fact, 13 897 files were analyzed for 14 976 Linux-based malicious code executable files, which amounted to 92.80%. For dynamic analysis, 67% of malicious codes were unable to perform all of the original malicious actions, but in the process, data that can be obtained only by dynamic analysis and the cause of errors can be collected.

Padawan analysis system, which supports static and dynamic analysis for various architectures, takes a total of 500 days to analyze 14 976 malicious codes with a limit of 30 analyses per day. In the case of the time taken to analyze a single malicious code, the time varies depending on the software used by each system or the degree of analysis, but Padawan supports analysis through a web server rather than an open source code, so the aspect of mass analysis Also, it is not efficient considering the time to transmit the malicious code. In addition, since the code is not an open system, it is a system that can check and receive only specified data, so if the user does not have the desired side

data, a well-established static analysis and dynamic analysis environment is meaningless or a separate work process must be performed directly.

In the case of Cuckoo Sandbox, which is the most famous and used, it is explained that it can be analyzed for multiple systems, but the weight of the actual analysis system is largely skewed toward Windows malware. If the goal is to perform automatic analysis of Windows malware, Cuckoo sandbox is a very good system, but it has little analysis function for Linux-based malware, and because the user directly prepares and builds the virtual environment image of the sandbox, it is virtually user-friendly. Should create a function.

Habomalhunter said that the structure of the analysis system is not suitable for mass analysis, and there is no news of an update on the development while the completeness is poor. A lot of errors are generated in the process of performing basic analysis, and users must correct the whole in order to use this analysis system.

Limon Sandbox is easy to use and faithful to basic analysis functions, so it is easy for users to use and convenient to improve according to the purpose of use. But basically, only 39% of 5838 individuals can analyze 14 976 malicious codes because only two architectures of Intel 80386 and AMD64 can be analyzed using VMware virtual machines.

In order to improve the limitations and limitations of analyzing a large amount of IoT malicious codes in existing analysis automation systems, an analysis automation framework was designed and built. As a result of analyzing actual malicious codes, it provides various analysis environments and obtains and processes various data.

6 CONCLUSION

Currently, IoT malware continues to grow and various variants are also being identified. Several defense rules for detecting existing IoT malware are difficult to detect for variants, new types, encryption, and packed malware. This paper presented a new method for classifying malware that operates on IoT systems, and an analysis method that combines static analysis and dynamic analysis for unclassified malware. The researchers were able to construct a mass analysis environment through classification criteria and analysis method. Also, the IoT malware in Linux was classified based malware. IoT malware could be analyzed afterwards.

The new criteria presented in this paper are not based on existing IoT malware but are targeted at "malware that runs on IoT" and analyze general purpose IoT malware. IoT malware will continue to evolve and new types and variants will be introduced, with various attack techniques applied. Currently, several studies are under way for automated analysis of IoT malware, but there is no fully automated analysis tool for analyzing IoT malware with various targets, purposes, and characteristics, and it is not suitable for mass analysis. Through the randomly collected malicious codes, it was found that the problems of the proposed system and the existing analysis systems were compared and improved by comparing the limitations. However, in the design analysis results in this paper, there are limitations identified in the analysis process. In addition to the built-in architectural virtual machine, a few architectural malicious codes were identified, and malicious codes that performed dynamic analysis were often unable to complete malicious actions during the analysis process and ended in the middle. It is expected that performance can be further improved by supporting and constructing various architectures or the dependency of libraries generated in the dynamic analysis process in the future developed research direction.

Acknowledgements

This research was supported by the 2018 Yeungnam University Research Grant (218A061016, 218A380138) and the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. 2018R1D1A1B07050647).

7 REFERENCES

[1] Kim, J. T. (2015). Requirement of security for IoT application based on gateway system. *International journal of security and its applications, 9*(10), 201-208. https://doi.org/10.14257/ijsia.2015.9.10.18

[2] Patil, A., Bansod, G., & Pisharoty, N. (2015). Hybrid lightweight and robust encryption design for security in IoT. *International Journal of Security and Its Applications, 9*(12), 85-98. https://doi.org/10.14257/ijsia.2015.9.12.10

[3] Kolias, C., Kambourakis, G., Stavrou, A., & Voas, J. (2017). DDoS in the IoT: Mirai and other botnets. *Computer, 50*(7), 80-84. https://doi.org/10.1109/MC.2017.201

[4] Bertino, E. & Islam, N. (2017). Botnets and internet of things security. *Computer, 50*(2), 76-79. https://doi.org/10.1109/MC.2017.62

[5] Antonakakis, M., April, T., Bailey, M., Bernhard, M., Bursztein, E., Cochran, J., Zhou, Y., et al. (2017). Understanding the mirai botnet. *26th {USENIX} security symposium ({USENIX} Security, 17*, 1093-1110.

[6] Yang, J. H. & Ryu, Y. (2015). Design and development of a command-line tool for portable executable file analysis and malware detection in IoT devices. *International Journal of Security and Its Applications, 9*(8), 127-136. https://doi.org/10.14257/ijsia.2015.9.8.10

[7] Ghapar, A. A., Yussof, S., & Bakar, A. A. (2018). Internet of Things (IoT) architecture for flood data management. *International journal of future generation communication and networking, 11*(1), 55-62. https://doi.org/10.14257/ijfgcn.2018.11.1.06

[8] Nari, S. & Ghorbani, A. A. (2013). Automated malware classification based on network behavior. *International Conference on Computing, Networking and Communications (ICNC)*, 642-647. https://doi.org/10.1109/ICCNC.2013.6504162

[9] Mohaisen, A., Alrawi, O., & Mohaisen, M. (2015). AMAL: high-fidelity, behavior-based automated malware analysis and classification. *Computers & security, 52*, 251-266. https://doi.org/10.1016/j.cose.2015.04.001

[10] Kruegel, C. (2014). Full system emulation: Achieving successful automated dynamic analysis of evasive malware. *Proceedings of Black Hat USA Security Conference*, 1-7.

[11] Rieck, K., Holz, T., Willems, C., Düssel, P., & Laskov, P. 2008. Learning and classification of malware behavior. *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, 108-125. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-70542-0_6

[12] Kinable, J. & Kostakis, O. (2011). Malware classification based on call graph clustering. *Journal in computer virology, 7*(4), 233-245. https://doi.org/10.1007/s11416-011-0151-y
[13] Ahmadi, M., Ulyanov, D., Semenov, S., Trofimov, M., & Giacinto, G. (2016). Novel feature extraction, selection and fusion for effective malware family classification. *Proceedings of the sixth ACM conference on data and application security and privacy*, 183-194. https://doi.org/10.1145/2857705.2857713
[14] Zhang, Q. & Reeves, D. S. (2007). Metaaware: Identifying metamorphic malware. Twenty-Third Annual Computer Security Applications Conference (ACSAC 2007), 411-420. https://doi.org/10.1109/ACSAC.2007.9
[15] Monnappa, K. (2015). Automating linux malware analysis using limon sandbox. *Black Hat Europe*, 2015.
[16] Cozzi, E., Graziano, M., Fratantonio, Y., & Balzarotti, D. 2018. Understanding linux malware. *IEEE symposium on security and privacy (SP),* 161-175. https://doi.org/10.1109/SP.2018.00054
[17] Sungwon, L., Hyeon, K. J., Gi, H. P., Ji, H. K., & Jonghee M. Y. (2021). IoT malware static and dynamic analysis system. *Journal of Human-centric Science and Technology Innovation, 1*(1).

Contact information:

Sungwon LEE, M.S.
Dept. Of Computer Engineering, Yeungnam University,
280 Daehak-Ro, Gyeongsan, Gyeongbuk, Republic of Korea
E-mail: noke15@ynu.ac.kr

Hyeonkyu JEON, B.S.
Dept. Of Computer Engineering, Yeungnam University,
280 Daehak-Ro, Gyeongsan, Gyeongbuk, Republic of Korea
E-mail: in2etv@in2e.tv

Gihyun PARK, B.S.
Dept. Of Computer Engineering, Yeungnam University,
280 Daehak-Ro, Gyeongsan, Gyeongbuk, Republic of Korea
E-mail: shwo8713@gmail.com

Jonghee YOUN, PhD, Professor
(Corresponding author)
Dept. Of Computer Engineering, Yeungnam University,
280 Daehak-Ro, Gyeongsan, Gyeongbuk, Republic of Korea
E-mail: youn@yu.ac.kr