Journal of Human-centric Science and Technology Innovation
Vol.1, No.1 (2021), pp.43-48
http://dx.doi.org/10.21742/jhsti.2021.1.1.01

IoT Malware Static and Dynamic Analysis System

Sungwon Lee[1], HyeonKyu Jeon[2], GiHyun Park[3], JiHun Kim[4], Jonghee M. Youn[5]

[1,5]Dept. Of Computer Engineering, Yeungnam Univ., 280 Daehak-Ro, Gyeongsan, Gyeongbuk, Republic of Korea
[2]Korea Appraisal Board, 291, Innovalley-ro, Dong-gu, Daegu, Republic of Korea
[3]Korea Institute of Arboretum Management, 10, Jeongbu 2cheongsa-ro, Sejong-si, Republic of Korea
[4]Daegu Gyeongbuk Institute of Science and Technology, 333, Techno jungang-daero, Hyeonpung-eup, Dalseong-gun, Daegu, Republic of Korea
[1]noke15@ynu.ac.kr, [2]in2etv@in2e.tv, [3]shwo8713@gmail.com, [4]f13521@naver.com, [5]youn@yu.ac.kr

Abstract

As the spread of IoT systems increases, security of IoT systems has become very important for individuals and companies. IoT malware has been increasing exponentially since the emergence of Mirai in 2016. 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. It is inefficient in time and cost to construct an environment one by one to analyze numerous malicious codes and proceed with analysis. There are so many IoT malware to be analyzed that an efficient method of analysis is required. The goal of this paper is to improve the problems and limitations of the existing analysis system and provide a variety of analysis environments. In this paper, we build a system that automatically analyzes basic IoT malware. It provides an analysis environment by constructing a static analysis and dynamic analysis system suitable for various IoT malware. In the text, the analysis system is applied to the actual collected malicious code to check whether it is analyzed and to derive statistics. It describes the advantages of the designed system and the improvement of existing limitations through comparison with the most commonly used automation analysis tools.

Keywords: *Automated analysis, IoT malware classification, IoT malware analysis*

1. Introduction

After Mirai malware [1], which occurred in 2016 and attacked IoT devices around the world, interest in the IoT security has been continuously increasing. Since the emergence of Mirai malware, malicious codes and variants targeting IoT devices have increased [2][3].

The present study is intended to propose methods to efficiently analyze those pieces of IoT related malware that appear in exponential quantities as such and utilize the relevant results to contribute to the IoT security industry with an ultimate goal of effectively responding to IoT malware.

Article history:
Received (September 29, 2020), Review Result (October 30, 2020), Accepted (December 12, 2020)

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.

The goal of this paper is to improve the problems and limitations of the existing analysis system and provide a variety of analysis environments.

2. Related works

Since most IoT devices are based on the Linux operating system following recent development of the IoT industry [4], the risk of those pieces of Linux malware that target those IoT devices for attacks is coming to the fore. In order to respond to security threats, research on various IoT malware analysis systems for efficient analysis is ongoing [5][6].

[7] proposes an automated analysis tool for Linux malware pieces. [7] 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 focus only on x86 elf. Therefore, they have limitations in light of the fact that IoT devices use embedded system-based architectures.

[8] 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. [8] is very interesting since there studies related to the statistics of analyses of massive Linux malware and the characteristics were not actively conducted in the past. The method in [8] 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.

3. IoT malaria analysis system

In this study, to analyze IoT malware and check the results, a new automated malware analysis system was designed and implemented, and using the system, a massive Linux malware analysis environment was constructed. The malware analysis system 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.

The overall structure of the analysis environment is as shown in [Figure 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.

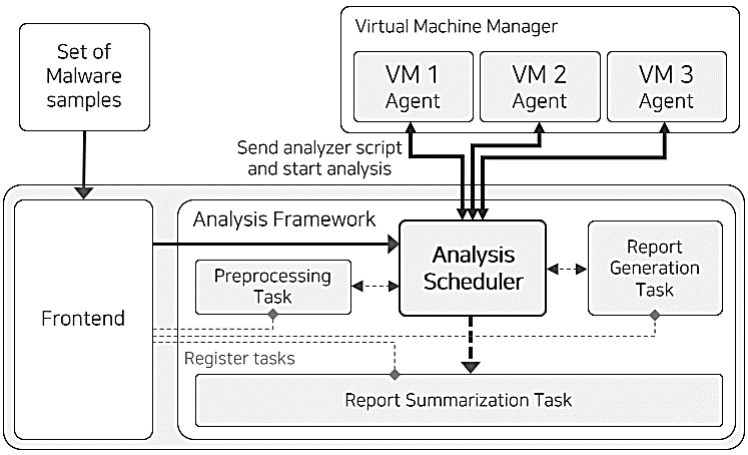

Figure 1 Structure of analysis environment

To support dynamic analysis of systems that use i386, amd64, and diverse other architectures, the VM manager and VM agent were implemented based on QEMU, which is virtualization software with the full-system emulation function. The VM manager and VM agent support parallel execution by the task scheduler of the analysis framework for smooth progression of analyses of a larger number of file sets and can analyze file sets consisting of multiple architectures by designating tag strings to individual VM profiles.

In the local system of the VM exist an agent and analyzer scripts as components that receive commands from the sandbox host and actually carry out analyses. In the VM, the agent receives commands from the sandbox host, downloads the actual analysis information and the latest analyzer script, and starts the analysis. It also uses the SMB protocol to transmit analysis result files and inform the current state of the VM to the host. Since the analyzer scripts are newly downloaded every time through the agent, analysis codes can be flexibly revised even without changing the VM snapshot and detailed analysis options including the trace mode divided into strace and ltrace are set through the parameters received through the agent.

The analysis system 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 [Table 1].

Table 1. 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

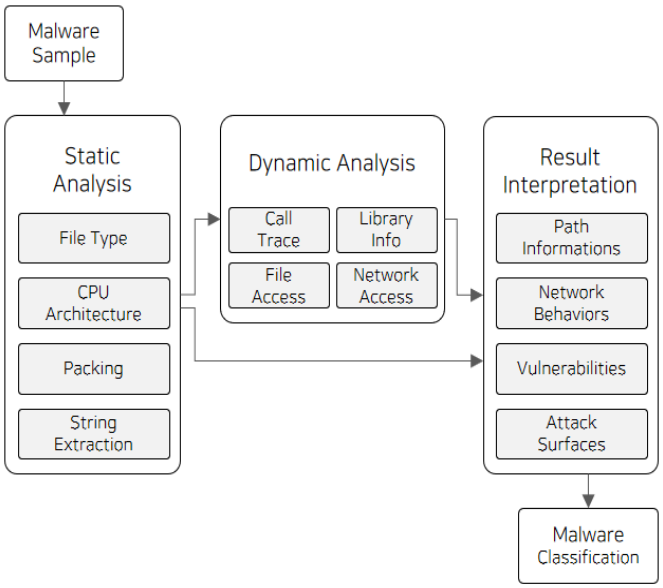

Figure 2 Entire analysis pipeline

The entire analysis pipeline was built using the analysis environment. 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 [Figure 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 Table1 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 was 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. 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. Results and discussion

The number of data samples analyzed for IoT malware is 3,098 in total and mainly those files that have Linux-based ELF formats were collected. Among the 3,098 Linux-based ELF format files, those files that satisfy the conditions are classified into IoT malware judging that they are operable in IoT systems even if they were not originally intended for IoT systems.

Table 2 Classification of the architectures of collected samples

	samples	Percentage
Intel 80386	1,784	57.59
AMD 64	181	5.84
ARM	737	23.79
AArch64	29	0.94
MIPS	148	4.78
SuperH	41	1.32
PowerPC	46	1.49
Sparc	58	1.87
MC68000	21	0.68
ARCompact	1	0.03
Tilera TILE-Gx	49	1.58
Intel MCU	1	0.03
Unknown	2	0.06
total	3,098	100

As can be seen in Table 2, the 3,098 samples can be classified into 12 architectures except for two samples with unknown architectures that cannot be classified. There are 332 (10.71%) packed files and 18 (0.58%) files for VM detection among the collected samples. When analyzing the files in the analysis environment, those packed files that cannot be easily analyzed statically can be analyzed dynamically and those VM detection files that cannot be easily analyzed dynamically can be analyzed statically. The ideal situation aimed at is to analyze all 3,098 malwares with 12 architectures identified through the designed analysis automation system, but the malware actually analyzed is 2,896 files with 5 architectures of Intel 80386, AMD64, ARM, MIPS, and PowerPC. By analysis, 93.48% of malicious code files were analyzed.

Network behavior classification and vulnerability classification, which are part of the analysis results, are summarized in [Tables 4] and [Table 5]. The percentages were prepared based on the entire number of the malware samples, which is 3,098.

Table 3. Network based statistics

Networking Information	Samples	Percentage
Browser Info	154	4.97%
Network Command	1087	35.09%
Network Info	738	23.82%
Network Strings	631	20.37%
SSL	109	3.51%
Web Access	602	19.43%

Improvements are confirmed through comparison with other systems mentioned in related works. Limon Sandbox [7] is easy to use and well implemented in basic analysis functions. But basically, only 63.42% of 1,965 individuals can analyze 3,098 malwares because only two architectures of Intel 80386 and AMD64 can be analyzed using VMware virtual machines.

Padawan analysis system [8] is limited to 30 analysis per day, and it takes a total of 103 days to analyze 3,098 malwares. In addition, since it is not an open code system, but a system that can only check and receive specified data, if the user does not have the desired data, the analysis environment is meaningless or the analysis must be performed directly.

Table 4 Vulnerability based statistics

Attack Surface	Samples	Percentage
Command Injection	693	22.37%
SQL Injection	73	2.36%
XSS	4	0.13%
XXE	24	0.77%

5. Conclusion

In this paper, we built a system to analyze IoT malware. It was designed with the aim of reducing analysis time by providing basic analysis and providing various analysis environments. To analyze IoT malware, it goes through static and dynamic analysis steps, and uses existing open source tools.

Existing automated analysis systems have improved limitations and limitations in analyzing IoT malicious codes, and as a result of analyzing actual malicious codes, it was possible to provide various analysis environments and obtain and process various data.

Acknowledgements

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

References

[1] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, and D. Kumar, “Understanding the mirai botnet,” security symposium, USENIX, pp.1093-1110, **(2017)**

[2] M. Kuzin, Y. Shmelev, and V. Kuskov, “New trends in the world of IoT threats,” Kaspersky, Sep **(2018)**

[3] E. Bertino, and N. Islam, “Botnets and internet of things security,” Computer, pp.76-79, **(2017)**

[4] C. Kolias, G. Kambourakis, A. Stavrou, and J. Voas, “DDOS in the IoT: Mirai and other botnets,” Computer, vol.50, pp.80-84, **(2017)**

[5] Q.D. Ngo, H.T. Nguyen, L.C. Nguyen, and D.H. Nguyen, “A survey of IoT malware and detection methods based on static features,” ICT Express, **(2020)**

[6] A. Costin, and J. Zaddach, “IoT malware: Comprehensive survey, analysis framework and case studies,” BlackHat USA Security Conference, **(2018)**

[7] K. Monnappa, “Automating Linux malware analysis using limon sandbox,” Black Hat Europe, **(2015)**

[8] Cozzi, and Emanuele, et al. “Understanding Linux malware,” 2018 IEEE Symposium on Security and Privacy (SP), IEEE, **(2018)**