Journal of Smart Technology Applications
Vol.2, No.1 (2021), pp.1-8
http://dx.doi.org/10.21742/jsta.2021.2.1.01

Expression of Malware Characteristics using API Sequence

Jihun Kim[1], SungWon Lee[2] and JongHee Youn[3]

[1]Daegu Gyeongbuk Institute of Science and Technology, 333, Techno jungang-daero, Hyeonpung-eup, Dalseong-gun, Daegu, Republic of Korea
[2,3]Dept. Of Computer Engineering, Yeungnam Univ., 280 Daehak-Ro, Gyeongsan, Gyeongbuk, Republic of Korea
[1]f13521@naver.com, [2]noke15@ynu.ac.kr, [3]youn@yu.ac.kr

Abstract

In the present age, the amount of malware is growing very rapidly, and the kinds and behaviors of malware are becoming very diverse. This poses social security threats such as data loss, leakage of personal and financial information, system damage, and the destruction of IT infrastructures. Unlike existing malicious codes, modified or new types of malicious codes are being identified, and it is very inefficient for analysts to manually analyze one by one from the beginning. Malware analysts to solve these problems are analyzed and studied effective way to reduce the time and cost of analysis. In this paper, we propose a way to express the characteristics by using the API Sequence for malware detection and classification. It compares and analyzes several existing expression methods and verifies the efficiency through actual malicious code samples. Using the expression method proposed in the paper, we detected four malicious behaviors: DLL Injection, Downloader, Key Logger, and Anti debugging. As a result, it was detected more than the conventional detection method, and it can be seen that the more complex the malicious behavior, the higher the detection efficiency. In addition, although static analysis was adopted as the main method, the flow of malicious behavior can be analyzed because it searches for execution condensation.

Keywords: *Malware analysis, API Sequence, Malware classification*

1. Introduction

Recently, with the rapid development of network and ICT technologies, the amount of malware has been increasing exponentially. In addition, malware variants that are made by altering existing malware have also been increasing rapidly, and the forms of malware attacks are also becoming quite diverse [1]. To prevent and respond to such security threats, analysts and anti-virus program producers are trying to maximize the efficiency of malware analysis using various analysis methods.

In general, malware analysis is performed in two formats, static analysis and dynamic analysis. In this paper, we propose a method of expressing malware characteristics using static analysis-based execution path search to maximize the efficiency of malicious behavior detection. Since the process is performed at the code level, it is based on static analysis, but you can expect the effect of dynamic analysis because it tracks the binary execution path to understand the behavior.

Article history:
Received (September 3, 2020), Review Result (October 6, 2020), Accepted (November 8, 2020)

eISSN: 2652-9807 JSTA

The composition of this paper is as follows. In Chapter 2, the methods of malicious behavior detection in previous studies are introduced and the limitations of those methods are mentioned. In Chapter 3, the method proposed in this paper is introduced. In Chapter 4, experiments are conducted based on the proposed method, the results are verified, and the efficiency of the method is mentioned and in Chapter 5, this paper is finished with conclusions.

2. Related works

Previous binary code based static analysis studies have been carried out by extracting the features of attributes in codes. Statistical algorithms were generally grafted on such studies and utilized for analysis. Such statistics are mainly utilized for comparison with normal programs. For instance, the statistics of op codes or strings extracted from the malware code sections are compared with those of general normal pro-grams for utilization in analysis. These methods simply collect signatures and identify malicious behaviors by extracting information on the structures of executable files. The most basic methods among the sequence-based malicious behavior identification methods mentioned above are those that list the sequences of op codes or the sequences of strings. These studies have been developed to carry out studies that identify malicious behaviors by using the n-gram technique [2], which cuts the information in the file according to a certain standard and processes the cut pieces of information. In addition, studies intended to express the byte sequences for binary codes with n-grams with a view to classifying malware were also carried out [3]. Such methods of collecting signatures for the internal structure of a file can be defined with the unique DNA of the file [4] and are used for similarity and classification of malware based on the foregoing. A clear and definite basis for judging malicious behaviors is the discovery of the functions used by malware. Previous studies have attempted to identify or classify malware by processing such APIs within programs. [5] First, methods that list the sequences of APIs, or collect log information on the use of APIs to determine malicious behaviors are representative. Since APIs are functions used when the program is executed, such APIs are either statically collected [6] or dynamically monitored [7]. In addition to the methods that simply list the APIs, there are methods that extract the features of malware according to the frequency of use of the APIs inside the file [8]. The above studies are statistical methods that have advantages such as not so large amounts of data to be stored, small amounts of operation, and high speed. However, they cannot respond to malware in real time and cannot accurately judge diverse malware behaviors because they are based on simple statistics. To compensate for the foregoing, some studies carried out recently grafted various algorithms onto the statistical properties as such to detect malicious behaviors. The eigenvalues of op code based graph images can calculated by measuring the distances between the nodes based on the K Nearest Neighbor Algorithm (KNN Algorithm), which is one of the machine learning algorithms [9]. In addition, the processed strings can be reprocessed with the Logistic Common Subsequence (LCS) algorithm to measure the eigenvalues of the strings [10]. In the studies introduced above, static analysis-based methods collect signatures or list the signatures in sequence, but cannot identify the accurate features of behaviors because they are based on code-based feature extraction. To compensate for this this problem, dynamic analysis is adopted as the main detection method [11] or a mixture of static and dynamic analyses is adopted [12]. However, since dynamic analysis is a method that directly executes malware for analysis, it has disadvantages of energy efficiency and analysis time [13]. In this paper, a method that is based on static analysis but tracks the execution flow will be proposed so that the effect of

dynamic analysis can be expected to compensate for studies in which static and dynamic analyses are mixed.

3. Proposed method

The core of the static execution path search in this paper is that the instruction set and subroutine are divided into true and false ones according to the branch instruction before they are searched. As for the branch point, comparison instructions such as cmp and test that occur before the branch instructions are issued are made through logical operation instructions such as xor. We divide true and false marks according to the branch instructions to search all instruction sets and subroutines.

The codes on the left of [Figure 1] are binary codes for showing static execution path searches. After CMP instruction in the fourth line, the binary code loc_401460 is branched into the instruction sets of loc_40148 due to the JNZ in the eighth line, which is a branch instruction. If the result of the comparison is true, the binary code will be branched into loc_401488, and if false, into loc_401484. To summarize finally, loc_401460 is a binary code, which is branched into loc_401488 when it is true and into loc_401488 when it is false. The binary code in Figure 1 is visualized as shown in the figure on the right. This mechanism is applied equally even when there are subroutines in the instruction set.

1	loc_401460 :		
2		mov	eax,[esp+argc]
3		sub	esp,44h
4		cmp	eas,2
5		Push	ebx
6		push	ebp
7		push	ebi
8		jzn	loc_401488
9	loc_401484:		
10		xor	eax,eax
11		jmp	short loc_40148D
12	loc_401488:		
13		sbb	eax,eax
14		sbb	eax,0FFFFFFFFh

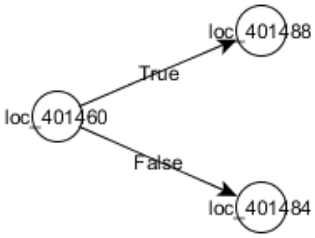

Figure 1. Example of static execution path exploration

In this study, malicious behaviors are detected based on Windows APIs. In previous studies, pieces of malware API information were simply collected based on signatures or simply listed as with n-grams to detect the similarity and behaviors of malware. These methods are efficient for simple classification of pieces of malware and the detection of the variants of the relevant pieces of malware because they simply list the APIs but they have a shortcoming that they cannot accurately detect the behaviors of malware.

The proposed method can detect malicious behavior based on API information found in binary execution path search. However, there is a problem that the number of APIs is too large to make the APIs into nodes. To solve such problems, the APIs will be reclassified into 25 upper categories through the functions of the APIs so that behaviors can be clearly judged and the temporal efficiency can be enhanced. For instance, Create File and Create Process are APIs that performs functions related to 'files' or 'processes' and APIs such as Get System Time and Get Local Time have the function to collect information on 'time' in the system.

Previous API sequence based static analysis studies had a shortcoming of being unable to accurately understand malware behavior because they simply listed or collected APIs. However, the method proposed in this study enables the understanding of the interactions between APIs because it uses execution path searches despite that it is a static analysis so that the effects of dynamic analysis can be expected.

As a representative example, when the malicious behavior of Trojan.Graftor.D4C56B has been analyzed by the method proposed in this paper, the graphic image shown in [Figure 2] appears.

[Figure 2] shows that the malware uses APIs such as String, System Information, Module, and Process. In particular, a detailed analysis of the red shaded API behaviors is as follows. In light of the fact that the relevant APIs use processes such as Open Process, Process32Next, Write Process Memory, and Virtual AllocEx and APIs used for manipulation of dlls, the API behaviors can be confirmed as DLL injection that inserts code into the remote process of calling Load Library to forcibly make the DLL to be loaded into the context of the relevant process. With regard to the behaviors shown in [Figure2], the existing simple API collection and listing method, the API Monitor based dynamic analysis, and the method proposed in this paper are compared as shown in [Table 1].

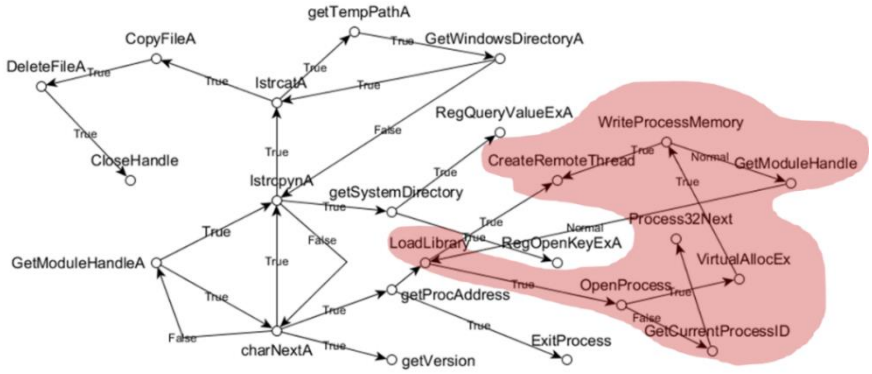

Figure 2. Trojan. Graftor.D4C56B's malicious behavior

Table 1. Comparison with dynamic analysis

API Sequence listing method	….LoadLibrary VirtualAllocEx lstrcmpA … OpenProcess GetCurrentProcessID Process32Next … GetModuleHandleA GetProcAddress getCurrentProcess … CreateRemoteThread …
Dynamic analysis	… NtCreateMutant … Process32Next OpenProcess VirtualAllocEx GetProcAddress WriteProcessMemory CreateRemoteThread … NtClose …
Proposed method	...OpenProcess → (True)VirtualAllocEx, → (True)WriteProcessMemory → (Normal)GetModuleHandleA → (Normal) CreateRemoteThread …

4. Results and discussion

The test set is 1236 pieces of randomly generated malware and all of them include an IAT (Import address Table) because the method proposed in this paper analyzes the interactions between APIs. First, through the identified common behavior graphs, each malicious behavior was analyzed based on the data set consisting of 1,236 pieces of malware.

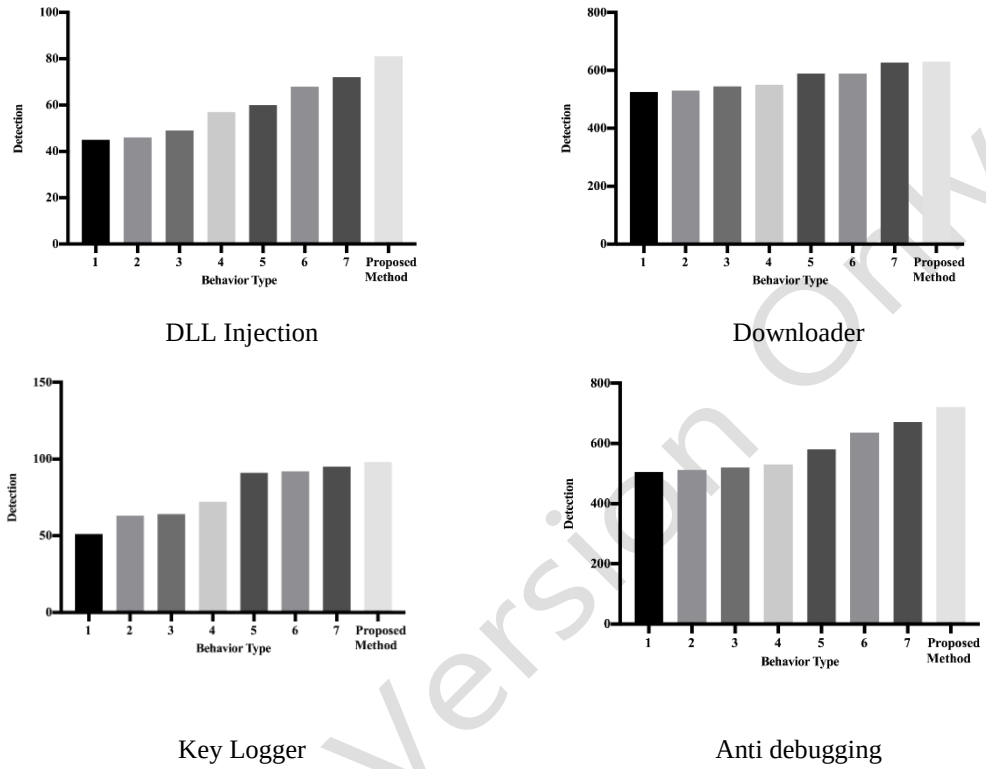

Figure 3. Comparison of malicious Behavior detection

[Figure 3] shows graphs of comparison between the method proposed in this study and the existing detection methods. It can be seen that the proposed methods show larger numbers of detection of the malicious behaviors, DLL injection and Anti Debugging when compared to the existing detection methods. However, it can be seen that the method proposed in this study shows very similar numbers of detection of other malicious behaviors such as Downloader and key logger when compared to the existing detection methods. In other words, the two malicious behaviors, Downloader and Key Logger are composed of two nodes, and the marks of the intermediate lines that show the interactions are True, so that only one sequence of each of the relevant behaviors is identified. This means that the structure of the sequence is too simple to detect malicious behaviors, which is the reason why the ccuracy of detection is lowered. However, it can be seen that the more complex malicious behavior, the higher the detection efficiency.

[Figure 4] is a graph showing the detection efficiencies according to the behavior complexity. The relevant efficiencies shown in the graph are the efficiencies of the method proposed in this paper in comparison with the highest efficiencies shown in the existing studies. The complexity of malicious behaviors in this study paper may also be regarded as the complexity of sequences. The efficiency of the method proposed in this paper was shown

to be 105% compared to existing studies when the complexity of sequences was low because the grounds for detection are reduced when the complexity decreases. On the contrary, the efficiency of the method proposed in this paper was shown to be 158% compared to existing studies when the complexity of sequences was high because the grounds for detection increase when the complexity increases. This means that the more complex the malicious behaviors, the higher the efficiency of detection.

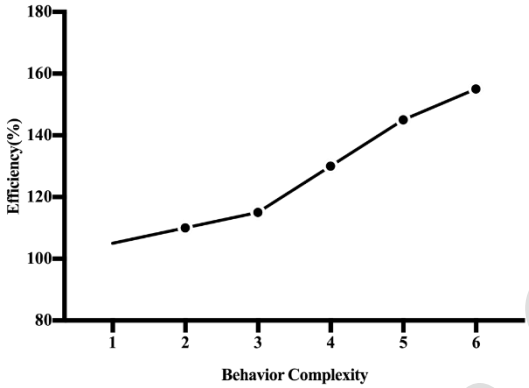

Figure 4. Detection efficiencies according to behavior complexity

5. Conclusions

Although static analysis is the main analysis, the method proposed in this paper enables analyzing the flow of behaviors because it searches execution paths. This means that although static analysis is adopted as a main method, the advantages of dynamic analysis that directly executes APIs to analyze the APIs are applied to method proposed in this paper. In this study, execution flows were analyzed according to branch instructions and the interactions of APIs collected during the flows were analyzed.

In future studies, the frequencies of behaviors will be added to prepare grounds for judgment of detailed behaviors. The utilization of such numerical data can be extended to apply machine learning and various statistics-based algorithms, and based on such data, malware will be visualized and malware similarity will be calculated.

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] S. COOK, “Malware statistics and facts for 2020,” Comparitech, Nov., **(2020)**

[2] I. Santos, Y. K. Penya, J. Devesa, and P. G. Bringas, “N-grams-based file signatures for malware detection,” ICEIS, vol.9, no.2, **(2009)**

[3] R. Moskovitch, C. Feher, N. Tzachar, E. Berger, M. Gitelman, S. Dolev, and Y. Elovici, “Unknown malcode detection using opcode representation,” Intelligence and Security Informatics, Springer, Berlin, Heidelberg, pp.204-215, **(2008)**

[4] Y. H. Choi, B. J. Han, B. C. Bae, H. G. Oh, and K. W. Sohn, "Toward extracting malware features for classification using static and dynamic analysis," Computing and Networking Technology (ICCNT), 2012 8th International Conference on. IEEE, pp.126-129, **(2012)**

[5] M. Zhang, Y. Duan, H. Yin, and Z. Zhao, "Semantics-aware android malware classification using weighted contextual api dependency graphs," Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security. ACM, **(2014)**

[6] M. Alazab, S. Venkataraman, and P. Watters, "Towards understanding malware behavior by the extraction of API calls," Cybercrime and Trustworthy Computing Workshop (CTC), 2010 Second. IEEE, pp.52-59, **(2010)**

[7] M. Rajagopalan, M. A. Hiltunen, T. Jim, and R.D. Schlichting, "System call monitoring using authenticated system calls," IEEE Transactions on Dependable and Secure Computing, vol.3, no.3, pp.216-229, **(2006)**

[8] M. Alazab, S. Venkatraman, P. Watters, and M. Alazab, "Zero-day malware detection based on supervised learning algorithms of API call signatures," Proceedings of the Ninth Australasian Data Mining Conference-Volume 121. Australian Computer Society, Inc., **(2011)**

[9] I. Firdausi, A. Erwin, and A.S. Nugroho, "Analysis of machine learning techniques used in behavior-based malware detection," Advances in Computing, Control and Telecommunication Technologies (ACT), 2010 Second International Conference on. IEEE, **(2010)**

[10] J. J. Blount, D. R. Tauritz, and S. A. Mulder, "Adaptive rule-based malware detection employing learning classifier systems: A proof of concept," Computer Software and Applications Conference Workshops (COMPSACW),2011 IEEE 35th Annual. IEEE, pp.110-115, **(2011)**

[11] O. Or-Meir, N. Nissim, Y. Elovici, and L. Rokach, "Dynamic malware analysis in the modern era—A state of the art survey," ACM Computing Surveys (CSUR), **(2019)**

[12] M. Ijaz, M. H. Durad, and M. Ismail, "Static and dynamic malware analysis using machine learning," 2019 16 th international bhurban conference on applied sciences and technology (IBCAST). IEEE, **(2019)**

[13] M. Egele, T. Scholte, E. Kirda, and C. Kruegel, "A survey on automated dynamic malware-analysis techniques and tools," ACM computing surveys (CSUR), vol.44, no.2, **(2012)**

This page is empty by intention.