Detecting Malicious Behaviors of Software through Analysis of API Sequence k-gramsⁱ

Hyun-il Lim

Department of Computer Engineering, Kyungnam University, South Korea

Copyright©2016 by authors, all rights reserved. Authors agree that this article remains permanently open access under the terms of the Creative Commons Attribution License 4.0 International License

Abstract Nowadays, software is widely applied to increase accuracy, efficiency, and convenience in various areas in our life. So, it is essential to use software in our recent computing environments. Despite of the valuable applications of software, malicious behaviors caused by vulnerability of software threaten our secure computing environments. So, it is important to identify and detect malicious behaviors of software for maintaining computing environments. In this paper, we propose an approach to detecting malicious behaviors of software by analyzing information of API function calls. API function calls are essentially used to make use of various services provided by operating systems or devices in developing software. In addition, API functions can describe the behaviors of software because they perform predefined specific operations during program execution. In this paper, we classify API functions in Microsoft Windows operating systems, and propose an approach to representing malicious behaviors of software with API functions. We propose an approach to detecting malicious behaviors of software by analyzing dynamic API function calls. To increase the efficiency and the tolerance of the analysis, malicious behaviors are abstracted as sets of k-grams, and they can be identified by calculating similarity between the sets of k-grams and a sequence of API function calls.

Keywords Malware Detection, Software Behavior Analysis, Malicious Behavior, Dynamic API Analysis

1. Introduction

Malware is an abbreviation of malicious software, which means software containing malicious behaviors in its execution. Malware code can be added or modified in a software system in order to have a bad influence on the system by intended malicious functions [4, 5]. So, malware is harmful in normal execution of software system. In recent computing environments, software is widely used in various areas to increase productivity, efficiency, and convenience in

our real life. Despite of the widespread useful applications of software, it is major concerns to solve the problems of malicious behaviors that abuses vulnerability of software in maintaining safe computing environments. So, detection of such malicious behaviors of software is helpful to improve the safety of our computing environments.

The detection methods of malicious behaviors are classified into two ways according to approaches to figure out the malicious codes contained in programs. The first way is signature-based methods, which can detect the existence of malicious codes in programs by finding the previously known signatures of malware. The signature of malware means specific patterns of code or data that are used to perform malicious operations in programs without users' permission. In applying this approach, the signature set of malicious behavior should be previously analyzed and defined before detecting malicious behaviors. This method is very effective in detecting previously known malware that is already analyzed and defined because it can be easily detected by confirming the existence of the signatures in programs. However, in the case of newly appearing malware that has no signatures yet, the method cannot be applied to detect such malware. On the other hand, the second way is behavior-based method that can detect the existence of malicious behavior in programs by analyzing static or dynamic behaviors in programs. The behaviors of programs can be described various ways to represent the essential goal and operations identified by static analysis or dynamic analysis during program execution. So if malicious behaviors of software can be described and identified through behavior analysis, malicious behaviors can be detected by confirming the existence of the behaviors. In this paper, we describe an approach to identifying malicious behaviors of software through dynamic analysis of API function calls.

This paper is organized as follows. Section 2 summarizes related work on detecting malicious behaviors of software. Section 3 describes the proposed algorithm for tracing API function calls in Microsoft Windows operating system. Section 4 describes the method for analyzing API function calls extracted during program execution. Section 5 presents

approaches to identifying malicious behavior of software through analysis of API functions calls and k-gram abstraction. Finally, section 6 concludes this paper.

2. Related Work

There have been several researches on detecting malicious behaviors of software. Wang et al. [1] presented an approach of virus detection based on analysis of the suspicious behaviors indicated by the sequences of APIs in Windows environments. This approach applied Bayes algorithm to identify suspicious behaviors and detect viruses. Beaucamps et al [2] presented an approach for malware detection by abstraction of program behaviors. The technique abstracts program traces by rewriting original information into abstract symbols representing their functionality. Then, suspicious behaviors are detected by comparing trace abstractions to identify malicious behaviors. Babic et al [3] proposed an approach to learning and generalizing from observed malware behaviors based on tree automata generated from system call dataflow dependency graphs. Alazab et al [5] presented an approach to analyzing and classifying the behavior of API function calls based on the malicious intent hidden within any packed program. The method classified six main categories of suspicious behavior of API call features and analyzed API call distribution of malware samples. Veeramani et al [8] proposed the malware detection method based on extracting relevant API calls from sub categories of malware. Malware is categorized based on their infection mechanism and actions performed according to the type of API calls in malware categories. Wang et al [9] presented an approach of malware detection that is based on analysis of representative characteristic and systemic description of the suspicious behaviors indicated by the sequences of APIs called under Windows. Faruki et al [10] proposed a behavior model that represents abstraction of a binary by analyzing the API strings made by Windows Portable Executable (PE) files. The method extracted temporal snapshots of malware and benign executables known as API Call-grams. Alazab et al [12] proposed an approach to detecting obfuscated malware by investigating the structural and behavioral features of API calls. This method identified malicious behavior by using n-gram statistical analysis of binary content. Elhadi et al [13] presented an approach to detecting malware using API call graph. In this approach, malware sample was represented as data dependent API call graph. Then, a graph matching algorithm based on longest common subsequence algorithm was used to calculate similarity between the input sample and malware API call graph. Ravi et al [14] proposed an approach to detecting malware by using the Windows API call sequence. This method used k-grams to model the API calls and applied iterative machine learning process combined with the run-time monitoring of program execution behavior.

3. Monitoring API Calls

In developing software, functions are used for operations to perform specific tasks during program execution. Design and use of functions increase reusability and productivity by modularizing software in software development process. In the viewpoint of software behavior, the calling information of functions in software can be used to summarize the specification of software behavior. So, the calling information of software is helpful to understand how the software behaves and interacts with each other. By monitoring API function calls, the information can be collected and analyzed to discover the behavior of software. To monitor API calls in software, hooking method can be used to intercept function calls and record analyzed information of the calls [1].

API functions and system calls are related with services provided by operating systems. API functions and system calls support various key operations provided by operating systems, such as networks, security, system services, file managements, and so on. In addition, they include various functions for utilizing system resources, such as memory, file system, network, or graphics. Because there is no other way for software to access system resources that are managed by operating systems without using API functions or system calls, such API functions or system calls are essentially used to access system resources during program execution. From the viewpoint of this characteristic of software, it is essential to analyze the patterns of API function calls that represent the interactions between software and operating systems. In addition, the patterns of API function calls can provide key information that can be used to detect the movement of software and to represent behaviors of the software, because they give meaningful information on the behaviors of software. So, analysis on API functions and system calls plays an import role in behavior analysis of malware.

In detection of malicious behaviors of software, API calling sequence can also provide important information, because API function calls may represent behaviors of software performed to use system resources in operating systems, such as file management, network access, system service access, and memory access.

Figure 1. Analysis of API calling sequences of program execution

Figure 1 shows the procedures for analyzing API calling sequences during program execution. Each API function can represent specific behavior of software in execution, and the sequence and patterns of the API function calls can

be used to summarize and represent malicious behaviors of software. To monitor behavior of software, API function calls are traced and analyzed during program execution.

In order to trace API function calls in Microsoft Windows runtime environments, we first should analyze the static characteristics of software through Windows PE file analysis. Then, we attach the runtime analyzer to the software to perform dynamic analysis. Figure 2 shows the main algorithm for tracing API functions called during program execution in Microsoft Windows operating system.

This analyzer consists of three phases, namely `main_analyzer`, `API_manager`, and `API_monitor`. At first, the `main_analyzer` initializes the overall analysis procedures before executing a program. It prepares for analysis procedures by attaching the `API_manager` analyzer to the program before executing the program. After calculating the OEP of the software, it sets a breakpoint to the location to initiate the API analysis procedure. The initialization procedure sets up analysis environments, and it begins execution of the program and analysis.

`API_manager` is a procedure for handling events that occur at the breakpoint set on the OEP of the program at the step (2). This procedure collects the information used in tracing API function calls during program execution, and

controls to continue the analysis and execution of software. It obtains the information of dynamic link library (DLL) loaded during program execution and the list of API functions from the import address table through analysis of PE files of the program. After calculating the memory load address of each API function, it saves the address and function name information to monitor the API function calls. Then, the procedure sets breakpoints on every address of API functions and attaches `API_monitor` procedure to analyze API traces when API functions are called in execution. After setting breakpoints and attaching the monitor to the program, it continues execution to monitor and analyze the API function calls.

`API_monitor` handles the events that occur at the breakpoints set on the addresses of API functions. Whenever API functions are called, `API_monitor` is called to monitor API function calls and analyze the behavior of software. `API_monitor` can calculate the start address of current execution program from the EIP, and it can find the name of API functions loaded at the current address through the information saved by `API_manager` at the step (7). After processing the information of the API function call, it continues the execution to perform next operations.

main_analyzer	<ul style="list-style-type: none"> (1) The <code>main_analyzer</code> finds the OEP (original entry point) of software through Windows PE file analysis [11]. (2) It sets a breakpoint to the OEP, and attaches <code>API_manager</code> to the software. (3) It continues the analysis.
API_manager	<ul style="list-style-type: none"> (4) When the execution program reaches the breakpoint set on the OEP of the software, the analyzer stops the execution. (5) The analyzer takes the import address table from the program, and it finds the dynamic library modules and the lists of API functions that can be used in the program execution. (6) It calculates the memory load addresses of API functions. (7) It saves the memory load addresses and API function names. (8) It sets breakpoints to the memory load addresses of API functions and attaches <code>API_monitor</code> to the software. (9) It continues the analysis.
API_monitor	<ul style="list-style-type: none"> (10) When the execution program reaches the breakpoint set on the memory load addresses of API functions, the analyzer stops the execution. (11) The analyzer finds the current EIP (instruction pointer) and get the API function names from the information saved at the step (7). (12) It traces and records the API functions called in the execution. (13) It continues the analysis.

Figure 2. Algorithm for tracing API functions called in program execution in Windows environments.

EIP: 0x7c7db741	API CALL: KERNEL32!GetModuleHandleA
EIP: 0x77be537c	API CALL: msvert!__set_app_type
EIP: 0x77bcf1db	API CALL: msvert!__p_fmode
EIP: 0x77bcf1a4	API CALL: msvert!__p_commode
EIP: 0x77bfee4f	API CALL: msvert!_controlfp
EIP: 0x77be9d67	API CALL: msvert!_initterm
EIP: 0x77bceeb	API CALL: msvert!__getmainargs
EIP: 0x77be9d67	API CALL: msvert!_initterm
EIP: 0x7c7d1ef2	API CALL: KERNEL32!GetStartupInfoA
EIP: 0x7c7db741	API CALL: KERNEL32!GetModuleHandleA
EIP: 0x77d363e4	API CALL: USER32!GetProcessDefaultLayout
EIP: 0x7c7e7023	API CALL: KERNEL32!GetCommandLineW
EIP: 0x77d0b1b0	API CALL: USER32!CharNext
...	

Figure 3. An extracted partial trace of API function calls during execution of the Windows calculator application

Figure 3 shows a partial trace of API function calls in the Windows calculator application analyzed by the API_monitor. In the trace, the list shows the API function calls and the addresses during the program execution. This results show the behavior of API functions for initializing the calculator application.

4. Analyzing API Calls

In this section, we describe an approach to analyzing the trace of API function calls to represent malicious behaviors of software. To abstract and summarize the behaviors of software, API functions are classified into several groups according to their functions. Then, malicious behaviors are represented as behavior automata to describe the specific behaviors of malicious software.

Table 1. The group of API functions to analyze malicious behaviors for exploiting file operations [7].

Access to the information of system folders	GetWindowsDirectory GetSystemDirectory
Access to files	FindFirstFile FindNextFile FindClose GetFileSize GetFileType ReadFile
Write files	CreateFile OpenFile WriteFile CloseHandle CopyFile DecryptFile EncryptFile DeleteFile ReplaceFile
Modify the attributes of files	GetFileAttributes SetFileAttributes
Modify the time of files	GetFileTime SetFileTime
Move the location of files	MoveFile MoveFileEx MoveFileTransacted MoveFileWithProgress

In general, malware tries to hide the presence of itself. However, it should contain suspicious behaviors that are distinctive from the behaviors of the other benign programs while performing intrinsic operations of the malicious behavior. So, we can detect the possibility of maliciousness of software if the software contains such suspicious behavior during program execution. Because most of these kinds of suspicious behaviors are performed in kernel mode

or as services provided by operating systems, API functions and behaviors can be classified into several groups according to the characteristics and the specifications of API functions. For example, Table 1 shows the group of API functions to analyze behaviors for exploiting file operations in malicious behaviors. Careful monitoring of classified API function calls is helpful to identify the behaviors of software and detect malicious behaviors.

Because the length and the number of API function calls continuously increase as the program continues the execution, an effective representation and identification method is required. The method should have a form that can efficiently express the behaviors of software during program execution. So, monitoring and analyzing all kinds of API function calls are not appropriate for representing and detecting malicious behavior, because there will be too many kinds of API functions that are unrelated to malicious behaviors. The group of API functions related to malicious behaviors can play an important role to express behavior of software efficiently. The first step is to refine the set of API functions to be analyzed according to individual malicious behavior to avoid the side effect that may be caused by unrelated API function calls. By refining the API function set of behaviors, we can decrease the length and the number of API function call sequence efficiently. Table 1 shows an example for refining the set of API functions to analyze the behaviors that are related to file operations.

Each API function call performs a basic operation in the course of performing complex tasks, which may be either malicious or good behaviors. So, the patterns and sequence of API function calls can be classified into several groups according to the tasks of software. Then, malicious behaviors are extracted from the patterns and sequence of API function calls. The malicious behaviors are represented as forms of behavior automata to effectively describe and abstract the maliciousness.

Figure 4. Behavior automaton for the Allaple.A excerpt [2]

Figure 2 shows a behavior automaton for the Allaple.A excerpt which are pinging remote hosts and scanning local drives [2]. The behavior automaton represents the distinguishing malicious behavior of the software. It shows operations of the the Allaple.A worm which try to ping remote host to find and infect other system and scan local drives to infect and propagate to other system without owner's permission.

The automaton for malicious behavior is constructed from the control flow graph of a program. At first, the control flow graph of a program is abstracted by extracting API calls in the program. From the abstract control flow

graph of API functions, its malicious behavior is analyzed by tracing API function calls of the behavior. Finally, the API flow graph of malicious behavior is represented as automaton for the malicious behavior. The behavior automaton of malicious software is represented, and it can be used to detect malicious behaviors in analyzing software during execution.

5. Identifying Malicious Behaviors

Malicious behaviors of software can be represented as forms of automata, and the behaviors are recognized by confirming that one of the behavior automata is matched with the sequence of API function calls during program execution [3]. The recognition is accomplished by tracing and matching malicious behavior automaton with the API calling sequence extracted from a program execution. So, after extracting a sequence of API function calls during program execution, the sequence is analyzed to decide whether it is matched with a malicious behavior or not.

In tracing malicious behavior automaton of a program, the first transition starts from the start state of the behavior automaton. The current state of the behavior automaton transits to the next state by following each API function call. In the process of tracing the malicious behavior automaton with a sequence of API function calls, if the next state reaches one of the final states of the malicious behavior automaton, the sequence is matched with the behavior automaton. Therefore, the malicious behavior is identified by confirming that the sequence is satisfied with the behavior automaton. Figure 5 shows the analysis procedure for identifying malicious behaviors with a trace of API function calls.

As the execution time goes by, the number and the length of API function calls rapidly increase. So, it is preferable to accomplish the detection process at the early stage of the analysis. On the other hand, if detection of malicious behaviors does not succeed at the early time of the analysis, the complexities of the analysis will rapidly increase as time goes by. To make matters worse, if there is minor mutation in the behavior of malware, the analysis for identifying the malicious behavior will fail due to the minor differences of API function calls caused by the mutation. To mitigate this problem, it is helpful to abstract the behavior automata of malicious behaviors and to identify the behaviors by calculating the similarity between the abstract behaviors and API function calls.

A k-gram denotes a contiguous substring of length k that can be comprised of letters, words, or opcodes in a binary program [6]. In this analysis, the set of k-grams generated from the behavior automaton is used to abstract a malicious behavior. Then, the set of k-grams is compared with a sequence of API function calls to calculate the similarity between the behaviors and API function calls. The k-gram method only considers the limited k consecutive subsequence of a full sequence of API calls in comparison. Although there is minor mutation in the behavior of malware, such as insertion or omission of a few calls of API functions, the mutation can affect only a few number k-grams adjacent to the change. So, the remaining k-grams of the API functions, which are not affected by the mutation, still can be matched with each other. If the similarity between the set of k-grams and the sequence of API function calls is considerably high, it can be a good evidence of partial inclusion of malicious behaviors in software.

Figure 5. The analysis procedure for identifying malicious behavior with a trace of API function calls

Figure 6. The analysis procedure for identifying a behavior automaton through k-gram abstraction

Figure 6 shows the analysis procedure for identifying malicious behaviors by comparing a trace of API function calls and abstract behaviors represented as a k-gram set with the length $k=3$. In the procedure, the original behavior automaton is abstracted into a set of k-grams, which are substrings of API functions with $k=3$ from the behavior automaton. In analyzing the abstract behavior of k-grams and API function calls, the behavior k-grams are compared with a trace of API function calls.

Let S be a set of k-grams of malicious behavior and A be a set of k-grams generated from a trace of API function calls during programs execution. The similarity between the behavior and the API function calls is calculated as follows:

$$sim = \frac{|S \cap A|}{\min(|S|, |A|)}$$

where $|S|$ denotes the number of elements in the set S . So, the similarity is decided by the number of identical k-grams between the set of the behavior and API function calls. Because the original behavior automaton is abstracted into a form of k-grams, the analysis can tolerate minor differences in API function call sequences to a certain degree, which can be defined by a difference ratio between behavior k-grams and API function calls.

Figure 7. An example behavior automaton of a malicious behavior

For example, let $\{A, B, C, D, E, F\}$ be a set of API

functions called in program execution, and let Figure 7 be the behavior automaton of a malicious behavior. Then, the set S of k-grams of the behavior automaton with $k=2$ is $\{(A, B), (A, F), (A, E), (B, C), (F, D), (E, A), (C, D), (D, F), (D, B), (D, E)\}$. Let $(A, B, C, D, B, F, D, E, A, C, F, D, B, C, D)$ be a sequence of API functions called during program execution. Then, the set of API k-grams is $\{(A, B), (B, C), (C, D), (D, B), (B, F), (F, D), (D, E), (E, A), (A, C), (C, F), (F, D)\}$. So, we can calculate the similarity between the behavior and the API function calls as follows:

$$sim = \frac{|S \cap A|}{\min(|S|, |A|)} = \frac{|\{(A, B), (B, C), (C, D), (D, B), (F, D), (D, E), (E, A), (F, D)\}|}{\min(10, 11)}$$

From the calculation, the similarity is decided to be 80%. If the similarity value is higher than predefined tolerance value, then we can identify the presence of the malicious behavior from the sequence of API function calls during program execution.

6. Conclusions

Nowadays, software is practically applied in various areas to increase accuracy, efficiency, and convenience in our life. Software can also provide valuable operations in our recent computing environments. In some cases, however, malicious behaviors caused by vulnerability of software have bad effects in the environments. So, it is one of major concerns to secure safe computing environments by detecting malicious behaviors of software. In this paper, we propose an approach to detecting malicious behaviors of software by analyzing information of API function calls. Malicious behavior of software is represented as a form of behavior automata generated from API flow graph of a program. To recognize the malicious behavior of software, its behavior automaton is

traced with a sequence of API function calls extracted during program execution. To increase the efficiency and the tolerance of the analysis, a behavior automaton is abstracted as a set of k-grams. Malicious behavior can be identified by calculating similarity between the set of k-grams and a sequence of API function calls.

REFERENCES

- [1] Cheng Wang, Jianmin Pang, Rongcai Zhao, and Xiaoxian Liu, Using API Sequence and Bayes Algorithm to Detect Suspicious Behavior, In Proceedings of International Conference on Communication Software and Networks (ICCSN '09), pages 544-548, Feb. 2009.
- [2] Philippe Beaucamps, Isabelle Gnaedig, and Jean-Yves Marion, Behavior Abstraction in Malware Analysis, In Proceedings of the First International Conference Runtime Verification (RV 2010), LNCS 6418, pages 168-182, November 2010.
- [3] Domagoj Babic, Daniel Reynaud, and Dawn Song, Recognizing malicious software behaviors with tree automata inference, Formal Methods in System Design, Vol. 41, No. 1, pages 107-128, August 2012.
- [4] G. McGraw and G. Morrisett, Attacking malicious code: A report to the infosec research council, IEEE Software, Vol. 17, No. 5, pages 33-44, 2000.
- [5] Mamoun Alazab, Sitalakshmi Venkataraman, and Paul Watters, Towards Understanding Malware Behaviour by the Extraction of API Calls, Workshop on Cybercrime and Trustworthy Computing (CTC), pages 52 – 59, July 2010.
- [6] Ginger Myles and Christian Collberg, K-gram based software birthmarks, In the Proceedings of the 2005 ACM symposium on applied computing (SAC '05), pages 314-318, 2005.
- [7] Windows Developer Center, <https://msdn.microsoft.com/en-us/windows>
- [8] Veeramani R and Nitin Rai, Windows API based Malware Detection and Framework Analysis, International Journal of Scientific & Engineering Research, Volume 3, No. 3, March 2012.
- [9] Cheng Wang, Jianmin Pang, Rongcai Zhao, Wen Fu, and Xiaoxian Liu, Malware Detection Based on Suspicious Behavior Identification, In the Proceedings of the First International Workshop on Education Technology and Computer Science (ETCS '09), pages 198-202, March 2009.
- [10] Parvez Faruki, Vijay Laxmi, M. S. Gaur, and P. Vinod, Behavioural detection with API call-grams to identify malicious PE files, in the Proceedings of the First International Conference on Security of Internet of Things, pages 85-91, 2012.
- [11] Microsoft Corporation, Microsoft Portable Executable and Common Object File Format Specification, 1999.
- [12] Manoun Alazab, Robert Layton, Sitalakshmi Venkataraman, and Paul Watters, Malware Detection Based on Structural and Behavioural Features of API Calls, in the Proceedings of the 1st International Cyber Resilience Conference, August 2010.
- [13] Ammar Ahmed E. Elhadi, Mohd Aizaini Maarof, and Bazara Barry, Improving the Detection of Malware Behaviour Using Simplified Data Dependent API Call Graph, International Journal of Security & Its Applications, Vol. 7 Issue 5, September 2013.
- [14] Chandrasekar Ravi and R Manoharan, Malware Detection using Windows API Sequence and Machine Learning, International Journal of Computer Applications, Vol 43, No. 17, pages 12-16, April 2012.

ⁱ This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science and Technology(2010-0024658).