J Inf Process Syst, Vol.17, No.4, pp.851~865, August 2021
https://doi.org/10.3745/JIPS.03.0163

ISSN 1976-913X (Print)
ISSN 2092-805X (Electronic)

JOURNAL OF INFORMATION PROCESSING SYSTEMS JIPS

Host-Based Malware Variants Detection Method Using Logs

Woo-Jin Joe* and Hyong-Shik Kim*

Abstract

Enterprise networks in the PyeongChang Winter Olympics were hacked in February 2018. According to a domestic security company's analysis report, attackers destroyed approximately 300 hosts with the aim of interfering with the Olympics. Enterprise have no choice but to rely on digital vaccines since it is overwhelming to analyze all programs executed in the host used by ordinary users. However, traditional vaccines cannot protect the host against variant or new malware because they cannot detect intrusions without signatures for malwares. To overcome this limitation of signature-based detection, there has been much research conducted on the behavior analysis of malwares. However, since most of them rely on a sandbox where only analysis target program is running, we cannot detect malwares intruding the host where many normal programs are running. Therefore, this study proposes a method to detect malware variants in the host through logs rather than the sandbox. The proposed method extracts common behaviors from variants group and finds characteristic behaviors optimized for querying. Through experimentation on 1,584,363 logs, generated by executing 6,430 malware samples, we prove that there exist the common behaviors that variants share and we demonstrate that these behaviors can be used to detect variants.

Keywords

Big Data, Host-Based Detection, Log, Malware Variants, Sysmon

1. Introduction

Enterprise networks in the PyeongChang Winter Olympics were hacked and approximately 300 hosts were compromised for the purpose of interfering with the Olympics in February 2018 [1,2]. To avoid such scenarios in the future, enterprise have no choice but to rely on vaccines because analyzing all programs executed in the host used by ordinary users is an overwhelming task. However, traditional vaccines cannot protect the host against variant and new malware because they cannot detect intrusions without signatures for malwares.

A host used by ordinary users can be easily infected because it is exposed to several attacks. Vulnerable personal computers can be infected simply by accessing malicious sites [3]. If one host in an enterprise network is infected with malware, then other hosts connected to the network can be infected because malware tends to duplicate itself through a network. When an infection occurs in an enterprise network, security officers attempt to find the origin of the attack. They perform digital forensics, which plays an important role in almost every criminal investigation encompassing the recovery and investigation of

Manuscript received October 16, 2019; first revision January 21, 2020; accepted March 16, 2020.
Corresponding Author: Hyong-Shik Kim (hkim@cnu.ac.kr)
* Dept. of Computer Science and Engineering, Chungnam National University, Daejeon, Korea (woojin.jo95@gmail.com, hkim@cnu.ac.kr)

material found in digital devices. However, digital forensics faces many challenges that require future research [4].

In the last decade, data mining techniques have been utilized to detect malware more efficiently. Frequent pattern mining, packet inspection using naive Bayes, and API call sequence detection using N-gram have been researched for detecting malwares using static and dynamic program information [5]. Despite these efforts to detect malwares, they are evolving to bypass such analysis methods. Malware is equipped with anti-debugging, which is a technique to kill the debugger when it is detected, anti-virtualization, which is a technique to terminate the malware itself when it is running in a virtual machine. In this manner, many anti-analysis techniques are applied to interrupt malware analysis [6].

Among them, the techniques to make variants are frequently used because they reduce the time and cost of making new malware. A malware variant is a slightly altered version of pre-existing malware to bypass a vaccine. According to the 2019 malware statistics presented by an IT security research institute (Av-Test), most detected malwares are not new ones but variants [7]. Therefore, there is a need for research on malware variants detection.

To detect malware variants, behavior-based analysis is an effective method because techniques to make variants do not change behaviors; they only modify the execution file to modify its signature. Previous studies have performed behavior analysis to overcome the limitations of signature-based detection. Most previous studies rely on a sandbox which analyzes only the information related to the target program; however, we could not analyze the programs executed in the host, in this manner, where many normal programs are generating excessive behavior information.

Therefore, this study proposes a method of detecting variants in the host through the logs rather than the sandbox. Since every program behavior occurred in the host is recorded in the logs, it is possible to detect the intruding variants by querying specific behaviors that can identify the variants. However, there are two problems in utilizing the logs, namely cost and accuracy. The query cost is expensive since normal program behaviors are also recorded in the logs and false positives can occur since the behaviors that malware perform can also be performed by normal programs.

Therefore, the objective of this study is to find the characteristic behaviors with high identifiability for variants in order to reduce query cost and improve accuracy. First, we extract common behaviors from a variants group, that is a collection of variants derived from a single malware, to exclude unnecessary behaviors for detecting variants. Then, we find the characteristic behaviors that return few logs and low false positives when the behaviors are queried. Through various experiments, we demonstrate the process of finding the characteristic behaviors and detecting variants from 1,584,363 logs generated by executing 6,430 malwares samples.

2. Related Work

Most malwares are generated by reusing code from other malware to reduce creation time. Thus, a lot of the code in new malware overlaps with existing malware. Based on this characteristic, Kim and Im [8] suggested a method to detect malware by examining the common code between two programs. They first extracted disassemble code from the programs by the Linear Sweep method and then divided that by function because many programs reuse code at a function level. Since there are a lot of functions in a

program, they selected major function which calls API, and they checked the function similarity of the two programs. They then calculated the similarity of functions using Jaccard similarity to extract common functions, and lastly, they filtered out the noisy functions that are used by a compiler. In an experiment, they confirmed that there are a few common functions between the programs that have no variant relationship and there are a lot of common functions between variants.

Modern malware variants apply sophisticated packers to bypass analyzing methods. To analyze the packed malwares, unpacking techniques and dynamic analysis need to be considered. However, unpacking is very difficult when sophisticated packers are used, and dynamic analysis may have low accuracy because packers cause noisy behaviors that can be included in a dynamic information result. To overcome these challenges, Zhang et al. [9] proposed extracting a series of sensitive system calls that are frequently used among the variants without unpacking. They assumed that there exists a significant difference in distributions between malware and normal programs, and some system calls are used more often in variants. In addition, they embedded their frequencies into a vector and adopted a deep leaning method. They proposed a principal component initialized multi-layer neural network as an efficient and effective classifier to classify the packed malicious variants from the packed legitimate ones. This method achieved 95.6% detection accuracy and 0.048 seconds of classification time cost, in addition to a very low false positive rate, indicating that the method seldom makes mistakes with packed benign programs.

An automated tool that can generate malware variants has recently emerged, and this has resulted in an increased creation rate of variants. Traditional detection methods such as signature-based detection cannot deal with the increased generation rate. To detect variants generated by the automated tool, Park et al. [10] conducted research on the detection of malware variants using common features. They assumed that variants generated by the automated tool have similar features because the essence of the logical structure is the same. They defined three static features (strings, DLLs, and APIs) and four behaviors (file creation, registry modification, process creation and network connection). Then, they used the sandbox to extract the defined features and they calculated their similarity by assigning different weights to each feature. They collected nine types of variant samples using the malware generation tool. Their experiments on the variant samples confirmed that variants have high similarity in their defined features (81.3% to 99%).

As an advanced persistent threat (APT) attack that has a specific target have become more widespread, studies on collecting behaviors in the host have been published. Since traditional detection methods such as firewalls, intrusion detection systems, and anti-viruses use signature-based methods, they cannot deal with new attacks like zero-day or malware variants. To overcome these limitations, Moon et al. [11] collected behavior events occurring in the hosts in order to detect APT attack. They defined 39 behavior features and collected them by monitoring APIs that implement the behavior features. They executed 3,133 malwares and 1,049 normal files to collect behavior events and achieved a false positive rate of 2.0% and a false negative rate of 5.8% by applying a decision tree. However, collecting 39 features is costly when there are multiple hosts.

Most of the previous studies utilized APIs or the sandbox that analyzes only the target program. These studies may provide effective analysis for one program. However, they are not able to analyze programs executed in the host since many normal programs are generating excessive information. Therefore, in this study, we focus on analyzing the programs executed in the host and detecting malware variants through the logs.

3. Proposed Model

A log is a record of an event and can be used for various purposes such as security audits, intrusion detection, and system error analysis [12]. We utilize the logs that record events occurring in a host to detect malware executed in the host. If we record the events occurring in the host, the behaviors performed by malware intruding the host will be recorded in the log files and can be detected through querying their behaviors from the logs. For this purpose, in Section 3.1 we introduce a logging tool that records the events occurring in the host and we introduce a method to detect the malware through the logs with low cost and high accuracy.

A malware variant is a slightly altered version of pre-existing malware to bypass a vaccine. It is made by obfuscation or encryption technique that modifies their signature so that a signature-based vaccine cannot detect it. However, the behaviors of the variant are unchanged because those techniques modify the execution file. As a result, variants derived from a specific malware have similar behaviors and unknown variant can be detected by the behaviors of previously detected variant.

The proposed method is based on this feature. Since the variants have similar behaviors, the variants intruding a host can be detected by querying the specific behaviors that other variants performed. However, even if the same program runs twice, some behaviors may not be similar, such as creating an arbitrary file. In addition, the logs may sometimes be too big to query since they also record normal program behaviors. Therefore, we must query the characteristic behaviors that return few logs and low false positives. For this purpose, we introduce a method to find characteristic behaviors in Section 3.2.

3.1 Method to Query Behaviors from Logs

In this section, we introduce a tool that records program behaviors in logs and we introduce a method to query the behaviors from those logs. To record the behaviors in the logs, we use a tool called Sysmon [13] which is a compound word of system and monitor and provided by Sysinternals to complement the Windows event log. When Sysmon is installed in a host, it is registered as a Windows system service and runs automatically in kernel-mode to monitor system activities and record them in the Windows event log.

Sysmon records 22 event types, and there are many data explaining the events. Fig. 1 is an example of a Sysmon log about an event “File creation time changed.” There are many data explaining the event such as the occurrence time in the UtcTime field and the process name in the Image field. We can obtain data about the occurred event through certain programs executed in the host and we can query specific data from the logs to trace the events that occurred due to malwares.

```
File creation time changed:
RuleName:
UtcTime: 2019-06-18 05:07:37.551
ProcessGuid: {f0468001-536e-5d08-0000-001016cbf902}
ProcessId: 12868
Image: C:₩Users₩jwj₩AppData₩Local₩Microsoft₩OneDrive₩OneDrive.exe
TargetFilename: C:₩Users₩jwj₩AppData₩Local₩Microsoft₩OneDrive₩settings₩Business1₩global.temp.ini
CreationUtcTime: 2018-12-28 10:14:51.333
PreviousCreationUtcTime: 2018-12-28 10:14:51.333
```

Fig. 1. Example of Sysmon log.

We need to define behaviors using log data to query the behaviors from a log. The data in the log should be fixed regardless of the execution environments so that we can utilize the defined behaviors for query from logs. For example, even if the same program runs second time, the execution time is different and the process id is reallocated every time it runs. These data are not fixed and are not suitable for constructing the behaviors.

Therefore, we define the behaviors as a pair of Event Id and Event Target which are fixed regardless of the execution environments. Event Id is an identifier of the occurred event and Event Target is a target or object of the event. For example, if some program created a file named “temp,” the Event Id is “11” which refers to a file creation event and the Event Target is “temp.” This behavior is fixed even if the program runs a second time.

To define behaviors using Event Id and Event Target, we need to extract them from the logs. In the case of Event Id, since it is meta data for the event, it is recorded in a log header in an “Event Id” field. In the case of Event Target, since the name of the field is different according to the Event Id, it should be extracted from a field that is appropriate for an Event Id. We summarize the fields appropriate for Event Id in Table 1. While Sysmon records 22 Event Ids, we use only 13 that are related to process behaviors as shown in Table 1. The flowchart for extracting behavior from the Sysmon log is as shown in Fig. 2.

Table 1. Event Id and Event Target field of Sysmon logs

Event Id	Description	Event Target field
1	Child Process creation	Image
2	File creation time change	TargetFilename
3	Network Connection	DestinationIP
7	Image loaded	ImageLoaded
8	CreateRemoteThread	TargetImage
10	Process Access	TargetImage
11	File Create	TargetFilename
12	Registry create, delete	TargetObject
13	Registry Value Set	TargetObject
14	Registry Key, Value Rename	TargetObject
15	FileCreateStreamHash	TargetFilename
17	PipeEvent (Pipe Created)	PipeName
18	PipeEvent (Pipe Connected)	PipeName

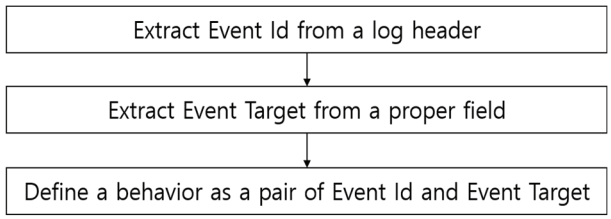

Fig. 2. Flowchart for extracting behavior from the logs.

First, the Event Id is extracted from the Event Id field in a log header. Then, Event Target is extracted from a proper field according to the Event Id by looking at Table 1. Lastly, a behavior is defined by combining the Event Id extracted from step 1 and the Event Target extracted from step 2.

In the case of Fig. 1, the Event Id is 2 which means “File creation time changed” and the Event Target is “C:\Users\jwj\...\global.temp.ini” in a “TargetFilename” field. If there is a path in Event Target, only the last word is extracted. Then, the behavior of Fig. 1 is <2, global.temp.ini>; which means “performed Event Id 2 to global.temp.ini”.

We extracted and analyzed the 13 types of behaviors from logs obtained by executing variants and confirmed that most of the behaviors were the same. However, some behaviors did not match because the Event Target was an arbitrary name or new behaviors were added to make them look different from existing malware. We need to exclude these behaviors which are unnecessary to detect variants.

To exclude behaviors unnecessary for detecting variants, we extract common behaviors from a variants group; that is, a collection of variants derived from a single malware. A common behavior is a behavior that all malware in a variants group perform, specifically where the Event Id and the Event Target match. If the Event Target is an arbitrary name, then it is unlikely to be equal within variants. If new behaviors are added to the variants, they are not extracted because other variants in a group do not perform them. We assume that every variant derived from a specific malware performs common behaviors because the techniques of making variants do not change their behaviors. Therefore, we can detect the unknown variants with the common behaviors. For further use, the common behaviors extracted from the variants group are stored in sets according to the Event Id.

Fig. 3 is an example of extracting the common behaviors from a variants group. There are three variants in a group and they perform three behaviors for each group. Since <1, A> and <7, B> are in common, these behaviors are extracted as common behaviors and stored into different set according to their Event Id. In the case of Event Id 11, the behaviors are not extracted since the Event Targets do not match. Because these common behaviors are also performed by the unknown variants, we can detect them by querying these common behaviors from logs.

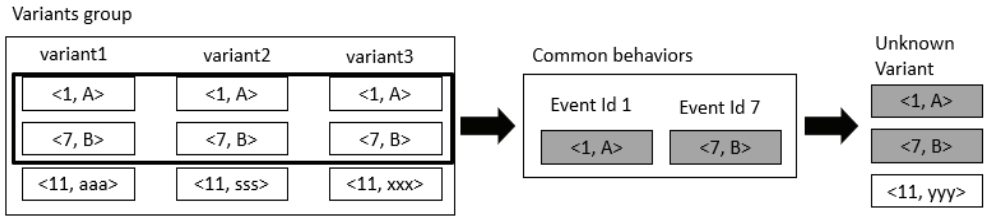

Fig. 3. Example of extracting common behaviors.

3.2 Method to Detect Malware Variants from Logs

Every common behavior extracted from a variants group can identify unknown variants; however, it can also identify normal programs that are false positive since all behaviors that malware perform can be also performed by normal programs. It is necessary to query several common behaviors to reduce the false positives; however, a massive number of logs will be returned since the behaviors of normal programs are also recorded in the logs. Therefore, we should consider the cost and accuracy of detecting malware variants using logs.

To detect variants using logs, we need to define characteristic behaviors that return few logs and low false positives. Some behaviors are so common that many programs, including normal programs, perform them resulting in large logs and high false positives. On the other hand, some behaviors are so unique that only programs with a specific purpose perform them resulting in returning few logs and low false positives. Therefore, we must define the characteristic behaviors that have high identifiability among the common behaviors.

We define a characteristic behavior as a behavior with the lowest occurrence frequency. A behavior with the lowest frequency means that it is not commonly performed, and thus, querying the behavior will return few logs. By obtaining the occurrence frequency from normal programs, we can expect that the lower the frequency, the less the normal programs will perform that behavior, which means low false positives. Therefore, characteristic behaviors can detect variants with few returned logs and low false positives.

To detect variants from the logs, the characteristic behaviors should be extracted from the common behaviors and then queried. The process of detecting variants can be divided into two steps.

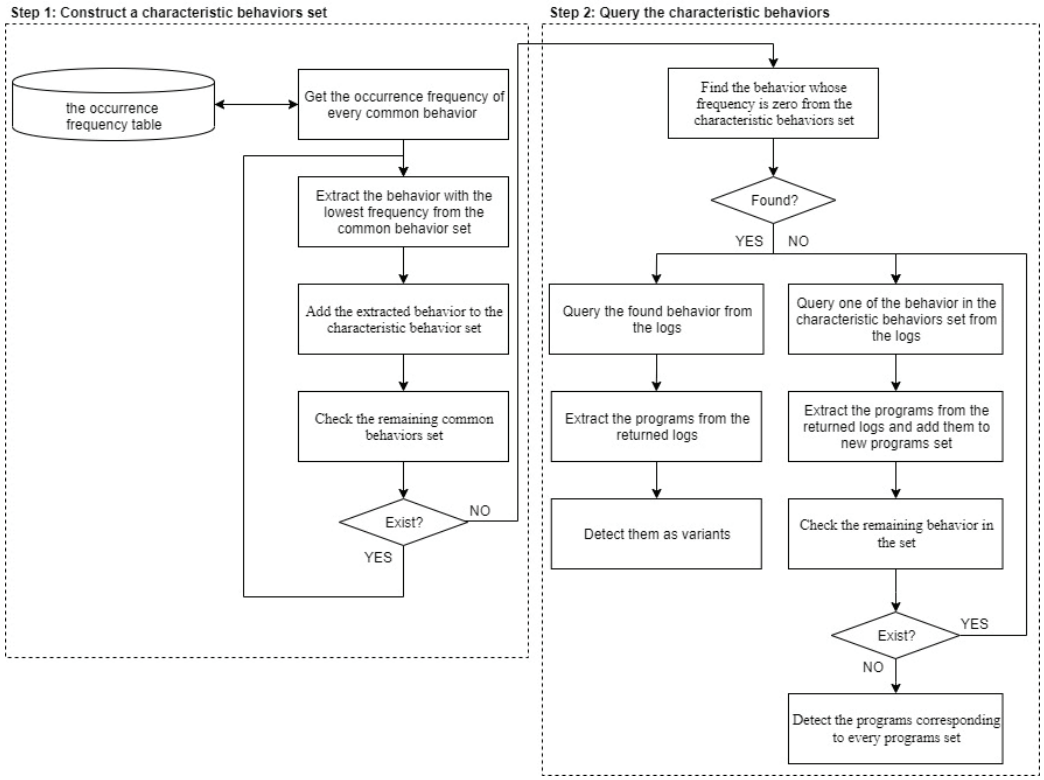

Fig. 4. Flowchart for detecting malware variants from the logs.

Step 1: Construct the characteristic behavior set

A single characteristic behavior is extracted from each common behavior set which exists for each Event Id. Since variants perform the same Event Ids as their original malware, a program that performs every characteristic behavior extracted from each common behavior set is highly likely to be a variant. The extracted behaviors are added to the characteristic behaviors set.

Step 2: Query the characteristic behaviors

After constructing the characteristic behaviors set, behaviors to be queried are extracted from the set. If there is a characteristic behavior whose frequency is zero in the set, only that behavior is queried since it is so unique that it could, alone, be an identifier for unknown variants. If there are no such behavior, all characteristic behaviors are queried to reduce the false positives. For each query, the programs are extracted from returned logs, and the extracted programs are added to a new programs set. The programs corresponding to every new programs set are detected as variants.

The flowchart for detecting variants is as shown in Fig. 4. We assume that the occurrence frequency table is given by executing arbitrary normal programs and counting the frequency of behaviors.

Fig. 5 is an example of extracting characteristic behaviors. The number to the right of the behavior is the occurrence frequency obtained by executing the arbitrary normal programs. In variants group 1, <1, a>, <7, i>, and <10, y> are added to the characteristic behaviors set. Furthermore, since the frequency of <10, y> is zero, only this behavior is extracted as a characteristic behavior. In variants group 2, <1, d>, <7, l>, and <10, o> are added to the characteristic behaviors set. Since there is no behavior whose frequency is zero, three characteristic behaviors will be used for detecting variants.

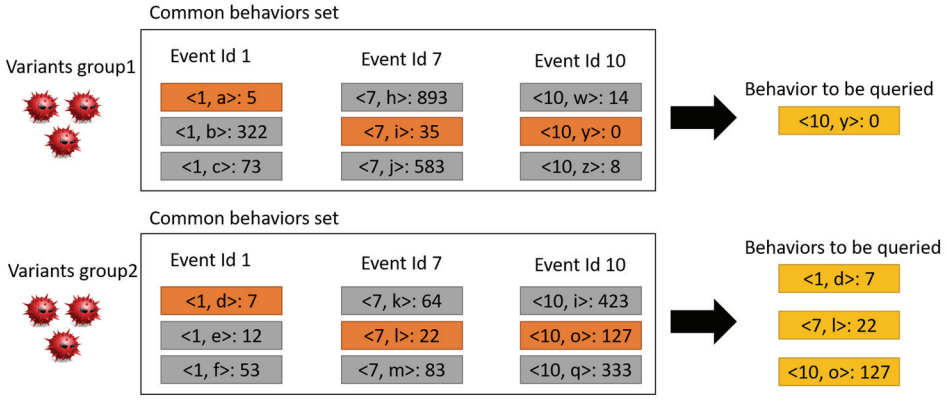

Fig. 5. Example of choosing characteristic behaviors.

4. Experiment

In the previous section, we assumed that malware variants perform common behaviors. In order to prove the existence of common behaviors that every variant performs, we extract the common behaviors in every possible situation to show that they are similar regardless of the number of variants in a group. In summary, we compare our method with previous studies. Since our proposed method focuses on detecting variants in the host, the compared method should be also available for the host. Since behavior analysis is limited in the host, we use the static analysis tool, ssdeep, that calculates the similarity of two files by dividing areas of a program and turning them into hash.

To demonstrate that the characteristic behaviors can detect variants with few returned logs and low false positives, we query the characteristic behaviors from logs and confirm how many logs are returned and how many variants are detected.

We experiment with six variants groups to extract common behaviors and with 1,151 normal programs to obtain the occurrence frequency. In addition, we confirm the result of querying characteristic behaviors

from the 1,584,363 logs generated by executing test samples consisting of 6,430 malwares by the Korea Internet & Security Agency. To store such a big log, we use the NoSQL database Elasticsearch, which facilitates fast data access using hash tables.

4.1 Extracting Common Behaviors from Variants Groups

Case 1: Three variant group

Table 2 shows behaviors of three variants (P1, P2, and P3) for six Event Ids. The behaviors that the variants performed are in the second column and the common behaviors are in the third column. In the case of the third column, we divided situations according to the number of selected variants from the group in order to confirm that common behaviors are similar in any situations. There are three situations when two variants are selected, three variants are selected, and four variants are selected.

To confirm that common behaviors are similar in any situation, we extract common behaviors in all possible situations. There are three cases when two variants are selected and one case when three variants are selected. Four variants cannot be selected since this group only has three variants. If there is a difference in each case, the number of common behaviors would be different. Since common behaviors extracted from all possible cases are the same, we confirm that common behaviors are the same regardless of the number of selected variants.

Table 2. Behaviors of three variants

Event Id	Number of behaviors			Number of common behaviors		
	P1	P2	P3	2 variants	3 variants	4 variants
1	2	2	2	1	1	-
7	32	32	32	32	32	-
10	29	29	29	29	29	-
11	4	4	4	4	4	-
12	1	1	1	1	1	-
13	1	1	1	1	1	-

Case 2: Four variant group A

Table 3 shows the behaviors of four variants (P1, P2, P3, and P4) for three Event Ids. It appears that all variants perform the same behaviors; however, in the case of Event Id 11, no common behavior is extracted. This is because the Event Target was not matched in any situation owing to the arbitrary name of the Event Target.

To confirm that common behaviors are similar in any situation, we extract common behaviors in all possible situations. There are six cases when two variants are selected, four cases when three variants are selected, and one case when four variants are selected out of four variants. We confirm common behaviors are the same regardless of the number of selected variants.

Table 3. Behaviors of four variants (group A)

Event Id	Number of behaviors				Number of common behaviors		
	P1	P2	P3	P4	2 variants	3 variants	4 variants
7	54	54	54	54	54	54	54
11	1	1	1	1	0	0	0
12	3	3	3	3	3	3	3

Case 3: Four variant group B

Table 4 shows the behaviors of four variants (P1, P2, P3, and P4) for four Event Ids. In the case of Event Id 1 and 10, all four variants perform the same behavior. In the case of Event Id 7, P1 performed one additional behavior as compared to the other variants, but this is excluded because other variants did not perform the behavior. In the case of Event Id 11, all variants performed two behaviors; however, since the Event Target is an arbitrary name, the behaviors are excluded.

To confirm that common behaviors are similar in any situation, we extract common behaviors in all possible situations. There are six cases when two variants are selected, four cases when three variants are selected and one case when four variants are selected from four variants. We confirm that in all cases, common behaviors are the same regardless of the number of selected variants.

Table 4. Behaviors of four variants (group B)

Event Id	Number of behaviors				Number of common behaviors		
	P1	P2	P3	P4	2 variants	3 variants	4 variants
1	1	1	1	1	1	1	1
7	38	37	37	37	37	37	37
10	1	1	1	1	1	1	1
11	2	2	2	2	0	0	0

Case 4: Four variant group C

Table 5 shows the behaviors of four variants (P1, P2, P3, and P4) for six Event Ids. Similar to case 1, all variants in the group performed exactly the same behaviors.

To confirm that common behaviors are similar in any situation, we extract common behaviors in all possible situations. There are six cases when two variants are selected, four cases when three variants are selected, and one case when four variants are selected out of four variants. We confirm common behaviors are the same regardless of the number of selected variants.

Table 5. Behaviors of four variants (group C)

Event Id	Number of behaviors				Number of common behaviors		
	P1	P2	P3	P4	2 variants	3 variants	4 variants
1	1	1	1	1	1	1	1
7	44	44	44	44	44	44	44
10	1	1	1	1	1	1	1
11	1	1	1	1	1	1	1
12	12	12	12	12	12	12	12
13	6	6	6	6	6	6	6

Case 5: Five variant group

Table 6 shows the behaviors of five variants (P1, P2, P3, P4, and P5) for six Event Ids. In the case of Event Id 1, all variants perform more than three behaviors, but since only one behavior's Event Target is matched, only one common behavior is extracted. In the case of Event Id 2, P5 performed two additional behaviors, but this is excluded because other variants do not perform the behaviors. In the case of Event Id 10, the numbers of behaviors are 26, 26, 26, 27, and 25. Such diversity can influence extracting common behaviors. For example, if P5 is included in a variants group, the possible number of common

behaviors is 25. However, if P5 is excluded, it is 26. In case of Event Ids 11, 12 and 13, P5 seems not to be a variant since P5 performed more behaviors than the others. However, when common behaviors are extracted, most of the behaviors are similar.

To confirm that common behaviors are similar in any situation, we extract common behaviors in all possible situations. There are ten cases when two variants are selected, ten cases when three variants are selected and five cases when four variants are selected out of five variants. In the case Event Ids 10 and 12, there are differences in common behaviors when two and three variants are selected; however, the added behaviors are not selected as characteristic behaviors.

Table 6. Behaviors of five variants

Event Id	Number of behaviors					Number of common behaviors		
	P1	P2	P3	P4	P5	2 variants	3 variants	4 variants
1	4	4	3	4	5	1	1	1
2	133	133	133	133	135	132	132	132
7	55	55	55	55	55	55	55	55
10	26	26	26	27	25	25, 26	25, 26	25, 26
11	117	117	117	117	121	110	110	110
12	6	7	7	7	14	6, 7	6, 7	6
13	12	12	12	12	13	12	12	12

Case 6: Six variant group

Table 7 shows the behaviors of five variants (P1, P2, P3, P4, and P5) for five Event Ids. With the exception of Event Id 7, all variants performed the same behaviors. In the case of Event Id 7, the number of behaviors is 94, 92, 94, 94, 92, and 92. Such diversity can influence extracting common behaviors. For example, if P1, P3, and P4 are included in a group, the possible number of common behaviors is 94. However, if the other variants are included in a group, the possible number of common behaviors is 92.

To confirm that common behaviors are similar in any situations, we extract common behaviors in all possible situations. There are fifteen cases when two variants are selected, twenty cases when three variants are selected, and fifteen cases when four variants are selected out of six variants. In the case Event Id 7, there are differences in common behaviors when two and three variants are selected. However, the added behaviors are not chosen as characteristic behaviors.

Table 7. Behaviors of six variants

Event Id	Number of behaviors						Number of common behaviors		
	P1	P2	P3	P4	P5	P6	2 variants	3 variants	4 variants
7	94	92	94	94	92	92	92, 94	92, 94	92
10	1	1	1	1	1	1	1	1	1
11	1	1	1	1	1	1	1	1	1
12	9	9	9	9	9	9	9	9	9
13	3	3	3	3	3	3	3	3	3

Summary of extracting common behaviors

We demonstrated the process of extracting common behaviors. Even if there are differences in common behaviors in cases 5 and 6, the differences were negligible. In addition, we confirmed that the common

behaviors were almost the same regardless of the number of variants in a group, which proves the existence of common behaviors that every variant is derived from a specific malware performs.

Table 8 is the result of calculating the similarity with ssdeep. Since ssdeep calculates the similarity between two files, we make a similarity matrix by comparing two variants in the group in all possible combinations. For example, in case 1, there are three variants (P1, P2, and P3) and the first row is the similarity from P1 to P1, P2, and P3. Note that the similarities under the principal diagonal are skipped. In case 3, the similarities from P1 to P2, P3, and P4 are 100%, 0%, and 0%, respectively. Even if the similarities seem to be diverse, ssdeep is still effective at detecting variants as even a 1% similarity can be considered a variant. However, in case 3, there is 0% similarity which signifies false negatives.

This can be affirmed because this is the limitation of static analysis when it comes it malware variants that modify the execution file. On the other hand, we confirmed that the variants in the variants groups have almost all the same behaviors, which means that behavior analysis has more accuracy than static analysis.

Table 8. ssdeep similarity

	Number of variants	Similarity matrix
Case 1	3	$\begin{pmatrix} & 55 & 55 \\ & & 100 \end{pmatrix}$
Case 2	4	$\begin{pmatrix} & 93 & 93 & 93 \\ & & 93 & 100 \\ & & & 93 \end{pmatrix}$
Case 3	4	$\begin{pmatrix} & 100 & 0 & 0 \\ & & 0 & 0 \\ & & & 0 \end{pmatrix}$
Case 4	4	$\begin{pmatrix} & 100 & 99 & 100 \\ & & 99 & 100 \\ & & & 99 \end{pmatrix}$
Case 5	5	$\begin{pmatrix} & 71 & 69 & 69 & 69 \\ & & 69 & 71 & 69 \\ & & & 72 & 68 \\ & & & & 72 \end{pmatrix}$
Case 6	6	$\begin{pmatrix} & 91 & 91 & 88 & 88 & 88 \\ & & 90 & 86 & 86 & 85 \\ & & & 90 & 90 & 88 \\ & & & & 100 & 86 \\ & & & & & 86 \end{pmatrix}$

4.2 Querying Characteristic Behaviors from the Logs

Table 9 shows the results of querying characteristic behaviors that are extracted from the common behaviors for each of the six cases. The first column is a case number and each row corresponds to case. The second column is the number of variants in a group. The third column is the number of common behaviors extracted by all variants in each group. The fourth column is the characteristic behaviors that are chosen among the common behaviors. If there is a common behavior with at frequency of zero, only

that behavior is chosen as a characteristic behavior. If not, a single characteristic behavior is extracted for each Event Id. The fifth column is the number of returned logs queried by the characteristic behaviors from the 1,584,363 logs.

In all cases except for case 2, there are common behaviors where the frequency is zero, so only one characteristic behavior is extracted. The reason why the frequency could be zero is that the behaviors are about creation event. Event Id 1 is "Child process create," Event Id 11 is "Fil create" and Event Id 12 is "Registry create." These events can have an arbitrary Event Target resulting in unique behaviors, so, the frequency could be zero. In case 2, since there is no such behavior, a single characteristic behavior is extracted from every Event Id. <7, wsock32.dll> and <12, Parameters> are extracted as characteristic behaviors in case 2, and these behaviors are usually performed by other programs resulting in a lot of logs.

We extract the program names for each Event Id from the returned logs and aggregate the extracted names. Only the names that are extracted from every Event Id are regarded as programs identified by characteristic behaviors. We then check whether the identified programs are variants or false positives. The number of false positives is in the sixth column and the number of variants is in the last column so that we can confirm whether characteristic behaviors can detect variants with a low false positive.

Table 9. Summary of six cases

	Number of variants	Number of common behaviors	Characteristic behaviors	Number of returned logs	Number of false positives	Number of variants
Case 1	3	68	<12, winsvc>	0	0	0
Case 2	4	57	<7, wsock32.dll>	4,729	31	0
			<12, Parameters>	16,992		
Case 3	4	39	<1, wmic.exe>	243	47	4
Case 4	4	65	<11, budha.exe>	5	3	2
Case 5	5	342	<12, 39013>	0	0	0
Case 6	6	106	<11, HELP>	52	0	18

We confirm that variants are detected in cases 3, 4, and 6, but there are false positives in cases 2, 3, and 4. The returned logs queried by two characteristic behaviors in case 2 are 21,721, but the total number of identified programs is only 31. This is because it is difficult to perform two characteristic behaviors if it is not a variant. If there were more characteristic behaviors in case 2, the number of false positives would decrease. In case 3, even if the characteristic behaviors <1, wmic.exe>'s frequency is zero, the number of false positives is 47. This means that it is rare for normal programs to perform <1, wmic.exe>, but it is common for malwares because the frequency is obtained only by the normal programs. If the frequency is also obtained by malwares, the other behaviors could be chosen as a characteristic behavior. In case 6, the characteristic behavior <11, HELP> is able to detect variants completely with no false positives. This means that <11, HELP> could be a unique identifier for variants that other programs do not perform.

The preceding experiments confirm that we could utilize characteristic behaviors to detect variants. It is encouraging that there is an identifier that could detect a variant with a low false alarm rate and low cost. It is discouraging; however, that the total number of detected variants through all cases is only 24. It seems very small considering that we have queried from the logs generated by as many as 6,430

malwares. Note that we use just six cases of variants groups to explain our detection mechanism. By using more variants groups, we could detect more variants.

We confirm that characteristic behaviors that we define may generate false positives. If we query more behaviors, we could reduce the false positives. However, since the more behaviors that are queried, the more logs are returned, we have to devise a way to find the characteristic behaviors with few returned logs and low false positives.

5. Conclusion

In this study, we proposed a method to detect malware variants in the host through the logs taking into consideration that a lot of normal programs are running in the host. To query the behaviors of programs using the logs, we define a behavior with the minimum data and extract the common behaviors that every variant in a group performs to exclude unnecessary behaviors for detecting variants. More importantly, to reduce the query cost and false positives, we define characteristic behaviors as behaviors with the lowest occurrence frequencies obtained by executing arbitrary normal programs. The lowest occurrence frequency means that it is not common, and thus, querying the behavior will return a limited number of logs. In addition, because we obtain the frequency from arbitrary normal programs, normal programs do not perform that behavior in a normal situation, and thus, querying the behavior will return a limited number of normal programs, namely low false positives. Therefore, the characteristic behaviors could detect variants from logs with few returned logs and low false positives.

Our experiments prove the existence of common behaviors that every variant performs by extracting the common behaviors in all possible situations. Even if there were differences in the common behaviors, the differences seem negligible. We demonstrated that characteristic behaviors could detect variants from logs with few returned logs and low false positives. However, we need to perform the experiments with more variants group since the six variants groups are not sufficient to support the validity of our method.

We confirmed that there could be false positives in some cases when querying characteristic behaviors. Even if these false positives could be removed by querying more behaviors, we cannot query many behaviors when considering the query cost. Due to this trade-off relationship between cost and accuracy, we should query only the meaningful behaviors. Since there are many ways to define the characteristic behaviors, a further study focusing on how to find the characteristic behaviors need to be done.

Acknowledgement

This work was supported by Institute for Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No. 2019-0-01343, Training Key Talents in Industrial Convergence Security).

References

[1] Y. Lee, “Hacking into PyeongChang Winter Olympics: a long time ago carefully prepared APT, system destruction attack,” 2018 [Online]. Available: https://byline.network/2018/05/3-13/.

[2] G. Andy, "The Untold Story of the 2018 Olympics Cyberattack, the Most Deceptive Hack in History," 2019 [Online]. Available: https://www.wired.com/story/untold-story-2018-olympics-destroyer-cyberattack/.

[3] S. Y. Choi, C. G. Lim, and Y. M. Kim, "Automated link tracing for classification of malicious websites in malware distribution networks," *Journal of Information Processing Systems*, vol. 15, no. 1, pp. 100-115, 2019.

[4] H. Arshad, A. B. Jantan, and O. I. Abiodun, "Digital forensics: review of issues in scientific validation of digital evidence," *Journal of Information Processing Systems*, vol. 14, no. 2, pp. 346-376, 2018.

[5] A. Souri and R. Hosseini, "A state-of-the-art survey of malware detection approaches using data mining techniques," *Human-centric Computing and Information Sciences*, vol. 8, article no. 3, 2018. https://doi.org/10.1186/s13673-018-0125-x

[6] Korea Internet & Security Agency, "Research for malware mutants group identification based on data mining," 2017 [Online]. Available: https://www.kisa.or.kr/public/library/report_View.jsp?regno=022709&searchType=&searchKeyword=&pageIndex=1.

[7] AV-TEST Institute, "Latest malware statistics and trends report," 2021 [Online]. Available: https://www.av-test.org/en/statistics/malware/.

[8] T. G. Kim and E. G. Im, "Code Reuse Analysis Techniques for Detection of Malware Variant," *Journal of the Korea Institute of Information Security and Cryptology*, vol. 24, no. 1, pp. 32-38, 2014.

[9] J. Zhang, K. Zhang, Z. Qin, H. Yin, and Q. Wu, "Sensitive system calls based packed malware variants detection using principal component initialized multilayers neural networks," *Cybersecurity*, vol. 1, article no. 10, 2018. https://doi.org/10.1186/s42400-018-0010-y

[10] S. B. Park, M. S. Kim, and B. N. Noh, "Detection method using common features of malware variants generated by automated tools," *Journal of Korean Institute of Information Technology*, vol. 10, no. 9, pp. 67-75, 2012.

[11] D. Moon, H. Lee, and I. Kim, "Host based feature description method for detecting APT attack," *Journal of the Korea Institute of Information Security & Cryptology*, vol. 24, no. 5, pp. 839-850, 2014.

[12] S. Kang, S. Kim, M. Park, and J. Kim, "Study on windows event log-based corporate security audit and malware detection," *Journal of the Korea Institute of Information Security & Cryptology*, vol. 28, no. 3, pp. 591-603, 2018.

[13] Microsoft, "Sysmon v13.23," 2021 [Online]. Available: https://docs.microsoft.com/en-us/sysinternals/downloads/sysmon.

Woo-Jin Joe **https://orcid.org/0000-0002-7374-8955**

He received B.S. degrees in School of Computer Science and Engineering from Chungnam National University in 2019. Since March 2019, he is with the School of Computer Science and Engineering from Chungnam National University as a M.S. His research interests lie on the big data and security.

Hyong-Shik Kim **https://orcid.org/0000-0001-5917-9541**

He received the B.S.E., M.S.E, and the Ph.D. in computer engineering from Seoul National University, Korea in 1988, 1990, and 1997, respectively. He joined Chungnam National University, Korea as a faculty member in 1999, and is currently a professor at the department of computer science and engineering. His research interests lie on the borderlands between computer system architecture and security.