ISSN 1330-3651 (Print), ISSN 1848-6339 (Online)
https://doi.org/10.17559/TV-20210820065715
Original scientific paper

Malware Visualization and Similarity via Tracking Binary Execution Path

Jihun KIM, Sungwon LEE, Doosan CHO*, Jonghee YOUN*

Abstract: Today, computer systems are widely and importantly used throughout society, and malicious codes to take over the system and perform malicious actions are continuously being created and developed. These malicious codes are sometimes found in new forms, but in many cases they are modified from existing malicious codes. Since there are too many threatening malicious codes that are being continuously generated for human analysis, various studies to efficiently detect, classify, and analyze are essential. There are two main ways to analyze malicious code. First, static analysis is a technique to identify malicious behaviors by analyzing the structure of malicious codes or specific binary patterns at the code level. The second is a dynamic analysis technique that uses virtualization tools to build an environment in a virtual machine and executes malicious code to analyze malicious behavior. The method used to analyze malicious codes in this paper is a static analysis technique. Although there is a lot of information that can be obtained from dynamic analysis, there is a disadvantage that it can be analyzed normally only when the environment in which each malicious code is executed is matched. However, since the method proposed in this paper tracks and analyzes the execution stream of the code, static analysis is performed, but the effect of dynamic analysis can be expected.The core idea of this paper is to express the malicious code as a 25 × 25 pixel image using 25 API categories selected. The interaction and frequency of the API is made into a 25 × 25 pixel image based on a matrix using RGB values. When analyzing the malicious code, the Euclidean distance algorithm is applied to the generated image to measure the color similarity, and the similarity of the mutual malicious behavior is calculated based on the final Euclidean distance value. As a result, as a result of comparing the similarity calculated by the proposed method with the similarity calculated by the existing similarity calculation method, the similarity was calculated to be 5-10% higher on average. The method proposed in this study spends a lot of time deriving results because it analyzes, visualizes, and calculates the similarity of the visualized sample. Therefore, it takes a lot of time to analyze a huge number of malicious codes. A large amount of malware can be analyzed through follow-up studies, and improvements are needed to study the accuracy according to the size of the data set.

Keywords: binary analysis; malicious codes; malware; similarity; visualization

1 INTRODUCTION

Recently, following the rapid development of computing technology and networks, Information and Communications Technology (ICT) is widely utilized in society and industries in general. At the same time, technologies that threaten information security are developing, and the most prominent example is security threats caused by malware. In recent years, the volume of malware has also been increasing rapidly. In addition, malware attacks are also becoming more diverse, and malware variants, which have been fabricated by modifying existing malware, are also showing explosive growth rates and are posing various security threats such as data loss, personal and financial information leakage, system damage, and Information Technology (IT) infrastructure destruction. Such high growth rates of malware and malware variants cause increases in malware analysts' efforts and analysis time leading to serious social costs. To cope with the foregoing problem, efficient analysis methods and studies of malware are required.

Malware analyzing methods are implemented through largely two mechanisms. First, static analysis is a technique to identify malicious behaviors by analyzing the structure or certain binary patterns of malware at the code level [1], which enables more in-depth and detailed analysis but will require much time and effort and add considerable difficulties to analysis if technologies to obstruct static analysis such as execution file compression and code obfuscation are applied to malware [2]. Another analysis method is dynamic analysis, which is executing malware in a virtual machine to analyze the malicious behavior [3]. This method has an advantage of enabling clear understanding of malicious behaviors even when execution file compression or code obfuscation has been applied to malware because malware is actually executed for analysis. However, this method is not suitable for analyzing trigger based malware that runs at a certain time or when the user's certain action is taken.

The method used for malicious code analysis in this paper is a static analysis technique. There is a lot of information that can be obtained through dynamic analysis, but there is a disadvantage that analysis is possible only when the environment in which each malicious code is executed matches. However, since the method proposed in this paper traces and analyzes the execution flow of the code, static analysis is performed, but the effect of dynamic analysis can be expected.

In addition, in the present study, malicious behaviors are analyzed through the Application Programming Interface (API) collected during analysis, and malware is made into images based on the frequencies and interactions of the APIs. Since the acts of malware imaging as such can be visually analyzed and malware variants and similar pieces of malware can be easily analyzed through comparison between different pieces of malware, in the present study, the similarities of malware images are also calculated.

2 RELATED WORKS

Currently, various studies related to malicious code analysis are being conducted, and there are many studies using both dynamic analysis and static analysis methods. Various studies are in progress, such as using a method that combines dynamic and static analysis to analyze, or how to bring advantages and improve disadvantages of each. Various studies have emerged according to the high necessity, and through the comprehensively grasped and organized thesis, it is possible to grasp the overall understanding of the methods of malicious code analysis and methods for detection [4-6].

Previous binary code based static analysis studies have been conducted based on signatures made by extracting internal attributes and features centering on code bases [7].

Such studies analyzed malware by applying statistical mechanisms based on collected information. The most representative methods extract strings in malware files [8] or extract only opcodes from disassembled files and compile statistics to analyze the features of the file [9-10]. Although such methods can access and identify file structures because they simply collect and list the signatures of files, they have a limitation; the behaviors of malware can be hardly identified with them [11]. Nevertheless, studies that adopted analysis methods based on file signature based collection methods as such have been increasing because methods to analyze only the features of files are more excellent in time efficiency than code based methods to analyze the inside because of the massification of malware. Therefore, methods that use the n-gram technique that cuts and processes information such as opcodes, strings, and bytecodes based on a certain criterion [12] or list the byte sequences of binary codes [13] were proposed. These methods have been developed so that they can be defined as intrinsic DNAs of files [14], and similarities between different pieces of malware are calculated based on them. As mentioned earlier, the disadvantage of these methods is that they have some limitations in understanding malicious behaviors. To solve this problem, a method to identify malicious behaviors through the extraction of those APIs that are used by malware has been presented [15] because a ground that becomes a clear and certain base in malicious behavior judgment is the discovery of the function that is used by malware. Although early studies that collected APIs to judge malicious behaviors listed API sequences, collected the log information regarding the use of APIs [16], or used the frequencies of APIs to judge malicious behaviors, recently, mathematical algorithms such as the nearest neighbour search algorithm [17] or the longest common subsequence (LCS) [18] are applied to increase the speed and enhance the accuracy. Many studies have also proposed various methods for identifying malicious behaviors and one example of which is visualization [19]. Basically, the statistical values or opcodes of the APIs called in the binary codes are shown as image or [20] in the form of file map images. In addition, dynamic analyses are sometimes utilized to express the statistics of the APIs used as thread maps [21] or extract and visualize the entropy of the files [22]. Such studies for visualization can more clearly identify malicious behaviors, and 2D based cross-sectional images are also useful for verifying the similarities of malware variants and families [23]. The similarity calculations as such can be applied with qualified algorithms such as Cosine similarity algorithm [24] and Jaccard distance algorithm [25] to calculate high similarities. However, such methods show classification rates that vary with how data are processed and used and show low performance speed because they have relatively large amounts of calculations. In addition, since they should calculate large amounts of data, they require large amounts of necessary information and their calculations may become complicated depending on the algorithm used. In the present study, the run stream will be searched to improve the efficiency of the speed to identify malicious behaviors based on the API information collected and the foregoing will be developed for visualization. In addition, similarities will be calculated to analyze the similarities of malware variants and malware families in order to complement the limitations of existing studies and maximize the efficiency.

3 PROPOSED METHOD

The present study adopts static analysis that analyzes malware at the code level as a basic mechanism. The IDA is a typical reverse engineering tool used by analysts who want to analyze binary files. The present study also uses malware disassembled through the IDA as analysis data. Previous analysts simply analyzed binary files using reverse engineering skills. However, the disassembled malware is so long that the analysis takes tremendous amounts of effort and time. Therefore, binary automation tools using taint analysis [26] or symbolic execution [27] have been released recently and they are used to find vulnerabilities or bugs in files. In the present study too, malware is analyzed using path searches such as taint analysis and symbolic execution. The two methods introduced are common in injecting variables to observe run streams in path searches, but whereas taint analysis cannot observe one execution path, symbolic execution can search all paths.

The code level analysis process in the present study follows a mechanism very similar to symbolic execution. However, the analysis in the present study does not inject any unknown quantity (symbol) in the process of observing the stream but does observes the entire stream to examine the entire stream of the malware being analyzed.

In addition, the frequencies and interacting relationships of the APIs collected are analyzed to use the APIs for imaging. The relevant API based images enable visual identification of malware's behaviors. The final purpose in the present study is to calculate the similarities of images that express malicious behaviors to figure out the similarities of malware variants and similar malware.

3.1 Static Execution Path Exploration

In the IDA, disassembled binary files consist of instruction sets composed of loc_xxxxxx prefixes and sub-routines composed of sub_xxxxxx prefixes. To search the run stream, tracing instruction seta and sub routines according to branch instructions is important. The branch instructions out of assembly commands are executed through commands such as jz and jnz and those commands such as cmp, test, and xor, which are issued before the relevant commands are issued, divide paths into true and false ones. In the present study, paths are divided into normal, true, and false ones according to branch instructions before performing path searches. In addition, the APIs appearing during searches will be collected to organize mutual relations between the APIs to show the malicious behaviors. The contents of this section were developed by referring to previous studies. [28, 29]

The codes in Tab. 1 are binary codes for showing static execution path searches. The code used a simple code for the sake of example, and it determines the value received as an argument and branches according to the value.

If loc_4017AA4 is found to be true through the compare instruction in the seventh line, it will branch to loc_41A5DD. If loc_41A5DD is found to be true through a compare instruction, it will branch to loc_428986. The

15th to 21st lines undergo the abovementioned search process identically, and in loc_4268DE, since the subroutine sub_488930 is simply called from line 23 without any compare instruction, it is defined as a normal mark instead of true or false.

Table 1 Example binary code to display static execution path search

1	loc_417AA4 :		
2		…	…
3		mov	[esp+128Ch+var_1274], ebx
4		call	ds:CreateMutexA
5		mov	[esp+128Ch+hObject], eax
6		call	ds:GetLastError
7		cmp	eax,0b7h
8		jz	loc_41A5DD
9	loc_41A5DD:		
10		..	…
11		call	ds:CloseHandle
12		…	…
13		cmp	byte ptr [esi+4], 0
14		jz	short loc_428986
15	loc_428986:		
16		…	…
17		call	ds:EnterCriticalSection
18		cmp	word ptr [ebx+40h]
19		Mov	[esp+78h+var_64], 1
20		Jnz	loc_4268DE
21	loc_4268DE:		
22		lea	ecx, [esp+78h+var_68]
23		call	sub_428930
24	sub_488930:		
25		…	
26		call	ds:LeavecriticalSection

In the binary codes in Tab. 1, it can be seen that Windows API functions such as CreateMutexA, GetLastError, and CloseHandle are called from lines 4, 6, 11, 17, and 26, respectively. In a previous study, to analyze the interaction between the API and the API operation, a mechanism for applying normal, true, and false marks was proposed to analyze the operation of the API that is equally applied to the API.

Fig. 1 is a visualization of the API mutual actions for the binary codes in Tab. 1.

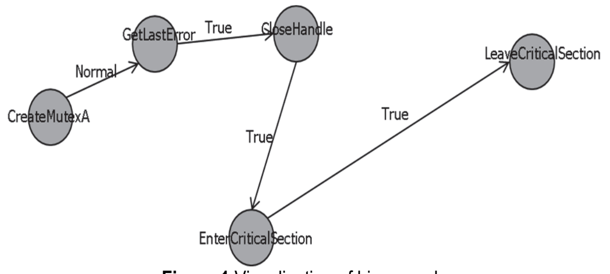

Figure 1 Visualization of binary code

3.2 API Classification

In the present study, malicious behaviors are visualized in the form of graphs expressed with nodes and edges as shown in Fig. 1. However, there is a problem that the number of APIs is too large to nodalize the APIs. The number of APIs used by Windows applications is over 20 000 based on Microsoft Developer Network (MSDN). If all the 20 000 plus some APIs are used for graph imaging, the analysis time may be too long and the readability may be degraded. In the present study, to solve such problems, the APIs will be reclassified and integrated into 25 upper categories based on the functions of the APIs so that clear judgment of actions can be expected and time efficiency can be enhanced. For example, APIs such as CreateFile and CreateProcess perform functions to "file" or "process" information and APIs such as GetSystemTime and GetLocalTime have the function to collect information on the "time" on the system. In addition, all APIs such as strcmp and strcat perform functions related to strings. Although such a classification has been already made in MSDN, not only there are many categories to which APIs belong commonly but also the categories are too abstract to understand the actions. For instance, all APIs related to processes are included in the process category but whether the relevant APIs for the creation of, deletion of, or access to processes or not is unknown. Therefore, the functions of the API were reclassified into three ones as CREATE_OR_OPEN, READ_OR_ACCESS, and CLOSE are shown. Tab. 2 shows the final 25 API categories.

Table 2 API categorization

FILE-CREATE_OR_OPEN	FILE-READ_OR_ACCESS	FILE_CLOSE
PROCESS-CREATE_OR_OPEN	PROCESS - READ_OR_ACCESS	PROCESS _CLOSE
NETWOKR - CREATE_OR_OPEN	NETWOKR - READ_OR_ACCESS	NETWOKR _CLOSE
REGEDIT-CREATE_OR_OPEN	REGEDIT - READ_OR_ACCESS	REGEDIT _CLOSE
SERVICE	STRING	DEBUGGING
RESOURCE	TIME	MUTEX
WINDOW-GUI-AND-BITMAP	SHELL-AND-CONSOLE	THREAD
STSTEM-INFORMATION	LIBRARY	HANDLE
HOOK		

However, not all APIs are reclassified into those four actions. Since APIs such as strcat and strcmp do not separately perform the function of creating or accessing to strings separately, APIs related to strings are defined as being in a single category, and APIs related to "time" such as GetLocalTime and GetSystemTime are included in the Time category because the action termed "time" has the most important value in the identification of malicious behaviors even if time information is obtained after accessing the system.

3.3 Matrixing of API Interaction

The examples mentioned in the present study are intended to understand the proposed method and are part of the entire binary codes. However, since the number of entire binary codes of actual malware is huge, the frequency of interaction of the 25 API categories could be very high.

In the present study, the mutual relations of APIs were marked as normal, true, or false using static execution path searches to analyze the actions. The frequencies in the entire binary codes in relation to the foregoing can be again made into a matrix, which will be used for the final visualization later. In the present study, there are 25 API categories, which indicate API actions, and since the actions are again marked as normal, true, or false, the frequencies were shown with three 25 × 25 matrices.

Tabs. 3 to 5 show examples to show the frequencies of the mutual actions of API categories made into matrices, which show examples of mutual calling relations ranging from 0 time to 5 times. Although only three categories, FILE-CREATE_OR_OPEN, HANDLE, and SYSTEM-INFORMATION are shown in the relevant tables, they are just to show examples, and all the 25 API categories are referred to for the final frequencies of mutual relations of APIs in the present study.

Table 3 API Frequency normal

	FILE CREATE OR OPEN	HANDLE	SYSTEM INFORMATION
FILE CREATE OR OPEN	0	0	0
HANDLE	0	0	2
SYSTEM INFORMATION	1	0	4

Table 4 API Frequency true

	FILE CREATE OR OPEN	HANDLE	SYSTEM INFORMATION
FILE CREATE OR OPEN	0	1	2
HANDLE	0	0	1
SYSTEM INFORMATION	1	0	1

Table 5 API Frequency false

	FILE CREATE OR OPEN	HANDLE	SYSTEM INFORMATION
FILE CREATE OR OPEN	0	0	0
HANDLE	0	0	0
SYSTEM INFORMATION	1	0	2

3.4 Graph Visualization

In the present study, 25 categorized API nodes and edges related to normal, true, and false marks are visualized to identify malicious behaviors. The information used for the visualization consists of 25 categorized API nodes and frequencies related to normal, true, and false marks.

Table 6 Composition of graph visualization

Node	Edge color			Color Characteristics of edges
	Normal	True	False	
25 API categories	*R*	*G*	*B*	Increase by 50 every time the frequency increases by 1

Tab. 6 shows composition of graph visualization. First, 25 nodes are fixed as absolute paths, and marks, i.e., normal, true, and false, constitute the color information of edges with *R*, *G*, and *B* colors, respectively. The The red, green, blue (*RGB*) colors consisting of 0 to 255 increase by 50 every time the frequency of the marks increases. This is because the maximum number of frequencies was identified as 5 in the mutual actions of the API categories in the present study. Of course, a larger number of frequencies may be identified. However, if the color information is close to 255, the fact that the frequency of API mutual actions is sufficiently meaningful and the increment value of 50 was derived as an appropriate value to enhance the visibility to show color information in comparison with the frequency numbers of other categories. Therefore, it can be seen that higher numbers of frequencies related to normal, true, or false marks indicate colors closer to red, green, or blue, respectively. For instance, with regard to the frequency numbers in Tabs. 3 to 5 in section 3.3, since the frequency numbers of SYSTEM-INFORMATION are Normal: 4, True: 1, and False: 2, the color information *RGB* of the edges consists of 200, 50, and 100.

Table 7 Color information according to API frequencies

	FILE CREATE OR OPEN	HANDLE	SYSTEM INFORMATION
FILE CREATE OR OPEN	0	(0, 1, 0) = *RGB*(0, 50, 0)	(0, 2, 0) = *RGB*(0, 100, 0)
HANDLE	0	0	(2, 1, 0) = *RGB*(100, 50, 0)
SYSTEM INFORMATION	(1, 1, 1) = *RGB*(50, 50, 50)	0	(4, 1, 2) = *RGB*(200, 50, 100)

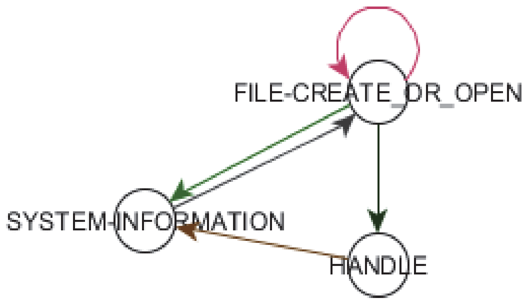

Figure 2 Imaging according to frequencies

Finally, the frequency numbers of API mutual actions in Tabs. 3 to 5 are integrated as shown in Tab. 7 and are visualized into a graph as shown in Fig. 2.

3.5 Malicious Behavior Graph Images and Pixel Images

Sections 3.1 to 3.4 are about the code analysis methods proposed in the present study. First, paths are searched with branch instructions to trace the run stream, and the interactions between APIs are analyzed utilizing the relationships between the APIs collected during the searches and the frequency numbers. These methods can be important information in analyzing malicious behaviors. In section 3.4, an example of graph imaging is shown based on the actions. In the present study, finally, the entire actions of malware are made into images and similarities are calculated based on the relevant images.

Fig. 3 shows a graph made by the imaging of the malicious behaviors of Gen:Variant.Zusy.210164, which is actual malware, according to the proposed method. Each fixed node consists of API categories and the colors of the edges connected between the nodes have *RGB* values that change the frequency numbers of mutual actions of the

APIs. The graph images as such enable malicious behavior analysis through the identification of API interactions.

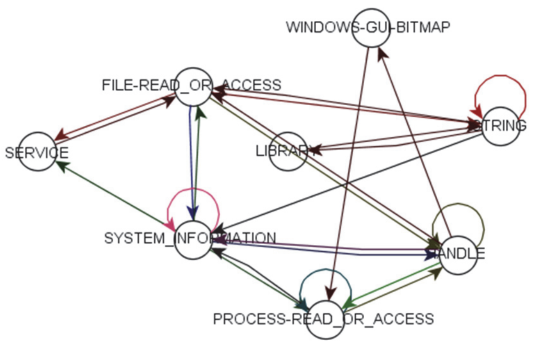

Figure 3 Gen:Variant.Zusy.210164 malicious behavior

However, the graph images as such are not suitable for calculations of similarities because the malicious behaviors in the present study are shown as API categories expressed with nodes and interactions expressed with edges and adoptability and accuracy cannot be expected from similarities calculated with the color information and the information such as the shapes of edges. This is because when color information on graph images is enlarged into pixel units, other colors than the colors of the edges appear due to distortion, etc. Therefore, in the present study, APIs' mutual actions are reconverted into simple 25 × 25 pixel images and used in the calculations of the similarities.

Fig. 4 shows the method of changing API mutual actions into 25 × 25 pixel image according to new API sum matrices made by summing up three API matrices for NORMAL, TRUE, and FALSE actions in the methods proposed in sections 3.3 and 3.4. In the present study, the similarities of malware variants and similar malware will be calculated based on the pixel images as such.

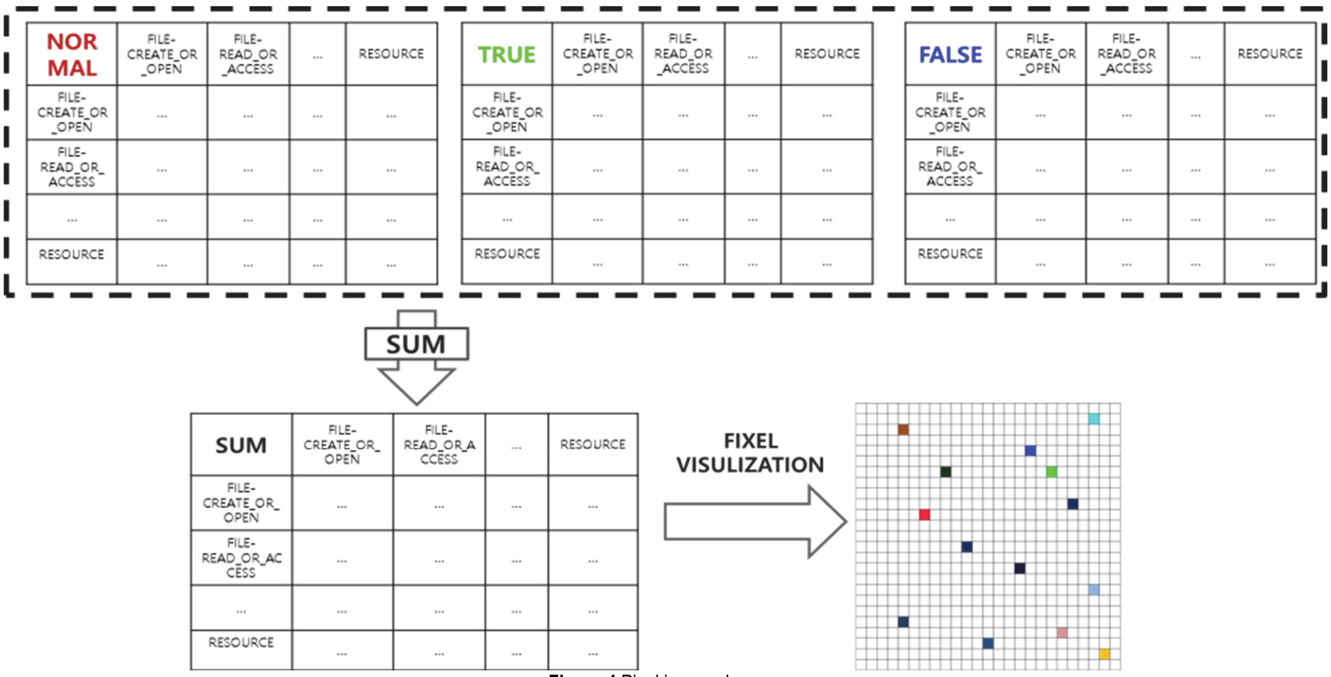

Figure 4 Pixel image changes

3.6 Pixel Image Similarity

In the present study, 25 API nodes composed of absolute paths and frequency numbers related to Normal, True, and False actions are constructed into *R*, *G*, and *B* color information, respectively, for visualization. The similarities of the malware images in which the actions were expressed as such can be compared through color comparisons between pixels.

In the present study, the similarities of the colors of all pieces of pixel information on two images being compared are calculated through the algorithm for Euclidean distances in the color space [30].

$$Color\ DISTANCE\ SIM = \\ = \sqrt{(R_2-R_1)^2+(G_2-G_1)^2+(B_2-(B_1)^2} \quad (1)$$

Fig. 5 shows 6 × 6 pixel images intended to present examples of pixel image similarity calculations and the pixels were enlarged for easy understanding. One square means one pixel.

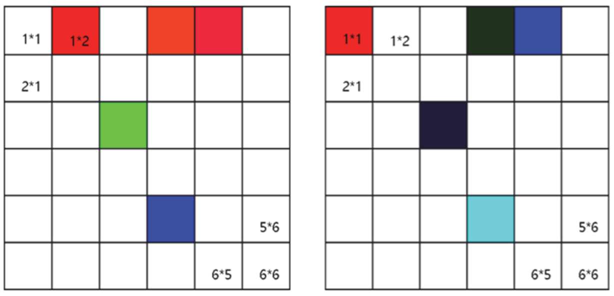

Figure 3 Test pixel images

Tab. 8 shows color information extracted from the entire pixels (36 pixels) of the two images in Fig. 5. The similarity between the color distances per pixel of these two images becomes closer to 1 as the colors become more similar and it can be seen that *RGB*: 255, 0, 0 and *RGB*: 255, 50, 0 are 99% similar colors. Therefore, if the colors of all the pixels of the two images are the same, color similarity is calculated as 1.

Table 8 Image similarity calculation

	Pixel Image 1	Pixel Image 2	Color DISTANCE (*d*)
1 × 1	255, 255, 255	255, 0, 0	0.4813
1 × 2	255, 0, 0	255, 255, 255	0.4813
1 × 3	255, 255, 255	255, 255, 255	1
1 × 4	255, 50, 0	0, 50, 0	0.6498
1 × 5	255, 0, 50	0, 0, 255	0.6138
…	…	…	…
3 × 3	0, 255, 50	0, 0, 50	0.6138
…	…	…	…
5 × 4	0, 0, 255	0, 255, 255	0.8432
…	…	…	…
6 × 6	255, 255, 255	255, 255, 255	1

This can be shown as Eq. (2) and the mechanism as such enables the calculation of the similarity between two images.

$$w*h:1=\sum_{1}^{w*h} d:sim \tag{2}$$

The entire number of pixels can be identified by multiplying the horizontal pixel w by the vertical pixel h of the image. If the similarity of the relevant pixels is 1, the images should have the same color information. The color similarity between two images can be measured by dividing the sum of d calculated using the color distance similarity algorithm by the total number of pixels. Here, the present study excludes the information of pixels with *RGB*: 0, 0, 0, i.e., the color of the white color, in both images. This is because only the color information of edges is needed from action images where malicious behaviors appear, and other information should be excluded. Since all other areas in the images except for edges are in white color, if all of the areas are used in similarity calculation, the similarities obtained may be inaccurate with too high values.

For the same reason as the reason why white color is excluded as mentioned above, nodes and nodes labels are excluded in the case of action images. Because color similarities are calculated, the colors of nodes and node labels may deteriorate the accuracy of similarities. In addition, since API nodes are composed of absolute paths, the shapes of edges are the same so that only color information is available. Therefore, the exclusion of nodes and node labels from the calculation of similarities enhances the accuracy.

4 EXPERIMENT AND DISCUSSION

4.1 Malicious Behavior Pixel Images and Similarity

In the present study, malware is made into 25 × 25 pixel images and the similarities are calculated.

In the present study, the similarities of different pieces of malware will be verified based on the similarities of the images of the relevant pieces of malware. The pieces of malware adopted for the verification of similarities are malware variants with similar malicious behaviors. The sets of malware variants classified into the same categories are called malware families. In the present study, the similarities of those pieces of malware that are in the malware families will be verified based on image based similarities.

In the present study, three pieces of malware in three families, which are Zusy, Conficker, and Deborm, will be randomly selected and the similarities will be verified.

Figs. 6 to 8 show the malicious behavior of three malware families, Zusy, Confiker, and Deborm, made into pixel images.

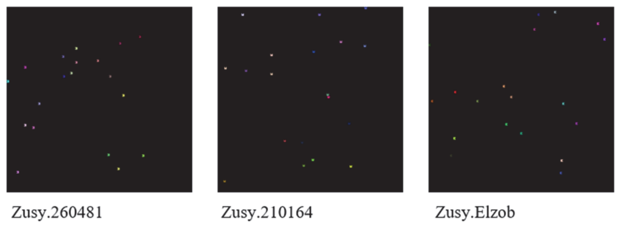

Figure 4 Gen: Variant.Zusy family visualization

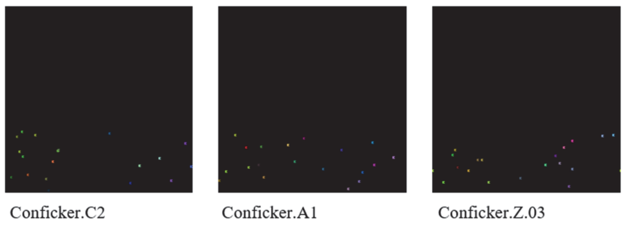

Figure 5 W32/Confiker.worm family visualization

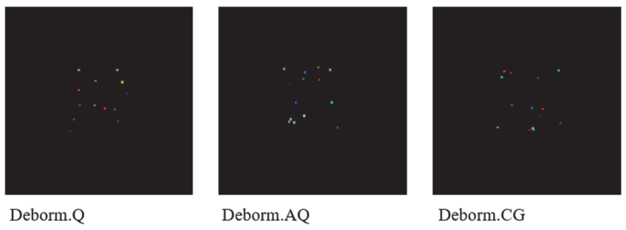

Figure 6 W32/Deborm.worm family visualization

It can be seen that Zusy uses complex APIs because the pixels are evenly distributed. However, compared to Zusy, Confiker and Deborm show intensively distributed pixels on some parts. This malicious behavior pixel image can be used as a basic data to grasp the malicious behavior through the API. Based on the malicious behavior pixel images as such, the APIs being mainly used can be identified and the results can be used as basic data to understand malicious behaviors.

Table 9 Gen: Variant.Zusy similarity

	Zusy.260481	Zusy.210164	Zusy.Elzob
Zusy.260481	1	79.1471	76.933
Zusy.210164		1	75.550
Zusy.Elzob			1

Table 10 W32/Confiker.worm similarity

	Conficker.C2	Conficker.A1	Conficker.Z.03
Conficker.C2	1	82.561	86.392
Conficker.A1		1	83.543
Conficker.Z.03			1

Table 11 W32/Deborm.worm similarity

	Deborm.Q	Deborm.AQ	Deborm.CG
Deborm.Q	1	82.561	80.183
Deborm.AQ		1	82.126
Deborm.CG			1

In the present study, the similarities of the malicious behavior pixel images shown in Figs. 6 to 8 were measured using the Color-Distance method proposed in this paper. The results are as shown in Tabs. 9 to 11.

First, with regard to Zusy, although the pixels are evenly distributed, it can be seen that the positions and colors of the pixels in three pieces of malware are shown to be similar. Therefore, the similarities were calculated to be at least 75%. In the case of Confiker and deborm, the fact that the positions the pixels indicated by the colors are intensive and the colors are similar can be seen through the calculated similarity, 82%.

4.2 Similarities of Malware Families

In the present study, the similarities of other pieces of malware than the three malware families experimented as explained in section 4.1 will be measured.

Tab. 12 lists the malware families experimented in the present study. A total of 12 malware families and 930 malware samples were experimented. In the present study, all the malware samples include Import Address Table (IAT) because binary analysis based APIs are collected to analyze malicious behaviors.

Table 12 Malware family sample data set

Type	Family	Data Set
Gen.Variant	Zusy	192
	Kazy	170
	Buzy	82
Gen:Heur	MSIL.Krypt	74
	Conjar	28
	KS	28
	Naffy	24
W32.Trojan	Graftor	149
	Clicker	24
W32.Virus	Sality	37
Win32.Worm	Allaple	75
Trojan:Downloader	Barys	47
Total	12	930

Table 13 Malware family similarity

Type	Family	Min Sim	Max Sim	Avr Sim
Gen.Variant	Zusy	67.203	78.382	73.192
	Kazy	66.645	79.128	72.961
	Buzy	77.016	84.071	74.896
Gen:Heur	MSIL.Krypt	73.772	79.837	73.689
	Conjar	70.338	89.734	76.661
	KS	73.396	85.676	75.789
	Naffy	71.903	83.845	73.874
W32.Trojan	Graftor	68.912	75.845	72.651
W32.Virus	Clicker	73.554	82.612	79.213
	Sality	65.537	88.343	72.568
Win32.Worm	Allaple	72.192	86.695	73.182
Trojan:Downloader	Barys	79.612	85.791	75.977

Tab. 13 shows the minimum, maximum, and average similarities of the malware families.

For all the families, average similarities not lower than 73% are shown. However, the growth rate of malware is currently increasing exponentially, and analysis for discrimination of new malware variants is necessary. To that end, in the present study, 10 randomly selected pieces of malware were made into pixel images and compared with the pixel images of malicious behaviors experimented earlier to experiment the detection of malware variants.

In the present study, 10 pieces of malware were randomly selected and were applied with the method proposed in this paper. Randomly selected 10 pieces of malware were finally made into pixel images and compared with the malware families experimented earlier in the present study. The results are shown in Tab. 14.

Table 14 Random malware similarity

Sample No.	Most Similarity Family	Similarity Score	Detection Result	Verifi-cation
1	Heur	73.038	Gen:Variant.FakeAlert.2	O
2	Heur	66.982	Gen:Variant.Zusy.3043	X
3	Krypt	81.098	Gen:Heur.MSIL.Krypt.2	O
4	Razy	79.778	Gen.Variant.Razy.90433	O
5	Graftor	77.391	Gen.Variant.Graftor.185658	O
6	Kazy	69.192	Backdoor.Fluxay.B	X
7	Razy	81.748	Gen.Variant.Razy.112591	O
8	Kazy	80.112	Gen:Variant.Kazy.28577	O
9	Kazy	83.799	Gen:Variant.Kazy.26444	O
10	Allaple	74.132	Win32.Worm.Allaple.Gen	O

In the table, the right end shows the detection results. In this experiment, 8 out of 10 pieces of malware were classified to coincide with their families. However, two pieces of malware were classified differently from the detection results. This is due to the fact that analysis samples for the relevant malware families were not prepared. In addition, the reason why the results of classification of these two families were shown to be Zusy and Kazy is that among the family samples experimented earlier in the present study, the numbers of samples in Zusy and Kazy families were overwhelmingly large. However, this is a limitation that can be resolved by preparing analysis samples for malware families. Although not so many family analysis samples were prepared in the present study due to difficulties in the collection of malware samples, the accuracy will be enhanced through continuous sample collection and analysis sample data construction.

4.3 Efficiency of Analysis

Many previous studies have calculated similarities between different pieces of malware to verify the similarities of malware families and variants. In the present study, three previous methods for measuring malware similarities and the method proposed in the present study will be compared in terms of analysis and time. The first similarity verification method is one that uses the Jacquard index [31], which measures the similarity between two sets. In the case of this method, the more identical the two sets are to each other, the closer to 1 the value is, and the less identical the two sets are to each other, the closer to 0 the value is. The relevant method enables finding similar objects based on known data and can become a base for resolving multiple data mining works. The second similarity verification method is the Normalized Compression Distance (NCD) [32, 33], which uses the compression algorithm to measure the similarity of two objects. Usually, compression algorithms attempt to compress the sequence to a shorter length for compression. The NCD uses the Sequence property of compression algorithms as such. That is, the NCD verifies how much the compressed length of the sequence made by combining two sequences has been shortened compared to the sum of the lengths of the two sequences compressed separately and the resultant value is closer to 0 when the two lengths are more similar. The last similarity verification method is the nearest neighbor search [34] algorithm. Since malware has been increasing very fast recently, many researchers apply machine learning algorithms to analysis for automation of analysis. The nearest neighbor search algorithm can be said to be a representative one, which is intended to find the point closest to the target object through learning using Euclidean distance calculation and linear search, etc.

In the present study, three similarity verification methods, that is, Jacquard Index, the NCD, and the nearest neighbor algorithm, are applied to the 930 samples experimented in the present study in the same system to analyze the proposed method and evaluate the time performance of the method. Here, the NCD measures similarities based on files, and the Jacquard Index and the nearest neighbor algorithm measure similarities based on codes. All the four methods including the proposed method indicate higher similarities when the values are closer to 100 and indicate the average values of similarities.

Fig. 7 is a graph of the statistics of detection rates compiled by applying the four similarity verification methods to 12 malware families.

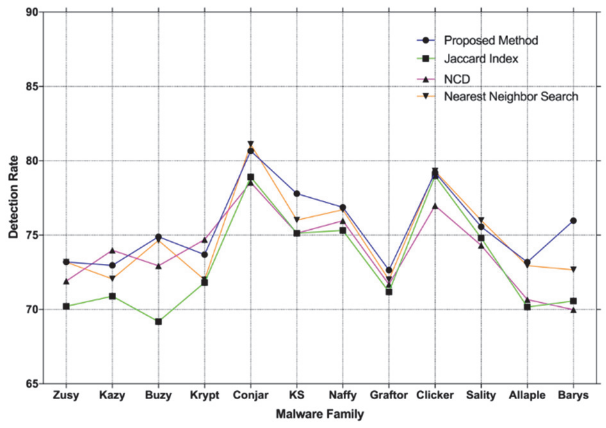

Figure 7 Detection efficiency

First, in the case of the NCD method, similarities were calculated as being in a range of 65% ~ 75% in all families. This is because the relevant method is based on the method of measuring the file based similarities excluding the internal codes. In addition, unlike other methods, the nearest neighbor search measured similarities based on codes and the similarities calculated using the relevant method were similar to the similarities calculated using the method proposed in the present study. Furthermore, the similarities measured using this method were 1% ~ 3% higher than those measured using the method proposed in the case of the Conjar and Clicker families but the accuracy cannot be easily verified in that precise similarities cannot be calculated because the relevant families consist of small numbers of samples in a range of 20 ~ 30. As can be identified in Fig. 7, the similarities of the Conjar, KS, Nafty, and Clicker families consisting of not more than 40 samples were calculated to be up to 20% higher compared to other families. On the other hand, the similarities of families with at least 100 samples such as Zusy, Kazy, and Graftor were calculated to be 70% ~ 73% and the method proposed in the present study showed higher similarities than other methods. This can highlight efficiency in automating the analysis of a large number of malware by calculating similarity to other similarity methods even for a family of large numbers of samples. This means that the method proposed in the present study will calculate similarities to be higher than other similarity verification methods even in the case of families consisting of many samples highlighting its efficiency in automation for analysis of massive malware.

5 CONCLUSION

As mentioned at the beginning of this paper, research on effective analysis methods is important because the amount of malicious code to be analyzed is too large. Therefore, in this paper, a method of visualizing and analyzing malicious code, which is a method different from the existing method, is proposed.

In the present study, malware was analyzed at the code level using static execution path searches. Although the basic analysis method is static analysis, the effect of dynamic analysis can be expected because the method proposed in the present study searches the execution paths and the mutual actions of the APIs collected during analysis and the frequency numbers are used to analyze malicious behaviors. In the present study, APIs were nodalized and the mutual actions of APIs were visualized as edges and the colors of the edges. Graphs based on the foregoing can be applied as data for clear judgment of malicious behaviors. In the present study, attempts were made to measure similarities based on the image information visualized as such. However, the images made into graphs had many constraints that can degrade the accuracy of measurement of the similarities of the colors of the edges, which represent the mutual relations of APIs. To solve this problem, the mutual actions of APIs and the frequency numbers were again made into matrices and malware was made into 25 × 25 pixel images based on the matrices. Since the 25 × 25 pixel images are composed of very intuitive images, they are useful for comparison of colors on images being compared. In the present study, the Euclidean distance algorithm was applied to measure color similarities and the similarities of mutual malicious behaviors are calculated based on the final values of Euclidean distances. Finally, the similarities calculated based on such methods were compared with the similarities calculated using existing similarity calculation methods and it was found that the similarities were calculated to be 5% ~ 10% higher on average by such methods.

Improvements and advantages of the method proposed in this paper were confirmed when compared to the existing method, but there are also disadvantages while conducting the research. The method proposed in the present study spends a lot of time in deriving resultant values because it analyzes samples, visualizes the samples, and calculates the similarities of the visualized samples. Therefore, it requires a lot of time for analysis of 10 000 or more malware samples. As a result, when performing analysis, it is necessary to divide the sample into less than 10 000 malicious code samples at a time to achieve better performance than the existing method. When conducting future research, we plan to come up with a plan to overcome the difficulties of analysis due to the mass production of malicious codes by optimizing the analysis method to compensate for these limitations. Future work will compare accuracy and performance by analyzing large amounts of data with various classification methods. In addition, in this paper, although previously known malicious codes and families are used, a method for detecting new malicious codes or new variants of malicious codes will be studied.

Acknowledgements

This research was supported by the Yeungnam University Research Grant.

6 REFERENCES

[1] Ayewah, N., Pugh, W., Hovemeyer, D., Morgenthaler, J. D., & Penix, J. (2008). Using static analysis to find bugs. *IEEE software*, *25*(5), 22-29. https://doi.org/10.1109/MS.2008.130

[2] Moser, A., Kruegel, C., & Kirda, E. (2007, December). Limits of static analysis for malware detection. *Twenty-Third Annual Computer Security Applications Conference (ACSAC 2007)*, 421-430. https://doi.org/10.1109/ACSAC.2007.21

[3] Vetter, J. S. & De Supinski, B. R. (2000, November). Dynamic software testing of MPI applications with Umpire. *SC'00: Proceedings of the 2000 ACM/IEEE Conference on Supercomputing*, 51-51. https://doi.org/10.1109/SC.2000.10055

[4] Idika, N. & Mathur, A. P. (2007). A survey of malware detection techniques. *Purdue University*, *48*(2).

[5] Huabiao, L., Xiaofeng, W., & Jinshu, S. (2013). SCMA: Scalable and Collaborative Malware Analysis using System Call Sequences. *International Journal of Grid and Distributed Computing*, *6*(2), 11-28.

[6] Ďurfina, L., Křoustek, J., Zemek, P., Kolář, D., Hruška, T., Masařík, K., & Meduna, A. (2011). Design of a Retargetable Decompiler for a Static Platform-Independent Malware Analysis. *International Journal of Security and Its Applications*, *5*(4), 91-106.

[7] Farhana Hordri, N., Azurati Ahmad, N., Sophiayati Yuhaniz, S., Sahibuddin, S., Fadillah, A., Ariffin, M., Afifah Mohd Saupi, N., Ahmad, N., Zamani, Y. J., & Efendy Md Senan, M. F. (2018). Classification of Malware Analytics Techniques: A Systematic Literature Review. *International Journal of Security and Its Applications*, *12*(2), 9-18. https://doi.org/10.14257/ijsia.2018.12.2.02

[8] Griffin, K., Schneider, S., Hu, X., & Chiueh, T. C. (2009, September). Automatic generation of string signatures for malware detection. *International workshop on recent advances in intrusion detection*, 101-120. https://doi.org/10.1007/978-3-642-04342-0_6

[9] Bilar, D. (2007). Opcodes as predictor for malware. *International journal of electronic security and digital forensics*, *1*(2), 156-168. https://doi.org/10.1504/IJESDF.2007.016865

[10] Santos, I., Brezo, F., Nieves, J., Penya, Y. K., Sanz, B., Laorden, C., & Bringas, P. G. (2010, February). Idea: Opcode-sequence-based malware detection. *International Symposium on Engineering Secure Software and Systems*, 35-43. https://doi.org/10.1007/978-3-642-11747-3_3

[11] Griffin, K., Schneider, S., Hu, X., & Chiueh, T. C. (2009, September). Automatic generation of string signatures for malware detection. *International workshop on recent advances in intrusion detection*, 101-120. https://doi.org/10.1007/978-3-642-04342-0_6

[12] Santos, I., Penya, Y. K., Devesa, J., & Bringas, P. G. (2009). N-grams-based File Signatures for Malware Detection. *ICEIS (2)*, *9*, 317-320. https://doi.org/10.5220/0001863603170320

[13] Santos, I., Brezo, F., Ugarte-Pedrero, X., & Bringas, P. G. (2013). Opcode sequences as representation of executables for data-mining-based unknown malware detection. *Information Sciences*, *231*, 64-82. https://doi.org/10.1016/j.ins.2011.08.020

[14] Choi, Y. H., Han, B. J., Bae, B. C., Oh, H. G., & Sohn, K. W. (2012, August). Toward extracting malware features for classification using static and dynamic analysis. *8th International Conference on Computing and Networking Technology (INC, ICCIS and ICMIC)*, 126-129.

[15] E. Elhadi, A. A., Maarof, M. A., Bazara, I., & Barry, A. (2013). Improving the Detection of Malware Behaviour Using Simplified Data Dependent API Call Graph. *International Journal of Security and Its Applications*, 7(5). https://doi.org/10.14257/ijsia.2013.7.5.03

[16] Fan, C. I., Hsiao, H. W., Chou, C. H., & Tseng, Y. F. (2015, July). Malware detection systems based on API log data mining. *39th annual computer software and applications conference*, *3*, 255-260. https://doi.org/10.1109/COMPSAC.2015.241

[17] Firdausi, I., Erwin, A., & Nugroho, A. S. (2010, December). Analysis of machine learning techniques used in behavior-based malware detection. *Second international conference on advances in computing, control, and telecommunication technologies*, 201-203. https://doi.org/10.1109/ACT.2010.33

[18] Mulder, S. A., Blount, J., & Tauritz, D. (2011). Adaptive Rule-Based Malware Detection Employing Learning Classifier Systems: A Proof of Concept (No. SAND2011-2448C). Sandia National Lab.(SNL-NM), Albuquerque, NM (United States).

[19] Wagner, M., Fischer, F., Luh, R., Haberson, A., Rind, A., Keim, D. A., & Aigner, W. (2015). A survey of visualization systems for malware analysis. *Eurographics Conference on Visualization (EuroVis)*,105-125.

[20] Nataraj, L., Karthikeyan, S., Jacob, G., & Manjunath, B. S. (2011, July). Malware images: visualization and automatic classification. *Proceedings of the 8th international symposium on visualization for cyber security*, 1-7. https://doi.org/10.1145/2016904.2016908

[21] Trinius, P., Holz, T., Göbel, J., & Freiling, F. C. (2009, October). Visual analysis of malware behavior using treemaps and thread graphs. *6th International Workshop on Visualization for Cyber Security*, 33-38. https://doi.org/10.1109/VIZSEC.2009.5375540

[22] Lyda, R. & Hamrock, J. (2007). Using entropy analysis to find encrypted and packed malware. *IEEE Security & Privacy*, *5*(2), 40-45. https://doi.org/10.1109/MSP.2007.48

[23] Vinod, P., Jaipur, R., Laxmi, V., & Gaur, M. (2009, March). Survey on malware detection methods. *Proceedings of the 3rd Hackers' Workshop on computer and internet security (IITKHACK'09)*, 74-79.

[24] Karnik, A., Goswami, S., & Guha, R. (2007, March). Detecting obfuscated viruses using cosine similarity analysis. *First Asia International Conference on Modelling & Simulation (AMS'07)*, 165-170. https://doi.org/10.1109/AMS.2007.31

[25] Jang, J., Brumley, D., & Venkataraman, S. (2011, October). Bitshred: feature hashing malware for scalable triage and semantic analysis. *Proceedings of the 18th ACM conference on Computer and communications security*, 309-320. https://doi.org/10.1145/2046707.2046742

[26] Newsome, J. & Song, D. X. (2005, February). Dynamic Taint Analysis for Automatic Detection, Analysis, and Signature Generation of Exploits on Commodity Software. *NDSS*, *5*, 3-4.

[27] King, J. C. (1976). Symbolic execution and program testing. *Communications of the ACM*, *19*(7), 385-394. https://doi.org/10.1145/360248.360252

[28] Hafner, J., Sawhney, H. S., Equitz, W., Flickner, M., & Niblack, W. (1995). Efficient color histogram indexing for quadratic form distance functions. *IEEE transactions on pattern analysis and machine intelligence*, *17*(7), 729-736. https://doi.org/10.1109/34.391417

[29] Jaccard, P. (1912). The distribution of the flora in the alpine zone. *1. New phytologist*, *11*(2), 37-50. https://doi.org/10.1111/j.1469-8137.1912.tb05611.x

[30] Kim, J. H., Lee, S. W., & Youn, J. H. (2021). Malware Visualization and Similarity via Tracking Binary Execution Path. *International Journal of Smart Home*, *15*(1).

[31] Kim, J. & Lee, S. (2021). Malicious Behavior Detection Method Using API Sequence in Binary Execution Path. *Tehnički vjesnik*, *28*(3), 810-818. https://doi.org/10.17559/TV-20210202132203

[32] Bennett, C. H., Gács, P., Li, M., Vitányi, P. M., & Zurek, W. H. (1998). Information distance. *IEEE Transactions on information theory*, *44*(4), 1407-1423. https://doi.org/10.1109/18.681318

[33] Bailey, M., Oberheide, J., Andersen, J., Mao, Z. M., Jahanian, F., & Nazario, J. (2007, September). Automated classification and analysis of internet malware. *International Workshop on Recent Advances in Intrusion Detection*, 178-197. https://doi.org/10.1007/978-3-540-74320-0_10

[34] Jegou, H., Douze, M., & Schmid, C. (2010). Product quantization for nearest neighbor search. *IEEE transactions on pattern analysis and machine intelligence*, *33*(1), 117-128. https://doi.org/10.1109/TPAMI.2010.57

Contact information:

Jihun KIM, MSc
Department of Computer Engineering, Yeungnam University,
280 Daehak-Ro, Gyeongsan, Gyeongbuk, Republic of Korea
E-mail: f13521@naver.com

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

Doosan CHO, PhD, Professor
(Corresponding author)
Electrical & Electronic Engineering, Sunchon National University,
Suncheon, Jeollanam-do, South Korea
E-mail: dscho@scnu.ac.kr

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