International Journal of Smartcare Home
Vol.1, No.1 (2021), pp.75-84
https://dx.doi.org/10.21742/26531941.1.1.07

Malicious Code Characteristics Visualization using API

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

[1,2]Student, Department of Computer Engineering, Yeungnam University, 280 Daehak-Ro, Gyeongsan, Gyeongbuk, Republic of Korea
[3]Professor, Department of Computer Engineering, Yeungnam University, 280 Daehak-Ro, Gyeongsan, Gyeongbuk, Republic of Korea*
[1]f13521@naver.com, [2]noke15@ynu.ac.kr, [3]youn@yu.ac.kr*

Abstract

*The massification of malware through the generation of malware variants poses security threats to overall social and industrial societies. Since the quantity of malware is too big to simply analyze and defend against malware per se, it is inevitable to maximize the efficiency based on efficient analysis methods. In this study, the API is divided into 25 categories, and the interaction and frequency of the API are made into 25 * 25pixel images based on the matrix using RGB values. The Euclidean distance algorithm is applied to measure color similarity, and the similarity of mutual malicious behavior is calculated based on the final value of the Euclidean distance. As a result of comparing the similarity calculated by this method with the similarity calculated using the existing similarity calculation method, the similarity was calculated to be 5% to 10% higher on average.*

Keywords: *Malware, Binary analysis, Visualization, Similarity*

1. Introduction

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, which enables more in-depth and detailed analysis but will require much time and effort and add considerable difficulties to analysis if technologies obstruct static analysis such as execution file compression and code obfuscation are applied to malware. Another analysis method is dynamic analysis, which is executing malware in a virtual machine to analyze the malicious behavior. This method has the advantage of enabling a clear understanding of malicious behaviors even when execution file compression or code obfuscation has been applied to malware because malware is 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 present study adopts static analysis as the main analysis method. Despite that static analysis is conducted; the effects of dynamic analysis can be expected because the run streams of codes are tracked and analyzed. In addition, in the present study, malicious behaviors are analyzed through the APIs 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

Article history:
Received (March 9, 2021), Review Result (April 3, 2021), Accepted (June 10, 2021)

eISSN: 2653-1941 IJSH

through comparison between different pieces of malware, in the present study, the similarities of malware images are also calculated.

2. Related works

Although early studies that collected APIs to judge malicious behaviors listed API sequences, collected the log information regarding the use of APIs [1] or used the frequencies of APIs to judge malicious behaviors, recently, mathematical algorithms such as the nearest neighbor search algorithm [2] or the Longest Common Subsequence (LCS) [3] 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 [4]. The statistical values or opcodes of the APIs called in the binary codes are shown as an image or [5] in the form of file map images. In addition, dynamic analysis is sometimes utilized to express the statistics of the APIs used as thread maps [6] or extract and visualize the entropy of the files [7]. 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 [8]. The similarity calculations as such can be applied with qualified algorithms such as the Cosine similarity algorithm [9] and Jaccard distance algorithm [10] 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 the 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 to complement the limitations of existing studies and maximize efficiency.

3. Proposed method

The present study adopts static analysis that analyzes malware at the code level as a basic mechanism. 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 observe 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 behaviors. The final purpose of 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

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

The codes on the left of [Figure 1] are binary codes for showing static execution path searches. The binary code in [Figure 1] is visualized as shown in the figure on the right. This mechanism is applied equally even when there are subroutines in the instruction set.

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

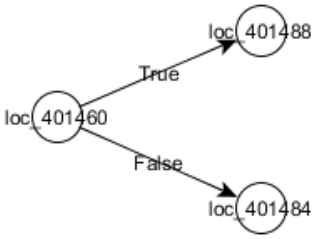


Figure 1. Example of static execution path exploration

In this study, the path is expressed based on the Windows API. However, there is a problem that the number of APIs is too large to make the APIs into nodes. To solve such problems, the APIs will be reclassified into 25 upper categories through the functions of the APIs so that behaviors can be judged and the temporal efficiency can be enhanced.

The method proposed in this study enables the understanding of the interactions between APIs because it uses execution path searches despite that it is a static analysis so that the effects of dynamic analysis can be expected.

3.2. 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 1. Composition of graph visualization

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

[Table 1] shows the 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 RGB colors consisting of 0 to 255 increases 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. Figure 2 shows a graph made by imaging 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.

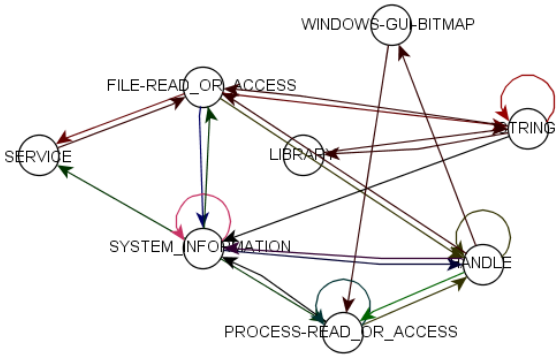


Figure 2. Gen:Variant.Zusy.210164 malicious behavior

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 adaptability 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.

3.3. 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. Figure 3 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.

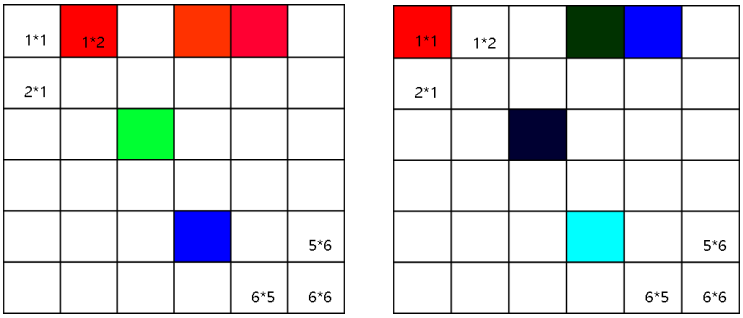


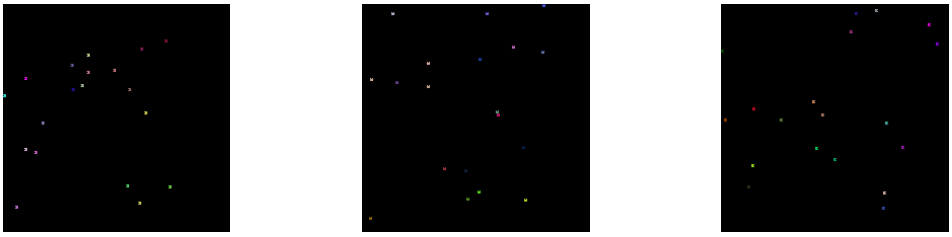
Figure 3. Test pixel images

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, the color similarity is calculated as 1.

4. Experiment and discussion

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.

[Figures 4] and [Figure 5] show the malicious behavior of two malware families, Zusy and Confiker made into pixel images.



Zusy.260481 Zusy.210164 Zusy.Elzob

Figure 4. Gen:Variant.Zusy family visualization

Conficker.C2 Conficker.A1 Conficker.Z.03

Figure 5. W32/Confiker.worm family visualization

It can be seen that Zusy uses complex APIs because the pixels are evenly distributed. However, compared to Zusy, Confiker show intensively distributed pixels on some parts. This malicious behavior pixel image can be used as 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 2. 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 3. 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

The results are as shown in [Table 2] and [Table 3]. First, about 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, the fact that the positions of the pixels indicated by the colors are intensive and the colors are similar can be seen through the calculated similarity, 82%.

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 with 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 four methods including the proposed method indicate higher similarities when the values are closer to 100 and indicate the average values of similarities.

Table 4 Malware Family Sample DataSet

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 4] shows sample data of malicious codes used for similarity comparison in this study.

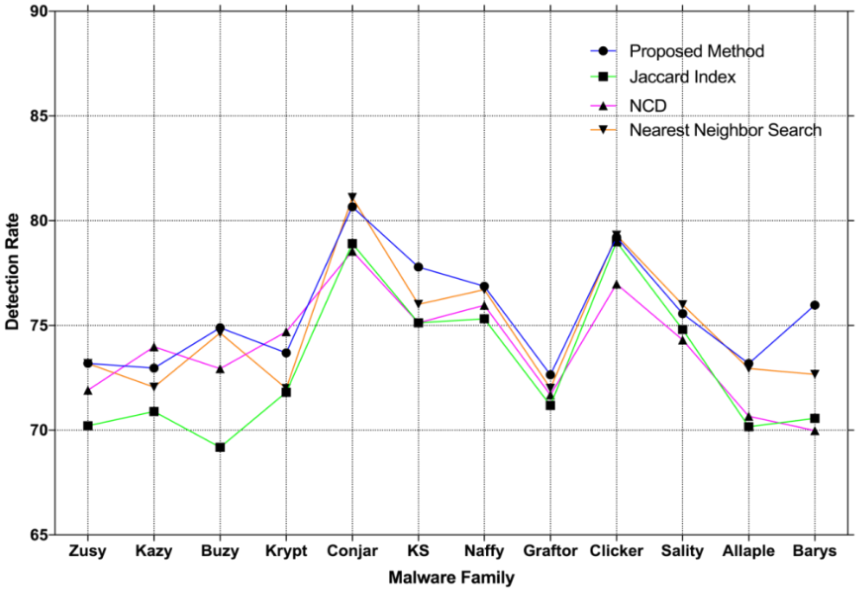


Figure 6. Detection efficiency

[Figure 6] is a graph of the statistics of detection rates compiled by applying the four similarity verification methods to 12 malware families. Overall, compared to other methods, we were able to confirm satisfactory performance. In particular, the similarity of families with more than 100 samples, such as Zusy, Kazy, and Graftor, showed that the method proposed in this study showed higher similarity than other methods.

This can highlight efficiency in automating the analysis 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

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, 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. By this method, the similarity was calculated to be 5% to 10% higher on average than other conventional methods. However, 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 the analysis of 10,000 or more malware samples. In future studies, to complement the limitations as such, measures to overcome the difficulties in the analysis due to the massification of malware will be prepared through the optimization of analysis methods.

Acknowledgments

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

References

[1] C. I. Fan, H.W. Hsiao, C. H. Chou, and Y. F. Tseng, “Malware detection systems based on API log data mining,” 2015 IEEE 39th annual computer software and applications conference, vol.3, **(2015)**

[2] I. Firdausi, A. Erwin, and A. S. Nugroho, “Analysis of machine learning techniques used in behavior-based malware detection,” 2010 second international conference on advances in computing, control, and telecommunication technologies. IEEE, **(2010)**

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

[4] M. Wagner, F. Fischer, R. Luh, A. Haberson, A. Rind, D.A. Keim, and W. Aigner, “A survey of visualization systems for malware analysis,” Eurographics Conference on Visualization (EuroVis), **(2015)**

[5] L. Nataraj, S. Karthikeyan, G. Jacob, and B. S. Manjunath, "Malware Images: Visualization and automatic classification," Proceedings of the 8th international symposium on visualization for cybersecurity, **(2011)**

[6] P. Trinius, T. Holz, J. Göbel, and F.C. Freiling, “Visual analysis of malware behavior using treemaps and thread graphs,” 2009 6th International Workshop on Visualization for Cyber Security, IEEE, **(2009)**

[7] R. Lyda and J. Hamrock, “Using entropy analysis to find encrypted and packed malware,” IEEE Security and Privacy 5.2, pp.40-45, **(2007)**

[8] P. Vinod, R. Jaipur, V. Laxmi, and M. Gaur, “Survey on malware detection methods,” Proceedings of the 3rd Hackers’ Workshop on computer and internet security (IITKHACK’09), **(2009)**

[9] A. Karnik, S. Goswami, and R. Guha, “Detecting obfuscated viruses using cosine similarity analysis,” First Asia International Conference on Modelling and Simulation (AMS'07), IEEE, **(2007)**

[10] J. Jang, D. Brumley, and S. Venkataraman, “Bitshred: Feature hashing malware for scalable triage and semantic analysis,” Proceedings of the 18th ACM conference on Computer and communications security, **(2011)**

Authors

Jihun Kim, M.S.

He received an M.S. degree in Computer Engineering from Yeungnam University, South Korea. His research interests include Binary analysis, Malware Analysis, and Visualization.

Sungwon Lee, M.S.
He received an M.S. degree in Computer Engineering from Yeungnam University, South Korea. His research interests include Binary analysis, Malware Analysis, and embedded systems.

Jonghee Youn, Ph.D, Professor
He received the B.S. degree in the school of electrical engineering and computer science from Kyungpook National University, Daegu, Korea, in 2003 and the Ph.D. degree in electrical engineering and computer science from the Seoul National University in 2011. He is currently an Assistant Professor in the Department of Computer Engineering at Yeungnam University. His research interests are broadly in the areas of a compiler, software optimization, embedded systems, mobile computing, and Security.

This page is empty by intension.