Using a Fine-Grained Hybrid Feature for Malware Similarity Analysis

Jing Liu(✉), Yongjun Wang, Peidai Xie, and Xingkong Ma

College of Computer, National University of Defense Technology, Changsha, Hunan, China
liujing_nudt@nudt.edu.cn

Abstract. Nowadays, the dramatically increased malware causes severe challenges to computer security. Most emerging instances are variants of previously encountered malware through polymorphism and metamorphism techniques. The traditional signature-based detecting methods are ineffective to recognize the enormous variants. Malware similarity analysis has become the mainstream technique of identifying variants. However, most existing methods are either hard to handle polymorphic and metamorphic samples based on static structure feature, or time consuming and resource intensive by using dynamic behavior feature. In this paper, we propose a novel malware similarity analysis method based on a fine-grained hybrid feature by exploiting the complementary nature of static and dynamic analysis. We integrate dynamic runtime behavior with static function-call graph. The hybrid feature overcomes the limitation of using static and dynamic feature separately and with more accuracy. Furtherly, we use graph edit distance, and inexact graph matching algorithm as metric to measure the distance between malicious instances. We have evaluated our algorithm on real-world dataset and compared with other approach. The experiments demonstrate that our method achieves higher accuracy.

Keywords: Similarity analysis · Function-call graph · Hybrid feature · Graph edit distance

1 Introduction

Malware poses a major threat to network security. According to the latest report of Symantec, more than 430,000,000 new malware samples were discovered in 2015, up 36 percent from the year before. The sheer volume of malware brings severe challenges to security vendors. However, research shows that the majority of new incoming malware instances are merely variations of encountered malware through polymorphism and metamorphism techniques. They share the same functionality while have different syntactic representations.

Malware similarity analysis has been put forward to efficiently cope with the tremendous number of variants. Through precisely measuring the similarity based on quantitative metric to determine whether a malware program is similar to a previously-seen sample. A large amount of time and resources could be saved to avoid

J.J. (Jong Hyuk) Park et al. (eds.), *Advances in Computer Science and Ubiquitous Computing*, Lecture Notes in Electrical Engineering 421, DOI 10.1007/978-981-10-3023-9_9

the duplicated analysis of variants. It is the basis for automatic malware detection. It is also the foundation of malware classification and phylogeny model generation.

In this paper, we propose a novel malware similarity analysis metric using a fine-grained hybrid feature, which combines static function-call graph and dynamic runtime traces in a way that taking advantage of both simultaneously. Firstly, we extract the function-call graph of programs using static analysis, which is resilient to low-level obfuscation, such as basic block-reordering, register reassignment. Each vertex in function-call graph represents a function and each edge represents a caller-callee relationship between functions. Then we extract the dynamic runtime trace sets as function labels. At last, we use graph edit distance, an inexact graph matching method as metric to calculate the similarity degree among malicious samples. We have evaluated our algorithm on real-world dataset and compared with other approach. The experiments demonstrate that our method achieves higher accuracy.

The rest of this paper is organized as follows. We review the related work in Sect. 2. In Sect. 3, we describe the overview of framework. Then we introduce the extraction of the hybrid feature in Sect. 4 and the calculation of similarity metric is presented in Sect. 5. Section 6 evaluates the result of experiment. Finally, a summary of the paper is given in Sect. 7.

2 Related Work

Malware similarity analysis has attracted considerable attention. Most existing methods based on either static features or dynamic features.

Static features are extracted from malicious programs without executing it. Shafiq et al. [1] proposed to extract distinguishing features from portable executable (PE) format using the standard structural information that Microsoft Windows operating system defined. Kolter et al. [2] used n-grams of byte codes presented in the malware binary as features. Xin et al. [3] employed function-call graph which is a high-level structural feature for malware classification. However, as mentioned by Moser et al. [4], static analysis is difficult to handle the advanced obfuscation or self-mutating instances, and thus affects the accuracy of static features.

Distinguishing from static features, dynamic features are extracted from execution traces which make them more resilient to encryption or other obfuscation techniques. Blokhin et al. [5] partitioned system call logs acquired from sandbox into system call sequences as features. Bailey et al. [6] described malware behavior at a high level abstraction in terms of system state change profiles that the malware causes on the system, like modified registry keys, network access. Wüchner et al. [7] presented quantitative data flow graphs (QDFGs) to model program behavior through using system calls integrated with quantifiable data flow. However, many newly malware instances are able to detect the instrumented environment and refuse preforming malicious activity to evade dynamic analysis.

3 System Overview

Malware similarity analysis is divided into two steps: feature extraction and similarity calculation. Figure 1 shows the overview of our framework. A malware sample is represented as a function-call graph. Each vertex in the graph corresponds to a function and edges represent the caller-callee relationship between functions. There are two kinds of functions: local functions and external functions. For each local function, we record its execution traces with dynamic instrumentation tool Pin as its label. For external functions, we take the string of function names as the label which indicates the functionality of each function. We integrate dynamic feature into function representations as a fine-grained hybrid feature.

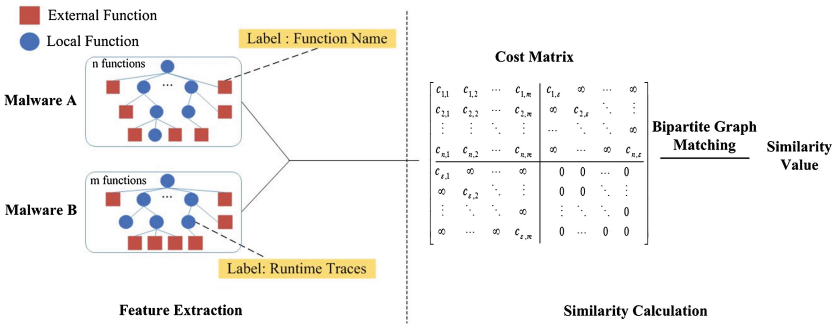

Fig. 1. The system overview

Then we use graph matching algorithm to calculate the similarity degree between pairwise samples. Graph matching methods fall into two categories: exact matching and inexact matching. Exact matching problems are NP-complete. Therefore, we use graph edit distance, one of the most widely used inexact matching algorithms. It is highly flexible and is applicable to types of graph integrated with special domain knowledge by means of cost functions. Graph edit distance is defined as the cost of the least expensive sequence of edit operations that are needed to transform one graph to another.

4 Feature Extraction

This section we first introduce the definition of function-call graph. Then, we describe the extraction of the dynamic runtime traces for each local function.

4.1 Function-Call Graph

G = (V,E) is a directed graph composed of vertex set *V* and edge set *E*. Each vertex in the graph corresponds to a function included in the program. Each edge represents the caller-callee relationship between functions. The vertex can be divided into two categories: local functions and external functions. Local functions are functions written by

malware authors. Statically-linked and dynamic-imported functions are the external functions.

Function identification is a tough challenge in binary analysis. In our paper, we use IDA Pro to identify the boundary of functions which has achieved reasonable accuracy. In IDA representation, local functions are named with "sub_xxxxx", external functions are named just the function names. IDA can provide the function-call graph of applications directly. But as pointed in [8], the performance of IDA is still imprecise, the missed function and misidentified function rate is alarming. In future, we will take this issue into account.

Each function in the graph has a label. The labels of external functions are function names. Whereas the label for local functions are the execution traces acquired from dynamic tools. We run the malicious program with instrumentation tool Pin and record each function's runtime traces to form a label vector F_l, which is composed of feature sets $F_l = \{f_1 f_2, \cdots, f_n\}$. Each f_i denotes an aspect of dynamic behavior of functions, like system calls the function invoked, the values the function written to memory, etc. We choose four feature sets: the values read from the stack f_1, the values written to stack f_2, the values read from memory f_3 and the values written to memory f_4. The feature vector is easily being extended.

5 Similarity Calculation

The central component of malware similarity analysis is to measure the distance among malware instances. The samples are represented as function-call graph, thus casts the problem into graph matching. Therefore, we use graph edit distance as a metric to weigh the similarity between variants.

Graph edit distance was first proposed in [9]. It is defined as the minimum cost amount of operation that is needed to transform one graph into another. Bipartite graph matching as an approximate computation method of graph edit distance has been proposed in [10]. It is a suboptimal method based on the procedure that mapping nodes and their local structures of one graph to nodes and structures of another graph.

Let $G_1 = (V_1, E_1, l_1)$ and $G_2 = (V_2, E_2, l_2)$ be the source and target graph as two parts of bipartite graph, where $V_1 = (v_1, \cdots, v_n)$, $V_2 = (u_1, \cdots, u_m)$. The cost matrix is as below:

$$C = \left[\begin{array}{cccc|cccc} c_{1,1} & c_{1,2} & \cdots & c_{1,m} & c_{1,\varepsilon} & \infty & \cdots & \infty \\ c_{2,1} & c_{2,2} & \cdots & c_{2,m} & \infty & c_{2,\varepsilon} & \ddots & \vdots \\ \vdots & \vdots & \ddots & \vdots & \cdots & \ddots & \ddots & \infty \\ c_{n,1} & c_{n,2} & \cdots & c_{n,m} & \infty & \cdots & \infty & c_{n,\varepsilon} \\ \hline c_{\varepsilon,1} & \infty & \cdots & \infty & 0 & 0 & \cdots & 0 \\ \infty & c_{\varepsilon,2} & \ddots & \vdots & 0 & 0 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \infty & \vdots & \ddots & \ddots & 0 \\ \infty & \cdots & \infty & c_{\varepsilon,m} & 0 & \cdots & 0 & 0 \end{array}\right]$$

In our cost matrix, $c_{ij} = 1/\varepsilon_{ij}$, where ε_{ij} denotes the similarity degree of two functons. For external function nodes, ε_{ij} is measured by the longest common substring (LCS) of function names. For local function nodes, ε_{ij} is measured by Jaccard similarity coefficient between two trace vectors F_l.

6 Evaluation

To conduct the accuracy of the algorithm we proposed, we use the data set from VX Heavens, which has already been classified into malware families. The data set has 17 malware families and 1,326 samples, including Worms, Trojans, and Virus. We compared the results with [11], which used static function-call graph to compute similarity of binaries. The results demonstrate that hybrid feature outperforms solely static function-call graph.

Virus.Win32.Sality is considered as one of the most complex and formidable family of malware according to Wiki. And Table 1 shows the similarity matrix between variants in family Sality. The values of leading diagonal of the matrix are equal to 1 which demonstrates comparing with themselves. The upper triangular half of the matrix are the data of our experiment and below are [11]. The results indicate that, our method outperforms [11] when the similarity value is low within family variants. This is because variants' dynamic behaviors may keep in step with each other, although their static structures are different due to obfuscations.

Table 1. The similarity matrix of Virus.Win32.Sality

	Sality.a	Sality.c	Sality.d	Sality.e	Sality.f	Sality.g
Sality.a	1	0.991	0.941	0.618	0.612	.0867
Sality.c	0.996	1	0.933	0.618	0.612	0.862
Sality.d	0.96	0.957	1	0.651	0.639	0.889
Sality.e	0.627	0.625	0.645	1	0.961	0.856
Sality.f	0.408	0.408	0.419	0.581	1	0.848
Sality.g	0.870	0.867	0.884	0.682	0.438	1

Table 2. The similarity matrix of Email-Worm.Win32.Klez

	Klez.a	Klez.b	Klez.c	Klez.d	Klez.e	Klez.g	Klez.h	Klez.i	Klez.j
Klez.a	1	0.964	0.976	0.911	0.791	0.789	0.783	0.783	0.783
Klez.b	0.959	1	0.939	0.939	0.801	0.8	0.793	0.793	0.801
Klez.c	1	0.959	1	0.916	0.792	0.791	0.784	0.784	0.792
Klez.d	0.869	0.91	0.869	1	0.811	0.811	0.806	0.806	0.812
Klez.e	0.614	0.639	0.614	0.672	1	0.996	0.967	0.968	0.994
Klez.g	0.614	0.639	0.614	0.672	1	1	0.965	0.967	0.993
Klez.h	0.615	0.637	0.615	0.656	0.948	0.948	1	0.999	0.949
Klez.i	0.615	0.637	0.615	0.656	0.948	0.948	1	1	0.949
Klez.j	0.614	0.639	0.614	0.672	1	1	0.948	0.948	1

Table 2 shows the similarity matrix of family Email-worm.Win32.Klez. Compared to [11], our result achieves better accuracy. Although the accuracy of a few pairs of our results are little bit lower than theirs, such as pair Klez.j and Klez.g, the similarity value of their method is 1 and ours is 0.993. But for the rest of pairwise samples, our results are much higher than [11].

7 Conclusion

The large volume of malware variations poses major challenge to Anti-Virus companies. Malware similarity analysis is a critical step for malware detection and classification. In this paper, we propose a novel fine-grained hybrid feature to calculate the similarity degree between malware instances. We merge dynamic runtime traces into static function-call graph representation as a hybrid one. In evaluation, we have conducted extensive experiments with 1,326 samples in 17 families. The result shows that our algorithm is more accurate. The main contributions of our work include: (1) a hybrid graph feature for computing the similarity between malware variants to improve the accuracy of the result; (2) integrate dynamic runtime traces into function node representation in order to take advantage of both simultaneously; (3) a fully study of the performance to validate its efficiency and accuracy with a 1,326 samples database.

Acknowledgement. This work is supported by the National Science Foundation of China (No.61472439, No.61271252).

References

1. Shafiq, M.Z., Tabish, S.M., Mirza, F., Farooq, M.: PE-Miner: mining structural information to detect malicious executables in realtime. In: Kirda, E., Jha, S., Balzarotti, D. (eds.) RAID 2009. LNCS, vol. 5758, pp. 121–141. Springer, Heidelberg (2009). doi:10.1007/978-3-642-04342-0_7
2. Kolter, J.Z., Maloof, M.A.: Learning to detect and classify malicious executables in the wild. J. Mach. Learn. Res. **6**(4), 2721–2744 (2006)
3. Hu, X., Chiueh, T.-C., Shin, K.G.: Large-scale malware indexing using function-call graphs. In: Proceedings of the 16th ACM Conference on Computer and Communications Security. ACM (2009)
4. Moser, A., Kruegel, C., Kirda, E.: Limits of static analysis for malware detection **68**(6), 421–430 (2008)
5. Blokhin, K., Saxe, J., Mentis, D.: Malware similarity identification using call graph based system call subsequence features. In: IEEE International Conference on Distributed Computing Systems Workshops (2013)
6. Bailey, M., Oberheide, J., Andersen, J., Mao, Z.,Morley, Jahanian, F., Nazario, J.: Automated classification and analysis of internet malware. In: Kruegel, C., Lippmann, R., Clark, A. (eds.) RAID 2007. LNCS, vol. 4637, pp. 178–197. Springer, Heidelberg (2007). doi:10.1007/978-3-540-74320-0_10

7. Wüchner, T., Ochoa, M., Pretschner, A.: Robust and effective malware detection through quantitative data flow graph metrics. In: Almgren, M., Gulisano, V., Maggi, F. (eds.) DIMVA 2015. LNCS, vol. 9148, pp. 98–118. Springer, Heidelberg (2015). doi:10.1007/978-3-319-20550-2_6
8. Bao, T., et al.: Byteweight: learning to recognize functions in binary code. In: USENIX Security Symposium (2014)
9. Sanfeliu, A., Fu, K.S.: A distance measure between attributed relational graphs for pattern recognition. IEEE Trans. Syst. Man Cybern. **SMC-13**(3), 353–362 (1983)
10. Riesen, K., Bunke, H.: Approximate graph edit distance computation by means of bipartite graph matching. Image Vis. Comput. **27**(7), 950–959 (2009)
11. Shang, S., et al.: Detecting malware variants via function-call graph similarity. In: International Conference on Malicious and Unwanted Software (2010)