A Behavior-Based Malware Variant Classification Technique

Guanghui Liang, Jianmin Pang, and Chao Dai

***Abstract*—The research on detection malware variants attracts much attention in recent years. However current variant classification methods either are interfered by some confusion technologies or have a high time or space complexity. In this paper, a classification technique using dynamic analysis based on behavior profile is proposed. We capture API calls and other essential information of running malware, then establish their multilayer dependency chain according to the dependency relationship of these function calls. In order to deal with the confusion, we remove sequence confusion, sequence noise, and other confusions to optimize the multilayer dependency chain. Finally, a similarity comparison algorithm is used to identify the degree of similarity between malware variants. The experimental results demonstrate that our classification technique is feasible and effective.**

***Index Terms*—Malware, variants, dependency chain.**

I. Introduction

Most Internet security problems are caused by malware, such as botnet, Trojan, rootkits, virus, worms. Some anti-malware companies get thousands of new malware samples every day. The threshold of making malware becomes increasingly low for malware developers, these malware developers easily use a variety of malware core on the network and other modification tools to produce numerous malware in short time. Malware and its variants bring a great challenge to the malware analysis area [1]. Variant recognition methods based on static scan usually lose their efficiency when dealing with this situation, because they only compare the static structure and semantic information.

Dynamic monitoring technology has become the main form of malicious behavior mining. Compared with static analysis, dynamic analysis ill not be affected by the obfuscation technology such as packers, polymorphic, etc [2]-[4].

The main contributions of this paper are as follows:

1) We implemented a dynamic analysis framework which can capture malware behaviors on Temu. We capture API functions while the malware running, according to its behavior on registry, service, process and so on. We also took the input parameters, output parameters and return value into consideration so that a more detailed characterization of the malware behavior is acquired.
2) We extracted a behavioral profile that accurately describes the runtime activity of malware. According to the control dependencies between different actions, we converted the initial API flow sequence into a multi-function behavior dependency chain. As variants often use some common confusing technology, we removed the rearrangement sequence, noise and other confusing information in behavior dependency chain to improve the ability to identify variants.
3) We proposed a weighted matching algorithm of malicious code variants based on behavior dependency chain. We design weighted Jaccard similarity matching algorithm according to the different behavior type of malware variants. The correctness of the proposed algorithm is verified in the experimental section finally.

Manuscript received July 29, 2014; revised November 20, 2014. This work was supported by the National Natural Science Foundation of China (Grant 61472447).

The authors are with the State Key Laboratory of Mathematical Engineering and Advanced Computing, Zhengzhou 450002, China (e-mail: lghray1987@163.com).

II. Related work

Classification of malicious code variants can be divided into two categories, which are static analysis and dynamic analysis respectively. L. Wu, who use software such as PEID or UPX to unpack the malware, and then call the static analysis method to get the program flow diagram, by matching similarity graph of vertices and edges to achieve the classification of malware variants [5]. This method is based on the malicious code correctly unpacking and disassembly analysis. However it is not very easy to do this.

In dynamic analysis, Forrest proposed fixed-length sequence of N-gram recognition model based on system call [6]. This method uses sliding windows to intercept short N fixed-length of system call and performs the anomaly detection and similarity analysis. After monitoring a large number of system calls experiments and they found that when the fixed-length is 6, the anomaly detection efficiency is relatively high. The algorithm is easy to implement, but false negative rate of detection is high, and requires a lot of test cases. Bayer [7] *et al*. found that the dynamic behavior analysis has a very big advantage in detecting malware variants. They proposed a behavior-based malware clustering approach, which establishes a multi-dimensional feature vector to describe the behavior of malware using the hash algorithm, but the method does not take the resistance techniques which malware often used into account, so this method may be unable to effectively deal with confusion and other anti-analysis malware so as to fail to achieve effective classification.

Yang Yi proposed a graph matching similarity-based identification method [8]. They generated control dependency graph and data dependency graph through dynamic taint propagation, then optimize the dependency graph by eliminating redundant calls. Finally, they compare the similarity of dependency graph between the malware. The

advantage of this method is that it utilizes the graph matching in variants classification, which can accurately describe malware behavior. However, the processing time is so time consuming that it is not suitable to handle large quantities of malware classification.

Sun Xiaoyan solved the question of the sequence of confusion, noise injection and simulation sequences in sliding window based on sequence analysis. She used the branch sequence Markov chain judgment and interactive objects to achieve the automatic classification of malware [9]. Hanlan Sheng used the information gain to evaluate API function call sequences and API input parameters as behavioral characteristics in dynamic detection of malware, which improve the recognition rate of malware detection [10].

In summary, when dealing with the malware variants classification, dynamic analysis performs better than static analysis, especially in against confusion technology and other anti-analysis techniques.

III. Our Technique

A. Motivation

Classification of malware variants has been concerned by analysts in a long period. Evolving malware generates a lot of variants and brings great challenges to analytical work. Although these variants change in the file format and appearance, but there are many similarities with the behavior on the specific function. For example, a Trojan named "rwxing" generated a dozen variants in December 2012. Although these variants look different with each other, the start-up mode, the injection mode and registry behavior are very similar. Therefore, analyzing malware variants on the specific behaviors and comparing these behaviors could realize the malware variants effective identification and classification.

In this paper, according to control dependencies between different API calls based on malware behavior, we convert the generally function flow to multilayer behavior chain. At the same time for some confusion and noise sequence characteristics, we optimize the multilayer behavior chain. Compared with the graph matching methods, Not only the efficiency of classification is ensured, this method also can reduce the analysis time.

B. Dynamic Monitor Based on TEMU

1) Introduction to TEMU

TEMU comes from a dynamic binary platform called Bitblaze [11], which was released by Professor Dawn Song's security team of University of California. Bitblaze includes three components, which are TEMU, VINE and Rudder respectively [12]. TEMU is one of the dynamic analysis module which is based on QEMU. TEMU adds taint analysis engine and semantic extraction module, providing a set of API functions and callback functions for users to develop functional plugin. Users can load the plugin and perform analysis work [13]. The architecture of TEMU is shown in Fig. 1.

2) Capturing behavior information

When the simulation system is Windows, TEMU provides a kernel module which is used to get the operating system semantic information. The kernel module runs as a driver on Windows systems. Whenever detecting a new process is loaded into memory, the module will get the address range and export symbol information. This paper analyzes malware in PE format, so the simulation system is Windows.

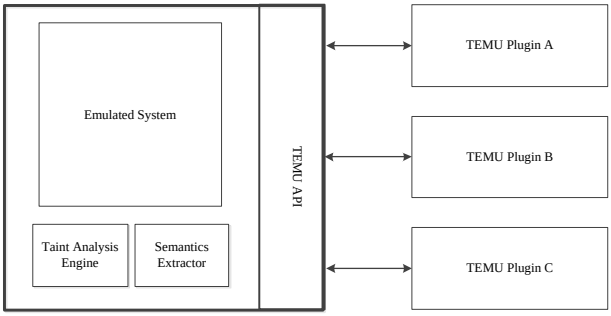

Fig. 1. Architecture of TEMU.

The plugin we developed communicates with the kernel module through I/O interface. Our plugin could get information containing currently executing instruction, basic block instruction information and module information. We analyze the captured information, and extract the API information during malware running, then get the input as well as output parameters and return values in stack and memory at the beginning or end of API calls. We combine all the information with the some semantic information for further analysis.

C. Behavior Profile: Multilayer Dependency Chain

This paper defines a function template. The behavior of the malware is divided into six sub-behaviors, which are file behavior, registry behavior, service behavior, process behavior, network behavior and the behavior gaining operating system information respectively. These behaviors cover the main aspects of malware behavior. In a function template, six categories of behavior are related to corresponding functions. For instance, file behavior is labeled A1 (Note that memory behavior is also treated as file behavior), the registry behaviors are labeled as A2, service behavior is marked as A3, process behavior is labeled as A4, network behavior is labeled as A5, behavior of gaining operating system information is marked as A6. With help of the continuous testing, the template contains a total of six categories of behavior about 160 API functions.

After being processed by the function template, each function captured by our plugin will be stored in the form of structures in memory. The definition of nodes as follows:

Corresponding definition:

1) Definitions of nodes

Each API function is a node structure, and each node contains information which is equivalent to a 6-tuple (Type,Name,IN_P,Out_P,Ret,Next).

Type: the type of function, there are four types. Type 1 means that the functions mainly produce function handle or other control handles, such as CreateFile (), CreateProcess (), etc; type 2 represents functions that rely on handles created by type 1, such as ReadFile (), WriteFile ()etc; type 3 represents functions that close these handles, such as CloseHandl (), RegClose () and other functions; type 0 represents

independent functions, that does not have control dependency, such as ShellExecute (), GetTickCount () and so on.

Name: function name;

IN_P: a structure storing the input parameters of function

Out_P: a structure storing the output parameters of function

Ret: value of function returned

Next: a pointer to the next node

2) Definition of chain

In behavior description, a complete sequence of operations is defined as operating chain L (N1, N2, ... NK), where N represents the function of each element nodes. Such as functions F1:. CreateFile, F2: ReadFile F3: CloseHandle can form the control function chain L (F1, F2, F3). These functions perform a complete file operation. Compared with dependency graph, dependency chain is one-dimensional and easier to compare and calculate.

We establish control dependency behavior chain of malware based on the relationship between function flow and dependency of different functions. The dependency chain could better describe software profile for analyzing malware.

For the function flow N1，N2，……NN, we process each node one by one. The step to make dependency chain as follow:

Step 0: analyze the type of the function node. If it is type 1, then turn to step 1; if it is type 2, turn to step 2; If it is type 3, then turn to step 3; If it is type 4, then turn to step 4;

Step 1: insert a new head node on the longitudinal control chain, and set the rely flag with the Ret field, then go to step 0;

Step 2: Traverse the active control chain from the scratch. According to the Name and IN_P field, decide which chain should be the right chain and insert the node at the end of the chain.

Step 3: Find the corresponding control chain. Turn off the active flag, skip to step 0;

Step 4: Insert a new control chain, and turn off the active flag. Skip to step 0;

When all the function nodes are inserted, the algorithm will generate a multilayer behavioral chain. Each behavior chain is a complete sequence of operations. For instance, open file to read and write files, and then close the file. For example, the multilayer behavioral chain of r Trojan "NetThief" shown in Fig. 2.

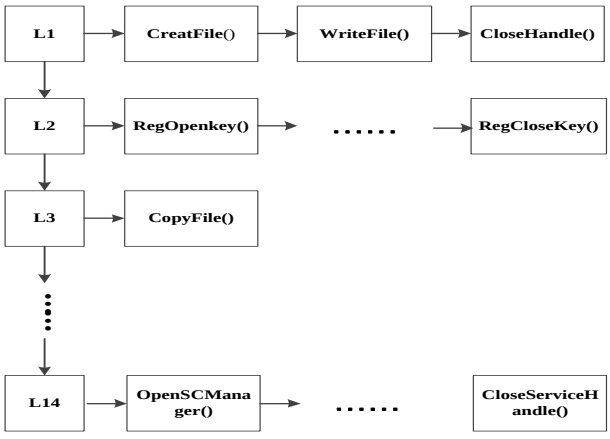

Fig. 2. Multilayer dependency chain.

The head node of each chain contains the information about the number of nodes, the type of operation (such as file, registry or services) and the active flag.

D. Removing the Confusion and Noise

The usual obfuscations used by malware variants are sequence confusing, sequence noise, parameters decomposition, which try to change the function sequence or control flow graph for the aim of interference identification.

1) Sequence confusion

For the sequence S1, S2; generally each of them has a complete operating behavior and means different behaviors. To confuse the observed sequence, malware often cross-random arrange various function and maintain their relative order in the sequence S1 S2. This will form numerous functional equivalent sequences, but which are different in appearance.

2) Sequence noise

There are normal sequence S1and redundant sequence S2, the distribute one or more S1 in S2 sequence, sequence S2 is called sequence noise.

3) Parameter decomposition

The function N will run multiple times because of decomposition of key parameters. It will have more than one function N. We take the function writefile() as an example. If one parameter is "c://windows//ststem32" , the malware author may divide this parameter into three parts, since "c://windows//ststem32" is a sensitive string. After division, function writefile() will appear three time.

We establish a confusion library, which include common confusion type mentioned above. So we use different methods to eliminate the confusion.

Eliminating the sequence confusion:according to the establishment of dependency chain in 3.3, we have summarized function of different types or dependencies to a different dependency chain. So if there are S1 and S2 in the function flow, they will be summarized to different chain.

Eliminating the function noise:Sun Xiaoyan used the Markov chain to determine the next sequence is noise or not [9]. This method's temporal overhead is high. The essential feature of the sequence noise is that they just disrupt the normal sequence of position, and will not cause changes in the function and system status. The sequence noise is always a short sequence used repeatedly. Part of them could be filtered by the function template in 3.3, and the rest are short operation chains during multilayer dependency chain. So we scan all the short chains which contain less than 4 function nodes, and compare them with the confusion library. If the short chain matches the noise in confusion library, then delete the chain from the multilayer chain.

Eliminating the parameters decomposition: parameter decomposition actually is an equivalent transformation. In the actual analysis, we found some of the malware which often read or write the key character string or the contents of the file several times, thereby changing the length of the sequence. Consequence of the parameters decomposition is that there will be numerous same function calls in a chain. When we scan the chain and find some continuous same function calls, we match the chain with the confusion library and check the parameters. If the result is true, we will merge the corresponding parameters.

E. Classification Algorithm

After the processing of dependence and optimization process, multilayer dependency chain represent the behavior profile of the malware. In this paper, we weighted similarity with Jaccard similarity. According to the multilayer chain, we calculate the similarity of each type of operation. For example, the file operation similarity, registry operations similarity will be calculated respectively. We calculate the weight of each type based on the proportion of each operation in the whole behavior profile.

For example, if file behavior of malware A and B is FA and FB. The similarity of file operation $S(F_A, F_B)$ is:

$$S(F_A, F_B) = \frac{|A \cap B|}{|A \cup B|}$$

If $S(F_A, F_B) = 1$, which means the file operation of A and B is equal to each other. After calculating the similarity of files, registry, services, etc., then we calculate the value of each operation based on the proportion of each sub-behavior. Weight of file operation is calculated as follows: if the total number of chain in multilayer chain is K and the number of file operation chain is F, then the file operation similarity value is $T_1=F/K$. For example, in Trojan "Lying", the file operation chain number is 6, total chain is 21, the weight of file operation similarity is $T_1 = 0.286$.

In summary, the similarity is calculated as follows:

$$P = A_1T_1 + A_2T_2 + ... + A_6T_6$$

According to the value of P, we could describe the degree of similarity between two malware about their behavior. After continuous variant experiment, there will be a reasonable threshold. A higher threshold represents a higher degree of similarity between the malware.

IV. Experiment and Evaluation

Our experimental platform is Lenovo E30, whose host OS is Ubuntu, guest OS is Windows XP SP3 on which we run the malware.

A. Similarity Detection between Different Variants

We use Trojan.Spook.a and Backdoor.Win32.Alicia.d as test samples. We run two dynamic variants on the analysis platform. In the capture and confusing process, Trojan.Spook.a generates 16 behavioral chains, Backdoor.Win32.Alicia.d generates 9 behavioral chains. The final similarity calculated is shown in Table I:

TABLE I: Similarity between Different Variants

File	Registry	Service	Process	System information	Network	Weighted similarity
35.2%	18%	40%	0%	20%	40%	27%

TABLE II: Similarity between Variants of the Same Malware

	File	Registry	Service	Process	System information	Network	Weighted similarity
Trojan.gh0st.a	85.2%	78%	82%	100%	86%	85%	86.17%
Trojan.gh0stb	90.6%	84%	74%	75%	91%	100%	86.16%
Trojan.gh0st.c	100%	100%	81%	100%	94%	86%	96.2%
Trojan.gh0st.e	100%	100%	100%	75%	63%	100%	89.4%
Trojan.gh0st.f	100%	100%	100%	100%	60%	100%	91%

We can see that the behavior of different variants show a very low degree of similarity, they could be easily distinguished by this method.

B. Similarity Detection between Variants of the Same Malware

We use malware gh0st as the experimental sample. Gh0st is a typical remote control Trojan. Because of its open source codes, there are a very large number of variants.

We detected five variants named Trojan.gh0st.a, Trojan.gh0st.b, Trojan.gh0st.c, Trojan.gh0st.e, Trojan.gh0st.f. The former three are changed from the source code, the latter two are processed by some packer tools. Similarity analysis of variants are shown in Table II.

To assess the validity and accuracy of the method, there are total 200 samples of 12 types downloaded from the online website anubis [14], which contains backdoors, remote Trojan , bot, remote downloader program, and cover the main types of malware.

In the experiment, the threshold P is set to 0.73. The classification results are shown in Fig. 3:

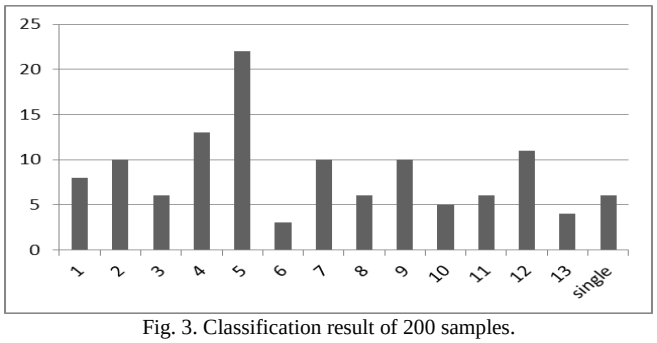

Fig. 3. Classification result of 200 samples.

We can see that 200 samples are divided into 13 categories. The classification results generally consistent with online result. Six samples among them are not able to be categorized. After manual analysis, we found that the reasons are two-folded. First, the sample program is damaged. Second, it is likely to have anti-analysis code in the programs, which could avoid detection and hide malicious behavior.

In the analysis of the performance, the total 200 samples running time is 80 minutes, while the average classification time of each variant is 40 seconds. The average consumed time of graph matching proposed by Yang Yi is 4 minutes. Our method could save more time than Yang's method.

V. Conclusions

In this paper, API calls and other key information of running malware are captured and extracted to establish dependency chain which could accurately describe the behavior of malware. After that, some confusion technologies commonly used by variants are eliminated. Finally, we use the weighted similarity of malware behavior to judge the similarity among different variants. Experimental results show that our method could classify the variants effectively.

In the future we will focus on two aspects. One is to enhance the detection of anti-judgment malware, and propose countermeasures for the existing anti-detection methods; the other is to add the data flow information in similarity analysis of malware, which will serve as an auxiliary of behavior analysis to further improve the classification accuracy of malware variants.

References

[1] Cert. [Online]. Available: http://www.cert.org.cn/publish/main/12/2014/20140328143100273454981/20140328143100273454981_.html.

[2] A. Henderson and A. Prakash, "Make it work, make it right, make it fast: Building a platform-neutral whole-system dynamic binary analysis platform[C]," *ISSTA*, 2014.

[3] M. E. Theodoor and Scholte, "A survey on automated dynamic malware-analysis techniques and tools," *ACM Computing Surveys*, vol. 44, no. 2, pp. 4-8, 2012.

[4] M. Lindorfer and C. Kolbitsch, "Detecting environment-sensitive malware," in *Proc. 14th International Symposium RAID*, Menlo Park, CA, USA, pp. 338-357, 2011.

[5] L.-F. Wu and M. Xu, "A novel malware variants detection method based on function-call graph," in *Proc. IEEE Joint International Computer Science and Information Technology Conference*, pp. 313-319, 2011.

[6] S. Forrest, "The evolution of system-call monitoring," *Computer Security Applications Conference*, no. 8, pp. 418-430, 2008.

[7] Ulrich Bayer, *Scalable, Behavior-Based Malware Clustering*, San Diego, California, USA: NDSS, 2009.

[8] Y. Yi *et al.*, "Dependency-based malware similarity comparison method," *Journal of Software*, vol. 10, no. 22, pp. 2439-2446, 2011.

[9] X.-Y. Sun, Y.-F. Zhu, and Q. Huang, "Study of malware detection based on interactive behavior," *Journal of Computer Applications*, vol. 30, no. 6, pp. 1489-1492, 2010.

[10] H. Lansheng and G. Kunlun, "Behavior detection of malware based on combination of API function and its parameters,"*Application Research of Computers*, vol. 30, no. 11, pp. 3407-3410, 2011.

[11] Bitbalze. [Online]. Available: http://bitblaze.cs.berkeley.edu/release/index.html

[12] D. Song, D. Brumle, H. Yin, and B. Blaze, "A new approach to computer security via binary analysis," in *Proc. the 4th International Conference on Information Systems Security*, pp. 1-25. Berlin, Heidelberg, 2008.

[13] J. Newsome and D. Song, *Dynamic Taint Analysis for Automatic Detection, Analysis, and Signature Generation of Exploits on Commodity Software*, NDSS 2005.

[14] Anubis. [Online]. Available: http://anubis.iseclab.org/

Guanghui Liang is a graduate student in State Key Laboratory of Mathematical Engineering and Advanced Computing, Zhengzhou 450002, China. His major is computer science and technology. He is a member of China Computer Federation. He is interested in information security and reverse engineering.

Jianmin Pang is a professor and doctoral supervisor in State Key Laboratory of Mathematical Engineering and Advanced Computing. His main research interests are focused on information security, binary translate, high performance computing.

Chao Dai is a Ph.D. student in State Key Laboratory of Mathematical Engineering and Advanced Computing. He is interested in web security, reverse engineering.