Semantic Malware Detection by Deploying Graph Mining

Fatemeh Karbalaie[1], Ashkan Sami[2] and Mansour Ahmadi[3]

[1]CSE&IT Department, Shiraz University
Shiraz, Iran

[2]CSE&IT Department, Shiraz University
Shiraz, Iran

[3]Young Researchers Club, Shiraz Branch, Islamic Azad University
Shiraz, Iran

Abstract

Today malware is a serious threat to our society. Several researchers are studying detection and mitigation of malware threats. On the other hand malware authors try to use obfuscation techniques for evading detection. Unfortunately usual approach (e.g., antivirus software) use signature based method which can easily be evaded. For addressing these shortcomings dynamic methods have been introduced. The aim of dynamic methods is to detect the semantic of malware family. Obfuscation of semantic based method is too difficult and results of these methods are promising. However deploying semantic based methods for real time detection have several complications. Current semantic methods are too time-consuming and usually need a robust virtual machine to obtain the behavior. In this paper we present an automatic detection method based on graph mining techniques with near optimal detection rate. That is 96.6% accuracy and only 3.4% false positive. In our method, first the malware is analyzed in a virtual machine environment to observe its semantic. A graph representation of malware behavior is constructed. The representation is based on relationships between system calls and allows rearrangement of system calls. Graph is used for representing the behavior of application because graph, especially labeled graph, can be used to model lots of complicated relation between data. At the next step we mine information graph and extract the most discriminative graphs that separate malware from benign. Finally, a classification method is used and the mentioned accuracy was obtained.

Keywords: *Semantic, Malware Detection, System call, frequent sub graph, labeled graph, subgraph isomorphism.*

1. Introduction

"Malware" is an abbreviation for 'malicious software' and is typically used as a catch-all term to refer to any software or program that damages computer systems or destroys valuable information stored in computers. Typical examples include viruses, worms, trojans, and spyware. Malware may be propagated using spam, may also be used to send spam, may take advantage of bugs, and may be used to mount DoS attacks. Recently the threat of malware has acquired an economic dimension as attackers benefit financially from compromised machines (e.g., by selling hosts as email relays to spammers) [1]. These considerations illustrate that addressing the problem of malware is necessary for improving computer security. Computer security is necessary to our society's critical infrastructure. Historically, detection tools such as signature based detection methods have performed poorly, particularly when facing previously unknown malware programs, novel variants of existing ones and polymorphic/ metamorphic malware. An important problem is that many of detection techniques rely on ineffective models. Ineffective models are models that do not capture natural properties of a malicious program and its actions but merely pick up artifacts of a specific malware instance. As a result, they can be easily evaded. Most of these models capture the sequence of system calls that a

specific malware program executes. The defect of these methods is that, when these system calls are independent, it is easy to change their order or add irrelevant calls, thus evading the captured sequence.

Today for above mentioned problems, researchers propose ways to capture the malicious behavior that characterizes a malware program. On one hand, some detectors [2, 3, 4] use sophisticated static analysis to identify the code that is semantically equivalent to a malware template. These actual semantic of program is unaffected by obfuscation, but at the other hand static analysis suffer from some limitation such as difficulty of static binary analysis, high cost of doing such analysis and the low speed in scanning large number of files [5]. In this paper we propose a novel and near optimal malware detection approach base on dynamic analysis. Also dynamic analysis techniques suffer from some limitation, such as necessity to run malware in virtual machine environment, but this limitation is the trade off for the good results dynamic analysis provides. Thus, we first generate effective model that cannot easily evaded by obfuscation. More accurately, we execute the malware program in a controlled environment and observe its interaction with the operating system.

In summary, our main contribution is to propose a framework based on graph mining approach. System calls are modeled as graphs, representing the program semantic. System calls were monitored because they are the primary interactions of malware with the operating system. Our algorithm infers the system-call graphs from execution traces, and then derives unique graphs that discriminate malware from benign. In other words, our method outperform all previous researches as we know and reached 96.6% detection rate with only 3.4% false positive. In contrast to use of graph mining techniques that are very time-consuming, our method does not take much time to perform. Unfortunately, it is observed that some researches have presented a very high accuracy. A close investigation of the paper reveals that the same data that was used for training were used to evaluate the accuracy. It is a very known error in evaluating the accuracy of a model called overfitting. Results of data mining models should be obtained based on cross validation to ensure evation from overfitting [6].

The rest of paper is organized as follows. Section II describes an overview of the system. Section III encompasses more detail about structure of our system. Section IV provides experimental results while Section V provides related work. Section VI concludes the paper.

2. System Overview

The goal of our system is to effectively and efficiently detect previously unseen and unknown malware. For this our detection method is based on the observation of the execution and monitoring the semantic of malware program in VM (Virtual Machine) environment. To model the program semantics and observe its security behavior, we used system call traces. System calls capture the interaction of program with its environment. Some malware use system calls for activating their malicious payload, so based on this fact; we can understand the malware author intent. In this paper we construct a graph based on system calls trace and our aim is to detect malware programs with high detection rate which outperform lots of previous research. An overview of system can be seen in figure 1.

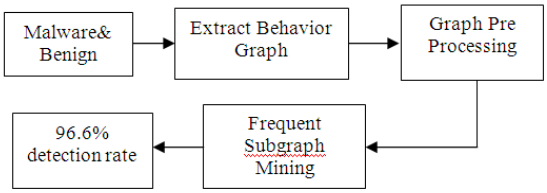

Fig. 1 System Overview

2.1 Modeling program semantic

Most of previous research focused on modeling program behavior by specifying permissible sequences of system calls [7, 8]. Malware authors have large degree of freedom in rearranging the code to achieve their goals. For example, it is very easy to reorder independent system calls or to add irrelevant calls. Thus, suspicious activity could not represent as system call sequence that we have observed. Instead a more flexible representation is required. In this paper the representation is based on relationship between system calls and allows rearrangement of system calls. Program semantic is represented as a semantic graph where nodes are (interesting) system calls. An edge is introduced from node x to node y when the return value of system call x is used as an input argument of system call y. Moreover, only a subset of system calls that are essential for

detecting malicious activity (detailed can be found in III) were considered.

2.2 Making detection more accurate

While constructing graph we take some considerations into account. That is, edges that its return value is 0x0 or 0x1 are not considered, because these return values means function failure or success. Any other return value is pointer value and is important for constructing the graph. This work makes the graph smaller and it makes any mining process quicker.

3. System Details

In this section, more detail about component of the detection system is provided. First our method to diagnose essential system calls for detecting malware is introduced. At the second step we discuss how to characterize program activity via semantic graphs. Then the techniques for extracting graphs automatically from observed traces are discussed. Finally we present our approach to detect graph of previously unknown malicious code.

3.1 Essential system call for detecting malicious behavior

Considering all DLL's of windows and all of system calls make graphs very large with lots of unnecessary edge for detection. To address this problem only 6 most important dll (including kernel32.dll, user32.dll, ws_s32.dll, advapi32.dll, wininet.dll and CreateProcess.dll) were used for malware analysis [9]. In addition, to find subset of system calls in these dll's that are used for malicious activities, data mining was used. Thus, 400 malware and 397 benign applications with all six considered dll's, monitoring all the API's were run, to diagnose which system call are more important for malware detection. Each malware and benign program ran for 3 second. All the system calls that each malware and benign called were collected. Then 10 fold cross validation with random forest classifier [10] were used to measure the accuracy rate of selected system call and the result get 89.5% detection rate. Next we used feature selection techniques to select most discriminative system call. On the other hand based on previous work [9] Malware's operations can be categorized as follows

File access
System information
Networking
Registry access
Processes

System information

It is more important for a malware to gather as much as possible of system information to insure that its software exploit is working. A software exploit is normally related to one specific operating system.

Registry access

In registry a lot of confidential information is stored, like keys or parameters for programs. It furthermore provides a mean to steer the processes that are launched during the machine's boot process. A lot of malware aim to be executed every time when the machine is started.

Processes

A running instance of an executable program is referred to as a process. A process consists of one or more threads, which is an atomic unit when it comes to processor time allocation. All threads that run in the context of a given process share the same address space, security context and environment variables [11].

Networking

The file I/O functions (CreateFile, CloseHandle, ReadFile, ReadFileEx, WriteFile and WriteFileEx) provide the basic interface for opening and closing a communication resource handle and for performing read and write operations. This means that when a process wishes to communicate through a communication device, it can perform a call to CreateFile specifying COM1 or LPT1 or another valid device name, and then write to the returned handle. The process can use the DeviceIoControl-call to send control codes to a device. Several types of malware perform operations against the local network and/or the Internet in order to infect other computers, receive updated malware code or interact with its creators.

We conclude that it is essential to consider all of system calls that are related to above operation for analyzing malware behavior [9]. We also added these system calls to our monitoring file.

3.2 Behavior Graphs: specifying program behavior

In general our graph is undirected labeled simple graph. Here is some preliminary concept that is essential for understanding our method [12].

Definition 1(Labeled Graphs) A labeled graph can be represented by a 4-tuple, G = (V, E, L, l), where

V is a set of vertices,
$E \subseteq V \times V$ is a set of edges
L is a set of labels,
$l : V \bigcup E \to L$, l is a function assigning labels to the vertices and the edges.
This definition can be generalized to include partially labeled graphs if the label set L includes an empty label. An example of undirected labeled graph can be shown in figure 2.

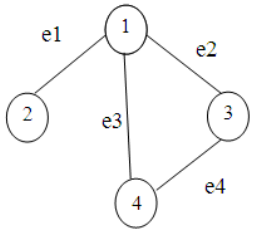

Fig. 2 An undirected labeled graph

Definition 2 (Subgraph, Induced subgraph)
A subgraph of a graph G, is a graph whose vertex set is a subset of that of G, and whose adjacency relation is a subset of that of G restricted to this subset.

Given a graph $G = (V(G), E(G), L(V(G)), L(E(G)))$, an induced subgraph of G, $G_s = (V(G_s), E(G_s), L(V(G_s)), L(E(G_s)))$, is a graph satisfying the following conditions.

$$V(G_s) \subset V(G), E(G_s) \subset E(G),$$
$$\forall u, v \in V(G_s), (u,v) \in E(G_s) \Leftrightarrow (u,v) \in E(G).$$

Where G_s is an induced subgraph of G, it is denoted as $G_s \subset G$ [13].

Definition 3 (Isomorphism, Automorphism, subgraph Isomorphism) An isomorphism is a bijective function $f : V(G) \to V(G')$, such that $\forall u \in V(G), l_G(u) = l_{G'}(f(u))$, and $\forall (u,v) \in E(G), (f(u), f(v)) \in E(G')$ and $l_G(u,v) = l_{G'}(f(u), f(v))$.

An automorphism of G is an isomorphism from G to G. A subgraph isomorphism from G to G' is an isomorphism from G to a subgraph of G'. If f is only injective, then G is monomorphic to G'.

Induced subgraph isomorphism can be considered as constrained subgraph isomorphism.

Definition 4 (Frequent Subgraph Mining) Given a graph dataset, GS = { $G_i \mid i = 0 \cdots n$ }, and a minimum support, minSup, let

$$\varsigma(g,G) = \begin{cases} 1 & \text{if g is isomorphic to a subgraph of G} \\ 0 & \text{if g is not isomorphic to any subgraph of G} \end{cases}$$

$$\sigma(g, GS) = \sum_{G_i \in GS} \varsigma(g, G_i) \quad (1)$$

$\sigma(g, GS)$ denotes the occurrence frequency of g in GS, i.e., the support of g in GS. Frequent Subgraph mining is to find every graph, g, such that $\sigma(g, GS)$ is greater than or equal to minSup. An example og graph mining approach can be show in figure 3.

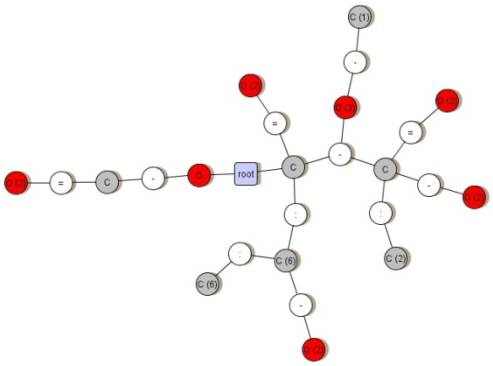

Fig. 3 An example of Graph mining

As a general data structure, graph, specially labeled graph, can be used to model many complicated relation among data. Labels of vertices and edges can represent different attribute of entities and relationship among them. In our setting, label of the nodes are system call

names and label of edges are number of unique values passed between the system calls. Table 1 shows two system calls that have an edge between each other.

Table 1. System calls and their parameters

System call name	Parameters	Return value
CreateFileW	lpFileName:0x00415F2C	**0x000025A8**
CloseHandle	hObject: **0x000025A8**	0x00000001

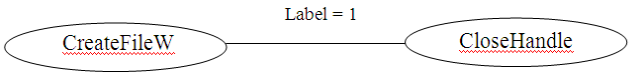

Fig. 4 Making graph based on table 1

If two system calls make an edge with only one memory address then the label of this edge is 1. If two or more unique addresses are used as an input of one API and return value of another, then the numbers of unique addresses are used as the edge label. Figure 4 shows the edge between two system calls with the label of edge and vertexes. Figure 4 is complete example for making edge from CreateFileW to CloseHandle system call based on execution traces. In this example 0x000025A8 is the return value of CreateFileW that is used as an input parameter of CloseHandle, so we draw an edge from CreateFileW to CloseHandle.

3.3 Why gSpan was Used

gSpan (graph-based Substructure pattern mining) is used for mining graphs that were generated based on description of previous step. It discovers frequent substructure without candidate generation. gSpan builds a new lexicographic order among graphs, and maps each graph to a unique minimum DFS code as its canonical label. Based on this lexico-graphic order, gSpan adopts the depth-first search strategy to mine frequent connected subgraphs efficiently. This algorithm has very good parallel and scale up properties and can incorporate constraints nicely in graph mining. It can find frequent subgraphs one by one, from small to long ones. Output of this algorithm is as below:

t # id * support
vertex-edge list
x graph_id list

where "id" is an integer, the serial number of the pattern , "support" is the absolute frequency of the graph pattern and "graph_id list" is a list of graphs that contain the pattern. We used this output for detecting subgraphs that discriminate malicious code from benign one.

3.4 Extract Dataset

gSpan was ran with different supports from 0.04 to 0.09. Each frequent subgraph in gSpan is used to be one feature in the final dataset. If one benign or malicious code includes the frequent subgraph, value of that feature is set to 1 otherwise 0 is assigned to the feature.

4. Evaluation

404 malware samples and 349 benign samples were collected from [11]. Our system has near optimal detection rate with very low overhead. In this section, system detection capability is presented.

Table 2. Detection Effectiveness of Our System

Name	**Number**
Constructor	188
Backdoor	162
Exploit	54

4.1 System Detection capability

To demonstrate our system detection capability behavior graphs for 3 popular malware families were generated. Table 2 shows an overview of these families and their counts. These malware families were selected because they are very popular according to lists compiled by anti-virus reports [14]. Some of the families use code polymorphism or metamorphism. It makes the detection harder for signature-based scanners. For each malware family more than 50 samples were selected randomly from our database. Specifically samples that did not modify the file system were not used. A single-path dynamic analysis of the samples for 120 second was performed to collect the execution trace. This time is selected because two minutes is generally enough time for most malware to execute its immediate payload, if it has one [15]. While some malware samples do not perform any malicious behavior in this period, these samples usually wait for some external trigger to execute their payload (e.g. network or system environment), and will not perform any behavior if left to execute without further action [15]. Each benign sample also ran for 120 second. The samples were then used for extracting behavior graph. All of the malware and benign graphs used as an input of gSpan to obtain frequent graphs. Because of strong preprocessing step for constructing graph, resulted graphs were very suitable for using graph

mining technique (in terms of size of graph) on the other hand these graphs include all of the information that may be needed for improving detection accuracy. gSpan is used with different support from 0.04 to 0.9 to evaluate different result of this tool. Count of frequent subgraph for each support is in table 3.

Table 3. Detection Effectiveness of Our System

Support	# of frequent subgraph
0.04	4187
0.05	1188
0.06	784
0.07	579
0.08	501
0.09	471

Support 0.9 considered as maximum support because for supports of more than 9 the output includes only graphs with one vertex that is not suitable for our purpose. Each frequent subgraph used as a feature for making final dataset. At the final step, 10-fold cross validation with random forest classifier was used to evaluate the detection rate of the system. Results are shown in Table 4.

Table 4. Detection Effectiveness of Our System

S	Recall	Precision	Fp	F-Measure	ROC Area
0.04	89.9	88.2	10.1	89.1	95.3
0.05	88.7	87.4	11.3	88	95.2
0.06	89.9	87.8	10.1	88.8	95.5
0.07	94.6	96.3	5.4	95.4	98.8
0.08	96.1	98.7	3.9	97.4	98.7
0.09	96.6	98.7	3.4	97.6	99

As shown in table 4, 96.6 percent detection rate with 3.4% false positive was obtained based on 0.09 support. Overall, an average 92.6% detection rate with 7.36% false positive was obtained. Figure 5 illustrates the relationship between detection rate and support, while figure 6 illustrates the relationship between support and false positive.

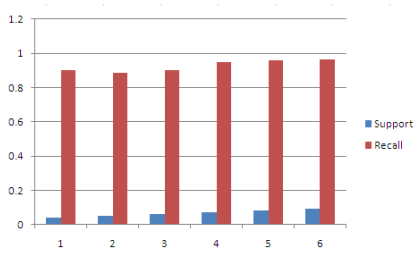

Fig. 5 Support and detection rate relationship

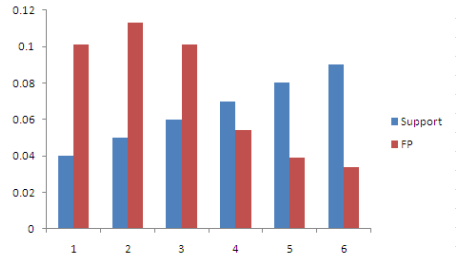

Fig. 6 Support and false positive relationship

5. Related Works

Even though behavioral detection seems a recent trend, in antivirus products as well as in virology research, its principles are not really new. In 1986, Cohen [13, 16] already established a basis for behavioral detection within his first formal works. At the other hand, there is a large number of previous works that studies the behavior [17, 18, 19] of different types of malware. Reik et al. proposed classification technique that uses support vector machines to produce class label for unknown malware [20]. Kolbitsch et al. proposed an effective and efficient method for detecting malware behavior at the end host. Their behavior graph was almost like our graph but their graph is more complicated than ours and also they do graph matching as detection method [21]. Egel et al. describe a behavioral specification of browser-based spyware based on taint-tracking [12], and panorama uses whole-system taint analysis in a similar vein to detect more general classes of spyware. Fredrikson and Jha et al. were automated clustering efforts to create initial sample partition for behavior extraction [15]. They demonstrated a technique for producing behavior graphs with 86% detection rate on new, unknown malware, with 0 false positive but they used 912 malware samples and only 49 benign programs for analyzing and these result cannot generalized to other setting. They used input and output parameter for construct graph, this makes graph

larger. We just consider the return value of system cal and input value to construct graph. Consequently our graph is simpler. We analyzed 404 malware and 349 benign program and we used random forest classification method to evaluate the detection rate of new, unknown malware. Our result shows 96.6% detection rate with only 3.4% false positive for new, unknown malware.

6. Conclusion

Malware detection is a tedious and complicated chore. We propose a method for detecting malicious code from benign based on graph mining techniques that resulted 96.6% detection rate with only 3.4 false positives. Graph is used for representing the behavior of application because graph, especially labeled graph, can be used to model lots of complicated relation between data. At the next step we mined information graph and extracted the most discriminative graphs that separate malware from benign.

Refrences

[1] M. Christodorescu, S. Jha, and C. Kruegel. "Mining specification of malicious behavior". In ESEC/FSE. 2007.
[2] M. Christodore Su, AND S. Jha. "Static Analysis of Executables to Detect Malicious Patterns". In Usenix Security Symposium. 2003.
[3] M. Christodorescu, S. Jha, S. Seshia, D. Song, AND R. Bryant. "Semantics-Aware Malware Detection". In IEEE Symposium on Security and Privacy. 2005.
[4] C. Kruegel, W. Robertson, and G. Vigna. "Detecting Kernel-Level Rootkits Through Binary Analysis". In Annual Computer Security Applications Conference (ACSAC). 2004.
[5] A. Moser, C. Kruegel, and E. Kirda. "Limits of Static Analysis for Malware Detection". In 23rd Annual Computer Security Applications Conference (ACSAC). 2007.
[6] Y. Ye, D. Wang, T. Li, and D. Ye. An intelligent pe-malware detection system based on association mining. In Journal in Computer Virology, 2008.
[7] S. Forrest, S. Hofmeyr, A. Somayaji, AND T. Longstaff. "A Sense of Self for Unix Processes". In IEEE Symposium on Security and Privacy. 1996.
[8] D. Wagner, and D. Dean. "Intrusion Detection via Static Analysis". In IEEE Symposium on Security and Privacy. 2001.
[9] http://msdn.microsoft.com/
[10] L. Breiman. " Random Forests ". Kluwer Academic Publishers. Manufactured in The Netherlands. 2001.
[11] A. Sami, B. Yadegari and H. Rahimi, N. Peiravian, S. Hashemi, A. Hamze. "Malware Detection Based on Mining API Calls". SAC'10 March 22-26, 2010, Sierre, Switzerland.
[12] X. Yan and J. Han. "gSpan: Graph-Based Substructure Pattern Mining". IEEE International Conference. 2002.
[13] A. Inokuchi, T. Washio, and H. Motoda. "Frequent Substructure from Graph Data". PKDD2000, Sept. 13-16, 2000, Lyon, France.
[14] F. Cohen. "Computer viruses". Ph.D. thesis, University of South California (1986)
[15] M. Fredrikson and S. Jha, M. Christodorescu and R. Sailer, And X. Yan. "Synthesizing Near-Optimal Malware Specification from Suspicious Behaviors". IEE Symposium on Security and Privacy. 2010, pp. 45-60.
[16] F.B. Cohen. "Computer viruses: Theory and experiments". Comput. Secur. 6(1), 22–35 (1987)
[17] M. Polychronakis, P. Mavrommatis, and N. Provos. "Ghost turns Zombie: Exploring the Life Cycle of Web-based Malware." InUsenixWorkshoponLarge Scale Exploits and Emergent Threats (LEET). 2008.
[18] M. , Rajab, J. Zarfoss, F. Monrose, and A. Terzis. "A Multi faceted Approach to Understanding the Botnet Phenomenon". In Internet Measurement Conference(IMC). 2006.
[19] S. Small, J. mason, F. Monrose, N. Provos, and A. Stubblefield. "To Catch A Predator: A Natural Language Approach for Eliciting Malicious Payloads". In 17th Usenix Security Symposium, 2008.
[20] K. Rieck, T. Holz, C. Willems, P. Dussel, and P. Laskov, "Learning and classification of malware behavior," in Proceedings of the 5th Conference on Detection of Intrusions and Malware and Vulnerability Assessment (DIMVA'08). Springer, 2008, pp. 108 – 125.
[21] C. Kolbitsch, P. Milani Comparetti, C. Kruegel, E. Kirda, X. Zhou, and X. Wang. "Effective and Efficient Malware Detection at the End Host". Secure Systems Lab [TU Vienna, Institute Eurecom Sophia Antipolis, UC Santa Barbara]Indiana University at Bloomington. 2009.

Fatemeh Karbalaie has obtained her B.S degree in Computer Science in 2007 at Isfahan Payamenoor University. Since 2009, she is a master student of Computer Engineering at Shiraz University. Her research interests include security and data mining.

Dr. Ashkan Sami has obtained his B.S. from Virginia Tech; Blacksburg, VA; U.S.A., M.S. from Shiraz University; Iran and Ph.D. from Tohoku University; Japan. He is interested in Data Mining, Software Quality and Security. Ashkan has been a member of technical committee of several international conferences like PAKDD, ADMA, HumanCon, and Future Tech and has more than 40 conference paper and nearly 10 journal papers. He is an associate member of IEEE and was among the founding members of Shiraz University CERT.

Mansour Ahmadi has obtained his B.S. in Applied Mathematics from Sistan Baloochestan University; Iran and his M.S in software engineering from Islamic Azad university, Arak. He Worked on malware detection as his M.S. thesis under supervision of Dr. Sami and is currently a researcher in Shiraz University CERT.