Fast Malware Classification by Automated Behavioral Graph Matching

Younghee Park, Douglas Reeves, Vikram Mulukutla, Balaji Sundaravel
Cyber Defense Laboratory
Department of Computer Science Department
NC State University, Raleigh, NC, USA
{ypark3,reeves,vmuluku,bsundar}@ncsu.edu

ABSTRACT

Malicious software (malware) is a serious problem in the Internet. Malware classification is useful for detection and analysis of new threats for which signatures are not available, or possible (due to polymorphism). This paper proposes a new malware classification method based on maximal common subgraph detection. A behavior graph is obtained by capturing system calls during the execution (in a sandboxed environment) of the suspicious software. The method has been implemented and tested on a set of 300 malware instances in 6 families. Results demonstrate the method effectively groups the malware instances, compared with previous methods of classification, is fast, and has a low false positive rate when presented with benign software.

Categories and Subject Descriptors

K.4.1 [**Computers and Society**]; D.2.8 [**Software Engineering**]: Invasive Software

General Terms

Security

Keywords

Malware, Graph Theory, Dynamic Analysis, Virtualization

1. INTRODUCTION

Malware continues to be a serious problem despite many attempts to detect and prevent it. The creators of malware continually develop new methods of evading detection by signatures, such as the use of encryption, packing, or obfuscation. A large number of new malware instances are reported each day; for instance, around half a million malware samples were recorded in the second half of 2007 [6]. The most challenging problem is how to efficiently and effectively detect new malware. Classification of malware is the first step in analyzing and detecting new malware instances [1, 2, 15, 10].

Signature-based detection is the most widely used approach, but is generally regarded as ineffective against attacks like code polymorphism and metamorphism [8, 17, 18]. Furthermore, commercial anti-virus software is not very good at classifying malware [1]. To overcome the limitation of signatures, researchers have turned to analysis of malware behavior. This analysis may be static (using disassembly of the binary) [18, 8, 17, 11] or dynamic (executing the suspicious code in a protected environment) [9, 2]. Run-time (dynamic) analysis of malware behavior is the most powerful technique, but previous methods have used expensive processing steps such as memory tracing (taint analysis), symbolic execution, and program slicing.

In this paper, we propose a new malware classification method. This method first captures the system calls executed by the malware, along with parameter values of those calls. Then, the similarity of the program's behavior to that of known malware instances is computed. A family shares the same or similar behaviors, so each family has a tendency to show similar characteristics [16], such as file system modification, spawning new processes, network connectivity checking, registry modification, network communication, etc.

To define the behavior of one program, a directed (behavioral) graph is generated from the system call traces. System calls have long been used to analyze malware, and the arguments of these calls indicate their relationships [1, 9, 2]. To compare two programs, the maximal common subgraph between the two behavioral graphs is computed [3]. The size of this subgraph is a measure of the similarity of the two programs. Based on this metric, an automated classification technique, called FACT, is proposed.

Experimental results show that the programs from the same malware family are similar according to this metric. Malicious behaviors coming from malware are different from non-malicious programs' behaviors. Generation of the behavioral graphs is fast, and does not require expensive techniques such as taint analysis and symbolic execution.

The paper is organized as follows. Section 2 presents a new classification technique based on maximal common subgraphs. Section 3 shows experimental results. Related work is described in section 4. Finally, conclusions and future work are presented in section 5.

2. A NEW APPROACH

This section proposes a new malware classification technique based on the maximal common subgraph detection problem. By defining similarity between two behavioral graphs based on system call traces, the proposed method

This material is based upon work supported by the National Science Foundation under Grant No. 0831081.

can identify and classify malware with similar behaviors.

2.1 Behavioral Graph Generation

In this section, we first define a behavioral graph to represent a program run. The behavioral graph corresponds to a system call graph for the behavior of the suspicious binary, called a dynamic system call dependence(DSCD) graph. The DSCD graph consists of a 4-tuple directed graph $G = (N, E, \mu, \upsilon)$. N is a set of nodes (i.e. a set of system calls), and each node $s \in N$ corresponds to a system call in the program execution. $E \subseteq N \times N$ is the set of dependence edges, and each edge $s_1 \rightarrow s_2 \in E$ corresponds to the dependence between the two arguments, arg_1 and arg_2, of the two system calls, s_1 and s_2. μ is a node labeling function $\mu : N \rightarrow L_N$ assigning system calls to nodes. υ is an edge labeling function $\upsilon : E \rightarrow L_E$ that describes the dependence between two system calls, according to their arguments.

A behavioral graph for a program is extracted when the program is executed. The graph represents a set of system call sequences holding their dependences. In detail, first, the system call traces are collected while running a program in a virtualized domain. The system calls and their arguments are obtained by intercepting every SYSENTER instruction performed in the domain. Then, we find a sequence of dependent system calls from the traces by matching their arguments. The matching is performed by comparing both the type and value defined in the Windows system call table [14]. The arguments of interest are those involving handles, which denote various OS objects such as files, processes, tokens, ports, threads, etc. A system call dependence is determined when the output handle for one system call is used as an input for another system call or if the handle value for subsequent system calls is the same. For each dependence, the system calls are represented in nodes of a graph, and a directed edge between them is added.

2.2 Similarity Measurement

In this section, the maximal common subgraph (MCS) [3] is defined to compute the similarity of DSCD graphs. Let $G_1 = (N_1, E_1, \mu_1, \upsilon_1, \lambda_1)$ and $G_2 = (N_2, E_2, \mu_2, \upsilon_2, \lambda_2)$, be two DSCD graphs. A graph $G'=(N', E', \mu', \upsilon', \lambda')$ is called a *common subgraph* of G_1 and G_2 if there exists a subgraph isomorphism from G' to G_1, and from G' to G_2 [13, 3]. G' of G_1 and G_2 is called a *maximal common subgraph (MCS)* if there exists no other common subgraph of G_1 and G_2 that has more nodes than G'.

Given two behavioral graphs G_1 and G_2, the similarity between them can be measured by a new graph edit distance. The new graph edit distance is based on the maximal common subgraph of two graphs, called MCS distance [3]. The distance(Δ) or *dissimilarity* of two graphs is defined as $\Delta(G_1, G_2) = 1 - \frac{|G_{MCS}|}{max(|G_1|, |G_2|)}$. $|G|$ is the number of nodes in a graph. The distance measurement has the property that $0 \leq \Delta(G_1, G_2) \leq 1$, and has other properties required to be a valid metric.

In the following, if two DSCD graphs have a smaller distance than a pre-defined threshold θ, the two malware instances are regarded as having similar behaviors. This metric does not require full graph isomorphism, but rather is a measure of similarity. There are many polynomial time algorithms to find MCS in the literature [5].

Based on the distance measurement, a Feedb$A$ck-based Classification Technique (called FACT) is proposed as follows; (1) A sample program P_1 is randomly chosen from the training set(T). The first classified group(C_1) is generated if the dissimilarity(Δ) between the binary(B) in T and P is less than a threshold(θ). (2) A sample program P_2 is randomly chosen again from T. The second classified group(C_2) is generated if Δ between B in T and P_2 is less than θ. (3) For the two groups, each family can be finally created if Δ between B_1 in C_1 and B_2 in C_2 is greater than θ. Otherwise, the binaries in the two groups should be back in T. Recursively, from the first step to the third step, the procedure should be continuously performed until no malware remains or any samples are not classified anymore. Note that the size of each classified group should be greater than another predefined threshold(MIN) to prevent the creation of many different groups. For instance, the size of C_1 is less than MIN, the group should come back in T. In addition, the third step makes sure that the classified groups are actually different. In other words, if the condition above is satisfied, B_1 in C_1 is classified in one family F_1. B_2 in C_2 satisfying the condition is classified in another family F_2. Otherwise, the binaries in the classified groups should be back in T again.

3. EVALUATION

The proposed method was evaluated to show its feasibility for malware classification. After presenting the experimental method, the results shown are: similarity between malware in the same family, dissimilarity between malicious programs and non-malicious programs, and the results of malware classification using FACT.

3.1 Experimental Setup

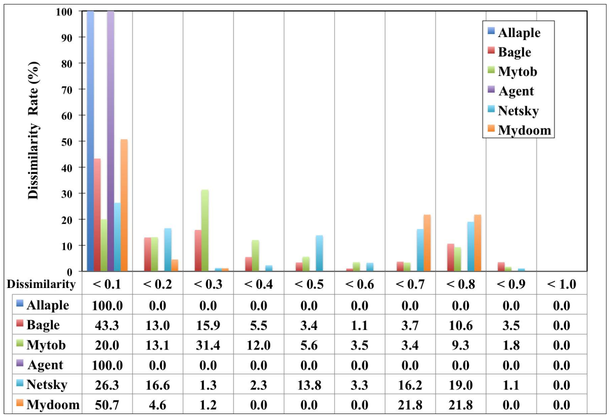
A prototype of the proposed method was implemented on Linux, using Ether [7], which is a malware analysis framework that leverages hardware virtualization extensions (specifically Intel VT) to remain transparent to malicious software. By using Ether, native system calls were intercepted for Windows XP SP2, the guest OS system. Graghviz [12] was used to display DSCD graphs constructed from the resulting system call traces.

To evaluate the proposed method, instances of malware from 6 different malware families were taken from the training set of [9]. The families were: Allaple (exploit-based worm), Agent (Trojan), Bagle, Mytob, Netsky, and Mydoom (mass-mailing worm) family. Each of the families consisted of 50 different instances, and most of them were packed. In addition, 80 benign Windows applications (such as Internet Explorer, Winzip, Realplayer, Adobe, Putty, Bittorrent, etc.) were used for comparison. Whenever a program instance ran on the guest operating system, Ether provided the system call traces with the arguments [1]. A behavioral graph was constructed for each program, as defined in section 2.1. All possible combinations were considered without repetition for each family and a set of benign applications. For instance, $\binom{50}{2}$ dissimilarities were computed for 50 worms for Bagle family. For classification using FACT, the threshold (limit of dissimilarity, θ) was set to 0.3, and the minimum group size MIN was set to 5; these values were determined by experimentation, as shown below.

[1] We could not obtain a complete trace for each binary in the Agent family. Most instances in the family crashed the guest OS, a problem shared with other methods of dynamic analysis of malware. However, even partial traces were sufficient for classification purposes.

3.2 Evaluation

Figure 1 shows that each family had highly similar behaviors according to the proposed distance metric. For example, the distance metric for all instances (100%) in the Allaple and Agent families was very low (less than 0.1). 77.72% of the binaries in Bagle family showed distances less than 0.3, while the Mytob family had 76.48% of the binaries with distances less than 0.3 dissimilarity. The results in this figure considered all combinations of the malware instances. Figure 2 shows results of comparing one instance of each family with all other instances of malware. Based on the results, FACT can correctly classifies the behavior of randomly selected instances.



Dissimilarity	< 0.1	< 0.2	< 0.3	< 0.4	< 0.5	< 0.6	< 0.7	< 0.8	< 0.9	< 1.0
Allaple	100.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Bagle	43.3	13.0	15.9	5.5	3.4	1.1	3.7	10.6	3.5	0.0
Mytob	20.0	13.1	31.4	12.0	5.6	3.5	3.4	9.3	1.8	0.0
Agent	100.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Netsky	26.3	16.6	1.3	2.3	13.8	3.3	16.2	19.0	1.1	0.0
Mydoom	50.7	4.6	1.2	0.0	0.0	0.0	21.8	21.8	0.0	0.0

Figure 1: Dissimilarity rates for each family in the training set. X-axis indicates the dissimilarity range (e.g. ”< 0.1“ means ”$0.0 \leq \theta < 0.1$“). Y-axis is a percentage of dissimilarity for each malware family in the training set.

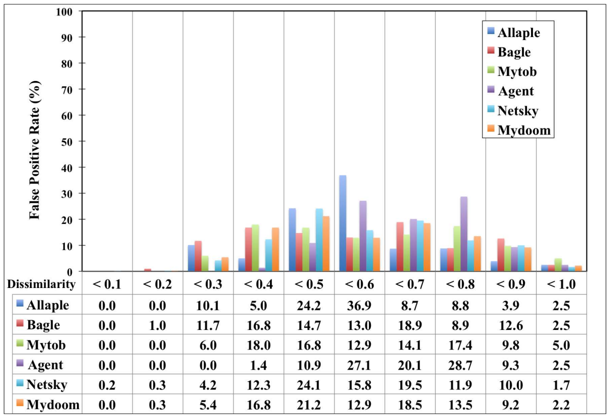
However, in case of Mydoom family, two distinct behaviors could be classified. While 56% of the binaries in the Mydoom family had distance less than 0.2, the rest had distances from 0.6 to 0.8. In addition, binaries in the Netsky family showed multiple behaviors. Only 46.48% of them had distances less than 0.3. In the results of [9], the effectiveness for Netsky family showed a low detection rate, indicating members of this family have distinctly different behaviors. This was further investigated using our method. As shown in Figure 4, most binaries in the set for each of the two families, Mydoom and Netsky, were classified into two different groups, Group 4 and Group 5. This result can also be seen in the results of classifying individual instances, as in Figure 2.

Figure 3 shows the distance between the malicious programs and benign (non-malicious) Windows programs, termed the false positive rate. Around 10% of the non-malicious programs had distances of less than 0.3 with members of the Allaple and Bagle families. The binaries in Mytob, Agent, Netsky, and Mydoom families showed around 5% false positive rates.

Figure 4 shows the results for FACT. For this experiment, all binaries in the training set are considered as unclassified malware. The experiment was run 7 times; only one run is shown (other results were similar). For this run, a random sample was differently selected, and 6 different groups were classified according to the proposed method. The Agent and Allaple families were correctly grouped for each run with high accuracy. Therefore, they should be considered as distinct families in the training set. Also, 86% of binaries in the Bagle family were grouped in the same family. However, only 60% of the binaries in Netsky were classified as having the same behaviors. For Mytob and Mydoom families, around 50% of them were grouped together. The results for the proposed method are consistent with the results of Figure 1. Further, malware families related to mass-mailing worms showed similar behaviors to each other. Only 5.3% of the total training set(300 samples) was not classified into any group. Therefore, based on the MCS distance, FACT can classify malware into groups having similar behaviors. In last, the results above are similar to the scanning result by McAfee, an anti-virus software. The anti-virus software labeled 15 binaries(30%) in the Mydoom family with a different name. Around 20% binaries for each of Mytob and Netsky family were classified as the different families.

Dissimilarity	Allaple.a	Bagle.b	Mytob.c	Agent.d	Netsky.e	Mydoom.f
$0.0 \leq \Delta < 0.1$	19	31	2	21	15	15
$0.1 \leq \Delta < 0.2$	30	10	13	28	14	18
$0.2 \leq \Delta < 0.3$	0	2	27	0	1	0
$0.3 \leq \Delta < 0.4$	0	1	2	0	0	0
$0.4 \leq \Delta < 0.5$	0	0	1	0	5	0
$0.5 \leq \Delta < 0.6$	0	0	0	0	0	0
$0.6 \leq \Delta < 0.7$	0	1	1	0	12	0
$0.7 \leq \Delta < 0.8$	0	3	3	0	2	16
$0.8 \leq \Delta < 0.9$	0	1	0	0	0	0
$0.9 \leq \Delta \leq 1.0$	0	0	0	0	0	0
Total #	49	49	49	49	49	49

Figure 2: Dissimilarity with one malicious program for each of the families. Each entry indicates the number of the binaries that is in the dissimilarity(Δ) range for each family.

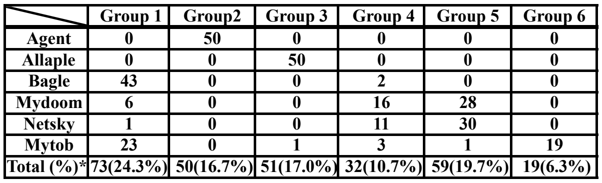


Dissimilarity	< 0.1	< 0.2	< 0.3	< 0.4	< 0.5	< 0.6	< 0.7	< 0.8	< 0.9	< 1.0
Allaple	0.0	0.0	10.1	5.0	24.2	36.9	8.7	8.8	3.9	2.5
Bagle	0.0	1.0	11.7	16.8	14.7	13.0	18.9	8.9	12.6	2.5
Mytob	0.0	0.0	6.0	18.0	16.8	12.9	14.1	17.4	9.8	5.0
Agent	0.0	0.0	0.0	1.4	10.9	27.1	20.1	28.7	9.3	2.5
Netsky	0.2	0.3	4.2	12.3	24.1	15.8	19.5	11.9	10.0	1.7
Mydoom	0.0	0.3	5.4	16.8	21.2	12.9	18.5	13.5	9.2	2.2

Figure 3: False positive rates between malware programs and benign applications. X-axis indicates the dissimilarity range. Y-axis is a percentage for the 80 non-malicious programs matching with the malicious programs in the training set.

4. RELATED WORK

Many methods of detection and classification of malware have been presented. Static analysis of binaries, depending on byte sequences, has long been used. These techniques are not effective against malware that employs concealment techniques, such as code polymorphism and obfuscation. Disassembly and static analysis of binaries has also been



	Group 1	Group2	Group 3	Group 4	Group 5	Group 6
Agent	0	50	0	0	0	0
Allaple	0	0	50	0	0	0
Bagle	43	0	0	2	0	0
Mydoom	6	0	0	16	28	0
Netsky	1	0	0	11	30	0
Mytob	23	0	1	3	1	19
Total (%)*	73(24.3%)	50(16.7%)	51(17.0%)	32(10.7%)	59(19.7%)	19(6.3%)

Figure 4: Results of Feedback-based Classification Technique(FACT). Each group can be a family. Each entry is the number of the binaries belonging to the family for each group. The symbol (*) indicates the group size over the training set with a percentage.

proposed, using control and data flow graph information. For instance, a malware indexing technique [8] classified a large number of malware samples. However, it needs full disassembly for analysis, and it can have different results according to the accuracy of unpacking malware that is packed. Another example is MetaAware [18], which measured the similarity between two malware binaries based on system calls resulting from disassembly. Kruegel et al. [11] detected polymorphic variants of malicious programs having the same control flow graph.

Due to the limitations of static analysis, dynamic (runtime) detection and classification of malware is gaining interest. The method is this paper is closely related to such work. Kolbitsch et al. [9] proposed a new technique to extract a behavioral graph for malware detection. This method uses dynamic program slicing to find the dependence between arguments of system calls. It aims to find indirect dependence through investigating memory log. Our method, in contrast, uses only structural information of the behavioral graph, rather than a memory log, avoiding the overhead of taint tracing. In addition, Bayer et. al. [2] suggested behavior-based malware clustering with high scalability by abstracting runtime behaviors into OS objects and operations. They used Jaccard Index and locality sensitive hashing technique to cluster similar behaviors. Bailey [1] proposed an automated malware classification method through profiling non-transient states of the system. Malspecs [4] suggested malware classification by extracting malicious behaviors that showed contrasting behaviors with benign software.

5. CONCLUSION AND FUTURE WORK

The proposed method presented a new malware classification scheme based on capturing system calls during execution, constructing a graph for system calls, and computing the similarity between the graphs of different programs. The maximal common subgraph is used for this purpose, followed by an automated classification scheme.

In future work, to improve accuracy, we will consider an attribute for each node and each edge. As each of the nodes and edges has a unique attribute, each binary can be indicated by a distinct behavioral graph. Furthermore, by mining a set of common behavioral graphs from malware, the common behavior can be generally used as a standard to classify malware with high speed. Finally, we will also consider how to defend against distortions of the graphs by system call injection attacks.

6. REFERENCES

[1] M. Bailey, J. Oberheide, J. Andersen, Z. M. Mao, and F. J. andJose Nazario. Automated classification and analysis of internet malware. In *Proceedings of 10th International Symposium in Recent Advances in Intrusion Detection(RAID)*, volume 4637 of *Lecture Notes in Computer Science*, pages 178–197, Gold Goast, Australia, September 2007. Springer.

[2] U. Bayer, P. Milani Comparetti, C. Hlauscheck, C. Kruegel, and E. Kirda. Scalable, Behavior-Based Malware Clustering. In *16th Symposium on Network and Distributed System Security (NDSS)*, 2009.

[3] H. Bunke and K. Shearer. A graph distance metric based on the maximal common subgraph. pages 255–259.

[4] M. Christodorescu, S. Jha, and C. Kruegel. Mining specifications of malicious behavior. In *Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering(ESEC-FSE '07)*, pages 5–14, New York, NY, USA, 2007. ACM.

[5] D. Conte, P. Foggia, and M. Vento. Challenging complexity of maximum common subgraph detection algorithms: A performance analysis of three algorithms on a wide database of graphs. *J. Graph Algorithms Appl.*, 11(1):99–143, 2007.

[6] S. Corp. Symantec global internet security threat report, April 2008. http://www.symantec.com/.

[7] A. Dinaburg, P. Royal, M. I. Sharif, and W. Lee. Ether: malware analysis via hardware virtualization extensions. In *ACM Conference on Computer and Communications Security*, pages 51–62, 2008.

[8] X. Hu, T.-c. Chiueh, and K. G. Shin. Large-scale malware indexing using function-call graphs. In *Proceedings of the 16th ACM conference on Computer and communications security(CCS'09)*, pages 611–620, Chicago, Illinois, USA, 2009. ACM.

[9] C. Kolbitsch, P. Milani Comparetti, C. Kruegel, E. Kirda, X. Zhou, and X. Wang. Effective and Efficient Malware Detection at the End Host. In *18th Usenix Security Symposium*, Montreal, Canada, August 2009.

[10] J. Z. Kolter and M. A. Maloof. Learning to detect and classify malicious executables in the wild. *Journal of Machine Learning Research*, 7:2721–2744, 2006.

[11] C. Krugel, E. Kirda, D. Mutz, W. K. Robertson, and G. Vigna. Polymorphic worm detection using structural information of executables. In *RAID*, pages 207–226, 2005.

[12] A. . T. R. Labs. Graphviz - graph visualization software. http://graphviz.org/.

[13] B. T. Messmer and H. Bunke. A new algorithm for error-tolerant subgraph isomorphism detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(5):493–504, 1998.

[14] T. M. Project. Windows system call table. http://www.metasploit.com/users/opcode/syscalls.html.

[15] K. Rieck, T. Holz, C. Willems, P. Dussel, and P. Laskov. Learning and classification of malware behavior. In *Proceedings of the 5th international conference on Detection of Intrusions and Malware, and Vulnerability Assessment(DIMVA)*, pages 108–125, Berlin, Heidelberg, 2008. Springer-Verlag.

[16] E. Stinson and J. C. Mitchell. Characterizing bots' remote control behavior. In *Proceedings of the 4th international conference on Detection of Intrusions and Malware, and Vulnerability Assessment(DIMVA '07)*, pages 89–108, Berlin, Heidelberg, 2007. Springer-Verlag.

[17] D. Wagner and R. Dean. Intrusion Detection via Static Analysis. In *Proceedings 2001 IEEE Symposium on Security and Privacy(S&P)*, pages 156–168, Oakland, CA, USA, May 2001.

[18] Q. Zhang and D. S. Reeves. Metaaware: Identifying metamorphic malware. In *23rd Annual Computer Security Applications Conference (ACSAC)*.