Scalable Malware Clustering Through Coarse-Grained Behavior Modeling

Mahinthan Chandramohan, Hee Beng Kuan Tan, Lwin Khin Shar

School of Electrical and Electronic Engineering
Block S2, Nanyang Technological University
Nanyang Avenue,Singapore 639798.
{mahintha001, ibktan, shar0035}@e.ntu.edu.sg

ABSTRACT

Anti-malware vendors receive several thousand new malware (malicious software) variants per day. Due to large volume of malware samples, it has become extremely important to group them based on their malicious characteristics. Grouping of malware variants that exhibit similar behavior helps to generate malware signatures more efficiently. Unfortunately, exponential growth of new malware variants and huge-dimensional feature space, as used in existing approaches, make the clustering task very challenging and difficult to scale. Furthermore, malware behavior modeling techniques proposed in the literature do not scale well, where malware feature space grows in proportion with the number of samples under examination.

In this paper, we propose a scalable malware behavior modeling technique that models the interactions between malware and sensitive system resources in a coarse-grained manner. Coarse-grained behavior modeling enables us to generate malware feature space that does not grow in proportion with the number of samples under examination. A preliminary study shows that our approach generates 289 times less malware features and yet improves the average clustering accuracy by 6.20% comparing to a state-of-the-art malware clustering technique.

Categories and Subject Descriptors

D.2.8 [**Operating System**]: Security and Protection

General Terms

Security, Malware clustering, Malware behavior modeling

Keywords

Malware clustering, Coarse-grained behavior modeling

1. INTRODUCTION

Despite the common use and widespread availability of various anti-virus tools, the growth of malware is phenomenal. According to Symantec, an anti-virus vendor, more than 286,000,000 new malware variants were detected in 2010 [2]. If we further zoom-in, on average around 55,000 new malware samples were reported per day during September, 2011 [4]. Identification of malware variants significantly improves the signature detection and reduces the size of malware signature database. Thus, it has become crucial for anti-virus vendors to analyze and cluster malware samples based on their malicious behavior. Manual inspection and clustering of malware samples are out of question due to their dramatic growth in the recent past [2, 4]. Therefore, it has to be automated [3, 4].

Malware clustering has been studied by several researchers in the past [5, 6, 7, 8]. Malware behavior modeling techniques, proposed in the literature, generally use n-gram analysis, non-transient state changes, system call trace analysis, taint analysis, system call dependency graphs and control flow graphs to extract malicious features. Given the number of new malware variants reported per day, "scalability" is a major problem in existing feature extraction techniques, where malware feature space grows in proportion with the number of samples under examination. This makes malware clustering a challenging task.

In this paper, we introduce a simple, yet efficient coarse-grained malware behavior modeling technique that captures the interactions between malware and sensitive system resources at a higher level of abstraction. Coarse-grained behavior modeling increases the applicability of our approach across all malware classes and importantly, enables us to generate malware feature space that does not grows in proportion with the number of samples under examination. Higher level of abstraction (i.e. agnostic to underlying low-level instructions) improves the robustness of our approach against basic obfuscation techniques [9]. Furthermore, abstracting away the malware and system specific details allows us to reduce large amount of noise (i.e. features with low information gain) in the feature space.

The paper is organized as follows. Section 2 describes our coarse-grained malware behavior modeling. Section 3 explains the scalability of clustering technique. Section 4 discusses our experiment results. Section 5 summarizes the related work. Finally, we conclude with Section 6.

2. COARSE-GRAINED BEHAVIOR MODELING

Our study of malware found that ***File, Registry, Process and Network*** are the most security-critical system resources that are widely attacked by malware. Hence, the proposed coarse-grained behavior modeling approach focuses on these four security-critical system resources. The modeling approach can be extended to cover more system resources if needed, Furthermore, our hypothesis is that for the identification of malware family, one need to analyze the types of security-sensitive actions it performs on these system resources. It is insignificant to analyze the number of times or the sequence these actions are performed as malware authors often use some basic obfuscation techniques to evade malware detection techniques [5]. Next, we shall describe our behavior modeling approach.

SIGSOFT'12/FSE-20, November 11 - 16 2012, Cary, North Carolina, USA

For file resources, we have identified five possible actions – {*create*, *modify*, *read*, *delete, memory_mapped*} – that could be performed by a malware. We only considered those actions that are closely related to system security. The tasks performed by these actions are self explanatory; *create* action creates a new file/directory or opens an existing file, *modify* action modifies the file in concern, *read* action reads the content of the file, *memory_mapped* action maps the file into a virtual memory that can be shared by several processes and finally *delete* action deletes the file/directory. Hence, a malware has a maximum of (2^5 – 1) = 31 possible sets of actions that could be performed on file resource. Each such possible set is modeled as a **File feature** of a malware. Therefore, altogether there are **31** possible file features of a malware.

For example, if a malware performs *create*, *modify*, and *delete* actions on a file *A*, then the malware has a file feature f_A = {*create*, *modify*, *delete*}. If the same malware performs *create*, *modify*, *read* and *deletes* actions on another file *B*, then it has another file feature f_B = {*create, modify, read, delete*}. However, if the same malware performs *create*, *modify*, and *delete* actions on a file *C*, then these actions on *C*, does not introduce any new feature to this malware as this feature is considered the same as file feature f_A. Therefore, this malware has only two file features $F = \{f_A, f_B\}$. As a result of abstraction, an identical set of actions performed on two different file objects (e.g. file *A* and *C*) are considered as a single malware feature. Our initial experiment that will be discussed later shows that such behavior modeling reduces the amount of noise in the malware features space while accurately capturing the underlying malicious characteristics.

Likewise, we have identified six possible actions – {create_*key, delete_key, monitor_key, modify_value, read_value, delete_value*} – that could be performed by a malware on registry resource. Registry actions are responsible for critical system behaviors such as startup procedures, internet settings and user-specific settings. *Create_key* action creates a new registry key or opens an existing one, *delete_key* action deletes the key from the registry, *monitor_key* action monitors for changes to any attributes or contents of a specified registry key, *modify_value* action creates or replaces a registry key's value, *read_value* action reads the value entry for a key and *delete_value* action deletes the value entry of a specified registry. Similar to file resource, a malware has a maximum of (2^6 -1) = 63 possible sets of actions that could be performed on a registry resource. Each such possible set is modeled as a **Registry feature** of a malware. Therefore, altogether there are **63** possible registry features of a malware.

For the process resource we have listed five possible actions – {*create_process, delete_process, create_thread, foreign_memory _read, foreign_memory_write*} – that could be performed by a malware. A process is an executing program and one or more threads can run in the context of a process. *Create_process* action creates a process to execute a program, *delete_process* action terminates the process and kill all its threads, *create_thread* action creates a new thread to run within an executing process, *foreign_memory_read* action allows the process to read the shared memory and *foreign_memory_write* action allows the process to write to the shared memory. Hence, a malware has a maximum of (2^5 -1) = 31 possible sets of actions that could be performed on a process resource. Each such possible set is modeled as a **Process feature** of a malware. Therefore, altogether there are **31** possible process features of a malware.

Finally, we have identified four possible actions – {address_scan, ping, connection_attempt, conversation} – that could be performed by a malware during networ k I/O. *Address_scan* action scans for a given network, *ping* action checks whether a remote target available over the network, *connection_attempt* action attempts to connect to a particular destination over the network and *conversation* action communicates with the destination. Similar to other system recourses, a malware has a maximum of (2^4 -1) = 15 possible sets of actions that could be performed during network I/O. Each such possible set is modeled as a **Network feature** of a malware. Therefore, altogether there are **15** possible network features of a malware.

From the description of our behavior modeling, it is understood that malware feature space is no longer a function of total number of malware samples under examination. Furthermore, upper bound for the malware feature space is predetermined before running the experiment and importantly, it doesn't depend on the number of malware samples under analysis. Thus, our approach enhances the scalability of malware clustering.

2.1 Feature Encoding

Once the features are extracted from the malware samples, we use bit vector to represent them. In total, there can be a maximum of 31 + 63 + 31 + 15 = 140 features on all four system resources. Hence, we use a bit vector of size 140 to represent the malware features. The first 31 bits are used to represent the file features next 63 bits are used to represent the registry features another 31 bits are used to represent the process features and the last 15 bits are used to represent the network features.

Formally, let *F*, *R*, *P*, and *N* be the set of file features, registry features, process features and network features respectively. Let $Z^{1-31} = \{x_1 | \ 1 \le x_1 \le 31,\ x_1 \in \mathbb{Z}^+\}$, $Z^{32-94} = \{x_2 | \ 32 \le x_2 \le 94,\ x_2 \in \mathbb{Z}^+\}$, $Z^{95-125} = \{x_3 | \ 95 \le x_3 \le 125,\ x_3 \in \mathbb{Z}^+\}$ and $Z^{126-140} = \{x_4 | \ 126 \le x_4 \le 140,\ x_4 \in \mathbb{Z}^+\}$. Let *f*, *r*, *p* and *n* be the mapping $F \to Z^{1-31}$, $R \to Z^{32-94}$, $P \to Z^{95-125}$, $N \to Z^{125-140}$ to assign a file, registry, process and network features respectively to a positive integer in the respective range that serves as the bit position in the feature vector to represent the feature. For a malware *m*, the feature vector *V* is defined as follows;

$$V[k] = \begin{cases} 1, \text{if } 1 \le k \le 31 \text{ and } m \text{ has a feature } x \in F, \text{such that } f[x] = k \\ 1, \text{if } 32 \le k \le 94 \text{ and } m \text{ has a feature } x \in R, \text{such that } r[x] = k \\ 1, \text{if } 95 \le k \le 125 \text{ and } m \text{ has a feature } x \in P, \text{such that } p[x] = k \\ 1, \text{if } 126 \le k \le 140 \text{ and } m \text{ has a feature } x \in N, \text{such that } n[x] = k \\ 0, \text{ otherwise} \end{cases}$$

The similarity between two malware feature vectors is calculated using Euclidian distance *d*. For example, the similarity between feature vectors v_i and v_j is calculated as;

$$d(v_i, v_j) = \sqrt{\sum (\widetilde{v_i} - \widetilde{v_j})^2}$$

Where, $\widetilde{v_i}, \widetilde{v_j}$ are normalized feature vectors of vectors v_i and v_j respectively. Due to normalization, $d(v_i, v_j) = 0$ for identical behavior and $d(v_i, v_j) = \sqrt{2}$ for completely different behavior.

2.2 Feature extraction from behavioral reports

Malware samples are executed in Anubis [1], a dynamic malware analysis environment, and the generated malware behavioral reports are used for our analysis purposes. These behavioral reports give a high-level understanding of malware behavior. From the behavioral reports, malware features are extracted in two steps as explained below.

Step 1 - extraction: System resources and the actions performed on those resources are extracted as *<resource_object_id: set_of_actions>* pairs, where *resource_object_id* represents the name of system resource object and the *set_of_actions* represent the set of actions performed by malware on the resource object.

Step 2 - abstraction: We abstract away the system specific details, that is; replace individual system resource object name (e.g. *a.txt*, *b.txt*, etc…) with the resource type; *File*, *Registry*, *Process* and *Network*. Finally, remove the duplicate features from the abstracted feature set.

```
<file_activities>
<file_created     name = "C:\a.txt"/>
<file_read      name = "C:\a.txt"/>
<file_read      name = "C:\b.txt"/>
<file_read      name = "C:\c.txt"/>
<file_modified     name = "C:\a.txt"/>
<file_deleted      name = "C:\b.txt"/>
<file_deleted      name = "C:\c.txt"/>
</file_activities>
```

Listing 1: Sample malware behavioral report

```
<C:\a.txt: {file_created, file_modified, file_read}>
<C:\b.txt: {file_read, file_deleted}>
<C:\c.txt: {file_read, file_deleted}>
```

Listing 2: Result of step 1

$$\text{File feature} = \left\{ \begin{array}{c} \{\text{file\_created, file\_modified, file\_read}\}, \\ \{\text{file\_read, file\_deleted}\} \end{array} \right\}$$

Listing 3: Result of step 2

A sample malware behavioral report, with only file activities, is shown in Listing 1, while output of step 1 and 2 are shown in Listings 2 and 3 respectively. Listing 2 shows the File features extracted in step 1 from malware behavioral report. It can be seen that the actions performed, by the malware, on file resource objects *b.txt* and *c.txt* are identical. Thus, in step 2 we remove these duplicate features. Listing 3 shows the final set of File features that are used to generate malware feature vector as explained in section 2.1.

2.3 Feature pruning

Once the malware features are extracted, we prune the features with low information gain. A feature that appears in only one sample (i.e. unique feature) doesn't help to find other samples that behave similarly [5] and common features that appear in most (or all) of the malware samples may lead to group two samples together even if they both actually belong to two different malware classes. Following the approach in [5], we prune those features that appear in only one malware and features that appear in 90% (or more) of the malware samples.

3. SCALABLE CLUSTERING

Apart from scalable behavior modeling, clustering algorithm itself has to be scalable in order to cluster large volume of malware samples. For large-scale clustering there are two major techniques proposed in the literature; (1) Locality Sensitive Hashing (LSH) based clustering used by Bayer et al. [5] and, (2) prototype based clustering technique introduced by Rieck et al. [6].

Prototype based clustering technique is simple, yet effective for malware clustering. From a set of malware samples, prototypes are selected based on a threshold value to represent the entire malware samples. These selected prototypes are then clustered using hierarchical clustering and finally, un-clustered malware samples are assigned to the closest prototype (refer to the original paper [6] for actual implementation). Since our focus, in this paper, is not on scalable clustering algorithms, we adopt the prototype based clustering technique to evaluate the performance and applicability of our behavior modeling approach to real-world malware clustering problems. It is also noted that we use F-Measure, introduced by Van Rijsbergen [10], to evaluate the clustering accuracy.

4. PRELIMINARY EXPERIMENTAL RESULTS

In this section, we discuss our preliminary experimental results to evaluate the accuracy and scalability of our coarse-grained malware behavior modeling approach. The experiment was carried out on a dataset provided by Bayer et al. [5]. Initially we had 2658 malware behavioral reports out of which 6 reports were discarded due to problems in file parsing. Hence, we carried out the experiment using 2652 malware samples. Based on these 2652 samples, we compare our clustering accuracy against a state-of-the-art malware clustering technique proposed by Bayer et al. [5] and the results obtained by Malheur [6]. Malheur is a tool developed by Rieck et al. [6] to automatically analyze the malware behavior. It is also noted that we use Malheur in "text" mode, where it supports both "text" for textual and XML reports, and "MIST" for reports using malware instruction set (refer to [6] for more details).

Bailey et al. [7] pointed out that malware labels given by anti-virus products suffer from inconsistency and their labels are frequently incorrect. Thus, we have decided to evaluate the clustering accuracy using malware labels given by five different anti-virus products. This enables us to evaluate the robustness of our malware clustering technique across five different benchmarks and therefore reduces the biasness. The anti-virus products include F-Secure, Ikarus, Symantec, Kaspersky and VirusBuster and the corresponding malware labels are obtained from Virustotal tool [3]. In this experiment, we only use malware family name and the variant name is ignored. For example, we group both *Worm.Allaple.A* and *Worm.Allaple.B* together and call it *Allapale* malware family.

The experiment results are shown in Table 1. In terms of clustering accuracy, from the experiment results it can be seen that our approach clearly outperformed Bayer et al. and Malheur. Using Bayer's results as base we measured the average improvement to clustering accuracy. Our approach improved the clustering accuracy by 6.20% whereas Malheur improved it by only 1.90%.It is also evident that our approach consistently performed well against all five benchmarks.

In terms of scalability, we analyzed the relationship between malware feature space and the number of samples under examination. We compared the number of features generated by our approach against Bayer et al. and Malheur. Table 2 shows the relationship between feature space and number of malware samples. From Table 2, it is evident that number of features generated by Bayer et al. and Malheur grow in proportion with the number of malware samples under examination. Whereas, in our approach the growth rate of malware feature space is insignificant and only 56 features were analyzed. Hence, the experiment result shows that our method reduces the malware feature space by 16,160/56 = 289 and 5,171,871/56 = 92,355 times comparing with Bayer et al. and Malheur methods respectively.

Table 1: Preliminary experiment results

Anti-Virus Vendors	F-Measure			Using Bayer's results as base	
	Bayer et al.	Malheur	Ours	Improvement by Malheur	Improvement by our approach
F-Secure	0.870	0.891	**0.923**	2.41 %	**6.09 %**
Ikarus	0.882	0.894	**0.948**	1.36 %	**7.48 %**
Symantec	0.874	0.889	**0.916**	1.72 %	**4.81 %**
Kaspersky	0.876	0.893	**0.930**	1.94 %	**6.16 %**
VirusBuster	0.870	0.888	**0.926**	2.07 %	**6.44 %**

In this experiment, our approach has an upper bound of 140 features. It is also noted that there is no strict upper bound for malware feature space as it can vary based on the types of sensitive system resources and the corresponding security-critical actions considered. Though we have not verified extensively, our initial experiment shows that our approach has the advantage of predetermining the feature space before clustering process in such a way that it doesn't grow in proportion with number of malwares.

Table 2: Relationship between malware sample size and feature space

Number of malware samples	Feature space		
	Bayer et al.	**Malheur**	**Ours**
500	6,380	18,059	42
1000	9,007	623,300	47
1500	11,349	1,561,985	51
2000	12,797	2,594,645	52
2652	**16,160**	**5,171,871**	**56**

5. RELATED WORK

Apart from scalability of malware feature space, which is common to all, there are several other issues in the existing malware clustering techniques. Lee et al. [8] proposed a behavior-based malware classification tool that compares the sequence of system calls to determine the similarity between two executables. The fine-grained approach makes it difficult to abstract the underlying malicious behavior. Hence, restricts it applicability. Bailey et al. [7] modeled the malware behavior as non-transient state changes. Though, state changes are a higher level abstraction than single system calls, this approach fails to identify the relationship between two states. Bayer et al. [5] models the runtime behavior of a malware using OS objects and operations in a fine-grained manner. The system and malware-specific details such as file names, registry values, IP addresses and other parameters of system operations, introduce unnecessary features and increase the amount of noise in the malware features space. Rieck et al. [6] proposed Malheur that models the system call sequences using n-grams. The malware feature space generated by Malheur is very huge and n-gram analysis is not obfuscation-resilient.

6. CONCLUSION AND FUTURE WORK

In this paper, we have introduced a novel coarse-grained malware behavior modeling technique that addresses the scalability problem in malware clustering. Our approach finds an upper bound for malware feature space and reduces the amount of noise in the extracted features. Furthermore, preliminary experiment result shows that our approach improves the clustering accuracy by 6.20% while reducing the feature space by 289 times against a state-of-the-art malware clustering technique.

In the future, we have planned to extend this experiment to larger dataset and evaluate the scalability of our behavior modeling technique in terms of time and space requirements. As malware may evolve their behavior from time to time, there could be a need to extend the proposed behavioral framework to cover more system resources and actions to address the evolution of malware behavior. We would like to conduct extensive experiments to statistically verify the growth of malware feature space in terms of number of malware samples examined. Last but not least, we would like to seek feedbacks from peers on the practicality of the proposed approach for malware clustering and the possible ways to improve it further.

7. ACKNOWLEDGMENTS

We would like to thank both Paolo Milani for providing us the dataset used in Bayer et al. [5] and Konrad Rieck for providing us the Malheur tool [6].

8. REFERENCES

[1] ANUBIS: http://anubis.seclab.tuwien.ac.at

[2] Threat Reportt-2010: http://www.symantec.com/threatreport/

[3] Virustotal tool: https://www.virustotal.com/

[4] A Look at One Day of Malware Samples: http://blogs.mcafee.com/mcafee-labs/a-look-at-one-day-of-malware-samples

[5] Bayer, U., Comparetti, P.M., Hlauschek, C., Kruegel, C and Kirda, E. Scalable, Behavior-based Malware Clustering. In Proceedings of the 16th NDSS, 2009.

[6] Rieck, K., Trinius, P., Willems, C and Holz, T. Automatic analysis of malware behavior using machine learning. TR, Berlin Institute of Technology. 2009.

[7] Bailey, M., Andersen, J., Mao,Z.M and Jahanian, F. Automated Classification and Analysis of Internet Malware. In Proceedings of RAID. 2007.

[8] Lee, T and Mody, J.J. Behavioral Classifcation. In Proceedings of EICAR, Hamburg, Germany. April 2006.

[9] You, I., Yim, K. Malware Obfuscation Techniques: A Brief Survey. Int. Conf. on Broadband, Wireless Computing, Communication and Applications pp. 297–300. 2010.

[10] Rijsbergen, V. C. J. Information Retrieval, 2nd edition. Dept. of Computer Science, University of Glasgow. 1979.