On Exploiting Static and Dynamic Features in Malware Classification

Jiwon Hong, Sanghyun Park, and Sang-Wook Kim(✉)

Hanyang University, Seoul, Republic of Korea
{nowiz,singhyun,wook}@hanyang.ac.kr

Abstract. The number of malwares is exponentially growing these days. Malwares have similar signatures if they are developed by the same group of attackers or with similar purposes. This characteristic helps identify malwares from ordinary programs. In this paper, we address a new type of classification that identifies the group of attackers who are likely to develop a given malware. We identify various features obtained through static and dynamic analyses on malwares and exploit them in classification. We evaluate our approach through a series of experiments with a real-world dataset labeled by a group of domain experts. The results show our approach is effective and provides reasonable accuracy in malware classification.

Keywords: Malware classification · Static analysis · Dynamic analysis · Feature extraction

1 Introduction

Malwares are continuously causing social and economic damages. Despite the combined effort of various companies in different countries such as Microsoft, Kaspersky, Ahnlab, and Avast, the number of malwares is growing more than ever in recent years.

There are two primary reasons for the fast growth of malwares: there are a large number of attackers who are developing new malwares continuously; due to a variety of tools, it is relatively easy to develop a malware. For years, a number of attackers are releasing new malwares while successfully evading laws. They often share their source codes, which could also be accidentally leaked out to the public. This enables other novice attackers to build new malwares without difficulty. A number of malwares with a similar method of attacking could come out in a short period of time. Moreover, new malwares may take advantage of polymorphism or metamorphism for evading detection [1–3].

Despite such efforts of attackers, it is highly likely that similar signatures can be found in two different malwares written by the same attacker when analyzed from diverse angles. It is because those malwares essentially share the same code or attack patterns. Using the shared signatures is a widely used technique for distinguishing malwares from ordinary programs [1,3].

J.J. Jung and P. Kim (Eds.): BDTA 2016, LNICST 194, pp. 122–129, 2017.
DOI: 10.1007/978-3-319-58967-1_14

In general, the goal of malware detection is simply to identify malwares from benign programs. However, if we are able to define classes of malwares and to automatically classify a given sample to its class, it would be more helpful than mere detection. Class information could be utilized for deeper researches on malwares such as finding countermeasures, developing treatments, and digital forensics [2].

In this paper, we present a classifier which successfully determines the class of a given sample with a set of classes defined by domain experts. We exploit various static and dynamic features at the same time to achieve high classification accuracy. Also, we show the effectiveness of our classifier via experiments.

In Sect. 2, we describe the class definition and the dataset used in our research. Section 3 explains the various features of malwares extracted for classification. Section 4 describes the classification method. We show the effectiveness of our classifier via experiments in Sect. 5. We conclude our paper in Sect. 6.

2 Class Definitions and Dataset

The definition of classes could differ for the objectives of applications. In this research, we are focusing on finding the groups of attackers who release malwares. We used a class set defined by a group of experts based on the similar attackers. A classifier learned from such a class set should be useful for inferring which attacker group released a given sample.

We collected 1,086 malware samples in 2015. Each sample is labeled with one of 7 classes by domain experts. Each class has 155 instances on average. While the largest one has 434 instances, the smallest one has 24. Table 1 shows the number of instances in each class.

Table 1. Number of instances in each class

A	B	C	D	E	F	G
434	24	261	113	48	147	61

3 Feature Extraction

For classifying malwares, features that describe each malware should be extracted. Some of these features may be signatures important to identify each malware. Classification would be successful only if the most informative and distinguishing features are identified and extracted.

The core files of a malware usually are in the executable binary form. Both static and dynamic analyses could be used for analyzing such binary files. Static analysis is conducted directly on the binary file [1]. For example, extracting public methods or printable strings contained in the binary file and disassembling the binary file are part of the static analysis. Dynamic analysis is conducted by

running the binary code inside a controlled sandbox such as virtual machines [1]. While running, we can investigate the detailed behavior of the malware, including network and filesystem usages.

As mentioned earlier, a large portion of malwares are polymorphic and/or metamorphic. In such cases, the classification would be accurate only if the features from the sufficiently various analyses are used. In this paper, we identified and extracted the following features for our classification.

3.1 Static Features

Static features are extracted by analyzing the binary file of a malware. However, static analysis can be seriously affected by the polymorphic characteristics of the malware. Static analysis should be conducted with the dynamic one at the same time.

Functions. We identify the groups of opcodes from a function chunk obtained by disassembling the given binary file as features. If there are reused code chunks in a new sample, we can find them by using these features. The function chunks have variable length and they also consume large storage capacity to be indexed. Therefore, we convert each chunk to a short fixed-length value using a one-way cryptographic hash function [6,7].

Strings. We identify each printable strings contained in the binary code of the malware as a feature. Most of such strings could be meaningless; however, some of them can be used as useful signatures when attackers inserted the same string habitually as a symbol in their own malwares [4].

Imports and Exports. Malwares could be composed of multiple files. In the cases, the malware has the interfaces for calling shared libraries or providing its public methods. We identify such information including the names of the methods as imports and exports features. These features are less affected by the polymorphic or metamorphic characteristics of the malware, thereby being unique.

3.2 Dynamic Features

Dynamic analysis is more suitable for extracting features from polymorphic malwares than static analysis. In this research, we executed the given malware sample in a virtual machine and identified the following features based on the running behavior of the sample in its execution.

System API Calls. A large portion inside the binary code of the malware consists of the system API calls provided by the operating system. We identify such system API calls made by each malware as features [5–7].

Mutexes. Mutexes are used for locking of a memory location or preventing the execution of multiple instances of the malware. The names of mutexes might be the same normally if they are created by the same group of attackers. It is more

likely if the codes are reused. We identify the names of mutexes used in each malware sample as features.

Networks. We analyze the network requests from the sample and identify their information as features. Typical examples of such information are DNS URLs and IP addresses. Malwares are likely to use the same DNS URLS and IP addresses if they are developed by the same attacker or developed for the same purpose.

Files. Most of malwares show the behavior of reading or writing some files on the filesystem. The behavior might be an act of attacking or concealing itself among normal files. In many cases, the address on the target filesystem indicates a location where a system file resides. The malwares developed by the same group of attackers may share these addresses because they are determined by the attackers heuristically. We identify such addresses as features as well.

Drops. Some malwares could extract a hidden file inside the executable binary code to store it in a filesystem or a memory location. They are also capable of downloading other files from the Internet. Such behaviors are called *drop*. We identify the storage locations, remote addresses, and hash values of such files as our features.

Keys. Microsoft Windows has a data storage called *registry* for storing settings for most of the Windows functions and other applications. A huge number of malwares designed for Microsoft Windows exploit it for attacking the system. We identify all the registry keys that a sample accesses as features.

4 Classification

In this paper, we employ Decision Tree and SVM [8], both well-known classification methods to build our classifier. Decision Tree builds a tree-shaped prediction model from the dataset by evaluating features that each item has. This prediction model is used to determine the class of a new sample.

Figure 1 shows an example of a decision tree prediction model, where each node represents a *decision*. Given a new sample, Decision Tree traverses to a child node which satisfies the condition for the sample, starting from the root node. The traverse is repeated until it reaches a leaf node. If Decision Tree reaches a leaf node, it classifies the given sample as the class label of the leaf node.

To learn such tree-shaped prediction model, Decision Tree first creates a new root node with a condition that splits the whole samples at best with a given splitting criterion. Second, it creates child nodes for the split set of samples and gives each of the nodes a condition that splits the samples best recursively. Finally, if each node cannot be split, then it makes the node a leaf node and labels the node with a class label that the most samples in the node have.

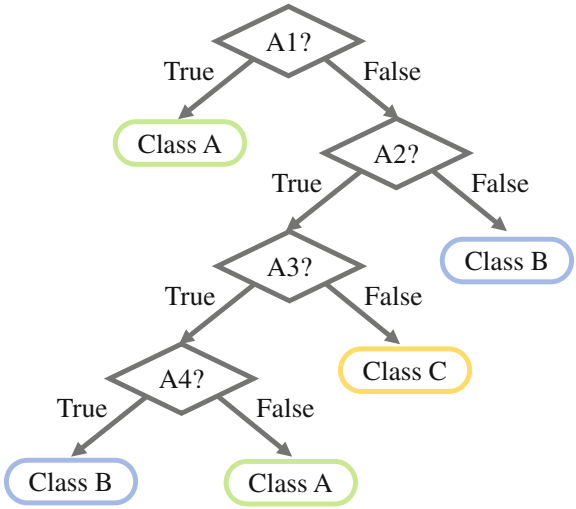

Fig. 1. An example of a decision tree.

We use *normalized information gain* [9] as the splitting criterion which is defined as follows:

$$Gain(D, A) = H(D) - \sum_{v \in V_A} \left(\frac{|D_v|}{|D|} \cdot H(D_v) \right)$$

$$SplitInfo(D, A) = - \sum_{v \in V_A} \frac{|D_v|}{|D|} \cdot log_2 \left(\frac{|D_v|}{|D|} \right)$$

$$GainRatio(D, A) = \frac{Gain(D, A)}{SplitInfo(D, A)}$$

Here, A is a feature; D is the set of all training samples for the node; V_A is the set of all possible values of A; D_v is the set of samples with the value v; and $H(D)$ represents the entropy of the set D. For each node in Decision Tree, the feature with the highest $GainRatio$ is selected as the splitting feature.

Support Vector Machine (SVM) [8] is another well-known classification model. SVM generally shows high classification accuracy and has ability to construct complex models. Given a dataset, it tries to identify an optimal hyperplane or an optimal set of hyperplanes which discriminate two classes of data clearly. An optimal hyperplane is the one that has the largest margin to both classes of data and thus helps lower the errors of the classifier. An optimal hyperplane should satisfy the following condition:

$$W \cdot X + b = 0$$

W is a normal vector to the hyperplane with n elements where n is the number of attributes, X is a set of training tuples, and b represents the bias. To find an optimal hyperplane, we regard the equation above as a constrained convex quadratic optimization problem to find optimal W.

5 Evaluation

5.1 Experimental Setup

In our evaluation, we verify the accuracy of our classifier by training a Decision Tree classifier with our dataset. We used the WEKA library [10] to accomplish our task. We conducted a five-fold cross-validation for the evaluation. We used the following measures to evaluate the accuracy of our model:

$$Precision(C) = \frac{|L_C \cap R_C|}{R_C}$$
$$Recall(C) = \frac{|L_C \cap R_C|}{L_C}$$
$$F1(C) = 2 \cdot \frac{Precision(C) \cdot Recall(C)}{Precision(C) + Recall(C)}$$

Here, C is a class label; L_C is the set of test samples that have the class label C; and R_C is the set of test samples that are classified as C by our classifier. We used an weighted average over all the classes as overall accuracy, where the weight in each class indicates the number of samples in the class.

5.2 Results and Analyses

Table 2 is a confusion matrix M that shows the number of samples in each class has been classified to each class by the Decision Tree classifier. The rows show the actual classes of the samples and the columns show the predicted ones. For example, $M(A, A) = 388$ indicates our classifier correctly classifies 388 samples in Class A as Class A; $M(A, B) = 16$ indicates it misclassifies 16 samples in Class A as Class B. We can see that most of errors occurred in Class A that has the largest number of samples. Also, we observe that Class D shows poor accuracy because it does not have a sufficient number of samples. Notably, all the samples in Class F are correctly classified. We conjecture that Class F is the most distinguished one. Table 3 is a confusion matrix M for the SVM classifier. Table 3 also shows similar however slightly more accurate results.

Overall, the Decision Tree classifier shows the precision of 83.60%, the recall of 84.20% and the F1-measure of 83.40%; the SVM classifier shows the precision of 88.30%, the recall of 88.40% and the F1-measure of 88.20%. The results verify that our classifier performs effectively.

Table 2. A confusion matrix M (DT)

	A	B	C	D	E	F	G
A	388	7	2	5	22	0	9
B	16	41	0	0	4	0	0
C	4	0	14	1	5	0	0
D	21	1	0	11	13	0	2
E	27	3	0	1	225	0	4
F	0	0	0	0	0	113	0
G	19	2	0	1	3	0	122

Table 3. A confusion matrix M (SVM)

	A	B	C	D	E	F	G
A	406	4	0	6	8	0	9
B	12	45	0	0	2	0	2
C	0	0	16	3	4	0	1
D	11	1	0	24	9	0	3
E	23	0	2	2	229	0	4
F	1	0	0	0	0	111	1
G	15	1	0	2	0	0	129

6 Conclusions

In this research, we have addressed a malware classifier that exploits various features extracted from static and dynamic analyses. Our goal is to classify an author group of the given sample. We also have verified that our classifier provides reasonable accuracy via experiments with a real-life dataset.

As a further study, we plan to develop more accurate classifiers for malwares. Using more features such as a control flow [3] and call frequency or identifying a new set of features could be a good starting point towards this direction [6,7]. Another direction is to improve the training speed of our classifier by selectively use only the features that help classification. Finally, new class definitions for other applications can be considered as well.

Acknowledgement. This work was supported by (1), (2) the National Research Foundation of Korea (NRF) Grant funded by the Korean Government (MSIP) (2014R1A2A1A10054151 and 2015R1A5A7037751) and (3) the MSIP (Ministry of Science, ICT and Future Planning), Korea, under the ITRC (Information Technology Research Center) support program (IITP-2016-H8501-16-1013) supervised by the IITP (Institute for Information & communication Technology Promotion).

References

1. Islam, R., Tian, R., Batten, L.M., Versteeg, S.: Classification of malware based on integrated static and dynamic features. J. Netw. Comput. Appl. **36**(2), 646–656 (2013)
2. Tian, R., Batten, L.M., Islam, R., Versteeg, S.: An automated classification system based on the strings of trojan and virus families. In: Proceedings of the 4th International Conference on Malicious and Unwanted Software (MALWARE), pp. 23–30 (2009)
3. Cesare, S., Xiang, Y.: Classification of malware using structured control flow. In: Proceedings of the 8th Australasian Symposium on Parallel and Distributed Computing (AusPDC), pp. 61–70 (2010)
4. Islam, R., Tian, R., Batten, L.M., Versteeg, S.: Classification of malware based on string and function feature selection. In: Cybercrime and Trustworthy Computing Workshop (CTC), 2010 Second, pp. 9–17 (2010)
5. Park, Y., Reeves, D., Mulukutla, V., Sundaravel, B.: Fast malware classification by automated behavioral graph matching. In: Proceedings of the 6th Annual Workshop on Cyber Security and Information Intelligence Research (CSIIRW), pp. 45–49 (2010)
6. Chae, D., Ha, J., Kim, S.-W., Kang, B., Im, E., Park, S.: Credible, resilient, and scalable detection of software plagiarism using authority histograms. Knowl. Based Syst. **95**(1), 114–124 (2016)
7. Chae, D., Kim, S.-W., Cho, S.-J., Kim, Y.: Effective and efficient detection of software theft via dynamic API authority vectors. J. Syst. Softw. **110**, 1–9 (2015)
8. Han, J., Kamber, M.: Data Mining: Concepts and Techniques. Morgan Kaufmann, San Francisco (2006)
9. Quinlan, J.R.: C4.5: Programs for Machine Learning. Morgan Kaufmann Publishers, San Francisco (1993)
10. Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.: The WEKA data mining software: an update. ACM SIGKDD Explor. Newsl. **11**(1), 10–18 (2009)