Clustering Microsoft Windows Executables based on TF-IDF and API Information

Jonghwa Park[1], Gyoosik Kim[1], Youngsup Hwang[2]*, and Seong-je Cho[1]
[1]Dankook University, Yongin, Gyounggi, Korea
{72150262, erewe4, sjcho}@dankook.ac.kr
[2]Sun Moon University, Asan, Chungnam, Korea
young@sunmoon.ac.kr

Abstract

The illegal software usage is 39% worldwide and malware is frequent in the illegal software. To protect attacks from malware, we use software filtering. The software filtering compares equivalence of a testing software to an original one. This requires comparison between all the legal programs in the market. So we have to reduce the number of comparisons by clustering programs in the market. Every market provides categories to programs such as image viewer, video player, audio player, and messenger, etc. But it is not clear that these categories are best fit to filter malware. We suggest new categories which are more suitable to classification experimentally. Our categories are automatically made from the K-means clustering algorithm based on TF-IDF and API information. Experimental results show that our clustering scheme is better than the existing categories to classify malware.

Keywords: Clustering, Windows Executable, TF-IDF, API, K-means, Random Forest

1 Introduction

According to BSA(the software alliance), 39% of software installed on computers around the world in 2015 is not properly licensed [1]. A strong connections exists between cyberattacks and the use of illegitimate or unlicensed software. To protect attacks from malwares, we use software filtering. Existing software filtering systems determine a suspicious program as illegal software if the program is identical or similar to one of programs maintained in the filtering database. It is easy to identify and filter an illegal program if the original program is distributed without any change. However, the existing filtering systems have limitations. If target programs are distributed as hacked (cracked, modified, etc.) or counterfeit versions, the existing filtering systems suffer from performance degradation on determining whether a suspicious program is the hacked or counterfeit version of its original one. Therefore, we need a method to efficiently measure similarity between a suspicious program and one of the programs in the database for determining whether the suspicious program is one of hacked or counterfeit versions from its original. In this case, performance overhead highly increases if the suspicious program is compared with all programs in the database for measuring software similarity. As a result, we have to reduce the number of comparisons between the suspicious program and the original programs in the database. If we can divide programs into several categories, we can reduce comparison times. Every software download site has software categories to group programs. But it is not clear that this grouping is suitable to automatic filtering. This paper presents a clustering scheme based on machine learning and compares this with the original website categories experimentally. We collected programs from 9 categories of software download sites which are most popular. Experimental results show that 7 or 9 clusters is appropriate in

Research Briefs on Information & Communication Technology Evolution (ReBICTE), Vol. 2, Article No. 7 (August 31, 2016)
*Corresponding author: Department of of Computer Science and Engineering, Sun Moon University, 221-70 Sunmoonro, Asansi, Chungnam, 31460, Korea, Tel: +82-41-530-2256

this case and new clusters can be classified more accurately than the original categories. If a suspicious program is classified into the exact cluster, our scheme can reduce the time to measure software similarity by 1/7 or 1/9 compared to the one where any software clustering scheme is not applied.

2 Related Works

As a software birthmark reflects developer's characteristics, it can be used a unique feature for each program [2]. API call frequencies, weighted IDF and string information are examples of birthmarks. There are several research works on software classification. In [9], a technique is presented that generates software montage for Android applications using the information on API calls, strings, and URLs. Based on the montage, applications are ranked in terms of similarity score. In the field of malware classification, most research extracts the feature of malware by detecting malicious behavior or monitoring behavior dynamically. In [5], malware family is classified by extracting the frequency of instructions. In [3], malware is classified using machine learning based on the feature of behaviors such as access to file, network and encryption. These techniques can classify malware effectively since they usually detect specific behaviors common to the corresponding malware family. However, these techniques cannot be directly applied to software classification because the used feature is not suitable for the functionality-based classification, and cannot be applied to software filtering because filtering systems require fast identification.

3 Binaries Clustering

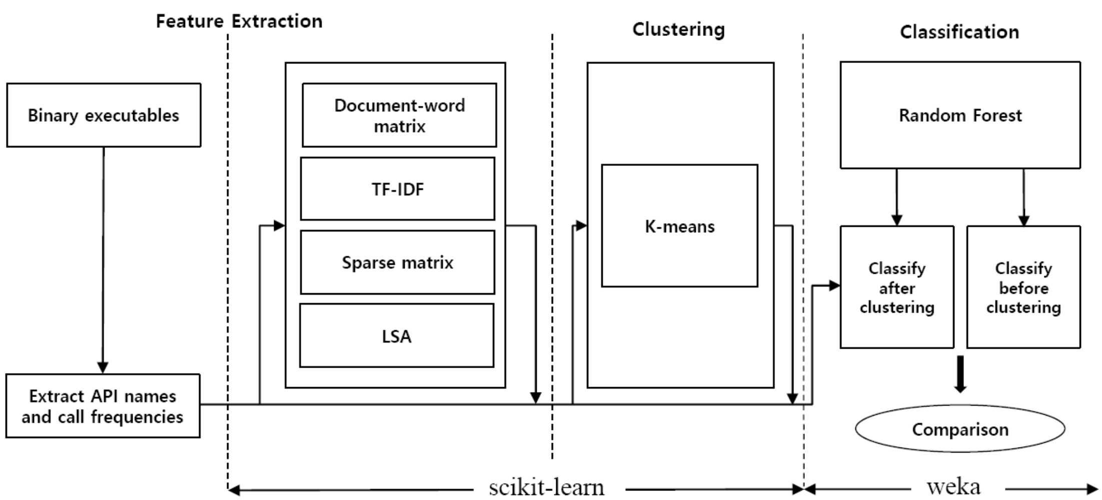

Figure 1: Overall Process of Experiments

3.1 Collecting programs

We collected programs, windows executables, from well-known websites such as AlternativeTo, Sorceforge, FileHippo and FreewareFiles that offer computer software for Windows. We also selected 9 categories- Audio Player, CD Writer, FTP, Image Viewer, Messenger, Text Editor, Video Player and Zip.

Selection criteria are popularity and familiarity. The number of collected programs for each category is 55 and the total number is 495.

Figure 2: Software website

3.2 Feature Extraction

Since a binary executable is large, we have to extract features to compare between an original program and a pirated one to reduce comparison time. The Windows API is Microsoft's core set of application programming interfaces (APIs) available in the Microsoft Windows operating systems. Almost all Windows programs interact with the Windows API. Thus the APIs and their call frequencies could be a birthmark to show the unique characteristics of a program. We can extract the information on APIs from .idata section. We extract IAT (Import Address Table) from .idata section and identify the name of API. From .text section, we extract instructions by disassembling, and examine whether the opernad of CALL or JMP instruction is the address of APIs in the IAT. Then we count the number of calls for each API.

3.3 Refining Data

The total numnber of different APIs extracted from the collected programs is about 10,000, which is too large and they contain redundant and useless information. Hence we refined the data using TF-IDF (Term Frequency-Inverse Document Frequency). TF-IDF reflect how important a word(i.e. an API in this case) is to a document(i.e. a program) in a collection. TF-IDF value increases proportionally to the number of times a word appears in the document. We use scikit-learn [6], a machine learning tool in Python to get TF-IDF. Scikit-learn provides the TfidfVectorizer module for TF-IDF. TF-IDF is defined in the TfidVectorizer as

$$tf(t,D) = \log_{t,D} + 1 \tag{1}$$

$$idf(t,D) = \log \frac{N}{|\{d \in D : t \in d\}|} \tag{2}$$

where t is a term, D is a document and N is the number of documents. Application of TF-IDF produces a matrix which contains many '0's. We reduced space as shown in the figure 3 where a row is for a program and a column is for an API. Then we applied LSA (Latent Semantic Analysis) to make the data

	[0]	[1]	[2]	[3]	[4]	[5]	[6]
[0]	0	0	**2**	0	0	0	**12**
[1]	0	0	0	0	**7**	0	0
[2]	**23**	0	0	0	0	0	0
[3]	0	0	0	**31**	0	0	0
[4]	0	**14**	0	0	0	**25**	0
[5]	0	0	0	0	0	0	**6**
[6]	**52**	0	0	0	0	0	0
[7]	0	0	0	0	**11**	0	0

$\Rightarrow$

<row, col, value>
<0, 2, 2>
<0, 6, 12>
<1, 4, 7>
<2, 0, 23>
<3, 3, 31>
<4, 1, 14>
<4, 5, 25>
<5, 6, 6>
<6, 0, 52>
<7, 4, 11>

Figure 3: Sparse Matrix

more meaningful and to reduce its size [8].

4 Clustering

We used K-means clustering algorithm to cluster the collected programs. Kmeans module of scikit-learn offers options to set the number of clusters and maximum iterations. We used silhouette coefficient to measure the clustering result [7].

$$s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}}, \quad -1 \leq s(i) \leq 1 \tag{3}$$

where $a(i)$ is the average dissimilarity of i with all other data within the same cluster, $b(i)$ is the lowest average dissimilarity of i to any other cluster, of which i is not a member. $s(i)$ means that as it goes to 1, i is clustered accurately. Table 1 shows silhouette coefficients and options of scikit-learn which produced highest value as we increases the number of clusters from 5 to 12.

Table 1: Silhouette coefficient

# of cluster	max_df	sublinear_tf	smooth_idf	silhouette coefficient
5	1.0	false	true	0.660
6	1.0	false	true	0.692
7	0.6	false	true	**0.708**
8	0.8	false	true	0.697
9	0.7	false	true	**0.704**
10	0.7	false	true	0.682
11	0.7	false	true	0.673
12	0.7	false	true	0.674

When the number of clusters is 7 and 9, the silhouette coefficient is high and these mean that the appropriate number of clusters is 7 and 9. So we analyzed the results for 7 and 9 clusters. Table 2,3 shows the number of members for each cluster when the number of clusters is 7 and 9.

Table 2: Experimental results for 7 clusters

	Audio	Browser	CD Writer	FTP	Image Viewer	Media Player	Messenger	Text Editor	Zip	Total
cluster 0	13	3	6	9	9	13	10	3	6	72
cluster 1	17	0	21	7	22	11	8	16	9	111
cluster 2	6	10	0	0	3	9	4	8	0	40
cluster 3	9	10	9	17	9	10	16	15	17	112
cluster 4	4	3	17	16	9	4	4	9	20	86
cluster 5	5	19	2	6	3	2	9	3	3	52
cluster 6	1	10	0	0	0	6	4	1	0	22
Total	55	55	55	55	55	55	55	55	55	495

Table 3: Experimental results for 9 clusters

	Audio	Browser	CD Writer	FTP	Image Viewer	Media Player	Messenger	Text Editor	Zip	Total
cluster 0	4	3	17	16	9	4	3	9	20	85
cluster 1	3	3	0	0	1	5	2	5	0	19
cluster 2	7	7	9	11	8	9	11	11	16	89
cluster 3	13	3	6	9	9	13	10	3	6	72
cluster 4	8	13	2	6	1	1	8	2	1	42
cluster 5	16	0	20	7	20	11	9	15	9	107
cluster 6	1	10	0	0	0	6	3	1	0	21
cluster 7	1	10	0	0	2	5	5	4	1	28
cluster 8	2	6	1	6	5	1	4	5	2	32
Total	55	55	55	55	55	55	55	55	55	495

5 Classification

To validate our clustering results, we classified programs into the created clusters and compared it with classification with the original categories. We used random forest method in WEKA as the training and classification tool [4]. We used the frequencies of API calls as features for random forest. The options of random forest in WEKA are as follows; numFeatures = 500, numTrees=199, 10-fold cross validation and the others are default. Table 4 shows the classification results for the original categories. And Table 5 and 6 show the classification results for the 7 and 9 clusters that we produced.

The F-measure for table 4 is 0.724, table 5 is 0.925 and table 6 is 0.901. Experimental results show that the clusters we produced can classify better than the categories that are manually selected.

6 Conclusion

We proposed a software clustering scheme using machine learning. Software clustering can reduce the number of comparisons by reducing the target programs and guess the features or similar programs of unknown software. Software classification must use unique features and those features can be extracted

Table 4: Classification results for the original categories

	A	B	C	D	E	F	G	H	I	Total	TP rate
A	29	1	3	2	3	5	4	5	3	55	0.527
B	0	54	0	0	0	1	0	0	0	55	0.982
C	1	0	46	0	3	1	0	3	1	55	0.836
D	1	0	1	47	0	3	0	0	3	55	0.855
E	1	1	7	1	32	2	4	2	5	55	0.582
F	2	0	1	9	1	37	2	3	0	55	0.673
G	5	5	4	4	3	0	31	2	1	55	0.564
H	5	0	4	0	0	2	1	43	0	55	0.782
I	4	0	4	2	0	3	0	0	42	55	0.764
Total	55	55	55	55	55	55	55	55	55	495	

TR rate	FR rate	Precision	Recall	F-measure
0.729	0.034	0.729	0.729	0.724

where A=Audio Player, B=Browser, C=CD Writer, D=FTP, E=Image Viewer, F=Messenger, G=Text Editor, H=Video Player and I=Zip.

Table 5: Classification results for the 7 clusters

	C1	C2	C3	C4	C5	C6	C7	Total	TP rate
C1	68	0	0	3	0	0	1	72	0.944
C2	0	105	1	5	0	0	0	111	0.946
C3	1	0	38	0	0	1	0	40	0.950
C4	1	4	1	105	0	1	0	112	0.938
C5	0	0	0	1	85	0	0	86	0.988
C6	0	1	8	2	0	40	1	52	0.729
C7	2	0	3	0	0	0	17	22	0.773
Total	72	110	51	116	85	42	19	495	

TR rate	FR rate	Precision	Recall	F-measure
0.925	0.014	0.930	0.925	0.925

rapidly and easily. We use TF-IDF to cluster programs and use API call frequency as unique features to classify programs. Experimental results show that the clusters produced by machine learning algorithm can be classified more accurately than the categories manually selected. When classification is correct, we can reduce the target programs to $1/N$ times faster, where N is the number of clusters.

Acknowledgments

This research was supported by Basic Science Research Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Education(no. NRF-2015R1D1A1A02061946)

Table 6: Classification results for the 7 clusters

	C1	C2	C3	C4	C5	C6	C7	C8	C9	Total	TP rate
C1	85	0	0	0	0	0	0	0	0	72	1.000
C2	0	12	0	0	0	0	3	4	0	19	0.632
C3	0	1	80	1	0	3	0	0	4	89	0.899
C4	0	0	3	69	0	0	0	0	0	72	0.958
C5	0	0	2	1	38	0	0	1	0	42	0.905
C6	0	0	2	0	0	105	0	0	0	107	0.981
C7	0	1	0	0	0	0	19	1	0	21	0.905
C8	0	4	0	1	1	1	2	19	0	28	0.679
C9	0	0	6	0	5	1	0	0	20	32	0.625
Total	85	18	93	72	44	110	24	25	24	495	

TR rate	FR rate	Precision	Recall	F-measure
0.903	0.013	0.901	0.903	0.901

References

[1] Unlicensed software use still high globally despite costly cybersecurity threats. `http://globalstudy.bsa.org/2016/index.html`, 2016.

[2] J. Choi, Y. Han, S. je Cho, H. Yoo, J. Woo, M. Park, Y. Song, and L. Chung. A static birthmark for ms windows applications using import address table. In *Proc. of the 7th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS'13), Taichung, China*, pages 129–134. IEEE, July 2013.

[3] I. Firdausi, C. lim, A. Erwin, and A. S. Nugroho. Analysis of machine learning techniques used in behavior-based malware detection. In *Proc. of the 2nd International Conference on Advances in Computing, Control and Telecommunication Technologies (ACT'10), Jakarta, Indonesia*, pages 201–203. IEEE, December 2010.

[4] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The weka data mining software: an update. *ACM Special Interest Group on Knowledge Discovery and Data Mining Explorations Newsletter*, 11(1):10–18, November 2009.

[5] K. S. Han, B. Kang, and E. G. Im. Malware classification using instruction frequencies. In *Proc. of the Research in Adaptive and Convergent Systems (RACS'11), Miami, Florida, USA*, pages 298–300. ACM Press, October 2011.

[6] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and Édouard Duchesnay. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12:2825–2830, October 2011.

[7] P. J. Rousseeuw. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of computational and applied mathematics*, 20:53–65, November 1987.

[8] D. Sin. A study on content-based information retrieval system using lsa. Master's thesis, Seoul National University, February 2000.

[9] H. P. Sung-Ha Choi. An automated classification technique for android application based on software montage. *Journal of KIISE: Computing Practices and Letterss*, 18(11):756–761, November 2012.

Author Biography

Jonghwa Park received the B.E. in Dept. of Software Science from Dankook University in 2015. He is currently a master student at Dept. of Computer Science and Engineering in Dankook University, Korea. His research interests include computer system security, mobile security, and software protection.

Gyoosik Kim received the B.E. in Dept. of Computer Science from Dankook University in 2016. He is currently a master student at Dept. of Computer Science and Engineering in Dankook University, Korea. His research interests include computer system security and mobile security.

Youngsup Hwang received the B.E. in Computer Engineering from Seoul National Univ. in 1989, the M.E. and the Ph.D. in Computer Engineering from POSTECH in 1991, 1997 respectively. During 1997-2002, he was a senior researcher in ETRI. He is a professor in the department of computer science and engineering, Sun Moon University, Korea. His research interests include pattern recognition, neural networks, bio-informatics, machine learning and software security.

Seong-je Cho received the B.E., the M.E. and the Ph.D. in Computer Engineering from Seoul National University in 1989, 1991 and 1996 respectively. He joined the faculty of Dankook University, Korea in 1997. He was a visiting scholar at Department of EECS, University of California, Irvine, USA in 2001, and at Department of Electrical and Computer Engineering, University of Cincinnati, USA in 2009 respectively. He is a Professor in Department of Computer Science & Engineering (Graduate school) and Department of Software Science (Undergraduate school), Dankook University, Korea. His current research interests include computer security, smartphone security, operating systems, and software protection.