J Supercomput (2018) 74:3489–3503
https://doi.org/10.1007/s11227-015-1594-6

Automatic malware mutant detection and group classification based on the n-gram and clustering coefficient

Taejin Lee[1] · Bomin Choi[1] ·
Youngsang Shin[1] · Jin Kwak[2]

Published online: 18 December 2015

Abstract The majority of recent cyber incidents have been caused by malware. According to a report by Symantec, an average of one million malicious codes is found daily. Automated static and dynamic analysis technologies are generally applied to cope with this, but most of the new malicious codes are the mutants of existing malware. In this paper, we present technology that automatically detects the n-gram and clustering coefficient-based malware mutants and that automatically groups the different types of malware. We verified our system by applying more than 2600 malicious codes. Our proposed technology does more than just respond to malware as it can also provide the ground for the effective analysis of new malware, the trend analysis of a malware group, the automatic identification of specific malware, and the analysis of the estimated trend of an attacker.

Keywords Malicious code · Mutant · n-Gram · Clustering coefficient

✉ Jin Kwak
jkwak.security@gmail.com

Taejin Lee
tjlee@kisa.or.kr

Bomin Choi
bmchoi@kisa.or.kr

Youngsang Shin
ysshin@kisa.or.kr

[1] Korea Internet and Security Agency, Seoul, Korea

[2] Department of Cyber Security, College of Information Technology, Ajou University, Suwon, Korea

1 Introduction

The majority of recent cyber incidents have been caused by malware. The major cyber incidents in Korea such as 7.7 DDoS in 2009, 3.4 DDoS in 2011, and the 6.25 cyber-attack in 2013 were all caused by malware. Such malicious codes are increasing drastically every year. According to the 2014 Symantec Security Intelligence Report, the number of malicious codes in 2014 increased by 26 % from 2013 to one million new malicious codes on a daily average and 317 million for the year [1]. The companies that analyze malware have been distributing technologies to quickly collect and respond to the numerous new malicious codes that appear every day. However, most of the one million new malicious codes on the average daily basis are not new types of malware, but are mutants of malicious codes already collected and managed. Aside from analyzing each malicious code and responding to it, analyzing the malware-mutant relationship will make it possible to intelligently analyze the significance, which was not possible in the past [2].

The benefits of intelligent analysis can be summarized as follows: first, static analysis and dynamic analysis are used to determine the maliciousness of malware, but there is the limitation of being able to accurately detect it. If an analyzed code is found to be similar to tens of known malicious codes, it will help lower the rate of incorrect/missed detection of malicious malware. Second, since a malware mutant is produced by reusing and varying the existing code, the analysis result of malware mutants provides the grounds to estimate if an intrusion attack is the action of the same attacker. Third, it can provide the priority to analyze and respond to the one million malicious codes that appear each day. When a malware with significant destructive force is registered in advance, an alarm can be automatically activated to enable a priority response when its mutant is identified among so many malicious codes. In the same way, a low priority can be assigned to malware, such as downloader and dropper, that is not as destructive. Fourth, observing the change of all the analyzed malware groups over time can lead to the analysis of the nature of recent malwares and the trend in changes regarding production techniques. Such malware mutant analysis technology makes it possible to not only respond to each malware but also to understand the overall meaning of analyzed malwares and to respond to them intelligently.

This paper is organized as follows: Sect. 2 introduces previous studies on the analysis of malware similarity, and Sect. 3 proposes the malware similarity and grouping technology developed in this study. Section 4 presents the test results of the developed system and typical malwares, while Sect. 5 summarizes the significance of the test results and describes the future work that we will carry out.

2 Related work

There are many ongoing studies on the analysis of malware similarity being carried out, and they can be mainly divided into the categories of static analysis and dynamic analysis. Xin Hu reported on the static analysis of the similarity of a call flow graph

Table 1 Comparison of related works and proposed model

Author	Approach	Proposed Model
Xin Hu	Call flow graph-based analysis	Behavior-based analysis
M. Alazab	Malicious API list extraction-based malware mutant analysis	Can detect malware mutant regardless of new malicious APIs
L. Wu	Regular expression-based analysis	The sequence of all and the subset are considered, as well as the frequency of APIs
Inoue	Element behavior-based analysis	Do not need to recognize the basic element of malicious behavior
Natani	Representative API and frequency-based analysis	API sequence and cosine similarity-based approach

with the original malware [3–5]. M. Alazab statically analyzed the malicious codes to extract a list of APIs that can call and measure the similarity based on it [6–8]. Although these studies apply some detection performance, their effectiveness is limited as the majority of malwares are packed (Table 1). Even if the known packer is unpacked in advance, it is still difficult to respond to a custom packing set by the attacker [9].

In the dynamic analysis of malware, there have been various studies based on the called API sequence [10]. U. Bayer cataloged the malicious behaviors of more than 90,000 malwares to provide the foundation to analyze the behaviors generated by many malwares [11], while L. Wu transformed the API sequence into a regular expression and detected the malicious code when a similar pattern of regular expressions occurred [12]. Inoue and Daisuke analyzed the unit function and malicious behavior of called API sequences in advance and determined the maliciousness based on the generation of the same pattern [13–16]. Although such studies enable the analysis based on the elemental malicious behavior identification data of malwares, there is the significant possibility of the incorrect identification of elemental malicious behaviors, and there is the limitation of not being able to detect new malware that is different from the known pattern. Natani and Pratiksha identified APIs that are frequently called by malicious codes and their frequencies in advance and analyzed malwares based on them [17,18]. There has also been a study that compared the calling sequence in the edit-distance method. The method of using leading APIs and their frequencies is limited in that API selection is not easy as normal files also use the APIs used by malicious codes and that the simple frequency can often result in incorrect detection. In addition, although the edit-distance-based similarity comparison can be effective for identifying the similarity of an entire API sequence of malicious code. However, the malware mutants not only reuse the known codes, but also include the additional malicious functions. Thus, comparing the whole sequence can be limited in measuring the accurate result. Furthermore, these studies only address the comparison of the similarity between two malwares but do not include the automatic identification of malware groups. In this paper, Sect. 3 presents the technology to group the malware similarities.

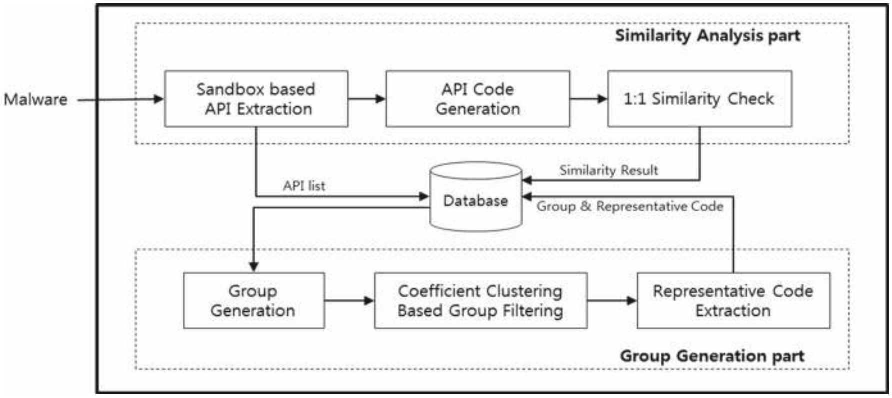
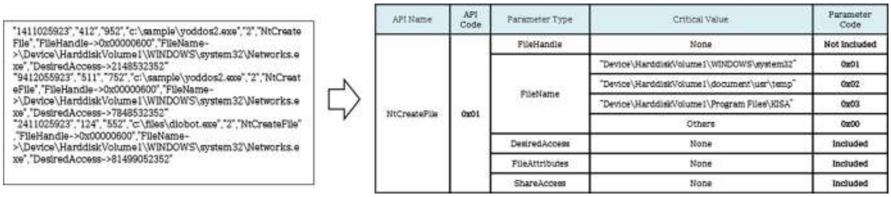


Fig. 1 System overview



```
"1411025923","412","952","c:\sample\yoddos2.exe","2","NtCreate
File","FileHandle->0x00000600","FileName-
>\Device\HarddiskVolume1\WINDOWS\system32\Networks.e
xe","DesiredAccess->2148532352"
"9412055923","511","752","c:\sample\yoddos2.exe","2","NtCreat
eFile","FileHandle->0x00000600","FileName-
>\Device\HarddiskVolume1\WINDOWS\system32\Networks.e
xe","DesiredAccess->7848532352"
"2411025923","124","552","c:\files\diobot.exe","2","NtCreateFile"
,"FileHandle->0x00000600","FileName-
>\Device\HarddiskVolume1\WINDOWS\system32\Networks.e
xe","DesiredAccess->81499052352"
```

API Name	API Code	Parameter Type	Critical Value	Parameter Code
NtCreateFile	0x01	FileHandle	None	Not Included
		FileName	"Device\HarddiskVolume1\WINDOWS\system32"	0x01
			"Device\HarddiskVolume1\document\usr\temp"	0x02
			"Device\HarddiskVolume1\Program Files\KISA"	0x03
			Others	0x00
		DesiredAccess	None	Included
		FileAttributes	None	Included
		ShareAccess	None	Included

Fig. 2 API extraction and code generation

3 Proposed scheme

3.1 System overview

This section describes the overall structure of the proposed technology. When a malware attacks, the system uses the Cuckoo Sandbox method to collect the API behavior data that are present when the malware is executed [19–21]. As the APIs are typically called in 2000–20,000 sequences, they are transformed into the formatted codes and the sequences of the transformed API data are grouped using an n-gram. The similarities between malicious codes are then calculated according to the frequency of the API sequence and the malware mutant groups are created based on them. Figure 1 shows the structure of the system.

3.2 Malware mutant detection

The malware similarity analysis is based on the API calls generated when the malware is executed. The API calls are collected using Cuckoo Sandbox. For effective calculation, the call data are separately stored and managed according to the API codes and parameters. Figure 2 shows an example of API call data and its codification.

Since the performance of a similarity comparison algorithm depends on how much it reflects the characteristics of the malware mutant, the following facts must be con-

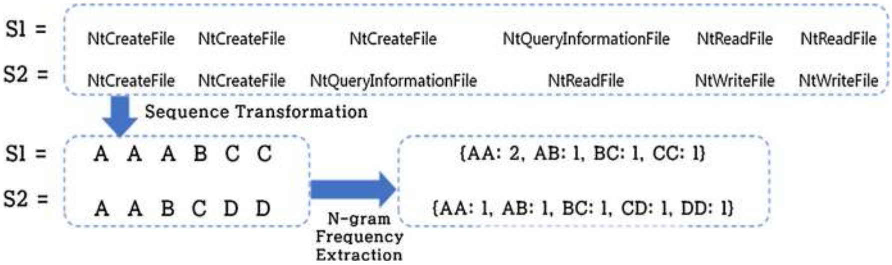

S1 =
NtCreateFile NtCreateFile NtCreateFile NtQueryInformationFile NtReadFile NtReadFile
S2 =
NtCreateFile NtCreateFile NtQueryInformationFile NtReadFile NtWriteFile NtWriteFile
Sequence Transformation
S1 = A A A B C C
S2 = A A B C D D
N-gram Frequency Extraction
{AA: 2, AB: 1, BC: 1, CC: 1}
{AA: 1, AB: 1, BC: 1, CD: 1, DD: 1}

Fig. 3 n-gram-based comparison set extraction

sidered: first, since a malware mutant is produced reusing the same code, the same API call sequences appear in the reused model. However, additional codes, such as the additional functions, are added to the common part. Thus, the sequence similarity of the subset and not the entire API sequence is important. Second, there is a limitation when using pattern-based detection since each malicious code uses different common API sequences. Therefore, the common subset must be automatically identified in all API sequences. Moreover, the system should also consider the fact that there can be multiple such subsets in different locations. Third, if the common API sequence occurs frequently, the weight factor in exponential form should be reflected in consideration of the statistical value of the frequency. In this paper, we reflect these characteristics and analyze the similarity among malicious codes in an n-gram-based comparison of the cosine similarities between API sequence subsets. The n-gram extracts the comparison sets as shown below (Fig. 3).

The cosine similarity method, which is used for comparing the similarity of two vectors, is used to compare the API sequences extracted in the n-gram method. The cosine similarity reflects the characteristics of the malware mutant described above. The calculated similarity has the value between 0 and 1, and the similarity mutant is determined based on the threshold value. As I described, malware mutant is generated with the same codes belong to the original malware. There is no clear definition but it is tightly related to the code reuse. Therefore, similarity threshold may be changed according the purposes. In the real world, if we want to find the same attacker's malware mutant for the particular malware, it needs the tight threshold. If we want to analyze the entire malware group trends, the lower threshold is appropriate. The equation for calculating the similarity between two malicious codes is shown as:

$$\text{Similarity} = \cos(\theta) = \frac{A \cdot B}{||A|| \, ||B||} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \times \sqrt{\sum_{i=1}^{n} B_i^2}}.$$

3.3 Malware group classification

The method of measuring the similarities among the malicious codes is described above. The method enables a total comparison of a large volume of malicious codes. Although the result of the total comparison of each malicious code is useful for listing

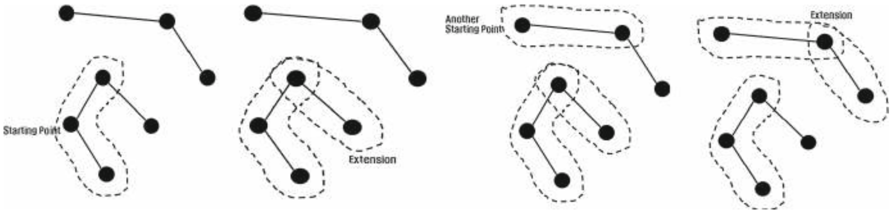


Fig. 4 Group generation step

Table 2 Group generation algorithm

```
for 1-n, each X(i) means malicious code
Each Group initialized.

for i=1 to n
  for j=i+1 to n
    if Similarity_check(X(i), X(j)) = true then
        if Find_Group(i,j) then Group.add(i,j)
        else new group is generated and Group.add(i,j)

Eliminate the duplicated Group member.
```

the codes in a manner similar to a malicious code, it does not provide much more significance than that. This paper presents a technique of grouping the malicious code groups based on the malware mutants. Figure 4 shows how the malware mutants are grouped.

First, all “n” malicious codes that are similar to a malicious code are grouped. Then, if a malicious code is similar to any of the malicious codes in the group, the similar malicious code is added to the group. If a malicious code is not similar to any other malicious codes, it is then categorized into a new group. Table 2 provides a summary of the malicious code grouping algorithm.

A malware mutant group means that all malicious codes in the group are in mutant relations and that the group has common characteristics. This algorithm analyzes how closely each group member is related to the group and filters out the insignificant malicious cods. It uses the local clustering coefficient to analyze the closeness of a malicious code to the group. The local clustering coefficient indicates how close a member of a group is to all other nodes in the same group.

The graph $G = (V, E)$ formally consists of a set of vertices, V, and a set of edges, E, between them. Here, a vertex means each malicious code, while $S_{i,j}$ means the similarity value obtained by the cosine similarity calculation of the malicious codes v_i and v_j. The symbol, t, means the threshold of the similarity value to determine the malware mutant. For example, if $S_{i,j}$ is larger than t, the malicious codes v_i and v_j are considered to be the malware mutants. When two malicious codes are determined to be the malware mutants, the vertices are connected by an edge. This can be expressed as follows:

$$e_{i,j} = \{e_{i,j} : S_{i,j} \geq t\} \quad \text{for every } i,\ j$$

Now, the equation for the local clustering coefficient is used to calculate the closeness of a malicious code to the group. Assuming that the vertex determined to be the mutant is N_i and that the number of all vertices to be connected to v_i is k_i, N_i can be expressed as follows:

$$N_i = \{v_j : e_{i,j} \in E \text{ and } e_{j,i} \in E\} \quad \text{for every } j$$

k_i means the number of edges that can be connected with v_i and the number of all the edges is $\frac{k_i \times (k_i - 1)}{2}$. Therefore, the local clustering coefficient of v_i can be defined as follows:

$$C_i = \frac{1}{n}\sum_{i=1}^{n} \frac{2|\{e_{jk} : v_j, v_k \in N_i, e_{jk} \in E\}|}{k_i\,(k_i - 1)} \bar{C} = \frac{1}{n}\sum_{i=1}^{n} C_i$$

C_i is a value between 0 and 1, which indicates how close v_i is to the group. If C_i of a malicious code is 0.9, the malicious code is clearly qualified to be a member of the group, while C_i of 0.4 means that the malicious code should be excluded from the group. If the average $\bar{C}$ of all the vertices of a group is 0.9, it indicates that the group is properly formed of the members of a group. If $\bar{C}$ is 0.5, it indicates that the malicious codes are incorrectly grouped. So far, we have described the grouping of malwares and indexes indicating the suitability of each member of the group. Grouping is determined by t, which is the threshold that confirms the similarity between malicious codes. In other words, there are groups that are formed by the value of t and the vertices existing in each group. Assuming that $G_1, G_2 \ldots G_n$ are the top n members of a malware group and that $\overline{C_{G,\iota}}$ is the average local cluster coefficient of G_i, the suitability of grouping can be determined by $\overline{C_G}$. Moreover, assuming that the number of malicious codes in a group is G_{ratio}, $\overline{C_G}$ can be expressed as follows:

$$\overline{C_G} = \frac{1}{n}\sum\nolimits_{i=1}^{n} \overline{C_{G,\iota}}\, G_{\text{ratio}} = \frac{\text{the number of malwares belonged to any group}}{\text{the number of all the malwares}}$$

Here, $\overline{C_G}$ increases, while G_{ratio} decreases as t increases. Therefore, the optimum value of threshold t needs to be set.

The next section describes the system development and the verification results from dealing with malicious codes.

4 Experimental results

4.1 Malware similarity analysis

We tested the system to check if it can automatically detect financial extortion malicious codes, which are known to be the malware mutants. The detection result is represented by a total comparison of 39 malicious codes. Table 3 shows examples of the similarity results amongst 39 malicious codes. It indicates that most of them have similarities.

Table 3 Similarity results between different malwares

	java.exe_db5..	lb.exe_7c2..	llp.exe_152..	nn.exe_233..	ops.exe_29a..	ops.exe_4ce..	opy.exe_82d..	pop.exe_cdd..	pop.exe_f23..	pos.exe_3b2..
java.exe_db5..	1	0.309	0.332	0.868	0.866	0.199	0.288	0.867	0.867	0.191
lb.exe_7c2..	0.309	1	0.646	0.198	0.258	0.3	0.706	0.196	0.196	0.145
llp.exe_152..	0.332	0.646	1	0.324	0.307	0.587	0.779	0.322	0.322	0.686
nn.exe_233..	0.868	0.198	0.324	1	0.882	0.257	0.266	0.999	0.999	0.284
ops.exe_29a..	0.866	0.258	0.307	0.882	1	0.17	0.24	0.882	0.882	0.192
ops.exe_4ce..	0.199	0.3	0.587	0.257	0.17	1	0.7	0.252	0.252	0.8
opy.exe_82d..	0.288	0.706	0.779	0.266	0.24	0.7	1	0.261	0.261	0.533
pop.exe_cdd..	0.867	0.196	0.322	0.999	0.882	0.252	0.261	1	1	0.28
pop.exe_f23..	0.867	0.196	0.322	0.999	0.882	0.252	0.261	1	1	0.28
pos.exe_3b2..	0.191	0.145	0.686	0.284	0.192	0.8	0.533	0.28	0.28	1
pos.exe_5f7..	0.179	0.363	0.519	0.224	0.148	0.846	0.844	0.218	0.218	0.7
pos.exe_838..	0.179	0.363	0.519	0.224	0.148	0.846	0.844	0.218	0.218	0.7
pos.exe_ca9..	0.199	0.3	0.587	0.257	0.17	1	0.7	0.252	0.252	0.8
pos.exe_fe5..	0.199	0.3	0.587	0.257	0.17	1	0.7	0.252	0.252	0.8
pup.exe_609..	0.179	0.141	0.273	0.259	0.175	0.232	0.242	0.258	0.258	0.255
si.exe_337..	0.871	0.474	0.524	0.929	0.847	0.265	0.425	0.929	0.929	0.27
si.exe_71c..	0.866	0.196	0.322	0.999	0.882	0.252	0.261	0.999	0.999	0.28
si.exe_727..	0.867	0.196	0.322	0.999	0.882	0.252	0.261	1	1	0.28
stup.exe_0eb..	0.554	0.028	0.075	0.464	0.692	0.082	0.054	0.464	0.464	0.106
xx.exe_f67..	0.867	0.196	0.322	0.999	0.882	0.252	0.261	1	1	0.28

Table 3 continued

	pos.exe_5f7..	pos.exe_838..	pos.exe_ca9..	pos.exe_fe5..	pup.exe_609..	si.exe_337..	si.exe_71c..	si.exe_727..	stup.exe_0eb..	xx.exe_f67..
java.exe_db5..	0.179	0.179	0.199	0.199	0.179	0.871	0.866	0.867	0.554	0.867
lb.exe_7c2..	0.363	0.363	0.3	0.3	0.141	0.474	0.196	0.196	0.028	0.196
llp.exe_152..	0.519	0.519	0.587	0.587	0.273	0.524	0.322	0.322	0.075	0.322
nn.exe_233..	0.224	0.224	0.257	0.257	0.259	0.929	0.999	0.999	0.464	0.999
ops.exe_29a..	0.148	0.148	0.17	0.17	0.175	0.847	0.882	0.882	0.692	0.882
ops.exe_4ce..	0.846	0.846	1	1	0.232	0.265	0.252	0.252	0.082	0.252
opy.exe_82d..	0.844	0.844	0.7	0.7	0.242	0.425	0.261	0.261	0.054	0.261
pop.exe_cdd..	0.218	0.218	0.252	0.252	0.258	0.929	0.999	1	0.464	1
pop.exe_f23..	0.218	0.218	0.252	0.252	0.258	0.929	0.999	1	0.464	1
pos.exe_3b2..	0.7	0.7	0.8	0.8	0.255	0.27	0.28	0.28	0.106	0.28
pos.exe_5f7..	1	1	0.846	0.846	0.218	0.229	0.218	0.218	0.071	0.218
pos.exe_838..	1	1	0.846	0.846	0.218	0.229	0.218	0.218	0.071	0.218
pos.exe_ca9..	0.846	0.846	1	1	0.232	0.265	0.252	0.252	0.082	0.252
pos.exe_fe5..	0.846	0.846	1	1	0.232	0.265	0.252	0.252	0.082	0.252
pup.exe_609..	0.218	0.218	0.232	0.232	1	0.263	0.258	0.258	0.071	0.258
si.exe_337..	0.229	0.229	0.265	0.265	0.263	1	0.929	0.929	0.395	0.929
si.exe_71c..	0.218	0.218	0.252	0.252	0.258	0.929	1	0.999	0.464	0.999
si.exe_727..	0.218	0.218	0.252	0.252	0.258	0.929	0.999	1	0.464	1
stup.exe_0eb..	0.071	0.071	0.082	0.082	0.071	0.395	0.464	0.464	1	0.464
xx.exe_f67..	0.218	0.218	0.252	0.252	0.258	0.929	0.999	1	0.464	1

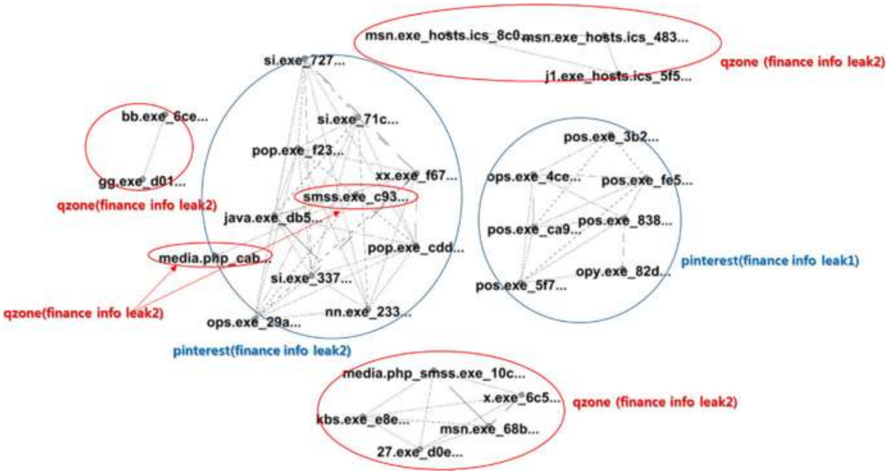


Fig. 5 Malware mutant relations

Figure 5 depicts the mutant relations of all compared samples. Each node represents a malicious code, and the nodes are connected with a line if they were found to be mutants of each other.

The detected result indicates that most of the 39 malicious codes were more than just similar and, thus, this result confirms that the financial extortion malwares can be further categorized into subgroups. We collected 2639 malwares randomly in addition to the tested 39 malware test samples and compared them to check if there are malicious codes that are similar to the test samples. The comparison detected many malwares that are in mutant relations with the 39 test samples. Table 4 shows the malwares that were identified to be in mutant relations with smss.exe and msn.exe.

4.2 Malware group classification analysis

The results of the similarity analysis of malicious codes have been described above. Multiple malwares were totally compared to manually group the malwares. If there are hundreds of thousands of malwares to be analyzed, the automatic malware grouping system is needed. This study analyzed 2639 malwares for grouping. The analysis showed 210 groups, and the top 10 groups contained 1121 malwares, which constitute around 42.8 % of the total malwares. That means that around 40 % of all malwares can be automatically categorized into 10 leading groups, and the characteristics of these groups can be analyzed in advance. This will be very useful in deducing the behavior and significance of malwares. $\overline{C_G}$, which represents the suitability of grouping, is affected by t. Figure 6 shows the grouping of 2639 malwares according to the threshold t.

As the threshold t increases, the number of members in each group decreases, but the accuracy of the members of the malware group increases. By the same token, as the threshold decreases, the number of members in each group increases, but the

Table 4 Malware mutant detection results

No.	Hash(MD5)	Component ID	Clustering coefficient
smss.exe's mutant list			
1	smss.exe_c939bd96d0dca428ae1d1617c69ff9d7	2	0.898907104
2	fddb5bcbbc7165020d54e64a341d6879	2	0.939393939
3	fd833f1f0d1ec07acd4fb2fb228c763b	2	0
4	fc8861e7c50b2d88e0a6a0f721abb37e	2	0.860887097
5	f94c3fdf02f5440ddf526712a0e9b1ff	2	1
6	f75e8139977d1e939cda9570416e034e	2	0.796536797
7	f739da8ad9cde94500c8dba13dbb0561	2	0.857142857
8	f71e727d69a49c7d19bdff28ea71e5ee	2	0.6
9	f18ba9f4c4e9eff3ff462c1afaecdc8e	2	1
10	efc4ac317608e7910f3eb4b023d06cb7	2	1
11	ea398ab7024e43e5ad734fd29e1e77f6	2	0.5
12	ea2498e819117ee10f978794f5fdbc25	2	0.666666667
13	e7a9bf89875af5369a6f38eead1c30bd	2	0.902259887
14	e66a9af5d2c1f2408c3eb70feb453df1	2	0.817956349
15	e63503dd5d6e3e9f255b3d9f0cd4de99	2	0.833333333
16	e4ce3af898d74c56ad3fedaeab7ee986	2	1
17	e33b44aa7d3a2f2562f83e622082c037	2	0.898907104
18	e2b7364425133698236ede46460d1f27	2	0.75
19	e27ef469e6aeb4f8c3d06a3126ba9997	2	0.333333333
20	dfa90d374da64764861b413f3ffa41a9	2	0.833333333

Table 4 continued

no.	Hash(MD5)	Component ID	Clustering coefficient
msn.exe's mutant list			
1	x.exe_6c524d4b40bc722b19eb23d5b6525017	13	0.824561404
2	msn.exe_68bd78fd01c117bca8b9fe6193a49d7c	13	0.824561404
3	media.php_smss.exe_10c89fbbbd678ad168fbe36a271488d7	13	0.824561404
4	kbs.exe_e8eb75d80701dad549eae429396ca1e5	13	0.824561404
5	e742f1f7231f028ca376316973941958	13	1
6	cbcf18e559b87afdd059cae1f03b18d1	13	1
7	ca4e531d111a58b8a5c9d902e374303a	13	1
8	ca316b0fbdad5c93932abbb46268286d	13	1
9	914d68177860ef41820eb669e1726dcd	13	0.824561404
10	8d950b0dcdf198b42377dc847cac46c1	13	1
11	8d146c15c56736373f022cfeb32ed73c	13	1
12	756dcd2630adf6b32805adeb7883b99f	13	1
13	6f73192d0ab23beae5f749e98402b869	13	1
14	5b52b7ce3dd4b3599595d32267181727	13	0.824561404
15	45f98789a0ae0185ea1f27124da6122b	13	1
16	4168a379af36d826683e024908a45756	13	1
17	3a4146b96f75c7215090f02e7a7419ca	13	1
18	27.exe_d0e54fea46b9a4e2a964e5bce172496e	13	0.824561404
19	21d05787123326b589ac82d4e4437557	13	0.824561404
20	05ccf1e293f40ec1d811e3b519e3fd89	13	0.824561404

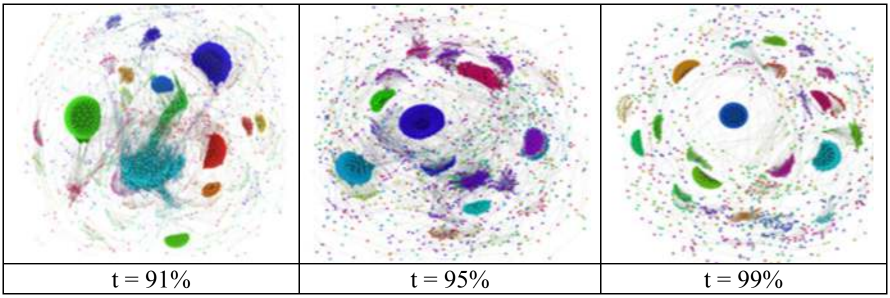


Fig. 6 Malware group visualization. $t = 91\,\%$, $t = 95\,\%$, $t = 99\,\%$

Table 5 G_{ratio}, $\overline{C_G}$ relation according to the threshold t

Threshold t (%)	No. of group members	No. of groups	G_{ratio} (%)	$\overline{C_G}$
99	1827	246	69.231	0.9963
98	1932	243	73.210	0.9585
97	2005	232	75.976	0.9341
96	2065	213	78.249	0.9065
95	2065	213	78.249	0.9065
94	2100	195	79.576	0.8979
93	2203	172	83.479	0.8615
92	2240	156	84.881	0.8695
91	2265	152	85.828	0.8712
90	2285	138	86.586	0.8804

accuracy of the members of the malware group decreases. Table 5 shows the changes of the G_{ratio} and $\overline{C_G}$ according to the threshold t through the test.

For example, if 90 % of the $\overline{C_G}$ is needed, the threshold t can be set to 95 %, and this means that 78 % of malicious codes can be grouped. With a 2.5 GHz CPU server, it took an average of 0.0097 seconds to compare the similarities of malicious codes, which means that approximately nine million malicious codes can be analyzed daily.

5 Conclusions

Malware is the key cause of cyber intrusion incidents. Since malware is continuously enhanced and concealed, its countermeasures are also being constantly studied. The malware similarity analysis and automatic grouping technology play the very important role of making the countermeasures more effective. They can automatically identify the key malwares from amongst a huge volume of malicious codes, a million of which are collected each day, and can automatically filter out the less destructive malicious codes, such as the downloader and dropper, and the malware mutants are useful in studying the trends and patterns of the same attackers. Although malware analysis companies manage them in their own ways, the development of a general purpose technology is still far-off. This paper has confirmed the usefulness of the proposed technology through verification, but the development of technology for selective comparison instead of total comparison is needed to analyze new malicious codes, so that it can be utilized in the general environment. We are considering the following approaches. We will select the representative malwares of each group and it is updated periodically according to the each malware's local clustering coefficient. If the new malware arrives, it is compared the representative malwares of the each group and it is compared entirely within only one group. This paper also analyzed the accuracy of grouping using $\overline{C_G}$ according to the threshold t. However, further studies are required since it does not represent the individual accuracy of each group member [22,23]. We intend to continue operating the developed system in general environments and to improve the system.

Acknowledgements This work was supported by the Institute for Information and communications Technology Promotion(IITP) grant funded by the Korea government (MSIP) (No.R0101-15-0175, The Development of Cyber Attacks Detection Technology based on Mass Security Events Analysing and Malicious Code Profiling).

Compliance with ethical standards

Conflict of interest The authors declare that there is no conflict of interests regarding the publication of this paper.

References

1. Symantec. http://www.symantec.com
2. Mohaisen A, Alrawi O (2013) Unveiling zeus: automated classification of malware samples. In: 22nd international conference on world wide web companion, pp 829–832
3. Xin H, Tzi-cker C, Shin Kang G (2009) Large-scale malware indexing using function-call graphs. CCS'09, November 9–13
4. Elhadi AAE, Maarof BB (2013) Improving the detection of malware behavior using simplified data dependent api call graph. Int J Secur Appl
5. Dullien T, Rolles R (2004) Graph-based comparison of executable objects. In: IEEE conference on detection of intrusions and malware & vulnerability assessment (DIMVA-2004), pp 161–173
6. Alazab M et al (2010) Towards understanding malware behaviour by the extraction of API calls. In: CTC 2010 second. IEEE, New York
7. Galal Hisham S, Mahdy YB, Atiea MA (2015) Behavior-based features model for malware detection. J Comput Virol Hack Techniq
8. Miao Q, Liu J, Cao Y et al (2015) Malware detection using bilayer behavior abstraction and improved one-class support vector machines. Int J Inf Secur, pp 1–19. doi:10.1007/s10207-015-0297-6
9. Moser A, Kruegel C, Kirda E (2007) Limits of static analysis for malware detection. In: Proceedings of the 23rd annual computer security applications conference
10. Youngjoon K, Eunjin K, Huy Kang K (2015) A novel approach to detect malware based on API call sequence analysis. Int J Distrib Sens Netw
11. Bayer U et al (2008) A view on current malware behaviors. In: USENIX workshop on large-scale exploits and emergent threats (LEET)
12. Wu L et al (2011) Behavior-based malware analysis and detection. In: 2011 first international workshop on complexity and data mining (IWCDM). IEEE, New York, pp 39–42

13. Inoue D et al (2008) Malware behavior analysis in isolated miniature network for revealing malware's network activity. In: IEEE international conference on communications, ICC'08. IEEE, New York
14. Cesare S, Xiang Y (2012) Software similarity and classification. In:Springer Briefs in Computer Science 2012. Springer Science & Business Media, Berlin
15. Sathyanarayan VS, Kohli P, Bruhadeshwar B (2008) Signature generation and detection of malware families. In: Information security and privacy. Springer, Berlin
16. Kephar JO, Arnolod WC (1994) Automatic extraction of computer virus signatures. In: 4th virus bulletin international conference, pp 179–194
17. Pratiksha N, Deepti V (2013) Malware detection using API function frequency with ensemble based classifier. In: Security in computing and communications. Springer, Berlin
18. Shankarapani MK, Ramamoorthy S, Movva RS, Mukkamala S (2011) Malware detection using assembly and API calls sequences. J Comput Virol
19. VirusTotal. https://www.virustotal.com
20. National Software Research Library. http://www.nsrl.nist.gov
21. Cuckoo Sandbox. http://www.cuckoosandbox.org
22. Rieck K, Trinius P, Willems C, Holz T (2011) Automatic analysis of malware behavior using machine learning. J Comput Secur, pp 639–668
23. Mojtaba E, Zeinab KH (2013) HDM-analyser: a hybrid analysis approach based on data mining techniques for malware detection. Sattar J Comput Virol Hack Techniq