Chapter 60
Malware Classification Methods Using API Sequence Characteristics

Kyoung-Soo Han, In-Kyoung Kim and Eul Gyu Im

Abstract Malware is generated to gain profits by attackers, and it infects many users' computers. As a result, attackers can acquire private information such as login IDs, passwords, e-mail addresses, cell-phone numbers and banking account numbers from infected machines. Moreover, infected machines can be used for other cyber-attacks such as DDoS attacks, spam e-mail transmissions, and so on. The number of new malware discovered every day is increasing continuously because the automated tools allow attackers to generate the new malware or their variants easily. Therefore, a rapid malware analysis method is required in order to mitigate the infection rate and secondary damage to users. In this paper, we proposed a malware variant classification method using sequential characteristics of API used, and described experiment results with some malware samples.

Keywords Malware · Malware analysis · Malware classification

60.1 Introduction

Establishing information networks have been largely extended with the distribution of computers and the Internet throughout the world. In addition, malware, such as viruses, worms, trojans, etc., has been increased with such increased uses

K.-S. Han · I.-K. Kim
Department of Electronics Computer Engineering,
Hanyang University, 17, Haengdang-dong, Seongdong-gu,
Seoul 133-791, South Korea

E. G. Im (✉)
Division of Computer Science and Engineering,
Hanyang University, 17, Haengdang-dong, Seongdong-gu,
Seoul 133-791, South Korea
e-mail: imeg@hanyang.ac.kr

K. J. Kim and S. J. Ahn (eds.), *Proceedings of the International Conference on IT Convergence and Security 2011*, Lecture Notes in Electrical Engineering 120,
DOI: 10.1007/978-94-007-2911-7_60,

of computers and the Internet. Attackers can easily generate new or modified malware using automated malware generation tools that leads to increased numbers and types of malware. However, countermeasures against such malware have been fallen behind compared to the rate at which malware is created. Moreover, the severities of direct and secondary damages caused by such malware have been significantly increased.

In this paper, we propose a system that detects and classifies malware variants using the sequence characteristics of API used. As malware has a PE (Portable Executable) file format executed in the Windows operating system (OS), our proposed method can identify the types of malware variants by extracting the API list included in the malware through a static analysis and calculating its similarities. In the experimental results, the similarities in the API sequential characteristics for the Trojan malware samples of Trojan-DDoS and Trojan-Spy are compared and described.

This paper consists of six sections. Section 60.2 introduces analysis methods of APIs and malware and preliminaries. Section 60.3 describes techniques related to detect and classify malware. Section 60.4 proposes a classification method of malware variants using the sequential characteristics in the API list. Section 60.5 shows the experimental results using the proposed method, and Sect. 60.6 represents the conclusion and the direction of future works.

60.2 Preliminaries

60.2.1 API

API (Application Programming Interface) is an interface that controls the functions provided by operating systems or programming languages in application programs and supports interactions with other application programs or operating systems [1]. It is provided as Windows APIs for application programs in the Windows operating system. Windows APIs are operated in user or kernel modes. In particular, APIs operated in the kernel mode are called Native APIs [2]. Table 60.1 shows the examples of APIs included in the major DLL of the Windows user mode.

60.2.2 Extraction Methods of the API List

Kernel hooking and IAT (Import Address Table) analysis methods are used to extract the API list. First, the kernel hooking is divided into SSDT (System Service Descriptor Table) hooking and IDT (Interrupt Descriptor Table) hooking and these are usually used to acquire the information of Native APIs. The SSDT hooking

Table 60.1 The example of DLLs and APIs

Kernel32.dll	Description	Processing of all works provided by Windows kernel—Memory management, File I/O, Process management
	Example APIs	LoadLibraryA, GetCurrentProcess, ExitProcess, TerminateProcess, GetFileType, CreateFileA, WriteFile
User32.dll	Description	User interface—Processing of all controls in Windows
	Example APIs	MessageBoxA, CreateWindowExA, SetCapture, SendMessageA
gdi32.dll	Description	Graphic user interface—Output of text and graphics
	Example APIs	DeleteDC, SetTextColor, GetWindowOrgEx, GetTextMetricsA, GetTextExtentPointA

extracts the used Native API list by modifying the memory address of the function employed in the table to get services in the kernel mode or to redirect the table itself to the memory inside the program. The IDT hooking changes the table used to process interrupts. Here, the information of Native APIs can be extracted by hooking a specific interrupt through changing the interrupt process inside IDT [2–4]. Second, the IAT analysis uses the table that stores the API information used by application programs. The PE file stores the names of APIs, which are used through DLLs, and the addresses, which are allocated during its execution [3]. The IAT analysis is applied to IAT (Import Address Table), which is an array of API entry points. Thus, it is possible to find IATs by analyzing PE files and to extract the API list from IATs. In this paper, the API list of malware is extracted using this method.

60.3 Related Work

60.3.1 Malware Analysis Methods

Both dynamic analysis method and static analysis method can be used to detect and classify malware. The dynamic analysis monitors and traces malicious behaviors and dangerous elements by executing malware in an environment. The static analysis does not execute malware but analyzes binary instructions to identify the structure of malware and APIs used. This method is usually used in the software reverse engineering fields [3].

60.3.2 Detection and Classification Methods Based on a Dynamic Analysis

The malware detection method based on a dynamic analysis requires analyzing characteristics malware behaviors.

Park et al. [5] generated behavior-based detection patterns for malware, and proposed a method to detect new malware and malware variants. The proposed method used both dynamic and static analyses to extract APIs which access the system. Then, the method generates signatures to detect malware dynamically by associating the behaviors of malware with APIs

Fredrikson et al. [6] proposed a method that automatically extracts the characteristics of behaviors presented in malware by using graph mining. The proposed method classified the malware, which represents similar malicious behaviors, and made clusters by identifying core control flow graphs for each similar malicious behavior which represent a malware family. Then, it generalized control flow graphs as a significant behavior for a malware family.

Miao et at. [7] developed 'API Capture' that records the major characteristics automatically, such as system call arguments, return values, error conditions, and so on, by monitoring behaviors of malware based on an emulator. Nair et al. [8] traced API calls in malware via dynamic monitoring within an emulator to detect malware generated from a metamorphic generator and measured its frequencies to extract CAPI (Critical API). In addition, the differences among CAPI rates were calculated using a statistical method after generating signatures.

60.3.3 Detection and Classification Methods Based on a Static Analysis

Various control flow graph based methods are used to define specific signatures for malware. Lee et al. [9] defined the call flows as signature obtained by analyzing malicious binaries and proposed a detection method. This method analyzed relationships of system calls in malware and represented it as a call graph. Then, it configures 32 system calls including process/memory/socket into API groups, to simplify system call to a code graph. Then, it is used to calculate similarities to detect malware variants. Cesare et al. [10] generated a flow graph signature using a control flow graph and calculated similarities.

Zhang et al. [11] proposed a method that generates some patterns by specifying the semantics and functionalities in malware binaries, which represent a specific malicious behaviors based on the system calls and library function calls of metamorphic malware, which is then analyzed using a code pattern matching technique that calculates similarities. Karnik et al. [12] represented a list of functions to vectors to compare two different files based on a static analysis for PE files. Then, a method for measuring similarities using a cosine similarity analysis after arranging the vectors as a descending order based on the instruction frequencies in functions was proposed.

Cha et al. [13] applied a feed-forward bloom filter (FFBF) to implement the idea that performs a fast detection by expanding ClamAV [14]. It scans the entire file as a length of w using a sliding window and generates bit vectors using a

bloom filter for the scanned contents. Then, it detects malware by comparing it with the bit vectors of the existing malware presented in a database.

60.4 Proposed Method

60.4.1 Overview

In this paper, a method that classifies malware by extracting the API list from IATs (Import Address Tables), which have DLL and API information required in executing PE files, and using the type and sequential characteristics of APIs is proposed. Figure 60.1 shows the overall flow in this method.

The method that detects the similarity between two different malware types, i.e., the presence of the same malware variant by measuring the similarity in the API list extracted using a static analysis. This method performs the following process to detect malware variants.

Step 1. It extracts the API list from each malwar. In general, a program manages the API list that is to be called by including IAT inside the program. The proposed method extracts APIs in the malware from the IAT.

Step 2. It generates a whitelist by storing the API list frequently included in benign programs according to DLLs. Then, the API included in the whitelist is removed from the API list extracted from malware.

Step 3. It generates the same API included in the API list, which is the subject in comparison with the API list in which the API of the whitelist is removed. That is, the similarity between two different malware types is calculated based on the characteristics of the sequence of the same API include in both API lists.

60.4.2 API Extraction

As mentioned in Sect. 60.2, the extraction method of the API list from malware can be performed by extracting the name of APIs included in the IAT, which is obtained using a static analysis. Figure 60.2 shows the example of the search of the API included in IATs.

60.4.3 Generation of the Whitelist

The method proposed in this paper is based on an assumption in which the sequences of the APIs called in executing malware and the APIs included in IATs are similar. However, the called APIs in the execution of malware are also to be called in benign applications programs. As shown in Fig. 60.3, in the case that

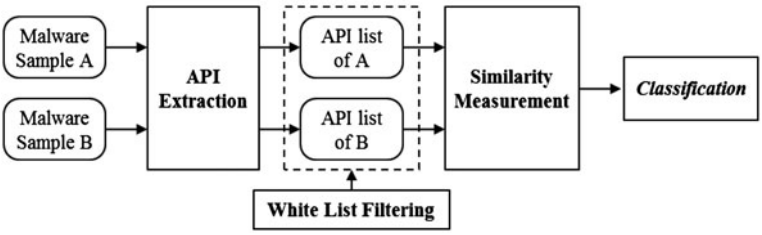


Fig. 60.1 Overview of the proposed method

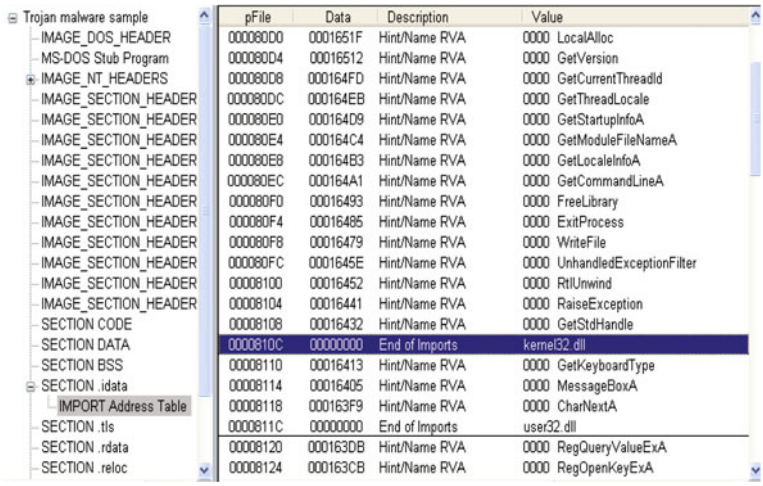


Fig. 60.2 IAT checking

represents similar functions in both malware and benign application programs, the APIs that are to be called for its execution will be included in IATs as a same manner. Therefore, the proposed method makes the APIs frequently used in various benign application programs as a whitelist in order to reduce the false positive caused by this common factor and then the whitelist can be used to implement a filtering. Then, the similarity between the malware API lists that are the subjects of the comparison is calculated.

60.4.4 Calculation of the Similarity

After removing APIs of the whitelist from the API list included in the IAT of malware, a set intersection of APIs is extracted to calculate the API similarity between two compared malware types. The objective of this paper is not to

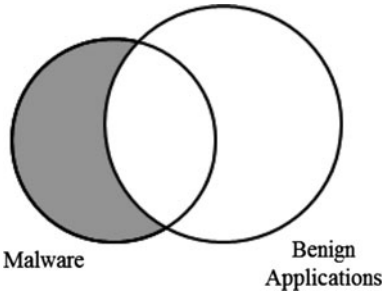


Fig. 60.3 APIs used in malware and benign applications

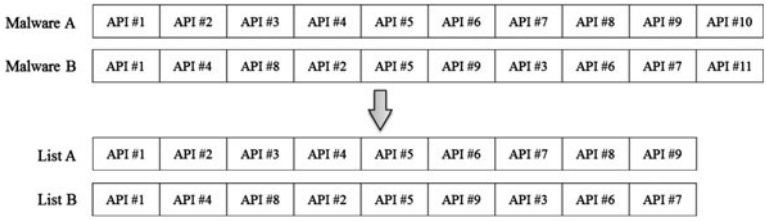


Fig. 60.4 Extraction of the intersection from 2 malware API lists

calculate the similarity by simply extracting a set intersection but to propose a method that calculates the similarity using the sequential characteristics in corresponding APIs. As represented in Fig. 60.4, the set intersection can be extracted by maintaining the existing sequence.

Then, a subset list of the original API list is generated by maintaining the sequential characteristics of the extracted set intersections. The algorithm used for this process is as follows.

Algorithm: SumOfSameSequence

```
1. SumOfSameSequence(String List A, B)
2. i, j, pre_i, pre_j, next_i, next_j ← 0
3. new String List A', B'
4. if(A.length == 1)
5.       return 0
6. end if
7.
8. while(i < A.length)
9.       while(j < B.length)
10.            if(A[i] == B[j])
11.                 i ← i + 1
12.                 pre_i ← i
```

```
13.                 j ← j + 1
14.                 while(pre_j < j)
15.                     B'[next_j] ← B[pre_j]
16.                     next_j ← next_j + 1
17.                     pre_j ← pre_j + 1
18.                 end while
19.                 pre_j ← j
20.                 go to line 4
21.             end if
22.             else
23.                 j ← j + 1
24.             end else
25.         end while
26.         A'[next_i] ← A[i]
27.         i ← i + 1
28.         j ← pre_j
29.         next_i ← next_i + 1
30. end while
31.
32. count ← A.length − A'.length
33.
34. if(count == 1)
35.         return SumOfSameSequence(A',B')
36. end if
37. else
38.         return count + SumOfSameSequence(A',B')
39. end else
```

In the proposed algorithm, the API list presented by the same sequence is removed from the original list based on the list A for identifying the sequential characteristics of the two malware types. As shown in Fig. 60.4, for instance, the APIs presented by the same sequence are removed from the lists A and B and the number of removals is counted. As API#4 and API#5 are presented after API#3, it is remained in the list because it has already presented in the list B. Then, API#6 and API#7 are removed from the list. It can be presented as a process that records APIs, which are not removed from the new lists A' and B'. In Fig. 60.5, the lists A' and B' that remove the APIs, which are presented by the same sequence, are generated. Then, the sum of the APIs presented by the same sequence in the entire set intersection by applying the same algorithm to the generated list recursively as shown in Fig. 60.6.

After calculating the sum of the API subset that has the same sequence using the proposed algorithm, the similarity can be determined by calculating the rate between the size of the subset and the union of the entire API list. It can be expressed as the following similarity calculation formula (60.1).

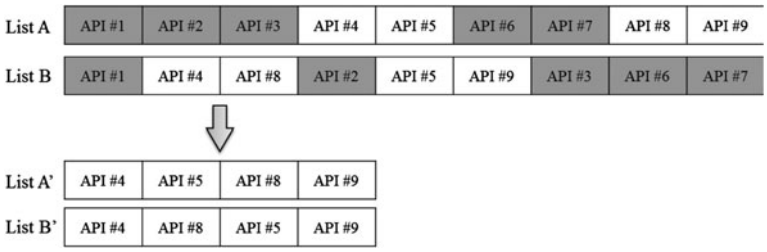
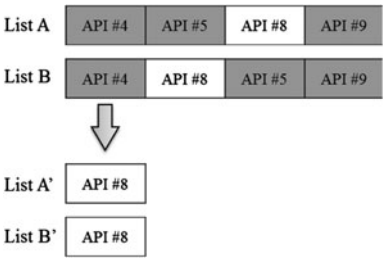


Fig. 60.5 Applied result of algorithm for API lists

Fig. 60.6 Recursively applied result of algorithm



$$P_{(A,B)} = \frac{n(\sum s(A,B))}{n(A \cup B)} \tag{60.1}$$

※ X : API List Set/n(X) : # of API List Set X/S(X,Y) : Same Sequence Subset of X,Y

It is expected that the proposed method can calculate the similarity exactly though reflecting the sequential characteristics of the API presented by the same sequence instead of calculating the similarity simply using a set intersection. In addition, because the entire APIs presented by the same sequence are extracted to calculate the similarity instead of using a longest common sub-sequence (LCSS) even though it uses a partial sum of that, the malware variants generated by changing the sequence of APIs can be classified as the same malware.

60.5 Experimental Results

60.5.1 Experiment Environment and Data

The method and algorithm proposed in this paper were implemented using the Eclipse that is a development environment in JAVA. In addition, malware samples were selectively collected from VX Heavens [15] for the experiment. The collected malware was the Trojan, which is executable in the Windows operating system. In the functional classification of the Trojan, Trojan-DDoS and Trojan-

Table 60.2 White list extraction from benign applications

Benign applications	White list
Mspaint, NateOn, Notepad, Hwp, Calcul, Explorer, iTunes, uTorrent, Excel, Groove, AlZip, AlFtp, Everything, Acrobat	BitBlt, CloseHandle, CoCreateInstance, CoTaskMemFree, DeleteCriticalSection, DeleteDC, DeleteObject, EnableWindow, EnterCriticalSection, exit, FindClose, FreeLibrary, GetACP, GetClientRect, GetCurrentProcess, GetCurrentProcessId, GetCurrentThreadId, GetCursorPos, GetDC, GetDeviceCaps, GetFocus, GetLastError, GetModuleFileNameW, GetModuleHandleA, GetModuleHandleW, GetParent, GetProcAddress, GetProcessHeap, GetSubMenu, GetSystemMenu, GetSystemMetrics, GetSystemTimeAsFileTime, GetTickCount, GetVersionExW, GlobalAlloc, GlobalFree, GlobalLock, GlobalUnlock, HeapAlloc, HeapFree, InitializeCriticalSection, InterlockedCompareExchange, InterlockedDecrement, InterlockedExchange, InterlockedIncrement, InvalidateRect, IsWindowVisible, LeaveCriticalSection, LoadLibraryA, LoadLibraryW, LocalAlloc, LocalFree, lstrlenW, memcpy, MessageBoxW, MulDiv, MultiByteToWideChar, PostQuitMessage, QueryPerformanceCounter, RaiseException, ReadFile, RegCloseKey, RegOpenKeyExW, RegQueryValueExW, ReleaseDC, ScreenToClient, SelectObject, SetActiveWindow, SetCursor, SetErrorMode, SetForegroundWindow, SetMapMode, SetUnhandledExceptionFilter, SetViewportExtEx, SetWindowPos, Sleep, TerminateProcess, UnhandledExceptionFilter, UpdateWindow, WideCharToMultiByte, WriteFile, SetLastError, VirtualAlloc, VirtualFree, GetCommandLineW

Spy were 125 and 420 respectively. The diagnosis name in each malware followed the diagnosis name determined by the Kaspersky anti-virus. The API list was extracted from the collected malware samples and the similarity was calculated for the sequential characteristics of the API list, which applies the whitelist using the implemented program.

60.5.2 White List

The proposed method determines the API list that may be included in benign application programs to a whitelist for performing an exact calculation of the similarity and reducing overheads. Table 60.2 shows the whitelist produced by the benign program used to extract the whitelist before performing this experiment. The APIs in the whitelist that were included in IATs more than five times were only filtered.

60.5.3 Results of the Similarity Calculation

In this experiment, the similarity was calculated for each Trojan-DDoS and Trojan-Spy samples using the proposed method. Regarding the process of this experiment, the similarity of the malware samples in same family was first

Table 60.3 Similarity between Trojan-Spy samples

			Trojan-Spy								
			.Zapchast				.GhostSpy		.PCSpy		
			.b	.f	.g	.i	0.52	0.40	.b	.c	.d
Trojan-Spy	.Zapchast	.b	1.000	0.414	0.414	0.414	0.171	0.207	0.071	0.060	0.065
		.f	0.414	1.000	0.414	0.414	0.161	0.232	0.057	0.070	0.068
		.g	0.414	0.414	1.000	0.414	0.161	0.232	0.057	0.070	0.068
		.i	0.414	0.414	0.414	1.000	0.161	0.232	0.057	0.070	0.068
	.GhostSpy	0.52	0.171	0.161	0.161	0.161	1.000	0.488	0.194	00,172	0.177
		0.40	0.207	0.232	0.232	0.232	0.488	1.000	0.187	0.111	0.115
	.PCSpy	.b	0.071	0.057	0.057	0.057	0.194	0.187	1.000	0.404	0.414
		.c	0.060	0.070	0.070	0.070	0.172	0.111	0.404	1.000	0.965
		.d	0.065	0.068	0.068	0.068	0.177	0.115	0.414	0.965	1.000

Table 60.4 Similarity between Trojan-DDoS samples

			Trojan-DDoS					
			.Desex		.Delf		.Boxed	
			.a	.b	.e	.i	.a	.j
Trojan-DDoS	.Desex	.a	1.000	0.625	0.118	0.032	0.042	0.036
		.b	0.625	1.000	0.056	0.000	0.041	0.071
	.Delf	.e	0.118	0.056	1.000	0.278	0.130	0.120
		.i	0.032	0.000	0.278	1.000	0.117	0.120
	. Boxed	.a	0.042	0.041	0.130	0.117	1.000	0.811
		.j	0.036	0.071	0.120	0.120	0.811	1.000

calculated, and then the similarity between these malware samples in different malware families was calculated.

Table 60.3 shows the results of the calculation of the similarities for the family samples of the Trojan-Spy, Zapchast, GhostSpy, and PCSpy. The similarities between the samples included in the same family were about more than 0.4. Also, it was verified that the similarities between the samples included in other families were below that value even though it was involved in the same Trojan-Spy.

Table 60.4 shows the results of the calculation of the similarities for the family samples of the Trojan-DDoS, Desex, Delf, and Boxed. Although the Desex and Boxed showed large similarity values, the Delf showed a small value relatively. It was due to the fact that a malware producer updated such malware for using different APIs, which perform the same function, as the producer built some variants for the malware.

Table 60.5 shows the results of the calculation of the similarities for the malware samples between the Trojan-Spy and the Trojan-DDoS. The similarities of the malware samples included in the same family were about more than 0.4 except

Table 60.5 Similarity between Trojan-Spy and Trojan-DDoS samples

			Trojan-Spy						Trojan-DDoS					
			.Zapchast		.GhostSpy		.PCSpy		.Desex		.Delf		.Boxed	
			.g	.i	0.52	0.40	.b	.c	.a	.b	.e	.i	.a	.j
Trojan-Spy	.Zapchast	.g	1.000	0.414	0.161	0.232	0.057	0.070	0.068	0.087	0.190	0.110	0.328	0.295
		.i	0.414	1.000	0.161	0.232	0.057	0.070	0.068	0.087	0.190	0.110	0.328	0.295
	.GhostSpy	0.52	0.161	0.161	1.000	0.488	0.194	0.172	0.045	0.056	0.214	0.400	0.092	0.103
		0.40	0.232	0.232	0.488	1.000	0.187	0.111	0.091	0.089	0.396	0.342	0.100	0.093
	.PCSpy	.b	0.057	0.057	0.194	0.187	1.000	0.404	0.032	0.016	0.140	0.173	0.050	0.048
		.c	0.070	0.070	0.172	0.111	0.404	1.000	0.014	0.014	0.069	0.150	0.036	0.041
Trojan-DDoS	.Desex	.a	0.068	0.068	0.045	0.091	0.032	0.014	1.000	0.625	0.118	0.032	0.042	0.036
		.b	0.087	0.087	0.056	0.089	0.016	0.014	0.625	1.000	0.056	0.000	0.041	0.071
	.Delf	.e	0.190	0.190	0.214	0.396	0.140	0.069	0.118	0.056	1.000	0.278	0.130	0.120
		.i	0.110	0.110	0.400	0.342	0.173	0.150	0.032	0.000	0.278	1.000	0.117	0.120
	.Boxed	.a	0.328	0.328	0.092	0.100	0.050	0.036	0.042	0.041	0.130	0.117	1.000	0.811
		.j	0.295	0.295	0.103	0.093	0.048	0.041	0.036	0.071	0.120	0.120	0.811	1.000

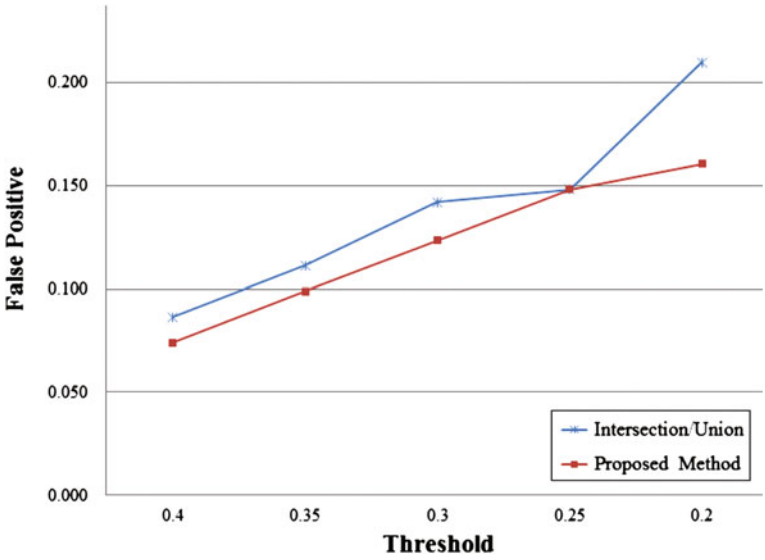


Fig. 60.7 Comparison of false positive rate

for the Trojan-DDoS.Delf. Also, the case that has different classes and families showed small similarity values. Here, the similarity between the Trojan-Spy.GhostSpy and the Trojan-DDoS.Delf; the Trojan-Spy.Zapchast and the Trojan-DDoS.Boxed showed large similarity values. It was considered that it contained the required APIs similarly for modifying the infected system by the malware including some operations, such as file and registry modification, process generation, and process termination, even though classes and families were differently classified.

Figure 60.7 shows the results of the comparison of the false positive rates. It can be generated according to the configuration of threshold values, between the calculation of the similarity based on the simple rate of a set intersection for the union of the APIs included in the IAT of the malware samples and the calculation performed by using the proposed method. As the threshold value was determined as 0.4, the false positive rate in the proposed method was 0.074. It was verified that the value was lower than that of the calculation based on the rate of a set intersection for the union.

60.6 Conclusion and Future Work

In this paper, we proposed a method that classifies malware variants using the sequential characteristics of the API list. The proposed method generate a whitelist for APIs which can be included in benign application programs, and it is used to reduce the overheads during the calculation of similarities. The proposed method

and similarity calculation algorithm were implemented. Also, we perform the experiments for the collected malware samples. As a result, the similarities of the samples included in the same family were about more than 0.4. As the threshold values were determined as $0.4 \sim 0.2$, the false positive rates were presented as $0.074 \sim 0.160$. However, it is necessary to conduct some future work on the more improved malware classification by complementing the existing algorithm using the proposed method for different malware samples in various classes.

Acknowledgements This work was supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MEST) (No. 20110029924).

References

1. Petzold C (1998) Programming microsoft windows, 5th edn. Microsoft Press, London
2. Wang M, Zhang C, Yu J (2006) Native API based windows anomaly intrusion detection method using SVM. In: Proceedings of IEEE international conference on sensor networks, ubiquitous, and trustworthy computing, vol 1, pp 514–519
3. Hoglund G, Butler J (2005) Rootkits: subverting the windows kernel. Addison-Wesley, Reading
4. Willems C, Holz T, Freiling F (2007) Toward automated dynamic malware analysis using CWSandbox. IEEE Secur Privacy 5(2):32–39
5. Park N, Kim Y, Noh B (2006) A behavior based detection for malicious code using obfuscation technique. J KIISC 16(3):17–28
6. Fredrikson M, Jha S, Christodorescu M, Sailer R, Yan X (2010) Synthesizing near-optimal malware specifications from suspicious behaviors. In: Proceedings of the 2010 IEEE symposium on security and privacy, pp 45–60
7. Miao Q, Wang Y, Cao Y, Zhang X, Liu Z (2010) APICapture—a tool for monitoring the behavior of malware. In: Proceedings of the 3rd international conference on advanced computer theory and engineering, pp 390–394
8. Nair VP, Jain H, Golecha YK, Gaur MS, Laxmi V (2010) MEDUSA: metamorphic malware dynamic analysis using signature from API. In: Proceedings of the 3rd international conference on security of information and networks, pp 263–269
9. Lee J, Jeong K, Lee H (2010) Detecting metamorphic malwares using code graphs. In: Proceedings of the 2010 ACM symposium on applied computing, pp 1970–1977
10. Cesare S, Xiang Y (2010) A fast flowgraph based classification system for packed and polymorphic malware on the endhost. In: Proceedings of the 24th IEEE international conference on advanced information networking and applications, pp 721–728
11. Zhang Q, Reeves DS (2007) MetaAware: identifying metamorphic malware. In: Proceedings of the 23rd annual computer security applications conference, pp 411–420
12. Karnik A, Goswami S, RGuha R (2007) Detecting obfuscated viruses using cosine similarity analysis. In: Proceedings of the 1th Asia international conference on modelling and simulation, pp 165–170
13. Cha SK, Moraru I, Jang J, Truelove J, Brumley D, Andersen DG (2010) SplitScreen: enabling efficient, distributed malware detection. In: Proceedings of the 7th USENIX conference on networked systems design and implementation
14. ClamAV, Available at http://www.clamav.net/
15. VX Heavens, Available at http://vx.netlux.org/
16. Han KS, Kim IK, Im EG (2011) Malware family classification method using API sequential characteristic. J JSE 8(2):319–335