Hindawi Publishing Corporation
International Journal of Distributed Sensor Networks
Volume 2015, Article ID 659101, 9 pages
http://dx.doi.org/10.1155/2015/659101

Research Article

A Novel Approach to Detect Malware Based on API Call Sequence Analysis

Youngjoon Ki,[1] Eunjin Kim,[2] and Huy Kang Kim[1]

[1]*Korea University, 145 Anam-ro, Seongbuk-gu, Seoul 137-713, Republic of Korea*
[2]*Kyonggi University, Gwanggyosan-ro, Yeongtong-gu, Suwon 443-760, Republic of Korea*

Correspondence should be addressed to Huy Kang Kim; cenda@korea.ac.kr

Received 7 November 2014; Revised 23 February 2015; Accepted 4 April 2015

Academic Editor: Praveen Rao

In the era of ubiquitous sensors and smart devices, detecting malware is becoming an endless battle between ever-evolving malware and antivirus programs that need to process ever-increasing security related data. For malware detection, various approaches have been proposed. Among them, dynamic analysis is known to be effective in terms of providing behavioral information. As malware authors increasingly use obfuscation techniques, it becomes more important to monitor how malware behaves for its detection. In this paper, we propose a novel approach for dynamic analysis of malware. We adopt DNA sequence alignment algorithms and extract common API call sequence patterns of malicious function from malware in different categories. We find that certain malicious functions are commonly included in malware even in different categories. From checking the existence of certain functions or API call sequence patterns matched, we can even detect new unknown malware. The result of our experiment shows high enough *F*-measure and accuracy. API call sequence can be extracted from most of the modern devices; therefore, we believe that our method can detect the malware for all types of the ubiquitous devices.

1. Introduction

Nowadays, power-saving techniques and enhanced computing power allow us to use sensors as multifunctional devices. They can perform complex tasks and are connected to the Internet all the time. They are now playing a major role in sensing and generating big data in IoT (Internet of Things) era. Besides, the increased use of the mobile smart devices also allows us powerful ubiquitous computing. Ubiquitous sensors and smart devices are becoming critical source of data. However, as their operating system and software are almost same to the traditional Windows based system or UNIX based system, traditional malware and exploit can work on these small smart sensors and devices, the use of which is exponentially increased. Therefore, it becomes critical to manage ever-evolving malware and related security risks in the era of ubiquitous sensors and smart devices.

Various approaches have been proposed for malware detection [1–3]. Detection techniques proposed earlier were based on static analysis. Static analysis examines the binary code, analyzes all possible execution paths, and identifies malicious code without execution [4]. However, analyzing binary code turns out to be difficult nowadays. As obfuscation techniques become more sophisticated, static analysis can be bypassed by various obfuscation techniques, such as polymorphism, encryption, or packing [4]. In addition, as static analysis relies on a prebuilt signature database, it cannot easily detect new unknown malware until the signature is updated [4, 5]. Besides, some execution paths can be only explored after execution [4, 5]. To overcome these limitations of static analysis and complement it, dynamic analysis has been proposed and is widely used to achieve more effective malware detection.

Techniques based on dynamic analysis execute malware and trace its behaviors. Two major approaches in dynamic analysis are control flow analysis and API call analysis. Both approaches detect malware based on analysis of similarity between the behavior of the new and the known ones. However, malware authors try to circumvent those techniques through inserting meaningless codes or shuffling the sequence of programs.

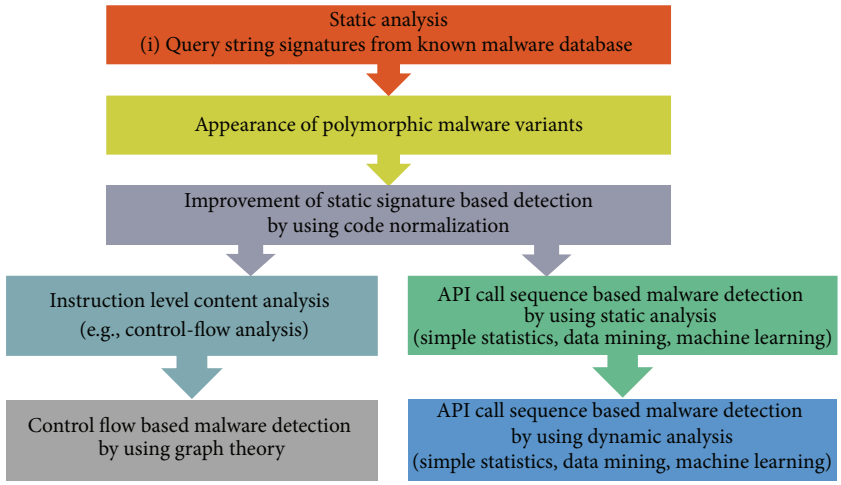


Figure 1: Chronicles of advances in malware analysis.

Many of currently available API call analysis techniques fail to detect malware of such circumvention. Some techniques focus on extracting APIs that are frequently observed in malware in each class [6, 7]. They monitor APIs that are called and calculate the frequency and total number of events that certain API function called. Even though they quickly reveal the characteristics of malware in the same class, they fail to show the sequence of malware behavior and can be easily evaded by malware authors' inserting and executing dummy and redundant API calls.

Others extract API call sequence for each class and develop static signature based on it [8–11]. They are better from the semantic view because they monitor the sequence of calls and the flow of programs. However, simply creating signatures from the extracting frequently found that call sequence for malware in each class does not allow them to detect malware in polymorphic or unknown form. It can be also evaded by malware authors' evading tricks such as inserting redundant API calls. This incurs the need for new approaches in API call sequence analysis.

Recent few studies focus on the fact that unless the main purpose or functions of the malware are not changed, the critical low-level system call sequence does not change. Therefore, instead of extracting API call sequence for malware in each class, they propose to focus on API call sequence for certain functions of malware [12–14]. However, such approach has not been empirically well studied comparing to the API call sequence analysis techniques proposed previously. In this study, with a large set of data, we empirically study whether such approach generates superior results comparing with the previous ones.

In this study, we adopt sequence alignment algorithm which is known to perform well in extracting the similar subsequences from the different sequences. Sequence alignment algorithm will make us less confused by meaningless codes inserted in malware in its detection. Sequence alignment algorithms have been applied in various areas such as natural language processing and biometrics and have proven their excellence [15]. In this paper, we propose a new approach in API call sequence analysis with introducing sequence alignment algorithm. The rest of the paper is organized as follows. In Section 2, we review the related literature. In Section 3, we present our methodology and experiment. In Section 4, we conclude our research and suggest future research direction.

2. Literature Review

2.1. Malware Analysis. Malware analysis has made its advances as shown in Figure 1. Signature based detection was proposed in its early stage. In this stage, automatic generation of malware's signatures as much as possible was assumed to be important and this increased pattern matching speed [16, 17].

However, the signature based detection method shows following weaknesses. It requires continuous updates of signature and high maintenance cost. In addition, such method could be easily evaded by malware in polymorphic form [5]. To overcome the weakness, it embraces code normalization to capture the canonized original maliciousness [18]. Through this, capturing malicious codes that vary by polymorphic techniques applied becomes possible. However, it is still weak in detecting obfuscated malware. Besides, some execution paths could be only explored after execution [4, 5].

Malware analysis technique kept its advance due to certain needs; hence, dynamic analysis was proposed. Dynamic analysis methods are known to perform well for obfuscated malware [3]. Dynamic analysis executes malware, monitors how it behaves, and detects unknown malware that shows similar behavior to the known ones [3]. Two major dynamic analysis methods that are well known are control flow analysis and API call analysis [19, 20].

API call information shows how malware behaves. API call information can be extracted by both static and dynamic approaches. With static approach [6, 10, 11, 21], API list can be extracted from PE format of the executable files. With dynamic approach [7–9, 22–24], the APIs that are called can

Table 1: Comparison of previous researches.

Reference	Method	# of malware	F-measure	Accuracy	Used feature
Alazab et al., 2011 [21]	Static analysis	66,703	0.984 (for detection)	0.985	Frequency of API usage
Sathyanarayan et al., 2008 [6]	Static analysis	800	0.909 (for detection)	0.841	API call sequence
Tian et al., 2010 [7]	Dynamic analysis	1,368	0.969 (for detection)	0.973	Frequency of API usage
Sami et al., 2010 [10]	Static analysis	32,000	0.878 (for detection)	0.983	Frequency of API usage
Ye et al., 2007 [11]	Static analysis	17,366	0.941 (for detection)	0.930	API call sequence
Ahmed et al., 2009 [22]	Dynamic analysis	416	— (not mentioned)	0.98	API call sequence
Rieck et al., 2011 [26]	Dynamic analysis	3,133	0.950 (for clustering)	—	API call sequence
Qiao et al., 2014 [23]	Dynamic analysis	3,131	0.909 (for clustering)	—	API call sequence
Qiao et al., 2013 [24]	Dynamic analysis	3,131	0.947 (for clustering)	—	API call sequence
Our method	Dynamic analysis	23,080	0.999 (for detection)	0.998	API call sequence

be observed by running the executable files (usually run on virtual machine).

There are two major ways to analyze the API call information gathered through the static approach. The first one applies simple statistical analysis, for example, counting the frequency of the called APIs, which can be used as a feature for classifying malware [6]. The second approach applies data mining or machine learning techniques to the collected API call information [21].

On the other hand, API call sequence information collected through the dynamic approach can be used for creating behavioral patterns. The information gathered through the dynamic approach can also be processed using simple statistics such as frequency counting [7] and data mining or machine learning [8, 22, 24].

Besides, researchers find other ways to process API call sequence information. Some previous studies applied API call graph [13]. There are many variations of call graph analysis. In [25], researchers adopted social network analysis methods to find meaningful features for call graph analysis. In [9], researchers calculate the similarity between API call sequences based on cosine similarity function and extended Jaccard measure. Recent works [19, 20, 26, 27] use additional information such as control flow information and API argument information to increase the accuracy in the mining process.

In Table 1, we summarize the previous empirical studies on API call analysis and their overall performances. We also compare them with our approach. In this study, we adopted dynamic method to extract the API call sequences. To get patterns that are rigor, we applied DNA sequence alignment algorithms (MSA and LCS). By using both extracted API call sequence patterns and critical API call sequences, we can detect the unknown malware or variants with high accuracy.

2.2. DNA Sequence Alignment. Sequence alignment algorithms are most widely used in the area of biometrics to calculate the similarity between two or more DNA sequences or to find certain DNA subsequences from full DNA. Sequence alignment algorithms fall into two categories, which are global and local alignment [15]. Global alignment aligns entire sequence. It is useful when we attempt to find the sequence of most similar length and strings. The well-known global alignment algorithm is the Needleman-Wunsch algorithm [28]. Local alignment finds the highly similar subsequences between given sequences [15]. The Smith-Waterman algorithm is the well-known local alignment algorithm [29].

Pairwise sequence alignment methods find global alignments or the locally similar subsequences of two sequences. Multiple sequence alignment algorithm extends pairwise alignment to handle multiple sequences at the same time [15]. Since we need to handle multiple API call sequences, we applied multiple sequence alignment algorithm in this study.

3. Methodology

3.1. Environment Setup for Dynamic Analysis. For our experiment, we set up a virtual environment to run malicious programs. To trace API call sequence in runtime, we hook the program. In this study, we used the Detours hooking library [30] supported by Microsoft to trace API call sequences from Windows executable programs.

The Detours hooking process proceeds as shown in Figure 2 [30]. Unlike the original process without Detours, the hooking process goes through the Detour function and the Trampoline function. Before the target function starts, the Detour function leaves the log of the target function's name. After the Detour function ends, the Trampoline function that saves the start address of the target function begins. After

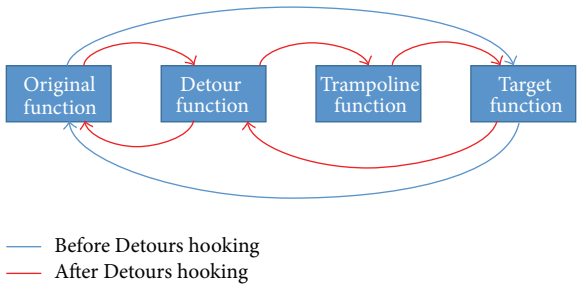


Figure 2: Hooking process of Detours.

the target function finishes, the Detours function also can check the end of target function. This allows us to trace API call sequence.

For experiment, we set up a virtual machine environment to run and monitor any suspicious executable programs' behavior. VirtualBox [31] was used to execute malware and observe its activity. We adopted 32-bit Windows XP Service Pack 3 as the virtual machine's operating system because most malware easily runs on a Windows XP environment. In addition, we set the maximum monitoring period as two minutes for the default value to trace each API call sequence. This monitoring period is configurable.

For DNA sequence alignment, we used the ClustalX [32]. ClustalX is widely used freeware in genome sequence analysis, such as DNA, RNA, or protein sequences. The program supports multiple sequence alignment (MSA) [33] and also provides visualization.

Figure 3 shows the result of sequence alignment by ClustalX.

Figure 3(a) presents the API call sequences found in some malware samples in the same class and Figure 3(b) shows the result after applying MSA. This shows that, for those given malware samples, malware in the same family shares much common call subsequences. In Figure 3(b), each horizontal line represents an API call sequence of each malware, and the vertical line shows the common API call subsequence among malware in the same class. The API subsequence is colored according to the related functionality.

3.2. Dataset Preparation. To create a dataset, we chose 23,080 malware samples randomly from the malware dataset of the Malicia-project [34] and VirusTotal [35].

We share our dataset used in this paper online. Table 2 shows simple statistics about our dataset. The whole dataset (including malware, benign software, and their call sequence list) is accessible from the URL (http://ocslab.hksecurity.net/apimds-dataset).

3.3. Limitation of Using Antivirus Program's Labeling Information. As we mentioned earlier in Sections 1 and 2, many previous studies on API call analysis classified malware based on the antivirus program vendors' labeling. However, such labeling can lead to an error as shown in Figure 4.

Figure 4 shows that malware in the same label group can vary in terms of the API call sequence. Malware listed in Figure 4 is labeled as Trojan-FakeAV.Win32.Security Shield.

Table 2: Dataset description.

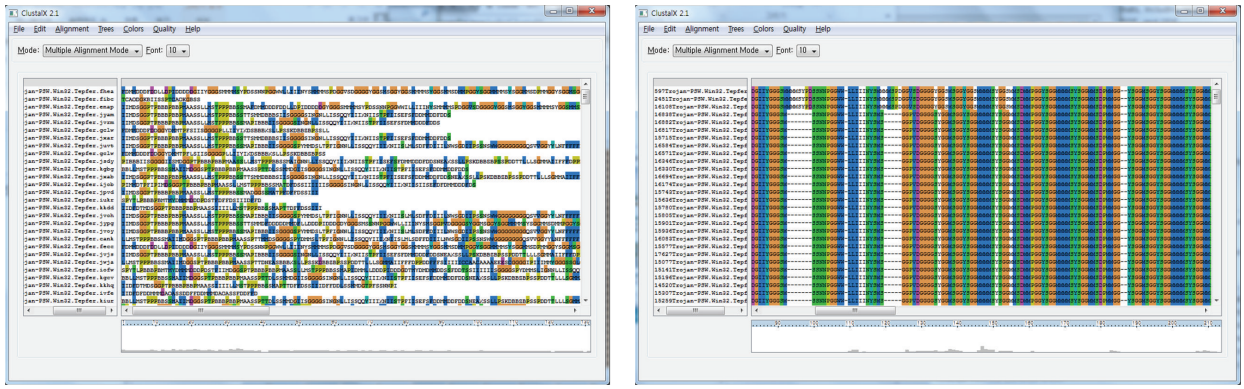
Category	Subcategory	Ratio (%)
Backdoor		3.37
Worm	Worm	3.32
	Email-Worm	0.55
	Net-Worm	0.79
	P2P-Worm	0.3
Packed		5.57
PUP	Adware	13.63
	Downloader	2.94
	WebToolbar	1.22
Trojan	Trojan (Generic)	29.3
	Trojan-Banker	0.14
	Trojan-Clicker	0.12
	Trojan-Downloader	2.29
	Trojan-Dropper	1.91
	Trojan-FakeAV	18.8
	Trojan-GameThief	0.63
	Trojan-PSW	3.79
	Trojan-Ransom	2.58
	Trojan-Spy	3.12
Misc.		5.52

For ease of explanation, we divide malware into groups (a), (b), and (c). While malware in groups (a) and (b) shows a similar API call sequence pattern within the group, we can only observe partial similarity between groups (a) and (b). Malware in group (c) shows a large difference in the API call sequence within the group. Group (c) also differs significantly from groups (a) and (b) in the API call sequence pattern.

As described above, malware in the same class can have quite different call sequences. On the other hand, malware in different classes can have common call sequences. For example, in our dataset, the three malware classes, "HEUR:Trojan. Win32.Generic," "Trojan.Win32. FakeAV.lhiz," and "Packed.Win32. Katusha.o," show high similarity among them.

3.4. Categorization of APIs. In our dataset, 2,727 kinds of API were found; we categorized them into 26 groups according to MSDN library [36]. Table 3 shows an example of API classification. For example, CallNextHook API and SetWindowsHook API are categorized as class A, the hooking functions. Likewise, DeviceIoControl API and DvdLauncher API are categorized as class Z, the device control. If there exists an API call sequence such as [CallNextHook, DeleteFile, DeviceIoControl], the categorized API call sequence becomes [A, B, Z].

3.5. Extraction of Longest Common Subsequence. To extract the common API call sequence pattern among malware, the longest common subsequences (LCSs) [37] were used. The formula is shown in (1). In the formula, X_i and Y_i represent



(a) (b)

Figure 3: Visualization of API call sequences of Trojan.PSW.Win32.Tepfer. (a) Visualization of API call sequence before applying multiple sequence alignment. (b) Visualization of API call sequences after applying multiple sequence alignment.

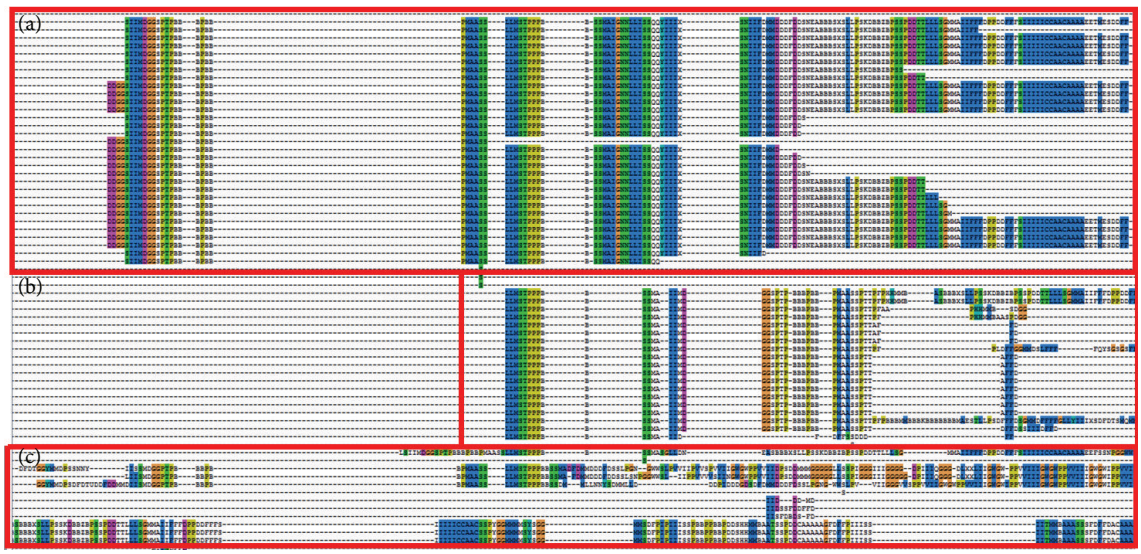


Figure 4: An example of MSA result of Trojan-FakeAV.Win32.Security Shield.

the ith character of sequences X and Y, respectively. For example, the LCSs of ABCD and ACB are AB and AC:

$$\mathrm{LCS}\left(X_i, Y_j\right) = \begin{cases} \emptyset & \text{if } i = 0, \\ & \text{or } j = 0, \\ \mathrm{LCS}\left(X_{i-1}, Y_{j-1}\right) + \text{common character} & \text{if } x_i = y_j, \\ \text{longest}\left(\mathrm{LCS}\left(X_i, Y_{j-1}\right), \mathrm{LCS}\left(X_{i-1}, Y_j\right)\right) & \text{if } x_i \neq y_j. \end{cases} \quad (1)$$

Since the LCS shows the longest malware API call sequence pattern, it can be treated as a malware's signature.

The LCS of malware can contain the same API call sequences that exist in benign programs. This occurs when malware uses generic APIs commonly used in any programs (e.g., API calls used in GUI). To reduce this kind of false positive error, we excluded these typical LCSs from our signature database.

3.6. Critical API Call Sequences of Known Malicious Activities. To improve the usefulness of API call sequences as features for detecting malware more accurately, we employed an additional feature. We assumed that there exists a specific API call sequence pattern that matches specific malicious activity. If such an API call sequence pattern exists, we can detect unknown attacks by checking whether a critical API call sequence pattern exists. To verify the existence of a critical API call sequence pattern of malware, we created profiles of the API call sequences of known attacks, which can be further used for unknown attack detection.

Examples of critical API call sequence patterns found in our sample malware are shown in Table 4. It should be noted that round brackets represent the OR relation. For example, in the case of IAT hooking, one possible critical API call sequence pattern is [LoadLibrary, strcmp, VirtualProtect].

Table 3: Description, example, and the number of APIs by API class.

Class	Description	Example	Number of APIs
Class A	Hooking functions	CallNextHookEx, SetWindowsHookEx	22
Class B	Files and directories	DeleteFile, CopyFile, CreateDirectory	360
Class C	Registry modification	RegCreateKey, RegDeleteValue	77
Class D	Synchronization	CreateMutex, CreateMutexEx	225
Class E	Memory allocation	HeapAlloc, GlobalMemoryStatus	126
⋮			⋮
Class Z	Device management	DeviceIoControl, DvdLauncher	37
Total			2,727

Table 4: Malicious activities and their critical API call sequence pattern.

Malicious activity	Critical API call sequence
DLL injection using CreateRemoteThread	OpenProcess, VirtualAllocEx, WriteProcessMemory, CreateRemoteThread
IAT hooking	LoadLibrary, (strcmp, strncmp, _stricmp, _strnicmp), VirtualProtect
Antidebugging	(IsDebuggerPresent, CheckRemoteDebuggerPresent, OutputDebugStringA, OutputDebugStringW)
Screen capture	(GetDC, GetWindowDC), CreateCompatibleDC, CreateCompatibleBitmap, SelectObject, BitBlt, WriteFile

In this case, strcmp can be replaced with strncmp and it becomes another possible critical API call sequence pattern [LoadLibrary, strncmp, VirtualProtect].

The following descriptions explain how certain malicious activities are carried out using the critical API call sequence pattern found.

(i) DLL injection: various methods for injecting Dynamic Link Library (DLL) into a target process exist, the most popular of which is to use CreateRemoteThread API, introduced by Jeffery Ritcher in the 1990s [38]. First, the method gets a handle of the target process by using OpenProcess. Then, it allocates some space in the target process' memory using VirtualAllocateEx and writes the DLL's name, including full path, to the allocated memory by using WriteProcessMemory. Finally, it makes the target process reload the DLL using CreateRemoteThread.

(ii) IAT hooking: the Import Address Table (IAT) contains the API's start address. By modifying the address, a different API can be called although the legitimate API is called. The process is as follows. First, it loads the target library using LoadLibrary. After finding the DLL from the IAT by comparing related APIs (e.g., strcmp), it modifies an attribute of the memory to be writable by using VirtualProtect and changes the address of the DLL.

(iii) Antidebugging: there are many methods of detecting a debugging process using the API. For example, if the return value of the IsDebuggerPresent API is 1, this means that the program is being debugged. Antidebugging itself may not be a malicious activity, but it is often observed in malware.

Table 5: Scanning result of malicious behavior.

Malicious behavior	Number of detections for malware	Number of detections for benign programs
DLL injection	23	0
IAT hooking	5,481	0
Antidebugging	16,385	51
Screen capture	14	0

(iv) Screen capture: backdoor often captures the screen and saves it as an image file. The process is as follows. First, it gets a handle of the window using an API, such as GetDC. Then, it creates a compatible space for saving the image using CreateCompatibleDC and CreateCompatibleBitmap. After choosing the image's pointer using SelectObject, it copies the image to the memory space using BitBlt. Finally, it writes the captured image as a file using WriteFile.

We checked whether such critical API call sequence patterns and related malicious activities found in malware distinguish malware from benign programs. As shown in Table 5, we can observe that DLL injection, IAT hooking, and screen capture activities and their related API call sequences are found only in malware. However, antidebugging activity was found both in malware and benign programs. This activity was detected in malware much more frequently. This result shows the existence of unique behaviors of malware and their related critical API call sequence. We believe that this behavior-based malware classification can be effective in the dynamic analysis of API call sequences.

3.7. Overall Process. To summarize, our overall process is described in Figure 5. In the creating signature process,

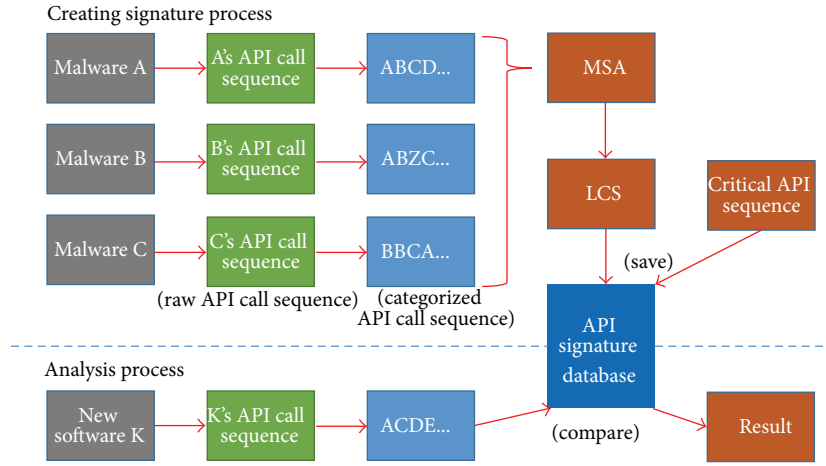


Figure 5: Overall process of our methodology.

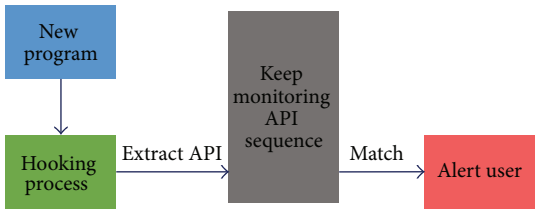


Figure 6: Overview of the proposed malware detection system, APIMDS.

the API call sequence of malware is extracted and stored in the signature database. MSA is applied to extract the LCS. The preanalyzed critical API call sequence is also stored in the signature database.

In the analysis process, the extracted API call sequence of a newly inserted program is compared with the API signatures in the database. If any LCS is matched with the signatures, the newly inserted program is classified as malware. In addition, when the newly inserted program has a matched critical API call sequence, the detected functionalities are reported as well.

3.8. Prototype Implementation. We implemented our malware detection system, API-based malware detection system (APIMDS) based on API call sequence analysis. As can be seen in Figure 6, when a new program needs to be traced, the hooking process monitors and tracks the program's API call sequences. After extracting the API call sequences from the program, the system compares them with our API call sequence database of APIMDS. If matched, APIMDS alerts the security administrator.

3.9. Accuracy Test. We used 70% of the malware and benign programs to train the process and tested the accuracy using the remaining 30%. The test results are shown in Table 6. There is no false positive because the benign LCS sequences were excluded, as explained in Section 3.5.

Table 6: Confusion matrix of classification result.

Category	Predicted class	
	Malware	Benign
Actual class		
Malware	6,910 (TP)	8 (FN)
Benign	0 (FP)	26 (TN)

FPR (False Positive Ratio) = FP/(FP + TN) = 0.
FNR (False Negative Ratio) = FN/(FN + TP) = 0.0011.
Recall = TP/(TP + FN) = 0.998.
Precision = TP/(TP + FP) = 1.
$F1$ score = 2TP/(2TP + FP + FN) = 0.999 (where TP is true positive, TN is true negative, FP is false positive, and FN is false negative).

To summarize, the precision is 1, and recall is 0.998. The $F1$ score is 0.999. This implies that our proposed malware detection method is highly reliable.

3.10. Limitations and Future Works. DNA sequence alignment algorithms consume much resources and time. MSA is known as NP-Complete, and computing LCS is known as NP-hard problem. Therefore, the computing cost of MSA and LCS is not negligible. However, most malicious programs' size is relatively smaller than benign programs. In our experiment, computing MSA of 1,000 sequences took around 30 seconds under Intel i5 core and 8 GB memory PC environment.

In addition, to overcome such complexity issue, we adopted blacklist and whitelist based filtering, which excludes well-known benign or malicious programs. In order to reduce the computing cost, we recommend updating well-known benign programs or malicious programs list continuously from trustable sites. For example, this software list can be retrieved from the National Software Research Library [39].

Recent malware can hide their activities by using logic bomb or checking the defender's presence (e.g., checking the debugger process). Furthermore, they tend to adopt the obfuscation techniques. Therefore, detecting malware's hiding functionalities becomes challenging. Although several works try to detect logic bomb [40, 41] by using the symbolic-execution method, this method is not cheap. Our system can

detect the antidebugging functionality that includes the API calls such as Sleep, GetLocalTime, and IsDebuggerPresent. However, there are other patterns to hide malware's activities. We need to enhance our system to better detect antidebugging functionality in the future.

The hooking process used in this study can only trace user-level APIs, and, therefore, the API call sequences cannot be logged if malware uses kernel-level APIs. In future work, we will consider malware using kernel-level APIs. We expect that this will improve the accuracy of the proposed system.

In the future, with the proliferation of mobile platform and the introduction of IoT (Internet of Things), we need to adjust our system for the use in those eras.

Our prototype is not well elaborated for practitioners. Hence, further study or elaboration is needed in the future.

4. Conclusion

In this paper, we proposed a novel method of API call sequence analysis. We found that antivirus vendors' labeling of malware could be less accurate to be applied in the dynamic analysis of API call sequences. Therefore, instead of extracting API call sequence for each class, we extract API call sequence patterns from malware in different categories, focusing on common malware functions. Then, we developed a signature database and the proposed APIMDS for detecting malware based on the signature. Experimentally, our system showed promising detection results with high accuracy and extremely low error rate.

Many malware detection systems rely on the signature of a malware's static information, such as file size, process, and its artifacts. Therefore, they fail to detect new unknown malware until the signature has been updated. In contrast to the signature of the malware's static information, our signature database of dynamic information of critical API call sequence patterns allows us to generalize malware behavior and facilitates the effective detection of new unknown malware. Our method can be applied to both of the traditional PCs and new smart devices if the devices can extract and send the extracted API call sequence information to the analysis module outside. Because API call sequence is well abstracted behavioral data that can be extracted from most of the device, therefore, our method can detect the attack by analyzing the big data that are extracted from the ubiquitous devices such as sensors, smart devices, PCs, and servers.

We believe that our proposed system can be applied in a new type of cyber security intelligence system that is required to investigate ever-evolving malware.

Conflict of Interests

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

Acknowledgments

This research was supported by the MSIP (Ministry of Science, ICT and Future Planning), Korea, under the ITRC (Information Technology Research Center) support program (NIPA-2014-H0301-14-1004) supervised by the NIPA (National IT Industry Promotion Agency). This work was also supported by the ICT R&D Program of MSIP/IITP [14-912-06-002, The Development of Script-Based Cyber Attack Protection Technology].

References

[1] N. Idika and A. P. Mathur, *A survey of malware detection techniques [Predoctoral Fellowship, and Purdue Doctoral Fellowship]*, Purdue University, 2007.

[2] P. Vinod, R. Jaipur, V. Laxmi, and M. S. Gaur, "Survey on malware detection methods," in *Proceedings of the 3rd Hackers' Workshop on Computer and Internet Security (IITKHACK '09)*, 2009.

[3] S. Cesare and Y. Xiang, *Software Similarity and Classification*, Springer Science & Business Media, 2012.

[4] P. Okane, S. Sezer, and K. McLaughlin, "Obfuscation: the hidden malware," *IEEE Security & Privacy*, vol. 9, no. 5, pp. 41–47, 2011.

[5] A. Moser, C. Kruegel, and E. Kirda, "Limits of static analysis for malware detection," in *Proceedings of the 23rd Annual Computer Security Applications Conference (ACSAC '07)*, pp. 421–430, December 2007.

[6] V. S. Sathyanarayan, P. Kohli, and B. Bruhadeshwar, "Signature generation and detection of malware families," in *Information Security and Privacy*, Springer, Berlin, Germany, 2008.

[7] R. Tian, M. R. Islam, L. Batten, and S. Versteeg, "Differentiating malware from cleanware using behavioural analysis," in *Proceedings of the 5th International Conference on Malicious and Unwanted Software (MALWARE '10)*, pp. 23–30, Nancy, France, October 2010.

[8] M. Shankarapani, K. Kancherla, S. Ramammoorthy, R. Movva, and S. Mukkamala, "Kernel machines for malware classification and similarity analysis," in *Proceedings of the International Joint Conference on Neural Networks (IJCNN '10)*, pp. 1–6, July 2010.

[9] M. K. Shankarapani, S. Ramamoorthy, R. S. Movva, and S. Mukkamala, "Malware detection using assembly and API call sequences," *Journal in Computer Virology*, vol. 7, no. 2, pp. 107–119, 2011.

[10] A. Sami, B. Yadegari, H. Rahimi, N. Peiravian, S. Hashemi, and A. Hamze, "Malware detection based on mining API calls," in *Proceedings of the 25th Annual ACM Symposium on Applied Computing (SAC '10)*, pp. 1020–1025, ACM, March 2010.

[11] Y. Ye, D. Wang, T. Li, and D. Ye, "IMDS: intelligent malware detection system," in *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 1043–1047, ACM, August 2007.

[12] S. Peisert, M. Bishop, S. Karin, and K. Marzullo, "Analysis of computer intrusions using sequences of function calls," *IEEE Transactions on Dependable and Secure Computing*, vol. 4, no. 2, pp. 137–150, 2007.

[13] J. Bergeron, M. Debbabi, J. Desharnais, M. M. Erhioui, Y. Lavoie, and N. Tawbi, "Static detection of malicious code in executable programs," in *Proceedings of the Symposium on Requirements Engineering for Information Security (SREIS '01)*, 2001.

[14] H.-M. Sun, Y.-H. Lin, and M.-F. Wu, "API monitoring system for defeating worms and exploits in MS-Windows system," in *Information Security and Privacy*, vol. 4058 of *Lecture Notes in Computer Science*, pp. 159–170, Springer, Berlin, Germany, 2006.

[15] Sequence Alignment, http://en.wikipedia.org/wiki/Sequence_alignment.

[16] K. Griffin, S. Schneider, X. Hu, and T.-C. Chiueh, "Automatic generation of string signatures for malware detection," in *Recent Advances in Intrusion Detection*, vol. 5758 of *Lecture Notes in Computer Science*, pp. 101–120, Springer, Berlin, Germany, 2009.

[17] J. O. Kephart and W. C. Arnold, "Automatic extraction of computer virus signatures," in *Proceedings of the 4th Virus Bulletin International Conference*, Abingdon, UK, 1994.

[18] M. Christodorescu, J. Kinder, S. Jha, S. Katzenbeisser, and H. Veith, "Malware normalization," EPFL-REPORT 167534, University of Wisconsin, Madison, Wis, USA, 2005.

[19] M. Rajagopalan, M. A. Hiltunen, T. Jim, and R. D. Schlichting, "System call monitoring using authenticated system calls," *IEEE Transactions on Dependable and Secure Computing*, vol. 3, no. 3, pp. 216–229, 2006.

[20] M. Abadi, M. Budiu, Ú. Erlingsson, and J. Ligatti, "Control-flow integrity," in *Proceedings of the 12th ACM Conference on Computer and Communications Security*, pp. 340–353, November 2005.

[21] M. Alazab, S. Venkatraman, and P. Watters, "Zero-day malware detection based on supervised learning algorithms of API call signatures," in *Proceedings of the 9th Australasian Data Mining Conference (AusDM '11)*, vol. 121, pp. 171–182, Australian Computer Society, December 2011.

[22] F. Ahmed, H. Hameed, M. Z. Shafiq, and M. Farooq, "Using spatio-temporal information in API calls with machine learning algorithms for malware detection," in *Proceedings of the 2nd ACM Workshop on Security and Artificial Intelligence*, pp. 55–62, November 2009.

[23] Y. Qiao, Y. Yang, J. He, C. Tang, and Z. Liu, "CBM: free, automatic malware analysis framework using API call sequences," in *Knowledge Engineering and Management*, pp. 225–236, Springer, Berlin, Germany, 2014.

[24] Y. Qiao, Y. Yang, L. Ji, and J. He, "Analyzing malware by abstracting the frequent itemsets in API call sequences," in *Proceedings of the 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom '13)*, pp. 265–270, July 2013.

[25] J.-W. Jang, J. Woo, J. Yun, and H. K. Kim, "Mal-netminer: malware classification based on social network analysis of call graph," in *Proceedings of the Companion Publication of the 23rd International Conference on World Wide Web Companion (WWW Companion '14)*, pp. 731–734, International World Wide Web Conferences Steering Committee, 2014.

[26] K. Rieck, P. Trinius, C. Willems, and T. Holz, "Automatic analysis of malware behavior using machine learning," *Journal of Computer Security*, vol. 19, no. 4, pp. 639–668, 2011.

[27] C. M. Linn, M. Rajagopalan, S. Baker, C. Collberg, S. K. Debray, and J. H. Hartman, "Protecting against unexpected system calls," in *Proceedings of the 14th USENIX Security Symposium*, pp. 239–254, Baltimore, Md, USA, August 2005.

[28] S. B. Needleman and C. D. Wunsch, "A general method applicable to the search for similarities in the amino acid sequence of two proteins," *Journal of Molecular Biology*, vol. 48, no. 3, pp. 443–453, 1970.

[29] T. F. Smith and M. S. Waterman, "Identification of common molecular subsequences," *Journal of Molecular Biology*, vol. 147, no. 1, pp. 195–197, 1981.

[30] G. Hunt and D. Brubacher, "Detours: binary interception of Win32 functions," in *Proceedings of the 3rd Conference on USENIX Windows NT Symposium (WINSYM '99)*, 1999.

[31] VirtualBox, https://www.virtualbox.org.

[32] ClustalX, http://www.clustal.org/.

[33] D. G. Higgins and P. M. Sharp, "CLUSTAL: a package for performing multiple sequence alignment on a microcomputer," *Gene*, vol. 73, no. 1, pp. 237–244, 1988.

[34] Malicia Project, http://malicia-project.com/dataset.html.

[35] VirusTotal, https://www.virustotal.com.

[36] MSDN Library, http://msdn.microsoft.com/en-us/library/.

[37] Longest Common Subsequence Problem, http://en.wikipedia.org/wiki/Longest_common_subsequence_problem.

[38] J. Richter, *Programming Applications for Microsoft Windows*, vol. 22, Microsoft Press, Washington, DC, USA, 1999.

[39] National Software Research Library, http://www.nsrl.nist.gov/.

[40] D. Brumley, C. Hartwig, Z. Liang, J. Newsome, D. Song, and H. Yin, "Automatically identifying trigger-based behavior in malware," in *Botnet Detection*, vol. 36 of *Advances in Information Security*, pp. 65–88, Springer, New York, NY, USA, 2008.

[41] H. Yin and D. Song, *Automatic Malware Analysis: An Emulator Based Approach*, Springer Science+Business Media LLC, 2012.