# Malware Similarity Analysis using API Sequence Alignments

In Kyeom Cho[1], TaeGuen Kim[1], Yu Jin Shim[1], Haeryong Park[2], Bomin Choi[2], and Eul Gyu Im[1]*
[1]Hanyang University, Seoul, Korea
{dlsrua1004, cloudio17, luvtdw, imeg}@hanyang.ac.kr
[2]Korea Internet & Security Agency, Seoul, Korea
{hrpark, bmchoi}@kisa.or.kr

**Abstract**

Malware variants could be defined as malware that have similar malcious behavior. In this paper, a sequence alignment method, the method widely used in Bioinformatics, was used to detect malware variants. This method can find the common parts of Malware's API call sequences, and these common API call sequences can be used to detect similar behaviors of malware variants. However, when a sequence alignment method is applied to compare the API call sequences, the performance depends on lengths of API call sequences and if the lengths are too long, the performance would be very poor. Therefore, in this paper, we devised a malware similarity calculation system to detect malware variants and suggested an improved process which can reduce sequence alignment overheads. Finally, our proposed system was tested with two given malware families and it can be used to verify whether the given malware variants have similar behaviors. Experimental results show that our method can be leveraged in the malware detection system.

**Keywords**: malware analysis, dynamic analysis, API sequence, sequence alignment

## 1 Introduction

Since the first malware emerged in the Internet, the number of malware has increased rapidly[7, 14, 11]. One of important factors that increase the number of malware is structural transformation of the malicious code that already exists. Most of malware writers develop zero-day malware for specific malicious purpose and they transform malicious modules to avoid the malware detection systems, such as anti-virus solutions. Therefore, malware variants have different structural static features like mnemonic frequencies, control flow graphs and so on. This limitation implies that static analysis on malware cannot handle newly generated malware variants properly, because most of static analysis systems use signatures of already exsiting malware[16, 18, 10].

To overcome this kind of problems, we used dynamic analysis to classify the malware variants which have similar behavior contexts[9, 8]. Although malware variants are different in static features but they have common behaviors like file related activities, registry related activities, and so on. For this purpose, our proposed malware similarity calculation system uses behavioral features to verify relations of two malicious codes. API call sequences are used to represent the behavioral features of malware. The reason why we choose API call sequences to calculate the similarity of two malware is that APIs are basic interfaces to do some malicious activities in operating systems. A similarity calculation algorithm that we use to capture the similar behaviors of two malicious code is a sequence alignment algorithm which is widely used in the bioinformatics field. As simple sequence comparison methods like n-gram based methods are not suitable to get accurate similarity results of API call sequences which are slightly

*Corresponding author: Department of of Computer Engineering, Hanyang University, 720, IT/BT Building, Hanyang Univ., Haengdang 1-dong, Seongdong-gu, Seoul, Korea, Tel: +82-(0)2-2221-2381, Web: `http://usecurity.hanyang.ac.kr` noises, we utilized a sequence alignment algorithm. The sequence alignment algorithm provides an alignment solution to match the maximum common sequence of two target sequences. Therefore, our proposed method can handle sequences which have slightly different with noises by aligning the positions of components in sequences. Although the sequence alignment algorithm is useful, its time complexity is high. According to our experiments, the number of components of API call sequences exceeds to ten millions in many cases and the long sequences cause high computational overheads to be processed using a sequence alignment algorithm. To handle this problem, we devised a method to remove repeated subsequences in the whole API call sequences.

Our proposed similarity calculation system consists of four components. Each component is performed sequentially. The first component is the behavior monitor module which monitors the API call sequences of two malicious binary files. The second component is the sequence refining module which reduces the size of API call sequences. The third component is the sequence alignment module which generates optimal alignment results which make the two aligned sequences have as many common subsequences as possible. The fourth component is the final similarity calculation module. The module computes the similarity based on subsequences that are matched in the two aligned API call sequences.

The rest of the paper is organized as follows: Section 2 discusses related work. Section 3 presents sequence alignment as background, and describes the problem when it is applied in malware detection. Section 4 presents the proposed similarity calculation system to detect malware variants and describes modules which organized our framework in detail. Section 5 shows the experimental results which show possibilities of malware variant detection with variant samples and the other evaluation results. Section 6 summarizes our research and provides the contributions, limitations, and future work of this ongoing research.

## 2 Reltaed Works

There are some researches which are related with our works. Most of previous researches use API features to define the malicious behaviors and devised their own comparison methods.

Natani et al. [12] proposed a method for the malware detection by using the API frequencies and ensemble based classifiers. In the machine learning, the concept of ensemble is the approach that improve the accuracy and stability of the machine learning algorithms using boosting or bagging. In their paper, authors used multi-classifiers instead of a single classifier to analyze malware. They analyzed 100 malicious behaviors and choose 24 APIs used for that malicious behaviors. In addition, they measured the APIs‘ frequencies using a sandbox tool. The frequencies of the APIs were defined as the ratio of the number of invocations of certain APIs to the total invocations of APIs. These frequencies are used for the analysis with ensemble based classifiers. Finally they measured the accuracy of the analysis.

Wagener et al. [17] extracted the behavior information of malware by observing that malware invokes the system functions. The extracted information was the API invocation information of malware. Then, they compared the malware’ API invocation information and calculated similarities among malware variants. Their proposed method defined binary code for each API. The binary code was used to store API invocation patterns of malware. For similarity calculation, they used an edit distance matrix, and also used their own formular. Finally they calculated the similarities of malware variants.

Xu et al. [20] proposed the approach for the various malwares detection in the Windows platform. They extracted the API call sequences by analyzing PE binary code and calculated the similarities of the API sequences between the unknown malware and the known one. They implemented a PE binary parser tool themselves, because the third-party disassemblers extract some unnecessary features of malware samples and these unnecessary features made the performance of the analysis system very low. This tool generates the API sequences of malware, and if the malware are known, the sequences were stored in a signature database system. Those sequences are compared with the API sequences of unknown malware, then the similarity of the two sequences is calculated.

Liu Wu et al. [19] investigated techniques of malware behavior extraction based malicious API invocation, and presented the formal Malware Behavior Feature (MBF) extraction method. This Malware Behavior Feature was expressed in Boolean, and they proposed the malicious behavior feature based malware detection algorithm. Finally, they designed and implemented the MBF based malware detection system, and the experimental results showed that the accuracy rate of their MBF based detection system is high and it can detect newly appeared unknown malware.

Martin Apel et al. [5] investigated distance metrics to detect polymorphic malware effectively. Distance metrics are different distance measures in detail and they discussed desirable properties of a distance measure. They focused on behavioral features of malware and compared and experimentally evaluated different distance measures for malware behaviors. They selected an appropriate distance measure for grouping malware samples based on similar behaviors.

Some research had been conducted on file birthmarks using API invocation features, there are few researches that relates to features of malicious code. But Mamoun Alazab et al. [3] presented an automated method to extract API invocation features from binary executable files and analyzed them in order to understand their usages for malicious purpose. To address this gap, they attempted to automatically analyze the API features and classified behaviors of APIs based on the malicious intents hidden within any packed malicious files. They used the four-step methodology to develop a fully automated system to categorize six main categories of suspicious behaviors of API invocation features.

These existing researches used the API related features to capture the malicious behaviors. However, the existing researches mostly focused on how to define the malicious behaviors. They did not concentrate on elaborated comparison methods to generate more accurate similarity results. It is difficult to use very complex algorithms to compare the API features, and an advanced similarity calculation method is required to improve performance of the malware detection system. In this paper, we explain how to apply the sequence alignment algorithm which makes similarity results more accurate.

## 3 Background: Sequence Alignment

Sequence Alignment is widely used in the bioinformatics field. In the bioinformatics field, DNA sequences or RNA sequences are targeted to be aligned by a certain sequence alignment algorithm.[13] It is possible to find the similar or exactly same subsequences by aligning the DNA or RNA sequences. A DNA sequence consists of four kinds of characters (A, G, C, T) and each character is defined as a token. A sequence alignment algorithm relocates positions of tokens of each sequence in order to match as many tokens as possible.

DNA sequences have always noisy tokens. Therefore, if two DNA sequences are slightly different by some noisy tokens, it is necessary to arrange the sequences to reduce effects of noises. Sequence alignment is essential to find the common patterns in two DNA sequences. API call sequences share this aforementioned property of DNA sequences. API invocations in loops or some redundant API invocations could be regarded as noises. Accordingly, sequence alignment is required to match API call sequences. This is the reason why we selected a sequence alignment method to calculate the similarities of API call sequences.

The Sequence Alignment algorithm carries out the following steps. Assume that there are strings $a$ and $b$ and sizes of $a$ and $b$ are m and n respectively. s($a$, $b$) is a similarity of $a$ and $b$ and H(i,j) is the maximum similarity-score between a suffix of $a$[1:i] and a suffix of $b$[1:j] and W is the gap-score, and matrix H can be defined as follows.

$$H(i,0) = 0 \qquad ,0 \leq i \leq m \tag{1}$$
$$H(0,j) = 0 \qquad ,0 \leq j \leq n \tag{2}$$
$$H(i,j) = max \begin{cases} 0 \\ H(i-1,j-1) + s(a_i,b_j) \\ max_{k\geq 1}\{H(i-k,j) + W_k\} \\ max_{l\geq 1}\{H(i-k,j) + W_k\} \end{cases} \qquad ,1 \leq i \leq m, 1 \leq j \leq n \tag{3}$$

The each element of matrix H has the maximum value in four cases and the value of an element represents the similarity score, and the matrix H of which elements have these scores is used to find out the optimal aligned sequences. Eventually, building a matrix H of given two sequences is the most important process in sequence alignment and it takes most of computational resources.

There are two kinds of sequence alignment: local alignment and global alignment. Local alignment allows arbitrary-length segments of each sequence to be aligned, with no penalty for the unaligned portions of the sequences.[6, 4] Otherwise, closely related sequences with the same length are appropriate for global alignment. Here, the alignment is carried out from the start of the sequence till the end of the sequence to find out the best possible alignment. In this research, our method uses a local alignment algorithm. The local alignment algorithm is more applicable rather than the global alignment algorithm, because summation of local sequence similarities is more meaningful than the one global sequence similarity.

## 3.1 Problem to Apply a Sequence Alignment Method

When two sequences are arranged by the local alignment algorithm, computation time depends on the size of each sequence. If the lengths of API call sequences are very long, then sequence alignment time would be extremely high. This could be a signigicant problem when the similarity calculation system should analyze a large amount of malware variants. In order to confirm this problem again, we implement the sequence alignment program and used it to measure the execution time. In our experiments, we increased the length of two targeted sequence evenly and measured the execution time of each case. Figure 1 shows the results of the experiments. As shown in Figure 1, the execution time increases exponentially. This means that execution performance is affected by the lengths of two sequences and the lengths should be minimized to reduce the execution time of malware similarity calculation.

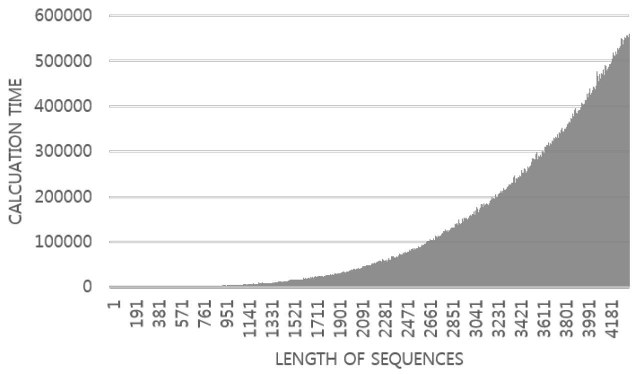


Figure 1: Execution Time for Sequence Alignment

The API call sequences include many repeated subsequences. The repeated subsequences occur when APIs are invoked in a loop. The number of iterations could be different between API call sequences of malware variants in the same family. Therefore, the repeated subsequences should be removed to make the final similarities not to be affected by the number of iterations. In addition, the removal of repeated subsequences also decreases the lengths of API call sequences. As a result, the overall execution time for alignment could be reduced.

## 4 Malware Similarity Calculation Method

We developed the similarity calculation system that can be used to identify malware variants. The system measures similarities with API call sequences of given malware samples. Figure 2 shows the overall architecture and the processing flow of the system. The processing flow has four steps:

- `Step 1.` – The behavior monitor module executes two given malware and logs the API calling sequences of them.
- `Step 2.` – The sequence refining module removes the repeated patterns in the API calling sequences.
- `Step 3.` – The sequence alignment module arranges the two API calling sequences.
- `Step 4.` – The final similarity result of aligned sequences is computed and reported.

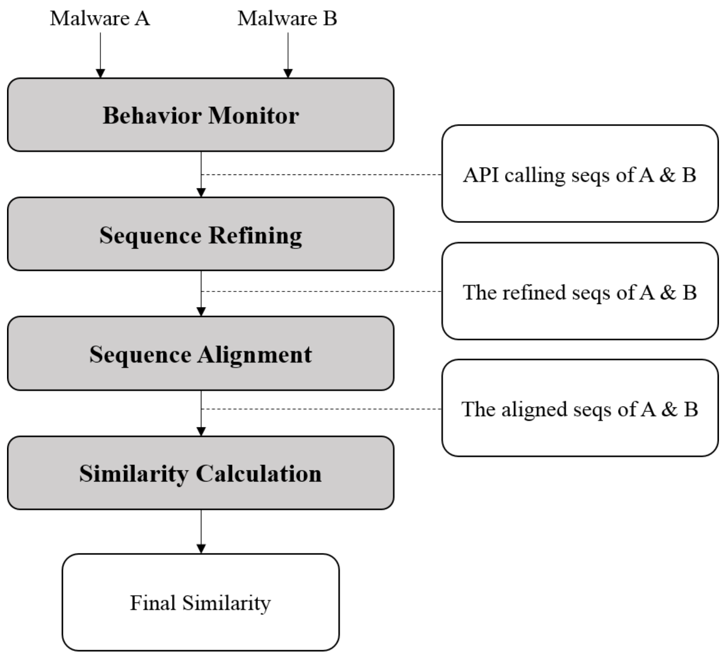


Figure 2: The Overall Architecture of Similarity Calculation System

The details about overall steps are explained in the next subsections.

## 4.1 The Behavior Monitoring Module

API's main purpose is to define a set of functionalities which can be used to control the operation system. Therefore, malware should use APIs in order to conduct various malicious behaviors. For example, if malware wants to download and install additional files into a victim's machine, it should use APIs that are related with network activities and file creation activities like connect(), recv(), CreateFile() and so on. Thus, our proposed system defines the behaviors of malicious code as a API call sequence and used it to compute the behavior similarities of two given malware samples.

The behavior monitor module captures the API call sequences of given malware samples. This module utilizes Cuckoo Sandbox which is an Open Source software to perform behavior monitoring[1]. We designed the behavior monitor module to log one hundred eighty two malicious APIs which are defined in Cuckoo Sandbox. Table x represents the categories and each number of APIs which are monitored. This behavior monitor module executes the given malware and logs the API call sequences. The execution time of the malware is set to two minutes. We found out that two minutes are sufficient to capture the malicious behavior by analyzing some malicious code manually.

## 4.2 The Sequence Refining Module

The sequence refining module's purpose is to remove the repeated API subsequences in the whole API calling sequence. This process makes the performance of next sequence alignment step faster by reducing the lengths of sequences. A repeating pattern is defined as a subsequence which is repeated at least twice in a whole sequence. For instance, assume that there is a string ”ACACACTA.” The repeating pattern of this string is the ”AC” string, because the ”AC” string is repeated three times in a whole string. As a result of refining process, redundant ”AC” strings are removed and ”ACACACTA” becomes ”ACTA”

The following pseudo code shows how the repeated pattern of sequence is removed.:

```
Function: remove_repeatition_in_api_seq()
Input    : api_sequence, max_len_repeatPattern
Output   : refined_api_seq

seqIndex ← 0;
while true do
    if seqIndex = lenapi_sequence then
        break;
    else
        for i ← 1 to max_len_repeatPattern do
            if found_repeat_pattern = true then
                break;
            end
            found_repeat_pattern ← remove_a_pattern(api_sequence, seqIndex, i);
            seqIndex += i ;
        end
    end
end
```

**Algorithm 1:** Removal of Repeated Patterns in API Sequences

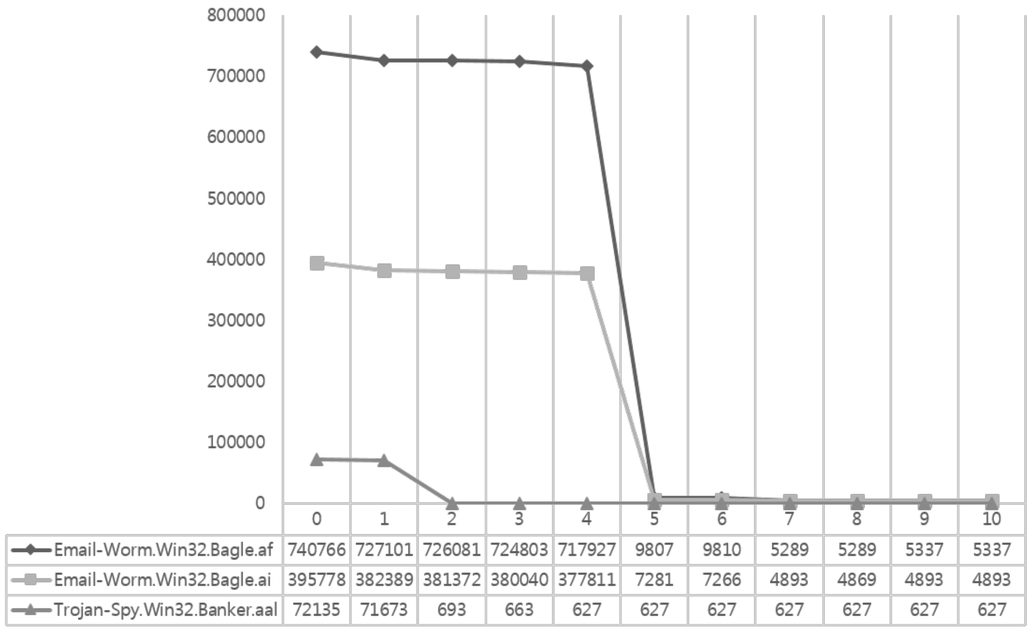
```
Function: remove_a_pattern()
input   : api_sequence, seqIndex, len_of_pattern
output  : found_repeat_pattern

$pattern \leftarrow api\_sequence[seqIndex : (seqIndex + len\_of\_pattern - 1)]$;
while $(temp \leftarrow getNextSubSeq(pattern\_len))$ != $null$ do
    if $temp = pattern$ then
        excludeSubseq(api_sequence, temp);
    else
        break;
    end
end
```

**Algorithm 2:** Removal of a Pattern

This sequence refining process can be a problem if it degrades the accuracy of similarity calculation. By comparing the similarity results which the sequence refining process is applied and the similarity results with the original sequences, we found out that this process makes no loss in the accuracy.

In some cases, the similarity results of the refined sequences were more accurate. In other words, removing the repeating patterns in the given sequences results in performance enhancement with no degradation of accuracy. Checking whether all of possible subsequences could be repeated is time-consuming. Thus, when the sequence refining module removes the repeating patterns in a sequence, the maximum length of the repeating pattern should be determined. we decided to check the sequence which is less than or equal to length seven. This maximum length of target subsequence for repetition checking is determined by an experimental evaluation.

Figure 3. shows the results of the experiments. we used three malicious samples which have many repeating patterns. While the module increases the maximum length of the repeating pattern which should be checked, the length of refined API call sequences is measured.



| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Email-Worm.Win32.Bagle.af | 740766 | 727101 | 726081 | 724803 | 717927 | 9807 | 9810 | 5289 | 5289 | 5337 | 5337 |
| Email-Worm.Win32.Bagle.ai | 395778 | 382389 | 381372 | 380040 | 377811 | 7281 | 7266 | 4893 | 4869 | 4893 | 4893 |
| Trojan-Spy.Win32.Banker.aal | 72135 | 71673 | 693 | 663 | 627 | 627 | 627 | 627 | 627 | 627 | 627 |

Figure 3: The Lengths of Refined Sequences

In Figure 3, the *x* axis is the maximum length of repeating patterns and the *y* axis is the total length of the refined API cal sequence. The total length of the refined API call sequence is reduced dramatically, when the maximum length of the repeating pattern is set to 5. Additionally, when the maximum length of the repeating pattern is set to 7, each refined API call sequence of three malicious samples is shortest.

### 4.3 The Sequence Alignment Module

After the sequence refining process is conducted, the refined API call sequences are arranged by the sequence alignment algorithm. The sequence alignment algorithm in this module is Smith-Waterman algorithm. The Smith-Waterman algorithm was first proposed by Temple F. Smith and Michael S. Waterman in 1981 [15] and it is widely used in various fields. The aligned API call sequences can be similar to the examples in Figure 4. The Smith-Waterman algorithm inserts the Gap into sequences to make them aligned properly.

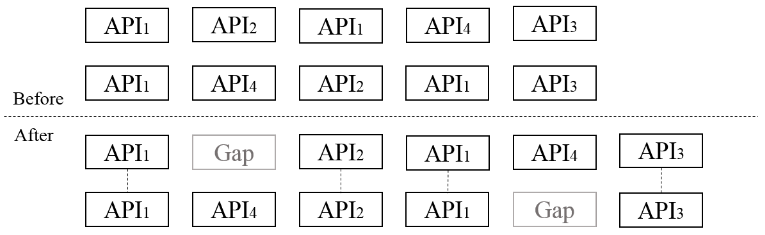


Figure 4: The Example of Sequence Alignment

### 4.4 The Similarity Calculation Module

If two refined API call sequences are arranged by the sequence alignment algorithm, the matched subsequence can be extracted from them. The similarity calculation equation is as follows:

$$Sim(A,B) = \frac{\sum_{i=1}^{k} L_i}{(m+n)/2} \qquad ,where \quad m = length(A) \; \& \; n = length(B) \tag{4}$$

*L* is the length of the matched subsequences. *m* and *n* are the lengths of each refined API call sequence respectively. The range of similarity is 0 to 1, and the bigger value means that the two sequences are more similar.

## 5 Experiment

### 5.1 The Measurement of Similarities

We measured the similarities of the malicious code using the proposed system to detect the malware variants. We collected malware samples from VXheaven[2]. We used one hundred fifty malware samples from ten malware families. Each malware family has fifteen malware variants. The similarity results are shown in Figure 5. In Figure 5, the *x* axis means the name of each malware families and the *y* axis means the average similarity values. The black bar represents the average of the similarities among malware samples in the same family and the gray bar represents the average of the similarities between malware samples from different families. Most of similarities of samples from the same malware family are higher

Table 1: Performance Time

| | Calculation time without sequence refinement(ms) | Calculation time with sequence refinement(ms) | Reduced time(%) |
|---|---|---|---|
| Dadobra.am vs Dialer.ap | 81169.4 | 450.445 | 99% |
| Dadobra.am vs Banload.aaa | 150010 | 2375.75 | 98% |
| Dadobra.am vs .Dadobra.af | 30384.5 | 303.324 | 99% |
| Dialer.ap vs Banload.aaa | 54962.9 | 2499.52 | 95% |
| Dialer.ap vs Dadobra.af | 9736.45 | 320.248 | 97% |
| Banload.aaa vs Dadobra.af | 26098.3 | 2362.08 | 91% |

than those from different malware families. Therefore, The results show the larger difference between the similarities of samples in the same family and those from different families. This means that our method can effectively distinguish malware samples in the same family.

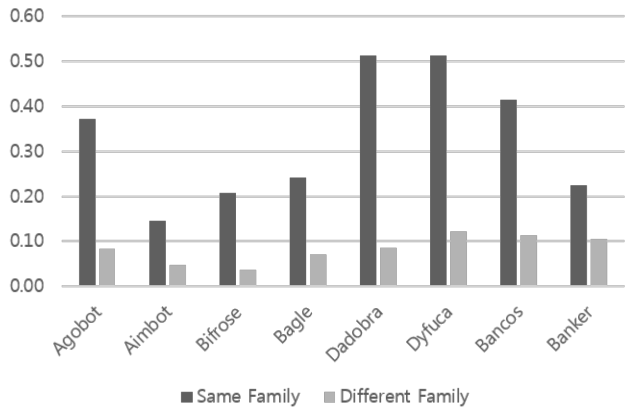


Figure 5: The Malware Similiarity Result

## 5.2 The Measurement of Overall Processing Time

We had experiments to prove our performance improvement. First, we measured the similarity calculation time when the sequence refinement is not applied. Secondly, we measured the similarity calculation time when the sequence refinement is applied. We used four malicious samples in these experiments and the results are shown in Table 1. The similarity calculation with sequence refinement results has better performance. The ratio of reduced time is in the range from 91% to 99%.

## 6 Conclusion

In this paper, we proposed a similarity calculation system which can be used to detect malware variants of the same family. The system consists of four processes. The behavior monitor module has a role to capture the API call sequences of input malicious files and the sequence refining module removes the repeating patterns in the API call sequences to reduce the total length of it. The sequence alignment module aligns the sequences and finds the matched sequences from two given API call sequences. The similarity calculation module computes final similarities. In the refining process, especially, we suggested that repeating patterns should be removed in order to improve the performance of the alignment process. As a result, alignment processing time could be reduced dramatically. To verify whether the system could be used to detect the malware variants, we have a similarity calculation experiments and it confirms that our proposed system can be used to distinguish the malware variants of the same family.

The followings are the limitations and future work. We will improve the sequence alignment algorithm to enhance its performance further. Although our method to reduce the lengths of input sequences is proposed in this paper, an algorithmic solution is basically needed. We will experiment with the large number of malicious samples to test our proposed system in detail.

## 7 Acknowledgments

This work was supported by ICT R&D program of MSIP/IITP. [10044938, The Development of Cyber Attacks Detection Technology based on Mass Security Events Analysing and Malicious code Profiling]

## References

[1] Cuckoo Sandbox. http://cuckoosandbox.org/.

[2] VxHeaven. http://vxheaven.org/.

[3] M. Alazab, S. Venkataraman, and P. Watters. Towards understanding malware behaviour by the extraction of api calls. In *Proc. of the 2nd Cybercrime and Trustworthy Computing Workshop (CTC'10), Ballarat, VIC*, pages 52–59. IEEE, July 2010.

[4] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman. Basic local alignment search tool. *Journal of molecular biology*, 215(3):403–410, October 1990.

[5] M. Apel, C. Bockermann, and M. Meier. Measuring similarity of malware behavior. In *Proc. of the IEEE 34th Conference on Local Computer Networks (LCN'09), Zurich, Swiss*, pages 891–898. IEEE, October 2009.

[6] J. Cohen. Bioinformatics—an introduction for computer scientists. *ACM Computing Surveys (CSUR)*, 36(2):122–158, June 2004.

[7] M. Egele, T. Scholte, E. Kirda, and C. Kruegel. A survey on automated dynamic malware-analysis techniques and tools. *ACM Computing Surveys (CSUR)*, 44(2):6, February 2012.

[8] K.-S. Han, I.-K. Kim, and E. G. Im. Detection methods for malware variant using api call related graphs. In *Proc. of the 1st International Conference on IT Convergence and Security (ICITCS'12), Suwon, Korea, LNCS*, volume 120, pages 607–611. Springer-Verlag, December 2012.

[9] K.-S. Han, I.-K. Kim, and E. G. Im. Malware classification methods using api sequence characteristics. In *Proc. of the 1st International Conference on IT Convergence and Security (ICITCS'12), Suwon, Korea, LNCS*, volume 120, pages 613–626. Springer-Verlag, December 2012.

[10] G. Hunt and D. Brubacher. Detours: Binary interception of win32 functions. In *Proc. of 3rd USENIX Windows NT Symposium (WINSYM'99), Seattle, WA, USA*, pages 135–144. USENIX Association, July 1999.

[11] B. Kang, T. Kim, H. Kwon, Y. Choi, and E. G. Im. Malware classification method via binary content comparison. In *Proc. of the 1st 2012 ACM Research in Applied Computation Symposium (RACS'12), New York, USA*, pages 316–321. ACM, October 2012.

[12] P. Natani and D. Vidyarthi. Malware detection using api function frequency with ensemble based classifier. In *Proc. of the 1st Security in Computing and Communications (SSCC'13), Mysore, India, LNCS*, volume 377, pages 378–388. Springer-Verlag, August 2013.

[13] 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*, 48(3):443–453, March 1970.

[14] Y. Qiao, Y. Yang, L. Ji, and J. He. Analyzing malware by abstracting the frequent itemsets in api call sequences. In *Proc. of the 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom'13), Melbourne, VIC*, pages 265–270. IEEE, July 2013.

[15] T. F. Smith and M. S. Waterman. Identification of common molecular subsequences. *Journal of molecular biology*, 147(1):195–197, March 1981.

[16] A. H. Sung, J. Xu, P. Chavez, and S. Mukkamala. Static analyzer of vicious executables (save). In *Proc. of the 20th Computer Security Applications Conference (ACSAC'04), Tucson, AZ, USA*, pages 326–334. IEEE, December 2004.

[17] G. Wagener, A. Dulaunoy, et al. Malware behaviour analysis. *Journal in computer virology*, 4(4):279–287, November 2008.

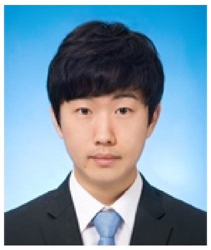
[18] C. Willems, T. Holz, and F. Freiling. Toward automated dynamic malware analysis using cwsandbox. *ACM Computing Surveys (CSUR)*, 5(2):32–39, March 2007.

[19] L. Wu, R. Ping, L. Ke, and D. Hai-xin. Behavior-based malware analysis and detection. In *Proc. of the 1st International Workshop on Complexity and Data Mining (IWCDM'11), Nanjing, Jiangsu*, pages 39–42. IEEE, September 2011.

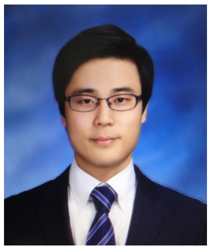
[20] J.-Y. Xu, A. H. Sung, P. Chavez, and S. Mukkamala. Polymorphic malicious executable scanner by api sequence analysis. In *Proc. of the 4th International Conference on Hybrid Intelligent Systems (HIS'04), Kitakyushu, Japan*, pages 378–383. IEEE, December 2004.

## Author Biography

**In Kyoem Cho** is a student in master's course at Hanyang University, Seoul, Korea. He received a B.S. from Hanyang University in 2014, Seoul, Korea. His primary researches are parallel computing, malware similarity analysis and categorization.



**TaeGuen Kim** is a student in doctor's course at Hanyang University, Seoul, Korea. He received B.S. and M.S. from Hanyang University in 2011 and 2013, respectively. His primary researches are in the areas of malware analysis, scada security and software vulnerability assessment.



**Yu Jin Shim** is a student in master's course at Hanyang University, Seoul, Korea. He received a B.S. from Seokeong University in 2014, Seoul, Korea. His primary researches are in the areas of malware analysis and malware similarity Analysis.

**Haeryong Park** is a member of Information Security Technology Development Department of Korea Internet & Security Agency, Seoul, Korea. He received a B.S. from Chonnam National University in 1999, Chonnam, Korea, , a M.S. from Seoul University in 2001, Seoul, Korea, and Ph.D. from Chonnam National University in 2006,Chonnam, Korea. His primary research are cryptographic algorithm design & analysis , cyber balckbox technology development and cloud service security.

**Bomin Choi** is a member of Information Security Technology Development Department of Korea Internet & Security Agency, Seoul, Korea. she received a B.S. and a M.S. from Gachon National University in 2012 and 2014, Gachon, Korea. Her primary researches are profiling malware, big data and intelligence algorithm.

**Eul Gyu Im** is a faculty member of Department of Computer and Software at Hanyang University, Seoul, Korea. He received a B.S. and a M.S. from Seoul National University in 1992 and 1994, respectively. He received Ph.D. from University of Southern California in 2002. Before he joined Hanyang University, he had worked for National Security Research Institute in Daejeon, Korea. His primary researches are in the areas of malware analysis, scada security, software vulnerability assessment and RFID security.