Discovering Similarities in Malware Behaviors by Clustering of API Call Sequences

Fatima Al Shamsi[1], Wei Lee Woon[2], and Zeyar Aung[2(✉)]

[1] Abu Dhabi Systems and Information Centre, Abu Dhabi, UAE
Fatima.AlShamsi@adsic.abudhabi.ae
[2] Khalifa University of Science and Technology, Masdar Institute, Abu Dhabi, UAE
{wei.woon,zeyar.aung}@ku.ac.ae

Abstract. New genres of malware are evading detection by using polymorphism, obfuscation and encryption techniques. Hence, new strategies are needed to overcome the limitations of current malware analysis practices. In this paper, we propose an unsupervised learning (clustering) framework to complement the supervised learning (i.e., classifier-based malware detection) approach. We cluster malware instances to discover similarities in their dynamic behaviors and to detect new malware families. For that, we utilize Application Programming Interface (API) call sequences to represent the behaviors of malware in dynamic runtime environment. We investigate three sequence comparison algorithms, namely, Optimal Matching (OM), Longest Common Subsequence (LCS), and Longest Common Prefix (LCP) for calculating sequence–sequence distances to be used for hierarchical clustering. Among the three algorithms, LCP is found to be both the most effective in terms of clustering quality and the most efficient in terms of time complexity (linear-time).

Keywords: Malware · API calls · Clustering · Malware patterns

1 Introduction

Nowadays malware becomes prevalent everywhere: from conventional servers and PCs to mobile and IoT devices [1]. Among an average of 1 million malware instances appear every day, most of them are variants of the existing ones [2]. Polymorphism, obfuscation and encryption techniques are applied by malware developers to evade detection by anti-malware programs [3,4]. They generate different malware instances from the same malware family to avoid traditional detection techniques. As a result, members of the same malware family are functionally similar to one another [5] although their binaries can be quite different. A recent study revealed that there was a decrease in the number of new malware families detected between the year of 2014 and 2015. However, there was noticeable increase with a percentage of 36% in the number of malware variants detected between 2014 and 2015 [6]. With the increased number of malware

L. Cheng et al. (Eds.): ICONIP 2018, LNCS 11304, pp. 122–133, 2018.
https://doi.org/10.1007/978-3-030-04212-7_11

instances, effective and computationally efficient methods are required to detect similar malware behaviors in order to accelerate the malware analysis process. Leveraging data mining techniques like clustering can help us fulfill this requirement.

In this paper, we conduct a clustering analysis of malware based on their dynamic behavioral patterns represented by Application Programming Interface (API) call sequences. Although supervised learning (classification) approach, such as [7–10], is the most common data mining method for malware analysis (detection), unsupervised learning (clustering), such as [11–14], is also a useful malware analysis tool that can complement supervised learning. The benefits of a clustering analysis are as follows.

1. Organizing malware into homogeneous clusters can help us better understand the malware activities [15]. Visualization of clustering results can provide us with the "bird-eye view" of the similarities/differences of malware in their behavioral patterns such as host infection procedures, attack routines, and dissemination mechanisms.
2. Clustering analysis enables "variant detection". It allows us to uncover behaviorally similar malware that should belong to the same family even though identified as distinct and sometimes even assigned to different malware categories by commercial anti-malware programs. This facilitates malware analysis experts to better sample malware variants of the same/different malware families when carrying an in-depth manual analysis process [16].
3. Clustering allow us to identify the emergence of new malware families by examining the formation of new clusters and/or sub-clusters. In fact, clustering is recommended as an adaptive technique to look for unknown patterns potentially corresponding to new types of attacks [15,17].

In our study, we first transform API call sequences into symbolic strings to allow us to detect similar patterns. Then, we calculate all-against-all pairwise similarities (and hence distances) of malware samples in our dataset in order to construct the distance matrix and subsequently perform a hierarchical clustering using the Weighted Pair Group Method with Arithmetic Mean (WPGMA) algorithm.

The new contributions of this paper are as follows.

1. We have investigated three different similarity/distance functions, namely Optimal Matching (OM), Longest Common Subsequence (LCS) (as in [14]), and Longest Common Prefix (LCP) and compare the results in terms of clustering quality (silhouette coefficient) and time complexity.
2. It turns out that LCP offers the best results with the best clustering quality and the lowest time complexity (linear-time). To our best knowledge, we are the first to apply the LCP function in API sequence-based malware analysis and demonstrate its usefulness. The results suggest that prefix (i.e., initial API subsequence) could be used as a quick but reliable indicator for overall similar behaviors in malware.

This paper is a summary version of the master's thesis [18] by the first author.

2 Related Work

Supervised learning (classification) and unsupervised learning (clustering) are the two main approaches used in data mining-based malware analysis. The works [7–10] are some examples of malware classification. Some examples of malware clustering are described below.

Bayer et al. [11] provided a scalable approach for malware clustering. They investigated malware applications that share similar behavioral traits through three phase model, namely, dynamic analysis phase, identification of behavioral signatures based on system calls phase, and the clustering phase. To suggest a feasible solution to the proximity search of nearest neighbor, the clustering algorithm they used in their study is based on locality sensitive hashing. Their experiment effectively and efficiently clustered samples sharing similar behavioral patterns.

Kim et al. [12] calculated the similarity between android malware applications by utilizing Androguard to construct control flow graphs. Similar behaviors were compared by reviewing the structural information collected from the flow graphs to match target applications. Moreover, clustering sub-families of each family and identifying representative malware samples help reduce the number of similarity calculation. The clustering algorithm used was DBSCAN.

Qiao et al. [13] clustered API sequences based on frequent pattern mining algorithms. The analysis framework consists of three steps, API call abstraction, frequent itemset mining, and analysis. For API call abstraction, dynamic analysis tools, namely, CWSandbox and Cuckoo Sandbox were used to extract behavioral reports of malware binaries. API pattern mining for frequent itemsets was identified using Apriori Program. Nevertheless, frequent API calls are used as an input to calculate the similarity between different malware binaries. The output served as an input for computing signature sequences. Instead of API calls, strings of opcodes representing malware behaviors are clustered using a suffix tree in the recent work by Oprişa et al. [19].

Zhong et al. [14] proposed analysis system, namely ARIGUMA, which is composed of four components, Sandbox, code analyzer, visualizer, and web User Interface (UI). The system use code analyzer to classify malware applications based on their behavior. Comparison between malware instructions was completed using LCS. Given that LCS algorithm requires polynomial time by dynamic programming, the computational time was reduced by filtering out the numeric features of codes.

3 Proposed Malware Clustering Framework

3.1 Generating Encoded API Call Sequences

Instead of looking directly into the characteristics of malware through its binaries (executable codes), we examine its Application Programming Interface (API) call information in order to better understand its behavioral pattern. This approach gives us three advantages. (1) It enables a dynamic analysis of malware

though run-time API calls instead of a static analysis. (2) It is much more efficient to deal with API call sequences (whose lengths are typically less than 1000) than malware binaries (whose lengths are typically a few megabytes). (3) Two instances of malware from the same family may not have a detectable common signature in their binaries because of polymorphism and/or obfuscation; however, their behaviors in terms of API call sequences are expected to be quite similar.

The API-based malware detection system (APIMDS) dataset used in this study is obtained from Ki et al. [20]. It contains the API system call sequences for 6 nominal malware categories, namely, backdoor, packed, PUP (Potentially Unwanted Programs), Trojan, virus and worm. Those sequences were obtained by executing the malware samples using virtual machine environment such as VirtualBox. (The detailed process can be referred to in the paper [20].) From the whole dataset, we randomly select 1,797 of them for our clustering studies.

After obtaining the API call sequences of those selected malware, we preform "sequence encoding". Its purpose is to reduce the dimensionality of sequence unique states as well as to reduce the time complexity for calculating distances between the different sequences. API code sequence generation step is carried out by referring to MSDN library [21] for categorizing API system calls into 26 categories as given in Table 1.

Table 1. Codes for API call categories.

Category	Code	Category	Code
I/O Create	A	Window class functions	N
I/O Open	B	File I/O	O
I/O Write	C	Data exchange	P
I/O Find	D	Keyboard	Q
I/O Read	E	Volume management	R
I/O Access	F	Windows GDI	S
Load	G	Menus and Other resources	T
Debugging & Error handling	H	Interprocess communication	U
Memory management	I	Process and thread	V
Delete/Destroy	J	Networking and internet	W
Get info	K	System information/services	X
Windows	L	Device management	Y
COM	M	Other	Z

3.2 Generating Distance Matrix

The concept of "similarity" can be applied to a pair of encoded API call sequences representing dynamic behaviors of malware. Similar malware instances

share a certain amount of behaviors in common. The most common algorithms that used to measure the similarity between the two given sequences are: (i) Optimal Matching (OM), (ii) Longest Common Subsequence (LCS), and (iii) Longest Common Prefix (LCP) [22,23].

The purpose of these algorithms are to extract the lengths of the "common patterns" of malware behaviors in their own unique ways. The longer the two sequences that follow a common pattern, the more similar they are and the lesser the difference (a.k.a. "distance") between them. Distance can be regarded as the inverse of similarity and can be easily derived from it, and vice versa.

Given a dataset of n malware samples, for each distance function, we need to construct an all-against-all (i.e., $n \times n$) distance matrix, where each matrix entry d_{ij} is the pairwise distance between two malware sequences indexed i and j. In fact, we only need the compute the distance values in the upper triangle of the matrix (whose size is $n(n-1)/2$), because the lower triangle is the mirror image of the upper one. This distance matrix is to be used as an input in the subsequent clustering procedure.

In our research, we use R's `TraMineR` package [23] to compute the OM, the LCS, and the LCP distances. Their brief descriptions are given below.

Optimal Matching (OM) Distance: OM produces edit distance by calculating the minimal amount of edits required to change a pair of malware behaviors X and Y into a different pair such that they are similar to some extent [14]. It is based on Needleman-Wunsch sequence alignment algorithm [24], which is commonly used in Bioinformatics. The insertion/deletion cost is single value to be specified by the user, while the substitution cost matrix (of replacing one API call category for another) can be produced by assigning constant values, or using the transition rate between sequences observed in the dataset. All the insertion/deletion/substitution costs are set to 1 in our experiment. Equation 1 defines the OM distance recursively [22].

$$d_{OM}(X_i, Y_j) = \min \begin{cases} d_{OM}(X_{i-1}, Y_j) + \text{deletion cost} \\ d_{OM}(X_i, Y_{j-1}) + \text{insertion cost} \\ d_{OM}(X_i, Y_{j-1}) + \text{substitution cost if } X_i \neq Y_j \end{cases} \tag{1}$$

The OM distance of two sequences of lengths $|X|$ and $|Y|$ can be computed in a quadratic $O(|X| \times |Y|)$ time [22,24]. It should be noted that OM directly computes the distance without needing to compute the similarity first.

Longest Common Subsequence (LCS) Distance: Given two sequences X, and Y, LCS calculates the largest number of common sequential elements between them. For example, let the two sequences be $X =$ **A-B-A-C-A-D** and $Y =$ **A-B-C-A-P-Q-R**. Then, their LCS is **A-B-C-A**. LCS measures the similarity and difference of instructions between sequences that have same relative order, but not necessarily adjacent. For the two sequences X_i, and Y_j, where i and j represent the length of the longest common subsequence present in both sequences, Eq. 2 defines the length of LCS recursively [22].

$$LCS(X_i, Y_j) = \begin{cases} 0 & \text{if } i{=}0 \text{ or } j{=}0 \\ LCS(X_{i-1}, Y_{j-1}) + 1 & \text{if } X_i = Y_j \\ \max(LCS(X_i, Y_{j-1}), LCS(X_{i-1}, Y_j)) & \text{if } X_i \neq Y_j \end{cases} \quad (2)$$

The LCS distance d_{LCS} is derived from the LCS length using Eq. 3.

$$d_{LCS}(X, Y) = |X| + |Y| - 2{\cdot}LCS(X, Y) \quad (3)$$

LCS was also used in the previous malware clustering study by Zhong et al. [14]. Computing LCS of two sequences with $|X|$ and $|Y|$ elements using dynamic programming is of a quadratic $O(|X| \times |Y|)$ time complexity [14].

Longest Common Prefix (LCP) Distance: Given two sequences X, and Y, LCP calculates the length of the longest common prefix shared by them. For example, let the two sequences be $X =$ **A-B-A-C-A-D** and $Y =$ **A-B-C-A-P-Q-R**. Then, their LCP is **A-B**. LCP offers a quick first-order approximation of the similarity of two given sequences by looking at their prefixes only. For the two sequences X, and Y, Eq. 4 defines the length of LCP [22].

$$LCP(X, Y) = |\{u \neq \lambda : X^{|u|} = u = Y^{|u|}\}| \quad (4)$$

where λ is an empty string, and $X^{|u|}$ and $Y^{|u|}$ ($|u| \leq \min(|X|, |Y|)$) are the longest prefix sequences of length $|u|$ of X and Y respectively. Just like the LCS distance, the LCP distance d_{LCP} can be derived from the LCP length in the same manner as in Eq. 3.

LCP of two sequences with $|X|$ and $|Y|$ elements can be calculated in linear time, which is $O(\min(|X|, |Y|))$.

3.3 Clustering

Clustering enables grouping of similar encoded API sequences representing malware behaviors to detect repeating patterns. Moreover, sub-patterns in the same cluster can also be recognized and further analyzed. To evade binary code signature-based detection by anti-malware tools, malware developers make minor changes (polymorphism) or apply obfuscation to the source code of existing malware to produce new variants. This greatly increases the number of variants for the same malware with different signatures. API sequence-based clustering can help reveal those kinds of variants easily.

Clustering uses the distance matrix generated from the above step. In our case, we conduct 3 distinct clustering exercises using 3 distance matrices with OM, LCS, and LCP distance functions respectively. We employ a hierarchical clustering algorithm called Weighted Pair Group Method with Arithmetic Mean (WPGMA) [25] to generate clusters. (Note: we also tried k-means clustering, but the results are not as good as those by WPGMA. We do not show the k-means results in this paper because of the page limit.)

The WPGMA algorithm constructs a rooted tree (dendrogram) that reflects the structure present in a pairwise distance matrix. At each step, the nearest

two clusters, say i and j, are combined into a higher-level cluster $i \cup j$. Then, its distance to another cluster k is simply the arithmetic mean of the distances between k and members of $i \cup j$ [25].

$$d_{(i \cup j),k} = (d_{i,k} + d_{j,k})/2 \tag{5}$$

Clustering n elements using WPGMA takes $O(n^2)$ space and $O(n^2)$ time. There are $n-1$ iterations, with $O(n)$ work in each one [26]. We use R's `Cluster` package [27] to perform WPGMA clustering.

4 Results and Discussions

In this section, we will discuss the results of our proposed framework on the 1,797 malware samples in our experimental dataset. The will identify the suspicious API calls and examine the suspicious call patterns based on the clustering results. Visualization and statistical validation of the clustering results are also performed along with a running time analysis.

4.1 Clusters of Similar Malware Behavioral Patterns

Clustering helps us to find similar groups among malware API sequences. Hierarchical clustering (WPGMA) algorithm is used to cluster sequences and identify similar patterns of malicious behavior. Hierarchical clustering divides data into clusters and produce a tree structure to display the results. As mentioned above, three distinct clustering tasks using OM, LCS, and LCP distance matrices are performed. However, only the results of LCP will be shown here as an example.

Figure 1 represents clustering results for LCP. By referring to the figure, we can notice that there is a clear distinction between the clusters, except for Cluster #1 which is not very homogeneous. Each cluster exhibits a different set of behaviors. For example, in Cluster #4, we can notice that there are 6 sub-clusters that are very similar in initial subsequences yet differ in overall sequence length. These sub-clusters follow the same pattern, namely, `Networking and Internet > Get Info > Debugging and Error Handling > I/O Create > File I/O > Get Info > I/O Open > Window Class`. This pattern is an example of the initial subsequence taken by all sub clusters within the same cluster.

Likewise, for a large and heterogeneous cluster like Cluster #1 (with 1278 members), we can also apply sub-clustering to generate more homogeneous sub-clusters. Figure 2 shows the sub-clustering result for Cluster #1. It should be noted that for the sub-clusters that are still heterogeneous and sufficiently large, such as Sub-Cluster #1 (with 984 members), it is still possible to further sub-divide them. (The sub-division results are not shown in this paper.)

One notable observation is that malware variants which belongs to different nominal categories (like virus, backdoor, worm, and Trojan) can exhibit similar behaviour and thus belong to the same cluster. An example can be seen in Fig. 3, which shows malware variants with different nominal categories having similar API sequence patterns—indicating a strong possibility that they are variants of each other, albeit differently categorized by the anti-malware companies.

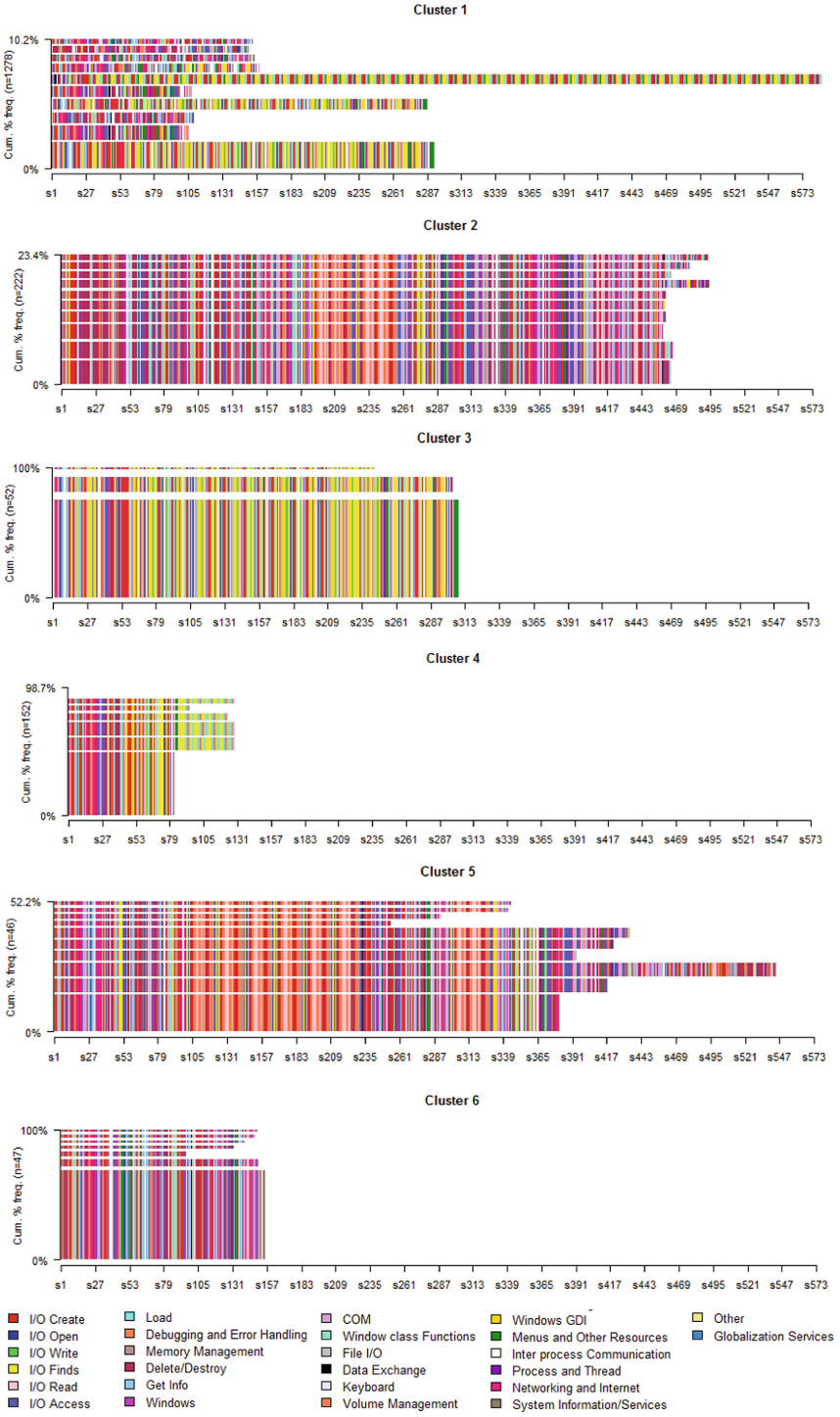

Fig. 1. Clustering results with LCP distance function (number of clusters = 6).

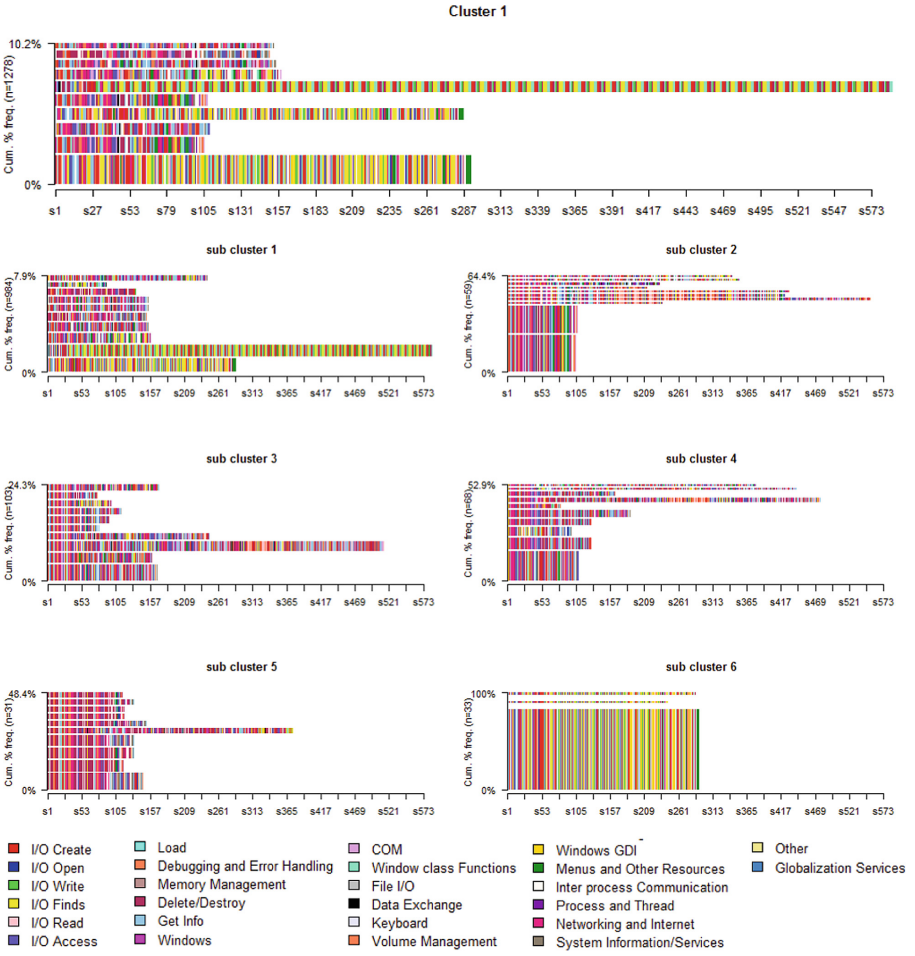

Fig. 2. Sub-clustering of Cluster #1 in Fig. 1.

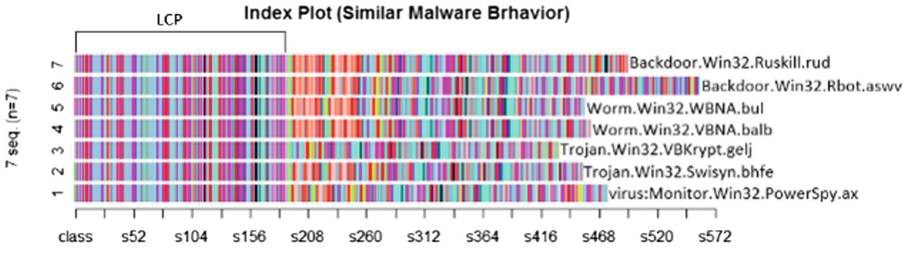

Fig. 3. Zooming into a subset of Cluster #2 of LCP clustering result, exhibiting different nominal categories of malware in the same cluster.

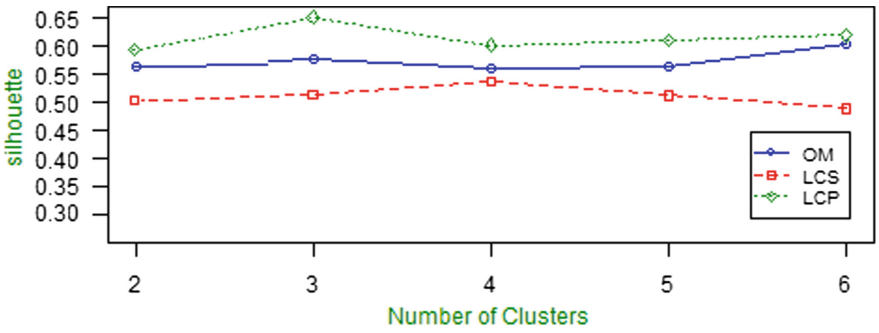

Fig. 4. Silhouette coefficients by 3 distance functions (OM, LCS, and LCP).

4.2 Clustering Quality

To statistically validate clustering results, silhouette coefficient ($\bar{s}$) is used to determine the optimal number of clusters which produces best clustering results. Given the distance matrix, the normalized distance m_{ij} between two sequences indexed i and j is calculated from the matrix element d_{ij} as: $m_{ij} = d_{ij}/d_{\max}$, where $d_{\max}$ is the maximum distance value in the matrix.

The average intra-cluster distance of the sequence i to other members of its own cluster C (donated as a), the average inter-cluster distance of the sequence i to the members in the nearest cluster C' (denoted as b), and finally the silhouette coefficient ($\bar{s}$) for the sequence i are defined in Eq. 6. The silhouette coefficient value always lies between -1 and 1. The general guideline is that a coefficient greater than 0.5 represents a meaningful clustering result [28]. The higher the coefficient, the better the clustering quality.

$$a = \sum_{j \in C \setminus \{i\}} \frac{m_{ij}}{|C| - 1}; \quad b = \min\Big(\sum_{j \in C'} \frac{m_{ij}}{|C'|} \Big); \quad \bar{s} = \frac{b - a}{\max(a, b)} \tag{6}$$

Figure 4 shows the average silhouette coefficients for 3 WPGMA clusterings results of the 1,797 malware samples based on 3 different distance functions of OM, LCS, and LCP. Here, LCP is found to offer the best result with the highest average silhouette coefficient of **0.65** when the number of clusters is 3.

5 Conclusion

In this paper, we have proposed a clustering framework to detect similar malware behaviors based on the comparison of API call sequences. We have investigated the performances of three sequence–sequence comparison functions, namely, OM, LCS, and LCP, as a basis for clustering. To the best of our knowledge, we are the first to apply the LCP function in API sequence-based malware analysis, which offers the most promising results in terms of clustering quality and time

complexity (linear-time). This gives us a strong indication that prefix (i.e., initial API subsequence) can serve as a quick and reliable descriptor for overall similar behaviors in malware.

References

1. Alwahedi, S., Al Ali, M., Ishowo-Oloko, F., Woon, W.L., Aung, Z.: Security in mobile computing: attack vectors, solutions, and challenges. In: Agüero, R., Zaki, Y., Wenning, B.-L., Förster, A., Timm-Giel, A. (eds.) MONAMI 2016. LNICST, vol. 191, pp. 177–191. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-52712-3_13
2. Lee, T., Kwak, J.: Effective and reliable malware group classification for a massive malware environment. Int. J. Distrib. Sens. Netw. (2016). Article ID 4601847
3. Cho, I.K., Im, E.G.: Extracting representative API patterns of malware families using multiple sequence alignments. In: Proceedings of the 2015 ACM Conference on Research in Adaptive and Convergent Systems (RACS), pp. 308–313 (2015)
4. Stamp, M.: Information Security: Principles and Practice, 2nd edn. Wiley, New York (2011)
5. Dinh, A., Brill, D., Li, Y., He, W.: Malware sequence alignment. In: Proceedings of the 2016 IEEE International Conferences on Big Data and Cloud Computing (BDCloud), Social Computing and Networking (SocialCom), Sustainable Computing and Communications (SustainCom), pp. 613–617 (2016)
6. Symantec Enterprise Security: 2016 symantec internet security threat report. Technical report (2016). http://www.symantec.com/security-center/threat-report
7. Narayanan, A., Chen, Y., Pang, S., Tao, B.: The effects of different representations on static structure analysis of computer malware signatures. Sci. World J. **2013**, 8 (2013). Article ID 671096
8. Kate, P.M., Dhavale, S.V.: Two phase static analysis technique for Android malware detection. In: Proceedings of the 3rd International Symposium on Women in Computing and Informatics (WCI), pp. 650–655 (2015)
9. Milosevic, N., Dehghantanha, A., Choo, K.K.R.: Machine learning aided Android malware classification. Comput. Electr. Eng. **61**, 266–274 (2017)
10. Al Ali, M., Svetinovic, D., Aung, Z., Lukman, S.: Malware detection in Android mobile platform using machine learning algorithms. In: Proceedings of the 2017 IEEE International Conference on Infocom Technologies and Unmanned Systems (Trends and Future Directions) (ICTUS), pp. 763–768 (2017)
11. Bayer, U., Comparetti, P.M., Hlauschek, C., Krügel, C., Kirda, E.: Scalable, behavior-based malware clustering. In: Proceedings of the 2009 Network and Distributed System Security Symposium (NDSS), pp. 1–18 (2009)
12. Kim, J., Kim, T.G., Im, E.G.: Structural information based malicious app similarity calculation and clustering. In: Proceedings of the 2015 ACM Conference on Research in Adaptive and Convergent Systems (RACS), pp. 314–318 (2015)
13. Qiao, Y., He, J., Yang, Y., Ji, L.: Analyzing malware by abstracting the frequent itemsets in API call sequences. In: Proceedings of the 2013 IEEE 12th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom), pp. 265–270 (2013)
14. Zhong, Y., Yamaki, H., Yamaguchi, Y., Takakura, H.: ARIGUMA code analyzer: Efficient variant detection by identifying common instruction sequences in malware families. In: Proceedings of the 2013 IEEE 37th Annual Computer Software and Applications Conference (COMPSAC), pp. 11–20 (2013)

15. Cordeiro De Amorim, R., Komisarczuk, P.: On partitional clustering of malware. In: Proceedings of the 1st International Workshop on Cyberpatterns: Unifying Design Patterns with Security, Attack and Forensic Patterns (CyberPatterns), pp. 47–51 (2012)
16. Perdisci, R., U, M.: VAMO: Towards a fully automated malware clustering validity analysis. In: Proceedings of the 28th ACM Annual Computer Security Applications Conference (ACSAC), pp. 329–338 (2012)
17. Monshizadeh, M., Yan, Z.: Security related data mining. In: Proceedings of the 2014 IEEE International Conference on Computer and Information Technology (CIT), pp. 775–782 (2014)
18. Al Shamsi, F.: Mapping, Exploration, and Detection Strategies for Malware Universe. Master's thesis, Masdar Institute of Science and Technology, Abu Dhabi, UAE (2017)
19. Oprişa, C., Cabău, G., Pal, G.S.: Malware clustering using suffix trees. J. Comput. Virol. Hacking Tech. **12**, 1–10 (2016)
20. Ki, Y., Kim, E., Kim, H.K.: A novel approach to detect malware based on API call sequence analysis. Int. J. Distrib. Sens. Netw. (2015). Article ID 659101
21. Microsoft: MSDN: Learn to develop with Microsoft developer network (2018). https://msdn.microsoft.com/
22. Elzinga, C.H.: Sequence analysis: Metric representations of categorical time series. Department of Social Science Research Methods. Technical report, Vrije Universiteit Amsterdam, The Netherlands (2006)
23. Gabadinho, A., Ritschard, G., Müller, N.S., Studer, M.: Analyzing and visualizing state sequences in R with TraMineR. J. Stat. Softw. **40**, 1–37 (2011)
24. Needleman, S.B., Wunsch, C.D.: A general method applicable to the search for similarities in the amino acid sequence of two proteins. J. Mol. Biol. **48**, 443–453 (1970)
25. Gonnet, G.H., Scholl, R.: Scientific Computation, 1st edn. Cambridge University Press, New York (2009)
26. Itsik, P.: WPGMA (2001). http://www.cs.tau.ac.il/~rshamir/algmb/00/scribe00/html/lec08/node21.html
27. Maechler, M., Rousseeuw, P., Struyf, A., et al.: Methods for cluster analysis (2018). https://cran.r-project.org/web/packages/cluster/index.html
28. Kaufman, L., Rousseeuw, P.J.: Finding Groups in Data: An Introduction to Cluster Analysis. Wiley, New York (2005)