Malware Signature Generation using Locality Sensitive Hashing

Hassan Naderi[1], Vinod P.[2], Mauro Conti[2], Saeed Parsa[1], Mohammadhadi Alaeiyan[1]

[1]School of Computer Engineering, Iran University of Science and Technology, Narmak, 16844 Tehran, Iran
[2]Department of Mathematics, University of Padua, 35122 Padua, Italy

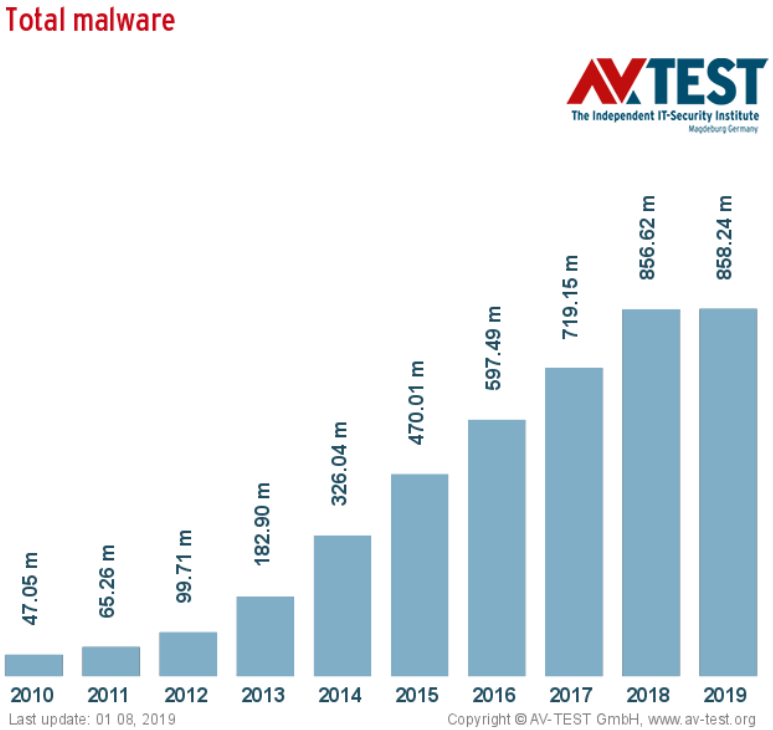
January 9, 2016

Introduction

Problem statement

- Malicious software has become a serious threat to computer systems.
- In 2018, the number of new malware increased by 19.1% more than the samples discovered in 2017*.

* https://www.av-test.org/en/statistics/malware/



Introduction

Problem statement

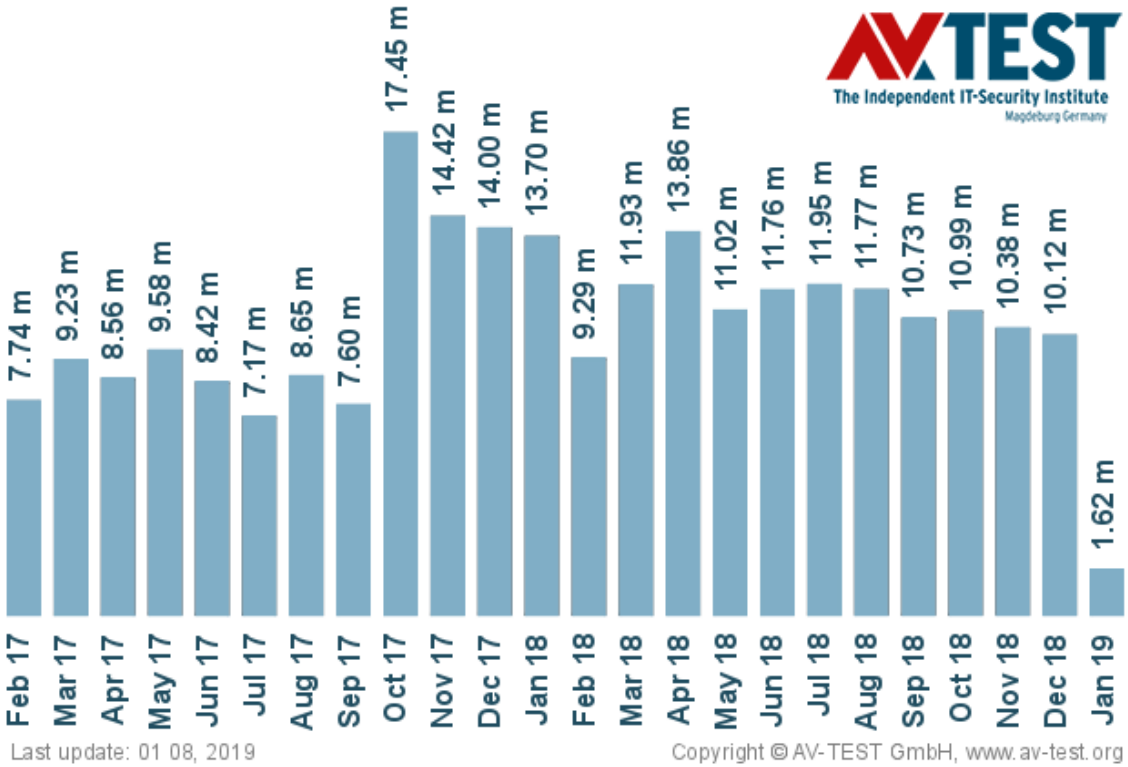


Figure: The number of new discovered malware since Feb. 2017.

Introduction

Main problem

Malicious users increase the number of malware by using various techniques to avoid the detection from anti-malware tools.

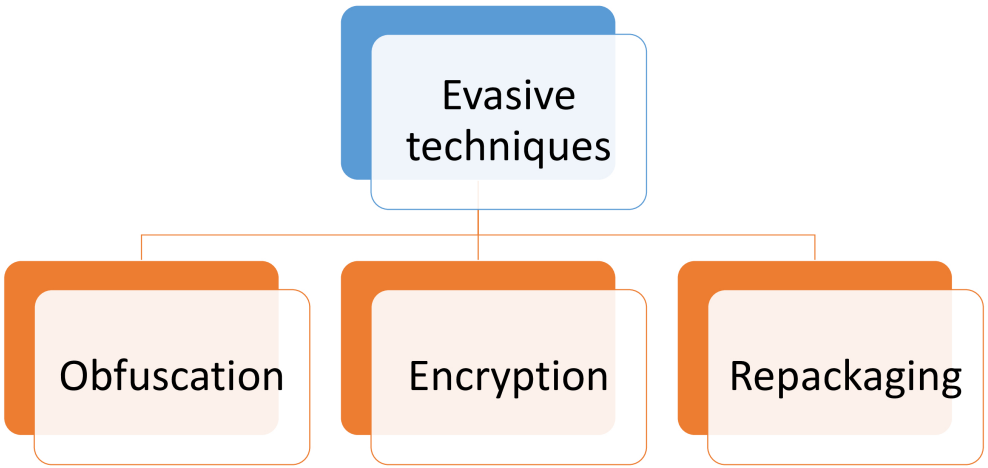


Figure: Popular evasive techniques

Introduction

The idea to reduce the number of signature

- The existence of a huge number of metamorphic and polymorphic malware collected by semi-automatic approaches are used to generate a large list of signatures.
- Then, we should reduce the number of signatures.
- Clustering is a way to determine malicious programs which are structurally similar and to decrease the number of signatures.
- However, the minimum time required for clustering n elements is $O(n^2)$.

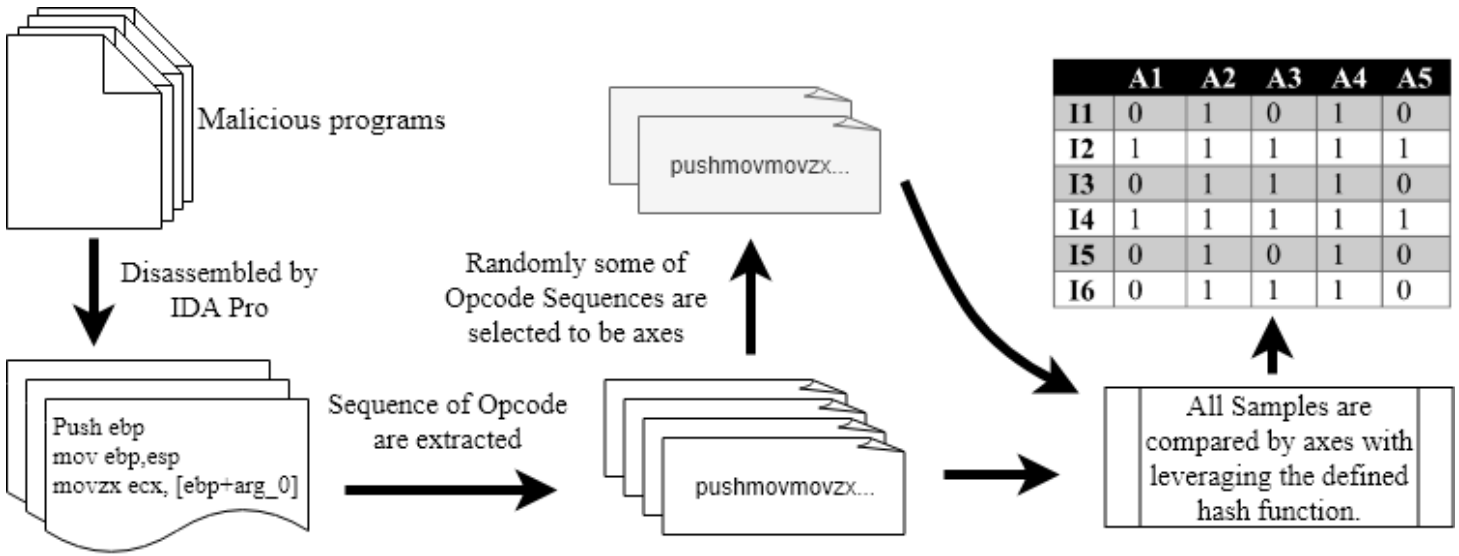
Solution

- We will leverage Locality-Sensitive Hashing (LSH) to identify similar elements with time complexity of $O(nA)$ where A is the number of axes and is less than n.
- We have proposed a new Locality-Sensitive Function (LSF) and provide a clustering methodology to provide significate signature for malicious code detection.

Steps of proposed methodology

- Provide the hash table.
- Cluster instances based upon the hash table.
- Generate signatures for each cluster.

Proposed methodology

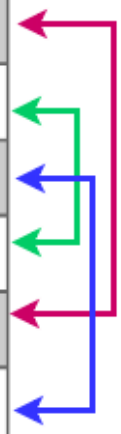


	A1	A2	A3	A4	A5
I1	0	1	0	1	0
I2	1	1	1	1	1
I3	0	1	1	1	0
I4	1	1	1	1	1
I5	0	1	0	1	0
I6	0	1	1	1	0

Programs are dissembled and their Opcode sequences are extracted. Some of Opcode sequences are selected as axes. All samples are compared with axes to provide the hash table. Equation 4 defines our proposed hash function.

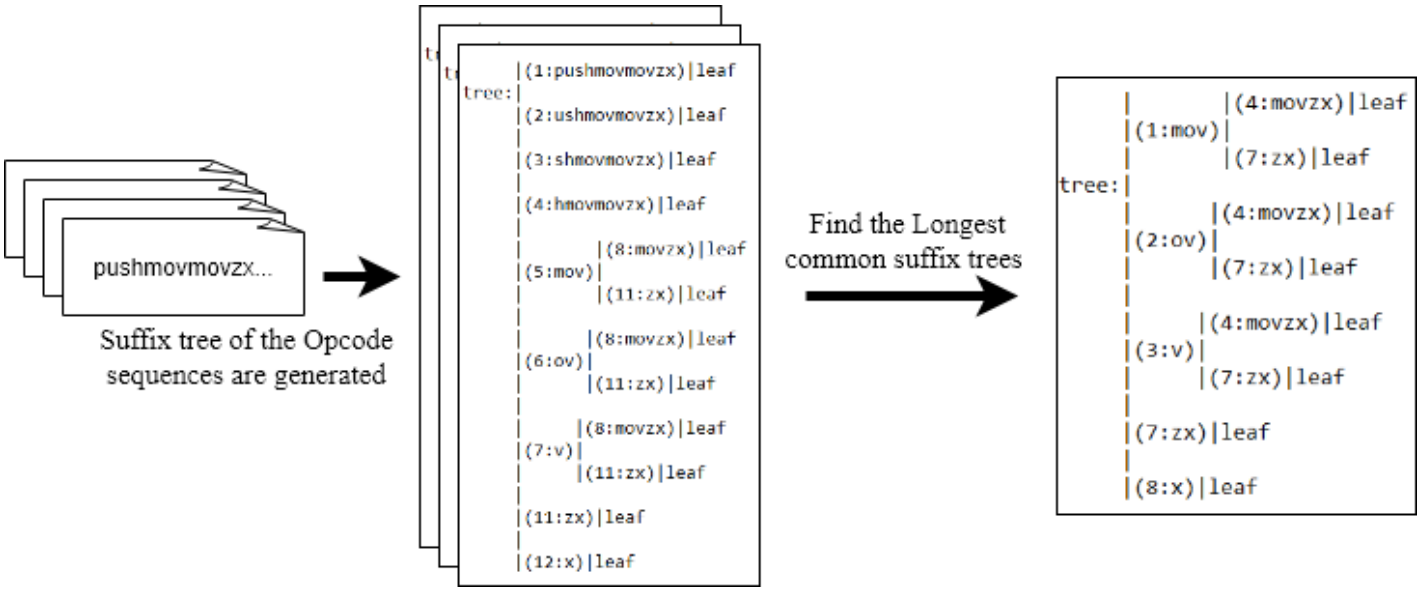
Proposed methodology

	A1	A2	A3	A4	A5
I1	0	1	0	1	0
I2	1	1	1	1	1
I3	0	1	1	1	0
I4	1	1	1	1	1
I5	0	1	0	1	0
I6	0	1	1	1	0



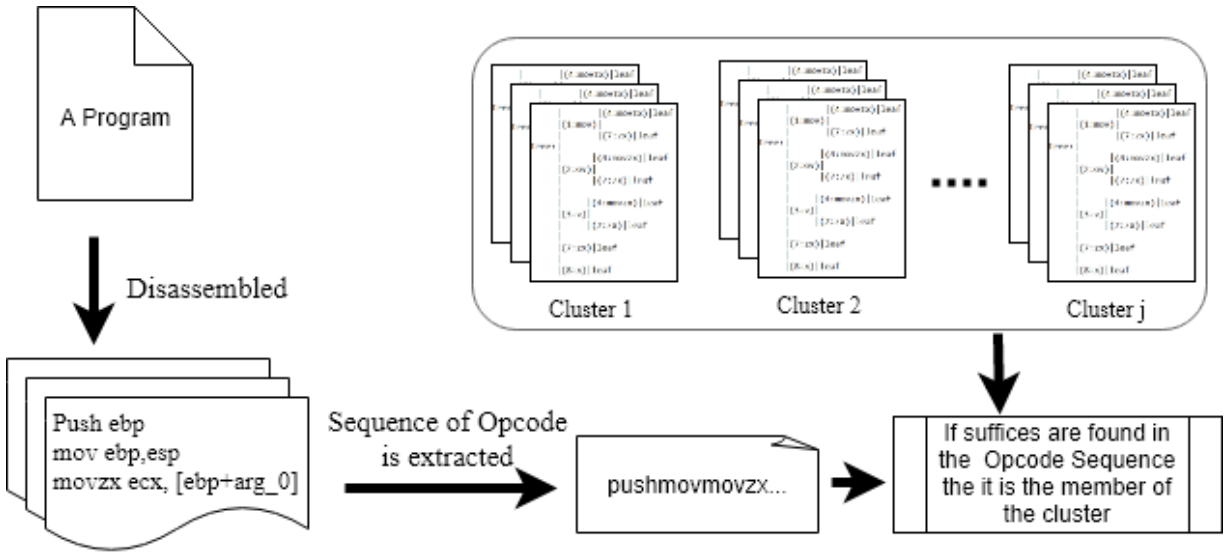
Rows determine the comparison results with axes. Those samples that have equal values are members of one cluster.

Proposed methodology



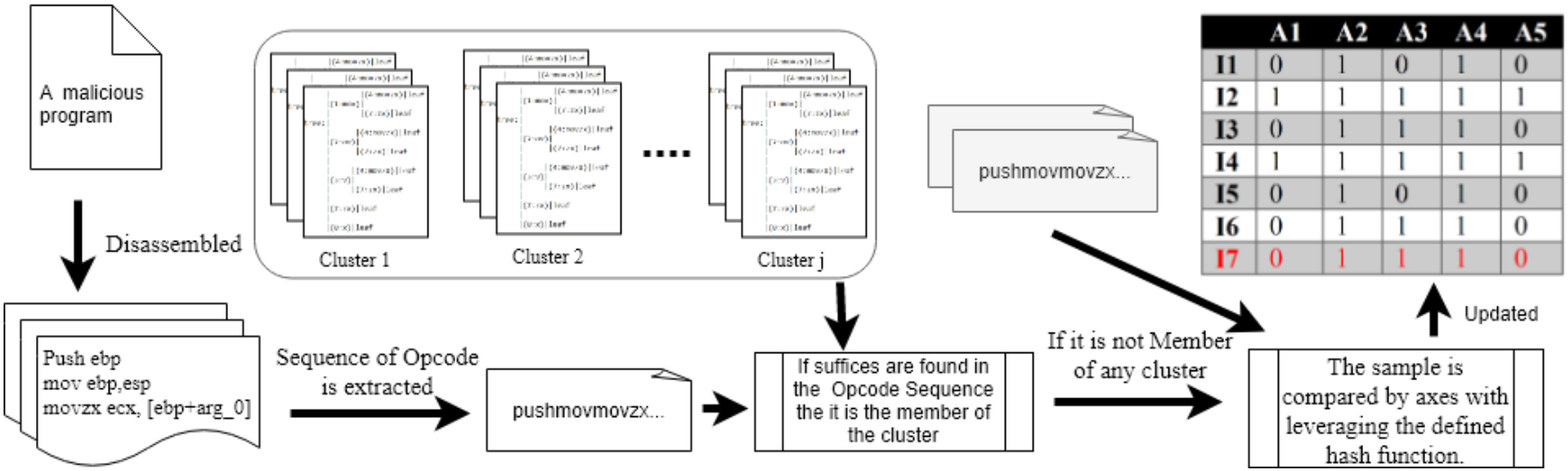
The suffix tree of Opcode sequences of a cluster are obtained. Then, the longest common sub-suffix trees are gathered.

Proposed detection methodology



The instance are disassembled. Then, its Opcode sequence are extracted. If a longest common Opcode sequence of clusters is found into the Opcode sequence of the given program, them the program is member of that cluster.

Proposed update methodology



If the instance is not detected as member of any cluster, then its comparison result with the axes would be inserted into the hash table. Thus, the suffix trees of the related cluster would be re-generate to extract longest common sub-suffix trees.

Locality-Sensitive Hashing (LSH)

- The general idea of LSH is the selection of a number of elements as axes to be compared with other elements.
- LSH has a Locality-Sensitive Function (LSF) which it defines a similarity hash function.
- Despite the popular hash function which a small change in elements causes large variations in the amount of created hash value, LSF should be insensitive to variations of similar elements.
- Those elements, having equal vectors are considered to belong to a cluster.

Locality-Sensitive Hashing (LSH)

Suppose S is a set of elements and d is a distance function. Therefore,

$$d : S \times S \rightarrow [0, \infty) \tag{1}$$

A LSH for a similarity d is a probability distribution over a set h of hash functions such that

$$P(h(X) = h(Y)) = similarity(X, Y) \tag{2}$$

for each $X, Y \in S$.

Locality-Sensitive Hashing (LSH)

Definition 1. A hash function $h : S \to H$ is a $(d_1, d_2, p_1, p_2)-$sensitive if H is an infinite set and $d_1 < d_2$. We have the following conditions for $X, Y \in S$.

1. If $d(X, Y) \leq d_1$, then $P(h(X) = h(Y)) \geq p_1$
2. If $d(X, Y) \geq d_2$, then $P(h(X) = h(Y)) \leq p_2$

Hash Function for Sequential Data

Our proposed LSF

Our proposed distance function is presented in Equation 3. X and Y are two sequences and LCS is the longest common subsequence of X and Y.

$$d(X,Y) = |X| + |Y| - 2|LCS(X,Y)|. \tag{3}$$

The characteristics of this distance function are:

a) Always $d(X,Y) \geq 0$.

b) Always we have symmetry, $d(X,Y) = d(Y,X)$.

c) There is triangle inequality, $d(X,Y) + d(Y,Z) \geq d(X,Z)$, in this function.

Hash Function for Sequential Data

Our proposed LSF

Equation 4 presents our similarity function.

$$similarity(X, Y) = \frac{2 * |LCS(X, Y)|}{|X| + |Y|} \tag{4}$$

Malware Clustering Algorithm

- We have leveraged the disassembled code of binary files to cluster malware.
- The dataset is a sequence of the disassembled instruction set for each malicious or legitimate program.
- We have used Suffix tree largest common sequence algorithm, having time complexity of order of $O(|X| + |Y|)$.

Evaluation

Implementation

- The proposed algorithm is written in C++.
- The system was a 20 core CPU Intel(R) Xeon(R) E5-2650 v3 @ 2.30GHz, and 48GB memory.

Evaluation

Dataset

- Malware samples aggregated from Virusshare.
  - VirusShare 00146.zip
  - VirusShare 00148.zip
- The instances are classified by Kaspersky internet security.
- 1700 legitimate programs, collected from PortableApps and popular and newly installed windows default applications

Name	The number of instance
Trojan horses	12152
Rootkits	79738
Backdoors	110231
Spywares	100017
Keyloggers	90172
Total	501684

Evaluation

Influence of A.

Table: The comparison to proposed method and represented method in [4]. A is the number of axes and T is similarity percentage. $T = 20$.

	$A = 10^4$		$A = 2 * 10^4$		$A = 3 * 10^4$	
Evaluation Metrics	our method	prior approach [4]	our method	prior approach [4]	our method	prior approach [4]
Accuracy	0.9094	0.8991	0.9172	0.9176	0.9559	0.9359
F-Measure	0.9351	0.9304	0.9366	0.9423	0.9656	0.9580
Precision	0.9559	0.9576	0.9563	0.9776	0.9735	0.979
Recall	0.9152	0.9047	0.9176	0.9095	0.9578	0.9374
#clusters	3564	3754	3863	4023	4387	4543
#signatures	40549	49938	36493	47394	34239	46903

Evaluation

Influence of T.

Table: Effect of T on the number of signature and clusters. $A = 3 * 10^4$.

	$T = 10$	$T = 20$	$T = 30$	$T = 40$
The number of clusters	4075	4387	4836	4914
The number of signatures	37253	34239	36365	36821

Evaluation

Influence of the number of instances on Time of clustering.

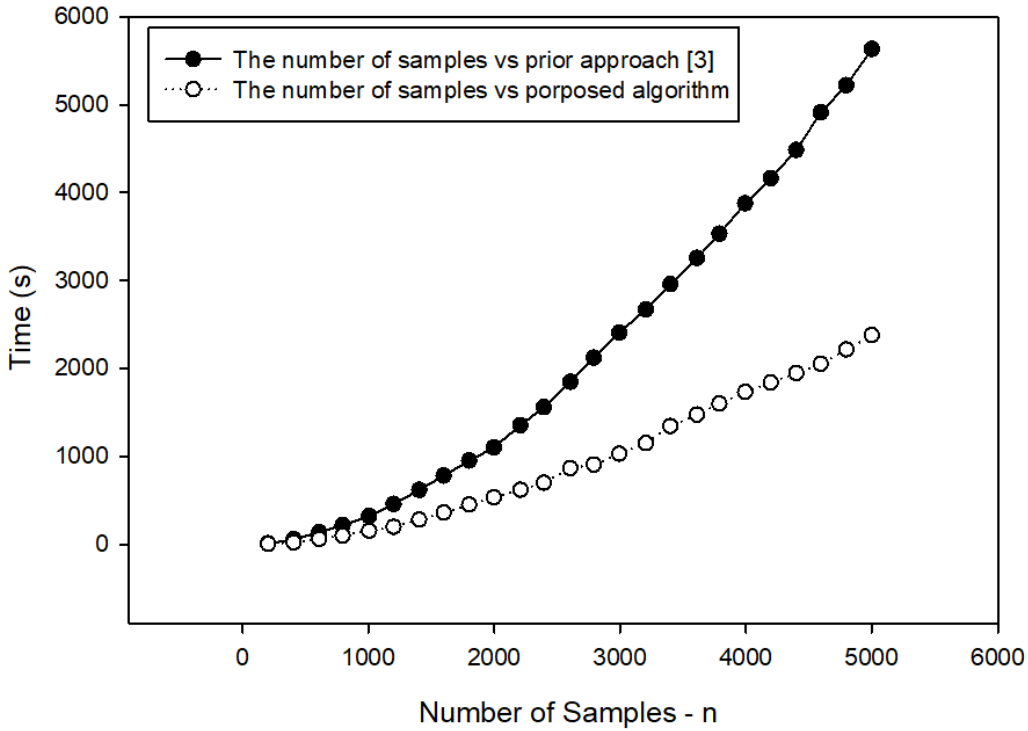


Figure: Clustering time of Suffix tree clustering algorithm [3] vs proposed clustering algorithm.

Future works

- We present a hash function with higher accuracy than previous methods to cluster malware.
- Unlike the previous works that their features have a fixed-length, our hash function is based on dynamic-length features.
- the achieved signature detected the dataset with an accuracy of 95.59% by saving 30% in the number of signatures.

Future works

- Prototyping the proposed approach for deploying in a real-world online malware detection by reducing the number of signatures.

Reference

1. Miao, Q., Liu, J., Cao, Y., Song, J.: Malware detection using bilayer behavior abstraction and improved one-class support vector machines. International Journal of Information Security 15(4), 361–379 (2016)
2. Newsome, J., Karp, B., Song, D.: Polygraph: Automatically generating signatures for polymorphic worms. In: Security and privacy, 2005 IEEE symposium on. pp. 226–241. IEEE (2005)
3. Oprisa, C., Cabau, G., Pal, G.S.: Malware clustering using suffix trees. Journal of Computer Virology and Hacking Techniques 12(1), 1–10 (2016)
4. Oprisa, C., Checiches, M., Nandrean, A.: Locality-sensitive hashing optimizations for fast malware clustering. In: Intelligent Computer Communication and Processing (ICCP), 2014 IEEE International Conference on. pp. 97–104. IEEE (2014)
5. Sharma, S., Krishna, C.R., Sahay, S.K.: Detection of advanced malware by machine learning techniques. In: Soft Computing: Theories and Applications, pp. 333–342. Springer (2019)
6. Wang, T., Xu, N.: Malware variants detection based on opcode image recognition in small training set. In: Cloud Computing and Big Data Analysis (ICCCBDA), 2017 IEEE 2nd International Conference on. pp. 328–332. IEEE (2017)
7. Zhang, J., Qin, Z., Zhang, K., Yin, H., Zou, J.: Dalvik opcode graph based android malware variants detection using global topology features. IEEE Access (2018)