IDENTIFY THE SAME FAMILY OF MALWARE BY USING PATTERN MATCHING TECHNIQUE

[1]BEWAR NEAMATTAHA, [2]SUBHI R. M. ZEEBAREE

[1]Department of Software Engineering, Firat University, Elazig, Turkey
[2]Department of Information Technology, Duhok Polytechnic University, Duhok, Iraq
E-mail: [1]Bewar_nemat@outlook.com, [2]Subhi.rafeeq@dpu.edu.krd

Abstract - The evolution of computing technology in the past decade until now has been raised threats against users, especially in the malware form, because the most cybersecurity threat in now day's world is malware applications, and the new malware is introduced every day. But most of that malware is not created exactly from scratch. This paper analyzes the Methods for matching the string to identify the same family of malware, through researching and comparing the effectiveness of five algorithms of pattern matching which are Naïve, Rabin-Karp, Brute-Force, Knuth-Morris-Pratt, and Boyer Moore Algorithms, while after searching these algorithms proven effective in detecting chain similarity between malware.

Keywords - Malware Analysis, Static and Dynamic Analysis, String Matching Algorithms, Naïve- algorithm, Rabin-Karp Algorithm, Brute-Force Algorithm, Knuth-Morris-Pratt Algorithm, Boyer Moore Algorithm, Similarity string.

I. INTRODUCTION

Malware is the summary expression of all various kinds of malignant software that are designed and installed to your device to implement the strange mission predominating for another extremity service [1].
There are many different types of malware which can disrupt our devices, like virus, spyware, adware, worm, Trojan, and etc. [2,3], that can be pilfering the passwords and delete our data and files from our devices or even to stopping the device at all from working [3].
Malware making some programs without our knowledge to transfer information to an unwanted entity [3].
Indeed same families of malware are created easily to make money often through stealing confidential information from the computer [3].

II. LITERATURE REVIEW

Regarding the history of the Malware one form of such threats against users since the creation of the first malware in 1970, in recent decades malware targeted primarily PC users but mobile OS are becoming increasingly common, the target of malware has also turned towards users of the mobile OS, recently Private corporate networks has been the main target of malware, possible next malware attack may target cloud computing network [4]. As Identification the threats of the same family of malware is great importance to establish fixture rule or formulas, because the same rule-based solution for malware attack can be applied within Malware of the same family, the unified effort between software companies and academic researchers, to devise analytical methods that identify and help resolve these threats, by applying string matching algorithms with appropriate threshold values [4].

III. PROBLEM DEFINITION

New malware is introduced every day. But most of the malware is not created exactly from scratch [4]. Malware is one of the fundamental problems in computer security from the past decades until now [5]. Malware can destroy our privacy and is a threat to the personal information making our files incorrect and slows our device through exploiting impairment of the computer systems [6]. In the first quarter of 2017 alone, there were listed 48 million unique malware samples, an increase of 7% percent compared to the previous year [7].

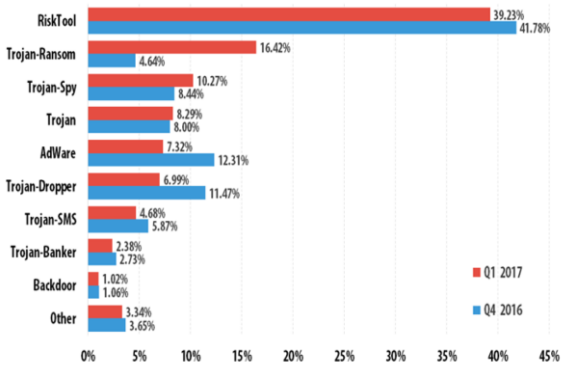


Figure 1: IT threat evolution from Statistic & Secure list 2016-2017.

IV. SIGNIFICANCE OF THE PROBLEM

Identifying the same family of malware is vital because malware is threatening computer users all over the world [4], especially as threats become more dynamic and diverse in a world where everything is connected [6]. Malware developers constantly innovate to increase return investment from their exploits. This means that most threats are carefully designed to evade detection and change quickly so that they can be preserved [6].

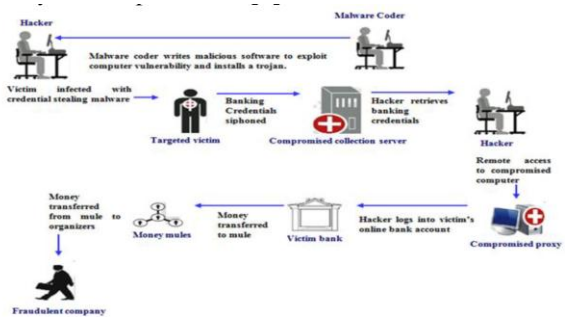


Figure 2: Malware Fraud Process

V. STATIC AND DYNAMIC ANALYSIS

Static and dynamic analyses are two of the most common types of code security checks.

1. STATIC ANALYSIS

Static and dynamic analyses are two of the most common types of code security checks. The static analyses are performed in a non-runtime environment. The static analysis tool will examine the program code for all behaviors possible runtime and look for cryptographic defects, background addresses and malicious code [8]. Sometimes called static "projection or estimate" which is simplified analysis the effect of the instant change is calculated on the system without regard to the system response long-term to that change. If the short-term effect is then extrapolated to the long term, such extrapolation is unsuitable [9]. Each type of static analysis contains something in common. Searching for similarities and differences in the structure of malware, while the similarity algorithms must be tested for reflecting their activity and efficiency [4] [10].

2. DYNAMIC ANALYSIS

The dynamic analysis relies on the opposite approach which is executed while running the malware program. The dynamic test will be monitoring system memory, functional behavior, response time and the general performance of the system. However, this method is 3 not entirely different from the malignant way a third party that may interact with an application [8].On the other hand, dynamic analyses are able to detect hidden bugs or very complex weaknesses while static analysis can't detect. Also, it can be the most effective way to test and evaluate the program by implementing data in real time. However, the dynamic test will only find the faults from the part of the code which is already being implemented [8]. Table 1 shows the difference between dynamic and static analysis of malware.

Table 1: Static vs. Dynamic Analysis

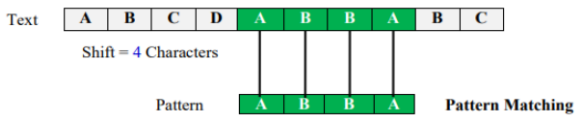
	Properties and Feature	Static	Dynamic
1	Need for a controlled environment	No	Yes
2	Works on running malware	No	Yes
3	Suitable for	Identifying malware	Malware result monitoring
4	Results	Detailed	Simplest

VI. THE MEANING OF STRING MATCHING ALGORITHMS

Some-times called string searching algorithm or pattern is aimed at finding where one string or several strings are created within a string or larger text. Conformity is usually divided into two sub-problems which are to find approximate sub-string matches within a particular string and find dictionary strings that match the style almost [11]. The way the string is formulated encoded can affect the string search algorithms. In particular, if the variable display encoding is in use, it may be slower to find the character "n", this may slow down some search algorithms significantly [12]. However one of the many possible solutions is to search for the sequence of code units, but doing so may result in erroneous matches unless the encoding is specifically designed to avoid it [12].

Text: is the group of character (T.length).
Pattern: is a substring search (P.length).

Table 2: Showing the pattern matching technique



Length of pattern sub-string should be< = Text string, otherwise, the "pattern" is not found.

VII. STRING MATCHING ALGORITHMS USED FOR MALWARE DETECTION

1. NAÏVE SUBSTRING SEARCHING ALGORITHM

The Naïve string matching algorithm slides the pattern one by one from starting from the leftmost corner. It needs to know the length of the text string and pattern substring. Then after all slide, one by one character is checked at the end of the current change and if all characters match, print the matching between text and pattern [13].

However, it should only look at one or two letters for each wrong situation to find out that it is a wrong placement.

From the average cases it will take O (n + m) while from the worst case it will take O (nm) [14]. Finding the whole valid shifts through using the loop that checks the condition P [1 . . . m] = T[s + 1 . . . s + m] for each of the n - m + 1 possible values of s [14].

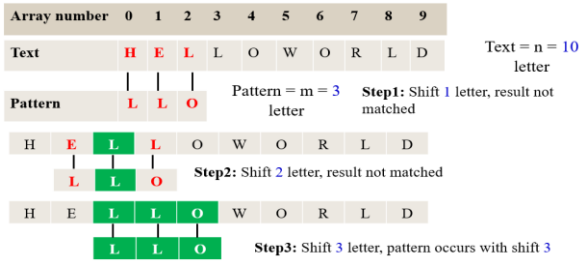
n ← length [T]
m ← length [P]
for s ← 0 to n – m
do if P [1...m] = T[s + 1..s +m]
Then print Pattern occurs with Shift s

If the Text "T" ≥ Pattern "P", which is T > = P, or Pattern "P" ≤ Text "T" which is P < = T. Pattern will be founded!

While if N ≥ M, while if M > N. Pattern not found!

Example:

Table 3: Founding Pattern Matching at index 2



2. RABIN-KARP SUBSTRING SEARCHING ALGORITHM

This algorithm proposed a string matching algorithm that performs their function well in practice which has been circulated from another algorithm for related problems, such as the matching of two-dimensional pattern. This basic algorithm uses numerical concepts such as a two-digit formula for a third digit unit [11].

However, this algorithm used quite differently to solve string matching problems. This method relies on fragmentation techniques. It needs to calculate of hash function h(x) for the pattern P[0...m-1] and then needs to look into matching through using the hash function itself for the length of every substring [m - 1] for the text [15].

It can efficiently generate a hash function or hash value through rulings hash function to calculate and recalculate hash [15].

X = value "Old hash" – value "Old character".
X = X/Prime when Prime = any number.
X = new hash = X + Prime ^ m-1.

From this situation, if it has only one pattern string to be compared with one text string, it needs to perform those operations.

hash (P) = O (m)
hash (T) = O (n – m + 1)

The worst-case running time of the Rabin-Karp algorithm is O (m (n - m + 1))

Example:

Table 4: Founding Pattern Matching at index 2

Pattern	L	L	O	H(LLO) = 61

Prime value = 3 For keeping calculation simple.

Array number "Index"	0	1	2	3	4	5	6	7	8	9
Text	H	E	L	L	O	W	O	R	L	D
Value of Each Character	3	2	4	4	5	7	5	6	4	1

H(HEL) = 3 + 2 * 3^1 + 4 * 3^2 = 45
H(ELL) = (45 – 3)/3 = 14 + 4 * 3^2 = 50
H(LLO) = (50 – 2)/3 = 16 + 5 * 3^2 = 61
H(LOW) = (61 – 4)/3 = 19 + 7 * 3^2 = 82
H(OWO) = (82 – 4)/3 = 26 + 5 * 3^2 = 71
H(WOR) = (71 – 5)/3 = 22 + 6 * 3^2 = 76
H(ORL) = (76 – 7)/3 = 23 + 4 * 3^2 = 68
H(RLD) = (68 – 5)/3 = 21 + 1 * 3^2 = 30

When
D = 1
E = 2
H = 3
L = 4
O = 5
R = 6
W = 7

3. BRUTE-FORCE SUBSTRING SEARCHING ALGORITHM

Is a very easy and appropriate technique to find a solution to general problems if found, such as NP (non-deterministic polynomial) problems. They are usually used when the size of the problem is limited because the tendency to grow rapidly with volume in many problems [16].

This algorithm is used to find the line of normal numbers [n] through multiplying and enumerating all integers from [1 to n] with taking each device [16].

Input: is the array of "Text" string T [0...n-1] and array of "Pattern" string P [0...m-1].

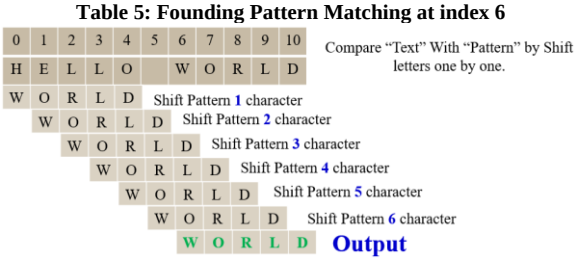
Output:

1. Position of the pattern string in the text string.
2. If the search is unsuccessful.

For i ←0 to n-m do
J ←0
While j < m and p[j] = T [i + j] do
j ←j + 1
If j = m return i
Return -1

Example:

Table 5: Founding Pattern Matching at index 6



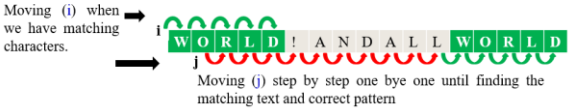
4. KNUTH-MORRIS-PRATT SUBSTRING SEARCHING ALGORITHM

The main drawback of this algorithm is that each time when a match between the pattern and a shift in the text fails. This algorithm uses the information provided through a particular table, obtained by pre-processing the pattern to avoid re-examining [15]. This algorithm is a more accurate linear time algorithm O (n + m) and is composed of two parts. The searching part consists of finding the correcttransitions in the text, where the complexity of time O (N), obtained by comparing the pattern and transitions of the text, and a pre-processing part which consists to preprocesses the pattern [15]. The goal of the pattern pre-processing is to get a table that

gives the following mode in the pattern to be processed after a mismatch. For P [0 ... m - 1], the table will give the result of pre-processing for each letter j [17].
Example:

Table 6: Founding Pattern Matching at index 12

Array of the first "index"	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Text	W	O	R	L	D	!	A	N	D	A	L	L	W	O	R	L	D
Array of Next "index"	0	0	0	0	0	0	0	0	0	0	0	0	1	2	3	4	5



5. BOYER MOORE SUBSTRING SEARCHING ALGORITHM [T, P, ∑]

This algorithm is a highly efficient string matching algorithm, a standard for a series of searches or matching of the literature. The algorithm performs pre-treatment of the searched string [15].
This algorithm will use the information that will be collected during the previous step to work and to skip the next part. This algorithm works faster along with the form. The main features of this algorithm are to search or match the tail style rather than the headers, as well as to navigate across the text with multiple jumps instead of searching for a full character within the text [15]. This algorithm needs to create a "Bad Match table". Any remaining character from the last character equal to the length value. Otherwise, if it is not already defined we can compare the pattern in the text by starting from the rightmost character in the pattern. When we see that the mismatch occurs the pattern is moved to the right-hand side of the value in the same table [18].
Create a "Bad Match Table".
Value = Length of Pattern – index – 1
Example:

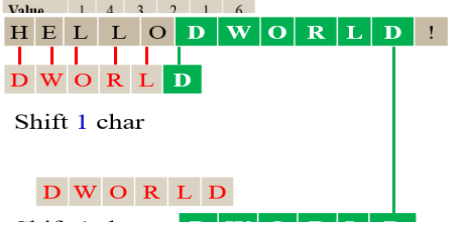
Table 7: Founding Pattern Matching at index 10

Array number = length.T	0	1	2	3	4	5	6	7	8	9	10	11	T.Length = 12
Text	H	E	L	L	O	D	W	O	R	L	D	!	
Array number = length.P	0	1	2	3	4	5	P.length = 6						
Pattern	D	W	O	R	L	D							

The max number between 0-1
Value = Max(1, length of Pattern – index of Actual Character – 1)

Bad Match Table

Letters	D	W	O	R	L	*
Value	1	4	3	2	1	6



Accordingly, it will update the value of actual character D from 5 to 1, because the character D was duplicated again.

CONCLUSION

Malware is an expression of all different types of malware that are designed and installed from a second person device to perform a strange task. Threatening to steal sensitive confidential information to serve the attackers or another task. For that many researchers presented an integrated method used both dynamic and static feature to detect malware. Signature-Based detection methods are one of the major methods used for malware detection.
By using the pattern matching technique we can identify the same family of malware. The main advantage of this technique have performance characteristics are remarkably different. So it needs to think about using smarter and better algorithms for finding substring from malware families.

REFERENCES

[1] Yu.Sh, Zhou.Sh, Liu.L, Yang.R and Luo.J (2010), Malware Variants Identification Based on Byte Frequency, 2nd NSWCTC and IEEE, retrieved from http://ieeexplore.ieee.org/document/5480417/ [1 March 2018]
[2] Kaur.J and Sharma.S, (2015), Study of Malware Based on Pattern Matching Techniques, IJEDR, Volume 3, Issue 2, ISSN: 2321-9939, retrieved from https://www.ijedr.org/papers/IJEDR1502194.pdf [1 March 2018]
[3] Liu.P and Wang.X, (2018), Inductive Learning in Malware Detection, IEEE, retrieved from http://ieeexplore.ieee.org/document/4681110/ [1 March 2018]
[4] Mastjik, F., Varol, C., and Varol, A. Comparison of Pattern Matching Techniques on Identification of Same Family Malware. IJISS, Vol.4, No.3. Available at: http://dergipark.gov.tr/download/article-file/147950 [Accessed 9 May. 2018].
[5] Pitolli, G., Aniello, L., Laurenza, G., Querzoni, L., and Baldoni, R. (2017). Malware family identification with BIRCH clustering. Madrid. ICCST and IEEE. Available at: https://ieeexplore.ieee.org/abstract/document/8167802/?reload=true [Accessed 9 May. 2018].
[6] Seideman, J., D., Khan, B., and Brahim, G., B. (2013). Determining vulnerability resolution time by examining malware proliferation rates.9th. Sardinia.IWCMC and IEEE. Available at: https://ieeexplore.ieee.org/document/6583808/ [Accessed 9 May. 2018].
[7] Security report. (2016-17). the AV-TEST Security Report. Magdeburg. Available at: https://www.av-test.org/fileadmin/pdf/security_report/AVTEST_Security_Report_2016-2017.pdf [Accessed 9 May. 2018].
[8] DePaul, N. (2013). Static Testing vs. Dynamic Testing.[Online]. Available at: https://www.veracode.com/blog/2013/12/static-testing-vs-dynamic-testing [Accessed 9 May. 2018].
[9] Static Analysis. (2018). [online]. Available at: https://en.wikipedia.org/wiki/Static_analysis [Accessed 9 May. 2018].
[10] Rouse, M. (2006). Static code analysis.[Online]. Available at: https://searchwindevelopment.techtarget.com/definition/static-analysis [Accessed 9 May. 2018].
[11] Pokiya, A. (2015). String matching algorithms. [Online slide share]. Available at: https://www.slideshare.net/Ashikapokiya12345/string-matching-algorithms52582907?next_slideshow=1 [Accessed 9 May. 2018].
[12] Choudhury, A. (2015). String matching algorithms. [Online slide share]. Available at: https://www.slideshare.net/alokeparnachoudhury/string-matching-algorithm [Accessed 9 May. 2018].

[13] Singh, P. (2013). Naive string matching algorithm.[Online]. Available at: https://www.youtube.com/watch?v=RhNM6jvvNjU [Accessed 9 May. 2018].

[14] Shakeel, Y. (2016). Naive pattern searching.[Online]. Available at: https://www.youtube.com/watch?v=xP5Ox-df_ik [Accessed 9 May. 2018].

[15] Gou, M. (2014). Algorithm for string matching.[Online]. Available at: http://www.student.montefiore.ulg.ac.be/~s091678/files/OHJ2906_Project.pdf [Accessed 9 May. 2018].

[16] Holczer, B. (2017). Brute-Force substring search algorithm. [Online]. Available at: https://www.youtube.com/watch?v=vtnpzDPgaU0 [Accessed 9 May 2018].

[17] Roy, T. (2015). Knuth-Morris-Pratt (KMP) Pattern Matching (substring search). [Online]. Available at: https://www.youtube.com/watch?v=GTJr8OvyEVQ [Accessed 9 May. 2018].

[18] Slade, M. (2014). Boyer Moore Horspool Algorithm.[Online]. Available at: https://www.youtube.com/watch?v=PHXAOKQk2dw [Accessed 9 May. 2018].

★★★