Pattern Matching Based Malware Identification

Bewar Neamat Taha
Department of Software Engineering
Firat University
Elazig, Turkey
Bewar_nemat@outlook.com

Cihan Varol
Department of Computer Science
Sam Houston State University
Huntsville, TX, USA
cvarol@shsu.edu

***Abstract*— The evolution of computing technology over the past decade has created threats for its users, especially in the form of malware. This is because most cybersecurity threats are now malware applications. In addition, new malware is being introduced every day. However, most malware is not created from scratch. As such, this research discusses methods of matching strings to identify families of malware. Application programming interface calls were researched and compared using the following five pattern matching algorithms: Naïve, Rabin-Karp, Brute-Force, Knuth-Morris-Pratt and Boyer Moore. In this research, the chosen algorithms proved effective in detecting chain similarities between malware applications.**

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

I. Introduction

The first form of malware was created in 1970. Over the last several decades, malware has primarily targeted computer operating systems. However, malware that targets mobile operating systems has become increasingly common. Recently, private corporate networks have been the main target of malware and it has been suggested that the next major malware attack may target the cloud computing network [1].

The term malware is an expression used to indicate various kinds of malignant software. This software is designed and installed into devices to implement the strange mission predominating for another extremity service [2]. Different forms and types of malware, such as viruses, spyware, adware, worms, and trojans are classified into families that disrupt our devices [3, 4]. Malware can obtain passwords and delete data or files from devices. In addition, they can prevent devices from working altogether Malware can make programs transfer information to an unwanted entity without the knowledge of the user. In the past, malware was used for fun. However, it is now often created to obtain money by stealing confidential information such as bank account credentials from computers [4]. Malware can be used to cipher information or hacktivism. It can access devices if users download infected files, visit infected websites or receive emails containing infected attachments or links. Each of these methods can be differentiated through various characteristics [4, 5].

Identifying the threats presented by malware families is important as this can indicate fixture rules or formulas. This is because the same rule-based solution for malware attacks can be applied to other malware within the same family. Efforts have been made by software companies and academic researchers alike to devise analytical methods of identifying and resolving the threat of malware. One method is to apply string matching algorithms with appropriate threshold values [1]. As such, identifying and determining malware families can be considered a fundamental issue for general users and computer security firms [5].

Because of the significance of identifying malware families, we perform research on improving the framework of identifying malware families using pattern matching techniques which is done through using an application within five pattern matching algorithms. Specifically, we evaluated the performance of known string matching algorithms to identifying the same family of malware, we approved that string matching algorithms can be used to differentiate malware of the same family members from distinct programs.

II. Background Study

According to Liu et al. [4], From the Malware software we can obtain the mischievous intention of attackers such as the Viruses, Trojan, worms, adware, spyware and Ransomware also with malignant application that are publicize quickly through the internet and it will be hand out by email or suspicious websites, these attacks had reason millions of dollar loss from company, government and services, for that the user want to renewing and updating their Software and anti-virus for overcome with new types of viruses and also the detection tools specially when we are facing strange and malware program, in this document the researcher had been insert the technique for automatically derive specifications of malignant in the sample malware, while some of those malware can be used through the malware detections. However they was suggest an algorithm for detecting the malware and the mode to popularization and allotment of the attack style through using the inductive learning which is suggested, that can be applied redeveloping and dilating the familiarity of database, so this process has capacity for detecting the obscure malware.

The author in [6], introduced into the new and notion path for detecting the malware codes establish from the various kinds of computer files that are using vital information tools, meaning the real of the shortest teach of the align into the next obstetrics arrangements, that are using approximate string

matching's, however one of the advantages of this process is the real perform from that path which doesn't needs loading the entire files from the memory, instead, loading prevent of the files rely from the physical memory of the personage matching.

The author in [3], the signature based of pattern matching techniques is the most popular and extreme for finding and detecting malware, while this technique has one drawbacks that cannot detecting and finding the new viruses or the new families of malware, while using the different kinds of pattern matching techniques for detecting the new malware from the program such KNN algorithm for classification, anomaly based and also emulation based signature based.

The author in [7], has been focused into the new effective characteristics for detecting malware techniques, several operation of the technique which is established from the judgment mining while some other different are established self-reproduce characteristic of origin techniques that will participate concept from the malware detecting techniques scope through produce an optimize mode for malware detecting, furthermore the financial institutions must be identifying the increasing dangers from the whole inside and outside sources then taking the functional measure for detecting the potential malware and intervention with the commercial operations.

The author in [8], presented an integrated way that have been used both dynamic and static feature for malware detecting, they have proven this thesis for combining dynamic and static feature that will be increase the detecting reliability alone of dynamic method and static method, while the outcomes classification shows that it's clear that the dynamic method analyzing is superior and bestead than the based codes of the static methods, however the dynamic method has extreme reliability than the static methods, and also it is clear that the inserted path raise the detecting reliability.

The author in [2], had been suggest and used the novel of "BFBDM" for detecting and identification the malware with his rate and also this novel has the capacity that can be using for detecting the variants of malware also can be using for nearness among of some particular files because the researcher had thinks that the result of this experience will be active for recognizing malware, then reality result of this experience offering that the result is very operative for selecting and identification the threat and the malware variants.

The author in [9], has been clarified the malware detecting method established from the side of API calling that can be limited and determine the duty and behavior of program, so the researcher suggest to using the algorithm that can opt the singular and special APIs calls then after that had been used education machine for assorting malignant and unharmed PE files, and they had used WEKA Tools Algorithm because our algorithm mostly will be used for ranking and for familiarity analyses and this algorithm established on java and it is restrain many vibes for analyzing such as ranking, ideation and assembly rules with using another algorithm of WEKA Software for testing this experience, from the final of this identification experience the researcher acquired estimated result that the SVM result is better than all of those gauge.

The author in [10], had been using the way for malware detecting which is established from the analyzing and drip of the rep and systematic characterization from the suspicious behaviors, which is specified through the concatenation of API's calling and it is called under the Windows environment then carry out the technique way for detecting the threat from the malignant binary practicable, the way of researcher from this research document is the Bayes algorithm technique which is using for detecting flowing of the suspected behavior by analyzing the API's task which is remember through the malignant and from the result of the document the researcher suggest that this algorithm can be used for detecting the malware and threads virus from wind32, while our while our technique was been used from the prototype system which is called RADUX .

The author in [11], approved that the SIEM is very powerful way for resolves networks and internet with the logs of security detecting and identification the stomachic hosts and also this way is very good for resolves the logs of proxy through suitable with the base of HTTP malignant list which is very powerful and actives for detecting the infected hosts, however, the researcher suggested to using the detecting methods which is resolves and analyzing the firewall as well as proxy logs because the researcher think that this method can contribute this way for improving the thoroughness which is compared with sole Proxy-based detection and this methods of detecting is infected-host which is using from both of the based-of protocols such as TCP/IP-based and HHTP-based malignant, while the whole of the malignant list will be created through analyzing effectives of threat and also logs of traffic networks, in finally the appraisal result of this research is that this method is capable for detecting malware infected host and also the opinion of researcher has been true because this method had been contributes 6% improving the thoroughness which is compared with sole Proxy-based detection and it's multiple layer which is more effective for improving the ability of malware detecting.

III. Defining the problem

One of the biggest problems faced from using the same family of malware is that viruses can infect computers without consent. This is done through opening a corrupt program, visiting an infected site, through software sharing or by forwarding attachments and files to computers and other devices. Another major problem is the large amount of new malware released daily. Most malware is not created from scratch, but the manual effort required to identify the influx of new malware and its byproducts is unprecedented [1]. Due to the complication of novel malware plant, as well as advanced mechanisms that use disruption codes to strengthen malignant specimens, malware is more likely to go undetected [12].

Malware has become a fundamental problem in computer security over the past decade [12]. When users open corrupt files or visit corrupt sites, these software programs infect devices without consent and install bad files to steal our

personal information and delete files. Malware is a threat to our personal information and can destroy our privacy. It making our files incorrect and slows our device through exploiting impairment from the computer system [13, 14]. In the first quarter of 2017, there were 48 million unique malware samples listed. This is an increase of 7% compared to the previous year [15].

IV. SIGNIFICANCE OF THE PROBLEM

The ability to identify malware families is vital, as is recognizing which family a malware application belongs to. Many kinds of malware can infect devices without consent when users open corrupt programs and visit corrupt sites. Other issues related to the complication of novel malware plant, as well as advanced mechanisms that use disruption codes to strengthen malignant specimens, go undetected. Recently, the numbers of threatening malware applications have increased [15].

Phishing attacks intended to obtain information are one of the most common security challenges faced by individuals and businesses. Irrespective of whether the phishing attack is intended to access passwords, credit cards or other information, attackers can use mobile phones, social media, email and other forms of telecommunication to steal data. Many developers create malware that aims to steal money and threaten targeted businesses [4]. Therefore, identifying the family of malware is vital because malware is a threat for users around the world and can leaking out every time [1, 13].

V. STATIC AND DYNAMIC ANALYSIS

Static and dynamic analyses are two of the most common methods to check the security of program codes.

A. STATIC ANALYSIS

Static analyses are performed in non-runtime environments. A static analysis tool will examine the program code for all behaviors possible during runtime. It aims to identify cryptographic defects, background addresses, and malicious code [16]. A static analysis is sometimes called a static "projection or estimate." Essentially, it is a simplified analysis.

The effect of the instant change is calculated by the system without considering the long-term system response. If the short-term effect is then extrapolated to the long term, such extrapolation is unsuitable [17]. A static analysis identifies similarities and differences in malware structures. Similarly algorithms must then be tested to determine their activity and efficiency [1, 18].

B. DYNAMIC ANALYSIS

A dynamic analysis relies on the opposite approach and is executed while running the malware program. A dynamic test will monitor system memory, functional behavior, response time, and the general system performance. However, this method is not entirely different from the malignant way a third party may interact with an application [16].

On the other hand, dynamic analyses can detect hidden bugs or very complex weaknesses while static analysis cannot. In addition, a dynamic analysis can be the most effective way to test and evaluate the program as it implements data in real-time. However, a dynamic test will only detect faults from the section of the code being implemented [16]. Table I below shows the differences between static and dynamic malware 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

Table I: Static vs. Dynamic Analysis

VI. STRING MATCHING ALGORITHMS

String searching algorithms or patterns aim to discover where strings, or several strings, are created within a larger string or text. Conformity is usually divided into two sub-problems: finding approximate sub-string matches in a particular string and finding dictionary strings that almost match the style [19]. The way the string formation is encoded can affect the string search algorithms. If the variable display encoding is in use, it may be slower to find the character "n". In turn, this may significantly slow down some search algorithms. One solution is to search for the sequence of code units. However, doing so may result in erroneous matches unless the encoding is specifically designed to avoid it [20]. The String matching algorithms used for malware detection from this research are:

A. NAÏVE SUBSTRING SEARCHING ALGORITHM

The Naive string-matching algorithm slides the pattern one by one from the starting point to the leftmost corner. The length of the text string and pattern substring must be ascertained. After all slides have been checked, the characters are then checked individually. If all of the characters match at the end of the substring search, the matches between the text and pattern are printed [21]. However, only one or two letters should be examined in each wrong situation to establish whether its placement is incorrect. On average, the time taken will be O (n + m). At worst, the time taken will be O (nm) [22]. Valid shifts should be obtained by using a loop to check the condition.

P [1 . . . m] = T[s + 1 . . . s + m] for each of the n - m + 1 possible values of s [13].

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

B. RABIN-KARP SUBSTRING SEARCHING ALGORITHM

This string-matching algorithm performs well. It was adapted from another algorithm created to address problems such as matching two-dimensional patterns. This basic algorithm uses numerical concepts including a two-digit formula for a third digit unit [19]. However, this algorithm is used quite differently to solve string matching problems as it relies on fragmentation techniques. The hash function h(x) must be calculated for pattern P [0...m-1]. Matching must then be conducted using the hash function to calculate the length of each substring [m-1] in the text. A hash function or hash value can be effectively generated through ruling a hash function to calculate and recalculate hash [23].

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

If there is one pattern string to compare with one text string, the following operations must be performed:

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

The slowest runtime for the Rabin-Karp algorithm is O (m (n - m + 1)).

C. BRUTE FORCE SUBSTRING SEARCHING ALGORITHM

Brute Force is a simple and appropriate technique to find solutions to non-deterministic polynomial (NP) problems. Brute Force algorithms are usually used when the size of the problem is limited as Brute Force algorithms tend to grow rapidly in volume when addressing many problems simultaneously. This algorithm is used to find the line of normal numbers [n] through multiplying and enumerating all integers from [1 to n] in each device [24].

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

D. KNUTH-MORRIS-PRATT SUBSTRING SEARCHING ALGORITHM

This algorithm uses information provided by a particular table to avoid reexamination. This is obtained by preprocessing the pattern. This algorithm is a linear time algorithm of O (n + m). It is composed of two sections. The search section consists of finding the correct transitions in the text where the complexity of time is O (N). This is obtained by comparing the pattern and transitions of the text. The second section involves preprocessing the pattern [23]. The goal of preprocessing the pattern is to obtain a table that shows the following mode to be processed after a mismatch. For P [0 ... m-1], the table will show the result of preprocessing each letter j [25].

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

This algorithm is a highly efficient string-matching algorithm. It is a standard method used for a series of searches or matching literature. This algorithm pretreats the searched string. Then it is uses information collected during the previous step to function and skip the next part. It works faster in conjunction with the form. The algorithm searches or matches the tail style, rather than the header style, and can navigate across the text with multiple jumps instead of searching for a single character within the text [23]. This algorithm must create a "bad match table" for any remaining character from the last character equal to the length value. If it is not already defined, the pattern in the text can be compared starting from the rightmost character of the pattern. When a mismatch occurs, the pattern can be moved to the right-hand side of the value in the same table [26].

Create a "Bad Match Table".
Value = Length of Pattern – index – 1

The term, "string matching", refers to finding all the occurrences of a letter pattern in a text. The hashing-based algorithm compares the hash values of letters in the text with the values of letters in the pattern. If all hash values are equal, a match may occur. The letters in the text and those in the pattern are then compared to verify that a match has occurred [27].

By using these five algorithms, we conduct an experiment with several families of malwares to differentiate and identifying malware families.

VII. ANALYZING AND REVERSING EXE. MALWARE FILES

This feature can be used by developers and users alike to analyze and reverse executable malware files. By using some reverse engineering tools such as OllyDbg and PeStudio to discover the malware family of each file. It provides positive results as an API-call function and then saves all results, except for repeated results, to use in other purpose and requirements.

Malware files are files with .exe extensions. When malware files are opened, the user's computer will become infected. If the attacker's malware contains remote access capabilities, it will then send sensitive information to the attacker. It is impossible to evaluate the malware file until it has been analyzed and information on its features has been obtained. Reverse engineering must then be performed to reach the source code of the malware file.

Reverse engineering should be completed to reverse (mirror) malware sample files and obtain their code. Tools such as OllyDbg, IDApro, and other debuggers can be used to reverse malware samples and obtain their source codes, memory locations. In addition, these tools can evaluate when the malware will be activated in the infected computer.

The API-call functions explain the behavior of the malware. These can be extracted from malware files using tools such as PeStudio and OllyDbg. The API-call functions are then compared to two or more malware samples from the same family or from different families to identify which family the malware belongs to.

VIII. TEST CASE AND RESULTS

After analyzing many malware applications, the implications of this research can be understood using the statistical result that was obtained by using five different pattern matching algorithms to ascertain the API-call functions of each malware sample. In order to differentiate between malware families, the most common API-call functions were established. All malware families may contain some of the API-call functions identified. Through establishing common API-call functions in different malware files, it is possible to determine the family each malware file belongs to.

To conduct the first experiment, three different malware families was compared. Furthermore, from each family we'll take three samples of malware belonging to each family with a total of nine samples, as shown in Table II:

Malware Families	Malware Samples
Adware	A, B and C
Trojan	A, B and C
Worm	A, B and C

Table II: Malware Families and samples list

A. DIFFERENTIATING BETWEEN MALWARE FAMILIES

For the purpose of experiment and achieve accurate ratios and precise results, this research proposed to extract and use the string of similar API-call functions from this malware families and compared between same families and cross-compared between different malware families to establish which malware family the tested file belongs to.

We extracted the common API-call functions from all malware files separately as follows. We worked on each of Adware A, Adware B, and Adware C to extract the common API-call functions of each of them. Later we compared and extracted the shared API-call functions between Adware A with Adware B, Adware A with Adware C and Adware B with Adware C, to get three results of shared API-call functions between them. At the end, we merged all common API-call functions without duplication to get a file that contained all common API-call functions from Adware samples. The same process was conducted on Trojan and Worm files to get all common API-call functions. However, when distinguishing between malware families, none of the malware families contain unique API-call functions.

The common API-call functions were used to differentiate between various malware families. It is possible to differentiate two or more malware samples by comparing the common API-call functions that they share. For example, all common API-call functions between Adware A and Adware B were extracted. A non-duplication function was then written for each file to differentiate it. The malware analysis extracted all common API-call functions from those files and then compared it to the others. To differentiate between malware families we need to compare each unknown malware file from Adware, Trojan and Worm with All Adware, All Trojan and All Worm. These files contain all common API-call functions of Adware, Trojan and Worm files, each file is tested in the three programs, to know which family the file belongs to.

For example, we compared Adware A with All Adware, All Trojan and All Worm. From this comparison we saw three different ratios for each file with five algorithms, the highest ratio meant that this unknown malware file belonged to the highest ration family, and the highest ratio of matches were selected to establish each of the malware files to the family they belonged. Then we did the same process for each of the other malware files from Adware, Trojan and Worm.

Table III below show the differentiation between Adware files with Trojan and Worm files using their API-call functions.

Malware 1	Malware 2	API Calls 1	API Calls 2	Common API	Naïve	Karp	Brute Force	Morrie's	Boyer
Adware A	All Adware	40	32	23	71.9	71.9	71.9	71.9	21.9
Adware A	All Trojan	40	183	24	13.1	13.1	13.1	13.1	2.2
Adware A	All Worm	40	17	4	23.5	23.5	23.5	23.5	5.9
Adware B	All Adware	58	32	23	71.9	71.9	71.9	71.9	15.6
Adware B	All Trojan	58	183	36	19.7	19.7	19.7	19.7	1.6
Adware B	All Worm	58	17	6	35.3	35.3	35.3	35.3	5.9
Adware C	All Adware	158	32	24	75.0	75.0	75.0	75.0	21.9
Adware C	All Trojan	158	183	36	19.7	19.7	19.7	19.7	1.1
Adware C	All Worm	158	17	5	29.4	29.4	29.4	29.4	11.8

Table III: Adware differentiation process with Trojan and Worm files

Table IV below show the differentiation between Trojan files with Adware and Worm files using their API-call functions.

Malware 1	Malware 2	API Calls 1	API Calls 2	Common API	Naïve	Karp	Brute Force	Morrie's	Boyer
Trojan A	All Adware	191	32	21	65.6	65.6	65.6	65.6	18.8
Trojan A	All Trojan	191	183	156	85.2	85.2	85.2	85.2	4.9
Trojan A	All Worm	191	17	5	29.4	29.4	29.4	29.4	5.9
Trojan B	All Adware	187	32	19	59.4	59.4	59.4	59.4	15.6
Trojan B	All Trojan	187	183	154	84.2	84.2	84.2	84.2	4.9
Trojan B	All Worm	187	17	4	23.5	23.5	23.5	23.5	5.9
Trojan C	All Adware	191	32	21	65.6	65.6	65.6	65.6	18.8
Trojan C	All Trojan	191	183	156	85.2	85.2	85.2	85.2	4.9
Trojan C	All Worm	191	17	5	29.4	29.4	29.4	29.4	5.9

Table IV: Trojan differentiation process with Adware and Worm files

Table V below show the differentiation between Worm files with Adware and Trojan files using their API-call functions.

Malware 1	Malware 2	API Calls 1	API Calls 2	Common API	Naïve	Karp	Brute Force	Morrie's	Boyer
Worm A	All Adware	14	32	4	12.5	12.5	12.5	12.5	6.3
Worm A	All Trojan	14	183	5	2.7	2.7	2.7	2.7	0.0
Worm A	All Worm	14	17	13	76.5	76.5	76.5	76.5	17.6
Worm B	All Adware	72	32	16	50.0	50.0	50.0	50.0	18.8
Worm B	All Trojan	72	183	16	8.7	8.7	8.7	8.7	1.1
Worm B	All Worm	72	17	15	88.2	88.2	88.2	88.2	23.5
Worm C	All Adware	72	32	16	50.0	50.0	50.0	50.0	18.8
Worm C	All Trojan	72	183	16	8.7	8.7	8.7	8.7	1.1
Worm C	All Worm	72	17	15	88.2	88.2	88.2	88.2	23.5

Table V: Worm differentiation process with Adware and Trojan files

B. COMPARING MALWARE FILES

a) COMPARING TWO OR MORE MALWARE FILES IN THE SAME FAMILY

When comparing two or more malware samples in the same family, a high percentage ratio of matching is obtained due to the high number of matching API-call functions between them. When we are comparing it, it gives us a high percentage ratio, such as the schedules of comparing file of Adware, Trojan, and Worm directly together. In the case of Adware with Adware, Trojan with Trojan and Worm with Worm four out of five algorithms give us the best ratio for

Adware, Trojan and Worm, as shown on the below tables. The algorithms that produced the best results when comparing malware API-call functions were: Naïve, Karp, Brute Force and Knuth–Morris–Pratt (KMP).

Adware 1	Adware 2	No. API.1	No. API.2	Common API	Naïve	Karp	Brute Force	KMP	Boyer
A	B	40	58	15	25.9	25.9	25.9	25.9	3.4
A	C	40	158	16	10.1	10.1	10.1	10.1	1.3
B	A	58	40	15	37.5	37.5	37.5	37.5	5.0
B	C	58	158	17	10.8	10.8	10.8	10.8	0.6
C	A	158	40	16	40.0	40.0	40.0	40.0	5.0
C	B	158	58	17	29.3	29.3	29.3	29.3	3.4

Table VI: Directly comparing Adware files

Trojan 1	Trojan 2	No. API.1	No. API.2	Common API	Naïve	Karp	Brute Force	KMP	Boyer
A	B	191	187	182	97.3	97.3	97.3	97.3	15.0
A	C	191	191	191	100.0	100.0	100.0	100.0	10.5
B	A	187	191	181	94.8	94.8	94.8	94.8	9.4
B	C	187	191	181	94.8	94.8	94.8	94.8	9.4
C	A	191	191	191	100.0	100.0	100.0	100.0	10.5
C	B	191	187	182	97.3	97.3	97.3	97.3	15.0

Table VII: Directly comparing Trojan files

Worm 1	Worm 2	No. API.1	No. API.2	Common API	Naïve	Karp	Brute Force	KMP	Boyer
A	B	14	72	14	19.4	19.4	19.4	19.4	2.8
A	C	14	72	14	19.4	19.4	19.4	19.4	2.8
B	A	72	14	14	100.0	100.0	100.0	100.0	14.3
B	C	72	72	72	100.0	100.0	100.0	100.0	9.7
C	A	72	14	14	100.0	100.0	100.0	100.0	14.3
C	B	72	72	72	100.0	100.0	100.0	100.0	9.7

Table VIII: Directly comparing Worm files

b) COMPARING TWO OR MORE MALWARE FILES IN DIFFERENT FAMILIES

When comparing two or more malware samples in different malware families, the ratio of matching will decrease as the API-call functions are more varied. During the comparison, a low percentage ratio will be obtained. This is seen in the below comparison tables for Adware versus Trojans, Adware versus Worms and Trojans versus Worms.

Adware	Trojan	No. API.A	No. API.T	Common API	Naïve	Karp	Brute Force	KMP	Boyer
A	A	40	191	25	13.1	13.1	13.1	13.1	3.1
A	B	40	187	25	13.4	13.4	13.4	13.4	3.7
A	C	40	191	25	13.1	13.1	13.1	13.1	3.1
B	A	58	191	41	21.5	21.5	21.5	21.5	1.6
B	B	58	187	40	21.4	21.4	21.4	21.4	1.6
B	C	58	191	41	21.5	21.5	21.5	21.5	1.6
C	A	158	191	44	23.0	23.0	23.0	23.0	2.6
C	B	158	187	41	21.9	21.9	21.9	21.9	3.7
C	C	158	191	44	23.0	23.0	23.0	23.0	2.6

Table IX: Comparing Adware files with Trojan files

Adware	Worm	No. API.A	No. API.W	Common API	Naïve	Karp	Brute Force	KMP	Boyer
A	A	40	14	3	21.4	21.4	21.4	21.4	7.1
A	B	40	72	12	16.7	16.7	16.7	16.7	2.8
A	C	40	72	12	16.7	16.7	16.7	16.7	2.8
B	A	58	14	5	35.7	35.7	35.7	35.7	7.1
B	B	58	72	16	22.2	22.2	22.2	22.2	2.8
B	C	58	72	16	22.2	22.2	22.2	22.2	2.8
C	A	158	14	3	21.4	21.4	21.4	21.4	7.1
C	B	158	72	31	43.1	43.1	43.1	43.1	2.8
C	C	158	72	31	43.1	43.1	43.1	43.1	2.8

Table X: Comparing Adware files with Worm files

Trojan	Worm	No. API.T	No. API.W	Common API	Naïve	Karp	Brute Force	KMP	Boyer
A	A	191	14	5	35.7	35.7	35.7	35.7	7.1
A	B	191	72	20	27.8	27.8	27.8	27.8	4.2
A	C	191	72	20	27.8	27.8	27.8	27.8	4.2
B	A	187	14	4	28.6	28.6	28.6	28.6	7.1
B	B	187	72	17	23.6	23.6	23.6	23.6	2.8
B	C	187	72	17	23.6	23.6	23.6	23.6	2.8
C	A	191	14	5	35.7	35.7	35.7	35.7	7.1
C	B	191	72	20	27.8	27.8	27.8	27.8	4.2
C	C	191	72	20	27.8	27.8	27.8	27.8	4.2

Table XI: Comparing Trojan files with Worm files

The results show several common API-call functions shared between two types of malware for each family. It should be noted that if two types of malware are compared and a ratio is obtained, reversing the comparison sequence between the two types of malware will result in a different ratio. This can be seen in the below comparison tables for Trojan versus Adware, Worms versus Adware, and Worm versus Trojans.

Trojan	Adware	No. API.T	No. API.A	Common API	Naïve	Karp	Brute Force	KMP	Boyer
A	A	191	40	25	62.5	62.5	62.5	62.5	7.5
A	B	191	58	40	69.0	69.0	69.0	69.0	8.6
A	C	191	158	43	27.2	27.2	27.2	27.2	2.5
B	A	187	40	25	62.5	62.5	62.5	62.5	7.5
B	B	187	58	38	65.5	65.5	65.5	65.5	8.6
B	C	187	158	39	24.7	24.7	24.7	24.7	2.5
C	A	191	40	25	62.5	62.5	62.5	62.5	7.5
C	B	191	58	40	69.0	69.0	69.0	69.0	8.6
C	C	191	158	43	27.2	27.2	27.2	27.2	2.5

Table XII: Comparing Trojan files with Adware files

Worm	Adware	No. API.W	No. API.A	Common API	Naïve	Karp	Brute Force	KMP	Boyer
A	A	14	40	3	7.5	7.5	7.5	7.5	0.0
A	B	14	58	5	8.6	8.6	8.6	8.6	0.0
A	C	14	158	3	1.9	1.9	1.9	1.9	0.6
B	A	72	40	13	32.5	32.5	32.5	32.5	5.0
B	B	72	58	15	25.9	25.9	25.9	25.9	5.2
B	C	72	158	31	19.6	19.6	19.6	19.6	2.5
C	A	72	40	13	32.5	32.5	32.5	32.5	5.0
C	B	72	58	15	25.9	25.9	25.9	25.9	5.2
C	C	72	158	31	19.6	19.6	19.6	19.6	2.5

Table XIII: Comparing Worm files with Adware files

Worm	Trojan	No. API.W	No. API.T	Common API	Naïve	Karp	Brute Force	KMP	Boyer
A	A	14	191	5	2.6	2.6	2.6	2.6	0.5
A	B	14	187	4	2.1	2.1	2.1	2.1	0.0
A	C	14	191	5	2.6	2.6	2.6	2.6	0.5
B	A	72	191	19	9.9	9.9	9.9	9.9	1.0
B	B	72	187	17	9.1	9.1	9.1	9.1	2.1
B	C	72	191	19	9.9	9.9	9.9	9.9	1.0
C	A	72	191	19	9.9	9.9	9.9	9.9	1.0
C	B	72	187	17	9.1	9.1	9.1	9.1	2.1
C	C	72	191	19	9.9	9.9	9.9	9.9	1.0

Table XIV: Comparing Worm files with Trojan files

If the first Adware sample (Adware A) is compared with the first Trojan sample (Trojan A), five comparison ratios will be generated (one for each of the five algorithms used). If the order of the comparison process is then reversed and the same samples are compared, the comparison ratio will be different. The comparison process is not reversible when comparing two or more malware families.

IX. CONCLUSION AND FUTURE WORK

The term malware is an expression used to indicate various kinds of malignant software. While there are many malware detection methods, this research utilized, Application Programming Interface call functions to identify malware families alongside five pattern matching technique algorithms that can be potentially used for string similarity detection which in turn can be used to differentiate a malware from another. With fine-tuned thresholds, the potential can be boosted further to provide a considerable degree of malware detection.

The main advantages of this technique are the significantly different performance characteristics. For that it needs to think about using the best pattern matching algorithms for finding substring from malware families, then differentiate and identifying malware families. Alongside reverse engineering tools such as OllyDbg and PeStudio. This combination provides an excellent method of reversing executable malware files and comparing them.

Potential future work would be creation of new string similarity detection algorithm to improve the efficiency of these five algorithms.

REFERENCES

[1] F. Mastjik, C. Varol & A. Varol, "Comparison of Pattern Matching Techniques on Identification of Same Family Malware", IJISS, Vol.4, No.3, retrieved from http://dergipark.gov.tr/download/article-file/147950 [Feb 28, 2018]

[2] S. Yu, S. Zhou, L. Liu, R. Yang & J. Luo, "Malware Variants Identification Based on Byte Frequency", 2010, 2nd NSWCTC and IEEE, retrieved from http://ieeexplore.ieee.org/document/5480417/ [Mar 1, 2018]

[3] J. Kaur and S. Sharma, "Study of Malware Based on Pattern Matching Techniques", 2015, IJEDR, Vol. 3, Issue 2, ISSN: 2321-9939, retrieved from https://www.ijedr.org/papers/IJEDR1502194.pdf [Mar 1, 2018]

[4] P. Liu and X. Wang, "Inductive Learning in Malware Detection", 2008 IEEE, retrieved from http://ieeexplore.ieee.org/document/4681110/ [Mar 1, 2018]

[5] C. Chio & D. Freeman, "Machine Learning and Security, Protecting Systems with Data and Algorithms", 2018 O'Reilly Media and CA, retrieved from https://books.google.iq/books?id=lyJJDwAAQBAJ&pg=PT163&lpg=PT163&dq=identify+the+same+family+of+malware&source=bl&ots=5qr1Csjak7&sig=ACfU3U36EVAg47__7uArpyJIMJ7x02IsXw&hl=ar&sa=X&ved=2ahUKEwi-lYqE79jnAhX5zMQBHWzmCNgQ6AEwDHoECAoQAQ#v=onepage&q&f=false [Feb 17, 2020]

[6] Alatabbi, A., Al-Jamea, M. and Iliopoulos, C., S., Malware Detection using Computational Biology Tools, (April 2013), IACSIT, Vol. 5, No. 2, retrieved from http://www.ijetch.org/papers/566-ST0031.pdf [Aug 17, 2020]

[7] Sahu, M., K., Ahirwar, M. and Hemlata, A., (2014), A Review of Malware Detection Based on Pattern Matching Technique, (IJCSIT), Vol. 5 (1), ISSN: 0975-9646, retrieved from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.641.8308&rep=rep1&type=pdf [Aug 17, 2020]

[8] Salim, A. & Shijo, P., V., (2015), Integrated static and dynamic analysis for malware detection, ICICT, Volume 46, Pages 804-811, retrieved from https://ac.els-cdn.com/S1877050915002136/1-s2.0-S1877050915002136-main.pdf?_tid=847de5ca-0130-11e8-b35b-00000aab0f01&acdnat=1516817047_76db72a5750e29a858f0b271d14e4589 [Feb 1, 2018]

[9] D. Uppal, R. Sinha, V. Mehra and V. Jain, (2014), "Exploring Behavioral Aspects of API calls for Malware Identification and Categorization", 6th ICCICN, retrieved from https://ieeexplore.ieee.org/document/7065596 [Feb 1, 2018]

[10] C. Wang, J.Pang, R. Zhao, W. Fu and X. Liu, (2009), "Malware Detection Based on Suspicious Behavior Identification", IEEE, 1st IWETCS, DOI 10.1109/ETCS, retrieved from https://ieeexplore.ieee.org/document/4959020 [Feb 1, 2018]

[11] K.Kamiya, K.Aoki, K.Nakata, T.Sato, H.Kurakami and Masaki Tanikawa, (2015), "The Method of Detecting Malware-Infected Hosts Analyzing Firewall and Proxy Logs", APSITT, IEICE, retrieved from https://ieeexplore.ieee.org/document/7217113 [Feb 1, 2018]

[12] G. Pitolli, L. Aniello, G. Laurenza, L. Querzoni & R. Baldoni, "Malware Family Identification with BIRCH clustering", 2017 ICCST and IEEE, retrieved from http://ieeexplore.ieee.org/abstract/document/8167802/?reload=true [Feb 28, 2018]

[13] J. D. Seideman, B. Khan & G. B. Brahim, "Determining Vulnerability Resolution Time by Examining Malware Proliferation Rates", 2013 IEEE, retrieved from http://manualzz.com/doc/36847796/determining-vulnerability-resolution-time-by-examining-ma... [Feb 28, 2018]

[14] Microsoft, "The Evolution of Malware and the Threat Landscape – a 10 – year review", 2012, retrieved from https://www.google.iq/search?dcr=0&source=hp&ei=UbCWWtLsBoXSsAeqkJOACQ&q=The+evolution+of+malware+and+the+threat+landscape+-+a+10-year+review%3A+key+findings&oq=The+evolution+of+malware+and+the+threat+landscape+-+a+10-year+review%3A+key+findings&gs_l=psy-ab.3...772.772.0.1095.1.1.0.0.0.0.185.185.0j1.1.0....0...1c.1.64.psy-ab..0.0.0....0.5sI5PRbGgoc [Feb 28, 2018]

[15] Security Report, "The AV-TEST Security Report", (2016/17), retrieved from https://www.av-test.org/fileadmin/pdf/security_report/AV-TEST_Security_Report_2016-2017.pdf [Feb 28, 2018]

[16] DePaul, N. "Static Testing vs. Dynamic Testing", 2013, retrieved from https://www.veracode.com/blog/2013/12/static-testing-vs-dynamic-testing [May 9, 2018]

[17] Static Analysis, (2018), retrieved from https://en.wikipedia.org/wiki/Static_analysis [May 9, 2018]

[18] M. Rouse, "Static Code Analysis", 2006, retrieved from https://searchwindevelopment.techtarget.com/definition/static-analysis [May 9, 2018]

[19] A. Pokiya, "String Matching Algorithms", 2015, retrieved from https://www.slideshare.net/Ashikapokiya12345/string-matching-algorithms-52582907?next_slideshow=1 [May 9, 2018]

[20] A. Choudhury, "String Matching Algorithms", 2015, retrieved from https://www.slideshare.net/alokeparnachoudhury/string-matching-algorithm [May 9, 2018]

[21] P. Singh, "Naive String Matching Algorithm", 2013, retrieved from https://www.youtube.com/watch?v=RhNM6jvvNjU [May 9, 2018]

[22] Y. Shakeel, "Naive Pattern Searching", 2016, retrieved from https://www.youtube.com/watch?v=xP5Ox-df_ik [May 9, 2018]

[23] M. Gou, "Algorithm for String Matching", 2014, retrieved from http://www.student.montefiore.ulg.ac.be/~s091678/files/OHJ2906_Project.pdf [May 9, 2018]

[24] B. Holczer, "Brute Force Substring Search Algorithm", 2017, retrieved from https://www.youtube.com/watch?v=vtnpzDPgaU0 [May 9, 2018]

[25] T. Roy, "Knuth–Morris–Pratt (KMP) Pattern Matching (substring search)", 2015, retrieved from https://www.youtube.com/watch?v=GTJr8OvyEVQ [May 9, 2018]

[26] M. Slade, "Boyer–Moore Horspool Algorithm", 2014, retrieved from https://www.youtube.com/watch?v=PHXAOKQk2dw [May 9, 2018]

[27] R. A. Rasool, A. Tiwari, G. Singla, & N. Khare, "String Matching Methodologies: A Comparative Analysis", 2012, retrieved from https://www.researchgate.net/publication/268273984_String_Matching_MethodologiesA_Comparative_Analysis [May 9, 2018]