# Comparison of Pattern Matching Techniques on Identification of Same Family Malware

Ferdiansyah Mastjik*‡, Cihan Varol*, Asaf Varol**

* Department of Computer Science, Sam Houston State University, 1806 Avenue J Huntsville Texas 77340 USA

** Department of Software Engineering,Firat University, Bilgi İşlem Dairesi, Enformatik Bölümü, 23119, Turkey

‡ Ferdiansyah Mastjik ; Department of Computer Science, Sam Houston State University, 1806 Avenue J Huntsville Texas 77340 USA:

Tel: +1 936 294 3846, e-mail: fxm010@shsu.edu

**Abstract-** Development in computing technology for the past decade has also given rise to threats against the users, particularly in form of malware. However, manual malware identification effort is being overwhelmed due to the sheer number of malware being created every day. Most of the malware are not exactly created from scratch; large numbers of them are byproducts of particular malware family. This means that same or slightly modified resolution can be applied to counter their threat. This paper analyzes string matching methods for identification of same family malware. We investigate and compare the effectiveness of three well-known pattern matching algorithms, namely Jaro, Lowest Common Subsequence (LCS), and N-Gram. After researching these three algorithms we found out thresholds of 0.79 for Jaro, 0.79 for LCS, and 0.54 for N-Gram showed to be effective for string similarity detection between malware.

**Keywords-** Jaro;Longest Common Subsequence; Malware Analysis; N-gram; String Similarity.

## 1. Introduction

In the annals of computing history, technological advances have enabled advances in threats against the users. One form of these threats is malware. Since malware's first creation in 1970, there have been numerous variations of malware circulating around the world. The complexity and attack vectors of malware have also advanced along with advancements in technology. In recent decades, malware mainly targeted personal computer users but as mobile operating systems become increasingly common, malware target has also shifted toward mobile operating system users. Similarly if recently private company networks were the main target of malware, it is possible that the next malware attack may be aimed to cloud computing network.

While the motivation behind malware creation varies, there is no doubt about its destructive nature by its exploitation of vulnerabilities. These vulnerabilities can be found any time, even on the same day the software is released [1].

Concerns regarding apparent vulnerabilities of computer users, coupled with the malware capability to elude detection, trigger a unified effort between software companies and academic researchers. Together these parties work to devise analysis methods that will identify and help resolve these threats.

Due to the significance of malware identification in formulating solution, we perform research on improving the framework of same family malware identification. Same family malware identification hold significant importance because the same base solution to a malware attack can be applied within malwares of the same

family. This research for same family malware identification is done through application of string matching algorithms with appropriate threshold values. Specifically, we have evaluated the performance of well-known string matching algorithms to identify same family malware. As reflected later in Section 4, we have shown that string matching algorithms can be employed for differentiating same family member malwares from distinct ones.

The rest of the paper is organized as follows. In Section 2, related studies in malware identification are discussed. The employed methodology is elaborated in Section 3. Test case and results are discussed in Section 4 and the paper is finalized with discussion and conclusion section.

## 2. Background Study

According to Islam et al. [2], manual malware analysis has become overwhelmed due to the higher volume of malware being produced every day. Contrary to this; however, Walenstein et al. [3] found through their research that the high volume of malware is a byproduct or a variant of a malware family. Therefore further research must be done to analyze malware similarities and differences for identification purpose.

Motivated by the facts presented by Walenstein [3] and Kendall et al [4], the method of malware identification and classification are categorized into two types of analysis: dynamic analysis and static analysis. According to [4], dynamic analysis is done through studying the behavior of a running malicious code in a controlled environment such as in a virtual machine. Dynamic analysis may yield immediate information on what the malware is doing instead of how it accomplishes its purpose. The authors in [4] stated that the static analysis performs its task by studying a malicious code structure and extracting assembly code without actually running it. Static analysis research can be slow and exhaustive, but the result gained is very detailed.

This research will focus on static analysis due to the fact that it is safer and may provide detailed results albeit in a slower and more complex way. According to [2] and [4], static analysis is usually done in one of these ways: Features analysis or contextual analysis. The authors in [2] explained that features analysis conducted by analyzing bytes, binary, and disassembly. This type of analysis measured certain patterns in the malware and was exemplified by [2], [3] and Park et al [5]. Features analysis key argument is that no matter what code is being executed, the end result is still the same. Contextual analysis utilizes command string/code analysis. This method examines strings inside suspected malware that is represented in a certain programming language or an unusual Windows PE API calls. Islam et al. [2], Lee et al. [6], and Sulaiman et al. [7] used this type of static analysis in their research.

The authors in [6] and [7] showed that contextual analysis yields more accurate results and provides better classifications. They further argued that by knowing which strings were used, it may show which programming language and Windows API are being employed. This will be helpful in determining exploits and the pattern being used by the malware. The authors in [5] proposed that both methods of malware analysis have something in common: both perform difference and similarity analysis on the malware. The similarities helped detect generic signatures while the differences helped on improving an existing solution. Combined, features analysis and contextual analysis were the main methods being used in malware identification and classification.

The authors in [2], performed classification of malware features through a combination of methods called Function Length Frequency (FLF) and Printable String Information (PSI). FLF measures the “length” of the function through the number of bytes of code in it, and the frequency these “length” occurs within a particular sample of malware. They found out that within the same malware family, the shape of the function’s “length” pattern was similar. PSI method extracted printable strings and analyzes of their occurrence in samples to form a pattern. PSI has a global list of available API calls and counts the occurrence to determine the pattern. The authors in [2], did not mention about similarity analysis being done in PSI but rather looked for the occurrence count. This can potentially be enhanced by introducing string similarity detection algorithm to reduce redundant occurrence count. Further the authors

claim classification efficiency being over 98 percent.

The authors in [3], proposed a method which analyzes distinctive feature comparison of information. This is opposite to the traditional method of looking at a particular signature. They employed a method to determine "n-grams" which is a sequence of n-characters found in succession inside disassembly. They also measured "n-perms" which is the same as n-grams except the ordering of the characters is not taken into account. The results were further weighted for relevance and applied into a vector model by calculating the result using standard cosine similarity formula. The lower the score, the less relevant the feature was. The relevant features that match a certain similarity threshold were then compared iteratively through a database where it was ranked in the order of similarity for further classification.

Research by [5] featured assembly instruction sequence similarity analysis through byte comparison and byte order analysis. This method calculated the frequency of certain instructions that appear in sequence of cmp blocks in assembly instruction. This frequency was modeled into a vector and further calculated to compare the cmp blocks between the source program and target program to find the similarity. The formula being used to calculate the vector of similarity is the standard cosine formula such as being used by [3].

The authors in [5] used the Levenshtein distance [6] to calculate the similarity based on the assembly instruction order. If there are some sections with similar frequencies of assembly instructions and similar order in the sequence, it is considered similar. The authors in [7] performed their research by extracting strings of malicious executable. They devised a method to filter these strings based on several qualifications such as the length of strings, and filtering out normal/common API calls. Once the data is filtered, they employed Jaro-Winkler algorithm with a modified Levenshtein distance algorithm to determine if one malware is a member of a certain family through its similarities. The researchers also managed to show that the lower level similarity in the same family of malware is a possible indication of polymorphism or mutation in the code.

In [8], the authors gathered a collection of API "snippets" extracted from binary into a readable windows PE string format. These snippets were a collection or sequences of unusual API calls. These sequences share characteristics between malware and the authors assumed they do not appear in a normal program. To support these assumptions, the snippets were compared to normal program to see if any of the bad snippets appear anywhere in the body of code from the extracted string. If there were no matches found within non-malicious program then it was considered a malicious characteristic. These characteristics were then organized further for classification and identification of certain malware family.

From [7] and [8] we see that [7] employs a certain kind of modified Jaro-Winkler algorithm, the approach that they used can definitely be improved through refining the filter and comparing the algorithm with other string similarity detection algorithms, while the pattern comparison in the other work, [8], can certainly benefit from actual use of string similarity algorithm to which it can potentially increase the efficiency of characteristics detection.

## 3. Methodology

Based on the previous study, we focused our research on leveraging the string similarity for the purpose of identification and classification of malware families. In order to do this, we choose three well-known algorithms for string similarity detection.

### 3.1. *Jaro Algorithm*

Jaro is a type of edit-distance algorithm which detects string similarity by counting minimum number of operations needed to transform one string to another [9]. The improved version of this algorithm, Jaro-Winkler, is relying on the literature results that claims most misspelling errors occur after the 4th character; thus, boosting the closeness degree if the mismatch is located later part of the word.

Therefore, we have modified the Jaro algorithm as follows:

$$D_j = ½ * (m/[S_1] + m/[S_2]) \quad (1)$$

Where S1 and S2 represent the number of characters in the strings respectively and m represents the total number of characters matching between S1 and S2.

### 3.2. *LCS Algorithm*

According to [10], LCS is a similarity detection algorithm which uses the concept of finding the longest subsequence common to all sequences in a group or series of sequences (often only two sequences). LCS can be a variant of edit distance algorithm when only insertion and deletion operations are allowed.

### 3.3. *N-Gram Algorithm*

According to [3], N-Gram algorithm is a string matching algorithm utilizing a collection of n-items from a contiguous sequence of strings. And n-gram of size one is referred as unigram, two is known as bigram, and three is a trigram. For string similarity matching usually a trigram is used as exemplified by [3] in their research. Borrowing from this previous research we establish our N-Gram test with tri-gram.

Using these three algorithms, we conduct an experiment with several families of malwares to determine the optimum condition for identifying same family malware.

However, in case of malware, the only reason for a misspelling to occur is in obfuscation and when that happens the scrambling can take place any part of the text. Moreover, Jaro algorithm takes the accidental transposition of two adjacent letters as one of the possible ways of creating mistyped data and this greatly impacts the distance score of two misspelled words. However, this scenario is rarely seen in malware; thus, having this measure can increase the false positive ratings.

## *4.* Test Case and Results

To conduct the first experiment, four random different malware families were obtained. Furthermore, we take randomly two or three random samples of malware belonging to each family with a total of eleven samples, as shown in Table 1 :

**Table 1**. Malware sample list

| Malware Family | Sample Name |
|---|---|
| Asylum | A1,A2,A3 |
| Bagle | B1,B2,B3 |
| Welchia | W1,W2 |
| Brontok | C1,C2,C3 |

For the purpose of experiment, the strings from these malware are extracted and compared between same families and cross-compared between different malware families. Strings extracted from the malware comprised of Windows API calls and non-Windows API calls. Non-API calls may contain either signature strings or patterns that can be used to identify malware families; however it may also contain meaningless strings. For our first series of tests the entire extracted strings are compared. Table 2 represents exact identical strings found in malware comparison.

A series of tests are run to establish optimum thresholds for two strings to be considered near identical. The thresholds are established using two criteria: 1. The adjusted thresholds need to show high quantity of near-identical strings between same-family malware. 2. The adjusted threshold needs to show zero or low quantity of near-identical strings between two malware of different families. Exact number of string similarity occurrence to be considered as high quantity and low quantity are refined throughout the experiment.

It is important to note that thirteen comparisons out of fifty five comparisons are being displayed as a demographic of the strings comparison algorithms effectiveness the reason for this is because other comparisons between different samples within same family or different family shows near or identical counts (e.g. C2vA1 shows the same result with C1vA1).

**Table 2**. Exact identical strings

| Malware Comparison | Modified JARO | LCS | NGRAM |
|---|---|---|---|
| C1vC2 | 18 | 18 | 18 |
| C1vC3 | 18 | 18 | 18 |
| C1vA1 | 2 | 2 | 2 |
| C1vB1 | 1 | 1 | 1 |
| W1vW2 | 12 | 12 | 12 |
| W1vA1 | 7 | 7 | 7 |
| W1vB1 | 3 | 3 | 3 |
| W1vC1 | 2 | 2 | 2 |
| A1vA2 | 51 | 51 | 51 |
| A1vA3 | 51 | 51 | 51 |
| A1vB1 | 20 | 20 | 20 |
| B1vB2 | 21 | 21 | 21 |
| B1vB3 | 26 | 26 | 26 |

**Table 3.** Near identical strings

| Malware Comparison | Modified JARO | LCS | NGRAM |
|---|---|---|---|
| C1vC2 | 5 | 3 | 0 |
| C1vC3 | 0 | 0 | 0 |
| C1vA1 | 0 | 0 | 0 |
| C1vB1 | 0 | 0 | 0 |
| W1vW2 | 5 | 4 | 1 |
| W1vA1 | 1 | 1 | 1 |
| W1vB1 | 4 | 4 | 2 |
| W1vC1 | 0 | 0 | 0 |
| A1vA2 | 14 | 14 | 1 |
| A1vA3 | 15 | 12 | 3 |
| A1vB1 | 10 | 9 | 2 |
| B1vB2 | 10 | 9 | 4 |
| B1vB3 | 11 | 8 | 3 |

Table 3 shows near identical strings found in malware comparison done with optimum thresholds of 0.79 for Jaro, 0.79 for LCS, 0.54 for n-Gram.

The result from the test also shows that any single string may have both identical and near identical results when compared. For example the string “lstrcpyA” from sample A1 has an identical match in sample A2 and a near identical match with the string “lstrcmpA” in the same malware sample.

In Figure 1, the same GET script appears on both sample A1 and A2, which is a very distinguishing similarity between the two. This line clearly indicates they belong to the same malware family

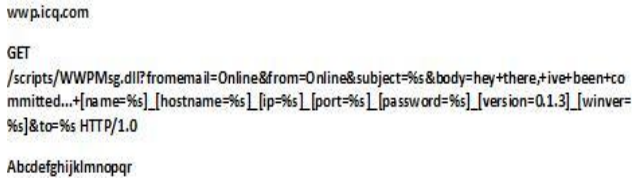


**Figure 1.** Identical Non-API Call from A1 vs A2

The conclusion derived from the comparison between full extracted strings shows Windows API calls can theoretically be used as a malware family identifying pattern as exemplified by [8], however, the test also shows Windows API calls might be just common program initializer that can occur on any kind of application. Non-API calls on the other hand clearly signify a pattern or may contain identifier.

To further gain focused result, another series of tests are done with the Windows API excluded from the string extracts. After removing the Windows API strings, the numbers of exact identical strings shows are shown in Table 4. This Windows API filtering resulted in a simplified and faster detection for malware differentiation.

The test impact is visible on the quantity of near -identical strings detection, it reduce the amount of near identical detection by almost 40 percent as shown on Table 5.

**Table 4.** Filtered identical strings

| Malware Comparison | Modified JARO | LCS | NGRAM |
|---|---|---|---|
| C1vC2 | 15 | 15 | 15 |
| C1vC3 | 0 | 0 | 0 |
| C1vA1 | 0 | 0 | 0 |
| C1vB1 | 0 | 0 | 0 |
| W1vW2 | 2 | 2 | 2 |
| W1vA1 | 0 | 0 | 0 |
| W1vB1 | 0 | 0 | 0 |
| W1vC1 | 0 | 0 | 0 |
| A1vA2 | 2 | 2 | 2 |
| A1vA3 | 0 | 0 | 0 |
| A1vB1 | 0 | 0 | 0 |
| B1vB2 | 0 | 0 | 0 |
| B1vB3 | 0 | 0 | 0 |

**Table 5.** Filtered near identical strings

| Malware Comparison | Modified JARO | LCS | NGRAM |
|---|---|---|---|
| C1vC2 | 5 | 3 | 0 |
| C1vC3 | 4 | 7 | 2 |
| C1vA1 | 0 | 0 | 0 |
| C1vB1 | 0 | 0 | 0 |
| W1vW2 | 5 | 4 | 1 |
| W1vA1 | 0 | 0 | 0 |
| W1vB1 | 0 | 0 | 0 |
| W1vC1 | 0 | 0 | 0 |
| A1vA2 | 10 | 10 | 1 |
| A1vA3 | 4 | 5 | 2 |
| A1vB1 | 0 | 0 | 0 |
| B1vsB2 | 7 | 7 | 3 |
| B1vB3 | 6 | 5 | 3 |

Removal of Windows API calls will speed up signature analysis, reduce false positives, and may potentially reduce the time needed for malware family classification when used in large batches of files.

Both tests show evidence that the appearance of meaningless strings is not a coincidence and may act as potential differentiating malware signature if, and only if:

1. The meaningless strings show as exact identical match.

2. It appears consistently over a significant population of a malware family.

Furthermore to support this argument, D. Plohman [11] showed that a string that seems meaningless might be a code reference in assembly, only that these codes are obfuscated.

As a final test, we have randomly selected forty six malware samples from twenty different malware families, and executed the algorithms to test the performance.

The results of the comparisons are four parameters used to calculate, sensitivity, specificity, precision, and accuracy of the algorithms. These four parameters are true positives, true negative, false positive, and false negative.

Any comparison showing negative similarity detection between two malware of different family is defined as true negative, while positive similarity detection between same family malware is called true positive. On the other hand, comparison showing negative similarity detection between two malware of same family is called false negative, while positive similarity detection between two malware of different family is called false positive.

Table 6 shows cumulative true positive, true negative, false positive, and false negative of all 20 families, however the calculation for sensitivity, specificity, precision, and accuracy is an average of all 20 families sensitivity, specificity, precision, and accuracy.

**Table 6.** Cumulative true/false positive and true/false negative

| Comparison Result | Modified Jaro | LCS | Ngram |
|---|---|---|---|
| Total True Positive | 18 | 13 | 8 |
| Total True Negative | 825 | 817 | 860 |
| Total False Positive | 57 | 49 | 18 |
| Total False Negative | 8 | 13 | 18 |

Sensitivity is a proportion of true positives [12] that are correctly identified by the malware comparison test. Specificity is a proportion of true negatives that are correctly identified by the comparison test [12]. Precision measures the relevancy of a data through fractional portion of true positive from the combination of true positives and false positives [13]. As shown in Table 7, modified Jaro algorithm has higher precision rates than the LCS and N-Gram.

**Table 7.** Performance results

| Algorithm | Specificity | Sensitivity | Precision |
|---|---|---|---|
| Original Jaro | 91.98% | 60.83% | 31.64% |
| Modified Jaro | 95.19% | 65% | 38.62% |
| LCS | 94.15% | 55% | 32.47% |
| N-Gram | 97.36% | 40% | 28.53% |

The last parameter we measure is the algorithms' accuracy, which is defined by the level of correct detection compared to the combined result of the test [13]. Average accuracy of 91.2% is counted for the original Jaro, 94.56% for modified Jaro, 93.33% for LCS, and 95.89% for N-Gram.

We found out one factor affecting the decision when considering two malwares as same family members. The numbers of near identicals need to be greater than 4 for Jaro, LCS, and N-Gram to achieve high specificity and precision values. One near identical often not enough as an adjustment for classification decision. This will yield to a large number of false positives.

## 5. Conclusion and future works

Jaro, LCS, and N-Gram algorithms 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.

Strings that belong to non-API calls category seems more likely to act as a pattern or a malware signature identifier, thus it is important to consider removing Windows API calls to reduce false positives and enhance analysis speed.

Potential future work can be directed in determining which Windows API can be classified as uncommon when found in a file. Instead of filtering Windows API completely, we can include a small set of uncommon Windows API which does not appear in normal clean executables or files. This may further aid the effort for malware family classification.

Another potential work would be creation of new string similarity detection algorithm to improve the efficiency of these three algorithms.

## References

[1] Microsoft, "The evolution of malware and the threat landscape - a 10-year review: key findings," 2012, http://download.microsoft.com/download/1/A/7/1A76A73B-6C5B-41CF-9E8C-33F7709B870F/Microsoft-Security-Intelligence-Report-Special-Edition-10-Year-Review-Key-Findings-Summary.pdf, Feb.2012 [Online; accessed September 2014]

[2] M.R.Islam , R.Tian, L.Batten, and S.Versteeg. "Classification of malware based on string and function feature selection." In Cybercrime and Trustworthy Computing Workshop (CTC), 2010 Second, pp. 9-17. IEEE, 2010.

[3] A.Walenstein, M.Venable, M.Hayes, C.Thompson, and A.Lakhotia. "Exploiting similarity between variants to defeat malware." In Proc. BlackHat DC Conf. 2007.

[4] K.Kendall, and C.McMillan. "Practical malware analysis." In Black Hat Conference, USA. 2007

[5] J.H.Park, M.Kim, B.Noh, and J.Joshi. "A Similarity based Technique for Detecting Malicious Executable files for Computer Forensics." In Information Reuse and Integration, 2006 IEEE International Conference on, pp. 188-193. IEEE, 2006.

[6] V.Levenshtein,"Binary codes capable of correcting deletions, insertions, and reversals". Soviet Physics Doklady 10 pp.707-710, USSR, 1966.

[7] J.Lee, C.Im, and H. Jeong. "A study of malware detection and classification by comparing extracted strings." In Proceedings of the 5th International Conference on

Ubiquitous Information Management and Communication, pp. 75. ACM, 2011

[8] A.Sulaiman, S.Mandada, S. Mukkamala, and A.Sung. "Similarity Analysis of Malicious Executables." In Proceedings of the 2nd International Conference on Information Warfare & Security, pp. 225. Academic Conferences Limited, 2007.

[9] M.Jaro. "Advances in record linkage methodology as applied to the 1985 census of Tampa Florida," In 84th Journal of the American Statistical Association, pp.414-420, 1989.

[10] L. Bergroth, H. Hakonen and T. Raita. "A Survey of Longest Common Subsequence Algorithms" In SPIRE (IEEE Computer Society), pp.39-48, 2000.

[11] D.Plohman. "Portable Executable 101 - a windows executable walkthrough", Internet: https://code.google.com/p/corkami/wiki/PE101?show=content, Aug.2014[Online, accessed August 2014]

[12] DG Altman and JM Bland. "Diagnostic tests. 1 :Sensitivity and specificity", In 38th Business Medical Journal ,1994.

[13] D.Olson and D.Delen, Advanced Data Mining Techniques, 1st ed, Springer, 2008, pp.138.