Embedding Fuzzy Rules with YARA Rules for Performance Optimisation of Malware Analysis

Nitin Naik[1], Paul Jenkins[1], Nick Savage[1], Longzhi Yang[2], Kshirasagar Naik[3] and Jingping Song[4]

[1]School of Computing, University of Portsmouth, United Kingdom
[2]Department of Computer and Information Sciences, Northumbria University, United Kingdom
[3]Department of Electrical and Computer Engineering, University of Waterloo, Canada
[4]Software College, Northeastern University, China
Email: {nitin.naik, paul.jenkins, nick.savage}@port.ac.uk, longzhi.yang@northumbria.ac.uk, snaik@uwaterloo.ca, songjp@swc.neu.edu.cn

***Abstract*—YARA rules utilises string or pattern matching to perform malware analysis and is one of the most effective methods in use today. However, its effectiveness is dependent on the quality and quantity of YARA rules employed in the analysis. This can be managed through the rule optimisation process, although, this may not necessarily guarantee effective utilisation of YARA rules and its generated findings during its execution phase, as the main focus of YARA rules is in determining whether to trigger a rule or not, for a suspect sample after examining its rule condition. YARA rule conditions are Boolean expressions, mostly focused on the binary outcome of the malware analysis, which may limit the optimised use of YARA rules and its findings despite generating significant information during the execution phase. Therefore, this paper proposes embedding fuzzy rules with YARA rules to optimise its performance during the execution phase. Fuzzy rules can manage imprecise and incomplete data and encompass a broad range of conditions, which may not be possible in Boolean logic. This embedding may be more advantageous when the YARA rules become more complex, resulting in multiple complex conditions, which may not be processed efficiently utilising Boolean expressions alone, thus compromising effective decision-making. This proposed embedded approach is applied on a collected malware corpus and is tested against the standard and enhanced YARA rules to demonstrate its success.**

***Index Terms*—YARA Rules; Fuzzy Rules; Fuzzy Logic; Fuzzy Hashing; Malware Analysis; Performance Optimisation; Ransomware.**

I. Introduction

YARA rules discover malware based on a string matching technique [1], which can be customised depending on the specific requirement of an individual or organisation to uncover targeted attacks and security threats. Both the quality and quantity of the YARA rules are crucial for analytic success as there should be an effective and sufficient number of YARA rules to improve the overall performance of the malware analysis, whereas the use of ineffective and superfluous YARA rules would adversely affect the overall performance of malware analysis [2]. Several approaches were proposed to generate more effective YARA rules to be used in improving the performance of YARA rules, however, there are some common issues related to its execution and outcome affecting the performance of majority of YARA rule-based systems utilising Indicator of Compromise (IoC) strings, for example:

1. YARA rules may not identify a sample as malware even on matching with several strings in any rule, whilst still remaining below the set threshold of the condition in that rule. Indeed, this set threshold of the condition in a rule is determined by the in-house security expert on YARA rule creation.
2. YARA rules may not identify a sample as malware even on matching with several strings in several rules although below the set threshold of condition in all the rules. However, the total matched strings in all the rules could be much higher than the set threshold of condition in any rule.
3. YARA rules are commonly used as a method to determine whether a sample is malware or not, irrespective of its other significant findings, thus not considering any probability between true and false (i.e., 1 and 0).

Nonetheless during the execution of YARA rules, the focus is to trigger YARA rules or not and thus often valuable information generated by YARA rules, which could have otherwise been captured, is lost. One resolution to this problem might be to utilise effectively either uncollected or unused information during the execution phase of YARA rules as it generates much useful information. The only requirement would be to capture uncollected information or utilise unused information through an effective mechanism. One such mechanism is the use of embedded fuzzy rules with YARA rules, which is designed to capture all the information generated by YARA rules and assess it through a fuzzy rule-based system to generate more useful and comprehensive outcomes, which would not usually be possible using standard YARA rules alone. The benefit of using fuzzy rules is that it can complement YARA rules for several fuzzy operations to optimise the performance of existing YARA rules without requiring any enhanced or AI techniques in the rule generation process.

This embedding of fuzzy rules with YARA rules demonstrates an initial concept where the most common condition

of *String Matching* and the additional condition of *Fuzzy Hash Matching* are used to develop and embed both rule types. However, it can be customised for a more complex analysis, by configuration of different parameters, multiple conditions and op-codes depending on the specific requirement of the malware analysis. The embedded approach is applied to the collected ransomware corpus, which includes four categories of ransomware WannaCry, Locky, Cerber and CryptoWall. Subsequently, its similarity detection result was compared against the similarity detection results of standard and enhanced YARA rules to demonstrate its success.

The paper is organised into the subsequent sections: Section II describes YARA Rules, Fuzzy Rules and Fuzzy Hashing. Section III discusses the proposed approach of embedding fuzzy rules with YARA rules and the development of fuzzy rules. Section IV presents the experimental analysis of standard YARA rules, enhanced YARA rules and embedded YARA rules and the comparative study of their similarity detection results. Finally, Section V presents the summary of the work and highlights the need of some future work.

II. Background

A. YARA Rules

YARA rules are developed to detect malware by primarily matching its signatures/strings with existing malware signatures/strings [1]. They contain predetermined signatures/strings related to known malware, used to attempt to detect a match when run against targeted files, folders, or processes [3]. These matching strings can be classified into three types: text strings; hexadecimal strings and regular expression strings (see Figs. 1 and 2). Text strings are generally a readable text complemented with some modifiers (e.g. nocase, ascii, wide, and fullword) to manage them more effectively [4]. Hexadecimal strings are a sequence of raw bytes complemented with three flexible formats wild-cards, jumps, and alternatives [4]. Regular expression strings are somewhat similar to text strings being a readable text complemented with some modifiers; which are available since version 2.0 and making YARA rules a more powerful tool [4].

Text strings and regular expression strings can be used to express a sequence of raw bytes through the use of escape sequences. The final part of YARA rules is a rule condition that specifies the number of signatures/strings that must be matched with the target to declare it as malware [5]. YARA conditions determine whether to trigger the rule or not, however, these conditions are Boolean expressions similar to those used in all other programming languages [4]. Consequently, this aspect of YARA rules can be strengthened by embedding fuzzy rules, thus improving the functionality and performance of YARA rules. This embedding may be very helpful for effective decision making, when YARA rules are more complex in nature, resulting in multiple complex conditions, which may not be dealt with efficiently on their own.

```
rule RuleName
{
meta:
  description = "descriptions of rule"
  author = "name"
  date = "dd/mm/yyyy"
  reference = "url"
strings:
  $text_string1 = "text1 you wish to find in malware"
  $text_string2 = "text2 you wish to find in malware"

  $hex_string1 = {hex1 you wish to find in malware}
  $hex_string2 = {hex2 you wish to find in malware}

  $reg_exp_string1 = /Perl like regular expressions1
you wish to find in malware/
  $reg_exp_string2 = /Perl like regular expressions2
you wish to find in malware/

condition:
  $text_string1 or $text_string2 or
  $hex_string1 or $hex_string2 or
  $reg_exp_string1 or $reg_exp_string2
}
```

Fig. 1. Structure of YARA Rules

```
rule WannaCry
{
meta:
  description = "Generic Signature of WannaCry"
  author = "Nitin Naik"
  date = "01/06/2018"
  reference = "www.mydomain.com"

strings:
  $text_string1 = "encrypt"
  $text_string2 = "bitcoin"

  $hex_string1 = {B6 D3 56 A5 78 43}
  $hex_string2 = {E8 27 F9 83 C4 82}

  $reg_exp_string1 = /md5: [0-9a-fA-F]{32}/
  $reg_exp_string2 = /state: (on|off)/

condition:
  $text_string1 or $text_string2 or
  $hex_string1 or $hex_string2 or
  $reg_exp_string1 or $reg_exp_string2
}
```

Fig. 2. Example of YARA Rules

B. Fuzzy Rules

Fuzzy logic is a superset of propositional or Boolean logic which is extended to represent the degree of truth/membership in the range of 0 (false/non-membership) and 1 (true/full-membership), and is shown by comparing the fuzzy set and crisp set in Fig. 3. Fuzzy rules are the core component of any fuzzy system that articulates the knowledge of that system in fuzzy logic [6]. A fuzzy rule is written as an If-Then rule in the form of: If antecedent(s) Then consequent(s), where antecedent and consequent are fuzzy propositions that contain linguistic variables. For example a descriptive fuzzy rule can be written as:

$$If\ x\ is\ A\ and\ y\ is\ B\ Then\ z\ =\ C \quad (Mamdani\ Fuzzy\ Rule\ [7]) \quad (1)$$

$$If\ x\ is\ A\ and\ y\ is\ B\ Then\ z\ =\ f(x,y) \quad (Takagi-Sugeno\ Fuzzy\ Rule\ [8]) \quad (2)$$

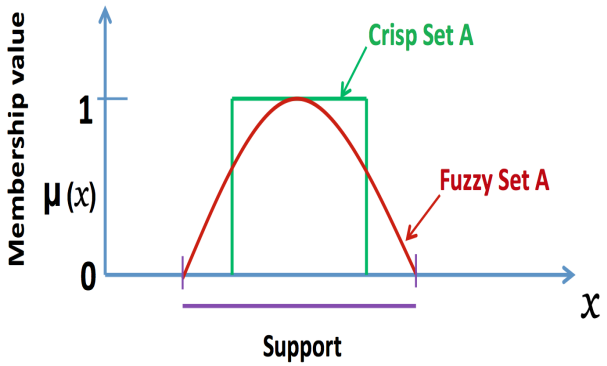

Fig. 3. Fuzzy Set and Crisp Set

These rules contain two inputs x and y (antecedents) and an output z (consequent), two fuzzy sets A and B in the antecedent and a fuzzy set C in the consequent. Fuzzy rules mimic human thinking and are based on human experience. These rules are derived by experts in the specific area or from the collected dataset [6]. Fuzzy rule-based systems can manage imprecise and incomplete data and include a broad range of conditions, which may not be possible in Boolean logic [9]. Consequently, fuzzy rules are the most effectual mechanism to resolve conflict in multiple criteria conditions and assessing the most proficient option accordingly [10]. Additionally, these rules are readily customisable similar way to that of YARA rules.

C. Fuzzy Hashing

Cryptographic hash and fuzzy hash techniques are utilised in security analysis in an attempt to detect malware when investigating both the integrity and similarity of files of interest. Of these two techniques it is the similarity which is of greater importance as malware developers base their code on previous examples leading to the development of new strains [11]. In fuzzy hashing analysis, the file of interest is divided into multiple blocks and a hash value is calculated for each block, with the final step being the concatenation of all hash values of the blocks to generate the fuzzy hash value as shown in Fig. 4. A number of factors affect the length of the fuzzy hash value, including the block size, the size of the file and the output size of the selected hash function [12]. Fuzzy hashing methods can be classified into several categories: Context-Triggered Piecewise Hashing (CTPH), Statistically-Improbable Features (SIF), Block-Based Hashing (BBH) and Block-Based Rebuilding (BBR) [13], [14], [15]. Forensic analysis of malware requires a thorough understanding of the degree of similarity between known malware samples and inert files in coming to a conclusion. This is especially important when considering the triaging and clustering of suspected malware in order to identify new variants. As a result the use of the similarity preserving property of fuzzy hashing is useful in forensic investigation when comparing unknown files with known malware families for their triage and clustering, where samples have the same functionality, yet different cryptographic hash values [2].

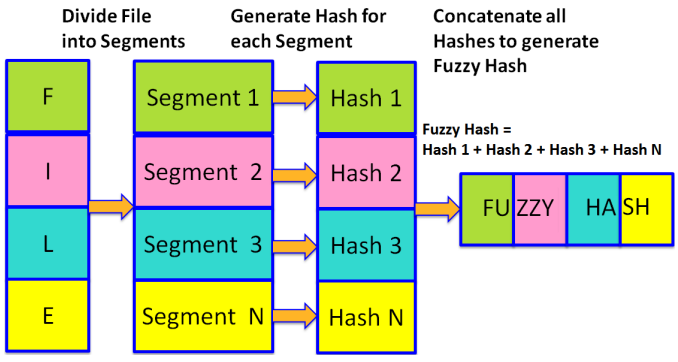

Fig. 4. Generation of Fuzzy Hash Value in Fuzzy Hashing Method

III. Proposed Methodology for Embedding Fuzzy Rule with YARA Rules

A. Embedding Approach

YARA rule conditions are used to determine whether to trigger a YARA rule or not, however, YARA rule conditions are Boolean expressions mostly focused on the binary outcome of the malware analysis. This proposed approach focuses on this aspect to optimise the performance of YARA rules during the execution phase, by extending the rule triggering condition of *String Matching* and the addition of a *Fuzzy Hash Matching* condition, to demonstrate an initial concept of embedding (see Fig. 5). However, it can be customised in a more complex way for a number of parameters, multiple conditions and op-codes depending on the specific requirement of the malware analysis. Almost all standard YARA rules rely on the most common *String Matching* to trigger the rule, if the string matching count is greater than the decided threshold in the rule, then the rule will be triggered and the sample is flagged as malware. However, if the string matching count is less than the set threshold condition, then the rule is not triggered and the sample is not flagged as malware. Despite the rule not being triggered, it generated valuable information but it was simply not collected or used.

This proposed method is designed to collect and use such uncollected and unused information of YARA rules by supplementing fuzzy rules, especially in the event of no rule being triggered. This embedding of fuzzy rules can complement YARA rules generating an improved indication where the YARA rules simply do not produce an alert due to the limitations of Boolean combinatorics. As discussed earlier, fuzzy rules are more effective when working with complex multiple conditions, therefore, another additional condition of *Fuzzy Hash Matching* is combined with the default *String Matching* condition of YARA rules to demonstrate the use of multiple conditions, and optimising the overall performance within this scenario. Fuzzy hash is a compact and effective mechanism to find structural similarity within malware samples, which produces a similarity result as a percentage [16], [17], [18]. Another advantage of the proposed method is that fuzzy rules can produce a degree of similarity which is not possible in standard YARA rules. The combination of these two conditions for YARA rules leads to several alternative outcomes, which can be efficiently managed with fuzzy rules to produce the best possible combined results. The logical approach for this implementation is shown using the pseudocode in the Algorithm 1. This approach is easily adaptable, as YARA rules are fully customisable according to the specific requirement, as is fuzzy rules.

B. Development of Fuzzy Rules

The two most notable benefits of fuzzy rules for YARA rules and this proposed embedded approach are: they can utilise a value range for any parameter (based on the degree of membership) instead a binary value, and combine multiple parameters and their conditions to produce one approximated

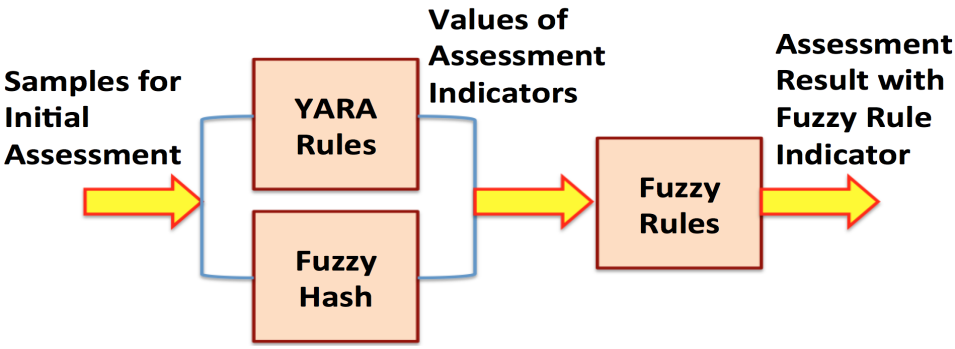

Fig. 5. Embedding of Fuzzy Rules with YARA Rules

Algorithm 1: Pseudocode to determine the use of Fuzzy Rules with YARA Rules

$\mathbb{S}$, Set of Samples for Investigation
$\mathbb{R}$, Set of YARA Rules
\$, Set of Strings in a YARA Rule
$\mathbb{F}$, Set of Fuzzy Hashes of Known Malware
F, Fuzzy Hash Value
β, YARA String Count; β_T, Threshold
δ, Fuzzy Hash Similarity; δ_T, Threshold
Δ, Degree of Similarity
C, Counter for Matched Strings

```
for (i = 1; i < |𝕊|; i++) do
    for (j = 1; j < |ℝ|; j++) do
        for (k = 1; k < |$|; k++) do
            if $_k ∈ 𝕊_i then
                C_{i,j}++
        if Σ_{k=1}^{|$|} C_{i,j} ≥ β_T OR Δ(F_{𝕊_i}, F_l) ≥ δ_T [F_l ∈ 𝔽] then
            return YARA Rule
    if Σ_{j=1}^{|ℝ|} C_i ≥ β_min OR Δ(F_{𝕊_i}, F_l) ≥ δ_min [F_l ∈ 𝔽] then
        if Σ_{j=1}^{|ℝ|} C_i ≥ β_T then
            return YARA Rule
        else
            return Fuzzy Rule
```

output. The proposed embedded approach extends the rule triggering condition of *String Matching* and adds another additional condition of *Fuzzy Hash Matching*, therefore, corresponding to these two conditions, two fuzzy input variables called YARA String Count (YSC) and Fuzzy Hash Similarity (FHS) are derived respectively. The fuzzy output variable called Fuzzy Rule Indicator (FRI) is derived from the two fuzzy input variables based on the Mamdanis inference method [7]. The three fuzzy sets Low, Medium and High for the first fuzzy input variable YSC - β are created in the range of β_{min} = 1 to β_{max} = |\$| (total number of strings in a YARA rule) and divided as shown in Fig. 6. Similarly, the three fuzzy sets Low, Medium and High for the second fuzzy input variable FHS - δ are created in the range of δ_{min} = 10 to δ_{max} = 100 (fuzzy similarity range in percentage) and divided as shown in Fig. 7. Finally, the fuzzy output variable is divided into three fuzzy sets Less Likely Malware, Likely Malware, and Most Likely Malware in the range of 1 to 100 as shown in Fig. 8, to display the appropriate result using fuzzy rules. The sample of fuzzy rules developed for the experimentation are illustrated below.

Fuzzy Rules

If YSC is Low AND FHS is Low
THEN FRI is Less Likely Malware
If YSC is Low AND FHS is Medium
THEN FRI is Likely Malware
If YSC is Medium AND FHS is Low
THEN FRI is Likely Malware
If YSC is Medium AND FHS is Medium
THEN FRI is Likely Malware
If YSC is Low AND FHS is High
THEN FRI is Most Likely Malware
If YSC is Medium AND FHS is High
THEN FRI is Most Likely Malware
If YSC is High AND FHS is High
THEN FRI is Most Likely Malware
If YSC is High AND FHS is Low
THEN FRI is Most Likely Malware
If YSC is High AND FHS is Medium
THEN FRI is Most Likely Malware

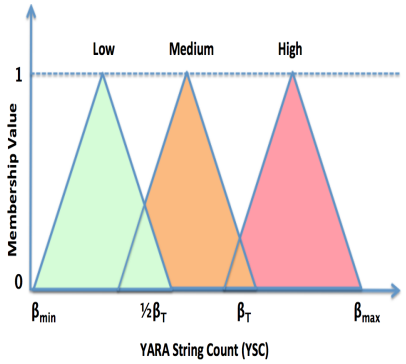

Fig. 6. Generic Fuzzy Input Variable - YARA String Count (YSC) and its Fuzzy Sets

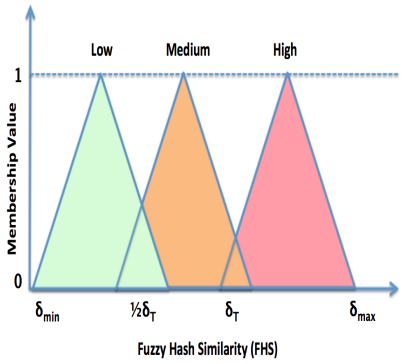

Fig. 7. Generic Fuzzy Input Variable - Fuzzy Hash Similarity (FHS) and its Fuzzy Sets

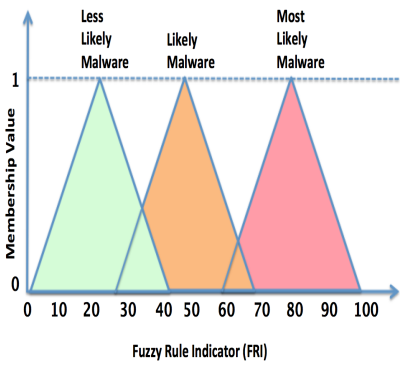

Fig. 8. Generic Fuzzy Output Variable - Fuzzy Rule Indicator (FRI) and its Fuzzy Sets

IV. Experimental Analysis

A. Malware Collection

To test and analyse the proposed embedded approach, one of the most prevalent forms of malware, ransomware was selected. Numerous types of ransomware were created and used in cyberattacks, though, some ransomware categories were worthy of more focus due to their severity of attacks and financial loss. Based on primary research, four ransomware categories were targeted for this work WannaCry, Locky, Cerber and CryptoWall [19], [20], [21]. Thousands of malware samples were downloaded from the two sources *Hybrid Analysis* [22] and *Malshare* [23]. Later, these samples were verified for their credibility as numerous samples were just bogus samples. It was critical to select only credible samples of a specific category as a ground truth to test this proposed embedding method successfully. These samples were investigated based on the information available on *VirusTotal* [24]. To determine that every sample was indeed genuine malware or ransomware and belonged to a specific ransomware category, the criteria was set that it must be identified as malware by at least 40 or more detection engines on *VirusTotal*. To check the ransomware category of collected samples, their category from WannaCry, Locky, Cerber and CryptoWall was verified manually on the recognized detection engines on *VirusTotal*. This sample collection and verification process was very lengthy and time consuming. Eventually, 1000 ransomware samples were selected out of several thousand samples, and equally divided (250 samples each) into four ransomware categories WannaCry, Locky, Cerber and CryptoWall. The four different categories of ransomware were chosen to avoid any biasing in favour of one type of sample or pattern.

B. Experiments

Here three types of YARA rules are tested on the collected ransomware samples and their detection success rate are compared: Standard YARA Rules, Enhanced YARA Rules and Embedded YARA Rules.

1) Standard YARA Rules: In the first experiment, the standard YARA rules were generated for the four ransomware categories by using the *yarGen* tool [25], [26]. This *yarGen* tool generates two types of rules simple rules and super rules, depending on malware types by utilising some intelligent techniques such as Fuzzy Regular Expressions, Naive Bayes Classifier and Gibberish Detector [26]. These generated standard YARA rules contained up to 20 Indicator of Compromise (IoC) strings based on their highest scores, which is the default setting of the *yarGen* tool. Later, the similarity detection rate of standard YARA rules was computed for all four ransomware categories as shown in Table I, which generated the overall malware analysis result of 62.2% (detection success rate) as shown in Fig. 9. However, there is a caveat here as these standard YARA rules were generated by *yarGen* with its default settings, it means different YARA tools may generate different rules which might produce different results. Furthermore, if the number of strings and attributes are increased or decreased then it may also change the analysis results.

2) Enhanced YARA Rules: In the second experiment, the standard YARA rules were modified and enhanced YARA rules were created utilising the fuzzy hashing method (SSDEEP) [3], and their similarity detection rate was computed for all four ransomware categories as shown in Table I, which generated the overall malware analysis result of 67.1% (detection success rate) as shown in Fig. 9. This analysis result of enhanced YARA rules was a moderate improvement (4.9%) as compared to standard YARA rules, due to the fuzzy hashing detection mechanism, which attempts to find the structural similarity in an entire file, rather than only selected strings. This different detection mechanism was complementary to YARA rules without affecting its performance significantly. However, the analysis result of enhanced YARA rules was still not satisfactory and requires further improvement.

3) Embedded YARA Rules: In the third experiment, the standard YARA rules were modified and embedded YARA rules were created utilising a fuzzy hashing method (SSDEEP) and made suitable for embedding with fuzzy rules. Subsequently, the similarity detection rate of embedded YARA rules with fuzzy rules was computed for all four ransomware categories as shown in Table I, which generated the overall malware analysis result of 73.5% (detection success rate) as shown in Fig. 9. This included the results based on the two

fuzzy categories Likely Malware and Less Likely Malware, which were not possible using standard or enhanced YARA rules alone. This analysis result of embedded YARA rules was again a moderate improvement (6.4%) as compared to the enhanced YARA rules using only fuzzy hashing. However, the overall improvement in similarity detection rate was noteworthy (11.3%) as compared to the standard YARA rules. These three experimental results show that embedded YARA rules with fuzzy rules can produce slightly better results due to its capability to detect malware below the set threshold conditions in the standard YARA rules. Thus, this approach can optimise the performance of YARA rules, irrespective of how they are generated and does not require any additional rule optimisation process.

TABLE I
COMPARISON OF SIMILARITY DETECTION RESULTS OF EMBEDDED YARA RULES WITH STANDARD YARA RULES AND ENHANCED YARA RULES FOR WANNACRY, LOCKY, CERBER AND CRYPTOWALL RANSOMWARE SAMPLES

Detection Rate for Particular Ransomware Category	**Standard YARA Rules* Similarity Detection Rate**	**Enhanced YARA Rules (with Fuzzy Hash) Similarity Detection Rate**	**Embedded YARA Rules (with Fuzzy Hash and Fuzzy Rules) Similarity Detection Rate**
WannaCry Ransomware Samples	89.6%	93.2%	95.2%
Locky Ransomware Samples	54.4%	59.6%	65.6%
Cerber Ransomware Samples	77.2%	77.2%	82.8%
CryptoWall Ransomware Samples	27.6%	38.4%	50.4%

Standard YARA Rules*: These rules are generated by **yarGen** tool utilising machine learning methods Fuzzy Regular Expressions, Naive Bayes Classifier and Gibberish Detector, where simple rules contain up to the 20 highest scored strings.

C. Benefits of the Proposed Embedded Approach

- Fuzzy rules can combine multiple parameters and their complex conditions to produce one approximated output.
- In addition to alerting samples as malware by YARA rules, fuzzy rules reveal the degree of similarity of malware (Less Likely Malware, Likely Malware, and Most Likely Malware).
- It can help security experts in analysing or classifying samples based on their fuzzy membership results to apply appropriate actions on specific groups without a further deep dive into the samples.
- Fuzzy hashing can complement YARA rules as it attempts to find structural similarity between the two files in their entirety, rather than focussing on specific strings which may not be found in the sample. Thus, it can still be triggered by fuzzy rules and labelled as possible malware.
- Another benefit of embedding approach is a possible indication of whether samples are unlikely to be malware, if they are not flagged by both YARA rules and fuzzy hash.

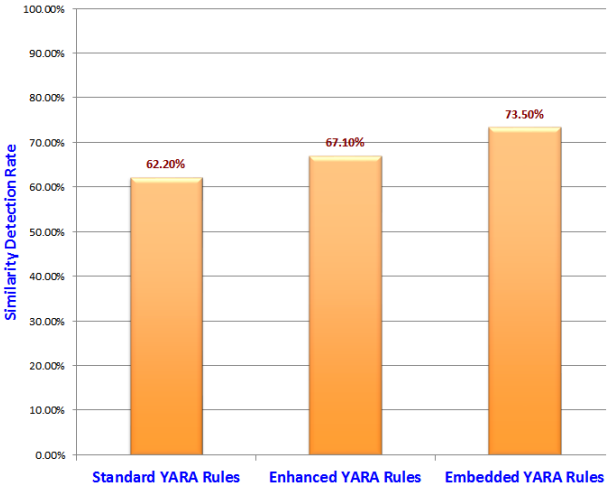

Fig. 9. Overall Similarity Detection Rate of Standard YARA Rules, Enhanced YARA Rules (with Fuzzy Hash) and Embedded YARA Rules (with Fuzzy Hash and Fuzzy Rules)

V. CONCLUSION

This paper presented the principle of embedding fuzzy rules with YARA rules to optimise the performance of YARA rules during the execution phase, irrespective of how they are generated. The embedding of fuzzy rules permitted the alerting feature to YARA rules to trigger in the event of a "no result" being produced by YARA rules when used on their own. This proposed approach and corresponding fuzzy rules were developed utilising the two rule triggering conditions of *String Matching* and additionally the *Fuzzy Hash Matching* to demonstrate an initial concept of embedding. The experimental evaluation was based on the collected ransomware corpus consisting of four categories of ransomware WannaCry, Locky, Cerber and CryptoWall. The proposed embedded YARA rules were tested on the collected ransomware samples and its performance was compared against standard and enhanced YARA rules. Importantly, the similarity detection result of embedded YARA rules was compared against the similarity detection results of standard YARA and enhanced YARA rules. The embedded YARA rules produced improved similarity detection results as compared to standard YARA rules and enhanced YARA rules as it could encompass missing conditions of YARA rules and the degree of similarity of malware. This embedded approach is a flexible and adaptable approach and can be customised according to the specific requirement of the malware analysis. Nonetheless, this proposed approach requires more rigorous testing in terms of parameters, conditions, op-codes, large sample size and complexity to confirm its successful implementation in large-scale investigation projects.

ACKNOWLEDGEMENT

The authors gratefully acknowledge the support of *Hybrid-Analysis.com, Malshare.com* and *VirusTotal.com* for this research work.

REFERENCES

[1] VirusTotal. (2019) YARA in a nutshell. [Online]. Available: https://virustotal.github.io/yara/

[2] N. Naik, P. Jenkins, N. Savage, and L. Yang, "Cyberthreat Hunting-Part 1: Triaging Ransomware using Fuzzy Hashing, Import Hashing and YARA Rules," in *IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*. IEEE, 2019.

[3] N. Naik, P. Jenkins, N. Savage, L. Yang, K. Naik, and J. Song, "Augmented YARA rules fused with fuzzy hashing in ransomware triaging," in *IEEE Symposium Series on Computational Intelligence (SSCI)*, 2019.

[4] V. Alvarez. (2019) Writing YARA rules. [Online]. Available: https://yara.readthedocs.io/en/v3.4.0/writingrules.html

[5] Readthedocs. (2019) Writing YARA rules. [Online]. Available: https://yara.readthedocs.io/en/v3.5.0/writingrules.html

[6] D. Dubois and H. Prade, "What are fuzzy rules and how to use them," *Fuzzy sets and systems*, vol. 84, no. 2, pp. 169–185, 1996.

[7] E. H. Mamdani and S. Assilian, "An experiment in linguistic synthesis with a fuzzy logic controller," *International journal of man-machine studies*, vol. 7, no. 1, pp. 1–13, 1975.

[8] T. Takagi and M. Sugeno, "Fuzzy identification of systems and its applications to modeling and control," *IEEE transactions on systems, man, and cybernetics*, no. 1, pp. 116–132, 1985.

[9] N. Naik, R. Diao, and Q. Shen, "Dynamic fuzzy rule interpolation and its application to intrusion detection," *IEEE Transactions on Fuzzy Systems*, vol. 26, no. 4, pp. 1878–1892, 2018.

[10] N. Naik, C. Shang, P. Jenkins, and Q. Shen, "Building a cognizant honeypot for detecting active fingerprinting attacks using dynamic fuzzy rule interpolation," *Expert Systems*, p. e12557, 2020.

[11] J. Kornblum, "Identifying almost identical files using context triggered piecewise hashing," *Digital investigation*, vol. 3, pp. 91–97, 2006.

[12] A. Tridgell, "Efficient algorithms for sorting and synchronization," Ph.D. dissertation, Australian National University Canberra, 1999.

[13] F. Breitinger and H. Baier, "A fuzzy hashing approach based on random sequences and hamming distance," in *Annual ADFSL Conference on Digital Forensics, Security and Law. 15*, 2012. [Online]. Available: https://commons.erau.edu/adfsl/2012/wednesday/15

[14] C. Sadowski and G. Levin, "Simhash: Hash-based similarity detection," 2007. [Online]. Available: www.webrankinfo.com/dossiers/wp-content/uploads/simhash.pdff

[15] V. Gayoso Martínez, F. Hernández Álvarez, and L. Hernández Encinas, "State of the art in similarity preserving hashing functions," 2014. [Online]. Available: http://digital.csic.es/bitstream/10261/135120/1/Similarity_preserving_Hashing_functions.pdf

[16] N. Naik, P. Jenkins, N. Savage, and L. Yang, "Cyberthreat Hunting-Part 2: Tracking Ransomware Threat Actors using Fuzzy Hashing and Fuzzy C-Means Clustering," in *IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*. IEEE, 2019.

[17] N. Naik, P. Jenkins, J. Gillett, H. Mouratidis, K. Naik, and J. Song, "Lockout-Tagout Ransomware: A detection method for ransomware using fuzzy hashing and clustering," in *IEEE Symposium Series on Computational Intelligence (SSCI)*, 2019.

[18] N. Naik, P. Jenkins, and N. Savage, "A ransomware detection method using fuzzy hashing for mitigating the risk of occlusion of information systems," in *2019 IEEE International Symposium on Systems Engineering (ISSE)*, 2019.

[19] K. Savage, P. Coogan, and H. Lau, "The evolution of ransomware - Symantec," pp. 1–57, 2015.

[20] Y. Klijnsma. (2019) The history of Cryptowall: a large scale cryptographic ransomware threat. [Online]. Available: https://www.cryptowalltracker.org/

[21] Malwarebytes. (2019) Ransomware. [Online]. Available: https://www.malwarebytes.com/ransomware/

[22] Hybrid-Analysis. (2019) Hybrid Analysis. [Online]. Available: https://www.hybrid-analysis.com/

[23] Malshare. (2019) A free Malware repository providing researchers access to samples, malicious feeds, and YARA results. [Online]. Available: https://malshare.com/index.php

[24] VirusTotal. (2019) Virustotal. [Online]. Available: https://www.virustotal.com/#/home/upload

[25] F. Roth. (2018) yarGen is a generator for YARA rules. [Online]. Available: https://github.com/Neo23x0/yarGen

[26] ——. (2017) How to post-process YARA rules generated by yarGen. [Online]. Available: https://medium.com/@cyb3rops/how-to-post-process-yara-rules-generated-by-yargen-121d29322282