Complex & Intelligent Systems
https://doi.org/10.1007/s40747-020-00233-5

ORIGINAL ARTICLE

Embedded YARA rules: strengthening YARA rules utilising fuzzy hashing and fuzzy rules for malware analysis

Nitin Naik[1] · Paul Jenkins[2] · Nick Savage[2] · Longzhi Yang[3] · Tossapon Boongoen[4] · Natthakan Iam-On[4] · Kshirasagar Naik[5] · Jingping Song[6]

Received: 6 July 2020 / Accepted: 5 November 2020

Abstract
The YARA rules technique is used in cybersecurity to scan for malware, often in its default form, where rules are created either manually or automatically. Creating YARA rules that enable analysts to label files as suspected malware is a highly technical skill, requiring expertise in cybersecurity. Therefore, in cases where rules are either created manually or automatically, it is desirable to improve both the performance and detection outcomes of the process. In this paper, two methods are proposed utilising the techniques of fuzzy hashing and fuzzy rules, to increase the effectiveness of YARA rules without escalating the complexity and overheads associated with YARA rules. The first proposed method utilises fuzzy hashing referred to as enhanced YARA rules in this paper, where if existing YARA rules fails to detect the inspected file as malware, then it is subjected to fuzzy hashing to assess whether this technique would identify it as malware. The second proposed technique called embedded YARA rules utilises fuzzy hashing and fuzzy rules to improve the outcomes further. Fuzzy rules countenance circumstances where data are imprecise or uncertain, generating a probabilistic outcome indicating the likelihood of whether a file is malware or not. The paper discusses the success of the proposed enhanced YARA rules and embedded YARA rules through several experiments on the collected malware and goodware corpus and their comparative evaluation against YARA rules.

Keywords Malware analysis · YARA rules · Fuzzy rules · Fuzzy logic · Fuzzy hashing · Cybersecurity · Ransomware · Indicator of compromise · IoC string

Introduction

YARA is an established malware analysis technique, discovering malware based on their strings and signature matching [47]. YARA rules are written based on reverse engineering malware families and finding Indicator of Compromise (IoC) strings. YARA rules are very effective due to their customisable features by which any individual or enterprise can develop their own rules as per their requirement for target-

✉ Nitin Naik
n.naik1@aston.ac.uk

Paul Jenkins
paul.jenkins@port.ac.uk

Nick Savage
nick.savage@port.ac.uk

Longzhi Yang
longzhi.yang@northumbria.ac.uk

Tossapon Boongoen
tossapon.boo@mfu.ac.th

Natthakan Iam-On
natthakan@mfu.ac.th

Kshirasagar Naik
snaik@uwaterloo.ca

Jingping Song
songjp@swc.neu.edu.cn

1 School of Informatics and Digital Engineering, Aston University, Birmingham, UK

2 School of Computing, University of Portsmouth, Portsmouth, UK

3 Department of Computer and Information Sciences, Northumbria University, Newcastle upon Tyne, UK

4 Center of Excellence in AI and Emerging Technologies, School of Information Technology, Mae Fah Luang University, Chiang Rai, Thailand

5 Department of Electrical and Computer Engineering, University of Waterloo, Waterloo, Canada

6 Software College, Northeastern University, Shenyang, China

Published online: 23 November 2020

ing specific attacks and security threats [23]. However, an inappropriate type or number of IoC strings could significantly affect the effectiveness and performance of YARA rules by impeding the malware analysis process [8]. Furthermore, the efficacy of the analysis is dependent on the relevance and number of rules, ensuring there is sufficient rules to perform the analysis, as if the applied YARA rules are unfitting, inadequate or excessive then it may affect the outcome and performance of the malware analysis negatively [15,28]. There are two common routes to improve YARA rules and its effectiveness: (A) Generating extensive YARA rules and (B) Generating optimised YARA rules.

Generating extensive YARA rules and associated issues

One of the most common and simplest ways to improve YARA rules is to increase the number of IoC strings by extending the search criteria. This will generate more extensive YARA rules covering a greater range of indicators assisting YARA rules to discover malware more effectively. However, generation of effective YARA rules using such a method may have some serious issues, for example:

1. Increasing a large number of IoC strings can gradually cause YARA rules to become more cumbersome and complex, which can adversely affect the performance of YARA rules [2,8].
2. Generating extensive YARA rules requires security expertise and is a difficult task for ordinary users whether by writing such YARA rules manually or modifying automatically generated rules, both styles [9,15].
3. Including a large number of IoC strings might match a large number of malware samples, including benign ones, thus increasing the number of false positive [11,15].

Consequently, discovering a simpler way to make YARA rules more efficient without increasing the IoC strings significantly is a laudable aim. The first technique proposed, called enhanced YARA rules, in which a fuzzy hashing technique is used to enhance YARA rules instead of increasing the number of IoC strings. This can improve the effectiveness of YARA rules without significantly increasing the complexity and overheads of YARA rules. The proposed method utilises an additional fuzzy hash function alongside basic YARA rules, thus both techniques complement each other, so that when one method cannot find a match, then the other can and vice versa. The experimental result demonstrates that the proposed enhanced YARA rules produce improved results in comparison with basic YARA rules.

Generating optimised YARA rules and associated issues

The majority of proposed methods to optimise YARA rules are focused on generating effective rules utilising some intelligent mechanisms. However, generation of effective YARA rules using such methods may be computationally complex or may not address several common issues related to its execution and outcome [9,11,15], which could be crucial in improving the effectiveness of YARA rules while utilising IoC strings, for example:

1. YARA rules may not identify a sample as malware even on matching with several strings in any rule, as it may be 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, as it may be below the set threshold of the condition in all the rules. However, the sum of the matched strings in all the rules could be much higher than the set threshold of the condition in any one 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).

All these issues are related to the execution of YARA rules, although not its generation; therefore, it is difficult to manage these issues through the optimised rule generation mechanism. However, all these issues can be resolved if all the information generated by YARA rules during the execution phase are collected and utilised effectively, which may be generally ignored or lost. Therefore, there is a requirement to discover a mechanism to capture such ignored or lost information and utilising it effectively to improve the successfulness of YARA rules. This paper proposes a second technique called embedded YARA rules, in which all the information generated by YARA rules during the execution phase is captured and utilised by fuzzy rules to enhance YARA rules instead of focusing on rule optimisation. The captured information is used in the fuzzy rule-based system to generate more useful and comprehensive outcomes, which is usually not possible using basic YARA rules and enhanced YARA rules individually. The benefit of using fuzzy rules is that it can complement YARA rules for several fuzzy operations and improve the effectiveness of existing YARA rules without requiring any intelligent mechanism in the rule generation process. The experimental result demonstrates that the proposed embedded YARA rules produce better results

in comparison with basic YARA rules and enhanced YARA rules.

This paper is organised into the following sections: "Background: techniques employed for malware analysis" discusses the chosen malware analysis techniques YARA rules, fuzzy hashing and import hashing. "Related work: recentmalware analysis techniques applied to ransomware" presents the malware (ransomware) related work and discusses some of the recently proposed static and dynamic detection method for ransomware. "Data collection: collection of Malware (Ransomware) and goodware samples" explains the collection and verification process of chosen malware (ransomware) samples including goodware samples. "Experimental evaluation of employed techniques: malware analysis using fuzzy hashing, import hashing and YARA rules" presents the experimental evaluation of the selected methods: fuzzy hashing, import hashing and YARA rules on the collected malware and goodware. "Proposed technique-I:malware analysis using proposed enhanced YARA rules" discusses the first proposed technique enhanced YARA rules and its testing results on the collected malware and goodware. "Proposed technique-II:malware analysis using proposed embedded YARA rules" discusses the second proposed technique embedded YARA rules and its testing results on the collected malware and goodware. "Advantages and limitations of the proposed technique" presents some of the main advantages and limitations of the proposed embedded YARA rules. Lastly, "Conclusion" presents the summary of the research work and suggests some future work.

Background: techniques employed for malware analysis

YARA rules

YARA rules are developed to detect malware by primarily matching its signatures/strings with the existing malware signatures/strings [33,47]. These rules contain predetermined signatures/strings related to known malware used in attempting to match against the targeted files, folders, or processes [32]. YARA rules consist of three sections: meta, strings and condition as shown in Fig. 1. Here, strings can be classified into three types of strings: text strings, hexadecimal strings and regular expression strings (see Fig. 1). Text strings are generally a readable text complemented with some modifiers (e.g., nocase, ASCII, wide, and fullword) to manage the process more effectively [1]. Hexadecimal strings are a sequence of raw bytes complemented with three flexible formats: wildcards, jumps, and alternatives [1]. Regular expression strings are similar to text strings as a readable text complemented with some modifiers, which are available since version 2.0 and increases the capability of YARA rules [1].

```
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 = /regular expressions1 you wish to find in malware/
  $reg_exp_string2 = /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
}
```

```
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. 1 YARA rules: syntax and example

Text strings and regular expression strings which 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 required matching with the target to alert the sample as malware [36]. YARA conditions determine whether to trigger the rule or not, however, these conditions are Boolean expressions similar to those used in a number of computer programming languages [1]. 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 efficiently on their own.

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. When considering these techniques, the behaviour of malware creators is worth reflecting upon, as many use existing malware as the basis of their new strain creation, and it is this behaviour which determines the importance of the different techniques. Therefore, of these two techniques, it is the similarity which is of greater importance [18]. In a fuzzy hashing technique, the file of interest is split into several blocks and each block is treated separately in calculating its hash, finally, hashes of all the blocks are concatenated to obtain the fuzzy hash of that file (see Fig. 2). A number of factors affect the size of the fuzzy hash of a file, comprising of the block size, the size of the file and the output size of the chosen hash function [45]. Fuzzy hashing methods are divided into different types namely: Context-Triggered Piecewise Hashing (CTPH), Statistically Improbable Features (SIF), Block-Based Hashing

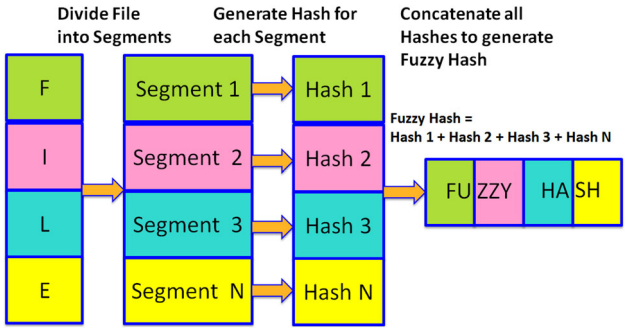

Fig. 2 Fuzzy hash generation process in a fuzzy hashing method

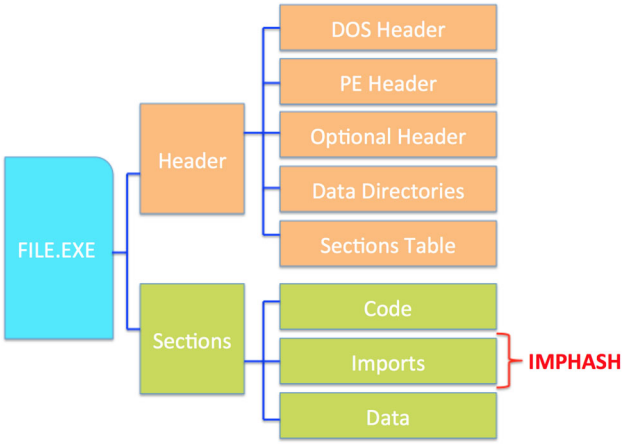

Fig. 3 Import hash (IMPHASH) is generated based on the Import Address Table (IAT) in a portable executable (PE) File

(BBH) and Block-Based Rebuilding (BBR) [5,12,41]. Forensic analysis of malware requires a thorough knowledge of the degree of similarity between known malware and inert files to assess files for their threat potential [35]. This is especially important when considering the analysis and clustering of suspected malware in order to discover new variants [30]. As a result, the use of the similarity preserving property of fuzzy hashing is useful in malware analysis while comparing unknown files with known malware families during malware analysis, where samples possess the similar functionality, yet different cryptographic hash values [28,29].

SSDEEP

The SSDEEP fuzzy hashing technique was specially created to distinguish spam or junk emails [18]. It splits a file into several blocks depending on the data given in the file. These blocks and their endpoints are created by employing Adler32 function involved in a rolling hash method [45]. Subsequently, a hash is created for each block and finally, hashes of all the blocks are concatenated to obtain the fuzzy hash of that file. The Damerau–Levenshtein distance measure is used to compute the similarity distance of concerning files.

SDHASH

The SDHASH fuzzy hashing technique discovers common and uncommon attributes in a file and matches the uncommon attributes with those in another file to find the degree of similarity of concerning files [40]. Normally an attribute is a 64-byte string and is detected based on the calculation of entropy. The SDHASH fuzzy hash of a file is computed by employing SHA-1 hash function and Bloom filters. A Bloom filter is a probabilistic and space-efficient data structure used to establish that an element is a member or not a member of the set. The Hamming distance measure is used to compute the similarity distance of concerning files.

mvHASH-B

The mvHASH-B fuzzy hashing technique focuses on preserving the data unchanged in the case where there is a minor change between files, it ensures the same hash value while preserving the similarity. Nonetheless, mvHASH-B uses the concept of majority voting to transform the input data, encoding the majority vote bit sequence with Run-Length Encoding (RLE), and finally generating the mvHASH-B fuzzy hash employing Bloom filters [4]. Furthermore, it employs its own outlined hash function which is comparable with the standard SHA-1 function and having better run time efficiency than it.

Import hashing

Import hashing is one of the fastest analysis techniques used to determine the similarity of two malicious programs [22]. Unlike other hashing techniques, which generate hash of a complete file, import hashing only generates a hash of a small part (i.e. Imports or Import Libraries) of a Portable Executable (PE) file (see Fig. 3). Imports/Import Libraries are simply functions called by a program (here, malware) from other programs, which are to be bound and linked with the program to build the final executable program [22]. The details of Imports/Import Libraries (DLLs) are maintained in the Import Address Table (IAT), which is the basis of the generation of IMPort HASH (IMPHASH) of a program, where the order in which the Import Libraries are called determines the value of the generated IMPHASH. Thus, two programs that were compiled with similar code except with a different order of Import Libraries will generate different IMPHASH values. This method is analogous to fuzzy hashing with regard to its speed, computation, complexity and hash size, however, it is noteworthy that IMPHASH provides

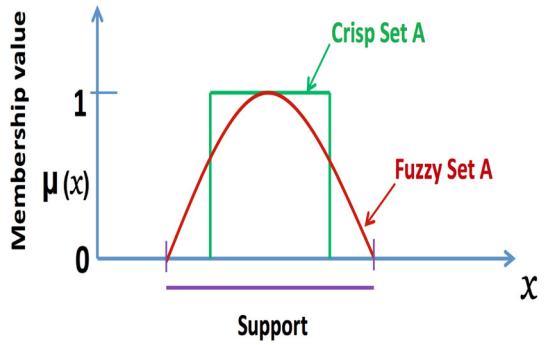

Fig. 4 Fuzzy set and crisp set

a binary similarity result, rather than the degree of similarity of two files.

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), which is shown by comparing fuzzy set and crisp set in Fig. 4. Fuzzy rules are the core component of any fuzzy system that articulates the knowledge of that system in fuzzy logic [10]. 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:

$$If\ x\ is\ A\ and\ y\ is\ B\ Then\ z\ =\ C \quad (Mamdani\ Fuzzy\ Rule\ [21]) \tag{1}$$

$$If\ x\ is\ A\ and\ y\ is\ B\ Then\ z\ =\ f(x,y) \quad (Takagi-Sugeno\ Fuzzy\ Rule\ [44]) \tag{2}$$

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 of the specific area or from the collected dataset [10]. 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 [24]. Consequently, fuzzy rules are the most effectual mechanism to resolve conflict in multiple criteria conditions and assessing the most proficient option accordingly. Additionally, these rules are readily customisable similar to YARA rules.

Related work: recent malware analysis techniques applied to ransomware

Numerous detection techniques for malware have been proposed previously, however, very few techniques focused on ransomware. Recently, some static and dynamic analysis methods have been proposed to detect ransomware, which are discussed here. The dynamic analysis method UNVEIL checks any tampering activity related to files and data [16]. Similarly, a dynamic machine learning method ElderRan monitors a set of actions performed by applications as signs of ransomware [43]. Another dynamic analysis method HelDroid is particularly designed for mobile devices that detects if an app is attempting to lock or encrypt the device without the user's consent [3]. The dynamic analysis method R-Locker is developed for detecting and preventing ransomware by employing honeyfiles (a FIFO file structure) to block ransomware once it starts reading the file [13].

These dynamic methods are more effective than static methods in detection, however, their disadvantage in detecting ransomware is that they require the successful loading and running of ransomware to demonstrate their expected behaviour or characteristics. If the infected ransomware is able to evade the method due to its silent behaviour for some time or waiting for some specific activity from users, then it is already too late for those affected users. Moreover, depending on the requirement of emulation or virtualization for the dynamic method, several efficient ransomware variants will not perform any action and thus cannot be detected by that method. Furthermore, if the detection policy of the dynamic method is matched with the activity of goodware then that dynamic method may generate several false positives.

Alternatively, a static analysis method can safely examine the ransomware samples without running them and affecting the data and user, leading to a safer analysis of malware. However, most static analysis and detection methods [6,7,37] require greater computational overheads and suffer from false positives and negatives. The proposed methods for detecting and predicting ransomware are a static analysis method, which require fewer computational overheads and performs rapid comparative analysis, in comparison to other static analysis methods.

Data collection: collection of malware (ransomware) and goodware samples

In this implementation, one of the most prevalent malwares, ransomware was selected to perform all analysis and evaluating the effectiveness and performance of the proposed techniques. Ransomware was selected for the experiment as

it is one of the most relevant and damaging malware that exploits victims for financial gain, business disruption and market share. Numerous types of ransomware were created and used in cyberattacks, though, some ransomware categories were worthy of greater focus due to their historical significance, severity of attack and financial loss. Based on primary research, four ransomware categories were targeted for this work WannaCry, Locky, Cerber and CryptoWall [17,20,37,42]. Thousands of malware samples were acquired from the two sources *Hybrid Analysis* [14] and *Malshare* [19]. Later, these samples were verified for their credibility as numerous samples were simply fake. It was critical to select only credible samples of a specific category as a reference to test all selected malware analysis methods and the proposed technique successfully. These samples were investigated based on the information available on *VirusTotal* [46]. To determine that every sample was indeed genuine malware or ransomware and were members of a specific ransomware category, the criterion 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 both lengthy and time consuming, leading to 1000 ransomware samples being selected out of several thousand samples, these were equally divided into 250 samples of four ransomware categories WannaCry, Locky, Cerber and CryptoWall. The four different categories of ransomware were chosen to evaluate how each employed and proposed malware analysis method works on the different categories of ransomware.

In addition to the collection of malware (ransomware) samples, equal numbers of goodware samples were collected to balance this analysis. These 1000 goodware samples were the files collected from ten commonly used software: JAVA, MS OFFICE, Google Chrome, MySQL, R, NMAP, McAFee, MATLAB, Python and Snort. These ten different software samples were chosen in such a way that it could cover wide range of benign programs and to evaluate how each employed and proposed malware analysis method works on the different types of benign program. Finally, a total 2000 samples were utilised to perform all the experiments applying all employed and proposed malware analysis methods.

Experimental evaluation of employed techniques: malware analysis using fuzzy hashing, import hashing and YARA rules

In this research work, three different malware analysis methods fuzzy hashing, import hashing and YARA rules are employed to perform static analysis on the collected ransomware (WannaCry, Locky, Cerber and CryptoWall) and goodware samples. All three analysis methods are employed to perform static analysis which is generally fast, efficient and resource-optimised [28]. Fuzzy hashing and import hashing are relatively compact, fast and resource-optimised analysis methods [26,27]. In addition to the accuracy of malware analysis results, these criteria are very decisive in determining the appropriate method for the analysis of a large volume of malware [34]. This section discusses the methodology and experiment of each analysis method for the collected ransomware samples. The experiment is aimed at illustrating the similarity detection success rate of each analysis method for each ransomware category separately and collectively. It is expected and most probable that each sample of the same category holds some similarity to other samples in that category. Therefore, experiments evaluate how many samples within one category are matched with at least one other sample of the same category by each analysis method.

Fuzzy hashing: methodology

Fuzzy hashing generates a fuzzy hash value when it is applied on an unpacked ransomware sample. This fuzzy hash value can be matched against either already identified ransomware samples or their fuzzy hash values. If the fuzzy hash of a sample in question matches with any of the pre-identified ransomware samples or its fuzzy hash value then the fuzzy hash result is generated as a degree of similarity between the two [34]. This fuzzy similarity result is presented in the range of 1% (least matched) to 100% (exactly matched), however, it is entirely at the discretion of security experts how this value is interpreted depending on their analysis requirement [34]. Generally, a threshold value can be set to accept or ignore the fuzzy similarity score and to determine as matched or not matched scenarios respectively. The fuzzy hashing should only be used as an initial investigation that may assist in any further analysis but not as a conclusive result [29].

Fuzzy hashing: experiment

The SSDEEP, SDHASH and mvHASH-B fuzzy hashing methods were used to detect similarity for each ransomware category separately. It was important to assess the performance of these three methods in different threshold conditions for comparison purposes; therefore, their similarity detection results were evaluated in four different conditions: (1) when all the fuzzy similarity scores were considered (1–100%), (2) when those fuzzy similarity scores were considered which are greater than 10%, (3) when those fuzzy similarity scores were considered which are greater than 20%, and (4) when those fuzzy similarity scores were considered which are greater than 30% [34]. The four evaluation results for four ransomware categories are presented in

Table 1. One of the most important findings in all four evaluation results is that the results of SDHASH and mvHASH-B fuzzy hashing methods decreased and in some cases quite significantly as the similarity threshold value increased. The detection rate of the SSDEEP fuzzy hashing method is lower, however, consistent in all four experiments. At the final similarity threshold limit of 30%, most SSDEEP results are superior to the other two fuzzy hashing methods. This finding is crucial when utilising these similarity detection results in further analysis as the proposed method and subsequent analysis (e.g., clustering or classification) results will be dependent on them [26,29].

Import hashing: methodology

Import hashing generates an IMPHASH hash value when it is applied on an unpacked ransomware sample. Later, this IMPHASH hash value can be matched against either existing identified ransomware samples or their IMPHASH hash values. If the IMPHASH hash matches with any of the pre-identified ransomware samples or its IMPHASH hash value, then the result is generated as a matched sample with one or more samples. However, it does not provide a degree of similarity, rather a binary output (i.e. either matched or not matched) [34]. The import hashing should only be used as an initial investigation that may aid in any further analysis but not as a conclusive result [28].

Import hashing: experiment

The import hashing method was used to detect similarity for each ransomware category separately. The similarity detection results for all the four ransomware categories are shown in Table 2. The import hashing result is a mixed result when compared with the fuzzy hashing results. In one case, it is somewhat better, however, in other cases, it is slightly lower. It is worth noting that import hashing can only be used on the PE file format, therefore, its effectiveness depends on the type of samples investigated [34].

YARA rules: methodology

Fuzzy hashing and import hashing both generate a hash value of the examined samples and find similarity with the existing malware samples, whereas YARA rules include IoC strings extracted from the existing malware samples to find their similarity with the examined samples. YARA rules are very different from hashing as rule generation requires a reverse engineering process. It requires an in-depth analysis of malware and their family to generate YARA rule(s) for specific malware or their family. Therefore, generation of effectual YARA rules demands effort and expertise, unlike both hashing methods, where untrained personnel can apply the process of hash generation to generate hashes and perform the analysis [34]. YARA rules can be generated manually or automatically, whilst automatic rule generation is easier than the manual process, however, it may require some post-processing operations to optimise them. Here *yarGen* tool [39] is employed to generate the YARA rules for all ransomware samples. This tool generates two types of rules ordinary rules and super rules depending on malware types by utilising some intelligent techniques such as Fuzzy Regular Expressions, Naive Bayes Classifier and Gibberish Detector [38]. All the basic YARA rules generated for ransomware samples contain up to 20 strings based on their highest scores and do not include IMPHASH as it is employed as one of the analysis method in this research work.

YARA rules: experiment

Once YARA rules are generated for all four ransomware categories separately, they are used to detect similarity for each ransomware category separately. The similarity detection results for all four categories are shown in Table 3. The result of YARA rules is a mixed result when compared with the both hashing results, as in two cases it is slightly improved, and in others it is not. However, there is a caveat here as these basic YARA rules were generated by *yarGen* with its default settings, it means different YARA tools may generate different rules which might produce different results [25]. Furthermore, if the number of strings and attributes are increased or decreased then it may change the analysis results. If the number of strings and attributes are significantly increased, then it adversely affects the performance of YARA rules as malware analysis is always performed on a large sample size.

When comparing the results of these three employed methods, each method performed slightly better and slightly worse, and it is difficult to determine the best analysis method for all given scenarios. Consequently, further investigation is required for how to improve the effectiveness of these analysis methods. YARA rules are customisable and contain some advanced features, whereas fuzzy hashing can be a compact and add more value to YARA rules. Therefore, their considered combination may offer some improvements in malware analysis.

Proposed technique-I: malware analysis using proposed enhanced YARA rules

Enhanced YARA rules: methodology

As previously mentioned, IoC strings are a critical component of YARA rules which determines its success, in particular, how many strings and how they are selected for a rule [34]. Conversely, attackers continuously try to identify

Table 1 Similarity detection results of SSDEEP, SDHASH and mvHASH-B fuzzy hashing methods

Fuzzy hashing	WannaCry ransomware			Locky ransomware			Cerber ransomware			Cryptowall ransomware		
Matching criteria	SSDEEP detection rate (%)	SDHASH detection rate (%)	mvHASH-B detection rate (%)	SSDEEP detection rate (%)	SDHASH detection rate (%)	mvHASH-B detection rate (%)	SSDEEP detection rate (%)	SDHASH detection rate (%)	mvHASH-B detection rate (%)	SSDEEP detection rate (%)	SDHASH detection rate (%)	mvHASH-B detection rate (%)
Fuzzy similarity scores (1–100%)	91.2	93.6	90	42	58.4	72.4	33.6	71.2	94.8	28	52.4	83.6
Fuzzy similarity scores > 10%	91.2	93.6	90	42	38.4	64	33.6	62.8	90.4	28	32.8	56.8
Fuzzy similarity scores > 20%	91.2	90	84.4	41.6	35.6	36.4	33.6	37.6	36.8	28	24	20.8
Fuzzy similarity scores > 30%	90.8	90	84.4	41.6	30.4	33.6	33.6	28.4	36	28	20.4	20.4

Table 2 Similarity detection results of import hashing

Ransomware category	Import hashing detection rate (%)
WannaCry ransomware	87.6
Locky ransomware	31.6
Cerber ransomware	61.6
CryptoWall ransomware	27.2

Table 3 Similarity detection results of YARA rules

Ransomware category	YARA rules* detection rate (%)
WannaCry ransomware	89.6
Locky ransomware	54.4
Cerber ransomware	77.2
CryptoWall ransomware	27.6

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

such detection methods and attempt evasion using intelligent modifications in their malware. If only few or none of the selected strings are found in the targeted samples then YARA rules do not flag samples as malware even though they may be malware [34]. Adding a large number of strings in rules may increase the computational complexity and overheads affecting the performance of YARA rules significantly. Moreover, to write such complex YARA rules or modify automatically generated rules, a high degree of expertise is required in the cybersecurity [2,8,9]. Consequently, it is essential to find a simple solution to make YARA rules more efficient without incurring all the complexities stated earlier. This requires exploring some alternative mechanisms other than IoC strings to enhance YARA rules. Fuzzy hashing is a relatively compact, fast and resource-optimised mechanism employed for analysis which may not be effective in isolation; however, it may complement YARA rules enhancing its analysis performance without affecting complexity significantly [28,31]. Fuzzy hashing attempts to find structural similarity between the two files in their entirety in circumstances where the selected IoC strings cannot be found in the sample [34]. Furthermore, fuzzy hashing can provide the degree of similarity of each matched sample alongside the outcome of YARA rules which is not achievable in YARA rules alone [34]. Occasionally, both methods can complement each other in finding a missed opportunity by one of the methods. Therefore, the collective search result of these two methods can increase the accuracy and confidence level of the overall analysis [34]. The logical approach for the implementation of enhanced YARA rules is shown using the pseudocode in Algorithm 1 and Fig. 5.

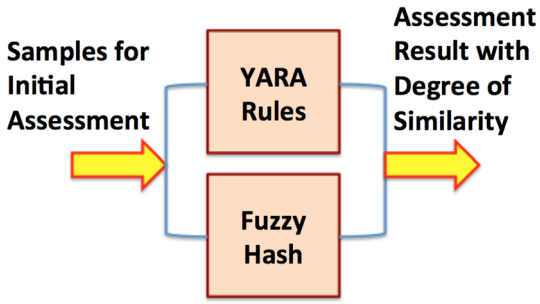

Fig. 5 Fuzzy hashing aided enhanced YARA rules

Algorithm 1: Pseudocode of Enhanced 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_T, YARA String Count Threshold
δ_T, Fuzzy Hash Similarity Threshold
Δ, Degree of Similarity
C, Counter for Matched Strings
for $(i = 1; i < |\mathbb{S}|; i++)$ **do**
  for $(j = 1; j < |\mathbb{R}|; j++)$ **do**
    for $(k = 1; k < |\$|; k++)$ **do**
      if $\$_k \in \mathbb{S}_i$ **then**
        $C_{i,j}++$
    if $\Sigma_{k=1}^{|\$|} C_{i,j} \geq \beta_T \quad OR \quad \Delta(F_{\mathbb{S}_i}, F_l) \geq \delta_T \ [F_l \in \mathbb{F}]$ **then**
      return $YARA\ Rule$

Enhanced YARA rules: experiment

The enhanced YARA rules were developed based on the three employed fuzzy hashing methods SSDEEP, SDHASH and mvHASH-B. These three different fuzzy hashing methods based enhanced YARA rules were evaluated to determine two hypotheses: whether this integration was successful or not, and if successful, then which fuzzy hashing method produced greater accuracy in results [34]. The similarity detection results of enhanced YARA rules utilising three different fuzzy hashing methods for all the four ransomware categories are shown in Table 4. Here, the fuzzy similarity scores greater than 30% were utilised for all the three fuzzy hashing methods. Noticeably, enhanced YARA rules with all the three fuzzy hashing methods showed a minor improvement, but SSDEEP fuzzy hashing contributed to the highest improvement in the overall result of analysis [32,34]. Interestingly, for one ransomware category Cerber, no fuzzy

Table 4 Similarity detection results of enhanced YARA rules utilising fuzzy hashing

Ransomware category	Detection rate of enhanced (%) YARA rules based on SSDEEP (similarity score > 30%)	Detection rate of enhanced (%) YARA rules based on SDHASH (similarity score > 30%)	Detection rate of enhanced (%) YARA rules based on mvHASH-B- (similarity score > 30%)
WannaCry ransomware	93.2	92.8	92
Locky ransomware	59.6	58	58.4
Cerber ransomware	77.2	77.2	77.2
CryptoWall ransomware	38.4	34.8	34.4

Table 5 Comparison of similarity detection results of enhanced YARA rules with SSDEEP, SDHASH, mvHASH-B, IMPHASH and YARA rules

Ransomware category	SSDEEP fuzzy hashing detection rate (%)	SDHASH fuzzy hashing detection rate (%)	mvHASH-B fuzzy hashing detection rate (%)	IMPHASH import hashing detection rate (%)	YARA rules detection rate (%)	Enhanced YARA rules detection rate (%)
WannaCry ransomware	90.8	90	84.4	87.6	89.6	93.2
Locky ransomware	41.6	30.4	33.6	31.6	54.4	59.6
Cerber ransomware	33.6	28.4	36	61.6	77.2	77.2
CryptoWall ransomware	28	20.4	20.4	27.2	27.6	38.4

hashing could improve the result of YARA rules, however, in this case, YARA rules already produced better results than other methods, which compensates the failure of fuzzy hashing [34]. Alternatively, in all those categories where YARA could not produce respectable results, fuzzy hashing assisted improvement in the results, which is crucial for the success of this integration. On the basis of this experimentation, the SSDEEP based result of YARA rules is recommended as the final results of enhanced YARA rules [32,34]. Furthermore, SSDEEP is more compact, faster and a resource-optimised fuzzy hashing method in comparison to the SDHASH and mvHASH-B methods.

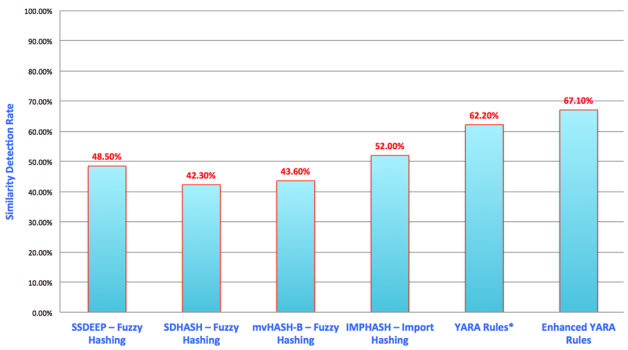

Fig. 6 Comparison of an overall similarity detection rate of enhanced YARA rules with all employed analysis methods

Comparative evaluation of the analysis results of enhanced YARA rules with different analysis methods

Comparison based on similarity detection results

Based on the experiment of enhanced YARA rules, the result of the selected enhanced YARA rules (based on SSDEEP) is compared against the analysis results of all other employed analysis methods as shown in Table 5 and Fig. 6. Here, basic YARA rules performed slightly better than the other two analysis methods fuzzy hashing and import hashing. Nevertheless, the proposed enhanced YARA rules (based on SSDEEP) produced a slightly better result (67.1%) in comparison to the result (62.2%) of basic YARA rules. Despite this improvement is not significant, it still shows the moderate success of the integration of YARA rules and a fuzzy hashing method.

Comparison based on evaluation metrics

For further comparative evaluation of the proposed enhanced YARA rules (based on SSDEEP) with different analysis methods, four evaluation metrics (Accuracy, Precision, Recall and F1-Score) were calculated as shown in Table 6. Here, the overall result of the proposed enhanced YARA rules (based on SSDEEP), except Precision, is better than all other analysis methods. In evaluating the effectiveness of any analysis method decisively, a balance of Precision and Recall is very important, therefore, F1-Score consisting of both may be more helpful in determining a relatively better analysis method. Here, the F1-Score of the proposed

Table 6 Comparison of evaluation metrics for enhanced YARA rules with SSDEEP, SDHASH, mvHASH-B, IMPHASH and YARA Rules

Evaluation metric	SSDEEP fuzzy hashing (%)	SDHASH fuzzy hashing (%)	mvHASH-B fuzzy hashing (%)	IMPHASH import hashing (%)	YARA rules (%)	Enhanced YARA rules (%)
Accuracy	74.25	71.15	71.80	76.00	79.80	83.55
Precision	100	90.19	90.08	100	95.99	96.27
Recall	48.50	42.30	43.60	52.00	62.20	67.10
F1-Score	65.32	57.59	58.76	68.42	75.49	79.08

enhanced YARA rules (based on SSDEEP) is 79.08%, which is better than the F1-Score of all other analysis methods. This confirms that the proposed enhanced YARA rules (based on SSDEEP) are more effective as compared to all other analysis methods.

Proposed technique-II: malware analysis using proposed embedded YARA rules

Embedded YARA rules: methodology

The activation of any YARA rule is dependent on its condition, which is a Boolean expression and mainly utilised for the binary decision, i.e. whether to activate that rule or not [33]. This proposed approach focuses on this aspect of YARA rules to improve the effectiveness of YARA rules during the execution phase. The proposed embedded approach extends the rule triggering condition of *String Matching* and adds another additional condition of *Fuzzy Hash Matching* [33], to demonstrate an initial concept of embedding (see Fig. 7). 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 for malware analysis. Almost all basic YARA rules rely on the most common *String Matching* to trigger the rule, if the string matching count is greater than or equal to the decided threshold in the rule, then the rule will be triggered and the sample is flagged as malware [33]. 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 has produced significant information but it is simply ignored or never utilised [33].

For the effective utilisation of such ignored or unused findings of YARA rules, this proposed approach employs fuzzy rules, especially in the event of no YARA rule being triggered [33]. This embedding of fuzzy rules can complement YARA rules to generate an improved indication where the YARA rules simply do not generate 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

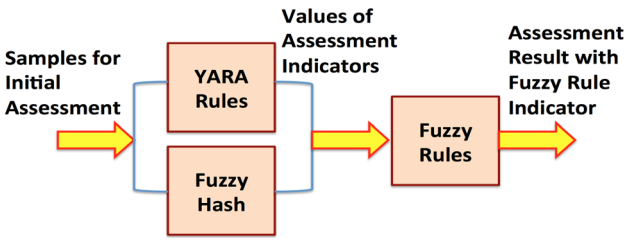

Fig. 7 Fuzzy rules and fuzzy hashing aided embedded YARA rules

Fuzzy Hash Matching is combined with the default *String Matching* condition of YARA rules to demonstrate the use of multiple conditions, and for optimising the overall performance within this scenario [33]. Fuzzy hash is a compact and effective mechanism to find structural similarity of malware samples, which produces a similarity result as a percentage, for ease of understanding [26,27,29]. Another advantage of the proposed method over the use of YARA rules on their own, fuzzy hash can produce a degree of similarity which is not possible in basic 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 the implementation of embedded YARA rules is shown using the pseudocode in Algorithm 2 and Fig. 7. This proposed method can be adapted easily, as YARA rules are fully customisable according to the specific requirement, and thus fuzzy rules.

Embedded YARA rules: development of fuzzy rules

The embedding of fuzzy rules with YARA rules offers several advantages, for example, fuzzy rules employ a value range of a parameter to determine the degree of membership rather than a binary membership, and generate an approximated output based on several values/conditions of several parameters [33]. In this proposed approach, in addition to the rule triggering condition of *String Matching*, another rule triggering condition of *Fuzzy Hash Matching* is added, and related to these two conditions, two fuzzy input variables called YARA String Count (YSC) and Fuzzy Hash Similarity (FHS) are

Algorithm 2: Pseudocode of Embedded YARA Rules

```
𝕊, Set of Samples for Investigation
ℝ, Set of YARA Rules
$, Set of Strings in a YARA Rule
𝔽, Set of Fuzzy Hashes of Known Malware
F, Fuzzy Hash Value
β, YARA String Count; β_T, Threshold
δ, Fuzzy Hash Similarity; δ_T, Threshold
Δ, Degree of Similarity
C, Counter for Matched Strings
for (i = 1; i < |𝕊|; i++) do
    for (j = 1; j < |ℝ|; j++) do
        for (k = 1; k < |$|; 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
```

derived respectively. The fuzzy output variable called Fuzzy Rule Indicator (FRI) is derived from the two fuzzy input variables based on the Mamdani's inference method [21]. The three fuzzy sets Low, Medium and High for the first fuzzy input variable YSC (β) are created in the range of $\beta_{min} = 1$ to $\beta_{max} = |\$|$ (total number of strings in a YARA rule) and divided them as shown in Fig. 8. Similarly, the three fuzzy sets Low, Medium and High for the second fuzzy input variable FHS (δ) are created in the range of $\delta_{min} = 10$ to $\delta_{max} = 100$ (fuzzy similarity range in percentage) and divided them as shown in Fig. 9. Finally, the fuzzy output variable is divided into three fuzzy sets Less Likely Malware, Likely Malware, and Most Likely Malware in the range of 1–100 as shown in Fig. 10, to display the appropriate result using fuzzy rules. The sample of fuzzy rules developed for this experiment is illustrated here. Similarly, customised fuzzy rules can be created for different and new parameters, multiple conditions and op-codes depending on the specific requirement for the specific malware analysis.

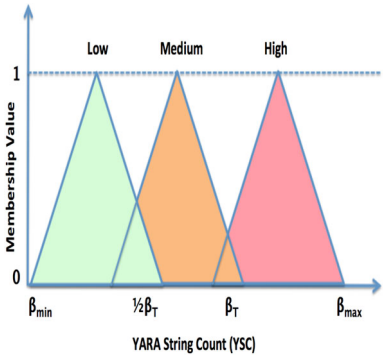

Fig. 8 Generic fuzzy input variable—YARA string count (YSC) and its fuzzy sets

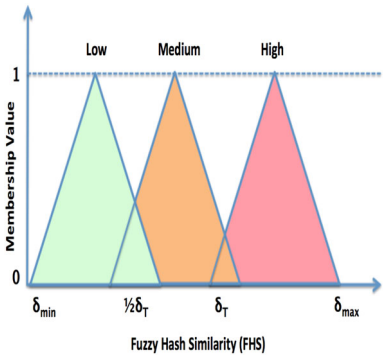

Fig. 9 Generic fuzzy input variable—fuzzy hash similarity (FHS) and its fuzzy sets

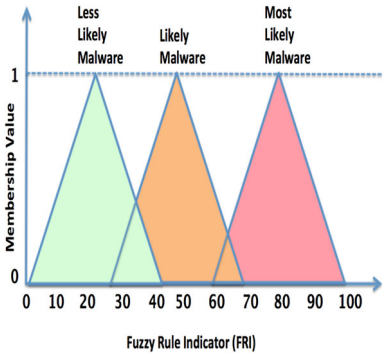

Fig. 10 Generic fuzzy output variable—fuzzy rule indicator (FRI) and its fuzzy sets

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

Embedded YARA rules: experiment

The embedded YARA rules were developed utilising a fuzzy hashing method and fuzzy rules. Later, the similarity detection rate of embedded YARA rules was computed for all four ransomware categories as shown in Table 7, which generated the overall malware analysis result of 73.5% (detection success rate) as shown in Fig. 11. This included the results based on the two fuzzy categories Likely Malware and Less Likely Malware, which were not possible using basic or enhanced YARA rules alone [33]. This analysis result of the proposed 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 basic YARA rules. This experimental evaluation shows that the proposed 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 YARA rules [33]. Thus, this approach can improve the effectiveness of YARA rules, irrespective of how they are generated and does not require any additional rule optimisation process.

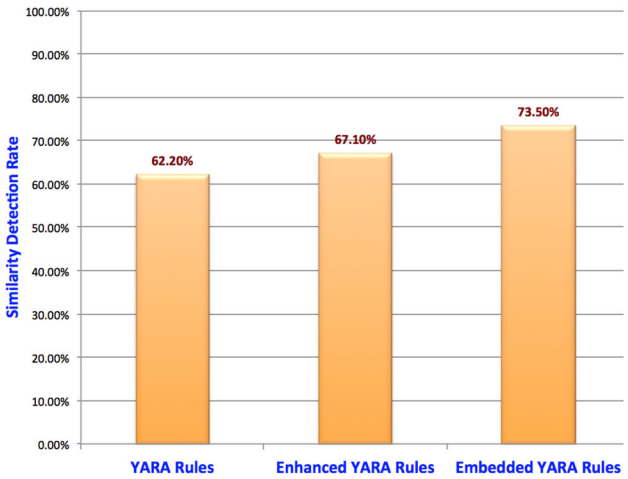

Fig. 11 Comparison of an overall similarity detection rate of YARA rules, enhanced YARA rules and embedded YARA rules

For a more rigorous comparative evaluation of the proposed embedded YARA rules with basic YARA rules and the proposed enhanced YARA rules, four evaluation metrics (Accuracy, Precision, Recall and F1-Score) were calculated as shown in Table 8. Here, the overall result of the proposed embedded YARA rules is better than both basic YARA rules and enhanced YARA rules, thus the values of all four evaluation metrics of the proposed embedded YARA rules are better than their counterpart YARA rules. However, the proposed embedded YARA rules generated false positive and false negatives but fewer in comparison to both basic YARA rules and enhanced YARA rules (see Precision Values and Recall Values respectively in Table 8). For evaluating the effectiveness of any analysis method decisively, a balance of Precision and Recall is very important, therefore, the F1-Score consisting of both may be more helpful in determining a relatively better analysis method. Here, F1-Score of the proposed embedded

Table 7 Comparison of similarity detection results of embedded YARA rules with enhanced YARA rules and YARA rules

Ransomware category	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	89.6	93.2	95.2
Locky ransomware	54.4	59.6	65.6
Cerber ransomware	77.2	77.2	82.8
CryptoWall ransomware	27.6	38.4	50.4

Table 8 Comparison of evaluation metrics for embedded YARA rules with enhanced YARA rules and YARA rules

Evaluation metric	Basic YARA rules (%)	Enhanced YARA rules (%)	Embedded YARA rules (%)
Accuracy	79.80	83.55	86.75
Precision	95.99	96.27	96.58
Recall	62.20	67.10	73.50
F1-Score	75.49	79.08	83.48

YARA rules is 83.48%, which is better than the F1-Score of both basic YARA rules (75.49%) and enhanced YARA rules (79.08%). This confirms that the proposed embedded YARA rules are more effective as compared to other two types of YARA rules.

Advantages and limitations of the proposed technique

Advantages of the proposed embedded YARA rules

The proposed embedded YARA rules offers several advantages, here some of the most notable advantages:

- *Extending search scope* Fuzzy rules can combine multiple parameters and their complex conditions to produce one approximated output.
- *Extending result scope* In addition to alert samples as malware by YARA rules, fuzzy rules reveal the degree of similarity of malware (Less Likely Malware, Likely Malware, and Most Likely Malware).
- *Aiding in analysis* 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 deep dive investigation into the samples.
- *Improving detection rate* Fuzzy hashing can complement YARA rules as it attempts to find structural similarity between the two files in their entirety in circumstances where the selected IoC strings cannot be found in the sample. Thus, it can still trigger fuzzy rules and detect more malware samples than YARA rules.
- *Maintaining performance* Fuzzy hashing is one of the fastest analysis methods and it generates a compact hash, which does not affect the overall performance of the combined analysis process.
- *Accuracy improvement* In case of fuzzy hashing found exactly matched sample(s), the strong similarity score 1 or 100% is generated, which increases the accuracy of the overall result and the further processing results of clustering or classification.

Limitations of the proposed embedded YARA rules

Despite offering several advantages, the proposed embedded YARA rules inherit some of the limitations of YARA rules and fuzzy hashing, here some of the most notable limitations are:

- *Dependency of fuzzy rules* The result of fuzzy rules is dependent on the values of YARA rules and fuzzy hash indicators, thus if both fails to discover any sample then the fuzzy outcome will also be missed out.
- *Not a rule optimisation approach* This proposed approach does not focus on generating optimised YARA rules rather its focus is to increase the effectiveness of existing YARA rules. Therefore, the success of this proposed approach is also dependent on the superiority of rules itself.
- *Trusted code* If YARA rules are created utilising trusted code then this will increase the number of false positive [15], which will affect outcomes of the proposed approach.
- *Fuzzy structural similarity* Fuzzy hashing can only discover structural or syntactic similarity, but not behavioural or semantic similarity, therefore, it is only a complementary method to YARA rules but does not offer the same effectiveness as YARA rules.
- *Fuzzy similarity scores* Similarity scores provided by fuzzy hashing could be analysed and utilised differently by different security analysts, resulting in different conclusions based on the same similarity scores.
- *Scalability of advanced YARA rules* Writing advanced YARA rules at scale is a challenging task in general [15], and this also applies to embedded YARA rules.

Conclusion

This paper proposed two different techniques utilising fuzzy hashing and fuzzy rules to strengthen YARA rules: enhanced YARA rules and embedded YARA rules. The first proposed technique enhanced YARA rules utilised a fuzzy hashing technique and the second proposed technique embedded YARA rules utilised fuzzy hashing and fuzzy rules to improve

YARA rules and its effectiveness. This improvement process was focused on the execution phase of YARA rules rather than optimising rules itself. Which improved the searching criteria and effectiveness of YARA rules without significantly increasing the complexity and overheads of YARA rules. The experimental results demonstrated that the proposed enhanced YARA rules and embedded YARA rules produce improved results in comparison with basic YARA rules. However, the most improved embedded YARA rules has some limitations due to the inherent limitations of the underlying YARA rules and fuzzy hashing, which require further investigations and improvement in the future.

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

Compliance with ethical standards

Conflict of interest On behalf of all authors, the corresponding author states that there is no conflict of interest.

References

1. Alvarez V (2019) Writing YARA rules. https://yara.readthedocs.io/en/v3.4.0/writingrules.html
2. Alvarez V (2019) YARA Documentation, Release 3.10. 0. https://buildmedia.readthedocs.org/media/pdf/yara/latest/yara.pdf
3. Andronio N, Zanero S, Maggi F (2015) Heldroid: dissecting and detecting mobile ransomware. In: International workshop on recent advances in intrusion detection. Springer, pp 382–404
4. Breitinger F, Astebøl KP, Baier H, Busch C (2013) mvhash-b-a new approach for similarity preserving hashing. In: 2013 Seventh international conference on IT security incident management and IT forensics. IEEE, pp 33–44
5. Breitinger F, Baier H (2012) A fuzzy hashing approach based on random sequences and hamming distance. In: Annual ADFSL conference on digital forensics, security and law. 15. https://commons.erau.edu/adfsl/2012/wednesday/15
6. Cabaj K, Gawkowski P, Grochowski K, Osojca D (2015) Network activity analysis of Cryptowall ransomware. Przeglad Elektrotechniczny 91(11):201–204
7. Chen Q, Bridges RA (2017) Automated behavioral analysis of malware a case study of wannacry ransomware. arXiv preprint arXiv:1709.08753
8. Culling CS (2018) Which YARA rules : basic or advanced?. https://vt-gtm-wp-media.storage.googleapis.com/2.0-Which-YARA-Rules-Rule-Basic-or-Advanced-1.pdf
9. Dias R (2014) Intelligence-driven incident response with YARA. https://www.sans.org/reading-room/whitepapers/forensics/intelligence-driven-incident-response-yara-35542
10. Dubois D, Prade H (1996) What are fuzzy rules and how to use them. Fuzzy Sets Syst 84(2):169–185
11. French D (2012) Writing effective YARA signatures to identify malware. https://insights.sei.cmu.edu/sei_blog/2012/11/writing-effective-yara-signatures-to-identify-malware.html
12. Gayoso Martínez V, Hernández Álvarez F, Hernández Encinas L (2014) State of the art in similarity preserving hashing functions. http://digital.csic.es/bitstream/10261/135120/1/Similarity_preserving_Hashing_functions.pdf
13. Gómez-Hernández J, Álvarez-González L, García-Teodoro P (2018) R-locker: Thwarting ransomware action through a honeyfile-based approach. Comput Secur 73:389–398
14. Hybrid-Analysis: Hybrid Analysis (2019) https://www.hybrid-analysis.com/
15. Intezer.com: Generate advanced YARA rules based on code reuse (2019). https://intezer.com/wp-content/uploads/2019/06/Intezer_YARA_White_Paper.pdf
16. Kharraz A, Arshad S, Mulliner C, Robertson WK, Kirda E (2016) Unveil: a large-scale, automated approach to detecting ransomware. In: USENIX Security symposium, pp 757–772
17. Klijnsma Y (2019) The history of Cryptowall: a large scale cryptographic ransomware threat. https://www.cryptowalltracker.org/
18. Kornblum J (2006) Identifying almost identical files using context triggered piecewise hashing. Digit Investig 3:91–97
19. Malshare (2019) A free Malware repository providing researchers access to samples, malicious feeds, and YARA results. https://malshare.com/index.php
20. Malwarebytes: Ransomware (2019). https://www.malwarebytes.com/ransomware/
21. Mamdani EH, Assilian S (1975) An experiment in linguistic synthesis with a fuzzy logic controller. Int J Man Mach Stud 7(1):1–13
22. Mandiant: Tracking malware with import hashing (2014). https://www.fireeye.com/blog/threat-research/2014/01/tracking-malware-import-hashing.html
23. Mohaisen A, Alrawi O, Mohaisen M (2015) AMAL: high-fidelity, behavior-based automated malware analysis and classification. Comput Secur 52:251–266
24. Naik N, Diao R, Shen Q (2018) Dynamic fuzzy rule interpolation and its application to intrusion detection. IEEE Trans Fuzzy Syst 26(4):1878–1892
25. Naik N, Jenkins P, Cooke R, Gillett J, Jin Y (2020) Evaluating automatically generated YARA rules and enhancing their effectiveness. In: IEEE symposium series on computational intelligence (SSCI). IEEE
26. Naik N, Jenkins P, Gillett J, Mouratidis H, Naik K, Song J (2019) Lockout-Tagout Ransomware: A detection method for ransomware using fuzzy hashing and clustering. In: IEEE symposium series on computational intelligence (SSCI). IEEE
27. Naik N, Jenkins P, Savage N (2019) 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). IEEE
28. Naik N, Jenkins P, Savage N, Yang L (2019) Cyberthreat Hunting-Part 1: triaging ransomware using fuzzy hashing, import hashing and YARA rules. In: IEEE international conference on fuzzy systems (FUZZ-IEEE). IEEE

29. Naik N, Jenkins P, Savage N, Yang L (2019) 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
30. Naik N, Jenkins P, Savage N, Yang L (2020) A computational intelligence enabled honeypot for chasing ghosts in the wires. Complex Intell Syst
31. Naik N, Jenkins P, Savage N, Yang L, Boongoen T, Iam-On N (2020) Fuzzy-import hashing: a malware analysis approach. In: IEEE international conference on fuzzy systems (FUZZ-IEEE). IEEE
32. Naik N, Jenkins P, Savage N, Yang L, Naik K, Song J (2019) Augmented YARA rules fused with fuzzy hashing in ransomware triaging. In: IEEE symposium series on computational intelligence (SSCI). IEEE
33. Naik N, Jenkins P, Savage N, Yang L, Naik K, Song J (2020) Embedding fuzzy rules with YARA rules for performance optimisation of malware analysis. In: IEEE international conference on fuzzy systems (FUZZ-IEEE). IEEE
34. Naik N, Jenkins P, Savage N, Yang L, Naik K, Song J, Boongoen T, Iam-On N (2020) Fuzzy hashing aided enhanced YARA rules for malware triaging. In: IEEE symposium series on computational intelligence (SSCI). IEEE
35. Naik N, Shang C, Jenkins P, Shen Q (2020) D-FRI-Honeypot: A secure sting operation for hacking the hackers using dynamic fuzzy rule interpolation. IEEE Trans Emerg Top Comput Intell
36. Readthedocs: Writing YARA rules (2019). https://yara.readthedocs.io/en/v3.5.0/writingrules.html
37. Richardson R, North M (2017) Ransomware: evolution, mitigation and prevention. Int Manag Rev 13(1):10–21
38. Roth F (2017) How to post-process YARA rules generated by yarGen. https://medium.com/@cyb3rops/how-to-post-process-yara-rules-generated-by-yargen-121d29322282
39. Roth F (2018) yarGen is a generator for YARA rules. https://github.com/Neo23x0/yarGen
40. Roussev V (2010) Data fingerprinting with similarity digests. In: IFIP international conference on digital forensics. Springer, pp 207–226
41. Sadowski C, Levin G (2007) Simhash: Hash-based similarity detection. www.webrankinfo.com/dossiers/wp-content/uploads/simhash.pdff
42. Savage K, Coogan P, Lau H (2015) The evolution of ransomware - Symantec pp 1–57
43. Sgandurra D, Muñoz-González L, Mohsen R, Lupu EC (2016) Automated dynamic analysis of ransomware: Benefits, limitations and use for detection. arXiv preprint arXiv:1609.03020
44. Takagi T, Sugeno M (1985) Fuzzy identification of systems and its applications to modeling and control. IEEE Trans Syst Man Cybern 1:116–132
45. Tridgell A (1999) Efficient algorithms for sorting and synchronization. Ph.D. thesis, Australian National University Canberra
46. VirusTotal: Virustotal (2019). https://www.virustotal.com/#/home/upload
47. VirusTotal: YARA in a nutshell (2019). https://virustotal.github.io/yara/