Evaluating Automatically Generated YARA Rules and Enhancing Their Effectiveness

Nitin Naik[1], Paul Jenkins[2], Roger Cooke[3], Jonathan Gillett[3] and Yaochu Jin[4]

[1]School of Informatics and Digital Engineering, Aston University, United Kingdom
[2]School of Computing, University of Portsmouth, United Kingdom
[3]Defence School of Communications and Information Systems, Ministry of Defence, United Kingdom
[4]Department of Computer Science, University of Surrey, United Kingdom
Email: n.naik1@aston.ac.uk, paul.jenkins@port.ac.uk, roger.cooke472@mod.gov.uk, jon.gillett704@mod.gov.uk, yaochu.jin@surrey.ac.uk

***Abstract*—Emerging as a widely accepted technique for malware analysis, YARA rules due to its flexible and customisable nature, allows malware analysts to develop rules according to the requirements of a specific security domain. YARA rules can be automatically generated using tools, however, they may require post-processing for their optimisation, and may not be effective for the specific security domain. This compels the requirement to enhance automatically generated YARA rules and increase their effectiveness for malware analysis without increasing computational overheads. Reflecting on the above requirement, this paper initially evaluates automatically generated YARA rules using three YARA tools: yarGen, yaraGenerator and yabin. These tools are Python-based open-source tools used to generate YARA rules automatically utilising different underlying techniques. Subsequently, it proposes a method to enhance automatically generated YARA rules using a fuzzy hashing method. This proposed enhancement method can improve the effectiveness of YARA rules irrespective of the chosen YARA tool used to generate YARA rules, which is demonstrated through several experiments on samples of collected malware and goodware.**

***Index Terms*—Malware Analysis; YARA Rules; Fuzzy Hashing; yarGen, yaraGenerator; yabin; Ransomware; Indicator of Compromise; IoC String.**

I. Introduction

The accelerating rate of malware incidents on daily basis indicates the magnitude of the problem in malware analysis. While malware analysts detect many malware attacks and incidents, keeping pace with the number and different types of attacks poses a significant challenge to malware analysts. There is no silver bullet with respect to malware, as there is no single malware analysis technique with the capability to treat all malware incidents, as a result analysts select the most suitable malware analysis technique for the specific security incident under consideration [1]. In recent years, YARA rules has emerged as a widely accepted technique for malware analysis due to its flexible and customisable nature, allowing malware analysts to develop YARA rules according to their specific requirements in targeting specific types of threats [2]. YARA rules are generated based on reverse engineering of malware samples to include the most common Indicator of Compromise (IoC) strings from those malware samples to find similar types of malware.

978-1-7281-2547-3/20/$31.00 ©2020 IEEE

The success of YARA rules is dependent on the effectiveness of generated YARA rules, which is determined by the types of IoC strings and the number of IoC strings utilised in its rules [3]. Therefore, the generation of the most effective YARA rules is the biggest challenge in applying YARA rules for malware analysis [4]. YARA rules can be generated either manually or automatically. Generating YARA rules manually requires a highly-specialized skill-set in a specific security area, whereas generating YARA rules automatically using a tool is a relatively easy task [5]. However, there are several issues with automatically generated YARA rules such as these rules require post processing operations for their optimisation, despite this they may not become very effective for certain types of threats [4], [5]. This drives the requirement to enhance YARA rules and make them more effective for malware analysis. There are a number of ways to achieve this aim, however, any chosen mechanism should not increase the computational overheads as certain types of YARA rules may slow down the operation when applied to a large sample of malware [2], [6], [7]. Reflecting on the above requirement and the further enhancement of YARA rules, this paper at first evaluates automatically generated YARA rules using three YARA tools yarGen, yaraGenerator and yabin. These tools are Python-based open-source tools used to generate YARA rules automatically utilising different underlying techniques. Subsequently, it proposes a method to enhance automatically generated YARA rules using a fuzzy hashing method. This proposed enhancement method can improve the effectiveness of YARA rules irrespective of the chosen YARA tool used to generate YARA rules [8], which is demonstrated through several experiments on the collected malware and goodware samples.

The paper is divided into the following sections: Section II discusses YARA rules and fuzzy hashing as the underlying methods. Section III describes the three employed tools for automatically generating YARA rules: yarGen, yaraGenerator and yabin. Section IV explains the collection and verification

process of ransomware and goodware samples. Section V performs an evaluation of automatically generated YARA rules using yarGen, yaraGenerator and yabin Tools. Section VI presents the proposed enhancement process of automatically generated YARA rules using above YARA tools by employing the fuzzy hashing method SSDEEP. Section VII explores advantages and limitations of YARA Rules. Lastly, Section VIII concludes the paper and outlines some future work.

II. YARA Rules and Fuzzy Hashing

A. YARA Rules

YARA rules are developed to detect malware by matching its signatures/strings with the existing malware signatures/strings [3], [9]. These rules contain predetermined signatures/strings related to known malware used in attempting to match against the targeted files, folders, or processes [10]. YARA rules consist of three sections: meta, strings and conditions as shown in Figs. 1 and 2. Here, strings can be classified into three types: text strings, hexadecimal strings and regular expression strings. Text strings are generally a readable text complemented with some modifiers (e.g., nocase, ASCII, wide, and fullword), to manage the process more effectively [11]. Hexadecimal strings are a sequence of raw bytes complemented with three flexible formats: wild-cards, jumps, and alternatives [11]. 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 [11]. Text strings and regular expressions which 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 declare the sample as malware [12]. 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 [11].

```
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
}
```

Fig. 1. YARA Rules: Syntax

```
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. YARA Rules: Example

B. Fuzzy Hashing

Fuzzy hashing is used to determine the similarity between digital files, which makes it a very useful method for malware analysis as several pieces of malware and their variants possess some similarity with each other, which is not detected by a cryptographic hash as it has a binary outcome i.e., either the two files are exactly identical or not [13], [14]. In a fuzzy hashing technique, the file of interest is split into several blocks and each block is treated separately for calculating its hash, finally, hashes of all the blocks are concatenated to obtain the fuzzy hash of that file (see Fig. 3). 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 [15]. Fuzzy hashing methods are divided into different types namely: Context-Triggered Piecewise Hashing (CTPH), Statistically-Improbable Features (SIF), Block-Based Hashing (BBH) and Block-Based Rebuilding (BBR) [16], [17], [18]. 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 [19]. This is especially important when considering the analysis and clustering of suspected malware in order to discover new variants [20], [21]. 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 similar functionality, yet different cryptographic hash values [22].

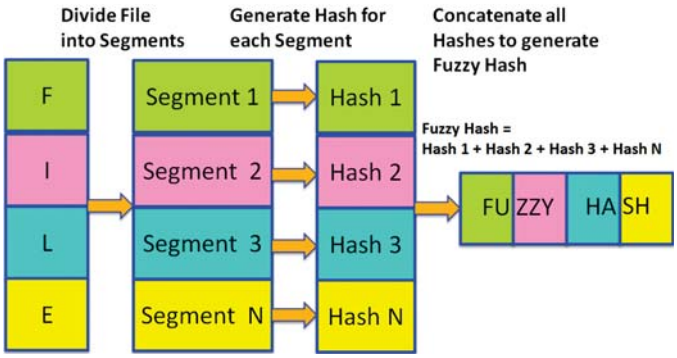

Fig. 3. Fuzzy Hash generation process in Fuzzy Hashing [10]

III. Employed Tools for Automatically Generated YARA Rules: yarGen, yaraGenerator and yabin

Generating YARA rules automatically is the most popular method in employing YARA rules in malware analysis. In this work, three different tools yarGen, yaraGenerator and yabin are used to generate YARA rules automatically for evaluating their effectiveness. Here, these three tools are explained with their advantages and drawbacks.

A. yarGen Tool

yarGen is a Python-based tool utilised to generate YARA rules, which is developed by Florian Roth [23]. It generates YARA rules utilising some intelligent techniques such as

fuzzy regular expressions, Naive Bayes classifier and Gibberish Detector [24]. The generated YARA rules include those strings and opcodes from malware which do not match with the provided goodware databases [23]. These YARA rules contain a predefined number of strings (generally up to 20 strings), based on their highest scores to maintain a reasonable operational speed. This tool generates two types of rules basic rules and super rules depending on the malware sample types, where basic rules can generally target a specific malware and super rules can target a set of malware or malware family.

```
import "pe"

rule
WannaCry_697158bcade7373ccc9e52ea1171d780988fc845d2b696898654e18954578920
{
meta:
        description = " Specific WannaCry Signature -
697158bcade7373ccc9e52ea1171d780988fc845d2b696898654e18954578920"
       author = "yarGen Rule Generator"
       date = "2018-06-01"
        reference = " https://github.com/Neo23x0/yarGen "
        hash =
"697158bcade7373ccc9e52ea1171d780988fc845d2b696898654e18954578920"
 strings:
   $s1 = "Tor/libevent_core-2-0-5.dll" fullword ascii /* score: '23.00'*/
   $s2 = "Tor/libevent_extra-2-0-5.dll" fullword ascii /* score: '23.00'*/
   $s3 = "Tor/libgcc_s_sjlj-1.dll" fullword ascii /* score: '22.00'*/
   $s4 = "Tor/libevent-2-0-5.dll" fullword ascii /* score: '22.00'*/
   $s5 = "Tor/libeay32.dll" fullword ascii /* score: '22.00'*/
   $s6 = "j GethS$k" fullword ascii /* score: '10.00'*/
   $s7 = "RJVt%Rt%_t" fullword ascii /* score: '9.00'*/
   $s8 = "\\\\%s\\C$\\%s" fullword ascii /* score: '8.50'*/
   $s9 = "kM:\\}kU tJ" fullword ascii /* score: '8.42'*/
   $s10 = "%!,I\"!*!&!.!!!MB" fullword ascii /* score: '8.42'*/
   $s11 = "%X7x:\\" fullword ascii /* score: '7.00'*/
   $s12 = "bv7.YCB" fullword ascii /* score: '7.00'*/
   $s13 = "I:\\Y&=" fullword ascii /* score: '7.00'*/
   $s14 = "LUCEKYC" fullword ascii /* score: '6.50'*/
   $s15 = "jeF&[ Y+ " fullword ascii /* score: '6.42'*/
   $s16 = "!j['?qw- " fullword ascii /* score: '6.42'*/
   $s17 = "S}SiSeSQSMS9S5S!S" fullword ascii /* score: '6.00'*/
   $s18 = "G@@LLJJFFIIEECCO" fullword ascii /* score: '6.00'*/
   $s19 = "XnL4;* 3T" fullword ascii /* score: '6.00'*/
   $s20 = "ipp0cN" fullword ascii /* score: '6.00'*/
 condition:
   uint16(0) == 0x5a4d and filesize < 10000KB and
   ( pe.imphash() == "c80a2354fd8e096ab6fd6b843b9a69f4" or 10 of them )
}
```

Fig. 4. yarGen Generated YARA Rule

1) Advantages- yarGen Tool:

- It allows generation of YARA rule based on both opcodes and strings.
- It supports the use of PE (portable executable) modules, which are used by the Windows operating system for executables such as DLL and COM files.
- It can be integrated with other anti-malware software for its more effective use.
- It reduces false positives by checking all strings against strings of goodware databases.
- Python script is simple and easy to use through command line interface.

2) Drawbacks- yarGen Tool:

- It requires post-processing of rules for increasing their effectiveness.

```
rule
WannaCry_0e663b46bf75806da90902174010a2074f910c07814a8ec863b7fb733a9eafa8
{
meta:
        description = " Specific WannaCry Signature -
0e663b46bf75806da90902174010a2074f910c07814a8ec863b7fb733a9eafa8"
        author = "yarGenerator Rule Generator "
        date = "2018-06-01"
        yaraGenerator = " https://github.com/Xen0ph0n/YARAGenerator "
         hash = "2f10e1fa578735fee822feb0b8f8a75b"
strings:
        $string0 = "}2lcb3"
        $string1 = "(x(B(((B((("
        $string2 = " documentation on asserts."
        $string3 = "Client hook re-allocation failure."
        $string4 = "else's kettle Clinque "
        $string5 = "7%8,82888G8P8U8"
        $string6 = "(e($([(H("
        $string7 = "BBN profile "
        $string8 = "(((8(((((("
        $string9 = "string != NULL && sizeInWords > 0" wide
        $string10 = "Lavasoft" wide
        $string11 = "7cGDi d)"
        $string12 = "CreateDialogParamA"
        $string13 = "(d(8(,("
        $string14 = " new[]"
        $string15 = "InsertMenuItemA"
        $string16 = "TLOSS error"
        $string17 = "strcpy_s(szOutMessage, 4096, szLineMessage)" wide
        $string18 = "f:\\dd\\vctools\\crt_bld\\self_x86\\crt\\src\\output.c"
condition:
        10 of them
}
```

Fig. 5. yaraGenerator Generated YARA Rule

- It requires significant resources for opcodes-based rules and loading goodware files.
- The rule generation process is slow.
- The creation of super rules could cause duplication of rules and redundancy.
- It requires installation of all dependencies and built-in databases for working successfully.

B. yaraGenerator Tool

It is a Python-based tool used for the generation of YARA rules, which is developed by Chris Clark [25]. It generates YARA rules with a completely different signature for different types of files such as EXEs, PDFs and Emails utilising string prioritization logic and code refactoring [25]. The generated YARA rules consist of strings only, including those strings from malware which do not match with the provided blacklist of strings [25]. It uses a database of 30,000 blacklisted strings divided based on different file formats. These YARA rules contain a large number of strings (depending on the types of samples) selected randomly as it does not compute a score or weighting for strings.

1) Advantages- yaraGenerator Tool:

- It can generate specialised rules of a specific file format.
- It supports the use of PE (portable executable) modules, which are used by Windows operating system for executables such as DLL and COM files.
- It reduces false positives by checking all strings against strings of blacklist files.

- Python script is simple and easy to use through command line interface.

2) Drawbacks- yaraGenerator Tool:

- It requires post-processing of rules for increasing their effectiveness.
- It generates YARA rules based on random selection of strings which may not select the most appropriate strings in many cases.
- It does not support the inclusion of opcodes.
- The project was developed as a work in progress and not updated afterwards.

C. *yabin Tool*

It is another Python-based tool used for generating YARA rules, which is developed by Alien Vault Open Threat Exchange (OTX) community [26]. It generates YARA rules by finding rare functions in a certain malware samples or families [26]. It recognises functions by checking function prologues which define the start of functions, for example, *55 8B EC* mostly specifies the start of a function in programs compiled by Microsoft Visual Studio. The generated YARA rules include those strings from malware which do not match with the provided whitelist of common library functions [26]. It uses a whitelist obtained from 100 Gb of non-malicious software to omit common library functions [26]. These YARA rules contain a list of hexadecimal strings to compare against suspected malware files for finding the similarity in their byte-sequences.

```
rule
WannaCry_1be0b96d502c268cb40da97a16952d89674a9329cb60bac81a96e01cf7356830
{
strings:
        $a_2 = { 558baad804000081fd000400005d7705 }
        $a_3 = { 558bec6aff680072001068b66b001064 }
        $a_4 = { 558bac24e40200005683fd0457894c24 }
        $a_5 = { 558be9896c24188a450484c0751e68cc }
        $a_6 = { 558bec6aff683072001068b66b001064 }
        $a_7 = { 558bec6aff681072001068b66b001064 }
        $a_8 = { 558be956578a4504484c0751e68ccd800 }
        $a_9 = { 558b2d9c70001056576a00ffd38b0dc4 }
        $a_10 = { 558b6c242833db563beb8bf10f843201 }
        $a_11 = { 558bac2438020000894424088b84243c }
        $a_12 = { 558b2dc070001057eb048b7424106aff }
        $a_13 = { 558bcee81adfffff8b066a018bceff10 }
        $a_14 = { 558bfbf3abb98100000033c08d7c2432 }
        $a_15 = { 558bec6aff682072001068b66b001064 }
condition:
        10 of them
}
```

Fig. 6. yabin Generated YARA Rule

1) Advantages- yabin Tool:

- It can be used to cluster malware samples based on the reuse of their code.
- The search patterns can be extended during the post-processing operation.
- It provides a large whitelist obtained from numerous non-malicious software to omit common library functions.
- Python script is simple and easy to use through command line interface.

2) Drawbacks- yabin Tool:

- It requires post-processing of rules to make them more effective.
- It may not work on some specific file formats.
- It only uses functions and does not use other types of strings.
- It can only work with unpacked executables.
- It is not designed to work on .NET executables, Java files and Microsoft documents.
- It is mainly developed for the testing purpose and not for the production.

IV. Collection of Malware and Goodware Samples

In this implementation, one of the most prevalent malware, ransomware was selected to perform all analysis and evaluation of the effectiveness and performance of the proposed techniques. Ransomware was selected for the experiment as it is one of the most relevant and damaging examples of 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/WannaCryptor, Locky, Cerber and CryptoWall [27], [28], [29]. Thousands of malware samples were acquired from the two sources *Hybrid Analysis* [30] and *Malshare* [31]. Later, these samples were verified for their credibility as numerous samples were simply bogus samples. 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 techniques successfully. These samples were investigated based on the information available on *VirusTotal* [32]. To determine that every sample was indeed genuine malware or ransomware and they were members of 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/WannaCryptor, 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/WannaCryptor, Locky, Cerber and CryptoWall. The four different categories of ransomware were chosen to evaluate how each employed YARA tool and its corresponding enhanced 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 10 different software samples were chosen in such a way that it could encompass a wide range of benign programs and to evaluate how each employed YARA tool and its corresponding enhanced method functions on the different types of benign program. Finally, a total 2000 samples were utilised to perform all the experiments applying all employed YARA tools and their corresponding enhanced methods.

V. Evaluating Automatically Generated YARA Rules Using YARA Tools

In this section, three selected tools that automatically generate YARA rules yarGen, yaraGenerator and yabin are evaluated. All the tools are applied on the collected and verified ransomware samples of four ransomware corpora WannaCry/WannaCryptor, Locky, Cerber and CryptoWall to generate YARA rules. The generated YARA rules from each tool are used to perform malware analysis and determine their malware detection success rate as explained later in the subsections. The experiment is aimed to illustrate the similarity detection success rate of each YARA tool for each ransomware category separately and collectively. It is expected and most probably 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 the generated YARA rules through each tool.

A. Evaluation Procedure of Automatically Generated YARA Rules

All the three tools yarGen, yaraGenerator and yabin are Python-based tools, therefore the YARA rules generation procedure for all the tools was quite similar by using the command line interface. However, the generated YARA rules are quite different as they are based on different methodologies. Utilising all three tools with their default settings and databases, YARA rules are generated for all four ransomware corpora WannaCry/WannaCryptor, Locky, Cerber and CryptoWall separately. Evidently, if the default settings are changed and the number of strings and attributes are increased or decreased then the same tool may produce different YARA rules. Furthermore, the automatically generated rules require post-processing to make them more effective. However, for the rational evaluation of three YARA tools, the generated YARA rules were evaluated without any post-processing operation.

B. Evaluation Results of Automatically Generated YARA Rules

Once the YARA rules are generated utilising all three tools for all four ransomware categories separately, they are used to detect the similarity for each ransomware category. The detection results for all four ransomware categories for all three YARA tools are shown in Table I. The detection results show that YARA rules generated by yarGen tool outperformed the YARA rules generated by the other two tools yaraGenerator and yabin. This result indicates that for the same malware samples, different tools generate different rules which produce very different results. This result is based on the default settings of each tool, however, if the default settings are changed and number of strings and attributes are increased or decreased then the same tool may produce different analysis results. Importantly, 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.

These three tools are further evaluated based on the values of False Positives and False Negatives. This evaluation is based on four standard evaluation metrics (Accuracy, Precision, Recall and F1-Score), which are calculated as shown in Table II. Here, the overall result of YARA rules generated by yarGen tool is better than the result of YARA rules generated by two other selected tools yaraGenerator and yabin. Moreover, to evaluate the efficiency of any tool 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 tool. Here, F1-Score of YARA rules generated by yarGen tool is 75.49%, which is better than the F1-Score of YARA rules generated by the other two selected tools. This shows that YARA rules generated by yarGen are more efficient as compared to YARA rules generated by the other tools yaraGenerator and yabin. Despite the relatively better result of YARA rules generated by yarGen tool, its result is not sufficient to consider it as a generic method for analysis in this particular case. Therefore, further investigation is required to improve the efficiency of these YARA rules.

TABLE I
Detection Results of Automatically Generated YARA rules using Tools yarGen, yaraGenerator and yabin for WannaCry, Locky, Cerber and CryptoWall Ransomware Samples

Ransomware Category	**yarGen-YARA Rules Detection Rate**	**yaraGenerator-YARA Rules Detection Rate**	**yabin-YARA Rules Detection Rate**
WannaCry Ransomware	89.6%	28.8%	44.8%
Locky Ransomware	54.4%	6.8%	11.2%
Cerber Ransomware	77.2%	10.4%	17.2%
CryptoWall Ransomware	27.6%	5.2%	9.6%

TABLE II
Evaluation Metrics for Automatically Generated YARA Rules using Tools yarGen, yaraGenerator and yabin

Evaluation Metric	**yarGen-YARA Rules**	**yarGenerator-YARA Rules**	**yabin-YARA Rules**
Accuracy	79.80%	56.4%	60.35%
Precision	95.99%	78.53%	88.09%
Recall	62.20%	12.8%	20.7%
F1-Score	**75.49%**	**22.01%**	**33.52%**

VI. Enhancing the Effectiveness of Automatically Generated YARA Rules with Fuzzy Hashing Method

A. Enhancing Procedure of Automatically Generated YARA Rules

Irrespective of the selected YARA rule generation tool, all YARA rules contain strings which are matched against strings of examined malware samples. The number and types of strings determine the success of generated YARA rules. Nonetheless, threat actors are equally intelligent and understand such mechanisms, and they frequently attempt evasion by using intelligent modifications in their malware. If only few or none of the selected strings are found in the examined samples then YARA rules do not flag samples as malware even though they may be malware. To enhance the effectiveness of YARA rules, the number of strings in YARA rules can be increased, however, adding a large number of strings in rules may increase the computational complexity and overheads affecting the performance of YARA rules significantly. Additionally, in order to write such complex YARA rules or modify automatically generated rules, a high degree of expertise is required in cyber security [2], [6], [33]. Consequently, it is essential to find a simpler solution to make YARA rules more effective without incurring the complexities stated earlier.

Therefore, the requirement is to explore alternative mechanisms other than strings to enhance YARA rules. Fuzzy hashing is a compact, fast and resource-optimised malware analysis method, which may not be effective on its own, nonetheless it can complement YARA rules enhancing its effectiveness without affecting complexity significantly [22]. Fuzzy hashing attempts to find structural similarity between the two entire files in circumstances where the selected strings cannot be found in the sample [34]. Therefore, they can complement each other in finding a missed opportunity by one of the mechanisms. Additionally, 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. Thus, the combined search result can increase the accuracy and confidence level of the malware analysis. The operational flow of this proposed fuzzy hashing aided enhanced YARA rules is shown in Fig. 7

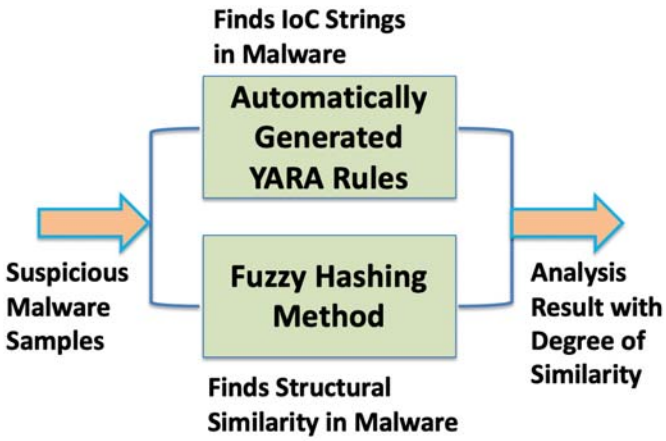

Fig. 7. Fuzzy Hashing Aided Enhanced YARA Rules

B. Enhancing Results of Automatically Generated YARA Rules

The generated YARA rules using three selected tools yarGen, yaraGenerator and yabin are adapted to incorporate fuzzy hashing method SSDEEP to evaluate their effectiveness on all four ransomware corpora WannaCry/WannaCryptor, Locky, Cerber and CryptoWall. The reason for the selection of a particular SSDEEP fuzzy hashing method over other fuzzy hashing methods (e.g., SDHASH and mvHASH-B) is explained in detail in the paper [8], [14], where SSDEEP is more compact, faster and a resource-optimised fuzzy hashing method in comparison to the other fuzzy hashing methods [35]. Here, the SSDEEP fuzzy similarity scores greater than 30% are utilised for all the three YARA tools [8]. The detection results of enhanced YARA rules utilising SSDEEP fuzzy hashing method for all three tools on the four ransomware categories are shown in Table III. Noticeably, enhanced YARA rules generated by all three tools (yarGen, yaraGenerator and yabin) have indicated an improvement in the detection result as compared to the original YARA rules generated by these three tools. The detection results show that enhanced YARA rules generated by yarGen tool again outperformed the enhanced YARA rules generated by other two tools yaraGenerator and yabin.

The enhanced YARA rules generated using fuzzy hashing and three tools are further evaluated based on the values of False Positives and False Negatives. Similarly, this evaluation is based on four standard evaluation metrics (Accuracy, Precision, Recall and F1-Score), which are calculated as shown in Table IV. Here, the overall result of enhanced YARA rules generated by yarGen tool is again better than the result of enhanced YARA rules generated by the other two tools yaraGenerator and yabin. Moreover, to evaluate the efficiency of any tool 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 tool. Here, the F1-Score of enhanced YARA rules is 79.08%, which is better than the F1-Score of enhanced YARA rules generated by other two tools yaraGenerator and yabin. This shows that enhanced YARA rules generated by yarGen are again more efficient as compared to enhanced YARA rules generated by other two tools yaraGenerator and yabin.

Finally, the selected three tools yarGen, yaraGenerator and yabin are compared based on their operational and functional parameters as shown in Table V. This shows yarGen is relatively better tool in terms of various features, functionalities and accuracy, however, due to its comprehensive features and functionality, it requires greater resources and computational overheads, resulting in its slower performance.

TABLE III
DETECTION RESULTS OF ENHANCED YARA RULES GENERATED USING FUZZY HASHING AND TOOLS YARGEN, YARAGENERATOR AND YABIN FOR WANNACRY, LOCKY, CERBER AND CRYPTOWALL RANSOMWARE SAMPLES

Ransomware Category	**yarGen-Fuzzy Hash Enhanced YARA Rules Detection Rate**	**yaraGenerator-Fuzzy Hash Enhanced YARA Rules Detection Rate**	**yabin-Fuzzy Hash Enhanced YARA Rules Detection Rate**
WannaCry Ransomware	93.2%	90.8%	90.8%
Locky Ransomware	59.6%	41.6%	41.6%
Cerber Ransomware	77.2%	33.6%	33.6%
CryptoWall Ransomware	38.4%	28%	28%

TABLE IV
EVALUATION METRICS FOR ENHANCED YARA RULES GENERATED USING FUZZY HASHING AND TOOLS YARGEN, YARAGENERATOR AND YABIN

Evaluation Metric	**yarGen-Fuzzy Hash Enhanced YARA Rules**	**yaraGenerator-Fuzzy Hash Enhanced YARA Rules**	**yabin-Fuzzy Hash Enhanced YARA Rules**
Accuracy	83.55%	74.25%	74.25%
Precision	96.27%	93.27%	94.54%
Recall	67.10%	48.5%	48.5%
F1-Score	**79.08%**	**63.81%**	**64.11%**

TABLE V
COMPARISON OF OPERATIONAL AND FUNCTIONAL PARAMETERS OF YARGEN, YARAGENERATOR AND YABIN TOOLS

Operational and Functional Parameters	**yarGen**	**yaraGenerator**	**yabin**
IoC Strings	Text, Hex, Regular Expressions, Opcodes	Text, Hex, Regular Expressions	Function Prologues
Weighing Scores of IoC Strings	Yes	No	No
Portable Executable (PE) Module	Yes	Yes	No
Use of Machine Learning Methods	Yes	No	No
Underlying Methods	Fuzzy Regular Expressions, Naive Bayes Classifier and Gibberish Detector	String Prioritization Logic and Code Refactoring	Finding rare functions by checking function prologues
Language	Python	Python	Python
Databases	good-exports.db, good-imphashes.db, good-opcodes.db, good-strings.db	blacklist.txt, regexblacklist.txt	whitelist database (db.db)
Malware Clustering	Yes	No	Yes
Resource Requirement	Highest	Lower	Lowest
Speed	Slowest	Slower	Fastest
Accuracy	Most Accurate	Least Accurate	Less Accurate
Open-Source	Yes	Yes	Yes

VII. ADVANTAGES AND LIMITATIONS OF YARA RULES

A. Advantages of YARA Rules

YARA rules offers several advantages over other malware analysis techniques, here are some of the most notable advantages:

- YARA rules offer an easy and efficient way of writing flexible and custom rules according to the requirements of a specific security domain.
- YARA rules are an open standard and work on most of the major platforms such as Windows, Linux and Mac OS.
- YARA rules can be easily integrated into Python and C/C++ programming languages.
- YARA rules can be used for both static and dynamic malware analysis.
- Several tools are available to generate YARA rules easily and efficiently.
- Several public repositories of YARA rules offer readily available rules for malware analysis.

B. Limitations of YARA Rules

YARA rules are one of the most established malware analysis techniques, however, they have some limitations, here some of the most notable:

- YARA rules are commonly written based on IoC strings, however, attackers can easily manipulate, replace or encrypt these IoC strings to evade them, which could make these rules less effective.
- IoC strings are extracted from existing malware and their families through a reverse engineering process, which requires a highly-specialized skill-set in a specific security domain.
- The success of YARA rules is dependent on the types and number of IoC strings included in rules, however, achieving the balance of both is a challenging task as an ineffective and inappropriate number of IoC strings could affect the performance of YARA rules adversely.
- YARA rules can be automatically generated using tools, however, they may require post-processing for their optimisation, and may not be as effective as manually generated YARA rules.
- YARA rules are effective in detecting malware which resemble similarity with the existing malware and their families, however, it may miss out new and unique malware variants.

VIII. CONCLUSION

This paper presented an evaluation of automatically generated YARA rules using three YARA tools yarGen, yaraGenerator and yabin, including a technique to enhance their effectiveness using a fuzzy hashing method. These three tools are applied on the collected ransomware samples of four ransomware corpora WannaCry/WannaCryptor, Locky, Cerber and CryptoWall to generate YARA rules. The generated YARA rules from each tool are used to perform malware analysis

and determine their malware detection success rate. Here, the yarGen tool provided relatively better detection results as compared to other two tools yaraGenerator and yabin. Later, the generated YARA rules using three selected tools yarGen, yaraGenerator and yabin are enhanced by incorporating the fuzzy hashing method SSDEEP and their effectiveness on all four ransomware corpora is re-evaluated. This proposed enhancement improved detection results for all three tools, however, yarGen performed relatively better as compared to the other two tools yaraGenerator and yabin. In the future, two important analyses should be performed: generating YARA rules by adapting various parameters of each tool and evaluation of additional YARA tools and their generated YARA rules.

ACKNOWLEDGEMENT

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

REFERENCES

[1] K. Baker. (2020) Malware Analysis. [Online]. Available: https://www.crowdstrike.com/epp-101/malware-analysis/

[2] C. S. Culling. (2018) Which YARA Rules : Basic or Advanced? [Online]. Available: https://vt-gtm-wp-media.storage.googleapis.com/2.0-Which-YARA-Rules-Rule-Basic-or-Advanced-1.pdf

[3] N. Naik, P. Jenkins, N. Savage, L. Yang, K. Naik, and J. Song, "Embedding fuzzy rules with YARA rules for performance optimisation of malware analysis," in *IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*. IEEE, 2020.

[4] D. French. (2012) Writing effective YARA signatures to identify malware. [Online]. Available: https://insights.sei.cmu.edu/sei_blog/2012/11/writing-effective-yara-signatures-to-identify-malware.html

[5] Intezer.com. (2019) Generate advanced YARA rules based on code reuse. [Online]. Available: https://intezer.com/wp-content/uploads/2019/06/Intezer_YARA_White_Paper.pdf

[6] V. Alvarez. (2019) YARA Documentation, Release 3.10. 0. [Online]. Available: https://buildmedia.readthedocs.org/media/pdf/yara/latest/yara.pdf

[7] F. Roth. (2019) YARA performance guidelines. [Online]. Available: https://gist.github.com/Neo23x0/e3d4e316d7441d9143c7

[8] N. Naik, P. Jenkins, N. Savage, L. Yang, K. Naik, J. Song, T. Boongoen, and N. Iam-On, "Fuzzy hashing aided enhanced YARA rules for malware triaging," in *IEEE Symposium Series on Computational Intelligence (SSCI)*. IEEE, 2020.

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

[10] 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.

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

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

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

[14] 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.

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

[16] 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

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

[18] 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

[19] N. Naik, C. Shang, P. Jenkins, and Q. Shen, "D-FRI-Honeypot: A secure sting operation for hacking the hackers using dynamic fuzzy rule interpolation," *IEEE Transactions on Emerging Topics in Computational Intelligence*, 2020.

[20] N. Naik, P. Jenkins, N. Savage, and L. Yang, "A computational intelligence enabled honeypot for chasing ghosts in the wires," *Complex & Intelligent Systems*, 2020.

[21] ——, "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.

[22] ——, "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.

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

[24] ——. (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

[25] C. Clark. (2013) yaraGenerator: Automatic YARA rule generation. [Online]. Available: https://github.com/Xen0ph0n/YaraGenerator

[26] C. Doman. (2018) yabin: A YARA rule generator for finding related samples and hunting. [Online]. Available: https://github.com/AlienVault-OTX/yabin

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

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

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

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

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

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

[33] R. Dias. (2014) Intelligence-Driven Incident Response with YARA. [Online]. Available: https://www.sans.org/reading-room/whitepapers/forensics/intelligence-driven-incident-response-yara-35542

[34] N. Naik, P. Jenkins, N. Savage, L. Yang, T. Boongoen, and N. Iam-On, "Fuzzy-Import Hashing: A malware analysis approach," in *IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*. IEEE, 2020.

[35] 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.