Automatic YARA Rule Generation

Myra Khalid
Department of Computer and Information Sciences
Pakistan Institute of Engineering and Applied Sciences (PIEAS)
Islamabad, Pakistan
myrakhalid94@gmail.com

Maliha Ismail
CESAT
The National Centre for Physics,
Islamabad, Pakistan
maliha.ismaeel@gmail.com,

Mureed Hussain
CESAT
The National Centre for Physics,
Islamabad, Pakistan
hmureed@yahoo.com

Muhammad Hanif Durad
Department of Computer and Information Sciences
Pakistan Institute of Engineering and Applied Sciences (PIEAS)
Islamabad, Pakistan
hanif@pieas.edu.pk

***Abstract*— Since 2010, the number of new malware released daily have become so high, that manual analysis is not an option anymore. The purpose of this work is to focus on the increased modern cyber-attacks and malware campaigns. This work devises a framework that automates the process of generating high quality, effective and efficient malware signatures in considerably less amount of time and effort. It also facilitates in the tedious task of malware analysis. The proposed framework presents a generic approach to automatic YARA rule-based signature generation. This approach is based upon cherry-picking the most promising core ideas of the related work. The testing of the prototype shows that it is capable of detecting samples with an average precision of 0.95.**

***Keywords*— Malware Analysis, YARA, Naïve Bayes, Signature Generation.**

I. Introduction

Any software that "deliberately fulfills the harmful intent of an attacker" is referred to as malicious software or malware [1]. "The quantity, diversity and sophistication of malware has significantly increased in recent years" [2]. The daily increase in the number of malicious samples warrants the automation of malware analysis procedure. Several methodologies have been proposed during the years, including a variety of machine-learning and data-mining approaches. However, the signature-based malware detection mechanisms have proved to be simple, fast and the most effective ones with regards to the endpoint security.

Signature matching technique is commercially applied by anti-virus products having a predefined database of known signatures. While scanning, antivirus products create an appropriate signature for each file (using MD5 or other cryptographic hashes) and compare them with the predefined list. Upon a match, the file is treated as a 'threat'. The "anti-virus product(s) must be frequently updated and virus dictionary must have specific signature for each new malware" [3].

The traditional Hash-based signature schemes have an inherent shortfall, that minor changes in the context can cause a large change in the resulting hash value. This renders the use of cryptographic hashes less effective as malware signatures. Cryptographic hashes require maintaining a huge database of already known malware signatures. These databases have to be actively maintained and updated.

After hash-based signatures, new approaches like string-based signatures and rule-based signatures were introduced. YARA [4] is another such technique. The YARA framework emerged as de-facto standard for malware identification and classification. It presents a highly efficient pattern-matching engine which is most commonly used to scan large data. A set of YARA rules is generated that is applied to malware samples. The creation of YARA rules is a tedious task that requires expert domain knowledge, and experience in malware. analysis. Maintaining and updating YARA rules is complex and time-consuming task.

The scope of this work includes automatically generating a set of effective YARA signatures of known malware samples. These signatures are capable of matching new malware variants with high precision and recall, while reducing the number of false positives. The proposed methodology targets the generation of signatures for Microsoft Windows executable.

Further sections of paper include: section II provides a study of various existing related works. Section III contains the proposed framework. Section IV discusses results. Section V concludes the research paper.

II. Related Work

Automatic signature generation is a complicated task since it requires the generated signature to be specific and sensitive at the same time.

Initially proposed systems like Autograph [5], and EarlyBird [6] generate malware signatures based on a single contiguous string or token. Honeycomb [7] used Longest Common Substring (LCS) [8] algorithm to spot similarities among malware samples to automatically generate signatures. PAYL sensor [9] and Netspy [10] generation of signatures by distinguishing most frequently occurring byte-sequences.

Later on, Hamsa [11] and Botzilla [12] introduced more complex methods based on the token subsequence signatures. Other researches like ProVex [13], AutoRE [14], ShieldGen [15], and [16] are focused on automatic signature generation for network traffic. The Nemean architecture [17] introduced clustering of similar sessions of network traffic and applying

This work was supported by NCCS (National Centre in Cyber Security), HEC Pakistan through CIPMA Lab

Authorized licensed use limited to: University of Cape Town. Downloaded on May 18,2021 at 23:24:33 UTC from IEEE Xplore. Restrictions apply.

machine-learning techniques to generate semantic-aware signatures for identified clusters.

In 2005, Newsome et. al. introduced Polygraph [18] to automatically generate IDS signatures for polymorphic worms. These signatures were based on the Token-subsequence algorithm by elaborating the concept of single substrings to conjunctions.

Tang et al. [19] used bioinformatics technique of sequence alignment to derive Simplified Regular Expression based signatures. Jeffrey et al. [20] presented a two-phase statistical methodology that involved infecting isolated machines with the malware and determining constant regions among different instances for candidate signatures. The second phase calculates the probability for each candidate signature to match a random block of bytes in a random program. The Hancock system [21] presented automatic signature extraction based on several heuristics. It generated a set of signature candidates, selecting the candidates that are not likely to be found in benign code.

Since the release of YARA, various automatic YARA rule generation tools have been proposed that offer full- or semi-automatic signature generation techniques. A few of the actively developed and popular projects are mentioned below:

A. *yarGen*

This tool [22] generates YARA rules for PE files based on strings and opcodes of malware samples, given that they are not present in database of strings and opcodes appearing in good-ware files. Naïve-Bayes Classifier is used to detect useful words instead of compression/encryption garbage. The tool requires 4GB-8GB of memory while running. To create a rule, each identified string is assigned points, based on the contents of the considered strings, adding negative points for strings present in good strings and opcodes database. The scoring system is hard-coded which is a shortcoming in this tool.

B. *yaBin*

AlienVault's Open Threat Exchange (OTX) community released this tool [23] to create YARA rules, based on the rare functions found in the executable code of malicious samples, by looking for function prologues. This tool also relies on the huge good-ware database to exclude common and non-malicious functions. The main idea behind the approach lies in the code re-use, that is, to identify shared part of the code among malware. One of the shortcomings of this tool is in that, it does not work on packed samples.

C. *yaraGenerator*

This tool [24] was developed to handle file types other than PE such as e-mail messages, pdfs, Microsoft Office documents, JavaScript or HTML documents etc. yaraGenerator is also based on strings but unlike yarGen, it considers a set of samples belonging to same malware family at a time and extracts strings common to all of them. This approach is restrictive and also not suitable for larger sample sets.

D. *BASS*

Cisco-Talos developed this tool [25] for automatic generation of ClamAV pattern-based signatures that are different from YARA rules, but the underlying principle is the same. They consist of sequence of characters and opcodes that uniquely identify malware. This tool relies on clustered-samples to retrieve shared code. This is done through a sequence of filtering and elaboration steps that generates single code sequences as signatures. BASS requires disassembly of the executable code. It searches for the longest connected subgraph of a function's similarities graph generated by BinDiff.

E. *YaYaGen*

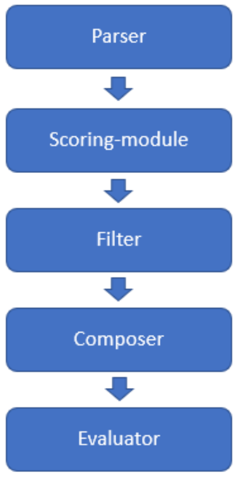
YaYaGen [26] is an automatic signature generator for android malware. It takes in a set of koodous reports, analyzes the reports of the target applications, extracts the analysis attributes, and identifies an optimal attribute subset. YaYaGen is the implementation of the algorithm mentioned in [27]. False positives are limited by applying few heuristic methods.

In summation, each of the above technique has at least one critical limitation. The efficacy of some of these tools relies on the completeness of good-ware strings and functions database which make these tools, memory intensive. Some of the solutions have a limitation on the size or file type of the malware sample. Most of the tools have been developed with an aim at supporting the rule creation, rather than completely replacing the role of expert malware analysts.

We propose an automatic signature generation technique capable of generating sensitive and specific signatures for malware of any size and file type. This technique minimizes the false positive cases by analyzing the malware at byte-code and functional level without executing the malware. Contrary to some of the previous tools, our approach does not suffer from memory constraint.

III. PROPOSED FRAMEWORK

The proposed methodology presented in this section is an improved and adjusted solution. It is derived by cherry-picking the most promising core ideas of the related work outlined in the previous section.



Authorized licensed use limited to: University of Cape Town. Downloaded on May 18,2021 at 23:24:33 UTC from IEEE Xplore. Restrictions apply.

Fig. 1. Overview of the YARA Rule Generator Module.

The proposed framework is modularized into five processing units: Parser, Scoring module, Filter, Composer and Evaluator. From a given input file, the generation of YARA rule file requires processing by each unit. Overview of the proposed framework is shown in the Fig. 1.

Parser module is responsible for reading the input file and necessary feature extraction. The implementation of parser is similar to that of yarGen for string extraction through regex and yaBin for function call extraction through function prologues.

Unlike yarGen, the *Scoring-module* is not hard-coded, rather it is based on a pre-trained Naïve Bayes model. The training of the model is based on the string literals extracted from a data set based on 1714 samples covering 322 families. Each string literal is assigned the family label of the sample from which it was extracted. It is then given as an input for the training of the model. Naïve Bayes assigns conditional probabilities to each string literal based on its occurrence for all given family names. The higher the probability of a string literal for a given family, the better the candidate string it will be for that family. The trained model is stored as a pickle file. This helps us relieve the memory requirement as presented by yarGen (4GB-8GB is required to store good-ware strings and opcodes). These probabilities are used as scores while generating YARA rules.

Filter is responsible for shortlisting few candidate string literals from the extracted ones. This shortlisting is based on the probability scores assigned in the previous step. The shortlisted string literals are then sorted. The highest scoring candidates that appears on top of the list are then selected as tentative choice for the YARA rule. The purpose of this module is to select only those feature strings that are common among given family samples, and are not present in other families.

The *Composer* takes previously shortlisted strings as input and generates a YARA-rule file according to the special syntax followed by the YARA tool. It makes sure that each of the section of YARA: meta, string and condition, is properly defined. Each section is inserted with its relative information as gathered and saved in earlier steps. Finally, the composed rule is written to a file.

Evaluator automates the process of executing YARA rules (over the complete malware data set) and generating an evaluation report. Even though this module is not part of the processing chain to generate YARA rule, it is required to test each of the generated rule. It also calculates the statistical information like true-positives, false-positives and false-negatives for each rule.

IV. RESULTS AND DISCUSSION

The data sets, VirusShare_PE32-Win_GUI_2012, 2013 and 2015, were downloaded from VirusShare website for generation and testing of YARA rules. A total of 13943 samples are present in these datasets out of which 1714 (322 classes) of VirusShare_PE32-Win_GUI_2013 were used for YARA-rule generation. VirusTotal reports were downloaded for each sample which were given as input to AVClass [28] to assign family labels.

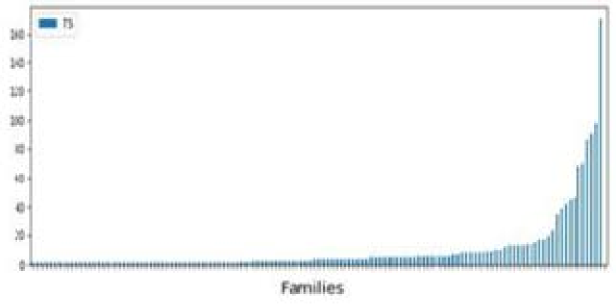


Fig. 2. A bar graph between the families on x-axis and number of samples on y-axis.

The data set used for generation of YARA-rules is diverse in the number of samples present in each family. There are at least two samples in a family, and more than 150 samples at most. This distribution can fairly capture and reflect any possible effects of family size on the quality of YARA-rules generated. The number of families included belong to various malware types, including ransomware, viruses, worms, Trojans, rootkits, downloaders, spyware, botnets etc. The diversity in the families can capture any possible effect due to variant behavior of different types of malware.

The evaluation of each rule was done by calculating the false-positives and false-negatives generated. Balancing between the number of relevant samples (precision) and the coverage (recall), the F1 score is an indicator that describes the trade-off between false-positives and false-negatives we face in the YARA rule generation process.

The TABLE I. contains average precision, recall and F1-score values for three previously existing solutions (yaraGenerator, yarGen, and yaBin) and the proposed framework (on the data-set used for YARA rule generation). yaBin has the best precision value (i.e. 0.959) which means that it has the maximum correct detections out of the total. Whereas, yaraGenerator has the best average recall (i.e. 1.00) which means it does not miss any samples. F1-score values represent a trade-off between both precision and recall. The prototype for the proposed framework has the best F1-score value (i.e. 0.96), which means that on average, the rules generated have the best precision and coverage simultaneously.

	Precision	Recall	F1 Score
yaraGenerator	0.571	1.000	0.661
yarGen	0.793	0.945	0.804
yaBin	0.959	0.929	0.930
Proposed Framework	0.958	0.988	0.960

Authorized licensed use limited to: University of Cape Town. Downloaded on May 18,2021 at 23:24:33 UTC from IEEE Xplore. Restrictions apply.

TABLE I. RESULT COMPARISION FOR FIRST DATASE

The TABLE II. contains comparison of the results for the larger data-set containing 13943 unknown samples (322 classes). yaBin showed second best results with previous data-set, but with a larger data-set, its recall value shows a drastic decrease from 0.92 to 0.48. This indicates that the rules generated by yaBin are highly specific to the data-set for which they are created. The results of the proposed system were better than all of the existing solutions for the larger data-set as well. It has the best F1-score (i.e. 0.7) with a 0.95 average precision, consistent with its previous result.

	Precision	Recall	F1 Score
yaraGenerator	0.487	0.737	0.411
yarGen	0.689	0.535	0.450
yaBin	0.940	0.486	0.568
Proposed Framework	0.952	0.623	0.705

TABLE II. RESULT COMPARISION FOR SECOND DATASET

Most of the generated YARA rules were able to cover the corresponding families without detecting others. The framework generated YARA signatures that reach nearly 0.97 average precision for known malware samples. Another key question is that, would it be possible to even detect samples that were not learned previously? As indicated by the results of second dataset, our approach does classify samples from unknown dataset correctly, with an average precision of 0.95.

V. CONCLUSION

In this paper, we introduced a fully automatic malware signature generation technique. The ultimate goal was to assist malware experts in the generation of malware family signatures. We developed a scalable framework with a generic approach to automatic YARA rule-based signature generation. The main purpose being the identification of new malware samples, while reducing false-positive detection.
Results show that signatures generated by the prototype of the proposed framework are indeed more accurate, with high precision and recall. As well as with a good F1-score compared to the majority of the previously existing tools which were tested. Our solution is more suitable for use in a practical context, where signatures will face thousands of new samples daily.

REFERENCES

[1] Bayer, U., Moser, A., Kruegel, C. and Kirda, E. (2006) "Dynamic Analysis of Malicious Code". Journal in Computer Virology, 2, 67-77. http://dx.doi.org/10.1007/s11416-006-0012-2

[2] Ye Y, Li T, Zhu S, Zhuang W, Tas E, Gupta U, Abdulhayoglu M (2011) "Combining file content and file relations for cloud based malware detection." Proc. ACM international conference on knowledge discovery and data mining (ACM SIGKDD), pp 222–230.

[3] F. Hsu, H. Chen, T. Ristenparty, J. Liz, Z. Su, "Back to the Future: A Framework for Automatic Malware Removal and System Repair", Proc. IEEE Annual Computer Security Applications Conference (ACSAC'06) IEEE Press, July 2006.

[4] YARA. The pattern matching knife. https://github.com/VirusTotal/yara. [Online; accessed 15 April 2019].

[5] H.-A. Kim, B. Karp, "Autograph: Toward automated distributed worm signature detection", Proc. USENIX Secur. Symp., vol. 286, pp. 19, 2004.

[6] S. Singh, C. Estan, G. Varghese, S. Savage, "Automated worm fingerprinting", Proc. OSDI, vol. 6, pp. 4, 2004.

[7] C. Kreibich, and J. Crowcroft, "Honeycomb: creating intrusion detection signatures using honeypots," ACM SIGCOMM Computer Communication Review, Vol. 34(1):51-56, 2004.

[8] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. Introduction to Algorithms. MIT Press, 2001

[9] K. Wang, G. Cretu, S. J. Stolfo, "Anomalous payload-based worm detection and signature generation", Proc. Int. Workshop Recent Adv. Intrusion Detection, pp. 227-246, 2005.

[10] H. Wang, S. Jha and V. Ganapathy, "NetSpy: Automatic Generation of Spyware Signatures for NIDS," Proc. of the 22nd Annual Computer Security Applications Conference (ACSAC'06), IEEE Press, pp. 99-108, 2006.

[11] Z. Li, M. Sanghi, Y. Chen, M.-Y. Kao, B. Chavez, "Hamsa: Fast signature generation for zero-day polymorphic worms with provable attack resilience", Proc. IEEE Symp. Secur. Privacy, pp. 15, May 2006.

[12] K. Rieck, G. Schwenk, T. Limmer, T. Holz, P. Laskov, "Botzilla: Detecting the 'phoning home' of malicious software", Proc. ACM Symp. Appl. Comput., pp. 1978-1984, 2010.

[13] C. Rossow, C. J. Dietrich, "ProVeX: Detecting botnets with encrypted command and control channels", Proc. Int. Conf. Detection Intrusions Malware Vulnerability Assessment, pp. 21-40, 2013.

[14] Y. Xie, F. Yu, K. Achan, R. Panigrahy, G. Hulten, I. Osipkov, "Spamming botnets: Signatures and characteristics", Proc. SIGCOMM, vol. 38, no. 4, pp. 171-182, 2008.

[15] W. Cui, M. Peinado, H. J. Wang, M. E. Locasto, "ShieldGen: Automatic data patch generation for unknown vulnerabilities with informed probing", Proc. IEEE Symp. Secur. Privacy (SP), pp. 252-266, May 2007.

[16] P. Wurzinger, L. Bilge, T. Holz, J. Goebel, C. Kruegel, E. Kirda, "Automatically generating models for Botnet detection", Proc. Eur. Symp. Res. Comput. Secur, pp. 232-249, 2009.

[17] V. Yegneswaran, J. T. Giffin, P. Barford, S. Jha, "An architecture for generating semantics-aware signatures", Proc. USENIX Secur. Symp., pp. 97-112, 2005.

[18] J. Newsome, B. Karp, and D. Song, "Polygraph: Automatically generating signatures for polymorphic worms," Proc. IEEE Symposium on Security and Privacy, IEEE Press, pp. 226-241, 2005.

[19] Y. Tang, S. Chen, "Defending against Internet worms: A signature-based approach," Proc. of IEEE INFOCOM'05, IEEE Press, Vol. 2:1384-1394, 2005.

[20] J.O. Kephart and W.C. Arnold, "Automatic Extraction of Computer Virus Signatures," 4th Virus Bulletin International Conference, 1994.

[21] K. Griffin, S. Schneider, X. Hu, T. Chiueh, "Automatic Generation of String Signatures for Malware Detection," Proc. of the 12th International Symposium on Recent Advances in Intrusion Detection, Springer Press, pp. 101-120, 2009.

[22] Roth F. yarGen. https://github.com/Neo23x0/yarGen. [Online; accessed 15 April 2019].

[23] AlienVault OTX. Yabin. https://github.com/AlienVault-OTX/yabin. [Online; accessed 15 April 2019].

[24] Clark C. yaraGenerator. https://github.com/Xen0ph0n/YaraGenerator. [Online; accessed 15 April 2019].

[25] BASS - BASS Automated Signature Synthesizer. https://github.com/Cisco-Talos/BASS. [Online; accessed 15 April 2019].

[26] Andrea Marcelli and Giovanni Squillero. YaYaGen - Yet Another Yara Rule Generator. https://github.com/jimmy-sonny/YaYaGen. [Online; accessed 15 April 2019].

Authorized licensed use limited to: University of Cape Town. Downloaded on May 18,2021 at 23:24:33 UTC from IEEE Xplore. Restrictions apply.

[27] A. Atzeni, F. Díaz, A. Marcelli, A. Sánchez, G. Squillero, A. Tonda, "Countering android malware: A scalable semi-supervised approach for family-signature generation", IEEE Access, vol. 6, pp. 59 540-59 556, 2018.

[28] MaliciaLab. AVClass. https://github.com/malicialab/avclass.

Authorized licensed use limited to: University of Cape Town. Downloaded on May 18,2021 at 23:24:33 UTC from IEEE Xplore. Restrictions apply.