AN EXPERIENCE IN ENHANCING MACHINE LEARNING CLASSIFIER AGAINST LOW-ENTROPY PACKED MALWARES

Shang-Wen Chen, Tzu-Hsien Chuang,
Chin-Wei Tien and Chih-Wei Chen

Cybersecurity Technology Institute, Institute for
Information Industry, Taipei, Taiwan R.O.C

ABSTRACT

Both benign applications and malwares would take packing for their different purposes to conceal the real part of the program processes. According to recent research reports, existing machine learning (ML) approach-based malware detection engines are difficult to effectively classify the packed malwares, especially when they are in low entropy packed.

Recently, we counted and found that the ratio of low-entropy packed ransomware is extremely high. This would cause a high error rate of the result on currently used ML approaches. Thus, we propose a new method to extract entropy-related features and use a stack model to build up an ML malware engine to effectively detect low-entropy packed malwares. We evaluate our method by using over 15,000 malware samples collected from VirusTotal and compare the result to related researches. This experience reports our adopted model and features can significantly lower the error rate of low-entropy packed detection from 11% to 1%.

KEYWORDS

Malware detection, low-entropy packing, machine learning classification

1. INTRODUCTION

Machine learning has already been widely used in many fields, such as data analytics, predictive analytics, natural language processing (NLP), sentiment analysis, computer vision, and information security. In the field of information security, malware detection has already been applied to improve detection accuracy.

In the normal process of machine learning, the selected features are extracted from each sample of the training set. These features are used to train a model that can be used to detect malware. If the model learns the features of samples correctly, then it can be used to detect malware, which has similar characteristics to such samples. However, the attackers try to use obfuscated techniques to disguise malware to a normal executable file. Attackers adopt several common obfuscation techniques, such as packing, encryption [12], data confusion. After such processing, the extracted features from the sample could differ from the original features. In addition, these incorrect features may lead to incorrect prediction results by the pre-trained model, and this case was discussed by Aghakhani et al. [4].

The above-mentioned problem is commonly solved by determining whether packing is performed before extracting features from the sample. According to the result of judgment, different

David C. Wyld et al. (Eds): CoSIT, AIAPP, SIGL, CRIS, NLPML - 2021
pp. 77-86, 2021. CS & IT - CSCP 2021 DOI: 10.5121/csit.2021.110406

processing methods are used for packed and non-packed samples. Thus, the correct judgment could be used to extract the correct features. Moreover, the correct features can lead to the correct prediction.

Previously, most researchers almost use entropy [1] [9] [10] to judge whether a new sample is a packed. Entropy is a metric used to measure the uncertainty in a series of numbers or bytes. Moreover, packing is a technique that can hide or disguise the internal behavior of samples. In addition, if an attacker uses a packer to pack a sample, the corresponding bytes between the original and packed samples differ considerably. Thus, the entropy value of a packed sample is assumed to be high. That is, a sample with high entropy is assumed to be the same as a packed sample.

However, in some exceptional cases, the results of known methods or tools show the packed samples to possess low entropy. Such cases have already been mentioned in previous studies [8]; however, these were not considered, because the authors claimed that such a case is extremely rare and can therefore be disregarded.

We collected ransomware samples from the VirusTotal dataset [13], with the time interval between July 2019 and June 2020. We used YARA tool [14] and PEPackerInfo [15] to judge whether these samples are packed and determine their entropy values. The threshold value that indicates whether the sample has low entropy is 7, which was also adopted by previous studies [2]. Table 1 shows that low-entropy packed samples not only exist but also take a large proportion in the dataset of our collected packed samples. Obviously, it is quite different with the claim of Han et al [8]. Thus, the larger percentage of low-entropy packing ransomware is, the higher possibility of detection error rate is. If the case of ransomware is extended to all malware, it is not to ignore the influence of low-entropy packing malware to detection error rate anymore.

Table 2 shows the best detection result of the error rate of the machine learning model with entropy-related features in the research of Mantovani et al. [3]. They attempted to determine the effect of low-entropy packed samples on machine learning. Table 2 shows an error rate of at least 11% by the machine learning model in the detection of packing. This proves that the effect of the low-entropy sample is too large to disregard.

Table 1. Statistics of the amount of different entropies in packed samples collected from VirusTotal

	packed	
Collected Time	**high**	**low**
19/7	1	6
19/8	0	23
19/9	1	12
19/10	0	16
19/11	16	66
19/12	3	60
20/1	328	1785
20/2	16	96
20/3	29	339
20/4	18	1209
20/5	36	1331
20/6	8	640

Table 2. Best result of error-rate detection using machine learning with entropy-related features [3]

Classifier	Train-Testing	$Err_{notPacked}(W)$	$Err_{packed}(W)$
MLP	75%~25%	6.34%	12.70%
	50%~50%	6.87%	16.14%
	25%~75%	6.89%	11.91%

We have had considered an assumption that if the threshold value is changed from 7 to another nearby values, the error rate of machine learning may be decreased. However, this assumption has been proved that it is almost not effective for reducing the threshold value. On the contrary, the error rate could be increased. Figure 1 shows 341 distinct entropy values in each range in our collected low-entropy dataset and the sum of the amount of the different entropies within the ranges 5-6 and range 6-7 is 93.5% of the total amount. Because the values are centralized near 7, if this threshold is decreased, the error rate could increase. Therefore, we tried to find a method to lower error rate without changing threshold value. That is, the objective of this study is to decrease the detection error rate of machine learning in low-entropy packing malware so that a new model could detect all low-entropy packing malware correctly.

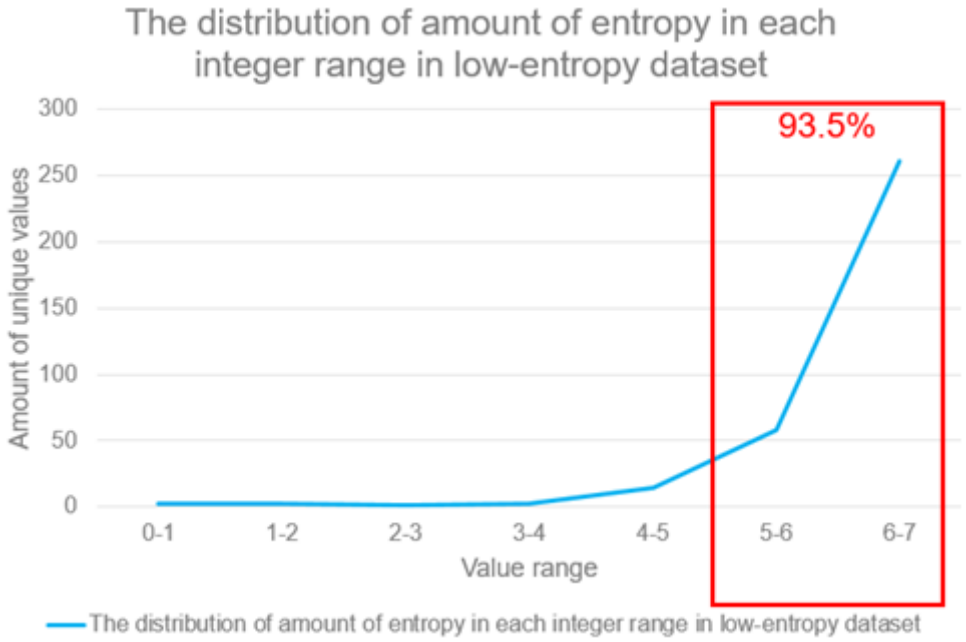

Figure 1. Distribution of the amount of distinct entropy values in each integer range in the low-entropy dataset

The remainder of this paper is organized as follows: Section 2 provides an overview of literature on entropy packing detection and describes the central concept of our adopted model. Section 3 introduces and details our proposed model. Section 4 details the results of experiments with different datasets. Section 5 presents the conclusions of this study.

2. RELATED WORKS

This section first reviews the literature on entropy packing detection in recent years, and then we detail as to why we adopted our model as the solution.

As described in section 1, entropy has become an indicator of packing in research. Thus, many studies [7] [8] have adopted it as main feature or use some of its features in packing.

In 2008, Perdisci et al. [11] proposed features that captured specific anomalies introduced by packers in the portable executable (PE) [29] file format. The authors applied pattern-recognition techniques for fast detection of packed executables so that only executables detected as packed are sent to an universal unpacker such as UPX [16], PackerID [17], and NFD [18]. However, the limitation of such a method is that unknown packed files cannot be unpacked because of the use of universal unpacking tools. In contrast, we aimed to detect all packing samples that are not limited to only those packed by universal packers.

In 2012, Ugarte-Pedrero et al. [5] selected entropy as the main unpacking feature, and conducted several experiments by using some samples of the Zeus botnet. Zeus is one of the first bot families to adopt a low-entropy packing scheme. However, their proposed method was customized to a single specific case. Thus, it fails to consider the samples that use other common low-entropy techniques.

In 2016, Raphel et al. [6] attempted to refine the use of entropy to recognize samples adopting an XOR-based scheme. XOR encryption is recognized as a form of obfuscation that is mainly used to encrypt small parts of a code such as shellcodes. However, their solution aims to a specific problem, and the method is therefore not applicable to common packing detections. In contrast, our solution can be used for common packing detection.

These researches attempted to solve the entropy-packing detection problem, but their methods could not either be a general solution or handle samples which are packed with unusual packers. As described in section 1, low-entropy packing problem displays that entropy does not be a directly indicator of packing anymore. However, we believe that entropy is still an effective factor to judge packing problem. Thus, we proposed a new usage of entropy-related features and used these features in our proposed model. Several different algorithms have been proposed for packing detection in recent years, for example, random forest [19], deep neural network [20]. Although each algorithm has its own advantages, we combined these advantages for building our model. Thus, we adopted a stacked model that can combine the results of different models to output a balanced result. As a result, the error rate of the model that uses new entropy-related features decreases effectively.

The contribution of this study to literature is two-fold. First, we propose a new method for extracting entropy-related features. Second, our adopted stacked model effectively decreases the error rate by verifying 15,000 samples from two datasets with 0.25% and 0.46% error rates.

3. PROPOSED METHOD

3.1. Detailed Description for Our Adopted Model

In this subsection, we discuss in detail why we chose to combine the advantages of algorithms. The first reason is the stability of performance of different algorithms. As described earlier, different algorithms have their own advantages because of their own characteristics. For example, the support vector machine (SVM) [21] is good at bisection question [33]. However, SVM does not perform well in other aspects. Thus, the total performance of a single algorithm is considered unstable, and therefore we disregarded the use of only one algorithm to build a model. The second reason is the overfitting problem. Each model has the possibility to overfit a specific dataset under different conditions. However, the possibility of overfitting multiple models is

lower than the possibility of overfitting of a single model. Thus, we adopt a stacked model for machine learning. The model comprises three characteristics: the use of multiple models, merging the results of multiple models, and lowering the error rate.

3.2. Model Framework Description

Figure 2 depicts the structure of stacked model the inputs of which are the features extracted from samples. The model outputs the percentage of unpacking and packing. The process of the stacked model is divided into two stages.

The first stage portrays multiple models, the inputs of which are the same as those of the stacked model. Although we adopted the four models displayed in the top part of Figure 2 in the first stage, they can be replaced by the other models when needed. The outputs of each model display the corresponding percentage value of unpacking and packing.

In the second stage, the inputs of the stack model comprise eight output values of the models in the first stage. These values are processed by the stacked model, and the balanced values obtained after processing are the outputs of the stacked model.

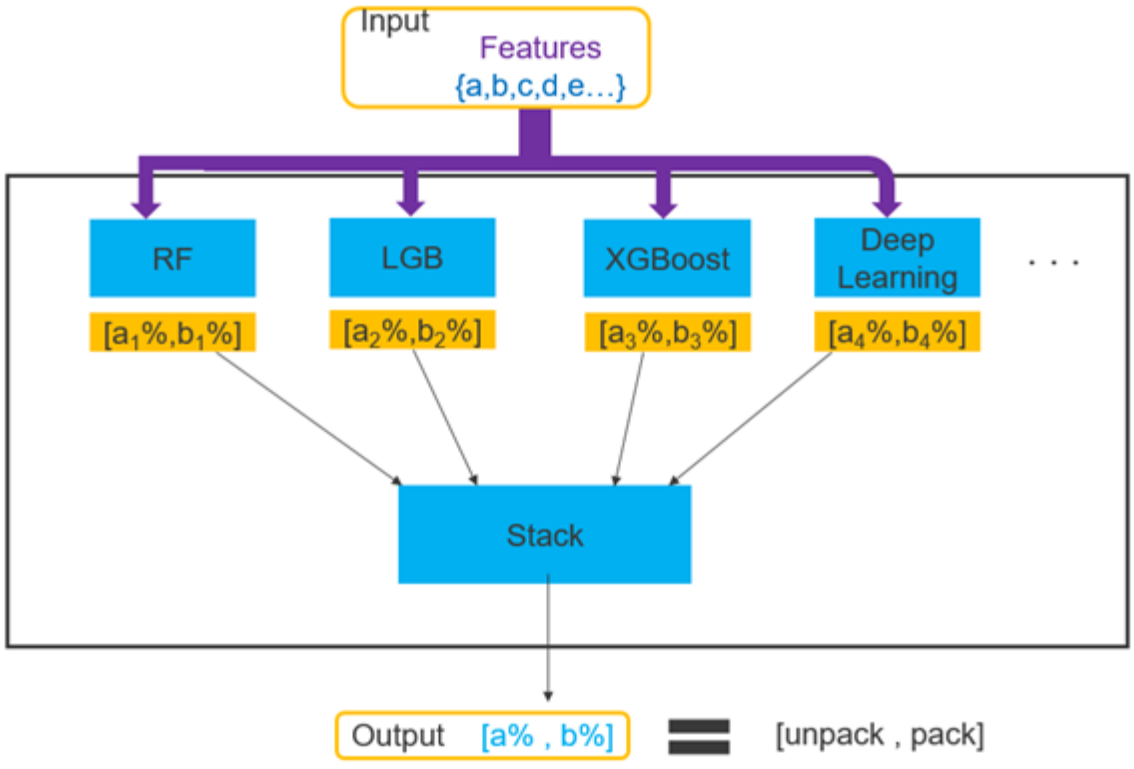

Figure 2. Structure of the proposed stacked model

3.3. Feature Selection

In this study, we adopted 7,068 features in our model, and the detailed composition is described as follows. Our features are divided into three categories: assembly, byte and keyword features. Moreover, we used IDA pro [22] and xxd [23], which is a Linux command, to generate the required assembly and byte files.

For assembly features, we adopted opcode, registers, and metadata, for which we selected 26 common registers in the x86 [24] architecture. Similarly, we also chose 93 common opcodes in the x86. Finally, we selected 26 metadata of a sample, comprising data such as file size and total lines of assembly file.

For byte features, we adopted 1-gram, metadata, byte string lengths, image, and entropy as bytes features. The image features were acquired using mahotas [25], which is a Python package, to calculate haralick [26] features of a byte image. Further, we adopted 15 entropy-related features in this study. Figure 3 shows the distribution of the number of sections in each sample in the collected dataset. As shown, the most common number of sections in a sample is five. However, 15 is the largest number of sections in a sample. To adapt the model itself to most sample cases, we decided to adopt 15 entropy values from sample sections. Assuming that the total number of sections in a sample is k; if $k < 15$, the remaining 15-k parameters will be assigned as zero. On the contrary, if $k \geq 15$, only the first 15 entropy values will be used, while the others are disregarded by the system.

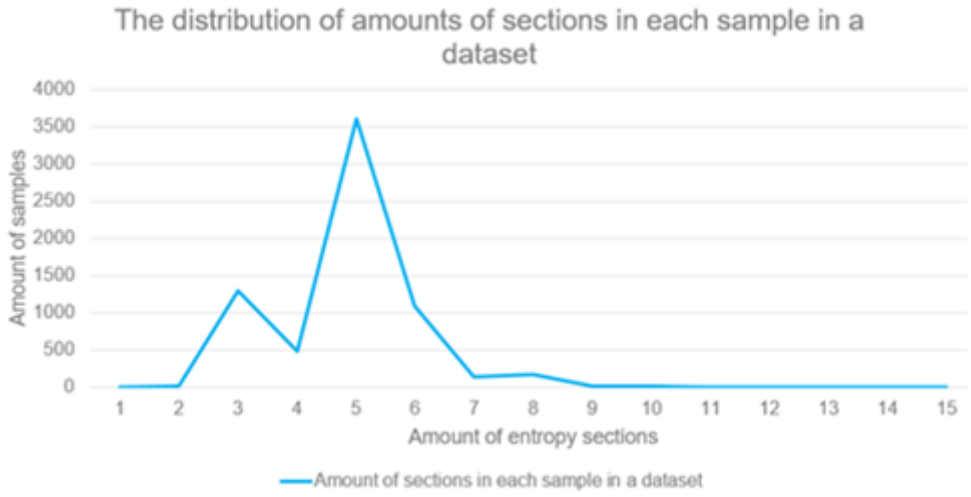

Figure 3. Distribution of amounts of sections in each sample in collected dataset

For keyword features, we counted the number of occurrences of 6,482 keywords. The keywords were decided by the following process. First, we filtered the keywords that occur over 100 assembly files. Then, these keywords were filtered twice using the feature_importance function of the XGBoost [32] model per 10,000 keywords. Next, we acquired 8,580 keywords from the filtered processing, and then erased unnecessary keywords, such as memory locations and uncompiled data. Finally, we obtained a total of 6,482 keywords.

3.4. Model-Training Method

To prove the effectiveness of the new usage of entropy-related features, we generate two feature sets: one containing all of the extracted features from the sample and the other containing the extracted features without the 15 entropy-related features. Then, these two feature sets were used to train their models separately.

4. EXPERIMENTS

4.1. Dataset

We used two datasets in our experiments: one was acquired from [3], and the other was collected from VirusTotal; these are termed as dataset1 and dataset2, respectively. The samples collected from VirusTotal are all ransomware, and the collecting period is between May 29 and June 30, 2020. We selected 15,000 and 5,000 samples from dataset1 for training and testing, respectively. Similarly, for dataset2, we selected 10,000 samples each as the training and testing sets.

4.2. Data Labelling

For dataset1, we adopted the labels used by Mantovani et al. [3]. The training set of dataset1 comprises 7,500 packed and unpacked samples each. In addition, the testing set of dataset1 comprises 2,500 samples each as packed and unpacked. For dataset2, the labels were decided using our proposed processing procedure, which is described in the following text. As a result, we obtained 5,000 samples each as packed and unpacked samples in the training set as well as in the testing.

The proposed processing procedure has two stages. In the first stage, the input was a new unknown sample, which was filtered by known tools first. The tools that we used in this stage were PackerID, NFD, DIE [27] and ExeScan [28]. If the output of any tool indicates that the unknown sample is packed, then this sample will be labelled as packed. However, if all of the outputs of the four tools indicate that the unknown sample is unpacked, it will enter the second stage.

The second stage uses five static features to distinguish whether the sample is packed. The names and detailed descriptions are listed in Table 3. We also tested the performance of several other static features, such as the amounts of executable sections, entropy of Portable Executable (PE) [29] header, and entropy of whole file. However, the most distinguishable features are the five features that we adopted.

Table 3. Description of static features

Features name	Descriptions
rwx section number (rwx)	The amounts of sections that can read, write, and execute simultaneously
Non-standard section number (nss)	The amounts of sections, the name of which do not exist in the Microsoft list
Execution only section number (exe)	The amounts of sections that can only perform the execute task
[.text section] virtual size > rawdata size (.text)	In general, the size of a virtual address is greater than the size of rawdata in the .text section
Import number of address table & .dll (iat & dll)	The amount of import address tables < 50 and dll < 4

Figure 4 shows the processing flow chart of the second stage. The sample was judged according to a specific order of static features sequentially. This process involves the following five steps, the results of which is one of True or False. Step 1 analyzes the sample to check whether there exists at least a section with the access authorization of read, write and executable simultaneously. In step 2, the sample is examined to check whether there exists at least a section name that is not listed in the standard section name list of Microsoft [30]. In step 3, the sample is

examined to check whether the virtual size of .text section is greater than the rawdata size of .text section. In step 4, the sample is analyzed to check whether there exists at least one section that owns only the accessibility of executable. In step 5, the sample will be checked to determine whether there exists at least one section with <50 import address table and <4 dynamic link library (dll) [31].

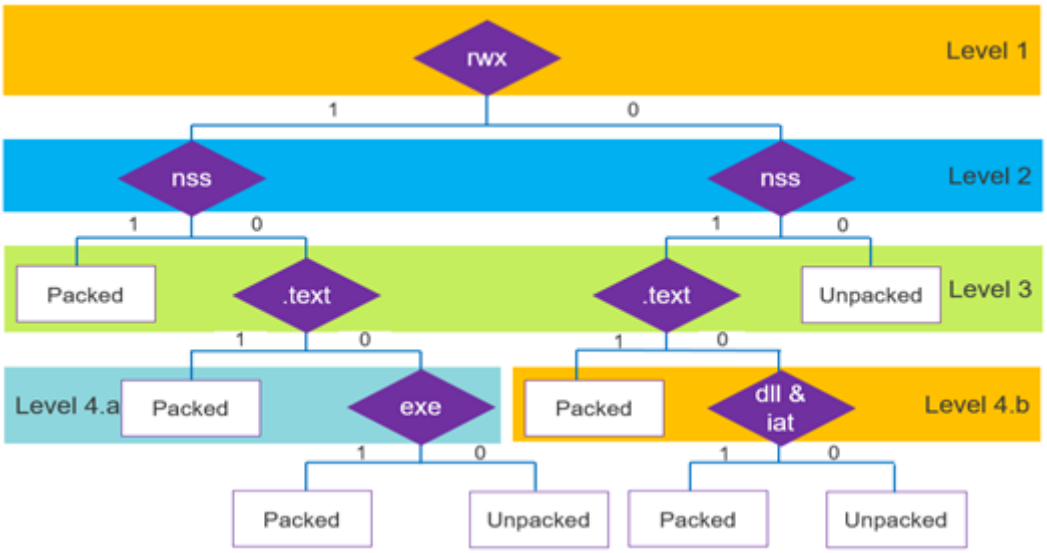

Figure 4. Diagram of processing flow chart of second stage in data labelling

There are five levels in Figure 4, and the concepts of labelling process in each level described as follows:

Level 1: judge if there exists any section that owns access authority of read, write and execute at the same time
Level 2: judge if there exists any non-standard section in sample
Level 3: judge if there exists the case that the size of virtual address > the size of rawdata in the text section
Level 4.a: judge if there exists any section which only owns execution authority
Level 4.b: judge if there exists any section whose amount of import address tables < 50 and dll < 4

4.3. Experimental Result

We conducted two experiments, one using dataset1 and the other using dataset2. In each experiment, we trained two models with different features. One uses all extracted features from the dataset, while the other uses all extracted features except the entropy-related features from the same dataset.

Table 4 shows the result of the error rate of dataset1. We also compared our results with the results of Mantovani et al. [3]. Parameter w indicates the vectors of all features, and parameter w' indicates the vectors of all features except the entropy-related features. We observed that the error rate of the proposed model with all features of packed samples is better than that of the model in [3]. Moreover, the error rate of the packed samples with all features in our model was only 0.25%.

Table 4. Performance of error rate of dataset$_1$

	$Err_{unpack}(w)$	$Err_{packed}(w)$	$Err_{unpack}(w')$	$Err_{packed}(w')$
Dataset$_1$	0.76%	0.25%	0.8%	0.29%
Mantovani et al. [3]	6.89%	11.91%	6.33%	12.93%

Table 5 displays the results of the error rate of dataset2 compared with the results of Mantovani et al. [3]. As shown, the error rate of packed samples using the proposed model with all features is still better that of the previous model [3]. In addition, the error rate of our model to dataset2 is 0.46%.

Table 5. Performance of error rate of dataset$_2$

	$Err_{unpack}(w)$	$Err_{packed}(w)$	$Err_{unpack}(w')$	$Err_{packed}(w')$
Dataset$_2$	1%	0.46%	0.88%	0.48%
Mantovani et al. [3]	6.89%	11.91%	6.33%	12.93%

Moreover, we observed that the error rate of packed samples with all features is better than the error rate of packed samples without entropy-related features, as shown in Tables 4 and Table 5; therefore, the effectiveness of the proposed usage of entropy-related features is proved. In summary, our proposed model can effectively lower the error rate of detection of packed samples, and our new usage of entropy-related features helps to reduce the error rate of the model.

5. CONCLUSION AND FUTURE WORK

In this paper, we proposed a new method for extracting entropy-related features. We combined these features and other common features and applied them to our adopted stacked model. Moreover, our stacked model effectively decreases the error rate. We verified the performance of our model using 15,000 samples from two different datasets, the error rates of which were obtained as 0.25% and 0.46%.

The training samples of dataset2 that we used in the experiments were collected from 29 May to 30 June, 2020. Moreover, we continuously collected ransomware samples from VirusTotal. We hope to train new models with different lengths of time intervals and evaluate their performances. Then, we can decide the best time interval to retrain the model to keep a better performance.

REFERENCES

[1] G. Jacob, P. M. Comparetti, M. Neugschwandtner, C. Kruegel, G. Vigna, “A Static, Packer-agnostic Filter to Detect Similar Malware Samples,” in Proc. of the Conference on Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA), 2013

[2] X. Ugarte-Pedrero, D. Balzarotti, I. Santos, P. G Bringas. “Sok: Deep packer inspection: A longitudinal study of the complexity of run-time packers.” in 2015 IEEE Symposium on Security and Privacy (SP) , 2015, pp. 659–673.

[3] A. Mantovani, S. Aonzo, X. Ugarte-Pedrero, A. Merlo, D. Balzarotti, “Prevalence and Impact of Low-Entropy Packing Schemes in the Malware Ecosystem,” in: Network and Distributed System Security (NDSS) Symposium, NDSS 20, 2020.

[4] H. Aghakhani, F. Gritti, F. Mecca, M. Lindorfer, S. Ortolani, D. Balzarotti, G. Vigna, C. Kruegel, “When Malware is Packin’ Heat; Limits of Machine Learning Classifiers Based on Static Analysis Features,” in Network and Distributed System Security (NDSS) Symposium, NDSS 20, 2020

[5] X. Ugarte-Pedrero, I. Santos, B. Sanz, C. Laorden, P. G. Bringas. “Countering entropy measure attacks on packed software detection,” in Consumer Communications and Networking Conference (CCNC), 2012, pp. 164–168.
[6] J. Raphel, P. Vinod. “Information theoretic method for classification of packed and encoded files,” in Proceedings of the 8th International Conference on Security of Information and Networks, SIN ’15, ACM, New York, NY, USA, 2015, pp. 296–303.
[7] R. Lyda, J. “Hamrock. Using entropy analysis to find encrypted and packed malware,” IEEE Security & Privacy, vol. 5, no. 2, 2007
[8] S.-W. Han, S.-J. Lee. “Packed pe file detection for malware forensics,” KIPS Transac.: PartC, vol. 16, no. 5, pp. 555–562, 2009.
[9] M. Z. Shafiq, S. Tabish, M. Farooq, “PE-Probe: Leveraging Packer Detection and Structural Information to Detect Malicious Portable Executables,” in Proc. of the Virus Bulletin Conference (VB), 2009
[10] R. Arora, A. Singh, H. Pareek, U. R. Edara, “A Heuristicsbased Static Analysis Approach for Detecting Packed PE Binaries,” International Journal of Security and Its Applications, 2013
[11] R. Perdisci, A. Lanzi, W. Lee. “Classification of packed executables for accurate computer virus detection,” Pattern Recog. Lett., vol. 29, no. 14, pp. 1941–1946, 2008.
[12] J. A. Clark, “Invited Paper. Nature-Inspired Cryptography: Past, Present and Future,” Citeseer, pp. 1647-1654, 2003.
[13] VirusTotal. https://www.virustotal.com
[14] GitHub–Yara-Rules/rules: Repository of yara rules. https://github.com/Yara-Rules/rules
[15] PEPackerInfo. https://sites.google.com/site/robertoperdisci/projects/cpexe
[16] UPX. https://upx.github.io/
[17] PackerID. https://github.com/sooshie/packerid
[18] NFD. https://github.com/horsicq/Nauz-File-Detector/releases/
[19] L. Breiman. Random forests. Machine Learning, 45(1): 5–32, 2001.
[20] I. Arel, D. Rose, R. Coop, "DeSTIN: A Scalable Deep Learning Architecture with Applicationto High-Dimensional Robust Pattern Recognition," Proc. of the AAAI 2009 Fall Symposiumon Biologically Inspired Cognitive Architectures (BICA), November, 2009
[21] Osuna E, Freund R, Girosi F (1997) Support vector machines: Training and applications
[22] IDA Pro – Hex Rays. https://www.hex-rays.com/products/ida/
[23] xxd linux command man page. https://www.commandlinux.com/man-page/man1/xxd.1.html
[24] coder32 edition | X86 Opcode and Instruction Reference 1.12. http://ref.x86asm.net/coder32.html
[25] mahotas. https://pypi.org/project/mahotas/
[26] Mahotas – Haralick features. https://www.geeksforgeeks.org/mahotas-haralick-features/
[27] DIE. https://github.com/horsicq/DIE-engine/releases
[28] ExeScan. https://github.com/cysinfo/Exescan
[29] M. Pietrek, “Peering Inside the PE: A Tour of the Win32 Portable Executable File Format,” Microsoft Systems Journal, vol. 9, no. 3, 1994, pp. 15-34.
[30] PE Format. https://docs.microsoft.com/en-us/windows/win32/debug/pe-format#special-sections
[31] Dynamic-link library. https://www.computerworld.com/article/2585730/dynamic-link-libraries.html
[32] XGBoost Documentation. https://xgboost.readthedocs.io/en/latest/
[33] Bisection method. https://www.sciencedirect.com/topics/engineering/bisection