Industrial Engineering Journal
ISSN: 0970-2555
Volume : 51, Issue 8, August : 2022

Design and development of anti-malware systems to ensure the identified threat using LSTM

[1]Dharavath Mahesh, [2]Potnuri Triveni, [3]Jhansi Modem, [4]Poreddy Preethi

[1,2,3]Assistant Professor, [4]UG Student, [1,2,3,4]Dept. of Computer science Engineering, Visvesvaraya College of Engineering and Technology, Mangalpalle, Telangana, India.

ABSTRACT
Due to the widespread use of information and communication technology (ICT) applications in daily life, malicious software threats and their detection are becoming a more important sub domain of information security. The identification of all malware is one of the most difficult problems in the design and development of anti-malware systems. Dynamic analytic algorithms' development, which allows for quick detection of polymorphic and metamorphic malware, is crucial. We present a technique for detecting malicious code by analyzing run trace data using Long Short-Term Memory (LSTM) (LSTM). Malicious and benign Portable Executable (PE) files' execution traces were modeled. Beginning with run trace outputs gathered through dynamic analysis of PE files, we built our first dataset. With a dataset that includes both benign and harmful applications, the proposed method was shown to have an accuracy rate of more than 98 percent after undertaking thorough testing.
Keywords: LSTM, malware, deep learning, classification

INTRODUCTION

Malware is software that has been created to do damaging actions including stealing confidential information, gaining root access, and incapacitating targeted computers. Meanwhile, a wide range of malware has emerged as a result of the Internet's and the software industry's fast expansion. The total amount of malware samples has increased by almost 34% over the last three quarters, reaching more than 774 million samples. Malicious software (also known as malware) has been increasing over time. Hence, malware detection is a topic that is constantly intriguing and important. How to recognize malware has been the subject of extensive research. Because they have a limited ability to detect new threats, static signature-based anti-viruses are frequently utilized to identify malware. New malware may easily avoid signature-based security measures detection if it was encrypted, obfuscated, or packed to avoid detection. This detection method may be circumvented by zero-day malware. Analyzing a system in real-time In contrast to code obfuscation strategies, lurenjie17@mails.ucas.ac.cnis a more effective malware detection tool. A safe and regulated environment, such as a virtual machine, simulator, sandbox, etc., is often required for dynamic behavior-based malware detection approaches [4] [5]. In the next step, behavioral analysis is carried out utilizing information gathered from interactions with the environment, such as API calls and DLL calls. Though extensively explored, these methods are inefficient when used on big datasets [6]. It takes a lot of time and effort to safeguard the operating environment from being tainted by dynamic behavior-based malware detection approaches. Malware detection strategies that use machine learning have been presented in the last several years. Amalware detection approach based on data mining was initially published in reference [7]. It usesthree kinds of static features: the PE header, a text sequence, and a byte sequence to identify malware. Using n-grams instead of byte sequences, Kolter and Maloof [8] examined the performance of naïve Bayes, decision trees, and support vector machines for virus detection. An artificial neural network [9]was also employed for malware detection in the later years [9,10]. There are also new approaches to detecting malware. Malware may be detected using image processing in both [11] and [12]. However, the prior effort has been successful enough in terms of malware detection. Machine learning classifiers are trained by manually analyzing malicious code and comparing it to the characteristics extracted from the code itself. An innovative and efficient approach to identify whether a Windows executable file is

malware has been proposed in this research to lower the expense of artificial feature engineering. It is necessary to retrieve the executable files' assembly format file by disassembling them using IDA Pro first. To extract the opcode sequence from each assembly format file, we need an algorithm. Word embedding technique [13] and long-short term memory (LSTM) [14] are then utilized to understand the feature vector representation of opcode and to automatically learn opcode sequence patterns of malicious software. After the second LSTM layer, we add a mean-pooling layer to boost the local feature representation's invariance. We ran a series of tests on a dataset including 969 malicious files and 123 benign ones to see whether our strategy worked. MalwarearXiv:1906.04593v1 [cs.CR] 10 Jun 2019 detection performance is evaluated in the experimental phase, and comprehensive performance comparisons with other similar studies are performed. The assessment results reveal that our suggested technique can obtain an average AUC of 0.99 for malware detection and an average AUC of 0.987 for malware classification.

Literature survey:

In contrast to Static analysis, malware analysis techniques based on Dynamic analysis are more resistant to obfuscation. Dynamic analysis was used to classify API requests that lasted less than five minutes in [1]. AUC, a quality metric, was computed using 170 samples and yielded a score of 0.96. Privately gathered samples of both benign and malicious software have been used to create a feed- forward network using API call feature sets. It performs well compared to previous methodologies,but it lacks research on the speed of execution, which is critical for real-time deployments. ESN and RNN tests were undertaken in [3] to learn malware's language. The ESNs performed better than RNNs in the majority of the studies. Experiments were undertaken in [4] to identify when to cease the virus execution about network traffic, as stated in [5]. Conventional procedures require 67 percent more time to complete than this approach. With API calls long sequences as features, the RNN and its version long short-term memory (LSTM) and CNN were used for malware classification in [6]. The main issue with the current approaches is that they need a lot of time to examine the behavior of the system during operation. It was used in [7] to classify malware using system call sequences as a CNN and RNN hybrid. SVM and the hidden Markov model had previously been used to acquire these system calls, but Dynamic analysis was employed to obtain them, and it was shown to be more effective (HMM). However, the biggest issue is the lack of discussion of the relevance of execution time in real-time virus detection. Using RNNs and two datasets, [13] proposes a technique. Additionally, they looked at the performance of several well-known classical machine learning classifiers. With a 5s execution time, they had claimed 94 percent accuracy. Static, dynamic, and hybrid analysis-based malware detection methods have been the subject of several investigations. HMM was used for both static and dynamic analysis of feature sets and a comparison of detection rates for a large number of malware types in [8]. In general, they found that Dynamic analysis had the best detection rate WindowsDynamic-Brain-Droid (WDBD) is a model we developed to compare and contrast several traditional machine learning algorithms (MLAs) and deep learning architectures to determine which technique is best for classifying Windows malware. The quantity of malware and benign samples in our datasets varies depending on the execution time, hence we make use of two separate datasets.

Proposed work

Long Short-Term Memory (LSTM)

Long Shor-Term Memory (LSTM) is a specialized RNN architecture and the most im- important feature of this advanced architecture is solving the vanishing gradient problem or at least decreasing the effect of the vanishing gradient problem on training performance. Similar to RNN, nodes in an LSTM neural network get hidden states from the previous step. However, a common LSTM unit, node, has an improved structure compared to RNN, which is the main factor providing long-term memory by decreasing the effect of the vanishing gradient.

A common LSTM unit gets an input value and generates an output value. During this operation, it uses two

Industrial Engineering Journal
ISSN: 0970-2555
Volume : 51, Issue 8, August : 2022

values, including the generated output value of the current cell and the cell state value of the previous cell that will be explained in the following paragraphs, transferred by the previous step. An LSTM unit was designed to carry out the following three tasks.

- Forget unwanted information in the current cell state through the forget gate
- Add new information to the current cell state through the input gate
- Create output of the current cell state through the output gate

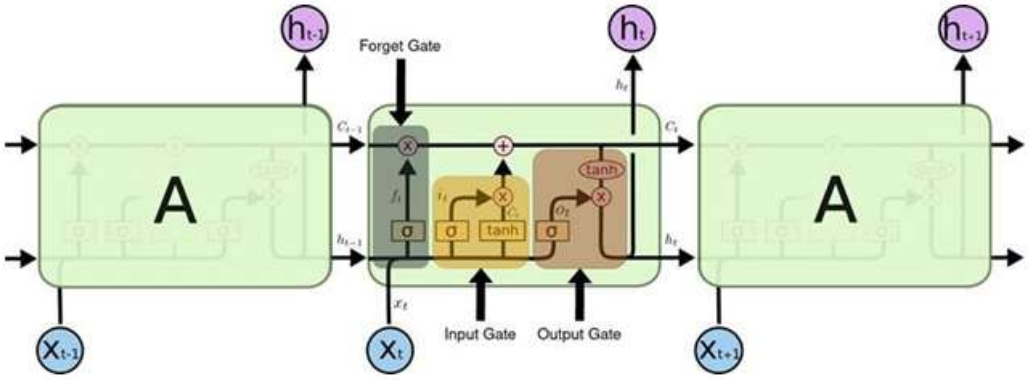

Fig-1: The interior design of a common LSTM cell [24]

As seen in Fig-1, the inside of an average LSTM unit is simple and functional. It is possible to use a sigmoid function to generate an output that is between 0 and 1 on the left side of the cell bypassingthe input from this current cell (Xt) and its output (ht1) to it from the previous cell (ht1). Afterward, this number, ft, is multiplied by the previous cell state, Ct1, to update and construct the currently displayed cell state. Essentially, the cell state is a value that travels across cells to transport information between them. This multiplication operation dictates which information will be forgotten and how much will be remembered in the next cells, and as a result, this section of the unit is referred to as the forget gate [26].

There are two sigmoid functions and one tan h function in the center of the cell, and their outputs are multiplied together. In this section, the sigmoid function is given the input of the current cell Xt as well as the output of the preceding cell ht1 as inputs for the second time. In contrast to the sigmoid function used in the forget gate, the output value of this sigmoid will be utilized to designate which value will be freshly added to the current cell state, as opposed to the sigmoid function used in the forget gate. The tan h function generates an array of potential values that may or may not be added to the current cell state in the future. The values that will be added to the current cell state are decided bymultiplying the output of sigmoid it by the output of tan h C t. This is done by multiplying the output of sigmoid it by the output of tan h C t. The prior cell state Ct1 that was modified by the forget gate is updated again with the new information from the input with the assistance of the add operation, resulting in the creation of the final current cell state. As a result, this portion of the LSTM unit is referred to as the input gate [26].

Industrial Engineering Journal
ISSN: 0970-2555
Volume : 51, Issue 8, August : 2022

Algorithm 1: Algorithm for Mawere analysis

input : run Trace Pool = {f1, f2, ..., fN } where
N = 290output: accuracy Rate, loss
for f ∈ run Trace Pool do lines ← f .read Lines (); if f is malware then
merged Malicious. Write
File (lines);else
merged Benign. Write File (lines);
for l ∈ merged Malicious do sequences. value ← Append(l); sequences. label ← Append(0);
/* "0" is used to label malicious sequences */
for l ∈ merged Benign do sequences. value ← Append(l); sequences. label ← Append(1);
/* "1" is used to label benign
sequences */ for s ∈ sequences do
ts ← Tokenize(s);
tokenized Sequences ←
Append(ts);for ts ∈
tokenized Sequences do
es ← Encode(ts);
encoded Sequences ←
Append(es);for es ∈

The sigmoid and the tan h functions, which are located on the right side of the cell, are used to indicate the output value of the current cell in the equation. This method obtains the most recent cell state, which may be referred to as the current cell state Ct, and uses it to generate an output value for the current cell that is between -1 and 1. A different approach is used by the sigmoid function, which generates an output Ot that will be used to determine which portions of the current cell state will be included in the output of the current cell ht. The output gate [26] is the name given to this section of the LSTM unit.

Overall, the current LSTM cell receives the output and cell state of the previous cell in addition to the input of the current cell and creates the output of the current cell by updating the cell state of the previous cell When compared to the basic RNN architecture, the sequential internal design of the LSTM architecture provides for superior efficiency when dealing with data consisting of extended sequences of events.

1. Results & discussionDataset:

The dataset (Drebin-215) also contains 215 functionalities from 15,036 app samples, of which 9476 were benign, and the remaining 5560 were malware samples from the Drebin project [4]. The dataset (Drebin-215) also contains 215 features from 15,036 app specimens, of which 9476 were normal, and the residual 5560 were malware data points. The Drebin samples are also freely accessible to the public and are extensively utilized in the scientific community. There are two datasets available for download in the supplemental material: Drebin-215 and Malgenome-215.

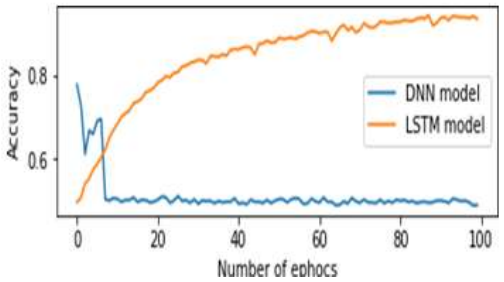

Fig-2: Accuracy

Industrial Engineering Journal
ISSN: 0970-2555
Volume : 51, Issue 8, August : 2022

Classification Whenever we say "accuracy," we typically refer to the degree to which something is accurate. The ratio of correct predictions to the total number of input samples is a valuable indicator of accuracy in a forecasting model.

$$\text{Accuracy} = (\text{TP} + \text{TN}) / (\text{TN} + \text{FP} + \text{TP} + \text{FN})$$

Here fig-2 represents the accuracy of the classification of malevolent samples and benevolent samples. And the graph compares the existing ANN model and proposed LSTM model. ANN model fails to perform an accurate variety of harmful and harmless pieces. Because malicious code is a seriesof actions, performed and ANN fails to memorize code sequences. But the LSTM model contains a memory unit it can give improved accuracy while the quantity of epochs rises. At the same time, ANN fails to provide enhanced accuracy while the number of ages is enlarged.

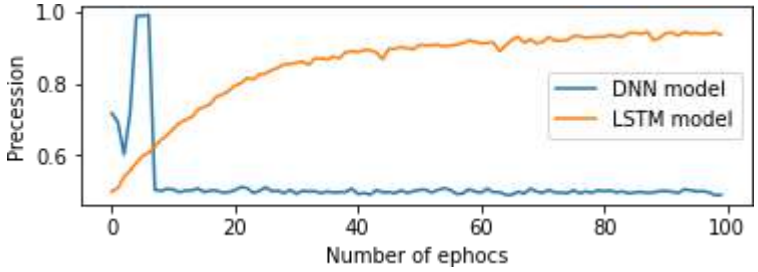

Fig-3: Precession

Precision may be defined as the ratio of actual positives to all true positives and false positives. Precision examines the data to determine how many false positives were included in the mix. If there are no false positives (those FPs), the model's accuracy is 100 percent. The greater the number of FPs introduced into the mix, the more ugly that precision will seem.

Precision = TP/(TP + FP)

Here fig-3 represents the classification of malicious samples and benign samples precession. And the graph compares the existing ANN model and proposed LSTM model. ANN model fails to give a better precession classification of malicious and benign examples. Because malicious code is a series of actions, performed and ANN fails to memorize code sequences. But the LSTM model contains a memory unit it can give better precession while the number of epochs increases. At the same time, ANN fails to provide when the number of epochs increases.

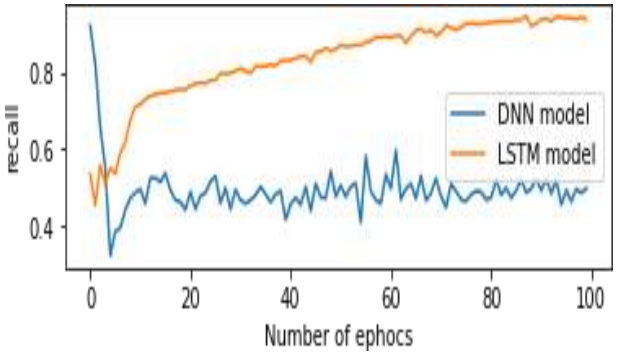

Fig-4: Recall

It is calculated as the quantity of accurate optimistic findings separated by the total amount of appropriate samples.

Precision = TP/(TP + FN)

Here fig-4 represents recall for the classification of malicious samples and benign samples. And the graph compares the existing ANN model and proposed LSTM model. ANN model fails to give better recall of malicious and benign examples. Because malicious code is a series of actions, performed and ANN fails to memorize code sequences. But the LSTM model contains a memory unit that can give better recall while the number of epochs increases. At the same time, ANN fails to provide when the number of epochs increases.

Industrial Engineering Journal
ISSN: 0970-2555
Volume : 51, Issue 8, August : 2022

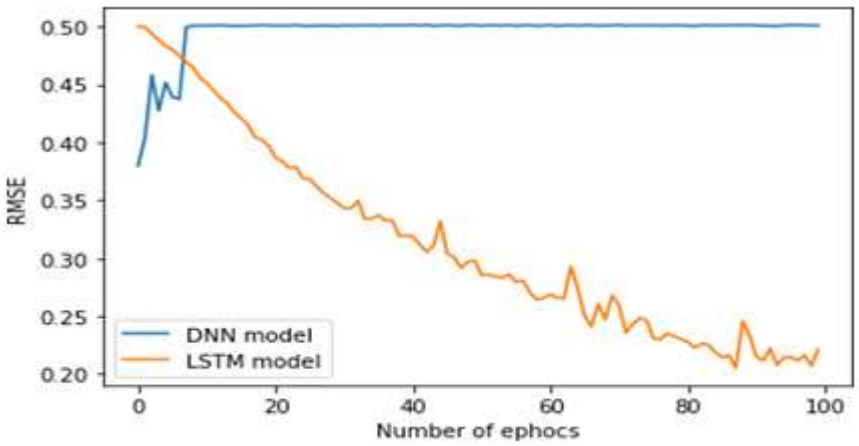

Fig-5: RMSE

In statistics, the standard deviation of the mistakes that occur when a prediction is made on a dataset is the RMSE. This is the same as MSE, except that the basis of the number is considered when evaluating the model's accuracy.

Here fig-5 represents the classification RMSE of malicious samples and benign samples precession. The graph compares the existing ANN model and proposed LSTM model. ANN model gives an error value between 0.4 to 0.6, whereas LSTM produces a low RMSE value. Because malicious code is a series of actions, performed and ANN fails to memorize code sequences. But the LSTM model contains a memory unit that can give better performance while the number of epochs increases. At the same time, ANN fails to provide when the number of epochs increases.

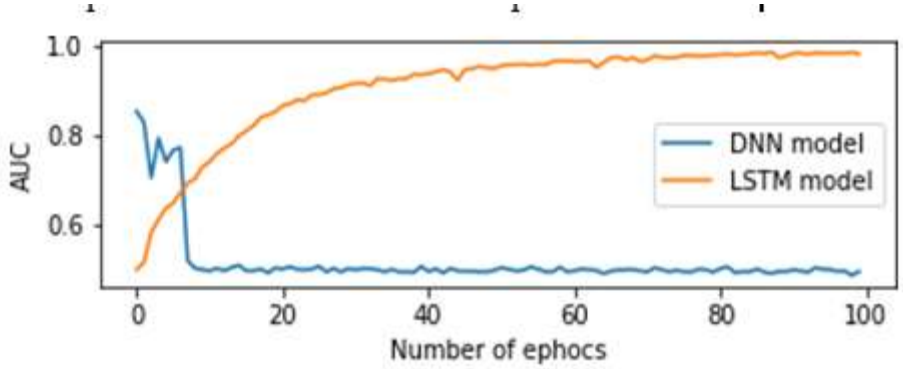

Fig-6: AUC

The Area Under the Curve (AUC) measures a classifier's ability to differentiate between classes, and it is used as a summary of the Receiver Operating Characteristics (ROC) curve. The higher the AUC, the better the user sees the model's ability to differentiate between the positive and negative classifications.

Here fig-6 represents AUC for the classification of malicious samples and benign samples. And the graph compares the existing ANN model and proposed LSTM model. ANN model fails to give better AUC of malicious and benign examples. Because malicious code is a series of actions, performed and ANN fails to memorize code sequences. But the LSTM model contains a memory unit that can give better AUC while the number of epochs increases. At the same time, ANN fails to provide when the number of epochs increases.

Industrial Engineering Journal
ISSN: 0970-2555
Volume : 51, Issue 8, August : 2022

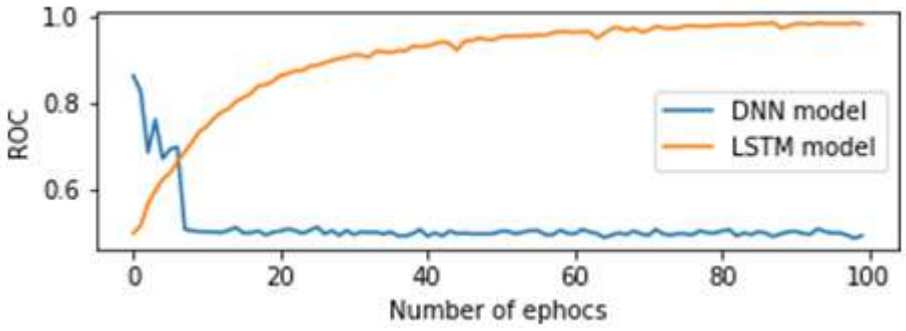

Fig-7: ROC

A ROC curve shows the relationship between TPR and FPR at various categorization levels. Lowering the classification threshold causes more objects to be classified as positive, which increases the number of False Positives and True Positives in the system.

Here fig-7 represents ROC for the classification of malicious samples and benign samples. And the graph compares the existing ANN model and proposed LSTM model. ANN model fails to give better ROC of malicious and benign examples. Because malicious code is a series of actions, performed and ANN fails to memorize code sequences. But the LSTM model contains a memory unit that can give better ROC while the number of epochs increases. At the same time, ANN failsto provide when the quantity of epochs rises.

CONCLUSION

Malware detection methods have been evolving since information systems became an important part of people's life. The sheer growth of information technology requires faster and more efficient malware detection methods. Also, the anti-detection methods such as obfuscation techniques developed by malware authors increase the need for smart and fully automated malware detection methods. In the light of these needs, the advancements in artificial intelligence made AI-based methods the best candidate to develop better malware detection methods. In the first AI-based studies, machine learning (ML) classification algorithms were popularly used to classify the data obtained from malicious and benign software. However, ML classification algorithms do not provide fully automated methods since they require time and effort for feature selection and extraction. So, in the following studies, the focus was shifted from ML-based methods to deep learning (DL) based methods since deep neural networks simulated the learning process better and provided smarter and faster agents. Nowadays, deep neural network architectures, particularly convolutional neural networks (CNNs) and recurrent neural networks (RNNs), are widely employedin academic researches to classify malicious and benign software. In this study, we proposed an approach to classify malicious and benign code pieces. We worked on assembly language. Unlike other studies, we implemented our approach on dynamic analysis data instead of static analysis andfocused on assembly code instead of focusing on just opcodes. With the deep learning architecture LSTM, which is a specialized RNN, we modeled malicious and benign software run traces like natural languages. Our proposed framework for the dynamic analysis of run trace data also makes the approach resistant to polymorphic and metamorphic malware.

REFERENCES

1 Tobiyama, S., Yamaguchi, Y., Shimada, H., Ikuse, T., & Yagi, T. (2016, June). Malware detection with adeep neural network using process behavior. In Computer Software and Applications Conference (COMPSAC), 2016 IEEE 40th Annual (Vol. 2, pp. 577-582). IEEE.

2 Huang, W., & Stokes, J. W. (2016, July). MtNet: a multi-task neural network for dynamic malware classification. In International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (pp. 399-418). Springer, Cham.

3 Pascanu, R., Stokes, J. W., Sanossian, H., Marinescu, M., & Thomas, A. (2015, April). Malware classification with recurrent networks. In Acoustics, Speech and Signal Processing (ICASSP), 2015

IEEE International Conference on (pp. 1916-1920). IEEE.

4 Shibahara, T., Yagi, T., Akiyama, M., Chiba, D., & Yada, T. (2016, December). Efficient dynamic malware analysis based on network behavior using deep learning. In Global Communications Conference (GLOBECOM), 2016 IEEE (pp. 1-7). IEEE.

5 Kolosnjaji, B., Zarras, A., Webster, G., & Eckert, C. (2016, December). Deep learning for classification of malware system call sequences. In Australasian Joint Conference on Artificial Intelligence (pp. 137-149). Springer, Cham.

6 Raff, E., Zak, R., Cox, R., Sylvester, J., Yacci, P., Ward, R., ... & Nicholas, C. (2018). An investigation of byte n-gram features for malware classification. Journal of Computer Virology and Hacking Techniques, 14(1), 1-20.

7 Anderson, H. S., & Roth, P. (2018). EMBER: An Open Dataset for Training Static PE Malware Machine Learning Models. arXiv preprint arXiv:1804.04637.

8 Damodaran, A., Di Troia, F., Visaggio, C. A., Austin, T. H., & Stamp, M. (2017). A comparison of static, dynamic, and hybrid analysis for malware detection. Journal of Computer Virology and Hacking Techniques,13(1), 1-12.

9 Nataraj, L. (2015). A signal processing approach to malware analysis. University of California, Santa Barbara.

10 Nataraj, L., Kirat, D., Manjunath, B. S., & Vigna, G. (2013, December). Sarvam: Search and retrieval of malware. In Proceedings of the Annual Computer Security Conference (ACSAC) Workshop on Next Generation Malware Attacks and Defense (NGMAD).

11 Nataraj, L., Yegneswaran, V., Porras, P., & Zhang, J. (2011, October). A comparative assessment of malware classification using binary texture analysis and dynamic analysis. In Proceedings of the 4th ACM Workshop on Security and Artificial Intelligence (pp. 21-30). ACM. [12] Nataraj, L., Jacob, G., & Manjunath, B. S. (2010). Detecting packed executables based on raw binary data. Technical report.

12 Farrokhmanesh, M., & Hamzeh, A. (2016, April). A novel method for malware detection using audio signal processing techniques. In Artificial Intelligence and Robotics (IRAN OPEN), 2016 (pp. 85-91). IEEE.

13 Nataraj, L., Karthikeyan, S., Jacob, G., & Manjunath, B. S. (2011, July). Malware images: visualization and automatic classification. In Proceedings of the 8th international symposium on visualization for cyber security (p. 4). ACM.

14 Nataraj, L., & Manjunath, B. S. (2016). SPAM: signal processing to analyze malware. arXiv preprint arXiv:1605.05280.

15 Kirat, D., Nataraj, L., Vigna, G., & Manjunath, B. S. (2013, December). Signal: A static signal processing-based malware triage. In Proceedings of the 29th Annual Computer Security Applications Conference (pp. 89-98). ACM.