Journal of Engineering Science and Technology
Special Issue on ICCSIT 2018, July (2018) 14 - 23

MALWARE CLASSIFICATION AND DETECTION USING ARTIFICIAL NEURAL NETWORK

BABAK BASHARI RAD*, MOHAMMAD KAZEM HASSAN NEJAD, MARYAM SHAHPASAND

School of Computing, Asia Pacific University of Technology and Innovation (APU), Technology Park Malaysia, Bukit Jalil, Kuala Lumpur, 57000, Kuala Lumpur, Malaysia
*Corresponding Author: babak@iraseat.com

Abstract

The steady transition towards higher computer dependency and usage has created a dangerous threat landscape that malefactors and cybercriminals are interested in. This has given the rise to an ever-changing series of malware being created aiming to do a series of malicious tasks. The Anti-Virus (AV) industry has implemented traditional methods, such as hash-based, signature-based, and heuristic-based detection techniques to detect malware, each of which has their own set of drawbacks that limit their ability to detect malware with high efficacy. To address these issues, security analysts and researchers have transitioned their focus to other disciplinary fields, most notably, machine learning. Although there have been notable works done in this domain, there yet lies a gap, as no work thus far has been able to achieve the ultimate detection rate with minimal performance overhead, therefore there's a need for exploring new methods or set of approaches for malware detection. This paper focuses on the investigation and implementation of a neural network binary malware classifier that can classify an unseen file as malicious or benign. The scope has been narrowed down to classify Windows Portable Executable (PE) files based on their imported library function calls. The implemented model achieved an average accuracy of 97.8%, with 97.6% precision, and 96.6% recall. These are very promising results, as they signify the model's ability to generalize against an independent set, thus accentuating the viability of the proposed and implemented a method for malware classification.

Keywords: Artificial neural network, Malware, Malware classification, Malware detection, Neural network.

1. Introduction

According to the latest security report by AV-TEST [1], the development of malware is in an ever-rising state, with researchers predicting over 600 million unique malware, majority of them were Windows-based PEs, by the end of 2016. In contrary, there were merely 2.6 million known unique malware in 2006. This unprecedented surge of innovative variants of malware has driven the AV industry and academic researchers to propose and/or deploy various methods of malware detection such as signature, hash, and heuristic-based methods. However, as it is evident through the comparative tests done by independent AV testers [2], solutions which implemented the latest detection techniques provided by the AV industry are yet lackluster, as the known malware detection techniques used by the AV pose their own set of limitations, and therefore restrict each engine from reaching the ultimate detection rate with zero false positive rates, and thus the discovery of the ultimate malware detection methods to combat the innovative malware produced by malware authors remains a challenge.

However, to mitigate the shortcomings presented by current techniques (hash, signature, and heuristic), researchers have transitioned their focus to investigating the use of machine learning for malware detection. Many notable works have already been done for malware classification/detection with impressive detection rates that target the classification of PE. Nevertheless, these works have their own set of limitations that introduces the gap of study. For instance, Kruczkowski, and Szynkiewicz [3] have utilized Support Vector Machine (SVM) by using a two-class pattern classification for heterogeneous data gathered from computer networks, capable of achieving a good generalization with low false-positive rate and high sensitivity, but the heavy computation necessary for the heterogeneity of data was a major concern with their approach.

Therefore, in this research, we focused on the utilization of a simple neural network binary classifier with the imported Windows library function calls to classify each unseen PE file as benign or malicious with the aim of diminishing the unexplored gap left in the field of utilizing machine learning techniques for malware detection, and perhaps achieving a better precision and accuracy compared to the work done by previous researchers.

The structure of the rest of this paper is as follows. Section 2 provides a literature review including static analysis and feature extraction, neural networks, and the recent similar works. Section 3 describes the operational framework that encompasses the proposed approach including the steps taken for sample collection, labeling, feature extraction, besides a description of the proposed neural network classifier model. Section 4 discusses the parameters and validation technique used for this experiment. In section 5, we evaluated and discussed the results of the experiment through comparative analysis. Lastly, a conclusion is presented with discussion on the implications and provide a pathway for the future works is provided in Section 6.

2. Literature Review

2.1. Malware

Computer malware, also known as malicious computer software, refers to any software which intends to steal information from a victim's computer, disable or disrupt the operation of the victim's computer, or provide control access to the

victim's computer. The first variations of malware were computer viruses. However, over the years many other common malware types have surfaced, such as logic bombs, worms, rootkits, ransomware, backdoor, and Trojans.

2.2. Static features

Static analysis refers to the process of examining the code and structure of a program without running it, while dynamic analysis is done by executing the program and examining its behavior [4, 5]. There are many features that are looked at using several static techniques that can be practiced for detecting malware, such as hard-coded printable strings, byte sequences, imported Dynamic Link Libraries (DLLs), library calls, and op-code n-grams, which allow all the possible execution paths to be followed [6]. Several of these features can be extracted from analyzing the PE metadata, such as imported DLLs, PE header information, entropy, file size, hard-coded strings, while others are done by examining the bytecode or the decompiled/disassembled instructions, as the majority of malware can be perceived as binary files [5].

For instance, Gonzalez and Vazquez [7] proposed an approach utilizing the number of Application Programming Interface (API) calls that are made from the DLLs imported by an application since user-level programs require APIs to communicate with the operating system and utilize hardware resources. They disassembled the binary samples obtained using IDA disassembler and used various learning algorithms to train their neural network model, achieving the maximum accuracy rate of 97.95% when utilizing Leven-Marquardt algorithm with one hidden layer for their neural network. Several researchers have also proposed techniques for malware detection revolving around statically analyzing the API calls of a sample. One of the major advantages posed by static analysis is the low resource intensiveness and better performance, especially when coupled with machine learning, where training the model itself can be very resource intensive, performance demanding, and time-consuming (high time complexity).

2.3. Neural network

Artificial Neural Network (ANN) refers to a computational model simulating the structure of neural networks in the brain providing a practical approach to learning continuous, categorical, and vector-valued functions from examples [8, 9]. A perceptron is the basic processing unit. It was the first supervised learning algorithm invented in 1957 [10]. It takes input signals $(x_1, x_2, \ldots, x_j)$ in form of numbers or vectors from its environment or output from other perceptron, with each containing a weight $(w_1, w_2, \ldots, w_j)$ to define a hyperplane to be divided into negative or positive space and thus output a negative or positive instance (0 or 1) based on the weighted sum, $\sum_j w_j * x_j + b$. The weight for each input is based on its importance and the impact of that input on the output, compared to other inputs [9, 11]. This is the simplest form that works on problems which are linearly separable. Therefore, a perceptron can be viewed as an ANN with one neuron.

2.4. Related works

Kolosnjaji et al. [12] proposed and implemented a method based on the utilization of neural networks to classify unseen malware samples into predefined malware

family sets. They explore the usage of a high number of hidden layers to model complex functions to classify malware by utilizing the advancements in GPU-enabled parallel processing. They construct their neural network layers using two different types, recurrent and convolutional. The convolutional networks use set of n-grams of system calls that are not modeled sequentially, but only counting the presence and relation of the n-grams in the function call trace, allowing for simplicity of sequence modeling, but resulting in potential loss of information fidelity. The recurrent networks utilize full sequential modeling depending on specific system call appearances from the sequence of previous system calls. This introduces a higher level of complexity, but they believe that if trained correctly, it could allow for better accuracy.

They have evaluated their network achieving 89.4% accuracy rate. The strength of their approach is the ability to utilize modern hardware, such as GPU, to enable efficient processing on a larger scale, however, their approach does not achieve an impressible accuracy rate, which might be due to the usage of dynamic features through a sandbox environment, which can be detected and evaded by malware resulting in irrelevant system call sequences. Saxe and Berlin have proposed a malware classification system constructed using the neural network with two hidden layers [13]. They are using static features, including PE import, entropy, metadata, and string 2d histogram. The two hidden layers consisted of 1024 Parametric Rectified Linear Units (PReLu), while their output layer consisted of a sigmoid neuron to classify the instance as malware or benign. They have achieved a 95% detection rate with 0.1% false positive rate when running an experiment on a 400,000 samples dataset. Furthermore, it should be noted that they achieved their highest true positive rate when using all the static features as opposed to single features. Their results demonstrate that it is possible to construct a highly accurate with low resource machine learning classification quickly. However, their findings lack the comparison with the shallow neural network or deep neural networks with a greater number of hidden layers that could improve their accuracy rate.

3. Proposed Solution

3.1. Sample collection

There are two general methods for obtaining malicious samples. Malware honeypots can be set up which will capture malicious samples over time through a machine that is configured to be vulnerable, however, it is often time-consuming. Another method is to collect samples that are stored in repositories or malware 'zoos' that are shared by researchers or organizations. The samples collected for this project are attained from a malicious repository shared by a fellow researcher [14]. It consists of varying malware types, such as Trojans, viruses, and rootkits that provides a more diverse dataset that allows for better model generalization. The clean PE samples were collected by traversing through a clean Windows system directory. There are no set rules for determining the sample size required for training and testing a neural network. However, based on the works done by previous researchers and the complexity of their neural networks, a dataset size of 4,000 (1,000 benign and 3,000 malicious samples) was used. However, initially, as many samples were collected as possible, as the numbers will skim down through the process of labeling and cleaning.

3.2. Labeling

The obtained samples are initially placed into their respective folders, for instance, all the obtained malicious samples will be placed into a folder called "*malicious_samples*", while all the benign samples will be placed into a folder called "*benign_samples*". This will eliminate the need for manually labeling each file. However, to ensure that each obtained sample is usable, and more importantly, is correctly labeled as malicious or benign, each sample has to meet the following criteria:

- It should be a valid PE file. This can be done by checking the first two bytes of a binary file to find the "MZ" magic number.
- The known packed malware are excluded from the dataset that can adversely affect the quality of the classifier model. This is done by performing a signature check to identify whether the malware is packed against known packers, such as UPX packer.
- The samples are checked against well-known AVes through the VirusTotal API, and for the malicious samples, samples that score below 75% detection rate will be removed, while for the benign samples, only samples that are not detected by any AV at all will remain (0% detection rate). These steps should be sufficient to omit any sample that might affect the generalization of the model.

3.3. Feature extraction

The imported Windows library function calls for each sample need to be extracted. These can be represented within a binary feature vector matrix. However, all the function calls need to be extracted for each respective sample, and a unique list of function calls needs to be generated from all the samples, after which, each sample's function calls are compared to the list, and if a function call exists within the sample file, the respective index is marked as 1 within the binary vector space. Therefore, each sample will have a binary vector space, with the size equal to the total number of unique function calls across all the sample files.

3.4. Neural network classification

The proposed malware classifier for this research is a simple neural network binary classifier. Each class label is represented using a one-hot vector indicating whether a sample is malicious $(0,1)$ or benign $(1,0)$. The input layer will consist of n inputs, where n is equal to the total number of unique function calls. Therefore, each input node is basically a feature from the list of function calls (extracted from the binary vector space). Backpropagation is done every time a forward pass is done, and an optimizer function, in this case, the Gradient Descent algorithm with decayed learning rate, is used to update the weights of the graph that reduce the cost associated to the model before the next epoch. The output neuron will use a softmax function as it is the most common activation function used in the output layer alongside a cross-entropy cost-function [15]. Figure 1 illustrates the overall architecture of neural network proposed in this research.

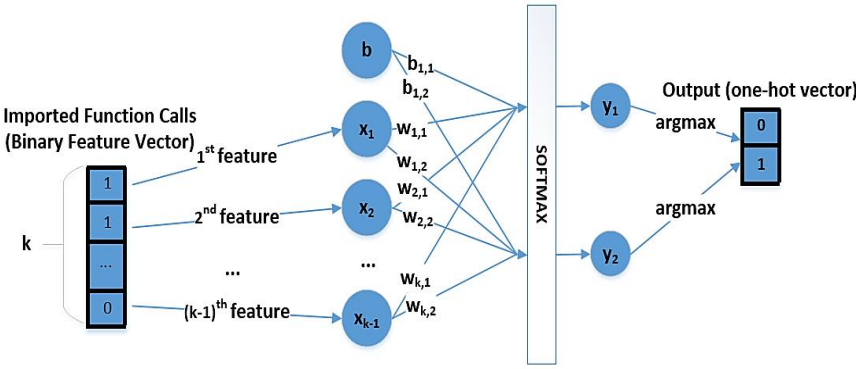


Fig. 1. Proposed neural network classifier with softmax output function and a bias unit.

4. Experiments

4.1. Trials

The proposed neural network classifier was developed using TensorFlow-GPU, a Python library that utilizes the GPU for parallel processing. It was run on an environment equipped with a CUDA Enabled NVIDIA GeForce GTX 970 4GB GDDR5 with 5.2 Compute Capability. Additionally, scikit-learn was used for carrying out the cross-validation experiment. There are several varying parameters that were defined to run the trials. The dataset consisted of 4,000 samples, of which, 3,000 were malicious, and 1,000 were benign. After pre-processing the dataset, the input (unique Windows library function calls) size was 34,087. These were split into batches of 128 samples, which were then used for training, validation, and testing. The learning rate was set at 0.01 with a decay rate of 0.975 for training, which consisted of 10 epoch runs.

4.2. Cross-validation experiment

To evaluate a predictive model, in terms of how well it can perform against an unknown dataset in practice, it is important to validate the model through means of cross-validation techniques, such as k-fold. This allows for the dataset to be folded k times, thus giving k unique sets of training and testing that the model can be trained and tested against. In this project, 10-folds cross-validation was used, and at the end of the process, the results were macro-averaged. The metrics used for evaluation are the accuracy, recall, and precision, each is captured and averaged to provide an average result of the trained model. By performing an 80/20 split, the dataset was split into 29 training batches and 2 testing batches in each iteration.

5. Results and Analysis

After cross-validation, the model can now be evaluated, as it is safe to assume that, in practice, it will perform based on the average results that were attained. Based on the defined formulas, the metrics were recorded with each iteration. Table 1 shows the comparative results between each fold and the macro-averaged result.

Table 1. Evaluation of the model performance based on 10-fold cross-validation.

Iteration No.	ACC*	PR*	RC*	TPR*	TNR*	FPR*	FNR*
1	98.75	99.3	97.22	97.22	99.6	0.39	2.78
2	98.25	97.47	98.09	98.09	98.35	1.65	1.91
3	97	97.2	94.55	94.56	98.42	1.58	5.44
4	97.5	98.71	95.03	95.03	99.16	0.84	4.97
5	97.25	95.71	96.4	96.4	97.7	2.3	3.6
6	97.75	98.09	96.25	96.25	98.75	1.25	3.75
7	98.25	97.33	97.98	97.99	98.41	1.59	2.01
8	98.5	98.65	97.33	97.33	99.2	0.8	2.67
9	98.25	96.53	98.58	98.58	98.07	1.93	1.42
10	96.5	96.62	94.08	94.08	97.98	2.02	5.92
Min	96.5	95.71	94.08	94.08	97.98	0.39	1.91
Max	98.75	99.3	98.58	98.58	99.6	2.02	5.92
Macro-Average	**97.8**	**97.6**	**96.6**	**-**	**-**	**-**	**-**

*ACC: Accuracy, PR: Precision, RC: Recall, TPR: True-Positive Rate, TNR: True-Negative Rate, FPR: False-Positive Rate, FNR: False-Negative Rate

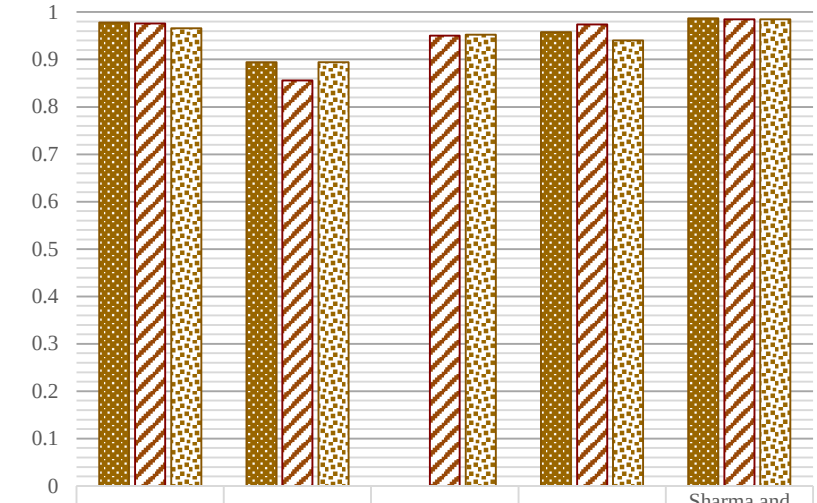
Based on the table above, it can be observed that throughout all the iterations, the false-positive rate has almost remained under 2% consistently, which is very important, as this indicates that clean files are rarely misclassified as malicious. This is important within the AV industry, as having a low false-positive rate can mean that a detection engine is more effective, as it does not classify clean files (such as Windows system files) as malicious. Moreover, the best precision was 99.3%, indicating that out of 1000 files that are classified as malicious, 993 of those are correctly classified. The best recall was 98.58%, indicating that out of every 10000 files, 9858 files are correctly classified as malicious. These results are very promising, as they indicate that the model has achieved what it had intended to do, with a very high accuracy, precision, and recall, and extremely low false-positive and false-negative rates. These signify the model's ability to generalize against an independent set, thus accentuating the proposed and implemented a method for malware classification.

The charts in Fig. 2 illustrates the visual and tabular comparison between the results of several works that were reviewed within the literature review against the macro-averaged results from the validated model of this experiment.

Based on the results above, it can be seen that the model outperforms previous works that were utilizing neural networks, such as the work done by Kolosnjaji et al. [12], which goes to show that the usage of deep-learning with dynamic features does not necessarily equate into a higher model quality, as not only does this introduce unnecessary complexity to the model that increases the performance overhead, but also through the use of dynamic features, the model is more susceptible to degrading in terms of generalization. Saxe and Berlin have also utilized deep-learning with static features, however, their model does not perform as well as validated model based on the metrics, perhaps this is due to their smaller dataset with a less diverse set of features, which is often needed when utilizing deep-learning [13].

Other than neural networks, other works have utilized different machine learning algorithms and techniques with which the macro-averaged model metrics

can be compared. For instance, Santos et al. proposed a framework which consists of a classifier based on the frequency of opcode sequences. Although the architecture they have proposed seems to be scalable, their results still do not perform this project's validated model as well, perhaps the utilization of neural networks with opcode sequences could further improve their results [16]. However, the Random-Forest classifier proposed by Sharma and Sahay slightly outperforms the proposed model. However, this was done based on their best results, and they did not report any forms of validation or averaging [17].



	The Current Research	Kolosnjaji et al. (2016)	Saxe and Berlin (2015)	Santos et al. (2013)	Sharma and Sahay (2016)
ACC	0.978	0.894		0.958	0.987
PR	0.976	0.856	0.95	0.974	0.985
RC	0.966	0.894	0.952	0.94	0.985

Fig. 2. Comparison of results with the previous related works.

6. Conclusion

In this paper, a neural network model is presented, which can classify an unseen PE (PE) file as benign or malicious based on its loaded library function calls. The implemented model achieved an average accuracy of 97.8%, with 97.6% precision, and 96.6% recall over a dataset size of 4,000 (3,000 malicious and 1,000 benign) PE files. These are very promising results, as they accentuate the viability of the proposed and implemented an approach for malware classification considerations for industrial or academic use.

This paper focused on samples that were not packed with known packers, as these would adversely affect the quality of the model due to the usage of static features. Therefore, the usage of results from performing hybrid analysis may result in the extraction of more meaningful features that could be used for classification. Furthermore, this paper investigates the usage of a simple neural network binary classifier for the classification of unseen files. However, with future advancements

in hardware technology and GPU-enabled parallel-processing, the investigation of deep/wide learning can be taken into consideration to develop more complex models for malware classification with a minimal performance overhead.

Abbreviations

ACC	Accuracy
ANN	Artificial Neural Network
API	Application Programming Interface
AV	Anti-Virus
DLL	Dynamic Link Libraries
FNR	False-Negative Rate
FPR	False-Positive Rate
GPU	Graphics Processing Unit
PE	Portable Executable
PR	Precision
PReLu	Parametric Rectified Linear Units
RC	Recall
SVM	Support Vector Machine
TNR	True-Negative Rate
TPR	True-Positive Rate

References

1. AV-TEST (2016). *SECURITY REPORT* 2015/16.
2. Comparatives, AV (2016). *File detection test of malicious software including false alarm test*.
3. Kruczkowski, M.; and Szynkiewicz, E.N. (2014). Support vector machine for malware analysis and classification. *IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT)*. 2, 415-420.
4. Sikorski, M.; and Honig, A. (2012). *Practical malware analysis*. san francisco, UNITED STATES: No Starch Press.
5. Bashari Rad, B. (2016). *Metamorphic computer virus detection using hidden markov model*. LAP LAMBERT Academic Publishing.
6. Cepeda, C.; Tien, D.L.C.; and Ordóñez, P. (2016). Feature selection and improving classification performance for malware detection. *IEEE International Conferences on Big Data and Cloud Computing (BDCloud), Social Computing and Networking (SocialCom), Sustainable Computing and Communications (SustainCom) (BDCloud-SocialCom-SustainCom)*, 560-566.
7. Gonzalez, L.E.; and Vazquez, R.A. (2013). Malware classification using Euclidean distance and artificial neural networks. *12th Mexican International Conference on Artificial Intelligence (MICAI)*, 103-108.
8. Bishop, C.M. (2006). *Pattern recognition and machine learning (information science and statistics)*. Springer-Verlag New York, Inc.
9. Gurney, K. (2014). *An introduction to neural networks*. CRC press.

10. Llauradó, D.G. (2016). *Convolutional neural networks for malware classification.* Master's Thesis. Department of Computer Science, Universitat Politècnica de Catalunya.
11. Alpaydin, E. (2014). *Introduction to machine learning*. Cambridge, UNITED STATES: MIT Press.
12. Kolosnjaji, B.; Zarras, A.; Webster, G.; and Eckert, C. (2016). Deep learning for classification of malware system call sequences. *AI 2016: Advances in Artificial Intelligence: 29th Australasian Joint Conference*. Hobart, TAS, Australia, December 5-8, 2016, Proceedings, Kang, B.; and Bai, Q., Eds. Cham: Springer International Publishing, 137-149.
13. Saxe J.; and Berlin, K. (2015). Deep neural network-based malware detection using two-dimensional binary program features. *Proceedings of the 2015 10th International Conference on Malicious and Unwanted Software (MALWARE)*.
14. Hussain, N. (2016). Static-Malware-Analysis. Retrieved October 5, 2017, from https://github.com/naisofly/Static-Malware-Analysis.
15. Olah, C.; and Shan, C. (2016). Attention and augmented recurrent neural networks. *Distill*.
16. Santos, I.; Brezo, F.; Ugarte-Pedrero, X.; and Bringas, P.G. (2013). Opcode sequences as a representation of executables for data-mining-based unknown malware detection. *Information Sciences*, 231, 64-82.
17. Sharma A.; and Sahay, S.K. (2016). An effective approach for classification of advanced malware with high accuracy. arXiv preprint arXiv:1606.06897.