Malware Detection with Neural Network Using Combined Features

Huan Zhou(✉)

Onescorpion, 125 Malianwa North Road, Haidian District, 100000 Beijing, China
zhouhuan@onescorpion.com

Abstract. The growth in amount and species of malicious programs are now turning into a severe problem that strengthens the demand for development in detecting and classifying the potential threats automatically. Deep learning is an acceptable method to process this increment. In this paper, we propose an innovative method for detecting malware which uses the combined features (static + dynamic) to classify whether a portable executable (PE) file is malicious or not. A thorough experimental research on a real PE file collection was executed to make comparisons with the results that was performed in diverse situations and the performances of different machine learning models. The experiments prove the effectiveness of our model and show that our method is able to detect unknown malicious samples well.

Keywords: Malware detection · Neural networks · Feature engineering

1 Introduction

The amount of malware is growing annually and various types of attacks are more progressive and complex than before. One issue in computer security is thence to discover malware, so that it can be blocked before reaching its targets, or at least so that it can be wiped out in case it has been detected.

However, hackers keep on accelerating the automation of malware construction applying approaches such as polymorphism at a shocking rate. Obviously, automatic detection using highly precise intelligent models may be the only selection to fight against the issue in the future.

In recent years, a convergence of three evolutions have raised the probability for success in approaches using machine learning, keeping the commitment that these methods may reach pretty good detection performance at very low error rates without the trouble of human signature production required by non-automatic approaches.

The growth of commercial threat information feeds is the first of these tendencies which supplies great volumes of new malware, representing that for the first time, promptly, labeled malware samples are accessible to the security community. The second tendency is that computing power is much stronger and cheaper nowadays, implying that researchers are able to go over malware detection machine learning models more swiftly and train much more sophisticated and deeper models. Ultimately, machine learning as a subject has developed, suggesting that investigators have more instruments to build models which can reach great performance not only in accuracy but also in scalability.

We propose an innovative method for detecting malware which uses combined features (static + dynamic) to classify whether a portable executable file is malicious or benign in this paper. Our method employs 2 kinds of neural networks to fit distinct property of respective work pipelines. The first type of neural network we use is recurrent neural network that is trained for extracting behavioral features of PE file, and the second type is convolutional neural network that is applied to classify samples. At the training stage of our method, we firstly extract static information of a PE file and use sandbox to record system API call sequences as dynamic behaviors. Then we extract static features based on predefined rules and dynamic features out of the trained RNN model. Next we combine them and use well design algorithm to create images. Lastly, we train and validate the concurrent classifier using images created in the previous steps labeled with 1(malicious) or 0(benign).

2 Related Work

In this section, we present published researches of deep neural network and malware detection.

2.1 Deep Neural Network

Neural networks (NN) have been studied for over thirty years which imitates the architecture referring to neuron collections in brain. NN consists of multiple layers. Deep neural networks (DNN) is a type of NN that comprises a lot of hidden layers.

Deep learning has become prevalent in many areas such as speech recognition [1] and computer vision [2] in recent times. Hinton et al. put forward an astonishing method called Dropout that can solve gradient vanishing problem well [3]. This approach decreases dependencies among neurons through omitting several results of neurons to prevent overfitting. The omitted neurons are selected stochastically. Therefore, all training is executed with distinct architectural network which decreases the dependency between neurons. Krizhevsky et al. use CNN which astonishingly reduced the false positive rate in the field of computer vision. Gers et al. put forward LSTM [4] which avoids the error disappearing issue.

2.2 Malware Detection

There are two types of malware detection approaches. The first type is detecting malicious files before they run to avoid endpoints being infected, and the second type is detecting endpoints which have been attacked to reduce the outspread of loss to the smallest possible degree.

Malware classification has been a popular research fields since 1990s. Mathur and Idika [5] proposed a good overview in this area. Kephart et al. [6] put forward an innovative method which utilizes neural networks for detecting malicious behaviors. Dahl et al. [7] made attempt to do malware classification utilizing neural networks and random projections on a large scale. Saxe et al. [8] try to do static analysis on samples by using feed-forward neural networks. Huang et al. [9] concentrated on assessing multi-task learning ideas and made use of deep feed-forward neural network. Pascanu et al. [10] built models based on system call sequences and utilized recurrent neural networks so as to build a "language model" for target files. They took measures to check performance on gated recurrent units (GRU) and long short-Term memory (LSTM) and reported good results.

3 Proposed Method

In this section, we propose an innovative method for detecting malware which uses combined features to classify whether a PE file is malicious or benign. We split the approaches into 4 stages. The first one extracts static feature information from PE file. The second stage records the system API sequences using sandbox and processes them by RNN. At the third stage, we combine the former static and dynamic features and convert them into fixed feature vectors which are going to be transformed into images. Finally, we train and classify the images using designed model based on CNN.

3.1 Overview

The overview of our proposal is shown in Fig. 1. For each file, many types of raw information are collected such as header, byte histogram, import list, etc., and a suit of application programming interface (API) call events.

Static information does not need thorough or sophisticated configuration for collection and multiplex static features have been raised for feature engineering of PE file: printable strings [11], opcodes, import tables, informational entropy [12] and byte n-grams [13]. We extract some basic features using approaches which have been used in previously published works.

File behaviors are consisted of a variety of activities such as registry operation, file management and so on which involve various operations. When we use API call sequences to represent dynamic information, a variety of API calls stand for an activity, and all of the recorded API calls will be regarded as dynamic features of target file. This hierarchical structure is the very picture of the composition of writings. A single writing is made up of multiple sentences which consist of various words. Therefore, we suppose that we are able to utilize language model like RNN to get the dynamic features of file.

The feature vectors extracted from static and dynamic information will be concatenated and converted into an image. And the generated image will contain combined information which will be use later. Our classifier is based on CNN since it has been proved to be very effective in image classification.

The training flow can be divided into four phases as shown in Fig. 1. First, collecting basic static and dynamic information of PE files. Second, the static features are extracted using predefined extractor and the RNN is trained using file API sequences to extract dynamic features. Third, features are combined and converted into feature images. At last, the neural network classifier will be trained and validated using labeled generated images.

After training the designed classifiers, we verify the effectiveness of our model. At the beginning, generating the images of PE files in validate dataset using the former steps. Finally, these files will be labeled whether 1 or 0 using model depended on the outputs and predefined threshold.

The specific details of every step are introduced in the following sections.

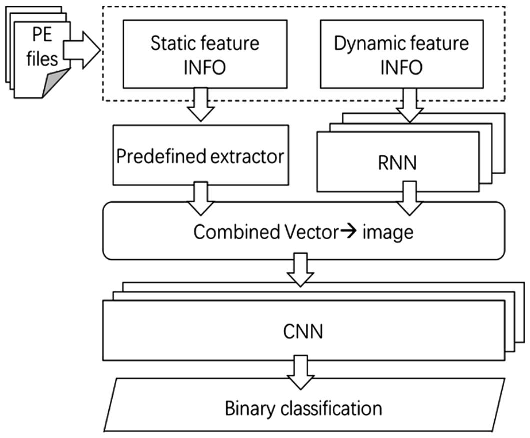

Fig. 1. Overview

3.2 Static Features

We predefined some basic features which will be extracted from PE file using methods employed in previously published works. The following Table 1 gives a summary of all target static features.

3.3 Dynamic Features

API Call Sequences. Before feeding the system API call sequences to designed models, we have to preprocess the input data. Dealing with redundant data and turning inputs into numerical vectors are two of the primary preparations. Firstly, we clean API sequences in which a single API is duplicate over 2 times. We merge these same API call sequences through applying maximum 2 successive duplicate system API call instances in the results. Moreover, we utilize 1-hot encoding method to create a specific binary vector for each system API call in our dataset. Along these lines, we get a set of numerical feature vectors rather than a suit of system API call names.

Table 1. Summary of target static features

Features	Description
Byte histogram	byte histogram (count + non-normalized) over the entire binary file
ByteEntropy histogram	2d byte/entropy histogram
Section information	section names, sizes and entropy: section_sizes_hashed, section_entropy_hashed, etc.
Imports information	imported libraries and functions from the import address table
Exports information	exported functions
General File information	general information about the file: size, vsize, has_debug, etc.
Header File information	machine, architecture, OS, linker and other information extracted from header: timestamp, machine, etc.
String extractor	extracts strings from raw byte stream: ‘numstrings’, ‘avglength’, etc.

Training LSTM. We employ LSTM which is a type of recurrent neural network to build our behavior model. Our model is consisted of an input layer X, multiple hidden layer (1 oridinary + 2 LSTM), and an output layer Y. The structure of our behavior model is illustrated in Fig. 2.

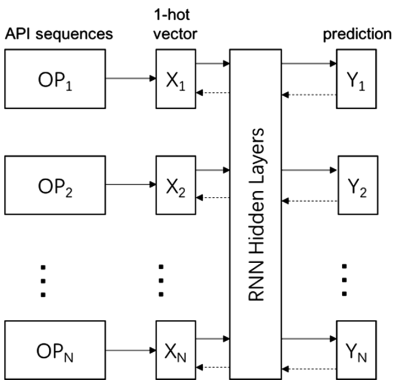

Fig. 2. RNN training process

Feature Extraction. We extract dynamic features of PE file by using trained model based on RNN. Our trained dynamic feature extractor is able to output the next predicted action from former sequences of inputs. Furthermore, fractional features are distilled in layers that near to the head of deep neural networks. And abstracted features are distilled in layers near to the bottom. Therefore, we are supposed to get behavioral features in deep layer of trained model.

3.4 Feature Selection and Imaging

Once getting static and dynamic features, we transform and concatenate them to build a combined vector. We design image classifier to receive fixed size of vectors. Thus, we have to transform the chains of vector to configured length since the sequences of system API are totally different between PE files.

$$V = \begin{pmatrix} v_1 \\ v_2 \\ \cdots \\ v_n \end{pmatrix} = \begin{bmatrix} v_{11} & v_{12} & \cdots & v_{1m} \\ v_{21} & v_{22} & \cdots & v_{2m} \\ & & \cdots & \\ v_{n1} & v_{n2} & \cdots & v_{nm} \end{bmatrix} \tag{1}$$

We transform value of feature matrix to the range of [0,1] through using sigmoid function. Then we multiply each element in the matrix with 255 to constitute image of the source file. At last, the matrix V is calculated as feature image with size of n $\times$ m.

3.5 Deep Neural Networks

Next, we train a deep feed forward concurrent neural network (CNN) for binary classification. The network architecture is shown in Fig. 3.

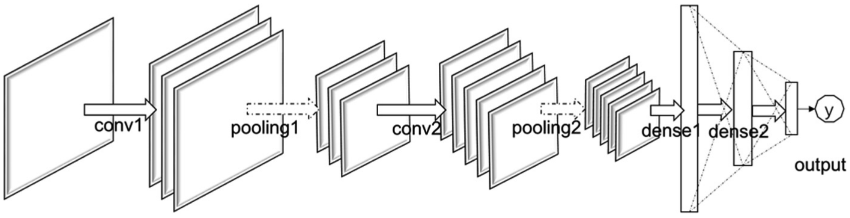

Fig. 3. Structure of CNN classifier

The CNN is consisted of an input layer, 2 convolution layers, 2 pooling layers, 2 fully connected layers, and an output layer. Each pooling layer obtains the result of the former layer and cut the output size into one half using average-pooling with step of two. The dimension of the output layer is 2 since we try to proceed binary classification.

By applying the classifier which has been trained, we compute the value of target file in the validation phase. When feed our trained classifier with a feature image of the validation file, it will output a 2 dimensional vector. And elements in the vector stand for benign and malicious extent. If the malicious value is bigger than the benign value, we can classify the source file as malicious. The probability value is computed through employing function below.

$$p = \mathrm{sigmoid}(y) = 1/1 + \exp(-y) \tag{2}$$

ReLU. The tanh and sigmoid activation function generally appear the gradient vanishing problem making models based on deep neural network difficult to train [14]. To overcome this trouble, we employ rectified linear unit (ReLU) and its activation function is as follows:

$$\mathrm{f}(\mathrm{x}) = \max(0, \mathrm{x}) \tag{3}$$

Dropout. Dropout is a regularization method which is designed for the training phase of DNN. The key operation is that the algorithm makes a choice to update part of the hidden units randomly when updating hidden layer. The intuition for this method is that while ignoring units in hidden layers randomly, the network will be coerced to get multiple different patterns with the same dataset. In our designed classifier, we utilize Dropout to solve the gradient vanishing problem.

Loss function. Deep neural networks learn various patterns of inputs in different layers. The bottom layer uses function called softmax to calculate two dimensional vector which stands for benign and malware. To fine tune our model, we employ the loss function called cross entropy to assess the quality of our model's results. The function is illustrated as

$$L_n(\theta(v)) = -\sum\nolimits_{n \in N} gtd_n(v) log \theta_n(\mathrm{v}) \tag{4}$$

where v stands for the input vector, n means category, N is the set of predicted categories, gtd stands for ground truth distribution, and θ(x) indicates probability distribution of classifier.

4 Experiment

4.1 Dataset

We use a published framework [15] to collect samples. Our dataset is consisted of files collected from 3 major sources: Virus Share [16], Maltrieve [17] and private collections. These origins offer a wide and multiplex amount of files for validation. Our final dataset contains 90,000 samples with 72,317 labeled as malicious and 17683 labeled as benign. We train our model on 60,000 of the collected samples. The test data contain 30,000 samples.

There are a lot of tools which are able to track the execution of files and record system API call sequences [18, 19]. We use an open source sandbox called Cuckoo which is very useful and the environment it provides is controllable. For each sample, we receive a set of system API calls and use them to train a RNN model which is able to extract dynamic features.

4.2 Evaluation Method

In the following part, we introduce the method applied to evaluate our experiment results.

In the evaluation phase, we utilize a type of 3-fold cross-validation. So we choose 2/3 of files as training data in each experiment, while the rest of data is allocated into the test set. As a matter of fact, in order to get a trustworthy capability estimation, we averaged the results of 10 cross-validation experiments, carried out with a different stochastic dataset arrangement each time.

Table 2. Confusion matrix

		Predicted value		
		$y = t$	$y \neq t$	
Original value	$x \in t$	True Positive (TP)	False Negative (FN)	P = TP + FN
	$x \notin t$	False Positive (FP)	True Negative (TN)	N = FP + TN

For multi-classification issue, Positive indicates a sample x can be classified as target class t because of surpassing a predefined threshold. On the contrary, it is Negative. Since we try to divide the sample into two categories, the issue turns into binary classification. y is the output of x. Under this circumstance, the confusion matrix is illustrated in Table 2 and we demonstrate the functions which we will use as follows.

$$\mathrm{TPR} = \mathrm{TP}/\mathrm{P} \tag{5}$$

$$\mathrm{FPR} = \mathrm{FP}/\mathrm{N} \tag{6}$$

$$\mathrm{AR} = (\mathrm{TP} + \mathrm{TN})/(\mathrm{P} + \mathrm{N}) \tag{7}$$

where TPR stands for true positive rate, FPR stands for false positive rate and AR means accuracy rate.

We assess the quality of our classifier using Area Under the Curve (AUC) that can be calculated from ROC curve which is a figure showing the relationship between FPR and TPR within threshold. In our method, samples are classified as benign or malicious based on output probability p calculated by (2). The value p and threshold both lie in the range of [0,1]. For each situation, we draw ROC curve through treating TPR as y axis and FPR as x axis. Furthermore, we evaluate classifier efficiency by comparing the AUC in each situation.

4.3 Result

In our first experiment, we want to know the performance of only using static features and the performance of only using dynamic features since we combined the static and dynamic features. Figure 4 show the ROC curve which illustrates that using combined features outperforms the other two methods.

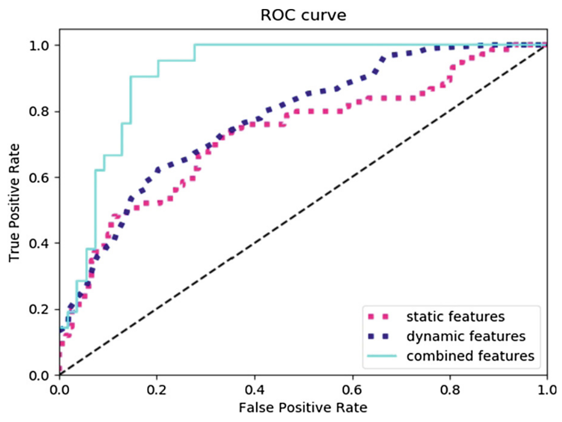

Fig. 4. ROC curves of different feature engineering methods

Then we want to evaluate the effectiveness of our methodology compared with the other machine learning methods. Thus, we compare the classification result of our designed neural network architecture with traditional machine learning methods such as Decision Tree, Random Forest, etc., as well as stat-of-the-art deep learning methods such as CNN, RNN because these models have been widely employed in researches which have been published. We show the results in Table 3 which proves the effectiveness of employing our designed method.

Table 3. Accuracy of different models

Methods	Models	ACC
Traditional machine learning methods	Decision Tree [20]	90.7%
	Random Forest [21]	95.4%
	Hidden Markov Models [22]	87.3%
	Support Vector Machine [23]	91.4%
	KNN [24]	94.7%
NN methods	CNN [25]	94.7%
	RNN [26]	95.6%
Our method	RNN + CNN	97.3%

5 Conclusion

We propose an innovative method for detecting malware which uses the combined features (static + dynamic) to classify whether a portable executable (PE) file is malicious or benign in this paper. Our method discovers malicious software through classifying the generated images using designed model. We make comparisons with the results that was performed in diverse situations and the performances of different machine learning models. The results show that our innovative method acquires the best results in all three situations and outperforms the other models which prove great effectiveness of the proposal. Our method is able to detect unknown malicious samples well.

References

1. Hinton, G., et al.: Deep neural networks for acoustic modeling in speech recognition. In: IEEE Signal Processing Magazine, vol. 29, pp. 82–97 (2012)
2. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Advances in Neural Information Processing Systems, pp. 1097–1105 (2012)
3. Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.: Dropout: a simple way to prevent neural networks from overfitting. J. Mach. Learn. Res. **15**(1), 1929–1958 (2014)
4. Gers, F.A., Schmidhuber, J., Cummins, F.: Learning to forget: continual prediction with LSTM. Neural Comput. **12**(10), 2451–2471 (2000)
5. Idika, N., Mathur, A.P.: A survey of malware detection techniques. Technical report, Purdue University, February 2007. http://www.eecs.umich.edu/techreports/cse/2007/CSE-TR-530-07.pdf
6. Kephart, J.O.: A biologically inspired immune system for computers. In: Proceedings of the Fourth International Workshop on the Synthesis and Simulation of Living Systems, pp. 130–139. MIT Press (1994)
7. Dahl, G.E., Stokes, J.W., Deng, L., Yu, D.: Large-scale malware classification using random projections and neural networks. In: IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP) (2013)
8. Saxe, J., Berlin, K.: Deep Neural Network Based Malware Detection Using Two Dimensional Binary Program Features. arXiv preprint arXiv:1508.03096 (2015)
9. Huang, W., Stokes, J.W.: MtNet: a multi-task neural network for dynamic malware classification. In: Conference on Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA) (2016)
10. Pascanu, R., Stokes, J.W., Sanossian, H., Marinescu, M., Thomas, A.: Malware classification with recurrent networks. In: IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP) (2015)
11. Schultz, M., Eskin, E., Zadok, E., Stolfo, S.: Data mining methods for detection of new malicious executables. In: Proceedings of the 2001 IEEE Symposium on Security and Privacy, 2001. S P 2001, pp. 38–49 (2001)
12. Weber, M., Schmid, M., Schatz, M., Geyer, D.: A toolkit for detecting and analyzing malicious software. In: Proceedings of the 18th Annual Computer Security Applications Conference, pp. 423– 431. IEEE (2002)
13. Abou-Assaleh, T., Cercone, N., Kesˇelj, V., Sweidan, R.: N-gram-based detection of new malicious code. In: Proceedings of the 28th Annual International Computer Software and Applications Conference. COMPSAC 2004, vol. 2, pp. 41–42. IEEE (2004)
14. Hochreiter, S., Bengio, Y., Frasconi, P., Schmidhuber, J.: Gradientflowinrecurrent nets: the difficulty of learning long-term dependencies. In: Kolen, J.F., Kremer, S.C. (eds.) A Field Guide to Dynamical Recurrent Neural Networks. IEEE Press. Wiley- IEEE Press (2001)
15. Webster, G., Hanif, Z., Ludwig, A., Lengyel, T., Zarras, A., Eckert, C.: SKALD: a scalable architecture for feature extraction, multi-user analysis, and real-time information sharing. In: International Conference on Information Security (2016)

16. Roberts, J.-M.: Virus Share, July 2018. https://virusshare.com/
17. Maxwell, K.: Maltrieve, April 2015. https://github.com/krmaxwell/maltrieve
18. Guarnieri, C., Tanasi, A., Bremer, J., Schloesser, M.: The Cuckoo Sandbox (2012)
19. Lengyel, T.K., Maresca, S., Payne, B.D., Webster, G.D., Vogl, S., Kiayias, A.: Scalability, fidelity and stealth in the drakvuf dynamic malware analysis system. In: Annual Computer Security Applications Conference (ACSAC) (2014)
20. HUANG Quanwei: Malicious Executables detection based on N-Gram System call Sequences. Harbin Institute of Technology (2009)
21. Zhang, J., Li, Y.: Malware detection system implementation of Android application based on machine learning. Appl. Res. Comput. **6**, 1–6 (2017)
22. Annachhatre, C., Austin, T.H., Stamp, M.: Hidden Markov models for malware classification. J. Comput. Virol. Hack. Tech. **11**(2), 59–73 (2014)
23. Arp, D., Spreitzenbarth, M., Hubner, M., Gascon, H., Rieck, K.: Drebin: effective and explainable detection of android malware in your pocket. In: Proceedings of NDSS (2014)
24. Dahl, G.E., Stokes, J.W., Deng, L., et al.: Large-scale malware classification using random projections and neural networks. In: 2013 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). IEEE 2013
25. Kolosnjaji, B., et al.: Empowering convolutional networks for malware classification and analysis. In: 2017 International Joint Conference on Neural Networks (IJCNN). IEEE (2017)
26. Athiwaratkun, B., Stokes, J.W.: Malware classification with LSTM and GRU language models and a character-level CNN. In: 2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pages 2482–2486. IEEE (2017)