An End-to-End Model for Android Malware Detection

Hongliang Liang
School of Computer Science
Beijing University of Posts and Telecommunications
Beijing, China
hliang@bupt.edu.cn

Yan Song
School of Computer Science
Beijing University of Posts and Telecommunications
Beijing, China
yansong@bupt.edu.cn

Da Xiao
School of Computer Science
Beijing University of Posts and Telecommunications
Beijing, China
xiaoda99@bupt.edu.cn

***Abstract*—Malware detection has been a difficult problem for a very long time. Since the wide use of smart devices in recent years, the number of malwares is increasing rapidly. Most existing methods for malware detection rely too much on manual interventions (e.g. pre-defined features and patterns), which can be easily deceived. In this paper, we propose a novel end-to-end deep learning model to detect Android malwares. Our model takes the raw system call sequence, which is generated during the application's runtime, as input and decides whether the sequence is malicious without any manual intervention. We evaluate the model on 14231 Android applications and obtain a detection accuracy of 93.16%, which is 2.81% higher than the contrast experiment in which we implement the method proposed by other researchers.**

***Keywords*—Malware detection; system call sequence; convolutional neural networks; deep learning.**

I. INTRODUCTION

Malware detection has been a serious problem for a very long time. Since the wide use of smartphones, the number of malwares has increased at a higher rate than ever before. Users' privacy information (e.g., payment accounts, passwords) and important data are facing more threats and dangers.

Researchers and industries have focused on detecting malware through static analysis, dynamic analysis or both. Also, it is a common way to detect malware by analyzing the system call sequence, which is generated during application runtime [2, 3]. Nevertheless, all of them depend on predefined patterns of malicious behaviors and there is no research trying to extract deep-seated information directly from the original long sequence without any pre-process (e.g., generate system call graph [3], cut down sequence length [2]).

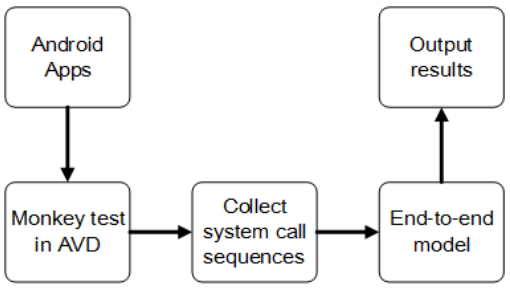

Figure. 1. Intuition of our architecture.

In this paper, we treat the system call sequence as texts and regard the malware detection task as theme extraction (one task of natural language processing). Based on this assumption, we propose a new end-to-end model to detect malwares by analyzing the sequences of system calls. An intuitional graph can be seen in Figure 1. The main differences between our research and other similar studies are not only we design a new deep learning model to do the detection, but also we aim to extract deep-seated information directly from the original long system call sequence (whose length varies from 40 to 300000) without any manual defined patterns applying on it. Our end-to-end model doesn't depend on any priori knowledge. The model doesn't know what patterns are malicious or what features should be concerned about and also doesn't care the exactly meaning of any intermediate output. All that needed are just underlying system call sequences produced in the run of applications. To our knowledge, this is the first paper that analyzes the system call sequence without any pre-process on it.

To evaluate the effect of our model, we build a dataset which consists of system call sequences of 14231 Android applications (4231 malware and 10000 normal), the length of sequences varies from 40 to 300000, and we implement another method proposed in Canfora et al. [2] as a baseline. Experimental results show that our method obtained an accuracy of 93.16%, which is 2.81 percent higher than the baseline method on the same dataset.

II. MODEL CONSTRUCT

The architecture of our model is inspired by Kim et al. [5]. Details are shown in Figure 2. First, an embedding layer is used to map the system calls to vector space. Then, a module, which is called vertical multiple convolution, is used to mine the potentially high-level information from the extremely large presentation matrices of the long sequence. Finally, a multilayer perceptron with softmax layer is used to finish the classification task. Here we assume that system calls are already been converted into indices as input.

978-1-5090-6727-5/17/$31.00 ©2017 IEEE

A. System Call Embedding

Word embedding [1] is a common method in natural language processing (NLP) where inputs need to be mapped to vectors. This method can reduce the computational complexity (sparseness), and enable similar words resulted in similar vector representations. For example, *open* and *close* are both file operating functions in Linux system, in order to retain more information for the subsequent steps, they should be given similar vector representations. Therefore, we first apply the embedding lookup method to the input sequences.

B. System-call-level Vertical Multiple Convolution

In order to cut down the length of system-call sequence (our experiments shows there are usually 10000 to 100000 calls in one app's sequence) to an analyzable level, we design a module called vertical multiple convolution to filter out extraneous information and output a new representation of the input. For a fixed size kernel in the vertical multiple convolution module, it will scan the sequence by a stride of its width to get the first output. Then, it will scan the output for a second time to get another output. This operation will repeat until the output is short enough. After that, a max-pooling will be applied on the last output to obtain the final representation of the sequence on this kernel. In the vertical multiple convolution module, we use different size kernels with different quantities to do the scanning, which mentioned above, to get enough information. Concatenate all the filters' final output will give us the feature vector of the sequence.

C. Multilayer Perceptron

At this step, the only thing needed to do is to transpose the output of upper layer to class labels. For convenience, we simply adopt a fully-connected layer with softmax to accomplish the task. This computation can be formalized as:

$$y = \text{softmax}(\mathbf{W}_l x + b_l) \quad (1)$$

Where $\mathbf{W}_l$ is the parameter matrix and b_l is the bias vector.

III. Experiment and Evaluation

A. Experiment Details

Our Android applications are collected from the dataset used in Kang et al. [4]. We randomly select applications from their dataset and remove the applications that cannot run or crashed in our experimental environment. The final dataset we used contains 10000 trusted and 4231 malwares. As for the collection of the system call traces, we run the applications in an Android Virtual Device (AVD) and simulate using it. In the meanwhile, we collect the system call sequences by monitoring the applications' processes [2]. Then, some strategies are used to make them a fixed length. We randomly select 70% of the data for training, the others are for testing.

In order to test out whether our model has the ability to detect malware from the long system call sequences, and also based on the reality that there is no standard data set or benchmark yet, we consider to implement other papers' methods and test them on our data set as a comparison. We choose the method in Canfora et al. [2] as a baseline because it also study on system call sequences. And the author claimed that they obtain a malware detection accuracy of 97% on their data set, which is a very high detection accuracy. Since they use SVM in their algorithm, we call the baseline SVM based model for convenience.

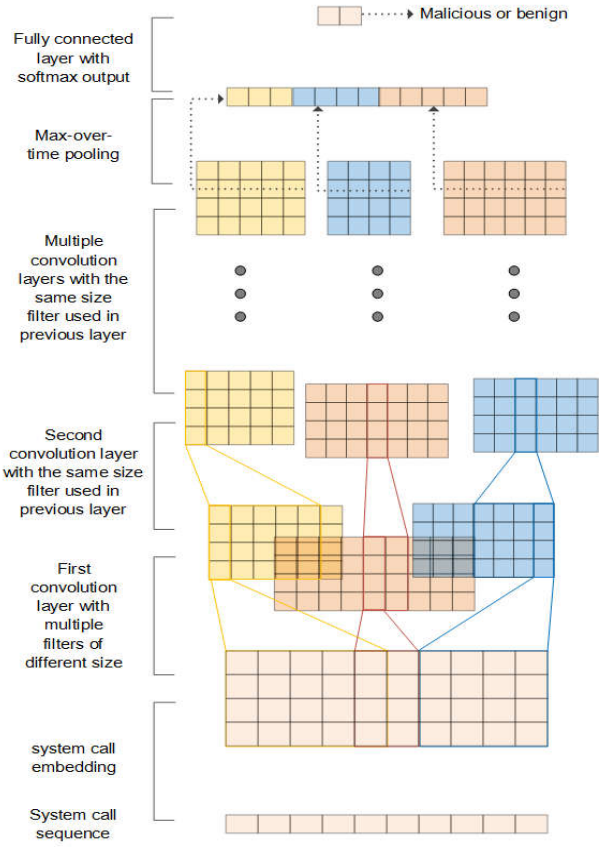

Figure. 2. The end-to-end model's architecture with three different size filters.

As for convolution module, we test the performance with different number convolution layers, different number filters and different size filters. Notice that we only do the convolution on time dimension. Thus, when telling a filter of size 2 in the following passage actually means 2×100 (100 is the embedding size). The details are shown in Table I (e.g., [2, 100, 5], which represents 5 convolution layers and each layer will have 100 filters of size 2). Finally, we use 6 different size filters which are 1, 2, 3, 4, 5, 6 and the amounts of each size filter are respectively 50, 100, 150, 200, 200, 200. Also, we choose multiple convolution layer to extract deep meaning of the sequence and a max-pooling layer is used on the last convolution layers' output. For example, we use 7 convolution layers for filters of size 2 and for filters of size 6 we use 3 layers, each of them will have one max-pooling layer. The batch size and embedding size we used respectively are 20 and 100, which making the best effect. We use cross-entropy error as loss function and AdaGrad optimize algorithm to training the model. After each epoch, all the learnable parameters will

TABLE I. PERFORMANCE COMPARISON OF DIFFERENT MODELS

Model	End-to-End	Baseline(SVM)	One-hot encoding
Accuracy	**93.16%**	90.35%	90.89%
TPR	79.01%	90.26%	75.72%
FPR	1.5%	9.59%	2.61%
Precision	95.73%	79.95%	92.54%
F-measure	86.57%	84.76%	83.29%

TABLE II. ACCURACY BETWEEN DIFFERENT NUMBER OF CONVOLUTION LAYERS, FILTERS AND DIFFERENT SIZE FILTERS

Filter size, filter number, convolution layer	Accuracy
[1, 50, 1]; [10, 100, 1]; [20, 150, 1]; [30, 200, 1]; [40, 200, 1]; [50, 200, 1]; [60, 200, 1];	81.99%
[1, 50, 5]; [10, 100, 5]; [20, 150, 5]; [30, 200, 5]; [40, 200, 5]; [50, 200, 5]; [60, 200, 5];	83.99%
[1, 50, 5]; [2, 100, 5]; [3, 150, 5]; [4, 200, 5]; [5, 200, 5]; [6, 200, 3];	93.07%
[1, 50, 7]; [2, 100, 7]; [3, 150, 6]; [4, 200, 5]; [5, 200, 5]; [6, 200, 3];	**93.16%**

be updated by backpropagation. We also play some other tricks in the training time, like dropout for prevent over-fitting, dynamic learning rate decay for training.

In order to test the performance of embedding module, we make a contrast experiment in the same architecture and same parameters which obtain the best classification accuracy, and only replace the embedding layer with one-hot encoding layer. Results are shown in the fourth column of Table I.

B. Results Analysis

In the baseline, the final classification accuracy we got is 90.35%, with a false positive rate of 9.59%. In our end-to-end model, we got a classify accuracy of 93.16%, with a false positive rate of 1.50%. Details are shown in Table I. We find that our end-to-end model has a much higher classification accuracy than traditional machine learning methods and much lower false positive rate, which means more exquisite in malicious sequence detection. In the SVM based model, it tends to predict all the test samples to be positive, which results in higher true positive rate and false positive rate in our test dataset. Our model also has a much higher precision, which means it is more confident to assert one sequence to be positive or not.

The data in Table II show that, as the number of convolution layer increases, the classification accuracy increases too. When only one convolution layer is applied, the accuracy is 81.99%. As adding more convolution layers, the accuracy reaches a saturation value of about 93%. Notice that the third (accuracy of 83.99%) and fourth row in Table II, larger filter size results in worse performance because too much noise in one filter.

Comparing the results of word embedding based model (column 2) with one-hot encoding based model (column 4) in Table I, we could find an obvious effect resulted from the embedding layer. Applying the embedding layer not only rises 2.27 percent accuracy, but also gets a 3.28 percent higher F-measure score. The true positive rate and precision rate are also 3.29 and 3.19 percent higher respectively. And the false positive rate is 1.11 percent lower than one-hot based model.

Based on these results, we could learn that the model build with vertical multiple convolution structure enable better feature extraction than the methods in Canfora et al. [2], which is a popular way to deal with the system-call sequences. Moreover, the advantages of less manual interventions also make it a more elegant and robust way to extract features. And the word embedding layer does finish a great job in transposing the system calls from text to vector representation.

IV. CONCLUSION

In this paper, we propose a novel end-to-end deep learning model to extract deep-seated information directly from the original long sequences and detect malwares without any pre-process or priori knowledge (e.g., feature selection), which is different from the existing studies. In order to test the performance of our model, we build a dataset which contains 14231 Android applications and implement the method in Canfora et al. [2] as a baseline. We also carry on experiments to test the effect of the word embedding layer. Experimental results show that word embedding representation is a better way to transpose text to vector and our end-to-end model is indeed effective in malware detection. We get an accuracy of 93.16% as a result, which is 2.81% higher than the baseline.

Based on the experimental results, we also prove that there is commonality between theme extraction and malware detection with system call sequence. Methods in natural language processing can also be applied in system call sequence analysis, which we think are much intelligent ways. In this paper, we don't analysis why a sequence being classified as malicious and how to extract the malicious fragments from the system call sequences, and we will work out these problems in the future works.

REFERENCES

[1] Y. Bengio, R. Ducharme, P. Vincent, and C. Jauvin, "A neural probabilistic language model," journal of machine learning research, vol. 3, no. Feb, pp. 1137–1155, 2003.

[2] G. Canfora, E. Medvet, F. Mercaldo, and C. A. Visaggio, "Detecting android malware using sequences of system calls," in Proceedings of the 3rd International Workshop on Software Development Lifecycle for Mobile, 2015, pp. 13–20.

[3] M. Dimjašević, S. Atzeni, I. Ugrina, and Z. Rakamaric, "Evaluation of Android Malware Detection Based on System Calls," in Proceedings of the 2016 ACM on International Workshop on Security And Privacy Analytics, 2016, pp. 1–8.

[4] H. Kang, J. Jang, A. Mohaisen, and H. K. Kim, "Detecting and classifying android malware using static analysis along with creator information," International Journal of Distributed Sensor Networks, vol. 2015, p. 7, 2015.

[5] Y. Kim, Y. Jernite, D. Sontag, and A. M. Rush, "Character-Aware Neural Language Models," in Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA., 2016, pp. 2741–2749.