Deep Learning Based Android Malware detection with Android packer classifier

Wu, Junji

(出版者 / Publisher)
法政大学大学院理工学研究科

(雑誌名 / Journal or Publication Title)
法政大学大学院紀要. 理工学研究科編

(巻 / Volume)
63

(開始ページ / Start Page)
1

(終了ページ / End Page)
3

(発行年 / Year)
2022-03-24

(URL)
https://doi.org/10.15002/00025358

Deep Learning Based Android Malware detection with Android packer classifier

Name: Wu Junji
Major: Applied Informatics Major, Graduate School of Science and Engineering,
Hosei University,
Supervisor: Atsushi Kanai

Abstract— **Deep learning as well as machine learning are grown to be one of mainstream methods of malware detections, including that on Android platforms. However, the impact of packer is often ignored in previous studies. Packers is program that aims at protecting the original code from being analyzed. It could also influence the efficiencies of selected features in machine learning. While some studies noticed this problem, the test and training are made on re-obfuscated samples, which is not convincible. Under this background, we proposed an end to end deep learning based Android malware detection method. The main idea of our method is that let the feature extraction layer to learn features of malware and packers simultaneously from part of the input, which is realized by multi-task learning. And this idea is partly inspired by the fact that unpacking is usually the first step in manual analysis of malware. To avoid re-obfuscation, conventional method is used to get obfuscation labels. The results show that our method averagely achieved 97.0%-99.8% accuracy on different datasets. The experiments also suggest that the use of packer information in multi-tasking learning could help suppress the overfitting.**

Keywords: Android Malware detection, Android packer, Deep learning, Machine learning

I. INTRODUCTION

There are numerous studies about utilizing machine learning and deep learning in Android malware detection. Arp el at. [1] developed hybrid Android malware analyze system called Drebin using both traditional analysis and machine learning. In their study, they achieved high accuracy of 95.9%, which was a game changing achievement at then. They also published their dataset as benchmark. Fine-grained Deep learning is also frequently used with the development of deep learning. For example, Niall el at. [2] used only raw opcode sequences in the one-hot format as feature and text-CNN as classifier. Although only simple data transformation is used, their model still achieved 80% to 87% of accuracies on large scale dataset with high processing speed.

However, those studies didn't analyze the impact of packer. Bacci el at. [3] illustrated that packers reduce the efficacy of static features. But static feature might still be useful when the training dataset contains obfuscated apps. Some studies tried to solve this problem. A representative method is features ranking. Suarez-Tangil el at. [4] made an obfuscated dataset and then passed all the listed features to Extra Tree algorithm to rank them by mean decrease impurity. The key idea of their study is that if a feature ranks high on both original dataset and obfuscated dataset, it is an obfuscation-invariant feature. Nevertheless, when they trained on original dataset but tested on obfuscated dataset, the accuracy dropped down significantly. Since the malicious apps are almost obfuscated by author to hide from analysis, the apps are re-obfuscated in their study, which is not likely to appear in real world.

In our study, we firstly use a tool called APKiD [5], which relies on conventional characteristic matching, to get obfuscating labels of each app. Secondly, we force the CNN layer to learn features of malware and packers in multi-task learning from part of input. While other input data are only used in malware classification. By experiments, 97.0%-99.8% accuracies on different datasets are obtained.

II. METHODS

A. Data selection

We make a hypothesis that if a kind of data is likely to be influenced or modified by packer (e.g. Dalvik opcode), then it carries information of both app itself and that of packer. Others may only contain the information of app itself. Hence, it's logical to divide features into two categories: obfuscate-variant and obfuscate-invariant. By reviewing features that are used in promised method, we choose required permissions as obfuscate-invariant feature and Dalvik opcode as obfuscate-variant feature.

To use them into deep learning training, data transformation is needed. The Dalvik opcodes are represented by Dalvik machine code and are put into a one dimensional sequence. As for the order, the opcodes of .mainActivity (entry of custom code) are firstly extracted. Secondly, the opcodes of the subclass of main class are extracted. Thirdly, the opcode of classes in the same folder of main class are extracted. If the length of sequence is still not long enough (the input length of deep learning model has to be fixed number), we fill in 0. And the length is experimentally decided as 1600. As for the permissions, they are represented by a one dimensional vector in which the corresponding value is set to 1 if a certain permission is found while the default value is 0.

B. Architecture of deep learning model

The architecture of our method is shown in Fig. 1.

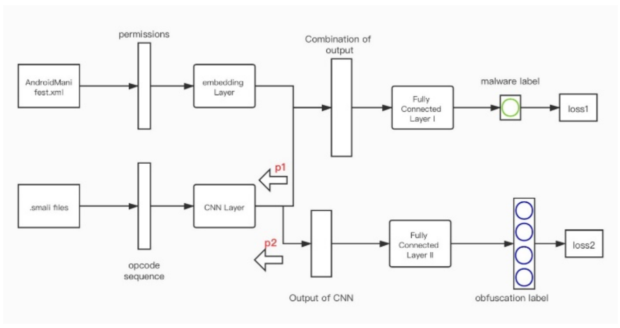

Fig 1. Architecture of deep learning model

The special part of this model is that the output of CNN layer is copied in to two branches for multi-task learning. This design is based on the key hypothesis of this study: learning features of malware and packer simultaneously could filter useless packer information and help suppress overfitting (caused by packers). It also could be the case that malware classifier learns the connection of both kinds of features and become more adaptive to packers. Hence, the obfuscate-variant features are input to the feature extraction layer (1-D CNN in this study) connected with both kinds of classifier. Meanwhile, in backpropagation, coefficients p1 and p2 are multiplied to the gradients, in order the control the influence of malware classifier and obfuscation classifier. For the reason that the report from APKiD is not 100% correct (almost impossible for conventional tools), which provides some noise labels, we keep the value of p2 small to reduce the negative impact. Moreover, it is inferred that information of packer is easier to be extracted for the fact that packer detection is easier than that of malware. In our study, we assume that p1 and p2 satisfy:

$$p1 + p2 = 1$$

Required permissions are used as obfuscate-invariant feature to enhance the robustness of malware classifier.

As for loss functions, the malware classifiers use binary cross entropy in mini-batch training while the obfuscation classifier uses mean squared error.

C. Dataset

We randomly choose 3000 apps from AMD datasets [6] and mix them with 3000 benign apps downloaded from pureapk [7]. The mixed dataset is marked as Dataset I. In our study, we use Dataset I for training in 10-fold cross validation. In order to verify the generalization ability of the model, we randomly choose another 4000 apps (exclude that in Dataset I) from AMD dataset and mark it as Dataset II. Mover, to evaluate the robustness, samples from other data source is needed. Hence, we select all 426 malicious app from CICAndMalware2017 dataset [8] to be Dataset III.

III. RESULTS

The experiment is realized by Pytorch in Ubuntu system and accelerated by a GEFORCE RTX 2080 Ti GPU. In each fold of cross validation, the trained model is tested with Dataset II and Dataset III. After all cross validations finished, the results are averaged. The average time consumed in 10-fold cross validation (including time of testing) is 1 minute and 29 seconds.

The experiment results are listed in Table 1.

Table 1. Experiment results

P1	P2	Dataset I	Dataset II	Dataset III
1	0	**97.4%**	99.4%	92.8%
0.95	0.05	96.9%	98.9%	92.4%
0.9	0.1	96.3%	99.1%	95.8%
0.8	0.2	96.5%	**99.8%**	**97.0%**
0.7	0.3	96.1%	97.1%	94.3%

IV. DISCUSSION

As we can see from Table 1, the best results are obtained when p1=0.8 and p2=0.2. And when the value of p2 is very small, the obfuscation classifier helps in negative ways. Obvious overfitting is observed when p2=0, in which case the model degenerates to single task model. Note that in experiments the result is sensitive to the hyperparameters, which suggest that maybe more samples are needed to evaluate the robustness.

V. CONCLUSION

In this paper, we proposed a deep learning based malware detection model. Both malware classifier and obfuscation classifier are used in multi-task learning, in order to force feature extraction layer to learn both kinds of features simultaneously. The required permissions are used as obfuscation-invariant feature and opcode sequences are used as obfuscation-variant features which are input to feature extraction layer. In order to get obfuscation labels, conventional detection technology is used. Hence, no re-obfuscation is needed, which is closer to real situation. Coefficients p1 and p2 are multiplied to gradients to control the influence of two different kinds of classifier.

By experiment, it is found that when p1=0.8 and p2=0.2 the best results are obtained on both Dataset II (99.8%) and Dataset III (97.0%). And obvious overfitting is observed when p1=1, which verifies the negative impact of packer on efficiency of static features.

ACKNOWLEDGEMENT

Thanks to Professor Kanai's guidance, I can successfully finish this research. Also thanks to the other staffs of Hosei University. Their kind help provide an amazingly good environment for study. Finally, thanks to my formal tutor Morita who graduated in spring 2021 for helping me manually download large number of apps.

REFERENCES

[1] Arp, Daniel, et al. "Drebin: Effective and explainable detection of android malware in your pocket." Ndss. Vol. 14. 2014.

[2] McLaughlin, Niall, et al. "Deep android malware detection." Proceedings of the seventh ACM on conference on data and application security and privacy. 2017.

[3] Bacci, Alessandro, et al. "Impact of Code Obfuscation on Android Malware Detection based on Static and Dynamic Analysis." ICISSP. 2018.

[4] Suarez-Tangil, Guillermo, et al. "Droidsieve: Fast and accurate classification of obfuscated android malware." Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy. 2017.

[5] https://github.com/rednaga/APKiD

[6] Li, Yuping, et al. "Android malware clustering through malicious payload mining." International symposium on research in attacks, intrusions, and defenses. Springer, Cham, 2017.

[7] https://apkpure.com/apk-install.html

[8] Shiravi, Ali, et al. "Toward developing a systematic approach to generate benchmark datasets for intrusion detection." computers & security 31.3 (2012): 357-374.