ISBN 978-93-86878-04-5
6th International Conference on Science, Engineering, Technology and Healthcare (SETH-2017)
Kuala Lumpur (Malaysia), Nov. 20-21, 2017

Practical Application of Machine Learning Algorithms to Android Malware Analysis

Long Nguyen-Vu, Haeun Cho and Souhwan Jung

School of Electronic Engineering
Soongsil University

Abstract**:** *The emergence of advanced mobile malware has become a great concern in recent years. Obfuscation techniques gradually defeat conventional source code analysis. In this paper, we propose a static analysis model to extract, refine manifest permissions inside each Android application package and eventually classify them as malicious or benign applications. We apply several classification Machine Learning algorithms on the same training and testing dataset to verify the proposed model. As a result, the analysis of 2400 Android applications have been conducted. We conclude that Support Vector Machine works better with smaller dataset, while Tree-based models produce better Recall.*

Keywords: *machine learning, android security, malware analysis.*

1. Introduction

The openness of architecture lets Android smartphones dominate 85% of the market-share in the first quarter of 2015, according to IDC [1]. Malware comes in different forms like ransomware (a type of malware that force users pay to unlock their devices), adware (aggressive advertising application) or chargeware (application associated with premium services like SMS or Phone Call) [2] [3]. Malware analysis is usually conducted by the two approaches: on device and off-device. Sometimes, in order to maximize the projection, several techniques are combined to enhance the accuracy of the detection, which include both static and dynamic analysis. Not only analyzing applications before they are installed on the device guarantees the security and privacy of the users, this approach also efficiently handles many applications at the same time by taking advantage of server-side hardware.

While different off-device dynamic analysis techniques are evolving, there is still a huge demand for static analysis in terms of performance and device neutral [4]. In other words, while dynamic analysis requires hardware compatible, resource control, time and effort to set up an environment, static analysis produces the result with much less cost, short analyzing time, easy to automate and only depends on decompiling tools to operate. As the emergence of advanced malware, the application source code is heavily obfuscated and protected with shared libraries. Consequently, static analysis starts suffering from semantic problems, originating from decompiling tools fail to defeat obfuscated code [5][6].

In this paper, setting aside the problem of dealing with obfuscation techniques, we propose a technique to extract, refine and convert Android permissions to training dataset of 4 classifiers: Decision Tree, Random Forest, Naïve Bayes and Support Vector Machine. Our study shows that with a relatively small training dataset, some classifiers may outperform the others and output accurate evaluation to enforce malware detection platform. We systematically design the platform to work with different API levels of Android. The training set of 2000 applications and 400 testing samples have been studied carefully to provide the insights about manifest permissions used in Android applications. We also provide concrete data to elaborate the proof of concept in this study.

In Section 2, we briefly provide the background and related works regarding Android malware detection. We propose our model and mathematical proof in Section 3, with result analyses in Section 4. Finally, we conclude our work in Section 5.

2. Background and Related Works

2.1 Android Permissions

Android Operating System lets each application run with different system identity, namely UID (User ID) and GID (Group ID). This design enforces each application to run in separated "sandboxes", also known as "process sandbox". Runtime permission is applied in Android Nougat 6.0, that means users can decide whether to grant the privilege to an application or not, thus provides an addition layer of security. As most users are unware of the potential harms caused by those permissions, some applications may incentivize them to accept the request. The process of request and grant permissions are described in Figure 1, both in installation and runtime.

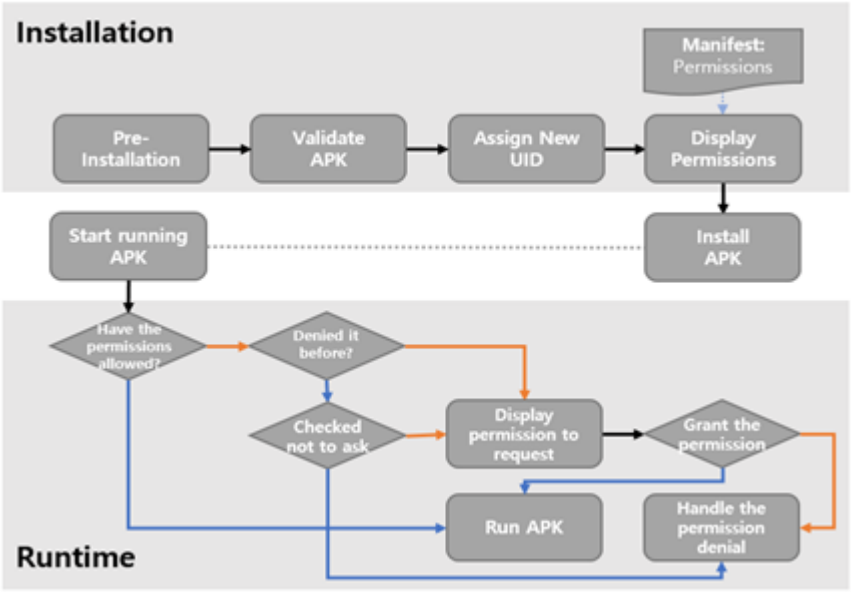

Fig. 1: The process of handling manifest permissions during installation and runtime

We realize the root cause of the problem lies on manifest permissions. Therefore, in this study, we focus on analysing more than 200 permissions in Android OS to enhance the performance of malware classifier.

2.2 Related Works

Android malware has been studied rigorously over the years. Recently several proposals regarding machine learning in detection are considered to resolve the problem with heavy code obfuscation. The work of Yerima et al. [14] on Naïve Bayes model focuses on classifying Android malware based on several features like API calls, permissions. The authors use Mutual Information to assert important factors to be used as independent features in Bayesian algorithm. The authors later extend their work to improve the accuracy with detection by performing paralleled analysis [15] [16] 17]. Our work, on the other hand, targets 3 more machine learning models to have better view of each model, as well as realize their own advantages and application.

Zhao Shuai et al. [18] exploit the behaviour rules and construct them with tree model, the study achieves the True Positive rate of 88.14%. In this paper, we try to consider additional models besides Tree-Based by adding ensemble learning from Random Forest to enhance the result and achieve 94.74% (the highest recall in Section 4, which belongs to Random Forest)

3. Proposed Analysis Model

3.1 Binary Permissions

Android OS comes with 141 default permissions and many developer-defined (or custom) permissions. These permissions allow application developers to equip their products a wide range of functionality. We consider only permissions that have higher impact to the corpus. By examining the number of permissions used in 2000 samples, we select 54 permissions to be used as features for training phase. These 54 permissions are converted into a 1-dimension binary array, which is used to represent permissions of each training data. Given an application with n types of permissions, the array represents permissions of this application will have bit 1 corresponding to permission that is available, and will have bit 0 for permission that is not available.

Below is an example of binary permissions for package name "com.jobkorea.app". For brevity, we reduce the length of bPermissions.

{"packageName": "com.jobkorea.app",

"bPermissions": [0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, …, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0]}

After this phase, all training apps will have their manifest permissions converted into binary permissions. All arrays are similar in length and order. The training set will be fed into 4 different models for the training phase.

3.2 Proposed Model

In this section, we use 4 mathematical based models to express the training data and create the classifier:

Decision Tree: We use CART (Gini Impurity) to select the split [8]

$$I_G(p) = 1 - \sum_{i=1}^{J} p_i^2$$

J: set of classes, in this case – 54 manifest permissions
Pi: the fraction of items labelled with class i (belongs to J)
Decision Tree approach will base on Gini Impurity to decide the split criteria, which are 54 permissions in the previous section.

- Random Forest: we use this ensemble learning method to verify the effectiveness of using bootstrapping technique compared to Decision Tree approach [9]
- Naïve Bayes [10]:

$$P(y|x1, \dots, xn) = \frac{P(y) * P(x1,\dots,xn \mid y)}{P(x1,\dots,xn)}$$

$y \in$ {malicious, benign}
P(x1,…, xn) is the probability of each feature (permission)

- **Support Vector Machine [12]:** We apply the Support Vector Classifier library (libsvm) to train and test the dataset. The details will be discussed in the next section.
  With the assumption that 54 permissions are independent to each other, we apply Naïve Bayes classifier to predict the malice of an application based on its permissions. We calculate the Precision, Recall, Accuracy of the 4 models as in Table I.

4. Experiment

We build 4 classifiers based on 4 models in the previous section. With the support of Scikit-Learn [13] we use them to predict the dataset of 200 malicious and 200 benign Android applications.

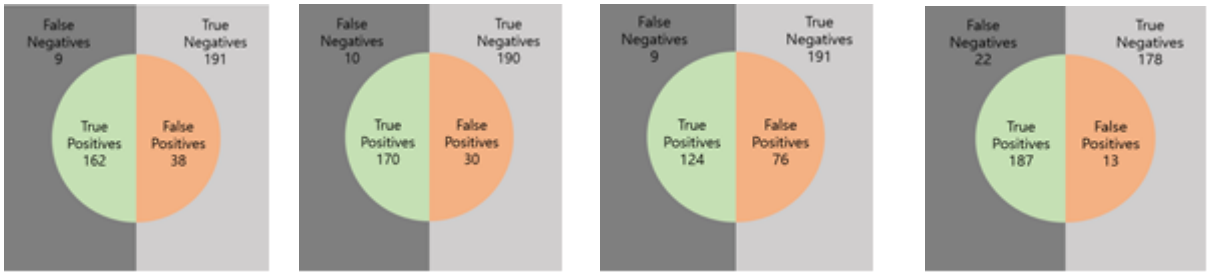

Fig. 2: Analysis results of 4 Machine Learning Models

Overall, Random Forest performs better than Decision Tree and has the widest coverage (recall) of the 4 algorithms. Support Vector Machine, as expected, outperforms all others with 93.5% of Precision and 91.25% of Accuracy but does not cover as many malicious applications as the others.

$$\text{Precision} = \frac{TP}{TP + FP}$$

$$\text{Recall} = \frac{TP}{TP + FN}$$

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

TABLE I: Analysis result of 400 testing samples & 4 Machine Learning algorithms

	Decision Tree	Random Forest	Naïve Bayes	Support Vector Machine
Precision	81%	85%	62%	93.5%
Recall	94.74%	94.44%	93.23%	89.47%
Accuracy	88.25%	90%	78.75%	91.25%

By training 2000 Android applications and testing with 400 samples, we realize that if the detection takes into account 4 Machine Learning models, then the accuracy will be significantly increased. Specifically, by assigning higher weights on models that perform better in each category (Precision, Recall, Accuracy), the classification of malicious and benign can result in approximately 0.5% of False Negative. That means the chance of incorrectly labelling a malicious application as benign is low, compared to others in related works.

5. Conclusion

In this paper, we propose the technique to convert manifest permissions in Android into binary arrays, which can be fed into different machine learning models for training and testing. While there are still many important factors to be considered, our analysis results show that the use of multiple machine learning algorithms on the same training and testing dataset can greatly enhance the performance of the detection system. As the work is still limited in size of dataset, as well as the semantics between features (permissions) are still not asserted properly, we believe a lot of future work should be done to bring this implementation into production. Nevertheless, our study provides a complete application of machine learning algorithms into real life problem. The source code of this work will be released promptly, as a contribution to the open community.

6. Acknowledgements

This research was supported by the MSIT (Ministry of Science and ICT), Korea, under the ITRC (Information Technology Research Center) support program(IITP-2017-2012-0-00646) supervised by the IITP (Institute for Information & communications Technology Promotion)

This work was supported by Institute for Information & communications Technology Promotion(IITP) grant funded by the Korea government(MSIT) (No.2017-0-01853, Machine Learning based Intelligent Malware Analysis Platform)

7. References

[1] Smartphone Market Share (2017). *IDC. [Online]*. Available: https://www.idc.com/promo/smartphone-market-share/os

[2] Yang Tianda et al., "Automated detection and analysis for android ransomware," *High Performance Computing and Communications (HPCC)*, 2015

[3] Felt Adrienne Porter et al., "A survey of mobile malware in the wild," *Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices*, 2011.

[4] Kimberly Tam et al., "The evolution of android malware and android analysis techniques," *ACM Computing Surveys (CSUR)*, 2017

[5] Schulz Patrick, "Code protection in android," *Insititute of Computer Science, Rheinische Friedrich-Wilhelms-Universitgt Bonn, Germany*, 2012

[6] Kovacheva Aleksandrina, "Efficient code obfuscation for Android." *International Conference on Advances in Information Technology, Springer*, 2013.

[7] Android Manifest Permissions, *Android Official Website. [Online]*. Available: https://developer.android.com/reference/android/Manifest.permission.htm

[8] L. Breiman, J. Friedman, R. Olshen, and C. Stone, "Classification and Regression Trees," Wadsworth, Belmont, CA, 1984

[9] Breiman L., "Random Forests Machine Learning," *View Article PubMed/NCBI Google Scholar*, 2001

[10] Harry Zhang, "The optimality of naive Bayes," *AA1.2*, 2004

[11] Wu, Lin and Weng, "Probability estimates for multi-class classification by pairwise coupling," *JMLR 5:975-1005*, 2004.

[12] C. Cortes, V. Vapnik,"Support-vector networks," *Machine Learning, 20*, pp. 273-297,1995

[13] Lars Buitinck et al., "API design for machine learning software: experiences from the scikit-learn project," arXiv preprint arXiv:1309.0238, pp. 108-122, 2013

[14] Yerima Suleiman et al., "A new android malware detection approach using bayesian classification," *IEEE 27th International Conference on Advanced Information Networking and Applications (AINA*," 2013.

[15] Yerima, Suleiman Y., Sakir Sezer, and Gavin McWilliams, "Analysis of Bayesian classification-based approaches for Android malware detection," *IET Information Security* 8.1, 2014

[16] Yerima, Suleiman Y., Sakir Sezer, and Igor Muttik. "Android malware detection using parallel machine learning classifiers." *Next Generation Mobile Apps, Services and Technologies (NGMAST), 2014 Eighth International Conference on*. IEEE, 2014.

[17] Allix Kevin, et al. *Machine Learning-Based Malware Detection for Android Applications: HistoryMatters!*, University of Luxembourg, SnT, 2014.

[18] Zhao Shuai et al., "Attack tree based android malware detection with hybrid analysis," *IEEE 13th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, 2014.