Detecting Android Malware Based on Analyzing Abnormal Behaviors of APK File

Cho Do Xuan [1†],
chodx@fe.edu.vn
Information Assurance dept. FPT University, Hanoi, Vietnam

Summary
The attack trend on end-users via mobile devices is increasing in both the danger level and the number of attacks. Especially, mobile devices using the Android operating system are being recognized as increasingly being exploited and attacked strongly. In addition, one of the recent attack methods on the Android operating system is to take advantage of Android Package Kit (APK) files. Therefore, the problem of early detecting and warning attacks on mobile devices using the Android operating system through the APK file is very necessary today. This paper proposes to use the method of analyzing abnormal behavior of APK files and use it as a basis to conclude about signs of malware attacking the Android operating system. In order to achieve this purpose, we propose 2 main tasks: i) analyzing and extracting abnormal behavior of APK files; ii) detecting malware in APK files based on behavior analysis techniques using machine learning or deep learning algorithms. The difference between our research and other related studies is that instead of focusing on analyzing and extracting typical features of APK files, we will try to analyze and enumerate all the features of the APK file as the basis for classifying malicious APK files and clean APK files.

Key words:
APK file; behavior analysis; deep learning; anomaly detection; machine learning

1. Introduction

The research [1] listed vulnerabilities in the Android operating system that make it a favorite target for cyber attackers. The document [2] reported the number of cyber-attacks through mobile devices using the Android operating system. In order to detect malicious APK files, recent approaches often rely on static analysis and dynamic analysis techniques to seek abnormal behaviors in these files. Regarding the static analysis method, the approaches often rely on signatures and extract signatures from the sample malware. Although it is effective in detecting known malware, it is not enough to detect unknown malware. Regarding the dynamic analysis method, the approaches often use virtualization tools to record the abnormal behavior of APK file types. The studies [1, 2] have pointed out that the approach of detecting malicious APK files based on dynamic analysis techniques using machine learning and deep learning algorithms has brought high efficiency due to the support of virtualization and big data technologies. The research [1] listed some approaches for detecting malicious APK files based on a process of extracting and handling abnormal behaviors of these APK files. However, such approaches have the disadvantage that data features do not exhibit generalization. This demonstrates that such extracted features could give high efficiency in some experimental datasets, but when applied to datasets of other APK files, such behavior cannot be extracted. To fix the above disadvantage, in this paper, we propose a novel method based on analyzing different components of APK files and then calculating and enumerating features on all of these components. Finally, based on the feature vector of all the components in these APK files, we use typical deep learning and machine learning algorithms for classification in order to detect clean APK files and malicious APK files. The process in our approach includes the following 3 main processing steps:

- Step 1: Collect behaviors of APK files based on dynamic analysis tools.
- Step 2: Extract and statistic the behaviors of APK files based on all the different components in these APK files.
- Step 3: Classify clean APK files and malicious APK files using machine learning and deep learning algorithms.

2. Related Works

In the study [3], Yi Zhang et al. proposed a method of detecting malware on Android devices using the Convolutional Neural Network (CNN) model. Accordingly, the authors use CNN network architecture to classify Android malware based on the dynamic analysis method. In the experimental section, the authors compared and evaluated the CNN model with traditional classification methods such as Linear Support Vector Machine (Linear-SVM) and k-nearest Neighbors (KNN). The experimental results showed that the CNN model gave better performance than other models on all measures.

Besides, Xu Jiang [4] proposed a method for detecting Android malware using Fine-Grained Features. Accordingly, the author sought ways to analyze the

Manuscript received June 5, 2021
Manuscript revised June 20, 2021
https://doi.org/10.22937/IJCSNS.2021.21.6.4

APKtool files according to AndroidManifest, Smail, Lib folder components and then extract the features on these components and then use KNN, SVM, J48 Decision Tree (J48), Naive Bayes (NB) algorithms to analyze to conclude about malware. In the experimental section, J48 algorithm gave better results than other algorithms on all measures.

Wang [5] proposed an API call analysis method to look for abnormal behaviors of malware and then use graphing techniques to analyze and evaluate API call behaviors to conclude about Android malware. Specifically, in this study [5], the author analyzed API calls into different components and then used embedding techniques including DeepWalk, Node2vec, Structural deep network embedding methods to train features of API call. Finally, to conclude about Android malware, the author used CNN, Long Short Term Memory (LSTM) models to compare with the author's proposed method. The experimental results showed that the proposed method of this team gave higher efficiency than the CNN, LSTM models on all measures.

In the study [6], Minghui Cai et al. proposed the Graph Convolutional Network (GCN) method to analyze and evaluate enhanced function call graphs. In the experimental section, the authors compared and evaluated the GCN model with other models such as Linear Regression (LR), Decision Tree (DT), SVM, KNN, Random Forest (RF), Multi-layer Perceptron (MLP), and CNN.

3. Proposing the Detection Method

3.1 The model architecture

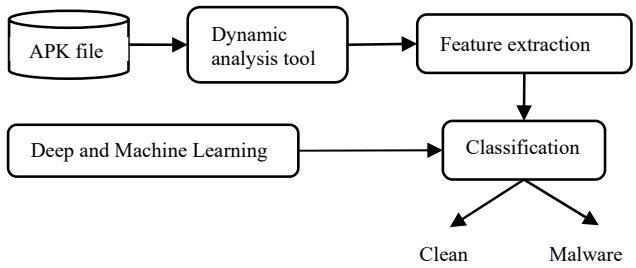

Fig. 1 Architecture of the Android malware detection system

From Figure 1, the architecture of the proposed Android malware detection system includes the following components:

- APK file: is an application file in the form of APK compression. These APK files will be analyzed to synthesize normal and abnormal behaviors in order to detect malicious APK files and clean APK files.
- Dynamic analysis tool: This is a virtualized environment that allows executable APK files in order to record and synthesize all abnormal behaviors of APK files.
- Feature extraction: Based on the results when executing APK files in a virtualized environment, we will analyze log files of dynamic analysis tools to collect abnormal behaviors. This paper will use 215 features based on feature groups collected in the JSON files from the log of the analysis tool. The list of features and feature groups will present in detail in section 3.2 of the paper.
- Data classification: After building the feature vector, this feature vector will be used for the malware analysis module using machine learning algorithms. The result of the system is an assessment of the malicious degree of the file. This study will detect malware in the Android application based on the manifest file and details of the Java source file, required permissions, hash value of the file.

3.2 Feature selection and extraction

Table 1: List of features extracted in the APK file

No	*Group*	*Name of Feature*	*Description*	*Type*
1	API call signature	abortBroadcast	Returns the flag indicating whether or not this receiver should abort the current broadcast	int64

...				
73		URLClassLoader	This class loader is used to load classes and resources from a search path of URLs referring to both JAR files and directories.	int64
74	Commands signature	/system/app		x
...				
79		remount	The adb remount command will help ADB to remount the /system, /oem, and /vendor partitions in read-write mode on your device	x
80	Intent	android.intent.action.ACTION_POWER_CONNECTED	Broadcast Action: External power has been connected to the device. This is intended for applications that wish to register specifically to this notification.	int64
...				
102		intent.action.RUN	The general action to be performed, such as ACTION_VIEW, ACTION_EDIT, ACTION_MAIN, etc	int64
103	Manifest Permission	ACCESS_COARSE_LOCATION	Allows an app to access approximate location derived from network location sources such as cell towers and Wi-Fi	int64
...				
215		WRITE_USER_DICTIONARY	Allows the app to write new words into the user dictionary	int64

Table 1 shows the features of each feature group extracted from the JSON file in the log of the dynamic analysis tool. From Table 1, seeing that this paper uses 4 main feature groups: API call signature [7, 8]; Commands signature [8]; Intent [9]; Manifest Permission [10]. The studies [1, 2] defined the role and importance of these feature groups in detail. In this paper, based on these 4 feature groups, we will extract features as the basis for detecting the Android malware.

3.3 Malware detection method

This paper uses a number of deep learning and machine learning algorithms to classify APK files into normal or malicious. Regarding machine learning, we choose to use the RF and SVM algorithms. The documents [11, 12, 13] described in detail the mathematical basis and operating principle of these algorithms. Regarding deep learning, we use 4 main algorithms: MLP, CNN, LSTM, Bidirectional LSTM (BiLSTM). The documents [14, 15, 16, 17] presented about how they work and their applicability. In this paper, we will proceed to apply algorithms in the task of detecting malware. Based on the experimental results, we will have a basis to evaluate the effectiveness of each algorithm in the task of detecting malware.

4. Experiments and evaluation

4.1 Experimental dataset

This paper uses 18,835 APK files consisting of 12,015 clean APK files and 6,820 malicious APK files [18, 19, 10]. These APK files are extracted into 215 features that are listed in Table 1.

4.2 Experimental scenario

In this paper, to evaluate the effectiveness of the Android malware detection model, we conduct 2 main experimental scenarios: i) detecting Android malware using 2 supervised machine learning algorithms: SVM and RF; ii) detecting Android malware using 4 deep learning models: MLP, CNN, LSTM, and BiLSTM. Both two scenarios experiment on a dataset that is randomly divided (where 80% is used for training and 20% is used for testing).

4.3 Classification Measures

- **Accuracy:** is the ratio between the number of correctly predicted files and the total number of files in the test dataset. It is calculated by the following formula:

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

- **Precision:** is the ratio of true positive files to the total number of files classified as positive. It is calculated by the formula:

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

- **Recall**: is the ratio of true positive files to the total number of actually positive files. It is calculated by the formula

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

- **F1 Score:** is harmonic mean of precision and recall (assuming that these two quantities are nonzero)

Where: **TP - True positive:** The number of malicious files classified correctly; **FN - False negative:** The number of malicious files classified as clean; **TN - True negative:** The number of clean files classified correctly; **FP - False positive:** The number of clean files classified as malicious.

4.4 Experimental results

Table 2: Experimental results when using machine learning algorithms

Algorithm	*Parameters*		*Precision*	*Recall*	*F1-score*	*Accuracy*	*cohen_kappa_score*
SVM	Gamma = auto; C=1.0; kernel=rbf		97%	97%	97%	97.07%	93.73%
	Gamma = 0.01; C=1.0	kernel=rbf	98%	98%	98%	97.61%	94.88%
		kernel=linear	98%	98%	98%	97.66%	95.01%
		kernel=poly	90%	89%	89%	89.06%	75.42%
		kernel=sigmoid	97%	97%	97%	96.94%	93.46%
Random Forest	class_weight = None; criterion=gini; max_depth = None; max_features = auto; min_inpurity_split = None; min_sample_split = 2; random_state = 45	n_estimators=10	99%	99%	99%	98.77%	97.39%
		n_estimators=50	99%	99%	99%	98.85%	97.56%
		n_estimators=100	99%	99%	99%	98.83%	97.50%
		n_estimators=250	**99%**	**99%**	**99%**	**98.88%**	**97.62%**

From the experimental results in Table 2, seeing that the RF algorithm gave better results than the SVM algorithm on all measures. In particular, with the SVM algorithm, the best results of the algorithm are Precision, Recall, F1-score as 98%, and Accuracy as 97.66% with corresponding parameters Gamma = 0.01, C = 1.0, kernel = linear. Meanwhile, the lowest results of the RF algorithm (Precision, Recall, F1-score as 99% and Accuracy as 98.77%) are also higher than SVM's highest result. In addition, overall, we noticed that there is not much difference in the experimental results of the RF algorithm because the difference between the lowest and highest result is only about 0.05%. And once again, these experimental results have proven the remarkable efficiency of the RF algorithm compared to other supervised classification algorithms.

Table 3: Experimental results when using deep learning algorithms

Algorithm	*Parameters (activation)*	*Precision*	*Recall*	*F1-score*	*Accuracy*
CNN	ReLU	99%	99%	99%	98.89%
LSTM	ReLU	**99%**	**99%**	**99%**	**99.37%**
	Sigmoid	98%	98%	98%	97.87%
BiLSTM	ReLU	99%	99%	99%	98.67%
	Sigmoid	98%	98%	98%	97%
MLP	ReLU	99%	99%	99%	98.88%

From the experimental results in Table 3, seeing that the deep learning models including CNN, MLP, LSTM, and BiLSTM have relatively good classification results. In which, the LSTM deep learning model with the ReLU activation function gave better results than other deep learning models on all measures. However, overall, we noticed that deep learning models applied in this study brought stable results with approximately the same accuracy. Especially, the BiLSTM model, which we expect is more effective than other deep learning models, did not promote the ability to remember and highlight abnormal features and behaviors in this experimental dataset. We believe that the cause of this problem is that in this experiment, the input of both the LSTM and BiLSTM were not processed by each group of features. This makes the LSTM and BiLSTM models do not have much data to learn, so the analysis and evaluation process of the model is no different from the CNN and MLP models.

Comparing the experimental results in Tables 2 and 3, we found that the deep learning models bring more stable and higher results than the machine learning models. One of the reasons for this problem is that in this study, we used 216 features that represent abnormal behavior of APK files, so machine learning algorithms do not promote their speed and accuracy. In contrast, deep learning models with the good ability to remember and select features have brought a clear effect.

5. Conclusion

Distributing malware through end-users via mobile devices has been, is, and will be a major challenge for malware detection systems. In this paper, based on deep learning models and abnormal behaviors of APK files, we proposed a method for detecting malicious APK files. In the experimental section, based on the scenario of comparing and evaluating the results of machine learning algorithms with deep learning models, we have provided a number of options and methods to build the Android malware detection system based on analyzing behaviors of APK files. Specifically, if only interested in the accuracy of malware detection systems, the LSTM deep learning model can be used as an optimal solution. However, in terms of detection time and computational complexity, it is clear that the RF algorithm brings more advantages. In the future, in order to improve the efficiency of this method, there are 2 problems that need to be optimized: i) remove redundant features; ii) use combined deep learning models or 2D models in order to increase the efficiency of the process of synthesizing and extracting features that are related to each other.

References

[1] Yue Liu, Chakkrit Tantithamthavorn, Li Li, Yepang Liu: *Deep Learning for Android Malware Defenses: a Systematic Literature Review*. ArXiv:2103.05292v1.

[2] Wang W., Zhao M., Gao Z., Xu G., Xian H., Li Y., Zhang X.: *Constructing Features for Detecting Android Malicious Applications: Issues, Taxonomy and Directions.* IEEE Access, vol. 7, pp. 67602–67631 (2019). doi:10.1109/access.2019.2918139.

[3] Yi Zhang, Yuexiang Yang, Xiaolei Wang: *A Novel Android Malware Detection Approach Based on Convolutional Neural Network*. In: Proc. of the 2nd International Conference on Cryptography, Security and Privacy, pp. 144–149 (March 2018). https://doi.org/10.1145/3199478.3199492

[4] Xu Jiang, Baolei Mao, Jun Guan, Xingli Huang: *Android Malware Detection Using Fine-Grained Features*. Scientific Programming (2020). https://doi.org/10.1155/2020/5190138.

[5] Abdurrahman Pekta, Tankut Acarman: *Deep learning for effective Android malware detection using API call graph embeddings*. Soft Computing. https://doi.org/10.1007/s00500-019-03940-5.

[6] Minghui Cai, Yuan Jiang, Cuiying Gao, Heng Li, Wei Yuan: *Learning features from enhanced function call graphs for Android malware detection*. Neurocomputing, vol. 423, pp. 301–307 (2021).

[7] Ali Feizollahm, Nor Badrul Anuar, Rosli Salleh, Guillermo Suarez-Tangil, Steven Furnell: *AndroDialysis: Analysis of Android Intent Effectiveness in Malware Dectection*. In: 30th IEEE/ACM International Conference on Automated Software Engineering (ASE) (2015).
[8] P. Faruki, V. Ganmoor, V. Laxmi, M.S. Gaur, A. Bharmal: *AndroSimilar: robust statistical feature signature for Android malware detection*. In: Proc. of the 6th International Conference on Security of Information and Networks, ACM, pp. 152–159 (2013).
[9] Asaf Shabtai Uri Kanonov, Yuval Elovici, Chanan Glezer, Yael Weiss: *Andromaly: a behavioral malware detection framework for android devices*. Journal of Intelligent Information Systems, vol. 38, pp. 161–190 (2012).
[10] *Dataset android malware permission*: https://www.kaggle.com/xwolf12/datasetandroidpermissions
[11] S.S. Shai, B.D. Shai: *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press (2014).
[12] JohnShawe-Taylor, ShiliangSun: *Kernel Methods and Support Vector Machines*. Academic Press Library in Signal Processing, vol. 1, pp. 857-881 (2014).
[13] Leo Breiman: *Random Forests*. Machine Learning, vol. 4(1), pp. 5–32 (2001).
[14] Daniel Svozil, Vladimir Kvasnicka, Jiří Pospíchal: *Introduction to multi-layer feed-forward neural networks*. Chemometrics and Intelligent Laboratory Systems, vol. 39(1), pp. 43–62.
[15] Zewen Li, Wenjie Yang, Shouheng Peng, Fan Liu: *A Survey of Convolutional Neural Networks: Analysis, Applications, and Prospects* (2020). ArXiv:2004.02806.
[16] Keiron O'Shea, Ryan Nash: *An Introduction to Convolutional Neural Networks* (2015). ArXiv:1511.08458.
[17] Sepp Hochreiter, Jürgen Schmidhuber: *Long Short-Term Memory*. Neural Computation, vol. 9(8), pp. 1735–1780 (1997).
[18] https://www.kaggle.com/razgallah/apps-base
[19] https://www.kaggle.com/tamirkh/apks-dataset
[20] https://www.kaggle.com/covaanalyst1/cova-dataset

Dr. Cho Do Xuan is currently a lecturer at the Faculty of Information Technology at Posts and Telecommunications Institute of Technology and FPTU in Vietnam
In 2008, received a bachelor's degree in the Saint Petersburg Electrotechnical University "LETI" on a specialty "Computer science and computer facilities", Russia. In 2010, graduated a masters from the Saint Petersburg Electrotechnical University "LETI" on a specialty "Computer science and computer facilities", Russia. In 2013, received a PhD in the Saint Petersburg Electrotechnical University "LETI", on a specialty CAD. Russia. Area of scientific interests - modeling, control systems, algorithmization, information security.
Email: *chodx@ptit.edu.vn* *chodx@fe.edu.vn*