ISSN 2278-3091
Volume 9, No.3, May - June 2020
International Journal of Advanced Trends in Computer Science and Engineering
Available Online at http://www.warse.org/IJATCSE/static/pdf/file/ijatcse154932020.pdf
https://doi.org/10.30534/ijatcse/2020/154932020

Detecting malicious applications on Android is based on static analysis using Deep Learning algorithm

Lai Van Duong[1], Tisenko Victor Nikolaevich [2], Do Hoang Long[3], Nguyen Quang Dam[4], Nguyen Quoc Hoang[5]
[1,3,4,5]Information Assurance dept. FPT University, Hanoi, Vietnam, duonglvse05009@fpt.edu.vn, longdhse05220@fpt.edu.vn, damnqse05820@fpt.edu.vn, hoangnqse06012@fpt.edu.vn
[2]Department Quality Systems, Peter the Great St. Petersburg Polytechnic University, Russia, St.Petersburg, Polytechnicheskaya, 29, v_tisenko@mail.ru

ABSTRACT

Attacks on users through mobile devices in general, and mobile devices with Android operating system in particular, have been causing many serious consequences. Research [1] lists the vulnerabilities found in the Android operating system, making it the preferred target of cyber attackers. Report [2] statistics the number of cyberattacks via mobile devices and mobile devices using Android operating system. The report points out the insecurity of information from applications downloaded by users from Android apps stores. Therefore, to prevent the attack and distribution of malware through Android apps, it is necessary to research the method of detecting malicious code from the time users download applications to their devices. Recent approaches often rely on static analysis and dynamic analysis to look for unusual behavior in applications. In this paper, we will propose the use of static analysis techniques to build a behavior of malicious code in the application and machine learning algorithms to detect malicious behavior.

Key words: Malicious applications on Android, static analysis, abnormal behavior, machine learning.

1. INTRODUCTION

In the development of the smart mobile market, Android, which is an open-source platform of Google, has become one of the most popular mobile operating systems. Along with the development of the Android operating system, the number of malware developed in this operating system is also increasing. In 2012, the number of newly discovered malware on the Android platform was 214.327 samples, by 2018 it increased to 8.246.284 newly discovered malware samples [2]. This leads to malicious software on Android also need to improve methods and techniques. There has been a lot of research focusing on malware detection on Android. One of the common methods includes signature-based methods, extracting signatures from malware samples methods. Although it is effective to detect known malware, it is not enough to detect unknown malware. There are also several methods based on the network activity analysis of the software. This method monitors the network traffic of a sample application and tries to detect malware by comparing it with a blacklist of DNS and IP addresses. This method cannot detect unknown malware, because the blacklist is only generated from activities of known malware. To overcome the disadvantages of these traditional methods, recent approaches have focused on researching and extracting the unusual behavior of Android applications. To extract unusual behavior in applications, studies often use static and dynamic analysis techniques. Static analysis and dynamic analysis process will produce a variety of data and formats. Each format and component will provide different features and behaviors of the application. In this paper, we choose to use static analysis techniques to analyze applications to synthesize and represent information about AndroidManifest files. We will then proceed to extract the app's unusual behavior based on AndroidManifest file analysis. To detect abnormal behavior of the application, we choose a supervised machine learning algorithm.

2. RELATED WORKS

Isohara [3] presents a method for detecting malware by analyzing the properties of files in application patterns. Although this approach can detect some unknown malware that isn't detected by the blacklist or signature-based analysis method, the cost of analysis depends on the number of files in the sample analyzed. Enck et al. [4] proposed a method to prevent the installation of applications with dangerous permissions or intent filters (a mechanism to perform cooperation between Android applications). However, the method can lead to inaccurate detection, because the information used in the method is not sufficient to distinguish malware from benign applications. There is also a malware analysis method based on the analysis of API calls in smalifiles as in the study of Wu et al. [5]. However, the implementation of the above method will raise the problem that huge analytical cost, it depends on the number and the size of the file in the original application. In this paper, we research the method for detecting malicious apps on Android, based on the abnormal behavior of AndroidManifest files using machine learning algorithms.

3. ANALYZE ANDROID APPLICATION BASED ON THE STATIC ANALYSIS METHOD

3.1 Introduce static analysis technique

The proposed malware analysis method model for Android applications consists of the following 5 main steps:

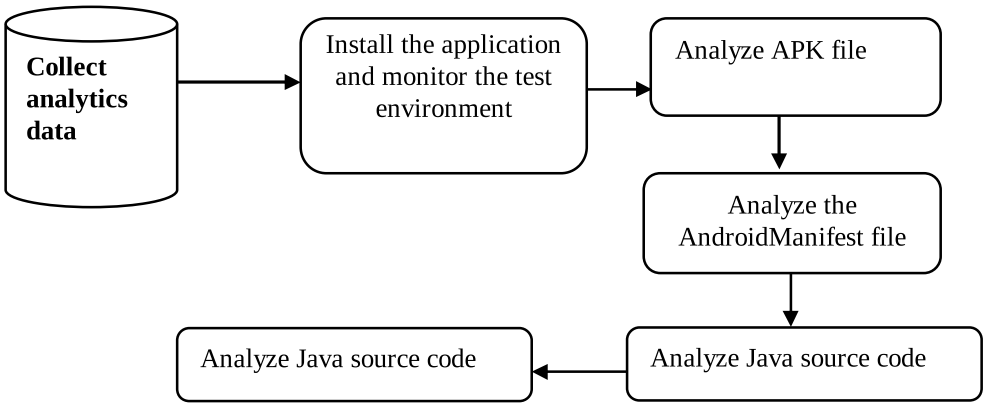
Collect analytics data
Install the application and monitor the test environment
Analyze APK file
Analyze the AndroidManifest file
Analyze Java source code
Analyze Java source code

Figure 1: Static analysis model

Figure 1 shows the static analysis procedure for an application of Android. Details of the steps in the analysis process are as follows:

Step 1: Install the application to test on the test environment When receiving an APK file is suspected of being malicious, it should first be installed on the environment sample simulator to monitor the behavior and permissions required by the application during the installation process. Signs that need to be monitored include: Icon; Application permissions required upon installation; Monitor network traffic when the application is installed, etc.

Step 2: Analyze the APK file: The APK file is essentially a zip file containing application resources. The APK file may contain malicious code to execute when called or run the file, which is modified extension, to deceive the system. Therefore, the APK files in the file will be meticulously analyzed. The result of step 2 is to get a list of suspicious files included in the sample APK file.

Step 3: Perform AndroidManifest file analysis: In the AndroidManifest file, the following features should be noted: Permission; Activities; Intent; Main. To extract these features, you can use some tools such as SmaliViewer; ApkTool. Particular should pay attention to the permissions required in the AndroidManifest file and the permissions required for installation (obtained in step 1).

Step 4: Analyze java source code: In Java source code analysis, reverse engineering will convert the program code into a readable form [6]. To convert from .Dex file to .jar format, you can use the dex2jar tool. Converting to .smali can be done using the ApkTool tool. To open the converted code, you can use some tools, such as JDGui, NotePad++ and ByteCodeViewer.

Step 5: Use automated analysis tools: In addition to the above four methods, it is recommended to combine automated analysis tools and dynamic analysis to get more information about malicious software and make comparisons leading to more accurate evaluation results. The tools that can be used here areas [6]: Mob SF; CuckuDroid.

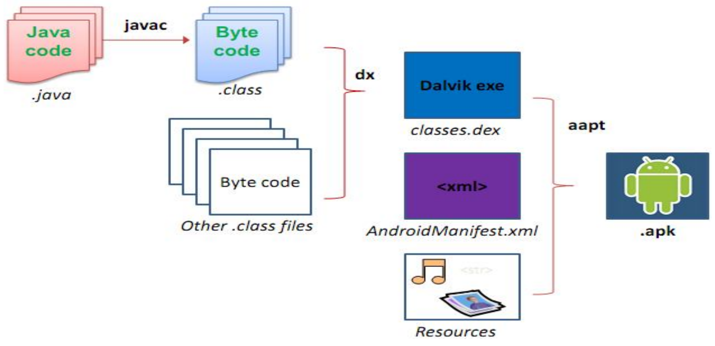

Figure 2: Components of the application file

3.2 Malware analysis technique in an Android application is based on Manifest Files analysis

An Android application consists of the following components:

AndroidManifest.xml: The file provides information necessary for the application to work properly with the Android system, in which the system will read this file before it can run other application code. Some of the information obtained includes the permissions that the application requires, the minimum APIs for the application to function, the list of libraries that application need, etc.

Classes.dex: Java source code is compiled to run in the Dalvik virtual machine.

Resources: consists of two main parts, a res directory containing the resources are not pre-compiled such as images, String, etc. and the resources.arsc file containing the pre-compiled resources.

META-INF: This directory contains some metadata such as the certificate of the application, the manifest file of the java application.

Lib: The libraries are precompiled to suit each hardware platform.

Assets: contains resources that the application can access through AssetManager.

Each Android application must have a manifest file, which presents essential information about the application. Our proposed method in this paper is based on the specific analysis of the Android manifest file and is effective for detecting known and unknown malware. Moreover, the cost of this method is very low because this method only analyzes the manifest file. The manifest file is very small compared to the size of the file like Resource or Smali. This method uses reverse engineering to extract information from the application's manifest file. In this study, we will focus on learning how to detect malicious apps on Android based on the manifest file characteristics.

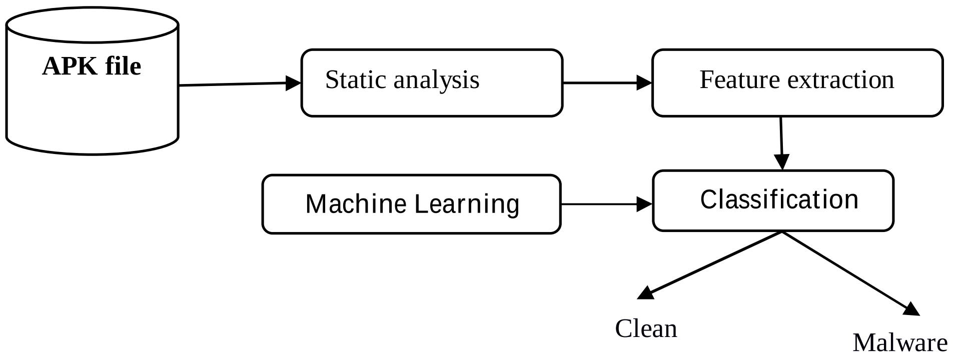

Figure 3: Model of an Android malicious app detection method using machine learning

4. MODEL OF DETECTING MALICIOUS ON ANDROID USING MACHINE LEARNING

4.1 Detection model

Figure 3 depicts a model of the malware detection method on the android application using the machine learning algorithm proposed in the article. Where:

- The input file is an application file in the form of APK. It will be analysed by tools to reverse. The static analysis process will extract specific features contained in the original file such as resource files, manifest files, smalifiles, Java source code, etc. These results are used for data feature extraction module.
- The data feature extraction module selects data and builds it into a feature vector including the permissions described in 4.2.
- Data classification: After constructing the feature vector, this feature vector will be used for a malicious analysis module using the machine learning algorithm. The result of the system is an assessment of the file's malicious level.

In this paper, we will perform malicious detection in the Android application based on the manifest file and the details of the Java source file, the required permissions, the file's hash value.

4.2 Select and extract feature

Permission: Selecting permissions is essential in identifying and classifying malware. As we know, each Android application needs permissions to be able to access data on the device. Android has a total of 324 permissions [7] which is divided into Normal Permission and Dangerous Permisson. When Normal Permission is required, it will be automatically licensed by the system without asking the user. On the contrary, with Dangerous Permission, when the application needs to use it, the system will ask the user whether to license this permission or not. However, based on the characteristics of Normal Permission that the system does not ask the user about licensing for it, attackers can exploit it to steal information. Therefore, this research will take all permissions contained in that .apk file.

API_Calls: According to [8], the authors analyzed and extracted features related to the behavior of malware via API and evaluated different classifications using the self-created feature set. Thereby, the results are up to 99% accurate and 2.2% error rate after using the KNN classification algorithm. After analyzing based on API levels, the team proposed features that malware regularly used and classified APIs by resources. Document [9] also listed the APIs which are often called during analysis in CuckooDroid.

Strings: In Android applications, unique strings are used to define a specific structure for Android to specify the files and IP addresses that an application uses and to list the classes and methods that are called through [8]. Therefore, the IPs will be selected as the feature to that .apk file. In addition, the object string also includes intents, which may or may not be in the "intent" object, so they will also be selected to avoid missing in the selection process.

Intent: There are 2 types of Intent: Explicit Intent and Implicit Intent. When the developer knows exactly which component uses which action, Explicit Intent will be used [10]. Explicit Intent is used for Intra-application and Inter-application. The developers use this type of Intent to navigate from one activity to other activities in the application, like transferring messages between two applications. For example, developers use Explicit Intent to request Android to open a link and use Google Chrome. On the other hand, the developers use Implicit Intent and request Android to open a link but don't specify an application. The intent has 3 components: Event, Classification, and Data. The Event section describes the type of action handled by Intents such as MAIN, CALL, BATTERY LOW, SCREEN ON, and EDIT. The Classification section consists of LAUNCHER, BROWABLE, and GADGET. The Data section provides the necessary data for the application. For example, the CALL event requires a phone number, and the EDIT event needs a document or an HTTP URL to complete the event. Thereby, we can see that the Intent is also a feature because it can be exploited to steal user information through the Event, Classifications, and Data components. In the intent object that was extracted as described in the previous section, all of them will be selected because the intent isn't classified as malicious or normal intent and all intents can be exploited for the bad purposes of malware.

Activity: As presented, the activity includes intents. If the intent is used in the malicious application, it means that the activity is also used. According to the criteria that select all the intents included in that application, all the activities will also be selected as features.

4.3 Classification algorithm

In this paper, we use the Convolutional Neural Network (CNN) model to classify normal and malicious applications. CNN is a Deep Neural Network Architecture and a type of Artificial Neural Network, a Multiplayer Perceptron but bringing some improvements which are Convolution and Pooling. The operating principle of CNN is as follows [11]:

Feature:

- CNN compares the images in pieces, the pieces which it finds called features.
- Each feature is like a mini image, a small two-dimensional array.
- Features match the general aspects of the images.
- For example with image X, the features include diagonals and crosses which will capture most of the important characteristics of most image X and these features will match the edges and the middle center of any image X.

Convolution:

- Convolution consists of two other concepts: Convolution Filter and Convolutional Layer.
- Convolutional Layer is also a hidden layer. Especially, it is a set of feature maps. Each of these feature maps is a scan of the original input, meaning the result of extracting to specific features. After scanning, they are put into the Convolution Filter or Kernel.
- This is a matrix that will scan the input data matrix, from left to right, top to bottom, and multiply each value of the input matrix and kernel matrix respectively and then sum it up, put it into activation function (such as sigmoid, relu, elu, etc.). The results are specific numbers. The set of these numbers is a matrix, which is the feature map.

Pooling: The purpose of Pooling is to reduce the number of hyperparameters that need to be calculated, thereby reducing computation time and avoiding overfitting.

- Hyperparameter is a special type of parameter (everything of the model is used to calculate the output). Hyperparameter is a conventional and relative concept. It is usually a slightly default parameter. For polynomial functions, the degree of polynomials can be considered as a hyperparameter.
- The most common type of pooling is max pooling which is the largest value in a pooling window.
- Pooling works similarly to Convolution, it also has a sliding window called a pooling window. This window slides through each value of the input data matrix 1 (is usually the feature map in Convolutional Layer), pick out a value from the values in the sliding window (with max pooling, we will get the maximum value).

Rectified Linear Units: Keeping values unbroken by adjusting each value a little. Relu's algorithm will change the negative values to 0.

Fully Connected Cayers: Fully connected layers will take the filtered images at a high level and convert them into votes. Fully connected layers is a major block in traditional neuron networks. Instead of being treated as a two-dimensional array, they are treated as a simple list and are all handled the same way. Each voting value (probability of falling into each class) is representative of an existing image.

5. EXPRIMENTS AND EVALUATE

5.1 Hardware requirements

- Ram 4GB.
- Hard drive capacity 8 GB.

5.2 Hardware requirements Software requirements

- Operating system: Windows 10.
- Python programming language version 2.7.
- Programming library: Scikit-learning [12].
- MobSF open-source application [6].
- Oracle JDK 1.7.
- Androguard

5.3 Experimental results

Testing with a set of 700 samples including 199 clean samples and 501 malicious samples on 7-layer, 6-layer, 5-layer, and 4-layer models, we have evaluation parameters.

- 7-Layer Model: consist of 7 convolutional layer, 7 max pooling layer, and 2 fully connected layer.
- 6-Layer Model: consist of 6 convolutional layer, 6 max pooling layer, and 2 fully connected layer.
- 5-Layer Model: consist of 5 convolutional layer, 5 max pooling layer, and 2 fully connected layer.
- 4-Layer Model: consist of 4 convolutional layer, 4 max pooling layer, and 2 fully connected layer.

Table 1: Experimental results

Model	7-Layer	6-Layer	5-Layer	4-Layer
TP	501	501	500	495
FP	0	0	0	0
TN	199	199	199	199
FN	0	0	1	6
Precision	1.0	1.0	1.0	1.0
Recall	1.0	1.0	0.998	0.988
F1_score	1.0	1.0	0.999	0.9939
Acc(%)	96.51	96.42	96.00	91.21

Looking at the above statistics table, we can see that 7-layer and 6-layer models have high accuracy (above 96%), do not miss positive samples, and have high F1_score (achieved absolute value). With the 5-layer and 4-layer models, we see that the model misses some positive samples. This can cause harm when we classify in reality because we can mistake the malicious sample into a clean sample. The number of samples which were mistaken is more with the 4-layer model. Comparing the 6-layer model with the 7-layer model, we can see that both models have good results but the 6-layer model will be simpler than the 7-layer model. This makes the 6-layer model will be more outstanding than the 7-layer model.

The following is a detailed accuracy statistics table on train, val and test sets of 6-layer model(table 2):

Table 2: The accuracy of 6-layer model

The number of samples	including 3000 clean codes and 4000 malicious codes
The ratio of Train/Test/Validation	80/10/10
Train_acc	99.60%
Val_acc	96.00%
Test_acc	96.42%

6. CONCLUSION

In this paper, we have proposed a system model to detect malicious applications on the Android operating system based on static analysis techniques and machine learning algorithms. The experimental results in the paper have shown the approach that based on static analysis to extract rules and CNN algorithms to detect abnormal behaviors is right and reasonable for the early detection of malicious applications. The science of our paper is not only expressed in the use of machine learning algorithms for unique application detection but also proposed the use of properties that are not too complex in terms of calculation and extraction but still bring highly effective in detecting malicious application abnormal behavior. The science of our paper expresses not only in the use of machine learning algorithms to detect malicious code, but only in the proposal to use features which aren't too complicated to calculate and extract but still being highly effective in detecting abnormal behavior of a malicious application. In subsequent studies, we will conduct research and use some new machine learning algorithms in combination with dynamic analysis to obtain faster and more accurate results.

REFERENCES

[1] Yajin Zhou, Xuxian Jiang. **Dissecting Android Malware: Characterization and Evolution**, Proceedings of the 33rd IEEE Symposium on Security and Privacy, San Francisco, CA, May 2012.

[2] McAfee Mobile Threat Report. https://www.mcafee.com/enterprise/en-us/assets/reports/rp-mobile-threat-report-2019.pdf. [access date 1/4/2020]

[3] Tiwari, Suman. (2019). **An Android Malware Detection Technique Based on Optimized Permissions and API**. 10.1109/ICIRCA.2018.8597225..

[4] Enck W.; McDaniel P.; Ongtang M. **On Lightweight Mobile Phone Application Certification**. Proceedings of the 16th ACM conference on Computer and communications security. November 2009 Pages 235–245. https://doi.org/10.1145/1653662.1653691

[5] D. Wu, C. Mao, T. Wei, H. Lee and K. Wu. **DroidMat: Android Malware Detection through Manifest and API Calls Tracing**, 2012 Seventh Asia Joint Conference on Information Security, Tokyo, 2012, pp. 62-69, doi: 10.1109/AsiaJCIS.2012.18.

[6] Mobile Security Framework: https://github.com/MobSF/Mobile-Security-Framework-MobSF [access date 1/4/2020]

[7] Dataset android malware permission: https://www.kaggle.com/xwolf12/datasetandroidpermissions. [access date 1/4/2020]

[8] Feizollah, Ali & Anuar, Nor & Salleh, Rosli & Suarez-Tangil, Guillermo & Furnell, Steven. (2017). **AndroDialysis: Analysis of Android Intent Effectiveness in Malware Detection**. Computers & Security. 65. 121-134. 10.1016/j.cose.2016.11.007.

[9] Faruki, Parvez & Ganmoor, Vijay & Laxmi, Vijay & Gaur, Manoj & Bharmal, Ammar. (2013). **AndroSimilar: robust statistical feature signature for Android malware detection**. SIN 2013 - Proceedings of the 6th International Conference on Security of Information and Networks. 152-159. 10.1145/2523514.2523539.

[10] Shabtai, A., Kanonov, U., Elovici, Y. et al. **“Andromaly”: a behavioral malware detection framework for android devices**. J Intell Inf Syst 38, 161–190 (2012). https://doi.org/10.1007/s10844-010-0148-x.

[11] Krizhevsky, Alex & Sutskever, Ilya & Hinton, Geoffrey. (2012). ImageNet Classification with Deep Convolutional Neural Networks. Neural Information Processing Systems. 25. 10.1145/3065386.

[12] Scikit-learn Machine Learning in Python: http://scikit-learn.org/ [access date 1/4/2020]