International Journal of Innovative Technology and Exploring Engineering (IJITEE)
ISSN: 2278-3075 (Online), Volume-9 Issue-6, April 2020

Detecting Malicious Applications on Android is based on Static Analysis using Machine Learning Algorithm

Tisenko Victor Nikolaevich, Lai Van Duong, Ha Tuan Anh, Nguyen Quang Dam, Nguyen Quoc Hoang

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 cyberattackers. 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.

I. 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.

Revised Manuscript Received on April 30, 2020.
***** Correspondence Author
Tisenko Victor Nikolaevich*, Department Quality Systems, Peter the Great St. Petersburg Polytechnic University, Russia. E-mail: v_tisenko@mail.ru;
Lai Van Duong, Faculty of Information Technology, FPT University, Hanoi, Vietnam. E-mail: duonglvse05009@fpt.edu.vn,
Ha Tuan Anh, Faculty of Information Technology, FPT University, Hanoi, Vietnam. E-mail: anhhthe130399@fpt.edu.vn,
Nguyen Quang Dam, Faculty of Information Technology, FPT University, Hanoi, Vietnam. E-mail: damnqse05820@fpt.edu.vn,
Nguyen Quoc Hoang, Faculty of Information Technology, FPT University, Hanoi, Vietnam. E-mail: hoangnqse06012@fpt.edu.vn

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.

II- 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.

III- 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:

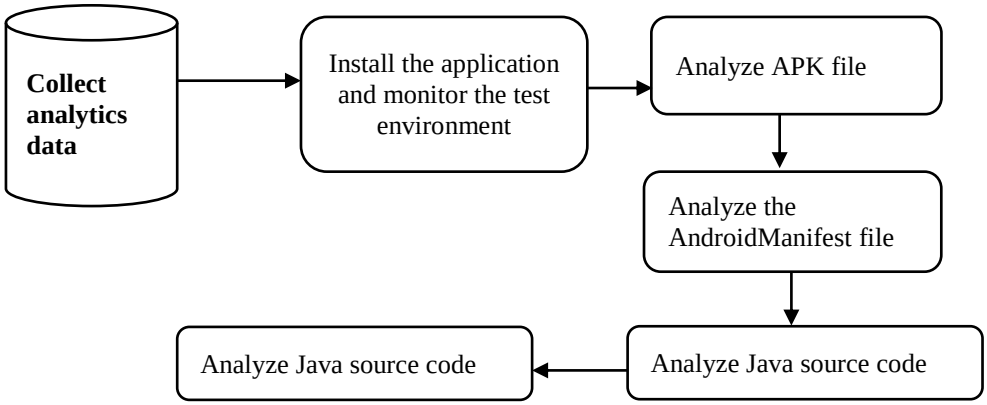

Fig. 1. Static analysis model

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...

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.

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

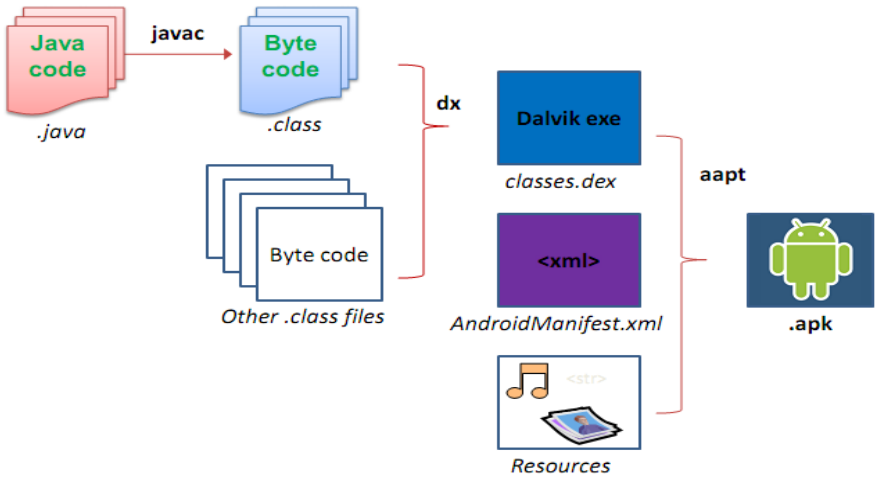

Fig. 2. Components of the application file

International Journal of Innovative Technology and Exploring Engineering (IJITEE)
ISSN: 2278-3075 (Online), Volume-9 Issue-6, April 2020

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...

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, ... 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.

IV- MODEL OF DETECTING MALICIOUS ON ANDROID USING MACHINE LEARNING

4.1. Detection model

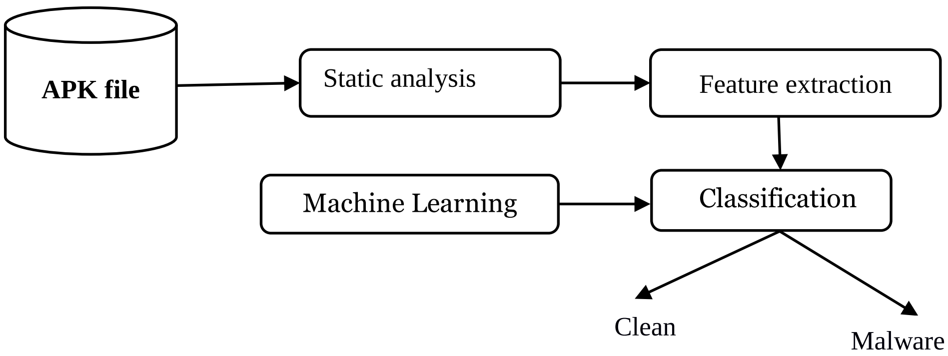

Fig. 3. Model of an Android malicious app detection method using machine learning

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 article, 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

4.2.1. Data features

The training data set used in the paper [7] includes 2000 data records including 1000 data from malicious applications and 1000 data from benign applications.

Malicious applications collected on the internet and rated malicious by Virustotal and malware scanners.

The benign applications are downloaded directly on Google Play.

For each application in the android application dataset, the data is extracted as a binary string $R_i = \{r_1, r_2, \ldots, r_n\}$ representing 254 permissions for the android application as shown in figure 4 and figure 5, where:

$$r_i = \begin{cases} 0, \mathit{If\ no\ rights\ exist} \\ 1, \mathit{If\ exists\ right} \end{cases}$$

Thus the permission value is expressed in binary form with the values 0 and 1.

The final value in the vector is the value that describes the object class:

$$r_n = \begin{cases} 0, \mathit{If\ the\ application\ is\ benign} \\ 1, \mathit{If\ the\ application\ is\ malicious} \end{cases}$$

0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,1,0,1,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,1,0,0,0,0,0,0,1,0,
1,0,1,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1

Fig. 4. Example record of rights against malicious applications

0,1,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0

Fig. 5. Example record of rights to the benign application

4.2.2. Optimize feature

Each of the permissions vectors collected above includes 254 permissions, many of which are redundant when not used or used very little in Android applications. These redundant permissions will be removed because they will likely make the probability of classification more misleading. The goal of data feature optimization is, therefore, to eliminate redundant information and retain beneficial information with the value that is equivalent to the original vector. In order to optimize the number of feature, in this paper, we use information gain calculation method (Information Gain) [8].

Information gain of each permission is calculated as follows [8]:

$$InfoGain(C, r_j) = entropy(C) - entropy(C|r_j)$$

Where $entropy(C)$ is the entropy amount of information. Where A and B are 2 random values, P is the probability, entropy is calculated as the following formula:

$$entropy(A) = \sum_i P(a_i)I(a_i)$$

Probability $entropy(A|B)$:

$$entropy(A|B) = \sum_{i,j} P(a_i, b_j)\log\frac{P(b_j)}{P(a_i, b_j)}$$

With the above scoring method, applying formula 1 to the calculation and using the data set [7] used in the paper, we have selected 20 permissions with the best information gain from the original data set. List of 20 permissions is described in table 1.

Table 1. 20 permissions used to classify malicious code.

No.	Permission	Scores
1	READ_PHONE_STATE	0.69
2	SEND_SMS	0.42
3	READ_SMS	0.16
4	READ_HISTORY_BOOKMARKS	0.16
5	INSTALL_SHORTCUT	0.09
6	INSTALL_PACKAGES	0.08
7	RECEIVE_BOOT_COMPLETED	0.07
8	SYSTEM_ALERT_WINDOW	0.07
9	WRITE_HISTORY_BOOKMARKS	0.37
10	ACCESS_NETWORK_STATE	0.41
11	WRITE_SETTINGS	0.15
12	RECORD_AUDIO	0.09
13	CALL_PHONE	0.06
14	CAMERA	0.39
15	WAKE_LOCK	0.03
16	GET_ACCOUNTS	0.15
17	RECEIVE	0.31
18	READ_EXTERNAL_STORAGE	0.39
19	MODIFY_AUDIO_SETTINGS	0.01
20	MOUNT_UNMOUNT_FILESYSTEMS	0.13

The sample result for the permission vector consists of 20 data features after the information gain step according to Table 1.

Benign applications: 0,0,0,1,0,0,0,0,0,0,0,0,1,0,0,1,0,0,0,0,0

Malicious applications:
0,0,0,1,1,0,0,0,0,0,1,0,1,0,0,0,0,0,1,0,1

4.3. Classification algorithm

In the field of detection of malicious applications on android can use machine learning algorithms such as [8]: Decision trees, Naive Bayes, Support Vector Machine (SVM)... In this article, we will apply SVM algorithm to detect malware on android apps. SVM is the supervised learning algorithm used for data classification [9]. SVM algorithm constructs a hyperplane or a set of hyperplanes in a multi-dimensional or infinite-dimensional space, which can be used for classification, regression, or other tasks. Intuitively, for the best classification, the hyperplane is as far away from the data points of all classes (called the margin function), because the larger the margin the lower the generalization error of the classification algorithm [10]. The SVM method requires data to be expressed as vectors of real numbers. Thus, if the input is not a number, it is necessary to find a way to convert them into the numeric format of SVM. The main steps of the SVM method include the following steps [9]:

- Preprocessing Data: Perform data transformation suitable for the calculation process, avoiding too large numbers to describe features. It is often advisable to scaling data to convert to [-1, 1] or [0, 1].
- Select the kernel function: Select the appropriate kernel function for each specific problem to achieve high accuracy in the classification process.
- Selecting data features: The selection of appropriate data features and the elimination of redundant features will help optimize implementation costs and training costs.
- Training with a sample set. In the training process, an algorithm will be used to optimize the distance between hyperplanes during the classification process, determine the classification function in the specific space by mapping data into the specific space by describing, the kernel, solve for both data is linear and non-linear in the characteristic space.
- Evaluate the correct probability of the algorithm.

V- EXPRIMENTS AND EVALUATE

5.1. Hardware requirements

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

5.2. Software requirements

- Operating system: Windows 10.
- Python programming language version 2.7.
- Programming library: Scikit-learning [11].
- MobSF open-source application [6].
- Oracle JDK 1.7.

5.3. Experimental results

In fact, in addition to machine learning algorithms, the performance of a model may depend on other factors such as the distribution of classes, the cost of misclassification, the size of training and test sets, performance measures. The problem is how to evaluate the accuracy of the algorithm and choose the algorithm that suits the problem requirements. In this paper, we will use the Cross-validation method to evaluate the accuracy of the SVM algorithm used [9]. In this paper, we evaluate the result with K = 10 and give accuracy is 92.88% as shown in figure 6.

```
[+] Performing 10-fold cross validation....
[fold 1]  score: 0.93035
[fold 2]  score: 0.92537
[fold 3]  score: 0.94527
[fold 4]  score: 0.90547
[fold 5]  score: 0.97015
[fold 6]  score: 0.94000
[fold 7]  score: 0.90000
[fold 8]  score: 0.90500
[fold 9]  score: 0.94500
[fold 10]  score: 0.92500
[+] Accuracy: 92.9161691542%
```

Fig. 6. Evaluating the accuracy of the SVM algorithm using K-fold Cross-validation method

After using the training dataset to train the model, the paper uses a test dataset of 100 benign applications and 100 malicious applications to check the results of the algorithm. Test results give relatively high accuracy: For 100 malicious applications give 95% probability and 100 benign applications give 97% correct probability as shown in figure 7.

```
[+] Testing for 100 goodware ...
=> Correct : 97.0%
[+] Testing for 100 malware ...
=> Correct : 95.0%
```

Fig. 7: Analysis results of 100 malicious and 100 benign applications

VI CONCLUSIONS

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 a static analysis-based approach to extract SVM's laws and 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. Y. Zhou and X. Jiang, "Dissecting Android Malware: Characterization and Evolution," *2012 IEEE Symposium on Security and Privacy*, San Francisco, CA, 2012, pp. 95-109.
2. McAfee Mobile Threat Report. https://www.mcafee.com/enterprise/en-us/assets/reports/rp-mobile-threat-report-2019.pdf. [Last accessed 26 February 2020]
3. Sato, Ryo & Chiba, Daiki & Goto, Shigeki. (2013). Detecting Android Malware by Analyzing Manifest Files. Proceedings of the Asia-Pacific Advanced Network. 36. 23. 10.7125/APAN.36.4.
4. William Enck, Machigar Ongtang, and Patrick McDaniel. On Lightweight Mobile Phone Application Certification. Proceedings of the 16th ACM conference on Computer and communications securityNovember 2009. pp 235–245.
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.
6. Mobile Security Framework: https://github.com/MobSF/Mobile-Security-Framework-MobSF. [Last accessed 26 February 2020]
7. Dataset android malware permission: https://www.kaggle.com/xwolf12/datasetandroidpermissions. [Last accessed 26 February 2020].
8. Smola, A.; Vishwanathan, S.V.N. Introduction to Machine Learning; Cambridge University Press: Cambridge, UK, 2008.
9. C.J.C. Burges Chris J.C. Burges. A Tutorial on Support Vector Machines for Pattern Recognition. Data Mining and Knowledge Discovery. 1998, 2, 121-167.
10. JohnShawe-Taylor, ShiliangSun. Kernel Methods and Support Vector Machines. Academic Press Library in Signal Processing. 2014, 1, 857-881.
11. Scikit-learn Machine Learning in Python : http://scikit-learn.org/. [Last accessed 26 February 2020]

AUTHOR PROFILE

First Author My position is the professor of Institute of computer sciences and technologies in Peter the Great Saint-Petersburg Polytechnic University. I have received the degree Doctor of Technical Sciences in 1998 in accordance of scientific speciality "Systems of automatic Design" in SPbPY. The area of scientific interest is use of new type of fuzzy logics in different applications. I think that we could cooperate intensively in future.
Email: ***v_tisenko@mail.ru***

Second Author: Authors Lai Van Duong, Ha Tuan Anh, Nguyen Quang Dam, Nguyen Quoc Hoang are fourth-year students majoring in information security at FPT University. These students have over 2 years of experience working with Android attack detection issues.
Email:duonglvse05009@fpt.edu.vn, anhhthe130399@fpt.edu.vn, damnqse05820@fpt.edu.vn, hoangnqse06012@fpt.edu.vn..