A Survey on Malicious Code Detection for Android using Instruction Signatures

Bhavesh Patadiya[1] Prof. Ashil Patel[2]
[1]M.E Research Scholar [2]Associate Professor
[1,2]Department of Information Technology
[1,2]L. D. College of Engineering, Ahmedabad

Abstract— as android operating system growing rapidly and one of the most popular platform, because of its vulnerable security system architecture, Android has been the most targeted platform for malware attacks. In this paper, we have explored various malware detection techniques. After understanding various research paper described below; we found that with using instruction based scheme for malware detection we can efficiently detect malicious apps. However, Existing system can be enhanced by providing additional permission based filtering scheme. The findings of the review have been well documented in this paper.

Key words: Android, Malware, Malware Detection, Instructions Signature

I. INTRODUCTION

Malware, short for malicious software, is any software used to disrupt computer operation, gather sensitive information, or gain access to private computer systems. Examples of this Malware include viruses, worms, and Trojan horses which is very popular now a days.

According to a survey[1], 99% mobile malicious programs target the Android platform in last few years, and very small amount targeting Java and Symbian-based Devices. Malware is a worldwide popular now a days and as everyone knows how malware's can be disastrous, the detection of malware app is an area of major concern for the research community and also for each people[2]. Researchers and Scholars developed algorithms and techniques for malware app detection and these techniques are realized through the implementation of malware detectors.

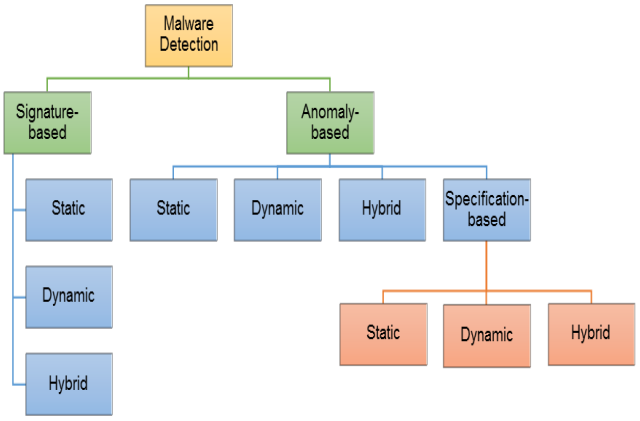

Fig. 1: Types of Malware Detection Techniques

A Malware detector used to take defends against malware, and the efficiency of a malware detector is determined by the techniques it uses. Now we will study malware detection techniques and understand their strengths and limitations.

The rest of this paper is organized as follows:In Section 2, we have explored various malware detection techniques and explored their advantages and disadvantages. In Section 3 to 5, we have analysed instruction signature-based schemes to detect Android malware. In Section 6, we conclude our work

Category	Features Analyzed	Advantages	Disadvantages
Technique			
Signature Based Detection	– This technique uses its characterization of what is known to be malicious to decide if the program under inspection is malware.	– Detect known attacks effectively – Using less computational resources	– Less effective to unknown or new malware
Anomaly Based Detection	– This technique uses its knowledge of what constitutes normal behavior to decide if the program under inspection is malware.	– Effectively detect unknown malware	– Require more computational resources – Produce some amount of false alert
Specification Based Detection	– It uses some specification or rule set of what is valid behavior in order to decide if the program under inspection is malware. Programs violating the specification are considered	– Effectively detect unknown malware	– Deriving detail specification for SPB is time consuming – Produce some false negatives

All rights reserved by www.ijsrd.com

	anomalous and usually, malicious.		
Analysis			
Static	– It uses syntax or structural properties of the program under inspection (PUI) to determine its maliciousness.	– Quickly detect malware	– Can prevent malware from installing – Can be evaded through obfuscation and encryption technique
Dynamic	– It uses runtime information (e.g. systems seen on the runtime stack) of the PUI (Process under Inspection).	– Obfuscation and encryption technique will not affect the detection	– Produce high generation of captured – Consume the storage space – Requires high computational powers to process the data – Times required to observe the appearance of the malicious activity cannot be define clearly

Table 1: Classification of Malware Detection Techniques and Analysis

II. ANALYSING ANDROID APPS WITH ITS PERMISSION-BASED SCHEMES

Permission-based scheme uses a computer security technique that allows the admin and operating system to restrict apps to use specific assets of application[3]. The restrictions used in this model are not designed by the users, but instead are designed by developers or admins. Therefore, the user's system security is dependent on the user's belief that the developers and administrators are legitimate, which can be easily exploited by malicious developers. Table 2 shows the most dangerous permission combinations used by malware apps.

Permissions (a.p.)	Potential Operation
(READ/MODIFY)_PHONE_STATE RECORD_AUDIO INTERNET	Eavesdropping or broadcasting voice calls.
PROCESS_OUTGOING_CALL RECORD_AUDIO INTERNET	Eavesdropping or broadcasting voice calls.
ACCESS_(COARSE/FINE)LOCATION INTERNET RECEIVE_BOOT_COMPLETE	Tracking user location.
RECEIVE_SMS WRITE_SMS	Malwares that are embedded in or controlled via SMS can be hidden from the user by automatic editing.
SEND_SMS WRITE_SMS	Mobile bots that SMS spam without being traced by the user.
INSTALL_SHORTCUT UNINSTALL_SHORTCUT	Converting shortcuts to legitimate applications to access malicious applications

Table 2: Dangerous Permission Combination and Their Potential Operations

By doing high quality testing analysis based on the network visualizations, this app able to verify several aspects of permission-based security models of APKs that had been observed and addressed in past research[4][5]. Also, by taking the relative frequency of the permission requests into consideration, we have discovered that APKs that belong to certain categories, such as Communication category, have more chance of causing a false positive detection when in a search of malwares. Although certain permission combinations (like a.p.INTERNET, a.p. RECEIVE_BOOT_COMPLETED, a.p.ACCESS_COARSE_LOCATION,anda.p.ACCESS_FINE_LOCATION) are also listed in Table 2, it seems to be unfair to conclude these applications to be dangerous, since many of them may legitimately require such set of permissions in order to offer the users with their services. This suggests that a deterministic analysis based on permission requests fails to provide users with credible alerts.

Category	Number	Potential Malware
Communication	102	30 (29.4%)
Entertainment	196	14 (7.1%)
Finance	50	4 (8.0%)
Health	75	2 (2.7%)
Multimedia	76	6 (7.9%)
Productivity	113	12 (10.6%)
Reference	126	2 (1.6%)
Shopping	45	0 (0.0%)
Sports	50	0 (0.0%)
Travel	65	9 (13.8%)
Games	101	0 (0.0%)
Total	999	79 (7.9%)

Table. 3: Units Statistics of Sample APKs

All rights reserved by www.ijsrd.com

Finally, similar characteristics of APKs' permission requests, researchers able to distinguish some APKs from others based on their fundamental functionality, which had also shown as a important and feasible technique for detecting malware in a nondeterministic approach, as shown in Table 3[7][8]. Although in most cases the distinction between APK groups were not clear, it consider with a hierarchical simplicity, or applying weight on permissions based on its frequency (i.e., common permissions and rare permissions may have more effects to the APK's functionality than other permissions), the grouping of APKs may result in a better classification of services.

III. ANDROID MALWARE DETECTION USING PERMISSION AND API CALLS

The proposed framework, as shown in Figure 2, consists of four major parts[6]. The first component is an App analyzer that decompresses the APK file of an App and extracts AndroidManifest.xml and class files, which are necessary for characterizing Apps. The second component aims to characterize each App based on its requested permissions and API calls. The third component, feature generator, carries out feature extraction, which includes permission features extracted from AndroidManifest and API call features extracted from class files. As a result of this this process, each App is represented as a single instance with binary permission and API call features, and a class label indicates whether the App is a benign or a malware. The last part includes the training of the classification models from the collected data.

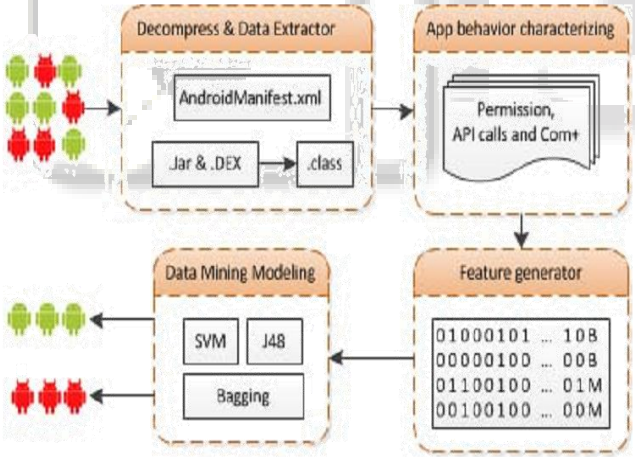

Fig. 2: The Proposed [6] Permission and API Call Feature-Based Malware Detection Framework

Experiments on real world data demonstrate the good performance of the framework for malware detection. The contribution of authors' proposed framework, compared to the existing solutions: (1) Authors have validated that combining permissions and API calls is effective for malware detection; (2) their framework does not involve any dynamical tracing of the Android applications; and (3) their framework can be generalized to all mobile applications for malware detection.

IV. INSTRUCTION SIGNATURES-BASED MALWARE DETECTION

The major contribution of this paper includes the following two parts [9]:

- A method that can improve efficiency. We get the signature from dex file and then get function calls or behavior sequences as signatures with Levenshtein's distance algorithm.
- Completing signature extraction as well as malicious code validation. We carry out the experiment based on the samples of DroidKungFu3, and the result shows that the signatures extracted by the method are representative.

Detection scheme based on signature code by detecting whether the file has the characteristics of known malwares (some special code or string) to determine whether it is malicious software. Behavior based analysis method is divided into dynamic analysis method and static analysis method. Dynamic analysis method is the use of a sandbox, a virtual machine to simulate the operation of a program, by way of monitoring or intercepting, analyzing the behavior of program running. Static analysis method, is the means by reverse extracting signature program, analyze the sequence of function call and compare them. Signature based detection method is based on the extracted signature from the malicious code which detected, compared with heuristic based detection methods, with high efficiency low false alarm rate, it is widely used among malicious code detection rules.

Based on analysis of executable dex files we can get key method and their code, which can be regarded as features, and thus to analyze the software, including the following steps:

A. Signature Extraction:

At first, malware samples are decompiled, and we get the source code. Then, we get malicious code which can achieve the malicious behaviour by analyzing the source code. Finally, the corresponding binary sequences will be the initial signatures.

B. Signature Optimization:

Get the best signatures by comparing with the same type of malicious binary sequences.

1) Signature Extraction:

The malicious code based on instruction signature extraction process will be done after obtaining malware samples, need to reverse engineered its source code, and then getting the corresponding obfuscate source code. After getting malicious code from the decompiled source code, write down the corresponding function and method in one of the editors. Using 010 editor can show the sequence of function and the function of each method, according to the sequence we can get the corresponding malicious binary byte.

2) Signature Analysis Algorithm:

In order to determine whether a certain piece of code is similar to malicious code, we need to analyze whether the corresponding binary sequence is similar to extracted signatures, so the essentiality is to determine whether two binary sequence is similar. However, comparing two binary byte code will consume too much resource, so it can be attributed to the final judgment to determine the similarity of two strings. In this paper, Levenshtein distance is used to compare two strings.

All rights reserved by www.ijsrd.com

Levenshtein distance between two strings means the minimum number of editing operations required to transfer a string to another string. Algorithm is as follows:

So decided to choose the Levenshtein distance algorithm. Levenshtein distance between two strings means the minimum number of editing operations required to transfer a string to another string.

3) Levenshtein Algorithm:

1) If the length of str1 or str2 is 0, return the other string's length.
2) if(str1.length=0) return str2.length;
3) if(str2.length=0) return str1.length;
4) Initialize the matrix d with the size of (n +1) * (m+1), and make the first row and column' values grow from 0.
5) Scan two strings (n * m level), if str1[i] == str2[j], use the temp to record it as 0. Otherwise record it as 1. Then assign the matrix d[i, j] with the minimum of d[i-1, j] +1, d[i, j-1] +1, d[i-1, j-1] + temp.
6) After scanning, return the last value of matrix d[n][m], which is the distance between them.

The similarity between two strings is calculated as follows:

$$\text{sim} = \frac{\text{d[n][m]}}{(\text{length(str1)} + \text{length(str2)}}$$

Where,
d[n][m] is the Levenshtein distance between two strings,
length(str1) is the length of the string str1, as well as
length (str2) is the length of the string str2.

4) Signature Selection:

– The signatures extracted by signature extraction may not meet the needs of detection very well, so we should explicit the effectiveness and scope of these signatures, from which ultimately extract the best malicious code signatures.
– Analysing the same class malware whose number is N, then we can get similar preliminary signatures whose number is also N, which make up a preliminary signature library. Selecting the same type software make up the sample library A whose number is M. Each sample of A compares with signature library, we can get the similarity Si , then taking the average similarity value Di , the formula is as follows:

$$D_i = \frac{1}{M}\sum_{i=1}^{M} S_i$$

Putting the best signature whose the last signature library Di is maximum into the last signature library.

5) Existing System:

Existing system model consists of three parts which are signature extraction, signature optimization and detection as described above. System model shows in Fig. 3. We get the source code by decompiling malicious software which type is known, then analyze the source code, and get the preliminary signatures which is optimized for the sake of the final signature. Comparing test data and the signature.

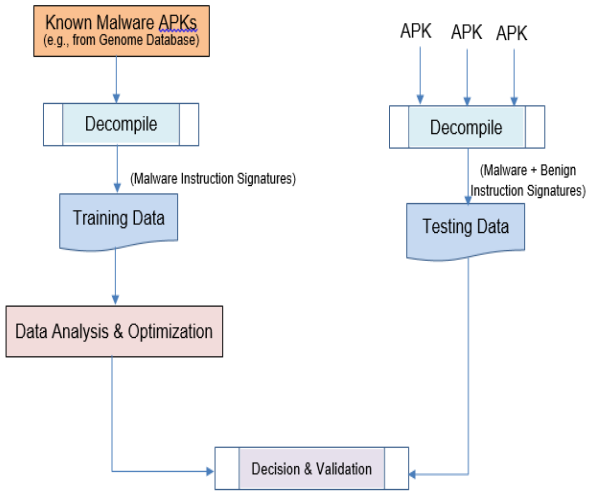

Fig. 3: System Flow

Library, we can make the final decision and validation.

V. CONCLUSION

This paper proposed a method for Android malicious code analysis using instruction sequences. Through manually analyse formal description of the program to get DEX file and obtain signatures, then get the best signatures through optimization algorithm. Through experiments on our sample system, it verify the signatures extracted has a good output and this detection method based on the signatures have a good recognition as well.

REFERENCES

[1] "99% of all mobile threats target Android devices" http://www.kaspersky.com/about/news/virus/2013/99_of_all_mobile_threats_target_Android_devices
[2] Nwokedi Idika, Aditya P. Mathur – "A Survey of Malware Detection Techniques" http://cyberunited.com/wp content/uploads/2013/03/A-Survey-of-Malware-Detection-Techniques.pdf
[3] Ittipon Rassameeroj and Yuzuru Tanahashi – "Various Approaches in Analyzing Android Applications with its Permission-Based Security Models", 2011 IEEE International Conference On Electro/Information Technology
[4] D. Barrera, H. G. u. c. Kayacik, P. C. van Oorschot, and A. Somayaji - "A methodology for empirical analysis of permission-based security models and its application to android,", Proceedings of the 17th ACM Conference on Computer and Communication Security, CCS'10, New York, NY, U.S.A, 2010.
[5] W. Enck, M. Ongtang, and P. McDaniel, "On lightweight mobile phone application certification," In Proceedings of the 16th ACM Conference on Computer and Communications Security, CCS '09, New York, NY, U.S.A., 2009.
[6] Naser Peiravian and Xingquan Zhu – "Machine Learning for Android Malware Detection Using Permission and API Calls", 2013 IEEE 25th International Conference on Tools with Artificial Intelligence

All rights reserved by www.ijsrd.com

[7] Desnos A. Android: Static Analysis Using Similarity Distance[C]. Maui, HI: 2012.
[8] Gostev A. Mobile malware evolution: An overview[J]. Viruslist.com, 2006.
[9] HuGe, LiTing, DongHang, YuHewei, ZhangMiao – "Malicious code detection for Android using Instruction Signatures", 2014 IEEE 8th International Symposium on Service Oriented System Engineering

All rights reserved by *www.ijsrd.com*