ISSN 2395-1621

# Malware Application Detection System for Android

[#1]Omkar Bhor, [#2]Dhananjay Gandole, [#3]Vikram Ghule, [#4]Mahesh Garje

[1]cuteomny5@gmail.com
[2]dhananjay.gandole1@gmail.com
[3]vickyghule14@gmail.com
[4]mahesh.garje@gmail.com

[#1234]Department of Computer Engineering
JSPM's
Imperial College of Engineering & Research
Wagholi, Pune- 412207

**ABSTRACT**

**Android applications are becoming increasingly because android phones are Widespread and steadily gaining popularity. Unfortunately, such phenomenon draws attention to malicious application developers so that malicious applications are increasing rapidly. These malicious applications are capable to stealing sensitive and private user information and to damage the phone hardware as well. Therefore it is highly essential to identify and detect these malicious applications which is present in both official Android Market and alternative application markets. We extract two features for the Android applications- Permissions extracted from AndroidManifest.xml file and the Strings extracted from the complied source code File (dex file). Permissions are the kernel access given to each Android application to access core functionalities of the phone hardware. We classify these permissions according to their usage by both genuine and malicious applications. We also extract the const-string labeled strings from the source code of applications. Since, source code analysis is used in detection of desktop malware, we apply the same mechanism to identify and detect Android malware. Machine Learning algorithms are used to train and test our classifier framework. We use 3 supervised learning algorithms for our project- NaiveBayes, J48 ( Weka Implementation of C4.5 algorithm) and NaiveBayes Updateable. We have collected 182 genuine applications and 49 malware applications to train and test our framework. The classifier framework is trained using a combination of all the genuine and malware applications and tested by providing individual samples of the malware applications**.

*Keywords:* **Mobile cloud computing, quality-of-service, bandwidth shifting, bandwidth redistribution**

ARTICLE INFO

**Article History**

Received: 26th October 2015

Received in revised form :

28th October 2015

Accepted: 2nd November, 2015

**Published online :**

**3rd November 2015**

## I. INTRODUCTION

Since Google developed an android platform, android phones have evolved over the years. Smartphone's are becoming increasingly popular. They have evolved into small and portable personal computers that allow users to browse the Internet, send e-mails or connect to a remote desktop. These small computers are able to perform complex computing tasks and communicate through different methods (e .g., GPRS, WiFi or Bluetooth). In the last decade, users of these devices had problems installing mobile applications. They had to download an application from a website and then, install it on the device. In order to protect the device and avoid piracy, several operating systems (e.g., Symbian) employ an authentication system based on certificates that causes several inconveniences for the users (e.g. , they cannot install applications despite having bought them). Nowadays, thanks to the deployment of Internet connections in mobile devices, there are new methods to distribute applications. Users can install any application without even connecting the mobile device to the computer. They only need an account of an application store in order to buy and install new applications. Apple's App Store was the first store to implement this new model and it turned to be successful. Other manufacturers such as Google, RIM and Microsoft have followed the same business model developing application stores accessible from the device. These facts have increased the number of developers for mobile platforms and the number of mobile applications. According to Apple1, the number of available applications on the App Store is over 350,000, while Android Market2 has over 200,000 applications. Regarding the application stores, while for Apple devices the App Store is the single official way to obtain applications, android allows users to

 

install applications that have been downloaded from alternative markets or directly from Internet. According to their response to the US Federal Communication Commission's July 20093, Apple applies a rigorous review process made by at least two reviewers. In contrast, Android relies on its security permission system and on the user's sound judgment. Unfortunately, users usually have no security consciousness and they do not read required permissions before installing an application. Although both App Store and Android Market include clauses in the terms of services that urge developers not to submit malicious software, both have hosted malware in their stores. To solve this problem, they have developed tools for removing remotely these malicious applications. Therefore both models are insufficient to ensure user's safety and new models should been included in order to improve the security of the devices. Machine learning techniques have been applied for classifying applications mainly focused on malware detection. Their goal is to classify applications on 2 main categories: malware or genuine. There are other works that try to classify applications specifying the malware class, e.g., Trojan, worms, virus. With regards to Android applications, there is a lack of malware samples. Anyway, the number of samples is increasing exponentially and several approaches have been proposed to detect Android Malware. Shabtaietal. Trained machine learning models using other feat uses, e.g., parsing apk contained xml and counting xml elements, attributes or namespaces. To evaluate their models, they selected features using three selection methods: Information Gain, Fisher Score and Chi-Square. They obtained 89% of accuracy classifying applications into 2 categories: tools or games. In light of this background, we propose here a new method for classifying Android applications into several categories (e.g., entertainment or productivity) using features extracted both from the Android Market and the application itself. Summarizing, our main findings in this paper are:

1.We describe the process of extracting features from the Android .apk files.

2.We propose a new representation for Android applications in order to develop an automatic categorization approach.

3.We perform an empirical validation of our approach and show that it can achieve high accuracy rates.

## II. RELATED WORK

I.WEKA (Machine Learning):

Weka (Waikato Environment for Knowledge Analysis) is a popular suite of machine learning software written in Java, developed at the University of Waikato, New Zealand. Weka is free software available under the GNU General Public License.
Weka (pronounced to rhyme with Mecca) is a workbench that contains a collection of visualization tools and algorithms for data analysis and predictive modelling, together with graphical user interfaces for easy access to this functionality. The original non-Java version of Weka was a TCL/TK front-end to (mostly third-party) modelling algorithms implemented in other programming languages, plus data pre-processing utilities in C, and a Make file-based system for running machine learning experiments. This original version was primarily designed as a tool for analysing data from agricultural domains, but the more recent fully Java-based version (Weka 3), for which development started in 1997, is now used in many different application areas, in particular for educational purposes and research.

II.J48 (OR):

C4.5 is an algorithm used to gene rate a decision tree developed by Ross Quinlan. C4.5 is an extension of Quinlan's earlier ID3 algorithm. The decision trees generated by C4.5 can be used for classification, and for this reason, C4.5 is often referred to as a statistical classifier. C4.5 builds decision trees from a set of training data in the same way as ID3, using the concept of information entropy. The training data is a set S = {s_1, s_2, ...} of already classified samples. Each sample s_i consists of a p-dimensional vector(x_{1,i}, x_{2,i},...,x_{p,i}),where the x_j represent attributes or features of the sample, as well as the class in which s_i falls. At each node of the tree, C4.5 chooses the attribute of the data that most effectively splits its set of samples into subsets enriched in one class or the other. The splitting criterion is the normalized information gain (difference in entropy). The attribute with the highest normalized information gain is chosen to make the decision. The C4.5 algorithm then recurs on the smaller sub lists. All the samples in the list belong to the same class. When this happens, it simply creates a leaf node for the decision tree saying to choose that class. None of the features provide any information gain. In this case, C4.5 creates a decision node higher up the tree using the expected value of the class. Instance of previously-unseen class encountered. Again, C4.5 creates a decision node higher up the tree using the expected value.

III.NAIVEBAYES:

Test data is data which has been specifically identified for use in tests, typically of a computer program. Some data may be used in a confirmatory way, typically to verify that a given set of input to a give n function produces some expected result. Other data may be used in order to challenge the ability of the program to respond to unusual, extreme, exceptional, or unexpected input. Test data may be produced in a focused or systematic way (as is typically the case in domain testing), or by using other, less-focused approaches (as is typically the case in high-volume randomized automated tests). Test data may be produced by the tester, or by a program or function that aids the tester. Test data may be recorded for re-use, or used once and then forgotten.

 

## III. EXISTING SYSTEM

Consider The existing system uses a framework that validates the permissions required by the applications based on the permissions it classifies whether the app is genuine or malware. Every android application requires Kernel permissions like "READ_SMS", "WRITE_SMS" ,"GET_WIFI_STATE","WRITE_EXTERNAL_STORAGE ","ACCESS_COARSE_LOCATION","ACCESS_FINE_L OCATION","ACTIVITY_RECOGNITION","GET_ACCO UNTS","MANAGE_ACCOUNTS","USE_CREDENTIAL S,"DISABLE_KEYGUARD",ACCESS_WIFI_STATE","A CCESS_NETWORK_STATE"-which are essential for applications to function. The existing system scans for more vulnerable kernel permissions which may steal or damage user information. The existing system modelled a framework to classify permissions as normal permissions and vulnerable permissions. Based on the us age of these vulnerable permission, the system classifies the applications as genuine or malware..

Disadvantages of existing system

- Classification of android apps based on only permissions is not sufficient. Many applications can be still infective by using normal permissions. Therefore detection using app permissions is not very effective. The existing system does not perform source code analysis to understand the behaviour of the application.

- Each Android apk consists of a dex file which comprises the compiled source code. Since the source code revels the working of the application, It is necessary to build a system that includes both the source code and kernel permissions.

## IV. PROPOSED WORK

Our proposed system builds a model that uses both kernel permissions and source code of applications. We analyse "AndroidMainfest.xml" file to obtain the kernel permissions that the application uses. We also decompile the "dex" file which is the compiled version of the source code to extract the strings used in the application. We decrypt the Manifest xml and decompile the dex file to obtain the required data for building the classifier framework. Since string analysis is used in detection of desktop malware, we use this mechanism to detect and identify android malware.

Advantages Of Proposed System

Our proposed system classifies the android apps by analysing both permissions and strings in the apps. Many android apps gets infected by classifying the apps only on the permission because many android apps are still infective with the genuine permission, so classifying the android apps with both strings and permission increase the efficiency of the system. Since string analysis is used to detect desktop malware we used this mechanism in our system.

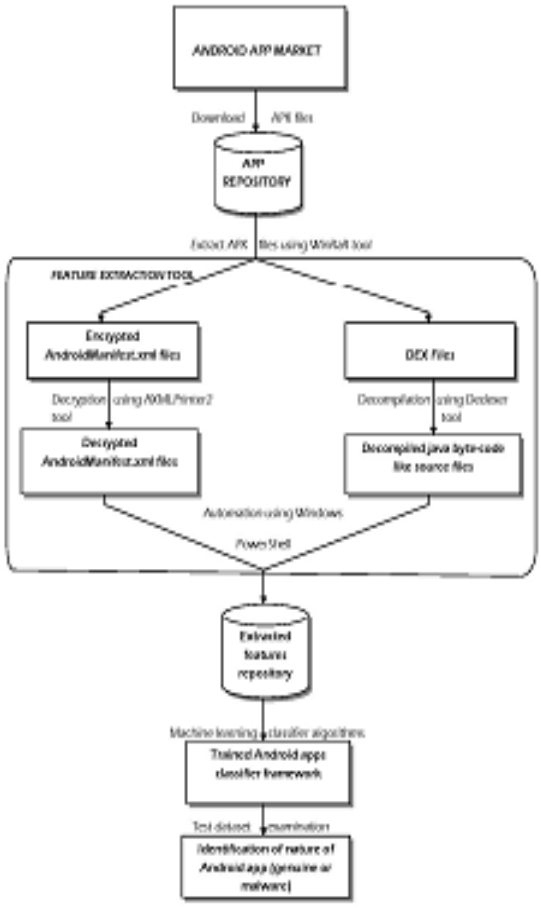


**Fig 1.** System Architecture

The overall architecture diagram is given above. Our system begins with the phase of downloading Android applications. Both genuine and malware applications need to be downloaded to train and test our system. Malware Android applications cannot be easily downloaded from the internet. We became a member in virusshare.com- a malware repository used for research and analysis purposes. After we have created a repository of Android applications, we need to extract the AndroidManifest.xml and classes.dex files from each of the .apk package.

Both the files will be in encrypted form. In order to decrypt the manifest xml file, we use AXMLPrinter2.jar tool. To decompile the dex file (compiled java source code), we use dedexer.jar tool. Both the tools are automated and executed using Windows PowerShell. Since we need to perform the decrypting of xml files and decompiling of dex files for all the .apk packages we have downloaded, it is literally impossible to separately use the tool for every application.

Therefore, we have written PowerShell scripts to automate the above process for all the Android applications (182 genuine and 49 malware applications). After extracting the files, we need to extract the features from both the files. From the manifest file, we need to extract system permissions and from the decompiled dex file we need to search and extract strings under the label "const-string". Both the feature sets will be the input to our classifier framework. We will train the framework using the feature set. After training phase comes the testing phase where individual applications are tested for their nature based on

 

their system permissions and strings used in the source code. Testing of the applications will be based on the feature set given as input for the training phase of the framework.

## V. CONCLUSION

In this system we propose a new method for detecting Android malware using string features to train machine-learning techniques. In order to validate our method, we collected several malware samples of Android applications. Then, we extracted the aforementioned features for each application and trained the models, evaluating each configuration. J48 and NaiveBayes were the best classier obtaining very high accuracy levels. Nevertheless, there are several considerations regarding the viability of our approach.

## REFERENCE

1. Wei Wang, Xing Wang, Dawei Feng, Jiqiang Liu, Zhen Han, Xiangliang Zhang, "Exploring Permissions-induced Risk in Android Applications for Malicious Application Detection"

2. Dong-uk Kim, Jeongtae Kim, Sehun Kim, "Malicious Application Detection Framework using Feature Extraction Tool on Android Market".

3. BorjaSanz, Igor Santos, Carlos Laorden, XabierUgarte-Pedrero and Pablo Garcia Bringas, "On the Automatic Categorization of Android Applications"

4. BorjaSanz, Igor Santos, Carlos Laorden, XabierUgarte-Pedrero and Pablo Garcia Bringas, "MADS: Malicious Android Applications Detection through String Analysis"

5. Matthew G. Schultz, ErezZadok, Salvatore J. Stolfo, "Data Mining Methods for Detection of New Malicious Executables"

6. Yajin Zhou, Zhi Wang, Wu Zhou, Xuxian Jiang, "Hey, You, Get Off My Market: Detecing Malicious Apps in Official and Alternative Android Markets"

7. Adam P. Fuchs, AvikChaudhuri, Jeffrey S. Foster , "SCanDroid: Automated Security Certification of Android Applications"

8. Steven Artz, Siegfried Rasthofer, Eric Bodden, "SuSi: A Tool for the Fullt Automated Classification and Categorization of Android Sources and Sinks"

9. AsafShabtai, YuvaiFledel, Yuval Elovici, "Automated Static Code Anaylsis for Classifying Android Applications Using Machine Learning"

10.Deepak Koundel, SurajIthape, VishakhaKhobaragade, Rajat Jain, "Malware Classification using Naïve Bayes Classifier for Android OS"

11.Franklin Tchakounte, "Permission-based Malware Detection Mechanisms: Analysis and Perspectives".

 