Android Platform Malware Analysis

Rubayyi Alghamdi
Information Systems Security
CIISE, Concordia University
Montreal, Quebec, Canada

Khalid Alfalqi
Information Systems Security
CIISE, Concordia University
Montreal, Quebec, Canada

Mofareh Waqdan
Information Systems Security
CIISE, Concordia University
Montreal, Quebec, Canada

***Abstract*—Mobile devices have evolved from simple devices, which are used for a phone call and SMS messages to smartphone devices that can run third party applications. Nowadays, malicious software, which is also known as malware, imposes a larger threat to these mobile devices. Recently, many news items were posted about the increase of the Android malware. There were a lot of Android applications pulled from the Android Market because they contained malware. The vulnerabilities of those Applications or Android operating systems are being exploited by the attackers who got the capability of penetrating into the mobile systems without user authorization causing compromise the confidentiality, integrity and availability of the applications and the user. This paper, it gave an update to the work done in the project.**

Moreover, this paper focuses on the Android Operating System and aim to detect existing Android malware. It has a dataset that contained 104 malware samples. This Paper chooses several malware from the dataset and attempting to analyze them to understand their installation methods and activation. In addition, it evaluates the most popular existing anti-virus software to see if these 104 malware could be detected.

Keywords—Smartphone Security; Malware Analysis; Android Malware; Static Analysis; Dynamic Analysis; SDK; VAD

I. INTRODUCTION

Several years ago, smartphone and tablet have become more common. They provide services such as social networking, banking, etc. Also, they are equipped with many features like Wi-Fi and GPS, make video calls and many more things. With all these features, there also comes a need for security for the mobile phones. This paper is focusing on the Android Operating System.

Android, which is open source operating system, will be more popular. Currently there are over 50 mobile phone companies are manufacturing smartphones with Android operating system. Increasing the number of the Android devices causes concern in term of user security. McAfee Labs report showed that in the first quarter in 2012, there is a large increase in mobile malware, and the increase was targeted almost only at the Android platform [1]. Figure 1 shows that there were 10 billion application downloaded by the end of the 2011. These rapid increases in applications download make Android to be the most targets for malware.

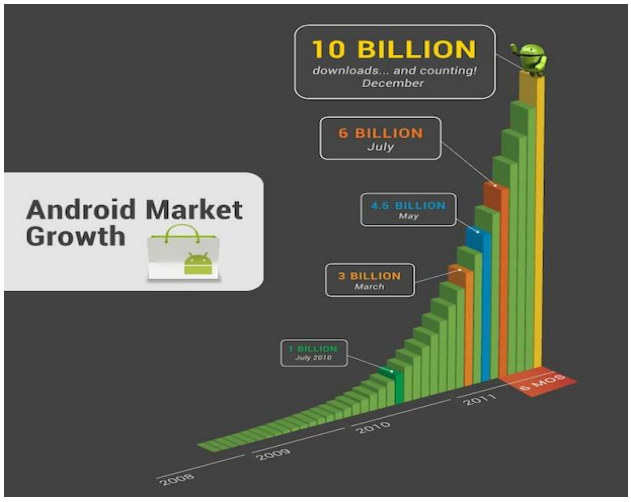

Fig. 1. Android Market Growth

In this paper, we are learning how a malware can target the Android phones and how it could be installed and activated in the device by performing a malware analysis using static and dynamic tools to understand the malware operations and functionalities. To achieve these tasks it is required to understand the Android architecture and its security model.

The rest of this report provide a description of the project and is organized as follows: Section II presents an overview of Android architecture Section III describe Android security model. After that, Section IV describes Android application Section V describes malware analysis followed by analysis result in Section VI. Section VII shows the detection results with four mobile anti-virus software. Section VIII discusses one way for future improvement. Lastly, it concludes the paper in Section IX.

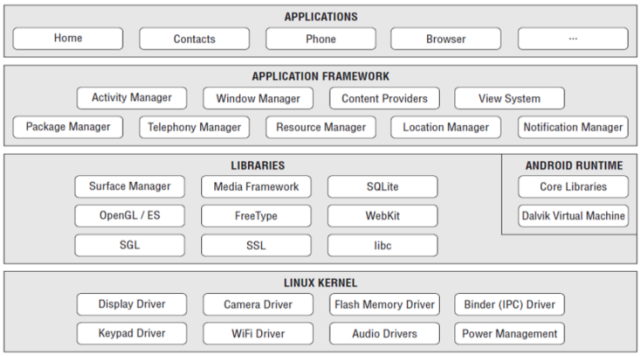

Fig. 2. Android Architecture [3]

II. Overview of Android Architecture

Android is open source software for mobile development developed by Google. The Android architecture as shown Figure2 can be divided into five layers. The first layer from the bottom is the kernel, which is based on the Linux 2.6 kernel. It is used as hardware abstraction layer. The reason Google are using Linux is because it provides memory management, process management, security model, networking, a lot of core operating system infrastructure that are robust and have been proven over time. The next level up is a native or basic library, which is written in C and C++. The next level is the Android runtime. The main component in the Android runtime is the Dalvik virtual machine. It was designed specifically for Android to meet the need of running in an embedded environment where you have limited battery, limited memory, limited CPU. The Dalvik virtual machine runs something called DEX files. Those files are bytes codes that are results of converting at java .classes and .jar files. Those files when are converted to .dex files become much more efficient bytes code that can run very well on small processors. They use memory very efficiency. The next level up from that are the core libraries. They are written in java programming language. It contains all of the collection classes, utilities, I/O, etc. The upper level is the application framework. This is also writing in java programming language. It provides abstractions of the underlying native libraries and Dalvik capabilities to application. Each Android applications run on its Dalvik virtual machine [4].

III. Android Security Modle

The whole idea behind mobile platform is the fact that the user can run a lot and a lot of different applications on the device. The user might be installing and downloading a banking application that can be doing some sensitive data. On other hand, the user might be installing a game application right next to previous application and running on the same device. The user obviously does not want the game application to be able to access the sensitive data that banking application is operation on. So to achieve this Android platform makes sure that any application is isolated from each other. Basically when the user download and install an application, it will be given a unique UID. In addition, each application will run on separate process on separate virtual machine. Therefore, application cannot read other application private data [4].

As it was mentioned on Section II, Android was built on the top of the Linux, so the Linux file permission are applied. Permission allows the user to protect his/her sensitive data that are stored on the device. Also, it protects access to content provider, which basically is a database in the device. Permissions are requested by an application at install time and they are granted or denied once at the install time which requires the user approval [4].

IV. Android Application

Android application has an extension file .apk which is stand for Android package. It is basically an archive file contains all the necessary files and folder in an application. Each application is divided to four main components namely Activities, Service, Broadcast Receivers and Content provider [4].

- Activity is essentially just a piece of User Interface (UI). So any visual screens that allow user to see and interact with in an Android application. It can consist of views such as Button View, Text view Table view, etc.
- Intent Recivier which is a way for which an application to register some code that will not be running until it's triggered by some external event. Developer can write some code through XML and register it to be running when something happens, e.g. network connectivity is established at a certain time, or when the phone is ring.
- Service is a task that does not have any user interfaces. It is a component running in the background. For example, when lunching a music player application, the first screen is an activity. But as soon as selecting a song to play and move to other application, the service keeps running in the background.
- Contant Provider is data storage, which allows the applications to share the data with other application.

Each application contains a manifest file named Androidmanifest.xml. This file declares applications components, specifies the application requirements, and contains the permissions. These permissions will be shown to the user, when he would be installing the application. Figure 3 is an example of the Androidmanifestfile.xml.

```
root@bt:~/Sample Malware # ./aapt d xmltree RU.apk AndroidManifest.xml
N: android=http://schemas.android.com/apk/res/android
  E: manifest (line=2)
    A: package="org.me.androidapplication1" (Raw: "org.me.androidapplication1")
    E: application (line=4)
      A: android:icon(0x01010002)=@0x7f020000
      E: activity (line=5)
        A: android:label(0x01010001)="Movie Player" (Raw: "Movie Player")
        A: android:name(0x01010003)=".MoviePlayer" (Raw: ".MoviePlayer")
        E: intent-filter (line=6)
          E: action (line=7)
            A: android:name(0x01010003)="android.intent.action.MAIN" (Raw:
"android.intent.action.MAIN")
          E: category (line=8)
            A: android:name(0x01010003)="android.intent.category.LAUNCHER" (Raw:
"android.intent.category.LAUNCHER")
    E: uses-permission (line=12)
      A: android:name(0x01010003)="android.permission.SEND_SMS" (Raw:
"android.permission.SEND_SMS")
```

Fig. 3. Example of Androidmanifest.xml

In the above example, it is clear that the application is trying to access the Send SMS feature of the phone, which is stated as the permission Android.permission.SEND_SMS.

The Android Manifest file also helps a user in determining whether an application is a legitimate one or it is a malicious one. For example, a game application does not need permissions such as SEND_SMSM, READ_CONTACTS. In this case, It should be known that if the application is a legitimate or not.

A. MALWARE INFECTION METHODS

There are several methods that the Android devices could be infected with malware. The following are four different methods which malware can be installed on the phone [1,11]:

1) Repackaging legitimate application

This is one of the most common methods used by the attackers. They may locate and download legitimate popular application from the market, disassemble it, add malicious code and then re-assemble and submit the new apps to the official or alternative Android market. Users could be vulnerable by being enticed to download and install these infected applications. It was found that 86.0% repackaged legitimate application including malicious payloads after analyzing more than 1,200 Android malware samples [1].

2) Exploiting Android's application bug

There could be a bug in the application itself. The attacker may use this vulnerability to compromise the phone and install the malware on the device.

3) Fake applications

It was also discovered that there are fake applications created to include malware which allows attacker to access your mobile device. Attackers upload on the market fake applications that seems are legitimate to users but they are malware by themselves. For example, Spyeye's fake security tool was found in the market which is a malware.

4) Remote Install

The malware could be installed in the user phone remotely. If the attacker could compromise users' credentials and pass them in the market, then in this case, the malware will be installed into the device without the user knowledge. This application will contain malicious codes that allow attacker to access personal data such as contacts list [1].

V. MALWARE ANALYSIS

Malware is a piece of code that is executed on the target machine like viruses, Trojans or worms. Sometime it is difficult to stop them since they use new signature, which prevents it from being detected.

Reverse Engineering process is used to analyze the Android Malware. It is a process that decompiling an application to understand its working and functionality by analyzing the codes and debugging it. Before explain the analysis, it is very important to understand how an APK (Android package) is made before reversing it. Figure 4 shows the process of building and reversing APK file.

Once the application is downloaded in the phone from Google Market, the file .apk is available . So, first of all, the file should be de- packaging by using command such as "unzip. Then, the following files and folders will be found [13,14]:

- **Meta-inf Folder:** this folder consists of information that allows users to make sure of the security of the system and integrity of the APK application.
- **Res Folder**: this folder contains XMLs defining the layout, attributes, languages, etc.
- **Classes.dex:** this file contains the entire Java source code that is compiled. This file is run on the Dalvik Machine. This file consists of the c omplete bytecode bthat the Dalvik Machinwill int erpret
- **AndroidMainifest.xml File:** this file is one of the most important XML file which contains information about the permissions that the application needs or accesses. In other words this file contains the Meta information concerning the application.
- **Resources.arsc:** this file is binary resource file that is obtained after compilation.

Here at this point, the focus will be on the classes.dex file, which is the compiled java classes and contains all the codes of the application. Then, decompiling the classes.dex file into readable code (Java files) using several tools such as JAd tool.

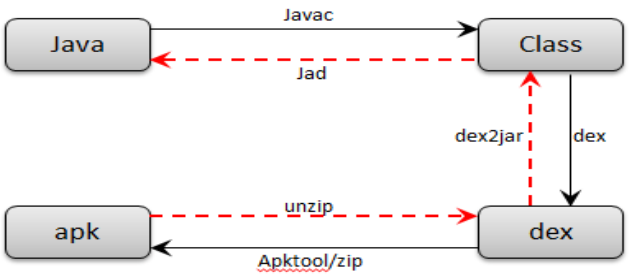

Fig. 4. Reverse APK file

The following sections describe the types of the malware analysis and the tools, which are used to perform a complete analysis.

A. Types of Malware Analysis

1) Dynamic Analysis

Dynamic analysis sometime is also called behavioral analysis, which is used to analyze and study the behavior of malware. Then studying how the malware interact with the system, services added, data capture, network connection made, open port and etc [11].

2) Static Analysis

Static analysis is called code analysis, which is used to analyze the code of the malicious software. The main purpose is to know the exact malicious code, which is embedded in the actual code [11].

B. Tools of Malware Analysis

Several tools were selected for both static and dynamic analysis. In this project, we use two methods to analysis the malware code. The first method involves the utilization of APKTOOL and editor such as Nodepad++. The second method is performed using tools Dex2Jar and JD-GUI. The following is a list of tools used for reverse engineering Android malware [11,12]:

1) Create an isolation environment test

In this project, the VMware workstation 8 was used for the testing. Also, Linux back track 5 r2 on three laptops and Windows 7 in other laptop.

2) Apk Tool

This tool is used to analyze Android application binaries. It has a capability of disassembling applications to practically original form and repackaging them after certain modification. It also is used to debug the smali code [6].

3) Dex2Jar

Dex2Jar tool was developed and used in order to convert .dex file (Dalvik Virtual machine format) to .class format. It helps to view the source code of an application as a Java code [7].

4) Android SDK

The Android Software Development Kit (SDK) is a collection of development tools that are used to create applications. There are several components are included on the SDK such as debugger, an emulator, sample source code, libraries and etc. [5].

5) JD-GUI

This tool is a java decompile that allows a user to view Java Source Codes of .class files. It shows log files and enables user to browse the hierarchy of the class files [8].

6) DroidBox

DroidBox is a dynamic analysis tool of Android applications. It is capable to identify information leaks of content, SMS data IMEI, GPS coordinates, and installed application, phone number and operation file [9].

C. Experimental

As mentioned earlier, there is a rapid growth of Android malware since 2011. In this section, we are explaining the basics of Android applications and showing the way to analyze them. Also, three different applications were chosen that were discovered back in 2010-2011. These sample applications can be downloaded from [10].

1) Android.FakePlayer

By start analyzing the first Android malware that was discovered in 2010 named Android.FakePlayer. It is a Trojan which sends SMS messages to certain numbers. It is distributed as an .apk file named “RU.apk". It pretends to be a movie player but does not actually play movies. This malware requires that the user install it on the device. To download the application sample it can be found here [10].

The file called RU.apk was found. It is a zip file that can be extracted using zip command. After extraction, several files were found. The most important file is AndroidManifest.xml. This is the metadata file that contains the information about the main class of the application as well as other information like permissions. Also, the file classes.dex was found which contains the actual compiled code on the dex file format.

The analysis starts by reading the AndroidManifest.xml file since it contains information about the main entry points of the application as well as other useful information like permission and services used by the application. They can give a general overview of what the application is doing.

The Android Manifest.xml inside the APK file is a binary XML file. The aapt tool was used to convert this format to a common XML format. After reviewing the file, it was obvious that the application is requesting the Android.permission.SEND_SMS that allows the application to send SMS messages.

It was noticed also that the activity that is launched when the application is executed is org.me.Androidapplication1.MoviePlayer. These are the entry points of the application. Since this is a movie player application, we know that the application does not need the send SMS massages.

Then, a disassembling Dex file was done by using Dex2Jar tool to get the readable original code to examine what the code is actually doing. We checked and reviewed the decompiled code of rg.me.Androidapplication1.MoviePlayer, which is the activity that will be lunched first as seen in the AndroidManifest file. We also noticed that it contains a method named onCreate that on Android is called when the activity is starting. The code has several calls to Android.telephony. SmsManager.sendText Message (String destinationAddress, Strings cAddress, String text, PendingIntent sentIntent, PendingIntent deliveryIntent). So it was understood that the first time the application runs, it tries to send a numeric SMS text message to (3353, 3354) [11,12].

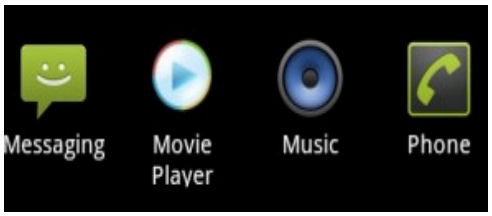

Fig. 5. Android.FakePlayer

To perform dynamic analysis of this malware, we install the application on Android 2,2. As shown in the figure 5, we have a new icon and application called Movie Player. When clicking on this application in our emulator, there is nothing happen. However, we know from our static analysis that after activating this application it will try to send out the SMS message.

2) Android.NickiSpy

The second malware that was analyzed is called Android Nickispy. It is a Trojan horse that steals information from Android devices and sends it to the remote server. It gets activated as soon as device finishes its boot. The package name of this malware is called "com.nicky.lyyws.xmall".

After finishing analyzing the manifest file of this application, it was found that this malware requests a lot of permissions which some of them are related to the conversation recording capabilities (Figure 6) [8, 9].

Fig. 6. NickiSpy Permissions

Also, It was noticed that there is a receiver declared in the Android Manifest file. The malware uses it to start the function after the device is booted. In addition, we found that there are many services declared to be run in the background. After the device boots and the malware activated, we found a list of service in the "running service".

Fig. 7. NickiSpy Services

Those services are running in the background without the user noticing they exist. We craft all the entry point from the manifest file and then we began reviewing the Java code. We found in onCreate() method that the record service is been activated.

Also, it coded that after the malware is installed on the device, it creates a XML file names XM_ALL_setting in the shared preference file.

These file will contain all setting configuration of the remote server to let the device to send the information to the server. Also, we noticed that the address of the remote server is hard coded in the code. It also can be founded in the XM_ALL_setting file.

The following code was written which contains the send function which is used to send the information to remote server.

```
public void send(DataOutputStream paramDataOutputStream)
  throws IOException
{
  int i = this.Headinfo.getMessageLength();
  paramDataOutputStream.writeInt(i);
  int j = this.Headinfo.getCommandId();
  paramDataOutputStream.writeInt(j);
  int k = this.Headinfo.getVersion();
  paramDataOutputStream.writeByte(k);
  char[] arrayOfChar1 = this.Headinfo.getIENO();
  byte[] arrayOfByte1 = new String(arrayOfChar1).getBytes();
  paramDataOutputStream.write(arrayOfByte1);
  char[] arrayOfChar2 = this.UserNumber;
  byte[] arrayOfByte2 = new String(arrayOfChar2).getBytes();
  paramDataOutputStream.write(arrayOfByte2);
  char[] arrayOfChar3 = this.Date;
  byte[] arrayOfByte3 = new String(arrayOfChar3).getBytes();
  paramDataOutputStream.write(arrayOfByte3);
  int m = this.CallType;
  paramDataOutputStream.writeByte(m);
  int n = this.CallLen;
  paramDataOutputStream.writeInt(n);
  char[] arrayOfChar4 = this.Reserve;
  byte[] arrayOfByte4 = new String(arrayOfChar4).getBytes();
  paramDataOutputStream.write(arrayOfByte4);
  paramDataOutputStream.flush();
}
```

Fig. 8. Send Method in NickiSpy

Later, we performed a dynamic analysis of this malware. After installing the application on Android 2, 2. We made a phone call between two emulators; the emulator which has Nickispy is installed on it records the conversation and save it in the SDcard under directories named "shangzhou/callrecord" [11,12].

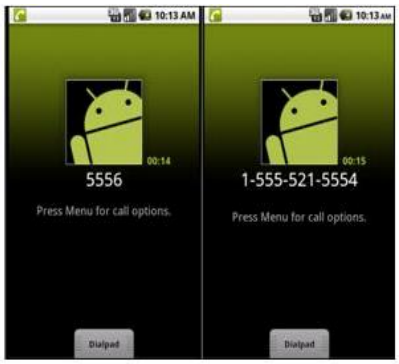

Fig. 9. Conversation Recording

The following code show that the malware sends SMS with the IMEI of the device to number '15859268161'

```
private class _cls1
    implements Runnable
{

    public void run()
    {
        Thread.sleep(600000);
    }
        SmsManager smsmanager;
        String s1;
        smsmanager = SmsManager.getDefault();
        StringBuilder stringbuilder = new StringBuilder("IMEI:");
        String s = imei;
        s1 = stringbuilder.append(s).toString();
        smsmanager.sendTextMessage("15859268161", null, s1, null, null);
```

Fig. 10. Code in Nickispy

The malware records other information other than the phone calls content. It also records GPS location and information and SMSs (received and sent).

3) Android. Seesmic Application

The purpose of the analysis to reverse the Seesmic application which is one of the most popular application in Google Market that allows a user to manage all the social networks . We aims to alter it by adding activities and more permission in the Android Manifest file and then recompile it and let it work on the phone. An attacker uses this process when to he downloads a legitimate application from the market and then embedded his/her malicious code.

We used Android 2.2 emulator and installed the Seesmic application. The application has the following permissions:

- Access to the SD card
- Access to the GPS location
- Full Internet Access and
- Access to phone calls

It is possible to add more permission for this application in order to access to private data such as browser history, SMS and so on. We used the apktool to reverse the Seesmic.apk file. The apktool generated a folder containing the AndroidManifest file and we modified the file by altering the version Name to be 1.6 and adding an activity which display a logo when the application is started and finally, adding more permissions to send and receive an SMS. Now that the Manifest file is altered. Then, we used the apktool to recompile the modified application. If the command executed successfully, a new folder named dist will be created which contain a file .apk. We renamed the file to Seesmic1-6.apk and signed the application using a self-signed certificate which is generated by using openSSL tool. Then, we installed the application on the phone. By checking the application information, it was noticed that the application version changed from (1.5 to 1.6) and that all permissions added in the AndroidManifest file are now available for the Seesmic application. So the Seesmic application has been successfully reversed, activity and permissions have been added on its source code, which was then recompiled to run on the phone [11].

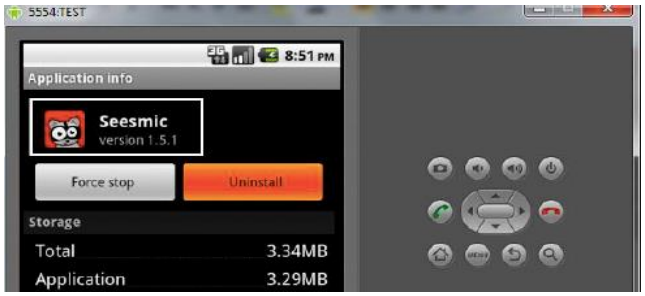

Fig. 11. Seesmic Application

VI. Malware Detection

We attempt to measure the effectiveness of existing mobile anti- virus software. We choose four mobile anti-virus software, i.e., AVG Antivirus Free, Lookout Security & Antivirus, BitDefender Mobile Security, and Avast Security Edition and download them from the official Android Market. We install each of them on a separate emulator running Android 2.2. We apply the default setting and enable the real-time protection. After that, we create a script that installs 104 applications in four emulators. If malwareis detected, these anti-virus software will pop up an alert window, and then we recorded it down [1].

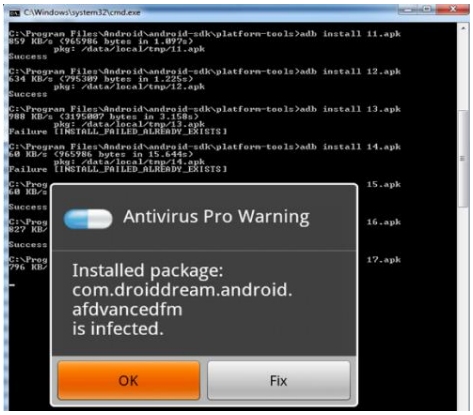

Fig. 12. AVG Detection

The table below shows the number of detected malware for each anti-virus, and its corresponding detection rate. The results are not encouraging: Avast detected 56 malwares; Lookout detected 51 malware; BitDefender detected 49 malware and finally AVG detected 46 malware.

TABLE I. The Number of Detected Malware for Each Anti-Virus

	AVG	BitDefender	Avast	Lookout
Detected	46	49	56	51
%	44.2	47.1	53.8	49.0

There are some malwares were not detected by the four anti-virus software. One reason is that existing mobile anti-virus companies may not update the database signature for the free version anti- virus available in the market. Today, many people use free antivirus software and they have to know that they are not protected from malware.

VII. Analysis Results

After completed the test, we agreed that it becomes very important for every user to check the permissions that any application he/she is downloading really requires access to them or not. One of the major disadvantages of Android applications is that without agreeing to grant access to all the permissions, an application cannot be installed on the device. For example, the application Movie Player is only supposed to play movies and do nothing more. Hence it is obvious that it does not require permission to send messages or receive SMS. Similarly, the application Seesmic accesses certain permissions in order for it to work normally. Since Seesmic is one application that allows a user to manage all the social networks, hence it requires

access to Internet, Location in order to update status; but it does not require access to the user's SMS and private data. After we modify the manifest file and install the application, the user needs to have certain responsibilities while installing the application.

VIII. Future Work

The Android Manifest file is the only file that has been altered in order to allow the application to access more sensitive data. 83% of the malware they found in their analysis that they are repackaging applications. Therefore, if the developer can implement a method or implement an algorithm to detect the modified Android Manifest file before the application is installed to the phone, then the risk of those malware will be mitigated.

IX. Conclusion

In the past few years smartphone users have increased quickly. There are attackers who are now targeting smartphones. The main reason for this because the lack of user awareness regarding how their devices can be compromised. Today, smartphones like Android are not just used as a portable telephone. Android devices can access the internet, make online bank transmissions, manage social networks, etc. All these functionalities of a mobile phone seem very attractive for an attacker to gain information of the user and use it to his/her benefit. Therefore, users need to be aware enough and have full responsibilities to read and understand the permissions requested by the application before agreeing to grant access.

Acknowledgement

This work is sponsored by Al- Baha University. Authors acknowledge the university for the kind support.

References

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

[2] Jew Mark, "Android Market Reaching the Same Growth as App Store "gadgetoz.com, Dec 7, 2011. [Online]. Available: http://www.gadgetoz.com/post/android-market-reaching-the- same-growth-as-app-store/. [Accessed: July 25, 2012].

[3] Technolgy, "Architecture of Android OS" techneology.com, Nov 2011. [Online]. Available: http://www.techneology.com/2011/11/architecture-of- android-os.html. [Accessed: July 25, 2012].

[4] Frank Ableson, "Introduction to Android development the open source appliance platform "ibm.com, 12 May 2009. [Online]. Available:http://www.ibm.com/developerworks/opensource/library/os-android-devel/. [Accessed: July 25, 2012].

[5] Android developer, Android-SDK. [Online]. Available: http://developer.android.com/sdk/index.html. [Accessed: July3, 2012].

[6] Reverse engineering tool for Android apk files, Android- Apktool. [Online]. Available: http://code.google.com/p/android-apktool/. [Accessed: July 3, 2012].

[7] Tools to work with android .dex and java .class files, Dex2jar. [Online]. Available: http://code.google.com/p/dex2jar/. [Accessed: July. 3, 2012].

[8] Decompiler, JD-GUI. [Online]. Available: http://java.decompiler.free.fr/?q=jdgui. [Accessed: July. 3,2012].

[9] Android Application Sandbox, DroidBox.[Online].Available: http://code.google.com/p/droidbox/. [Accessed: July. 3, 2012].

[10] Mobile malware mini dump, Malware dataset. Online]. Available: http://contagiominidump.blogspot.ca/. [Accessed: July. 3, 2012].

[11] Vibha Manjunath, "Reverse Engineering of Malware on Android" sans.org, Aug 31, 2011. [Online]. Available: http://www.sans.org/reading_room/whitepapers/pda/reverse-engineering-malware-android_33769. [Accessed:July. 10,2012].

[12] Jaime Blasco, "Introduction to Android Malware Analysis" [Magazine, Issue 34, June 2012]. Retrieved from :http://net-security.org\insecuremag.php. Last Accessed: 11 July, 2014