“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........................................................................................................  1

1. Introduction..................................................................................................................  2

2. Malware Analysis........................................................................................................  3
   2.1 Basics of Malware Analysis.............................................................................  3
   2.2 Types of Malware Analysis..............................................................................  3
   2.3 Components of Malware Analysis...................................................................  3
   2.4 Tools of Malware Analysis..............................................................................  4
   2.5 Goals of Malware Analysis..............................................................................  4

3. Operating System Overview........................................................................................  4
   3.1 Android Architecture........................................................................................  4
   3.2 Android Market................................................................................................  6
   3.3 Android Security Model...................................................................................  6
   3.4 Malware found in Operating Systems..............................................................  7

4. Methodology................................................................................................................  9
   4.1 Introduction......................................................................................................  9
   4.2 Android Malware Analysis..............................................................................  9
      4.2.1 List of Tools.........................................................................................  9
         4.2.1.1 Virtual Appliance.........................................................  9
         4.2.1.2 Emulator.....................................................................  10
         4.2.1.3 Disassembler..............................................................  11
         4.2.1.4 Unpacker....................................................................  11
         4.2.1.5 Text Editor.................................................................  12
         4.2.1.6 Monitoring Tool.........................................................  12
         4.2.1.7 Other Tools.................................................................  13
         4.2.1.7.1 Dex2Jar...........................................................  13
         4.2.1.7.2 JD-GUI...........................................................  13
4.2.2 Creating an ISO environment............................................................. 13
4.2.3 Baseline the environment................................................................... 14
4.2.4 Malware Acquisition.......................................................................... 15
4.2.5 Reverse Engineering on Android Malware........................................ 16
    4.2.5.1 Dynamic Analysis................................................................. 16
    4.2.5.2 Static Analysis........................................................................ 24
4.2.6 Results................................................................................................ 27

5. Conclusions................................................................................................................ 28
  5.1 Mitigation and Control................................................................................... 28
  5.2 Conclusion...................................................................................................... 28
  5.3 Future Work................................................................................................. 29

6. Appendix.................................................................................................................... 30

7. Bibliography................................................................................................................ 31
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 WinRar 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.
- Dynamic Analysis – Ollydbg, IDA Pro, Dex2jar, JD-GUI, Baksmani, 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
As we can see in the above architecture [6] Linux kernel [7] is the base of the architecture. It has a proprietary Dalvik VM.

<table>
<thead>
<tr>
<th>Criteria</th>
<th>Dalvik</th>
<th>JVM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture</td>
<td>Register-based</td>
<td>Stack-based</td>
</tr>
<tr>
<td>OS-Support</td>
<td>Android</td>
<td>All</td>
</tr>
<tr>
<td>RE-Tools</td>
<td>a few (dxdump, ddx)</td>
<td>many (jad, becl, findbugs, ...)</td>
</tr>
<tr>
<td>Executables</td>
<td>DEX</td>
<td>JAR</td>
</tr>
<tr>
<td>Constant-Pool</td>
<td>per Application</td>
<td>per Class</td>
</tr>
</tbody>
</table>

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

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
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 exploitd (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. 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:
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:

```xml
<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_CALL_LOG"/>
<uses-permission android:name="android.permission.READ_CALENDAR"/>
<uses-permission android:name="android.permission.READ_HISTORY_BOOKMARKS"/>
<uses-permission android:name="android.permission.READ_OWNER_DATA"/>
<uses-permission android:name="android.permission.READ_PHONE_STATE"/>
<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**


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:
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.

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.

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:

![Figure 9: Installation of Apktool Framework](image)

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 unzZipper 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](image)
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]

![Figure 11: Snapshot of Notepad++](image)

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.

![Figure 12: Snapshot of Wireshark](image)
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]

![Figure 13: Snapshot of Dex2Jar](image)

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.

![Figure 14: Snapshot of Java Decompiler](image)

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:

![Windows 7 Configuration](image)

**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:

![Android Virtual Device Configuration](image)

**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:

![Updated Windows 7 OS and Virtual Box](image1)

**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](image2)

**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:

![Figure 19: Contents extracted by 7-Zip](image)

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.

![Figure 20: Configuration in Android Manifest File](image)

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:

![Decompiling Malware APK file](image1.png)

**Figure 21: Decompling 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.

![Contents extracted after Decompling](image2.png)

**Figure 22: Contents extracted after Decompling**

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”.

![Code corresponding to the Service started](image3.png)

**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:
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.

```
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-eq v0, icond_0
    lget-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<-startActivity(Ljava/lang/String;)V
    invoke-virtual (v0, v1), Lcom/eguan/state/Dialog<-startActivity(Landroid/content/Intent;)V
```

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:

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.

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 local FileOutputStream.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:

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 “Droid KungFu”.

![Figure 30: Requests sent to Chinese Server](image)

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:

![Figure 31: GET Request and its corresponding Destination Address](image)

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:
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%3A%22HTC%22%3A%222C%22%3A%22smc_voda_wwe%22%3A%222ts%22%3A%2220%22%3A%22sv%22%3A%221.1%22%3A%22po%22%3A%22android%201.5%22%3A%22cid%22%3A%22162fbcbb180446ef5573b679baad3f30%22%3A%22h%22%3A%2280%22%3A%22A480)&out=0&rt=2010-05-29%2012%3A25%3A31&src=3&ver=1.0&sig=ERMdekHHiYznbRN%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:

Figure 32: Encoded URL and its response

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

Figure 33: Data decoded using Paros
Launching the Exploits

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

![Java code for Create Function](image1)

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

![Java Code for getPermission Method](image2)

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 Code for checkPermission Method](image3)
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:

![Java Code of the Exploit](image1.png)

**Figure 37: Java Code of the Exploit**

If successful it will launch the exploit as shown below:

![Method used to run the Exploit](image2.png)

**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:

![Contents of WebView.db.init file](image3.png)

**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:

![Virus Total Report]

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].

![Installing Malicious APK file in AVD]

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”.
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:
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.

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

Figure 43: Malicious Activity, Service and Process shown running

Figure 44: Java Code for Root exception

Figure 45: Error generated by the Malware
When we use Google translate to understand the error we get the following translation as shown:

![Google Translate Image](image)

**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 “Droid KungFu-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 “Droid KungFu-2A” Malware

File name: _com.allen.txt_hej_1.1.0_F438ED38B59F772E03EB2CAB97FC7685.apk
Submission date: 2011-08-02 06:40:51 (UTC)
Current status: finished
Result: 23 /43 (53.5%)
7. Bibliography


<table>
<thead>
<tr>
<th>Course Name</th>
<th>Location</th>
<th>Start Date</th>
<th>End Date</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>SANS Miami 2020</td>
<td>Miami, FLUS</td>
<td>Jan 13, 2020 - Jan 18, 2020</td>
<td>Live Event</td>
<td></td>
</tr>
<tr>
<td>Cyber Threat Intelligence Summit &amp; Training 2020</td>
<td>Arlington, VAUS</td>
<td>Jan 20, 2020 - Jan 27, 2020</td>
<td>Live Event</td>
<td></td>
</tr>
<tr>
<td>SANS Amsterdam January 2020</td>
<td>Amsterdam, NL</td>
<td>Jan 20, 2020 - Jan 25, 2020</td>
<td>Live Event</td>
<td></td>
</tr>
<tr>
<td>SANS Anaheim 2020</td>
<td>Anaheim, CAUS</td>
<td>Jan 20, 2020 - Jan 25, 2020</td>
<td>Live Event</td>
<td></td>
</tr>
<tr>
<td>MGT521 Beta Two 2020</td>
<td>San Diego, CAUS</td>
<td>Jan 22, 2020 - Jan 23, 2020</td>
<td>Live Event</td>
<td></td>
</tr>
<tr>
<td>SANS San Francisco East Bay 2020</td>
<td>Emeryville, CAUS</td>
<td>Jan 27, 2020 - Feb 01, 2020</td>
<td>Live Event</td>
<td></td>
</tr>
<tr>
<td>SANS Las Vegas 2020</td>
<td>Las Vegas, NVUS</td>
<td>Jan 27, 2020 - Feb 01, 2020</td>
<td>Live Event</td>
<td></td>
</tr>
<tr>
<td>SANS Vienna January 2020</td>
<td>Vienna, AT</td>
<td>Jan 27, 2020 - Feb 01, 2020</td>
<td>Live Event</td>
<td></td>
</tr>
<tr>
<td>SANS Security East 2020</td>
<td>New Orleans, LAUS</td>
<td>Feb 01, 2020 - Feb 08, 2020</td>
<td>Live Event</td>
<td></td>
</tr>
<tr>
<td>SANS New York City Winter 2020</td>
<td>New York City, NYUS</td>
<td>Feb 10, 2020 - Feb 15, 2020</td>
<td>Live Event</td>
<td></td>
</tr>
<tr>
<td>SANS Northern VA - Fairfax 2020</td>
<td>Fairfax, VAUS</td>
<td>Feb 10, 2020 - Feb 15, 2020</td>
<td>Live Event</td>
<td></td>
</tr>
<tr>
<td>SANS Dubai February 2020</td>
<td>Dubai, AE</td>
<td>Feb 15, 2020 - Feb 20, 2020</td>
<td>Live Event</td>
<td></td>
</tr>
<tr>
<td>SANS Scottsdale 2020</td>
<td>Scottsdale, AZUS</td>
<td>Feb 17, 2020 - Feb 22, 2020</td>
<td>Live Event</td>
<td></td>
</tr>
<tr>
<td>SANS San Diego 2020</td>
<td>San Diego, CAUS</td>
<td>Feb 17, 2020 - Feb 22, 2020</td>
<td>Live Event</td>
<td></td>
</tr>
<tr>
<td>SANS Brussels February 2020</td>
<td>Brussels, BE</td>
<td>Feb 17, 2020 - Feb 22, 2020</td>
<td>Live Event</td>
<td></td>
</tr>
<tr>
<td>Open-Source Intelligence Summit &amp; Training 2020</td>
<td>Alexandria, VAUS</td>
<td>Feb 18, 2020 - Feb 24, 2020</td>
<td>Live Event</td>
<td></td>
</tr>
<tr>
<td>SANS Training at RSA Conference 2020</td>
<td>San Francisco, CAUS</td>
<td>Feb 23, 2020 - Feb 24, 2020</td>
<td>Live Event</td>
<td></td>
</tr>
<tr>
<td>SANS Jacksonville 2020</td>
<td>Jacksonville, FLUS</td>
<td>Feb 24, 2020 - Feb 29, 2020</td>
<td>Live Event</td>
<td></td>
</tr>
<tr>
<td>SANS Manchester February 2020</td>
<td>Manchester, GB</td>
<td>Feb 24, 2020 - Feb 29, 2020</td>
<td>Live Event</td>
<td></td>
</tr>
<tr>
<td>SANS Zurich February 2020</td>
<td>Zurich, CH</td>
<td>Feb 24, 2020 - Feb 29, 2020</td>
<td>Live Event</td>
<td></td>
</tr>
<tr>
<td>SANS Secure India 2020</td>
<td>Bangalore, IN</td>
<td>Feb 24, 2020 - Feb 29, 2020</td>
<td>Live Event</td>
<td></td>
</tr>
<tr>
<td>Blue Team Summit &amp; Training 2020</td>
<td>Louisville, KYUS</td>
<td>Mar 02, 2020 - Mar 09, 2020</td>
<td>Live Event</td>
<td></td>
</tr>
<tr>
<td>SANS Northern VA - Reston Spring 2020</td>
<td>Reston, VAUS</td>
<td>Mar 02, 2020 - Mar 07, 2020</td>
<td>Live Event</td>
<td></td>
</tr>
<tr>
<td>SANS Secure Japan 2020</td>
<td>Tokyo, JP</td>
<td>Mar 02, 2020 - Mar 14, 2020</td>
<td>Live Event</td>
<td></td>
</tr>
<tr>
<td>SANS Munich March 2020</td>
<td>Munich, DE</td>
<td>Mar 02, 2020 - Mar 07, 2020</td>
<td>Live Event</td>
<td></td>
</tr>
<tr>
<td>SANS St. Louis 2020</td>
<td>St. Louis, MOUS</td>
<td>Mar 08, 2020 - Mar 13, 2020</td>
<td>Live Event</td>
<td></td>
</tr>
<tr>
<td>SANS Austin Winter 2020</td>
<td>OnlineTXUS</td>
<td>Jan 06, 2020 - Jan 11, 2020</td>
<td>Live Event</td>
<td></td>
</tr>
<tr>
<td>SANS OnDemand</td>
<td>Books &amp; MP3s OnlyUS</td>
<td>Anytime</td>
<td>Self Paced</td>
<td></td>
</tr>
</tbody>
</table>