JUNE 1, 2016

# BUILD A DYNAMIC MALWARE ANALYSIS SYSTEM FOR ANALYZING ANDROID APPS TARGETING ON ANDROID'S NEW RUNTIME (ART)

Research Paper

R R HWAWASAM

**SRI LANKA INSTITUTE OF INFORMATION TECHNOLOGY FACULTY OF POSTGRADUATE AND RESEARCH**

**M.SC. IN IT / IS / IM RESEARCH PROJECT**

# BUILD A DYNAMIC MALWARE ANALYSIS SYSTEM FOR ANALYZING ANDROID APPS TARGETING ON ANDROID'S NEW RUNTIME (ART)

Supervised by:

Mr.Lakmal Rupasinghe

# Abstract

Android users are free to download and install 3rd party applications. But just as the development of the PC brings computer viruses and results in a serious threat to Internet security. The rapid development of smart terminal also brings a variety of malicious software, mobile Internet is also facing a growing number of security issues. Android has more than 60% of the market share and it is in the initial stage.[1] The mobile phone security industry analysis reports in year 2012, that they have found 26850 of malware infections on Android.[1] Stealing confidential information (i.e IMEI numbers, mobile numbers or Contact details) and send SMS messages without user permission are the most damaging attacks done by the malware in the past. Now it is not like that, the smart terminals perform the same functionality like computers. Data transmission, online shopping using credit cards are few examples, today we perform using smart terminals. Therefore the cost of the vulnerability is very high than the past couple of years. Android device shipments are expected to top 1 billion this year and there are currently more than 1.100.000 apps available in the Play Store, which generated 50 billion downloads in 2013 alone, therefore, it is reasonable to assume that there is plenty of potential malware* and other security threats designed to take advantage of careless Android users.[6] Considering cost, there is a huge requirement to protect Android system from the malware infection. In this product I compatible an existing open source malware analysis tool to new Android ART (Android Runtime). DroidBox is very easy to use tool developed to offer dynamic analysis of Android applications. It is well known malware analysis sandbox and it will dynamically analyze the Android apps.

# Contents

# Introduction: Context and Background

Smart phones becoming more popular than the personal computers and laptops. In this device we are using open source operating system which has remarkable growth in a short period of time. This is mainly its openness and it is a free open source software. Android users are free to download and install 3rd party applications and also it freely promotes lots of developments on top of the Android operating systems. Because the character of open source Android is supported by a lot of companies, so it is developing very fast, by now it is the dominative Smartphone operating system.

Although Android inherited from Linux kernel, and it introduces many other's security mechanism, such as application signature, use virtual machine to run application, authorization management, and its security is higher. But not enough, it has vulnerabilities. Malware is coming with some interesting applications and users are compelled to install them with taking a risks. After it is installed, it will steal user's personal information and leak it to remote destination for evil purpose, and some malware will start paid service while the user are not known. This is a very raw domain and people are not aware of the security threats this field poses.

In such situation identifying malware is a challenge because thousands of apps are developed daily and there are millions of apps in the app stores. Antivirus companies are failed to identify threats in new viruses because of this. And currently, detection on mobile malware is based on traditional computer virus detection method based on signature or behaviors. We can say that:

- static analysis
- dynamic analysis

Most of the intrusion detection tools are from either static or dynamic analysis techniques are used.

Static Analysis: it is a reverse technology, extract Android package and decompile DEX file to get smali file to analysis the malware and the potential threats by detecting the sensitive API in the source code, and determines whether there is a sensitive data leakage. And it is analyzing the malware without executing the program.

Dynamic analysis: use smali files generated in the static analysis to add the corresponding log generated code after sensitive API function, and then repackage application and signature, which will install repackaged application to the modified simulator and the API will analyze all generated sensitive log.

However, it is not every time easy to do static analysis because the source code has been obfuscated after complication, packaging, and signing. The obfuscation tool detects and deletes useless class, field, methods and attributes, and deletes useless annotation to make the byte code optimized. Besides, it makes use of meaningless name to rename class, fields, and methods, thus, the source code has poor readability and greatly increases the difficulty of code analysis. So we think that it is more efficient and effective if we use both these techniques to identify intrusions in the android systems. In this paper I will dynamic analysis open source tool to determine malware infection into the Android OS. The project idea is taken from Honeynet Project.

The Honeynet Project is a leading international, non-profit security research organization, dedicated to investigating the latest attacks and developing open source security tools to improve Internet security. This idea is from the Honeynet project. (GSoC 2015 Project Ideas) [18]

DroidBox is very easy to use tool developed to offer dynamic analysis of Android applications. It is well known malware analysis sandbox and it will dynamically analyze the Android apps. DroidBox, authored by Patrick Lantz, is a sandbox for the Android platform. "It focuses on detecting information leaks that were derived from performing taint analysis for information-flow tracking on Android trojan applications. DroidBox is capable to identify information leaks of contacts, SMS data, IMEI, GPS coordinates, installed apps, phone numbers, and network traffic and file operations." DroidBox consists of an own system image and kernel meant to log one applications activities. Using adb logcat DroidBox will look for certain debug messages and collect anything related to the monitored app.[17] Nevertheless you can use DroidBox to get an overview of malicious activities triggered by the app. unfortunately this toold is built in Dalvik Runtime environment and currently it become obsolete since there is a new Android ART is in the market.

DroidBox will dynamically analyze the Android apps and providing following information.

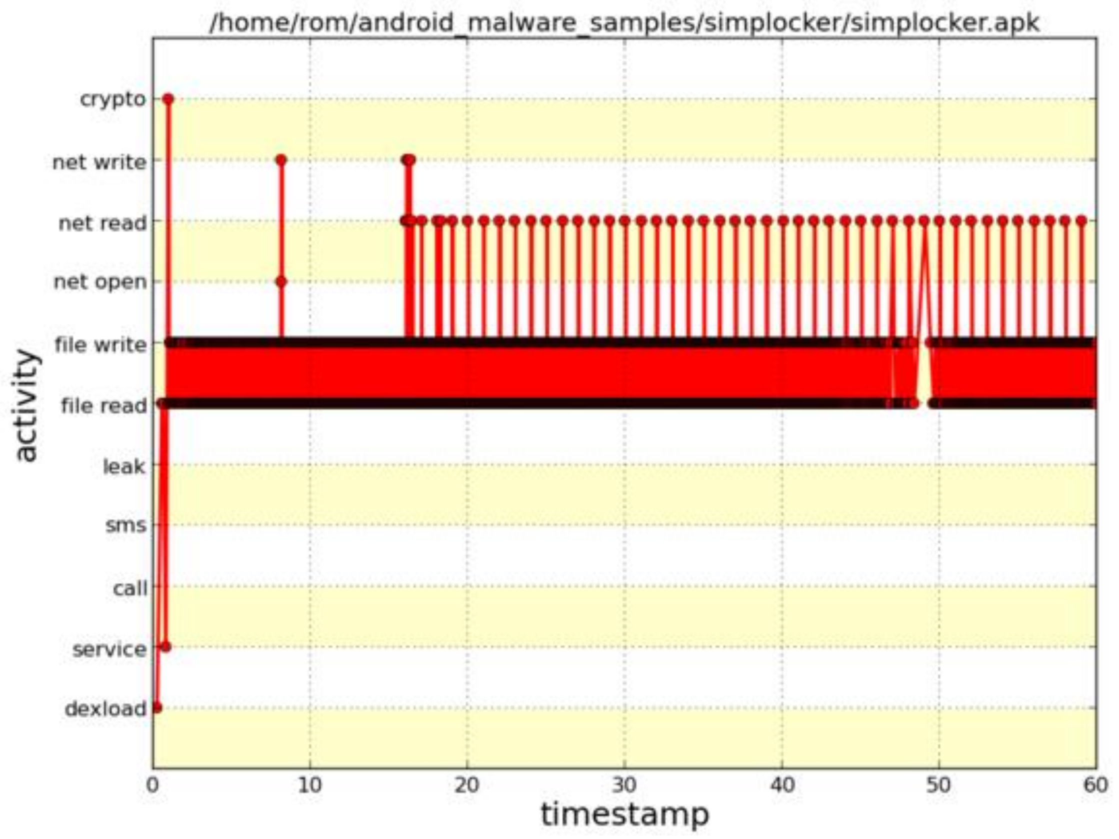
1. Hashes for the analyzed package
2. Incoming/outgoing network data
3. File read and write operations
4. Started services and loaded classes through DexClassLoader
5. Information leaks via the network, file and SMS
6. Circumvented permissions
7. Cryptographic operations performed using Android API
8. Listing broadcast receivers
9. Sent SMS and phone calls

With Android 4.4, a new Android runtime, **ART**. This runtime offers a number of new features that improve performance and smoothness of the Android platform and apps. Currently, ART is available on a number of Android 4.4 devices, such as the Nexus 4, Nexus 5, Nexus 7, and Google Play edition devices. At this time, all devices still use Dalvik as the default runtime. Android runtime (ART) is the managed runtime used by applications and some system services on Android. ART and its predecessor Dalvik were originally created specifically for the Android project. ART as the runtime executes the Dalvik Executable format and Dex bytecode specification. ART and Dalvik are compatible runtimes running Dex bytecode, so apps developed for Dalvik should work when running with ART. However, some techniques that work on Dalvik do not work on ART.

In this product I compatible an existing open source malware analysis tool to new Android ART (Android Runtime). Then I will analyze some most popular Android Operating systems and reviewed their inbuilt security features as well.

**How DroidBox works?**

DroidBox connects to target which is a emulator consist of malware. Then it will receive all the possible information and display in a text or graphical format. The diagram file shows overall distribution of operations.



Simplocker. Chart of the operation distribution by time

A chart showing the distribution of operations as a function of time

The generated trace files which records access information as following format.

```
"Section name" {
  "Operation time (from start of launch)" {
     "Name of parameter being traced (e.g., for sendnet, this could be desthost, destport, etc.)":
"Parameter value"
  }
}
i.e.
          “cryptousage”: {
          “0.9651350975036621”: {
          “algorithm”: “AES”,
          “key”: “95, –81, 109, &lt;…&gt;”,
          “operation”: “keyalgo”,
          “type”: “crypto”
          }
          &lt;…&gt;},
```

## Problem definition

From Android 4.4, Android system has a new runtime called ART [15][16] together with Dalvik. Most apps should just work when running with ART. However, some techniques that work on Dalvik do not work on ART. Since Android 5.0, Google totally abandoned Dalvik, so ART becomes the only runtime. Current dynamic analysis systems such as DroidBox, TaintDroid, DroidScope, etc., they are built on Dalvik VM, porting them to ART seems impossible since they depend on DVM heavily[15].

## Aim and objectives

DroidBox is developed to offer dynamic analysis of Android applications.

Already developed open source project has to work on new Android ART (Android Runtime).

The goal of this project is to build a dynamic malware analysis system on ART, which allows users to monitor the execution of potentially malicious apps. This includes the following sub-goals:

- Monitoring function calls
- Modifying parameters/return value before/after function's execution
- Dumping objects' contents
- Reporting layer that is compatible with existing systems

## Proposed solution

The solution of this project should guarantee two points: low performance overhead and easily maintainability of analysis environment for future new Android versions.

The open source DroidBox is enhanced to work on new Android versions, it requires skills of JAVA, C++, Linux, Android Systems, LLVM and ART.

## Literature review

Android which is based on Linux and is open source, which is mainly used for smart mobile devices, such as smartphones and pads. Android is mainly applied in the field of personal smart mobile devices, and this kind of devices always stored personal information. If operating system is vulnerable then the personal information can be stolen by malware applications. In many researches there are mainly two kinds of Android malware detection technique used:

1. **signature-based (static)**
2. **behavior-based(dynamic) detection**

A research has being done which is called N-gram[5] and it is a static analysis, based malware prevention techniques for Android based mobile phones. N-gram is a software program that uses machine learning based algorithm to detect the that given application is malware or not. In the paper [7], authors have investigated a method and designed a program that uses machine learning algorithm to detect the given mobile application is having malware or not.

Another research [8] based on static analysis; they have proposed a method to detect malware based on the permissions and packages of the applications. Rather than only considering the permission information more information (i.e.,the package information in DEX files) is also measured since some studies have shown that package information contains useful information of Android applications.

Further in some researches which use extended mechanisms rather than above two direct techniques. Some of them are as follows.

Bose[3] proposed a behavior detection system on ***Symbian***, training classification model using SVM; Schmidt et al. [4] made *classification detection on Symbian by statically extracting call functions and suggested the centroid Machine Learning algorithm*.

## How Android Supports Security and Current Study

Android is a mobile operating system (OS) developed by Google, and it is used by several Smartphone. Android phones typically come with several built-in applications and also support third-party programs. Developers can create programs for Android using the free Android SDK (Software Developer Kit). Android programs are written in Java and run through Google's "Dalvik" virtual machine, which is optimized for mobile devices. Users can download Android "apps" from the online Android Market.

## *Android Operation system Architecture*

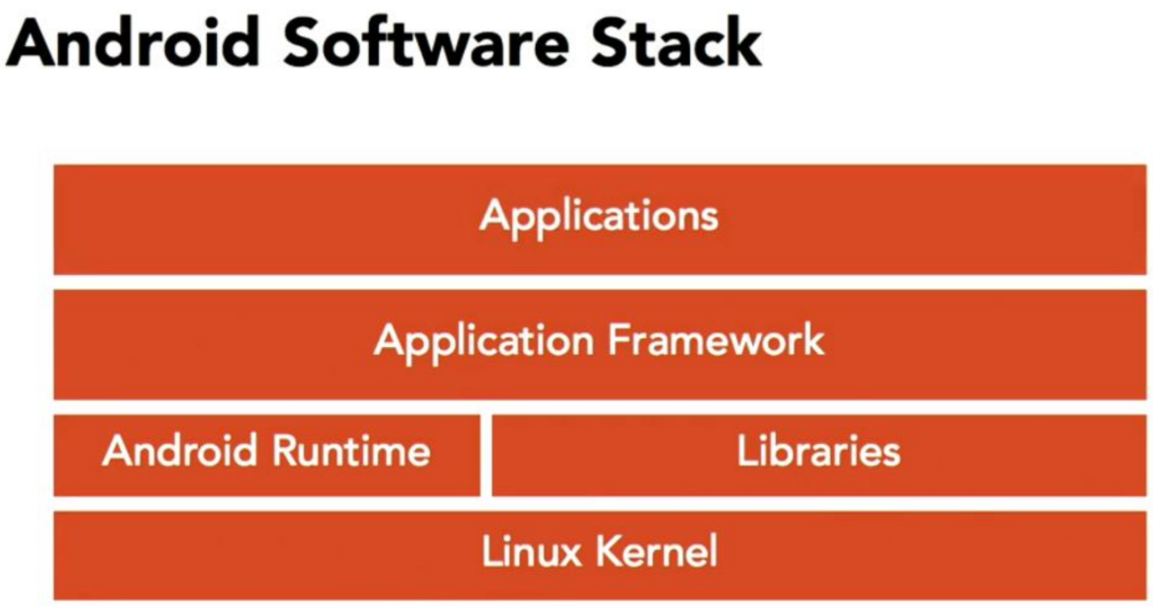


**Figure 0:**

Android is based on linux kernel, and it is highly optimized for mobile operating systems, and it made small as possible. On top of the kernel there is an Android runtime etc. It up to the Original equipment manufacturer (OEM) to customize these packages according to the their device. So Android is combination of google Android OS and customized drivers from OEM. These OEM like LG, Motorola, Nexus etc. Android Runtime include Core Libraries, and there are two separation for Android 4.4 and Android 5. That is Dalvik is in older versions (4.4 <) and ART is in Android 5 upwards. ART is introduced in KIT-KAT and completely replaced in Lollipop. ART uses a head of time compilation, that easy and faster to run apps, because they are compiled to machine code upon installation rather than application is initiated until feature is being called. To compile an app, you need to use compiler which include in Android SDK.The next level is application framework, it has module to controlling your application.

## *Application Components*

These are like building blocks of an Android application. Each component is a different and basically there are four types of application components.

Activities - An application has several activities, a user can see an activity from a user interface, if there are many activities, and then one of them should be marked as default activity.

- Services - A service is running in the background which used by the application.
- Content Providers - It is data supply component.

- Broadcast receivers - Which are responding to other application when communication is happened.
- Android application package file (APK)

This is a package file which is used in distributing and installing application software and middleware on top of the Android OS.

i.e *.dex files

The file extension should be end as .apk

## *The Compilation Process*

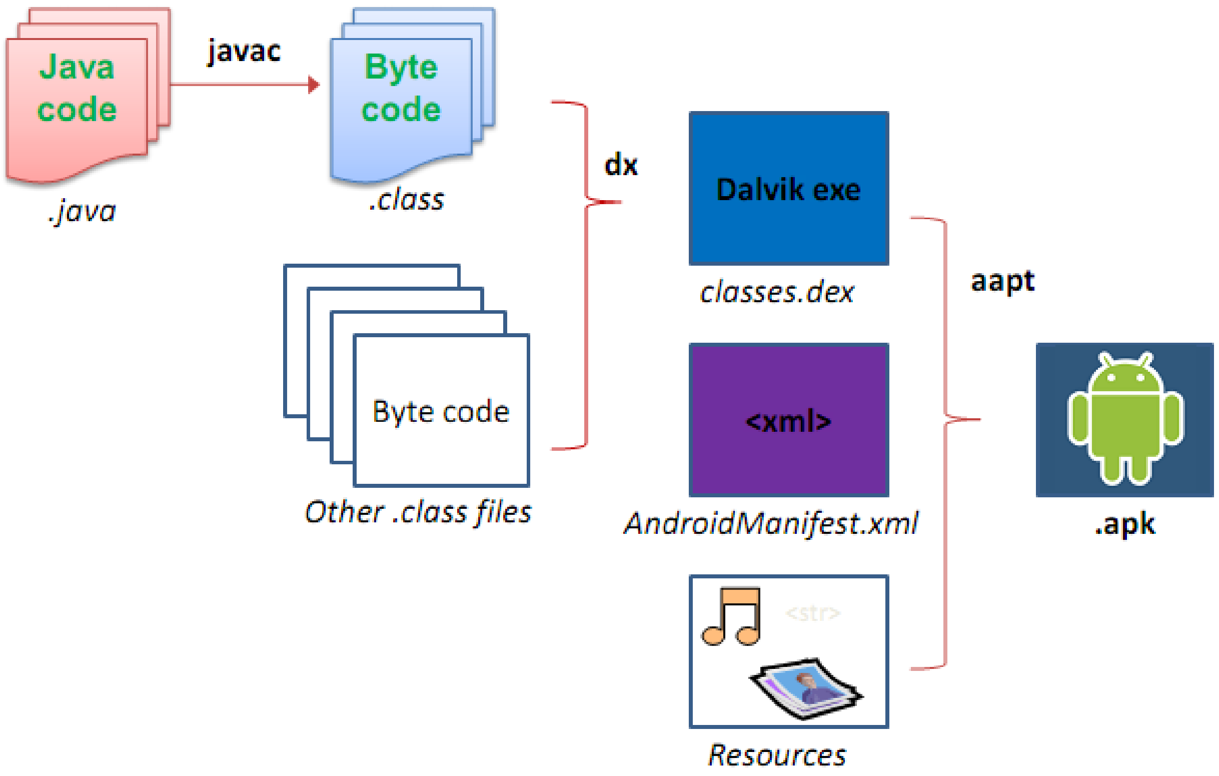


Figure: 1

The application source code is normal java code, it is a *.java, and it will be compiled from Oracle JDK. (javac). This will produced a class file, (byte code). There will be an optional third party tool called **ProGurd**$^{TM}$. This will be done for obfuscate the code, it will reduce the readability of the code, and so on. So decompilers cannot read the code correctly. Then regardless of the **ProGurd**$^{TM}$, then class file will be converted to *.DEX bytecode file. The dex file will be distributed via application package.

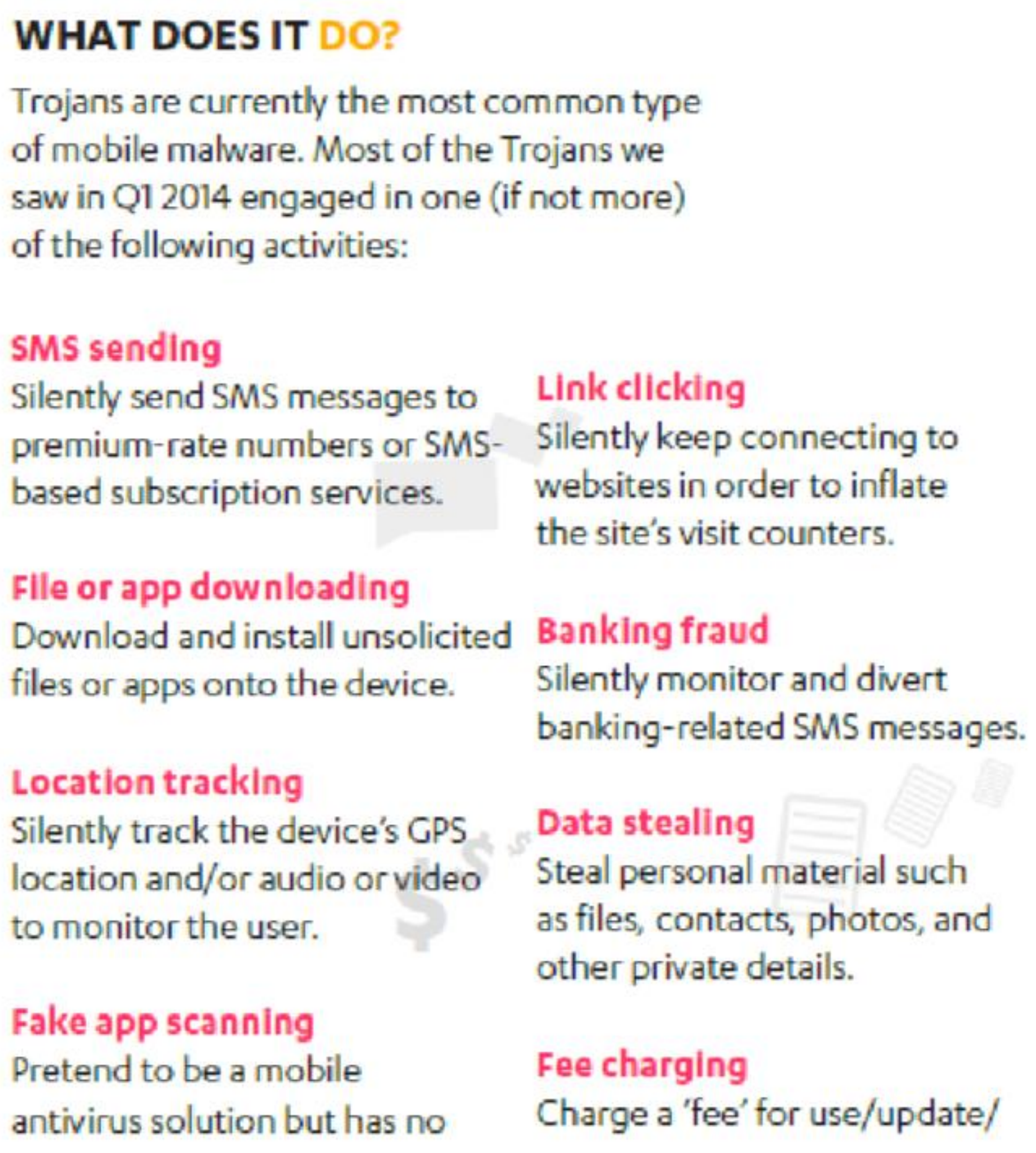
## *The Manifest File*

File which is including all the components details of the application. The manifest does a number of things in addition to declaring the application's components, such as:

1. Identify any user permissions the application requires;
2. Declare the minimum application program interface (API) Level required by the application;
3. Declare hardware and software features used or required by the application;
4. API libraries the application needs to be linked against.

## *Targeting Android*

Android is the target of 99 percent of the world's mobile malware, according to Cisco.[9] Cisco reported that the Android malwares are mainly spread as Trojanised applications designed to look like real, legitimate apps on third-party marketplaces.[9]



**WHAT DOES IT DO?**

Trojans are currently the most common type of mobile malware. Most of the Trojans we saw in Q1 2014 engaged in one (if not more) of the following activities:

**SMS sending**
Silently send SMS messages to premium-rate numbers or SMS-based subscription services.

**Link clicking**
Silently keep connecting to websites in order to inflate the site's visit counters.

**File or app downloading**
Download and install unsolicited files or apps onto the device.

**Banking fraud**
Silently monitor and divert banking-related SMS messages.

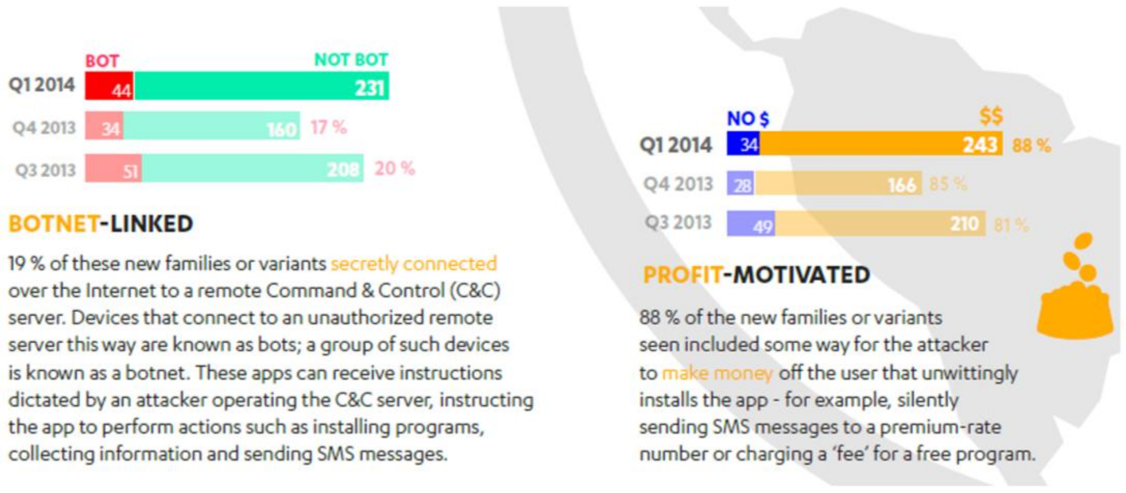
**Location tracking**
Silently track the device's GPS location and/or audio or video to monitor the user.

**Data stealing**
Steal personal material such as files, contacts, photos, and other private details.

**Fake app scanning**
Pretend to be a mobile antivirus solution but has no

**Fee charging**
Charge a 'fee' for use/update/

Figure[9] : 2



**BOTNET-LINKED**

19 % of these new families or variants secretly connected over the Internet to a remote Command & Control (C&C) server. Devices that connect to an unauthorized remote server this way are known as bots; a group of such devices is known as a botnet. These apps can receive instructions dictated by an attacker operating the C&C server, instructing the app to perform actions such as installing programs, collecting information and sending SMS messages.

**PROFIT-MOTIVATED**

88 % of the new families or variants seen included some way for the attacker to make money off the user that unwittingly installs the app - for example, silently sending SMS messages to a premium-rate number or charging a 'fee' for a free program.

Figure[9] : 3

## *What are the existing tools to catch Malware*

The number of automated malware analysis tools can be found in the play.google.com for free as  such as **Anubis** and **Mobile Sandbox**.

These are online tools which generates reports which are giving useful information such as permissions and URLs used by the application. Users can go through the reports and if there is a suspicion that an apk has been cloned and injected with code, then both apks (infected and uninfected apk)  can be uploaded to a sandbox, and compared to discover differences in permissions, used features, URL's etc. So these tools give only a comparison only and further there are limitations in each tools.

## *Andrubis*

Andrubis is the mobile user interface to the Andrubis analysis service. Andrubis executes Android apps in a sandbox and provides a detailed report about their behavior, including file access, network access, cryptographic operations, dynamic code loading and information leaks. In addition to dynamic analysis, Andrubis also performs static analysis, yielding information on e.g. the app's requested and used permissions, activities, services and external libraries. Based on the results of dynamic and static analysis Andrubis assesses the risk associated with an app in the form of a malice score between 0 to 10 (benign to malicious). This assessment is based on features learned from over 100,000 known benign and malicious applications and retrained on a regular basis.[10]

**Note**, that Andrubis only supports apps with a file size smaller than 8MB and an API level of up to 10.

## *Joe Sandbox Mobile*

Joe Sandbox Mobile analyzes APKs in a controlled Android environment and monitors the runtime behavior of the APK for suspicious activities. All activities are compiled into comprehensive and detailed analysis reports.

Analysis reports, which contain key information about potential threats, enable cyber-security professionals to deploy, implement and develop appropriate defense and protections.

Joe Sandbox Mobile enables you to install and use Joe Sandbox in your lab. Currently Joe Sandbox Mobile analyzes any malware targeting Android-based operating systems. [11]

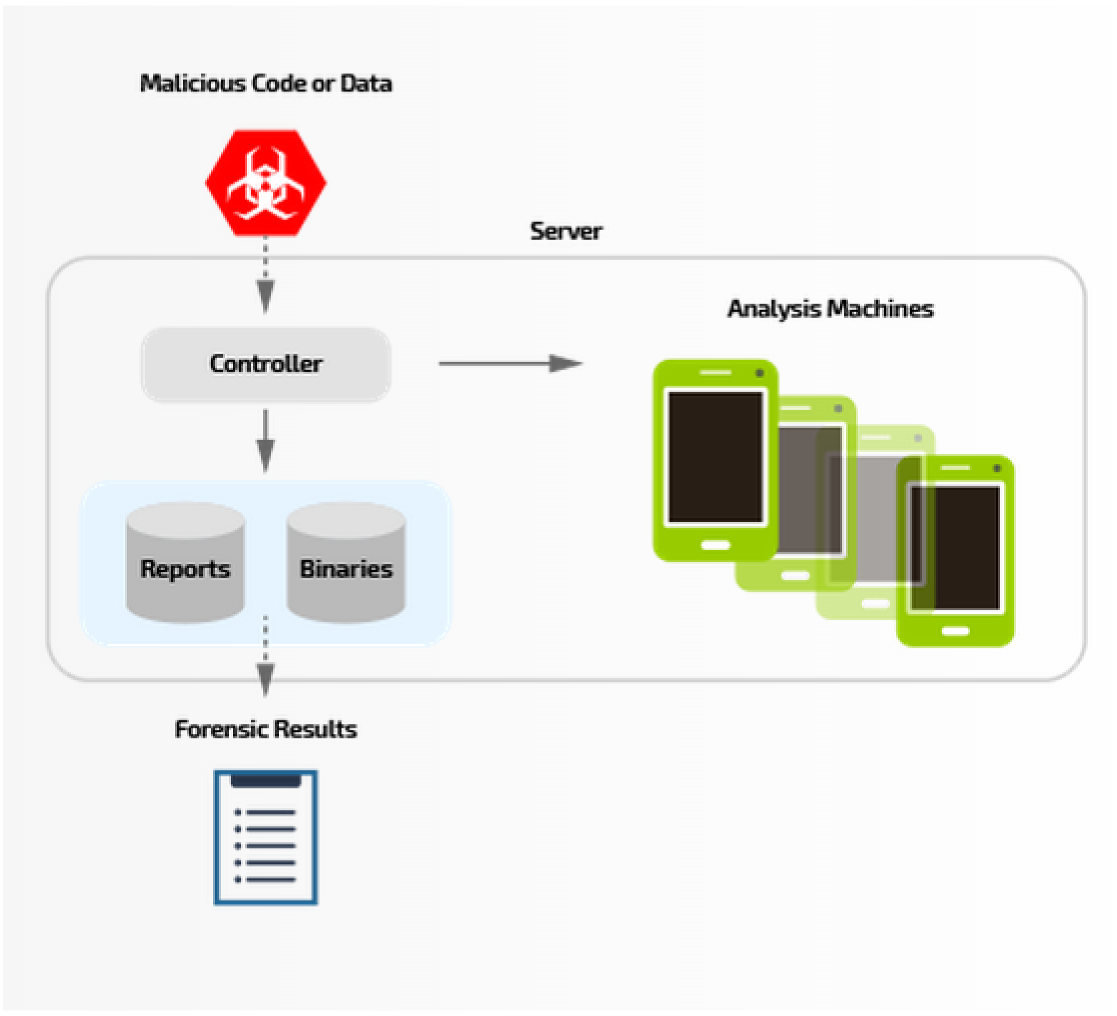


Figure 4:

Joe Sandbox Mobile's architecture is modular. It consists of at least one controller machine running Linux and multiple connected analysis machines (with Android installed) hosted by virtualization products such as VMware or VirtualBox. APKs are submitted from a user or submission scripts and sent via the Joe

Sandbox Mobile Web Interface to the controller's server. The Joe Sandbox Mobile server then stores the submissions in a local file database and forwards them to the connected analysis machines / phone, where the APK is installed and launched.[11]

Joe Sandbox Mobile's configurable and efficient instrumentation engine analyzes any activities during the APK execution and reports back behavior data instantly to the controller. Click to read more about Joe Security's unique technology called Hybrid Code Analysis (HCA) integrated into Joe Sandbox Mobile. [11]

Static and dynamic data is evaluated, and results, statistics, activities and code functions are compiled into a detailed and well-structured report.[11]

## *Inbuilt Android Security*

The in-built security features of Android OS and the latest Android versions I have used here is:

- Android 4.4 KITKAT
- Android 5.0 Lollipop

It is should know that not any operating systems are 100% secure, but we can do so many things to protect our systems, but every protection techniques are not easy, it may giving so much inconvenience to users.

The consideration to be taken to protect your device is depending on following factors.

1. Is your device is valuable
2. Is your device has valuable data/information
3. Location of you is important

### *Passwords:*

It is important to put a strong password to your device prior to taking any further security measurements. Develop a system to create a password and that should not easy to guess by others, the best practices to create strong password should be followed.

## *Dangerous software to steal password:*

## keystroke recorders.

## Other type of attacks to steal password

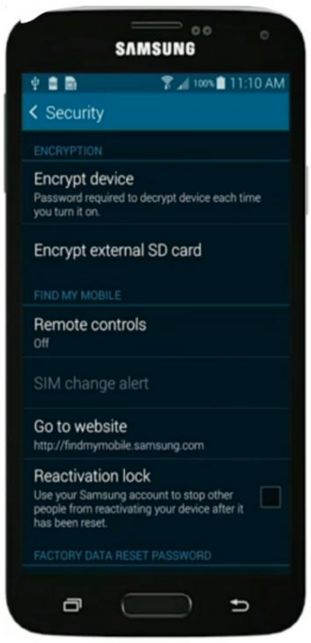
- Brute force attack
- Phishing Scams
- Visual - video, etc.

### *Precautions:*

1. Update your software: The bad guys are alway clever, and they finding security holes, but in the same time we can find fixes for those security holes, so it is necessary to update your OS with necessary updates. You must update both OS update as well as the updates to all the

applications you have installed. Fail to keep your software up to date is could leave you to open the security hole which are discovered and fixed by the software designers. Updating is one easy step to make your device is more secure.

2. Smart lock for trusted devices: Android lollipop version upward there is a feature called smart lock, which enables the Android device even there is a lock is configured when the phone is near the trusted device. So it is not very secure to have trusted device configured if it is not very trusted.
3. Smart lock for trusted places: This is also coming with Android lollipop version, which enables the device to users without putting the lock. So this is also not so secure feature when considering that there are intruders in every places.
4. Encrypting Android device: There is a configuration in  Android OS to encrypt your device and the data when the phone is locked, this is a cool feature that will protect the data in the Android OS when the device is connected to the Computer systems.



ENCRYPTION

Encrypt device
Password required to decrypt device each time you turn it on.

Encrypt external SD card

FIND MY MOBILE

Remote controls
Off

SIM change alert

Go to website
http://findmymobile.samsung.com

Reactivation lock
Use your Samsung account to stop other people from reactivating your device after it has been reset.

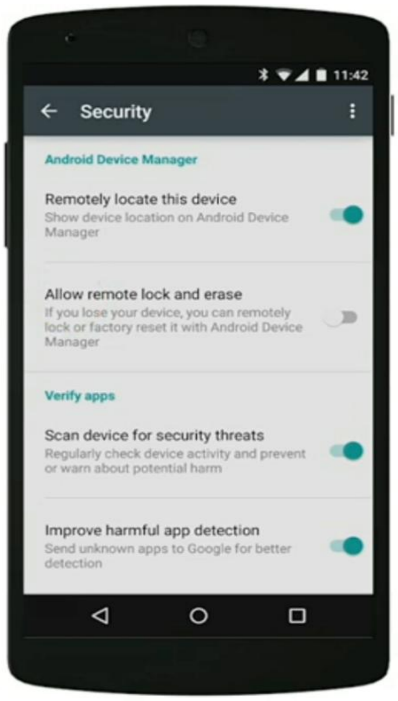
FACTORY DATA RESET PASSWORD

Figure 5:

- Enabling Android device manager: There are very handy two features giving by Android operating system that is to Remotely locate this device and Allow remote lock and erase options. Enabling these features are very handy when your device is lost or cannot find it. If your device is “On” and connected to the network, then it is easy to locate the device, putting a new pass codes and lock it, or wipe out all the data.

*How to do it?*

You simply log in to the Google play and go to settings to access device manager, in there you can find out your device and take necessary security actions to protect your device or data.


11:42

Security

Android Device Manager

Remotely locate this device
Show device location on Android Device Manager

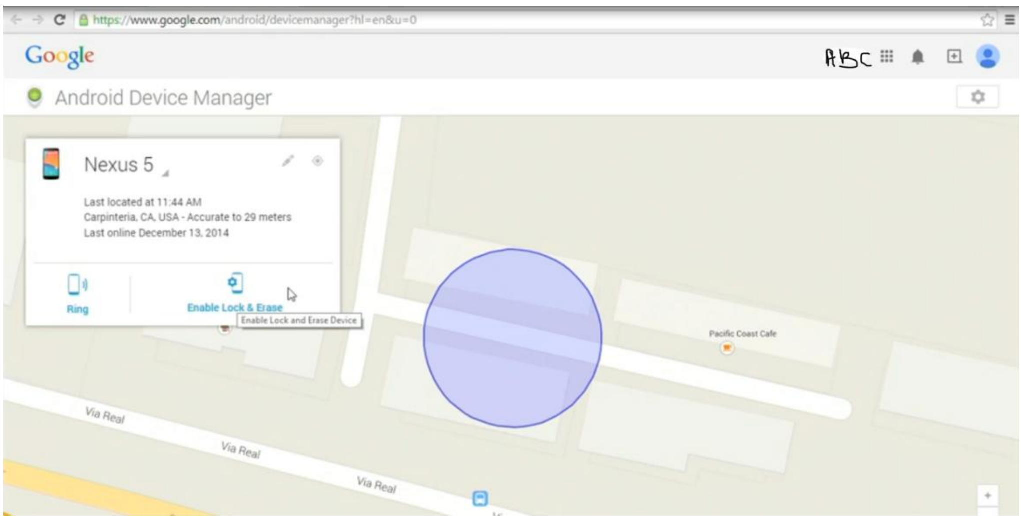
Allow remote lock and erase
If you lose your device, you can remotely lock or factory reset it with Android Device Manager

Verify apps

Scan device for security threats
Regularly check device activity and prevent or warn about potential harm

Improve harmful app detection
Send unknown apps to Google for better detection

Figure 6:


https://www.google.com/android/devicemanager?hl=en&u=0

Google

Android Device Manager

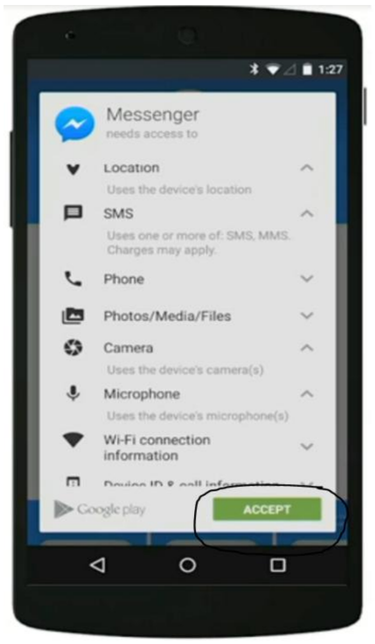
Nexus 5

Last located at 11:44 AM
Carpinteria, CA, USA - Accurate to 29 meters
Last online December 13, 2014

Figure 7:

- Restricting and get to know what app is accessing your personal data: Can you ever trust the developers who made these apps that they don not use your personal data for other purposes? i.e the facebook app can access your contact details etc. So what are the features giving to restrict these access or see what type of data is accessed by the application. Android is giving these option to user. In the following example I show that facebook messenger works on the Android OS.

- When you installing facebook messenger you can see a user accept permission screen. In that you can see what are the required information to install this application.



1:27

Messenger
needs access to

Location
Uses the device's location

SMS
Uses one or more of: SMS, MMS. Charges may apply.

Phone

Photos/Media/Files

Camera
Uses the device's camera(s)

Microphone
Uses the device's microphone(s)

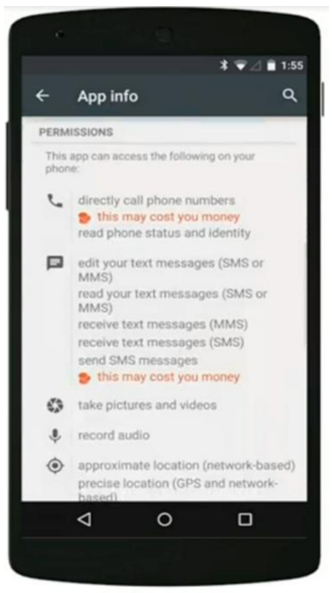
Wi-Fi connection information

Device ID & call information

Figure 8:

Further you can confirm after installing the application whether that application has accessed to particular information only. For example in the facebook messenger, it has said that your location is being monitored by the application. So what you need to do it to go to settings > location > in that screen you can see all the list of applications has access to your location. Then you can find the fb messenger and find out what are the accessing information from this application.

The application is not allowed to restrict each accessibility to information, so then you need to either uninstall the full application or allow it to access it. But the Android is giving option to turn off the location setting from the whole device, and this is a global setting. So in that you can restrict some personal information from the application.



PERMISSIONS

This app can access the following on your phone:

directly call phone numbers
this may cost you money
read phone status and identity

edit your text messages (SMS or MMS)
read your text messages (SMS or MMS)
receive text messages (MMS)
receive text messages (SMS)
send SMS messages
this may cost you money

take pictures and videos

record audio

approximate location (network-based)
precise location (GPS and network-based)

Figure 9 :

You should know that google is tracking your location through the phone, so anyone can able to access to your google account, then he can see what locations you have visited from the each day. Is this not worrying you? How many of us know this. And how many of us know how to stop this tracking.

Google account synchronized automatically of your certain information (example location) and share with other applications.

**Example:**

When you do a Google search then the suggestion are automatically comes 1st to related to your location.

When you search xyz certifications, then it will give suggestions to xyz certification to your home town. How can that know your location? Further Google can give you the traffic report in the morning that suggested that your location is keep tracked by the Google. This is a handy feature, but how do you know that Google is keeping these information in safe? Or else some other person can access to your google account and find out your location.

Do you need to disable this tracking? and delete past location history? So this is allowed from the Android OS, by you can go to “Google settings” > account history > Google location history > location history : there you can delete location history or view your location history. Same information can be found when log in to the Google via web browser and search maps.google.com/location history will show your tracked information.

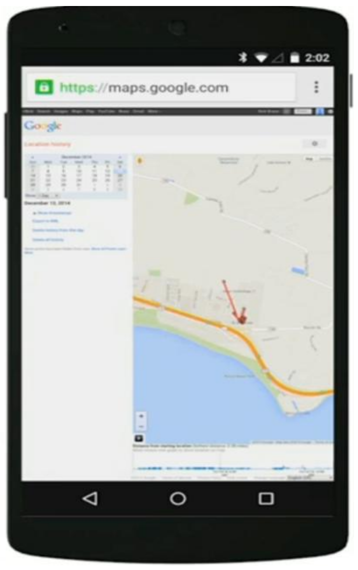


Figure 10:

That is only for location, so what else? yes it will track your browser history, YouTube history etc.

Turning off browser history and search history: incognito mode

You can entirely switch off from Google chrome account and stop syncing your browser history; else there is an option inbuilt in Android which is called Incognito Mode.

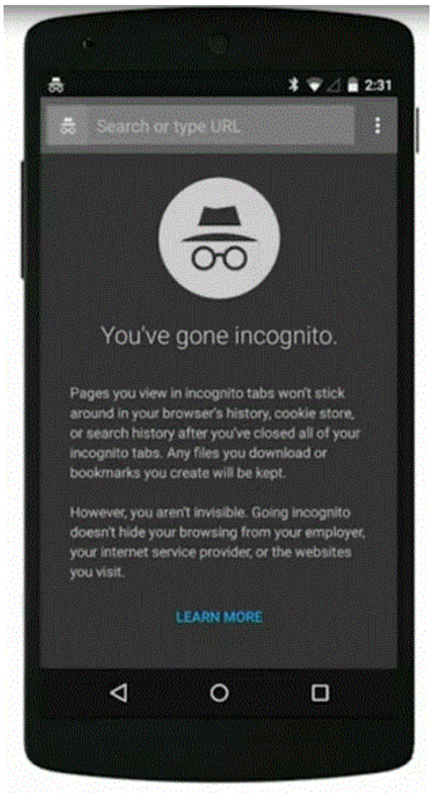
If you don't want Google Chrome to save a record of what you visit and download, you can browse the web in incognito mode.

## *Use incognito mode for private browsing*

You can open an incognito window on your computer and mobile devices to prevent Chrome from saving your browsing history.

How to enable incognito mode for Android device?

- Open a Chrome window
- Touch the Chrome menu, which may look like either three dots Menu or a tab Menu.
- Touch new incognito tab.
- A new window will open with a gray incognito icon.
- To close incognito mode:
- If you have Android 5.0 (Lollipop) or later, swipe from the top and touch Chrome: Close all incognito windows.
- If you have an earlier version of Android, go to the corner of each of your incognito windows and touch the X.



You've gone incognito.

Pages you view in incognito tabs won't stick around in your browser's history, cookie store, or search history after you've closed all of your incognito tabs. Any files you download or bookmarks you create will be kept.

However, you aren't invisible. Going incognito doesn't hide your browsing from your employer, your internet service provider, or the websites you visit.

Figure 11:

## Research Methodology

Android runtime (ART) is the managed runtime used by applications and some system services on Android. ART and its predecessor Dalvik were originally created specifically for the Android project. ART as the runtime executes the Dalvik Executable format and Dex bytecode specification. ART and Dalvik are compatible runtimes running Dex bytecode, so apps developed for Dalvik should work when running with ART. However, some techniques that work on Dalvik do not work on ART.

ART gives you as much context and detail as possible when runtime exceptions occur. ART provides expanded exception detail for java.lang.ClassCastException, java.lang.ClassNotFoundException, andjava.lang.NullPointerException. (Later versions of Dalvik provided expanded exception detail for java.lang.ArrayIndexOutOfBoundsException and java.lang.ArrayStoreException, which now include the size of the array and the out-of-bounds offset, and ART does this as well.) For example, java.lang.NullPointerException now shows information about what the app was trying to do with the null pointer, such as the field the app was trying to write to, or the method it was trying to call.

# Steps Followed in Research Methodology

1. Fulfil the knowledge GAP - learning Android development (Udacity Free Course)

Identified knowledge gap should be fulfilled by following a free course conducted by udacity. The estimated time to complete this course is one month. There I should able to setup the android development environment as well as the familiarized for the tools used in development.

2. Analyzing the existing features of DroidBox.

DroidBox is a tool we need to learn how it works. It is not easy to understand and learn it's functionality without having proper documentations. This is a trial and error task and it is a big challenge to compile the source code and build the Droidbox. This task will take nearly one month to complete.

3. Identifying existing features and Test planning.

1st create a test plan including details of how I should test the Droidbox then after create Test cases to cover the functionality of the DroidBox. It should cover all the identified features. The test data is not available in the web freely. Therefore I will create my own malware applications in order to test the application. There I will use Android Development knowledge gain in task 1.

4. Test Execution

The estimated time for test execution is 1 week. The test result should be properly documented and store for further reference.

5. Start Coding

Then I will work with the Coding and start work with the Development. The Development effort will be approximately 3 months starting. The sub tasks are as follows.

a. Upgrade ART environment
b. Compiling the Droidbox in ART platform and fixing issues.
c. Execute developer test cases to fix bugs

6. Test Execution

7. Bug Fixing

8. Regression Testing

## How to download DroidBox Source Code?

1st I will download the Droidbox source code from the following repository:

Download necessary files and un compress it anywhere

```
wget http://droidbox.googlecode.com/files/DroidBox411RC.tar.gz
```

Prerequisites : PyLab, sdk android

### *Installing pylab*

```
$ sudo apt-get install git

$ sudo apt-get install python-dev

$ git clone git://github.com/numpy/numpy.git numpy

$ cd /Downloads/numpy

$ python setup.py install –user

$ cd /Downloads/scipy

$ python setup.py install –user

$ sudo apt-get install python-matplotlib
```

PyLab is a module that belongs to the Python mathematics library Matplotlib. PyLab combines the numerical module numpy with the graphical plotting module pyplot. PyLab was designed with the interactive Python interpreter in mind, and therefore many of its functions are short and require minimal typing. This makes it a very efficient and convenient mathematical tool. If you want to install PyLab, you must also install Matplotlib. This process is fairly simple and follows the standard installation procedure of any application.[3]

## Setting up the development Environment

### *Install Java version 8*

Download a .tar.gz from Oracle
Extract it into somewhere

```
$sudo mv /path/to/jdk1.8.0_20 /usr/lib/jvm/oracle_jdk8
```

Before adding this jdk as an alternative, you can see that the new alternative is not listed:

```
$sudo update-alternatives --query javac
$sudo update-alternatives --query java
```

Next, add the new jdk alternatives (2000 is the priority and feel free to pick a different number):

```
$sudo update-alternatives --install /usr/bin/java java
/usr/lib/jvm/oracle_jdk8/jre/bin/java 2000

$sudo update-alternatives --install /usr/bin/javac javac
/usr/lib/jvm/oracle_jdk8/bin/javac 2000
```

Now you should see the new jdk listed and you can switch between the alternatives with this command:

```
$sudo update-alternatives --config java
$sudo update-alternatives --config javac
```

Create a file /etc/profile.d/oraclejdk.sh with the following content (adapt the paths to reflect the path where you stored your JDK):

```
export J2SDKDIR=/usr/lib/jvm/oracle_jdk8 export J2REDIR=/usr/lib/jvm/oracle_jdk8/jre
export
PATH=$PATH:/usr/lib/jvm/oracle_jdk8/bin:/usr/lib/jvm/oracle_jdk8/db/bin:/usr/lib/jvm
/oracle_jdk8/jre/bin export JAVA_HOME=/usr/lib/jvm/oracle_jdk8 export
DERBY_HOME=/usr/lib/jvm/oracle_jdk8/db
```

## 2. *Download Android SDK*

Login to Ubuntu terminal and type

```
$wget http://dl.google.com/android/android-sdk_r24.4.1-linux.tgz
```

Unzip the tgz file you need execute the install.sh script

```
$./install.sh
```

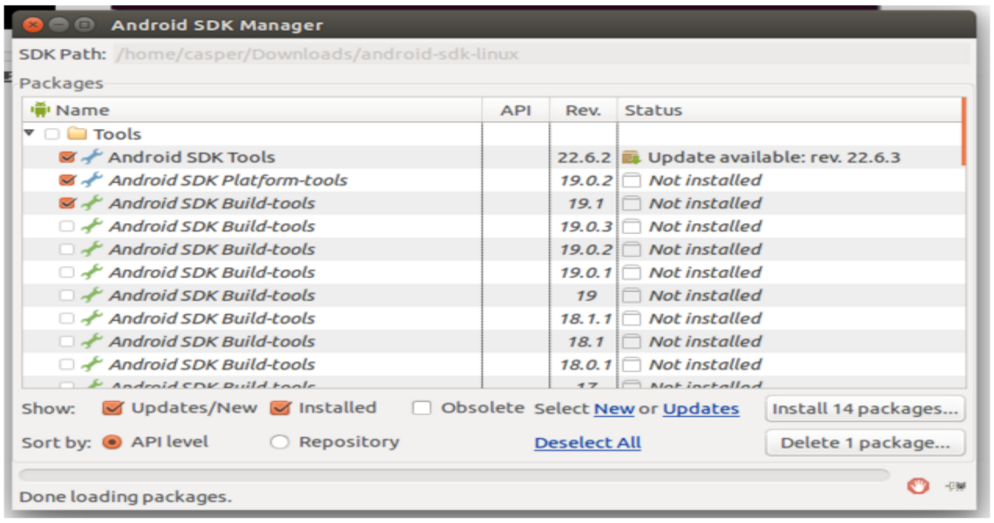
Setting up the environment variables (adding following entries in .bashrc file)

```
export ANDROID_HOME=/home/rukmal/Android/Sdk
export PATH=${PATH}:$ANDROID_HOME/tools:$ANDROID_HOME/platform-tools
```

## *Installing Android Packages*

Start SDK manager

```
$./android
```



Android SDK Manager

SDK Path: /home/casper/Downloads/android-sdk-linux

Packages

| Name | API | Rev. | Status |
|---|---|---|---|
| Tools | | | |
| ✓ Android SDK Tools | | 22.6.2 | Update available: rev. 22.6.3 |
| ✓ Android SDK Platform-tools | | 19.0.2 | Not installed |
| ✓ Android SDK Build-tools | | 19.1 | Not installed |
| Android SDK Build-tools | | 19.0.3 | Not installed |
| Android SDK Build-tools | | 19.0.2 | Not installed |
| Android SDK Build-tools | | 19.0.1 | Not installed |
| Android SDK Build-tools | | 19 | Not installed |
| Android SDK Build-tools | | 18.1.1 | Not installed |
| Android SDK Build-tools | | 18.1 | Not installed |
| Android SDK Build-tools | | 18.0.1 | Not installed |
| Android SDK Build-tools | | 17 | Not installed |

Show: [✓ Updates/New] [✓ Installed] Obsolete Select New or Updates Install 14 packages...

Sort by: [✓ API level] Repository Deselect All Delete 1 package...

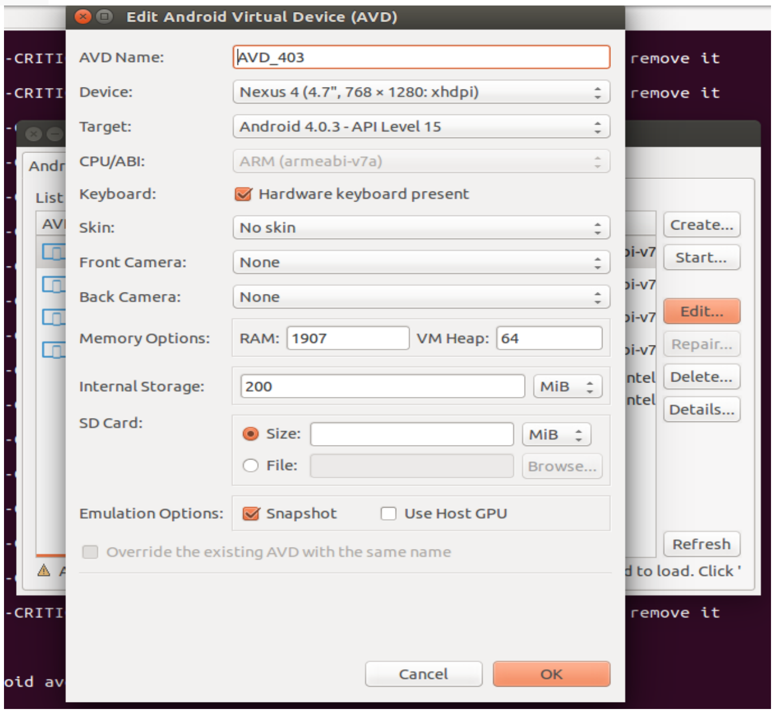
Done loading packages.

Android SDK Manager

In this window you can select necessary packages and SDK tools we need. I have to select following tools and packages to run the Droid Box.

Tools >>
Android SDK Tools (Rev. 24.4.1)
Android SDK Platform-tools (Ref. 23.1)
Android version >>
Android 4.2.2 (API 17)
Android 4.1.2 (API 16)
Android 4.0.3 (API 15)
Setting Up an Android Virtual Device

```
$./android avd
```


Edit Android Virtual Device (AVD)

AVD Name: AVD_403
Device: Nexus 4 (4.7", 768 × 1280: xhdpi)
Target: Android 4.0.3 - API Level 15
CPU/ABI: ARM (armeabi-v7a)
Keyboard: ✓ Hardware keyboard present
Skin: No skin
Front Camera: None
Back Camera: None
Memory Options: RAM: 1907 VM Heap: 64
Internal Storage: 200 MiB
SD Card: ● Size: MiB ○ File:
Emulation Options: ✓ Snapshot ☐ Use Host GPU
☐ Override the existing AVD with the same name

Android AVD Setup

In this screen we have to create an emulator for droid box. Target android version should be 4.0.3. There should be SD card and the size of the internal storage is 200 Mib.

## Setting up Droidbox and Emulator

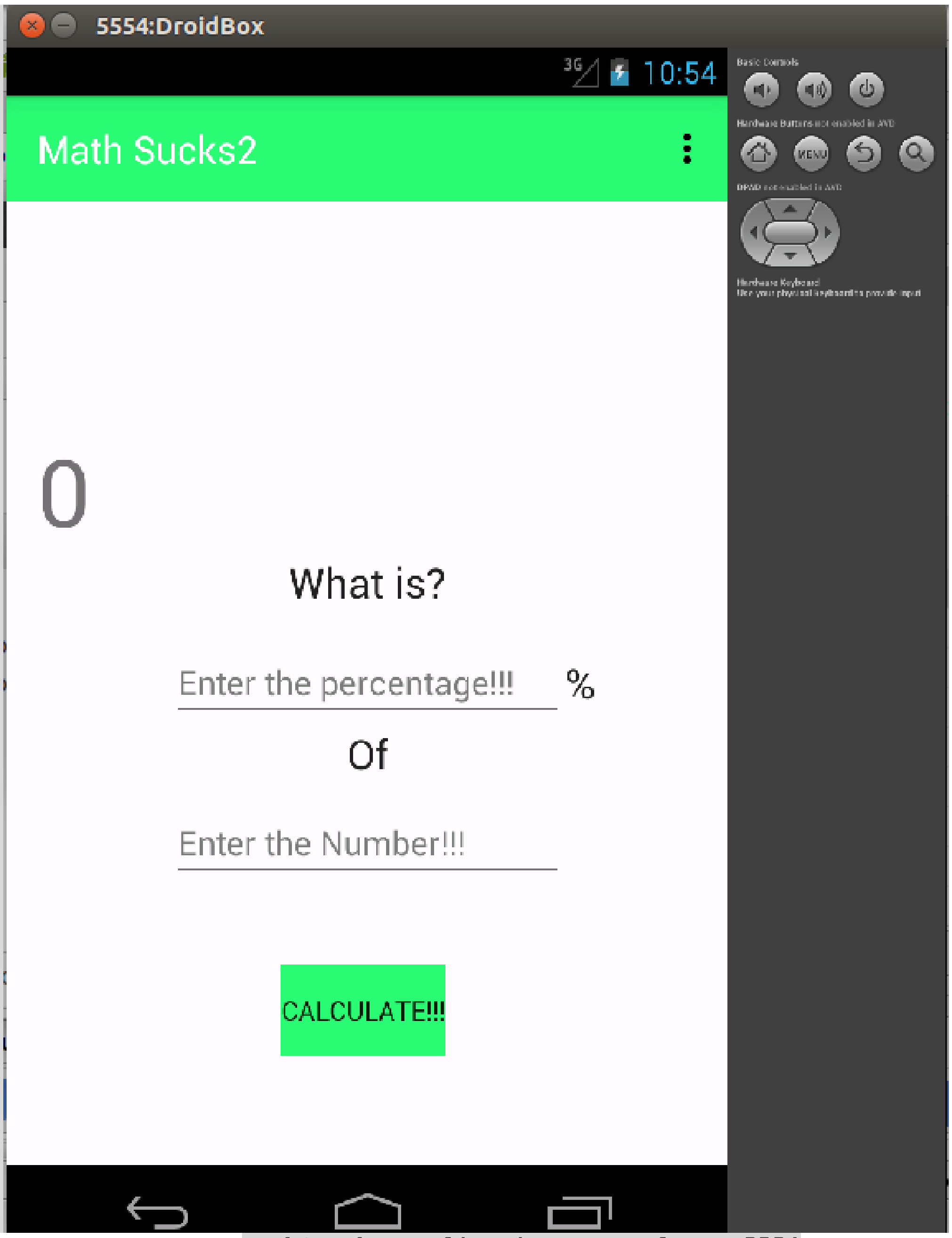
Create an AVD (Nexus 4 with Android 4.1.2) and launch it using the script located in the newly unpacked folder in Droidbox distribution.

```
$ ./startemu.sh DroidBox
```

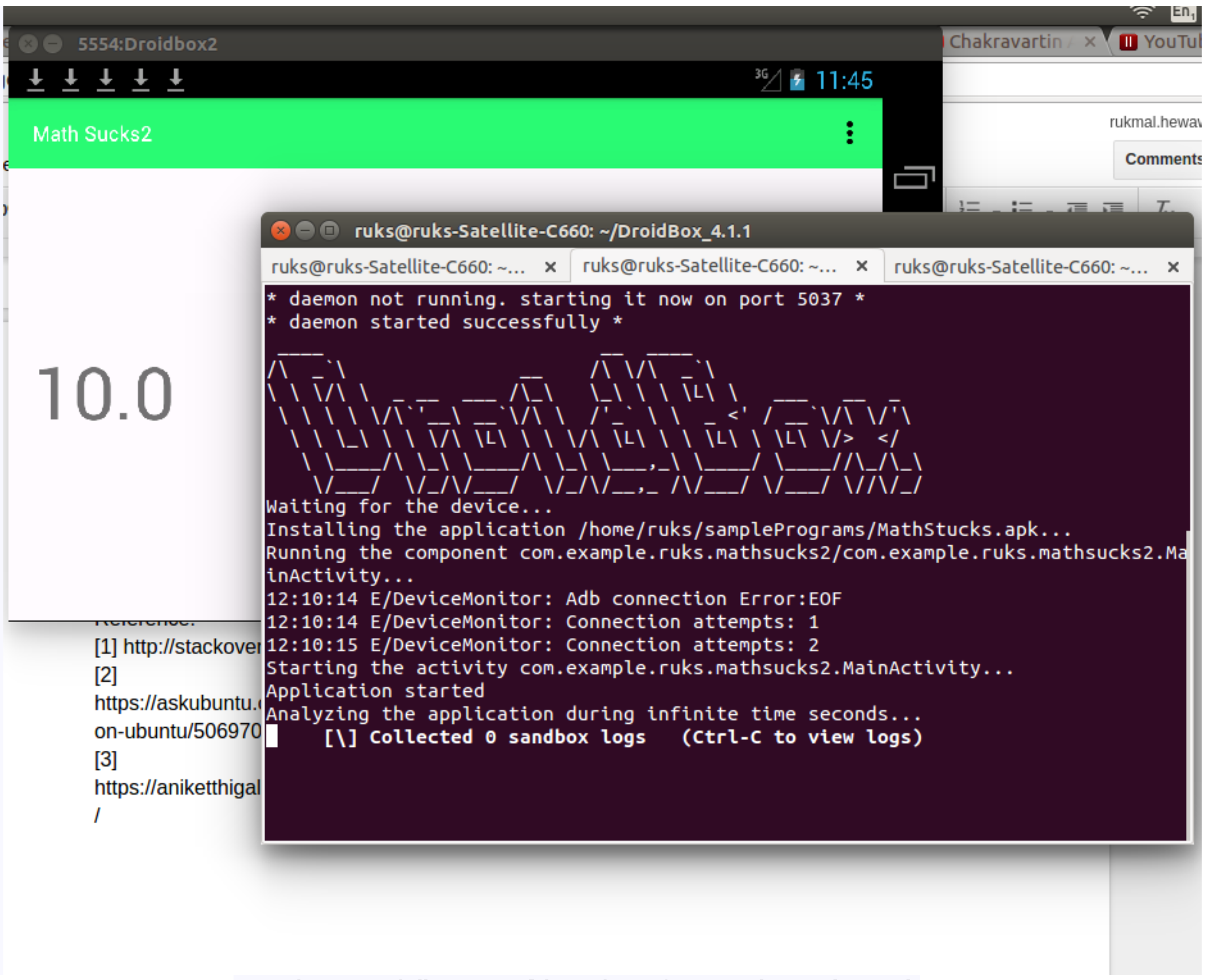
This script takes the name of the newly created AVD as a parameter, with this ultimately launching the system image that comes with DroidBox rather than the original system image.

```
$ cat startemu.sh
#!/usr/bin/env bash
emulator -avd Droidbox  -prop dalvik.vm.execution-mode=int:portable &
```
Execute MathStucks.apk on Emulator

```
$droidbox.sh  /home/ruks/samplePrograms/MathStucks.apk
```

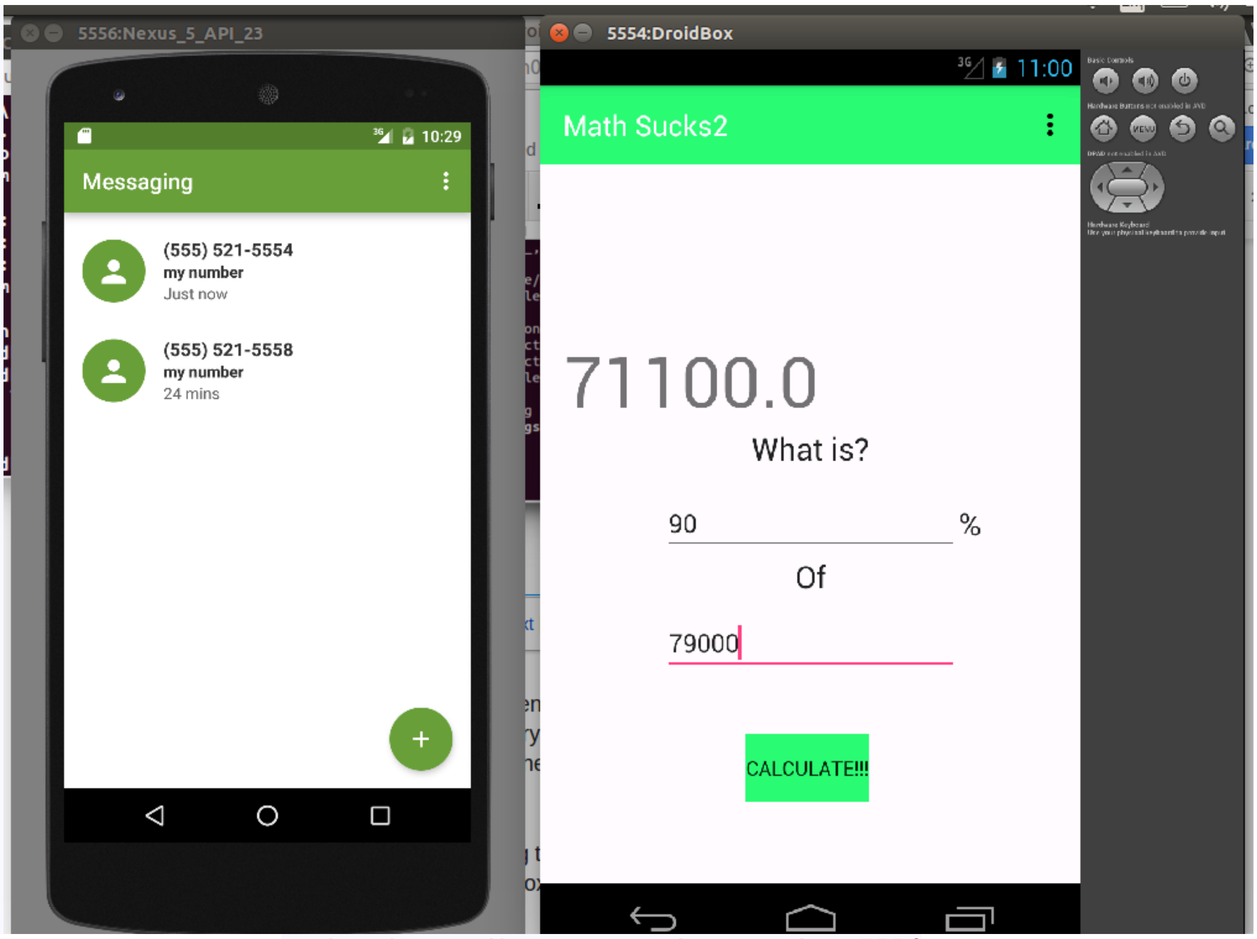


MathStucks Application on Emulator 5554


5554:Droidbox2
11:45
Math Sucks2
10.0
ruks@ruks-Satellite-C660: ~/DroidBox_4.1.1
ruks@ruks-Satellite-C660: ~...
ruks@ruks-Satellite-C660: ~...
ruks@ruks-Satellite-C660: ~...
Chakravartin
YouTu
rukmal.hewa
Comments

```
* daemon not running. starting it now on port 5037 *
* daemon started successfully *
 ____                        __  ____
/\  _`\               __    /\ \/\  _`\
\ \ \/\ \  _ __  ___ /\_\   \_\ \ \ \L\ \    ___   __  _
 \ \ \ \ \/\`'__\ __`\/\ \  /'_` \ \  _ <' / __`\/\ \/'\
  \ \ \_\ \ \ \/\ \L\ \ \ \/\ \L\ \ \ \L\ \ \L\ \/>  </
   \ \____/\ \_\ \____/\ \_\ \___,_\ \____/ \____//\_/\_\
    \/___/  \/_/\/___/  \/_/\/__,_ /\/___/ \/___/ \//\/_/
Waiting for the device...
Installing the application /home/ruks/samplePrograms/MathStucks.apk...
Running the component com.example.ruks.mathsucks2/com.example.ruks.mathsucks2.Ma
inActivity...
12:10:14 E/DeviceMonitor: Adb connection Error:EOF
12:10:14 E/DeviceMonitor: Connection attempts: 1
12:10:15 E/DeviceMonitor: Connection attempts: 2
Starting the activity com.example.ruks.mathsucks2.MainActivity...
Application started
Analyzing the application during infinite time seconds...
     [\] Collected 0 sandbox logs   (Ctrl-C to view logs)
```

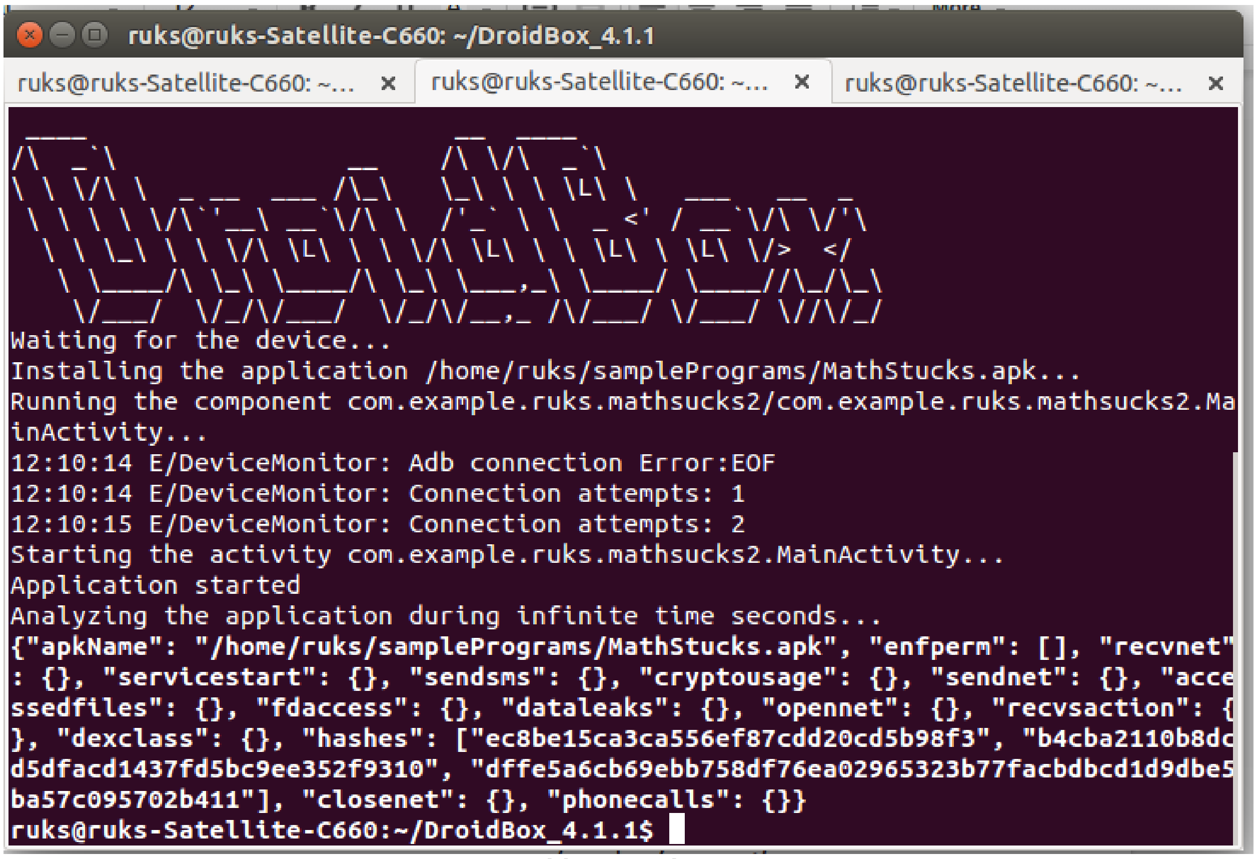
Running Droidbox application for MathStucks.apk



MathSucks sending SMS to phone number 5556

The emulator window will show the main screen of the application that will reopen after every attempt to close it. Wait for some time, try to do calculator to calculate percentage value of something and press Calculate button, and finally kill the emulator by pressing <Ctrl + C> in the terminal window.

As a result it should show three files according to the Droidbox readme. But I cannot see there are three files when I run the Droidbox and it only display the log.



```
 ____                 __  ____
/\  _`\               __           /\ \/\  _`\
\ \ \/\ \  _ __   ___ /\_\    \_\ \ \ \L\ \    ___   __  _
 \ \ \ \ \/\`'__\/ __`\/\ \   /'_` \ \  _ <' / __`\/\ \/'\
  \ \ \_\ \ \ \//\ \L\ \ \ \/\ \L\ \ \ \L\ \/\ \L\ \/>  </
   \ \____/\ \_\\ \____/\ \_\ \___,_\ \____/\ \____//\_/\_\
    \/___/  \/_/ \/___/  \/_/\/__,_ /\/___/  \/___/ \//\/_/
Waiting for the device...
Installing the application /home/ruks/samplePrograms/MathStucks.apk...
Running the component com.example.ruks.mathsucks2/com.example.ruks.mathsucks2.Ma
inActivity...
12:10:14 E/DeviceMonitor: Adb connection Error:EOF
12:10:14 E/DeviceMonitor: Connection attempts: 1
12:10:15 E/DeviceMonitor: Connection attempts: 2
Starting the activity com.example.ruks.mathsucks2.MainActivity...
Application started
Analyzing the application during infinite time seconds...
{"apkName": "/home/ruks/samplePrograms/MathStucks.apk", "enfperm": [], "recvnet"
: {}, "servicestart": {}, "sendsms": {}, "cryptousage": {}, "sendnet": {}, "acce
ssedfiles": {}, "fdaccess": {}, "dataleaks": {}, "opennet": {}, "recvsaction": {
}, "dexclass": {}, "hashes": ["ec8be15ca3ca556ef87cdd20cd5b98f3", "b4cba2110b8dc
d5dfacd1437fd5bc9ee352f9310", "dffe5a6cb69ebb758df76ea02965323b77facbdbcd1d9dbe5
ba57c095702b411"], "closenet": {}, "phonecalls": {}}
ruks@ruks-Satellite-C660:~/DroidBox_4.1.1$ 
```

**DroidBox Script**

For some reason unknown to me, the main script DroidBox in the archive does not generate graphics, and the old script that is located in the SVN-repository in the external folder does not work. Since time constraints I will not try to generate graphics in this project.[4]

## Analyzing the DroidBox logs

```
{"apkName": "/home/ruks/samplePrograms/MathStucks.apk", "enfperm": [],
"recvnet": {}, "servicestart": {}, "sendsms": {}, "cryptousage": {}, "sendnet":
{}, "accessedfiles": {}, "fdaccess": {}, "dataleaks": {}, "opennet": {},
"recvsaction": {}, "dexclass": {}, "hashes":
["ec8be15ca3ca556ef87cdd20cd5b98f3",
"b4cba2110b8dcd5dfacd1437fd5bc9ee352f9310",
"dffe5a6cb69ebb758df76ea02965323b77facbdbcd1d9dbe5ba57c095702b411"],
"closenet": {}, "phonecalls": {}}
```

The DroidBox tracing file is a record of actions in JSON format that has the following sections:

- accessedFiles — a list of files the application received access to;
- apkName — the name of the APK-file being analyzed;
- closenet — socket close operations;
- cryptousage — operations involving cryptAPIAndroid;
- dataleaks — leak of the user's personal data;
- dexclass — operations with DEX classes;
- enfperm — authorizations added (not used!) by the application;
- fdaccess — operations involving files;
- hashes — MD5-, SHA–1- and SHA–256-hashes of the analyzed APK;
- opennet — socket open operations;

- phonecalls — phone calls;
- recvnet — receive via network;
- recvaction — a list of intents to which the application responds;
- sendnet — transfer operations via network;
- sendsms — message sending;
- servicestart — service launch operations.

Almost all sections have the following format:

```
"Section name" {
  "Operation time (from start of launch)" {
     "Name of parameter being traced (e.g., for sendnet, this could be desthost, destport, etc.)":
"Parameter value"
  }
}
```

Although my application sending SMS to un-konwn number, we cannot see it from analysing the Droidbox logs. There I can see that “sendSms” is empty. (sendsms": {}). It means that Droidbox which is available in the git Repository is not working as expected.

## Troubleshooting Droidbox issues

Need to download Android source code to troubleshoot the Emulator issue faced during the analysis.

Following section shows how to download Android source code and the environment.

### *Setting up a Linux build environment [8]*

Ubuntu LTS (14.04), but most distributions should have the required build tools available.
There are no available supported OpenJDK 8 packages for Ubuntu 14.04. The Ubuntu 15.04 OpenJDK 8 packages have been used successfully with Ubuntu 14.04.

### *Installing required packages (Ubuntu 14.04)*

```
$ sudo apt-get install git-core gnupg flex bison gperf build-essential \

 zip curl zlib1g-dev gcc-multilib g++-multilib libc6-dev-i386 \

 lib32ncurses5-dev x11proto-core-dev libx11-dev lib32z-dev ccache \

 libgl1-mesa-dev libxml2-utils xsltproc unzip
```

To use SELinux tools for policy analysis, also install the python-networkx package

```
sudo apt-get update
sudo apt-get install python-networkx
```

### *Configuring USB Access*

Under GNU/Linux systems (and specifically under Ubuntu systems), regular users can't directly access USB devices by default. The system needs to be configured to allow such access.
The recommended approach is to create a file at /etc/udev/rules.d/51-android.rules (as the root user).

To do this, run the following command to download the 51-android.rules file attached to this site, modify it to include your username, and place it in the correct location:

```
$ wget -S -O - http://source.android.com/source/51-android.rules | sed "s/<username>/$USER/" | sudo tee >/dev/null /etc/udev/rules.d/51-android.rules; sudo udevadm control --reload-rules
```

## *Installing Repo*

Repo[9] is a tool that makes it easier to work with Git in the context of Android.
Repo is a repository management tool that built on top of Git. Repo unifies the many Git repositories when necessary, does the uploads to Google's revision control system, and automates parts of the Android development workflow. Repo is not meant to replace Git, only to make it easier to work with Git in the context of Android. The repo command is an executable Python script that you can put anywhere in your path. In working with the Android source files, you will use Repo for across-network operations. For example, with a single Repo command you can download files from multiple repositories into your local working directory.

You can install it by typing:

```
sudo apt-get install phablet-tools
```

## *Initializing a Repo client*

After installing Repo, set up your client to access the Android source repository:

Create an empty directory to hold your working files.

```
$ mkdir WORKING_DIRECTORY
$ cd WORKING_DIRECTORY
```

Configure git with your real name and email address.

```
$ git config --global user.name "rukmalhe"
$ git config --global user.email "rukmal.hewawasam@gmail.com"
```

Run repo init to bring down the latest version of Repo with all its most recent bug fixes. You must specify a URL for the manifest, which specifies where the various repositories included in the Android source will be placed within your working directory.

```
$ repo init -u https://android.googlesource.com/platform/manifest
```

To check out a branch other than "master", specify it with -b. For a list of branches, see Source Code Tags and Builds.

```
$repo init -u https://android.googlesource.com/platform/manifest -b android-4.1.1_r6
```

## *Downloading the Android Source Tree*

```
$ repo sync
```

The Android source files will be located in your working directory under their project names. The initial sync operation will take an hour or more to complete.

## *Issues faced while download android*

http://stackoverflow.com/questions/30071064/android-repo-sync-fatal-error
More rarely, Linux clients experience connectivity issues, getting stuck in the middle of downloads (typically during "Receiving objects"). It has been reported that tweaking the settings of the TCP/IP stack and using non-parallel commands can improve the situation. You need root access to modify the TCP setting:

```
sudo sysctl -w net.ipv4.tcp_window_scaling=0

repo sync -j1
```

## Step By Step Analysis malware sample - Static Analysis

Let’s see how we can identify the malformed behaviour on MathStuck.apk file.

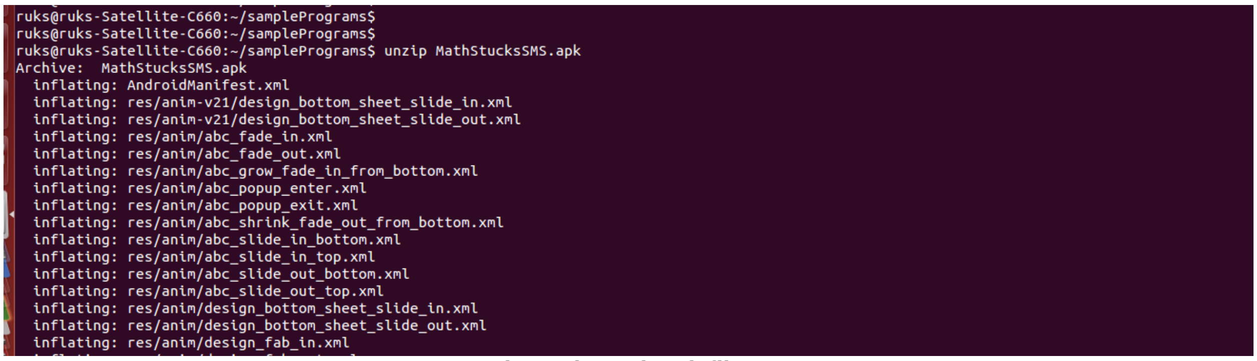
It goes without saying that malware analysis should never be performed in a production environment, as you never know what you will be dealing with, and we certainly do not want to infect our own machines! For this project, I know that my MathStuck is not doing big harm to the environment; otherwise I need to use a Backtrack Virtual Machine. Although the image already contains quite some tools that are ready-suitable for the analysis of APK archives. I will use following tools in my Linux environment.

- android-apktool[5]
- Dex2jar [6]
- jd-gui (java decompiler) [7]

Once you have these up and running, it's time to get our hands dirty!

### Checking Android Manifest File

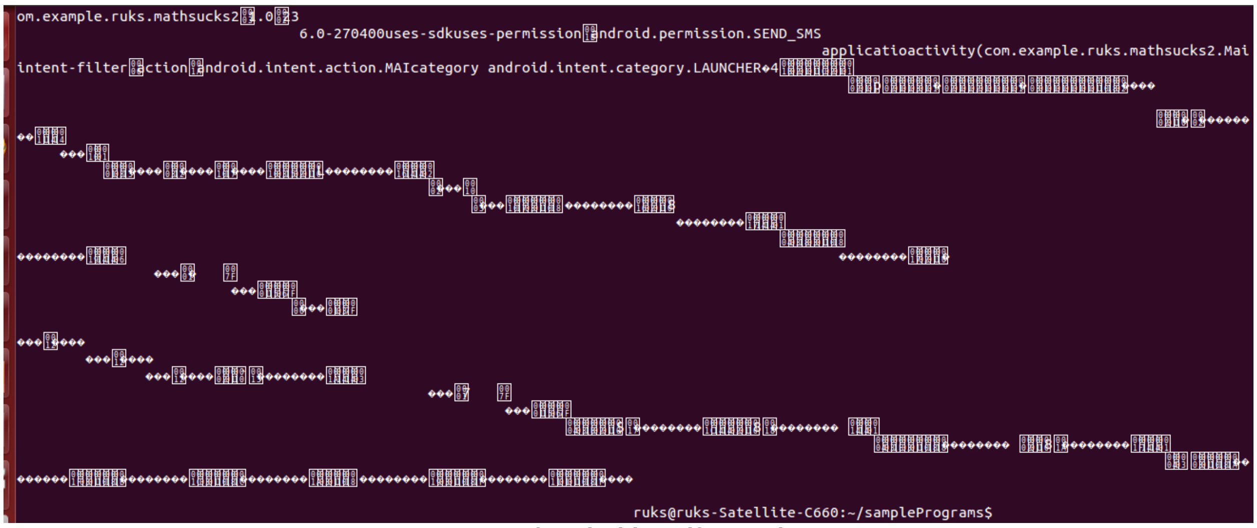
The first thing we do is figure out if the sample we are analyzing has the correct format. APK packages are nothing more than ZIP files with a predefined structure (including for example a manifest file).



```
ruks@ruks-Satellite-C660:~/samplePrograms$
ruks@ruks-Satellite-C660:~/samplePrograms$
ruks@ruks-Satellite-C660:~/samplePrograms$ unzip MathStucksSMS.apk
Archive:  MathStucksSMS.apk
  inflating: AndroidManifest.xml
  inflating: res/anim-v21/design_bottom_sheet_slide_in.xml
  inflating: res/anim-v21/design_bottom_sheet_slide_out.xml
  inflating: res/anim/abc_fade_in.xml
  inflating: res/anim/abc_fade_out.xml
  inflating: res/anim/abc_grow_fade_in_from_bottom.xml
  inflating: res/anim/abc_popup_enter.xml
  inflating: res/anim/abc_popup_exit.xml
  inflating: res/anim/abc_shrink_fade_out_from_bottom.xml
  inflating: res/anim/abc_slide_in_bottom.xml
  inflating: res/anim/abc_slide_in_top.xml
  inflating: res/anim/abc_slide_out_bottom.xml
  inflating: res/anim/abc_slide_out_top.xml
  inflating: res/anim/design_bottom_sheet_slide_in.xml
  inflating: res/anim/design_bottom_sheet_slide_out.xml
  inflating: res/anim/design_fab_in.xml
```

**Unzip MathStuck.apk file**

We can see AndroidManifest.xml file but when you view it’s content then I can see it is not readable.



```
om.example.ruks.mathsucks2 .0 3
                                    6.0-270400uses-sdkuses-permission android.permission.SEND_SMS
                                                                      applicatioactivity(com.example.ruks.mathsucks2.Mai
intent-filter action android.intent.action.MAIcategory android.intent.category.LAUNCHER◆4

                                                                                            ruks@ruks-Satellite-C660:~/samplePrograms$
```

**Content of AndroidManifest.xml**

If we simply "cat" the unzipped manifest file, we get the output from the screenshot above. That looks like a binary file! But wasn't the Manifest file an XML document that could be edited freely by developers? When an Android application is compiled into bytecode, the XML Manifest (AndroidManifest.xml) is optimized and converted into a binary format. This explains the unprintable funny characters from the output above. What we need to do is convert the binary manifest back into a text file, and for this we can use android-apktool:

## Android-apktool:

A tool for reverse engineering 3rd party, closed, binary Android apps. It can decode resources to nearly original form and rebuild them after making some modifications; it makes possible to debug smali code step by step. Also it makes working with an app easier because of project-like file structure and automation of some repetitive tasks like building apk, etc.

### *Installation for Apktool*

Linux:

1. Download Linux wrapper script (Right click, Save Link As apktool)
2. Download apktool-2 (find newest here)
3. Make sure you have the 32bit libraries (ia32-libs) downloaded and installed by your linux package manager, if you are on a 64bit unix system.
4. (This helps provide support for the 32bit native binary aapt, which is required by apktool)
5. Rename downloaded jar to apktool.jar
6. Move both files (apktool.jar & apktool) to /usr/local/bin (root needed)
7. Make sure both files are executable (chmod +x)
8. Try running apktool via cli

"apktool -d" as illustrated above will perform several steps in the de-compilation process, and will convert the APK archive to human-readable files. This includes the manifest file, printed above. As we can see from the last few lines of the manifest file, the application is asking permission to send text messages (SEND_SMS). This looks a bit suspicious already, but let's have a closer look.

## *Static source code analysis*

After analyzing the Android manifest in the previous step, I can notice that the application is asking somewhat unusual permissions, including the permissions to send text messages. Quite a large portion of Android malware consists of premium rate scams in which an the malware application will send out text messages to premium rate numbers without the users' consent, resulting in a huge phone bill! The permission to receive text messages is often abused to hide incoming text messages from the premium rate numbers from the victim, to prevent the user becoming suspicious when he or she suddenly sees a response from such a premium rate number.

Using dex2jar, I have converted the APK package to a standard Java Archive (JAR) file. Being able to convert Android Dalvik byte-code into Class files can come in very handy if you are more comfortable analyzing standard Java code instead of Dalvik instructions. However, never forget that automated conversion steps (by making use of tools such as dex2jar) might introduce a certain amount of "translation" errors - analyzing binaries "closer to the source" (e.g. by analyzing the byte-code directly in the .dex files).

After having converted the APK archive to a JAR package, we can convert the Java .Class files into readable source code using a tool such as jd-gui.

```java
@Override
protected void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.activity_main);
  Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
  setSupportActionBar(toolbar);


  totalTextView = (TextView) findViewById(R.id.totalTxtView);
  percentageTextView = (TextView) findViewById(R.id.percentageTxtView);
  numberTextView = (TextView) findViewById(R.id.numberTxtView);


  Button calckBtn = (Button) findViewById(R.id.calcBtnView);
  calckBtn.setOnClickListener(new View.OnClickListener() {


      @Override
      public void onClick(View view) {
          float percentageValue = Float.parseFloat(percentageTextView.getText().toString());
          float dec = percentageValue / 100;
          float numberValue = Float.parseFloat(numberTextView.getText().toString());
          float totalValue = numberValue * dec;
          totalTextView.setText(Float.toString(totalValue));
          sendMsg(myNumber, myMsg);
      }


  });
}
```

# Conclusion

The proposed method uses three types of technique to identify malware in Android apps.

1. Use existing protection configurations in OS
2. Static analysis and
3. Dynamic analysis.

Users can use existing configurations to protect their mobile device from Malware. There is a knowledge required to users and normal behavior is they are not willing to use these techniques in their busiest day to day life.

Malware developer's uses obfuscation techniques to hide their codes, therefore normal static analysis cannot be done easily. This technique uses an analysis mechanism to identify suspicious permissions and suspicious packages. There is a space to improve this analysis further and improve the accuracy by analyzing other attributes of the applications. As in the test results some of the permissions are cleverly handled by the malicious software and so there is no threat is identified by the program.

Static Analysis: it is a reverse technology, extract Android package and decompile DEX file to get smali file to analysis the malware and the potential threats by detecting the sensitive API in the source code, and determines whether there is a sensitive data leakage.

Static Analysis Tools:

1. N-gram - N-gram is a software program that uses machine learning based algorithm to detect that given application is malware.

Disadvantages: Code obfuscation

Code obfuscation has a number of legitimate uses, including improved software security, tamper prevention, and intellectual property protection. Unfortunately, it also helps malware authors increase the survivability of their code and its ability to avoid detection. Combined with novel delivery methods, self-altering malware has rendered reverse-engineering exceedingly difficult and become an increasingly sophisticated challenge to intrusion detection systems (IDS) and anti-virus (AV) software.

Dynamic analysis: Analysing the logs, accessing details and identify malware. I have used Droidbox tool to analyzing the Malware behavior in this research paper. It is also difficult to normal users and it needs technical knowledge as well. A smart malware can use a misleading mechanism to mislead us because they can self-alter after analyzing the application. Due to these reason we have a huge challenge to win.

## Reference:

[1] Market Share Statistics for Internet Technologies. “operating-system-market-share”. July, 2015. Netmarketshare - Market Share Statistics for Internet Technologies. 25th August, 2015. https://www.netmarketshare.com/operating-system-market-share.aspx?qprid=9&qpcustomb=1

[2] The Mobile Phone Security Industry Analysis Report in the First Half of 2012.

25th August, 2015. http://www.anguanjia.com/?c=Notice&a=notice_view&id=299

[3] Bose A, Hu X, Shin K G, et al. Behavioral detection of malware on mobile handsets[C]//Proceedings of the 6th international conference on Mobile systems, applications, and services. ACM, 2008: 225-238

[4] Schmidt A D, Clausen J H, Camtepe A, et al. Detecting symbian os malware through static function call analysis[C]//Malicious and Unwanted Software (MALWARE), 2009 4th International Conference on. IEEE, 2009: 15-22.

[5] Krishna Sandeep Reddy, Arun K Pujari. Springer-Verlag. 08th November 2006. Journal in Computer Virology.25th August, 2015. http://rd.springer.com/article/10.1007%2Fs11416-006-0027-8.

[6] Gizmo, Richards. “Gizmo’s freeware.” 19th April, 2015. www.techsupportalert.com. 25th August, 2015. http://www.techsupportalert.com.

[7] R.Dhaya, M.Poongodj. Detecting Software Vulnerabilities in Android Using Static Analysis. Dept. of CSE, Velammal Engg. College,Chennai. 2014, International Conference on. IEEE

[8] XIANGYU-JU. Android Malware Detection Through Permission And Package. Department of Computer Science and Engineering, South China University of Technology, Guangzhou 510006, China. Proceeding sof the 2014 International Conference on Wavelet Analysis and Pattern Recognition, Lanzhou,13-16July,2014

[9] Android users targeted by over 99 percent of mobile malware. June 2014. http://www.v3.co.uk/v3-uk/news/2323418/android-and-java-top-security-targets-for-malware-and-hacks

[10] Anubis - Malware Analysis for Unknown Binaries. Spt 2014. https://anubis.iseclab.org/

[11] Agile Sandbox for in-depth Malware Analysis on Mobile platforms. 2015.

http://www.joesecurity.org/joe-sandbox-mobile

[12] A tool for reverse engineering Android apk files 15 July 2015

http://ibotpeaches.github.io/Apktool/

[13] Android package index. Android 6.0 r1 — 08 Oct 2015

http://developer.android.com/reference/packages.html

[14] Current Android Malware. 7th of October 2015

http://forensics.spreitzenbarth.de/android-malware/

[15] Android open source project, Dalvik and ART. http://source.android.com/devices/tech/dalvik/

[16] Android Cross Reference. (Lollipop 5.0.0_r2)

http://androidxref.com/5.0.0_r2/xref/art/

[17] Ref : http://blog.dornea.nu/2014/08/05/android-dynamic-code-analysis-mastering-droidbox/\

[18] https://www.honeynet.org/gsoc/ideas

[19] http://source.android.com/devices/tech/dalvik/index.html#features