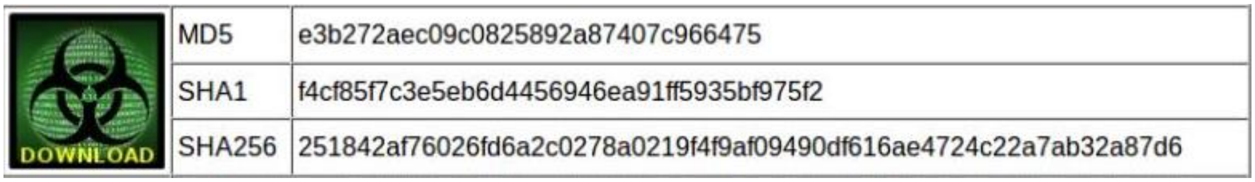
Reverse Engineering of Android Malware

Overview:

This document aims at explaining the steps to perform *static* analysis of an Android malware using the **Androguard** tool and provide the findings. This document has two parts. The first part explains the limited analysis that could be performed using **Androguard** on Santoku Platform. However, Part 2 provides deeper analysis performed using **Androguard** on Kali Linux platform.

Android Malware File Details:

The following screenshot captures the MD5, SHA1 and SHA256 of the android malware file downloaded from VirusShare website. We saved this malware under the name *Andromal1.apk* on our system.



MD5	e3b272aec09c0825892a87407c966475
SHA1	f4cf85f7c3e5eb6d4456946ea91ff5935bf975f2
SHA256	251842af76026fd6a2c0278a0219f4f9af09490df616ae4724c22a7ab32a87d6

<u>Part 1</u>

Tool used: Androguard.

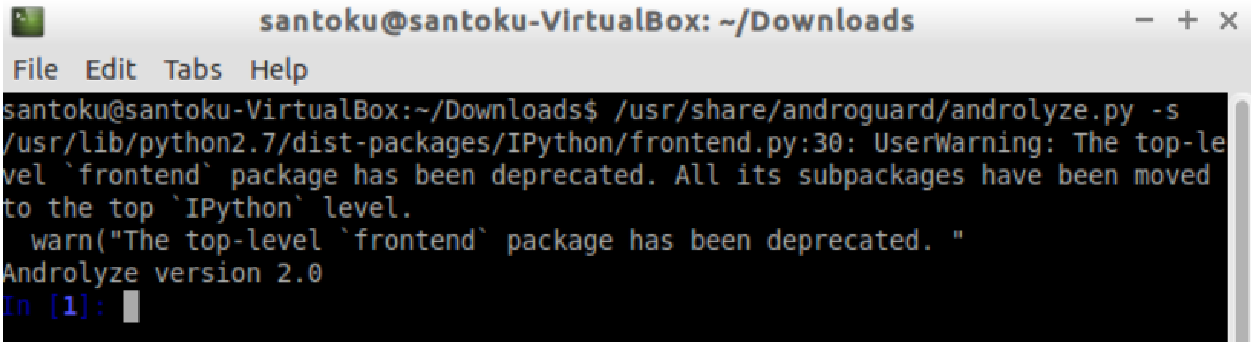
OS Platform: Santoku

Analysis

To start the analysis of the .apk file using androguard, we need to decompile (extract the source code) the apk first. Androguard works with three different decompilers for the apk: (1) dex2jar, (2) dad (this is the default decompiler), (3) ded.

- For interactive analysis of the .apk, run the Androlyze tool of Androguard as follows, this will start an interactive Ipython shell as shown in the screen shot:

```
$ androlyze.py – s
```

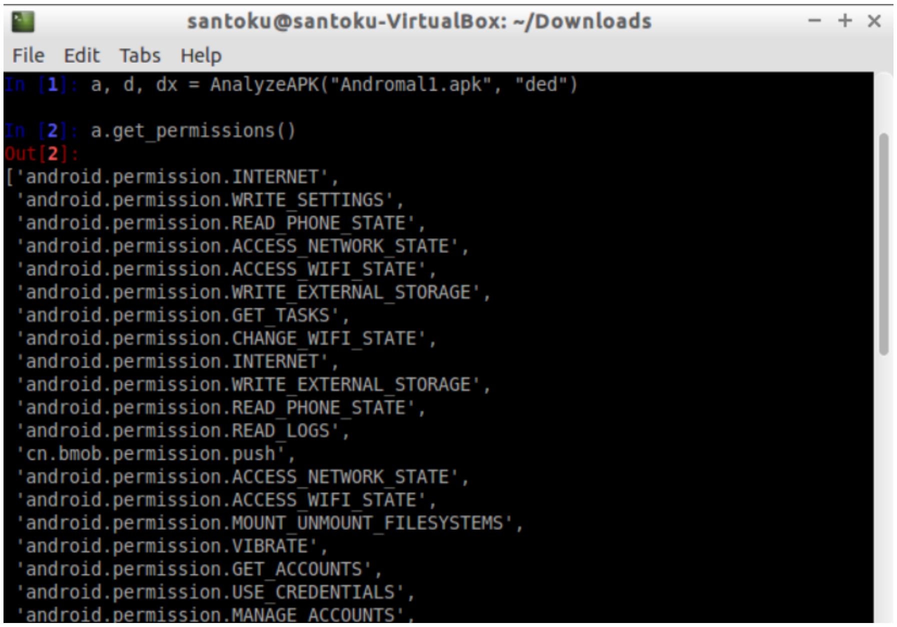


```
santoku@santoku-VirtualBox:~/Downloads$ /usr/share/androguard/androlyze.py -s
/usr/lib/python2.7/dist-packages/IPython/frontend.py:30: UserWarning: The top-le
vel `frontend` package has been deprecated. All its subpackages have been moved
to the top `IPython` level.
  warn("The top-level `frontend` package has been deprecated. "
Androlyze version 2.0
In [1]: 
```

Reverse Engineering of Android Malware

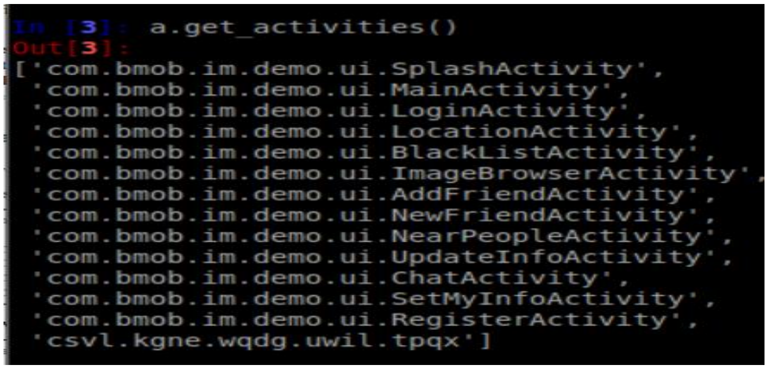
Extracting information about the APK

- Next in the interpreter window load the apk file for analysis by running the following command: `a, d, dx = AnalyzeAPK("Andromal1.apk", decompiler = “dad”)`
  The wrapper functions *AnalyzeAPK* , returns three objects. `a` is an *APK object*, `d` is an array of *DalvikVMFormat* object and `dx` is an *Analysis object*.
- Once we have the APK object, we can extract more detail information of the apk file.
  - Getting the permissions used by the apk:
    Command: `a.get_permissions()`
    Demo output:



```
In [1]: a, d, dx = AnalyzeAPK("Andromal1.apk", "ded")

In [2]: a.get_permissions()
Out[2]:
['android.permission.INTERNET',
 'android.permission.WRITE_SETTINGS',
 'android.permission.READ_PHONE_STATE',
 'android.permission.ACCESS_NETWORK_STATE',
 'android.permission.ACCESS_WIFI_STATE',
 'android.permission.WRITE_EXTERNAL_STORAGE',
 'android.permission.GET_TASKS',
 'android.permission.CHANGE_WIFI_STATE',
 'android.permission.INTERNET',
 'android.permission.WRITE_EXTERNAL_STORAGE',
 'android.permission.READ_PHONE_STATE',
 'android.permission.READ_LOGS',
 'cn.bmob.permission.push',
 'android.permission.ACCESS_NETWORK_STATE',
 'android.permission.ACCESS_WIFI_STATE',
 'android.permission.MOUNT_UNMOUNT_FILESYSTEMS',
 'android.permission.VIBRATE',
 'android.permission.GET_ACCOUNTS',
 'android.permission.USE_CREDENTIALS',
 'android.permission.MANAGE_ACCOUNTS',
```

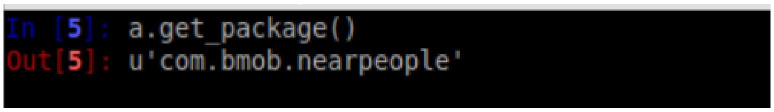
  - List of activities of the apk:
    Command: `a.get_activities()`
    Demo output:



```
In [3]: a.get_activities()
Out[3]:
['com.bmob.im.demo.ui.SplashActivity',
 'com.bmob.im.demo.ui.MainActivity',
 'com.bmob.im.demo.ui.LoginActivity',
 'com.bmob.im.demo.ui.LocationActivity',
 'com.bmob.im.demo.ui.BlackListActivity',
 'com.bmob.im.demo.ui.ImageBrowserActivity',
 'com.bmob.im.demo.ui.AddFriendActivity',
 'com.bmob.im.demo.ui.NewFriendActivity',
 'com.bmob.im.demo.ui.NearPeopleActivity',
 'com.bmob.im.demo.ui.UpdateInfoActivity',
 'com.bmob.im.demo.ui.ChatActivity',
 'com.bmob.im.demo.ui.SetMyInfoActivity',
 'com.bmob.im.demo.ui.RegisterActivity',
 'csvl.kgne.wqdg.uwil.tpqx']
```

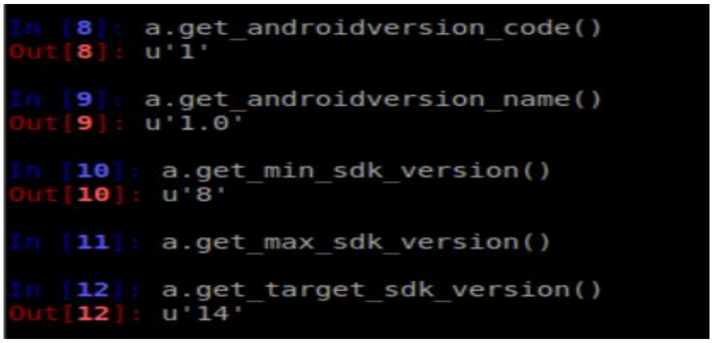
  - Package of the apk:
    Command: `a.get_package()`
    Demo output:

Reverse Engineering of Android Malware



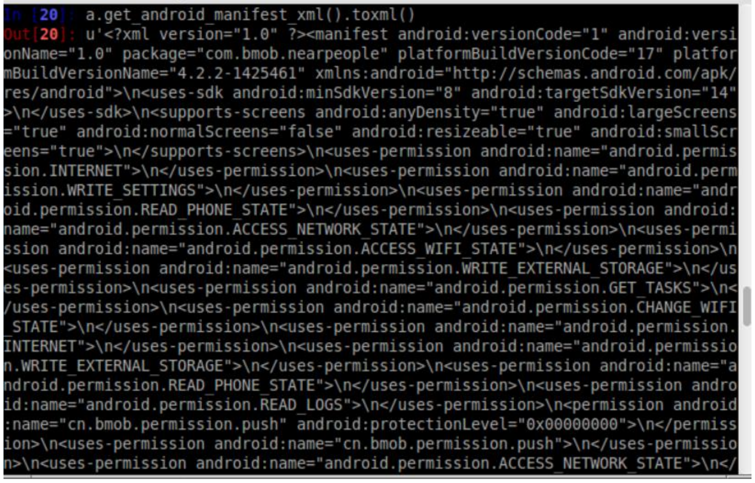
```
In [5]: a.get_package()
Out[5]: u'com.bmob.nearpeople'
```

- Get the numeric version and the version string, and the minimal, maximal, target SDK version:



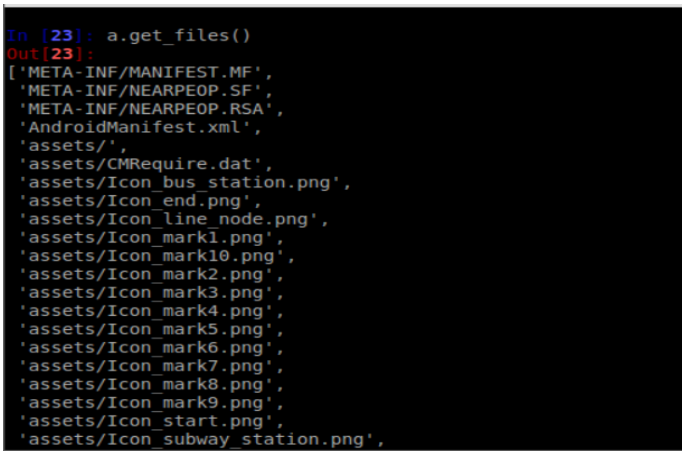
```
In [8]: a.get_androidversion_code()
Out[8]: u'1'

In [9]: a.get_androidversion_name()
Out[9]: u'1.0'

In [10]: a.get_min_sdk_version()
Out[10]: u'8'

In [11]: a.get_max_sdk_version()

In [12]: a.get_target_sdk_version()
Out[12]: u'14'
```

- Get the AndroidManifest.xml:



```
In [20]: a.get_android_manifest_xml().toxml()
Out[20]: u'<?xml version="1.0" ?><manifest android:versionCode="1" android:versi
onName="1.0" package="com.bmob.nearpeople" platformBuildVersionCode="17" platfor
mBuildVersionName="4.2.2-1425461" xmlns:android="http://schemas.android.com/apk/
res/android">\n<uses-sdk android:minSdkVersion="8" android:targetSdkVersion="14"
>\n</uses-sdk>\n<supports-screens android:anyDensity="true" android:largeScreens
="true" android:normalScreens="false" android:resizeable="true" android:smallScr
eens="true">\n</supports-screens>\n<uses-permission android:name="android.permis
sion.INTERNET">\n</uses-permission>\n<uses-permission android:name="android.perm
ission.WRITE_SETTINGS">\n</uses-permission>\n<uses-permission android:name="andr
oid.permission.READ_PHONE_STATE">\n</uses-permission>\n<uses-permission android:
name="android.permission.ACCESS_NETWORK_STATE">\n</uses-permission>\n<uses-permi
ssion android:name="android.permission.ACCESS_WIFI_STATE">\n</uses-permission>\n
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE">\n</us
es-permission>\n<uses-permission android:name="android.permission.GET_TASKS">\n<
/uses-permission>\n<uses-permission android:name="android.permission.CHANGE_WIFI
_STATE">\n</uses-permission>\n<uses-permission android:name="android.permission.
INTERNET">\n</uses-permission>\n<uses-permission android:name="android.permissio
n.WRITE_EXTERNAL_STORAGE">\n</uses-permission>\n<uses-permission android:name="a
ndroid.permission.READ_PHONE_STATE">\n</uses-permission>\n<uses-permission andro
id:name="android.permission.READ_LOGS">\n</uses-permission>\n<permission android
:name="cn.bmob.permission.push" android:protectionLevel="0x00000000">\n</permiss
ion>\n<uses-permission android:name="cn.bmob.permission.push">\n</uses-permissio
n>\n<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE">\n</
```

- Extract the files contained in the apk:



```
In [23]: a.get_files()
Out[23]:
['META-INF/MANIFEST.MF',
 'META-INF/NEARPEOP.SF',
 'META-INF/NEARPEOP.RSA',
 'AndroidManifest.xml',
 'assets/',
 'assets/CMRequire.dat',
 'assets/Icon_bus_station.png',
 'assets/Icon_end.png',
 'assets/Icon_line_node.png',
 'assets/Icon_mark1.png',
 'assets/Icon_mark10.png',
 'assets/Icon_mark2.png',
 'assets/Icon_mark3.png',
 'assets/Icon_mark4.png',
 'assets/Icon_mark5.png',
 'assets/Icon_mark6.png',
 'assets/Icon_mark7.png',
 'assets/Icon_mark8.png',
 'assets/Icon_mark9.png',
 'assets/Icon_start.png',
 'assets/Icon_subway_station.png',
```

Beyond these, no other Androguard functionalities worked on Santoku.

Androguard is a python based tool to perform extensive analysis on android files. It is supported by python3. However, the default python installation for Santoku is 2.7. Hence, perform the following steps to change the python3 installation as default:

1. Open the .bashrc file on an editor `vim ~/.bashrc`
2. Type `alias python=python3` and save the file.
3. At the command line run `source ~/.bashrc`

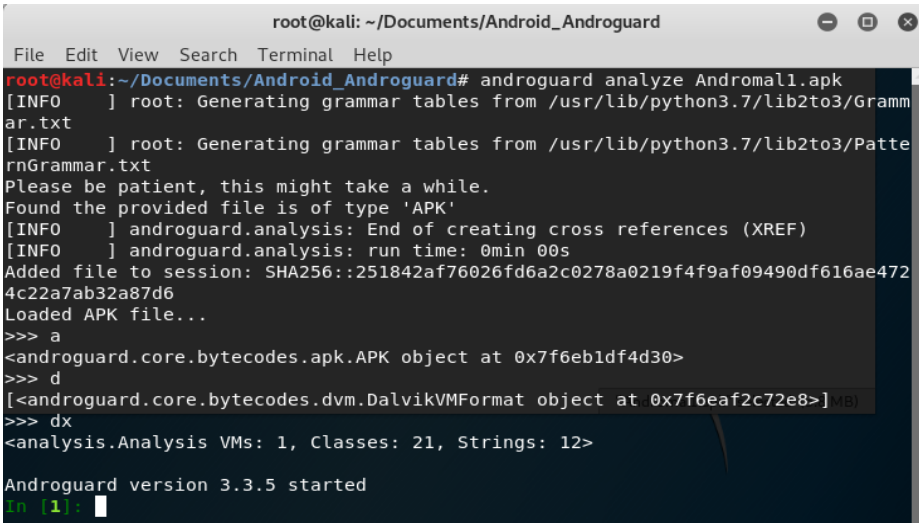
Even after upgrading python and the Androguard, the tool did not produce any further analysis. Hence, we switched to Kali Linux platform and the static analysis performed on this platform are presented in the following section.

Part 2

Start by extracting the source code of the .apk file using the following command, it will start an ipython shell.

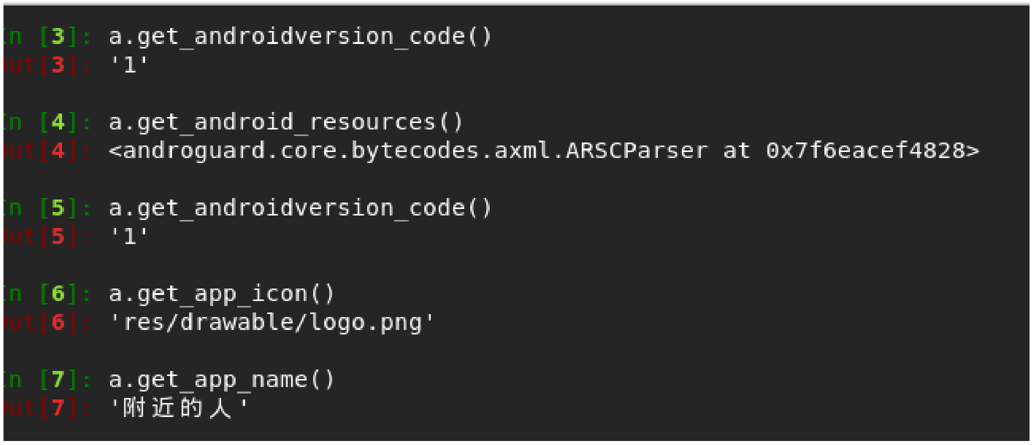
1. Executing the following command:

```
androguard analyze Andromal1.apk
```


root@kali: ~/Documents/Android_Androguard
File Edit View Search Terminal Help

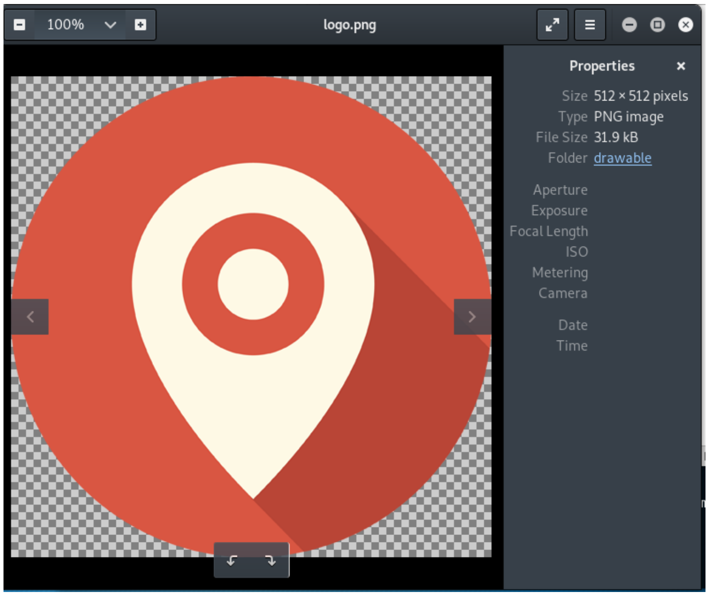
```
root@kali:~/Documents/Android_Androguard# androguard analyze Andromal1.apk
[INFO    ] root: Generating grammar tables from /usr/lib/python3.7/lib2to3/Gramm
ar.txt
[INFO    ] root: Generating grammar tables from /usr/lib/python3.7/lib2to3/Patte
rnGrammar.txt
Please be patient, this might take a while.
Found the provided file is of type 'APK'
[INFO    ] androguard.analysis: End of creating cross references (XREF)
[INFO    ] androguard.analysis: run time: 0min 00s
Added file to session: SHA256::251842af76026fd6a2c0278a0219f4f9af09490df616ae472
4c22a7ab32a87d6
Loaded APK file...
>>> a
<androguard.core.bytecodes.apk.APK object at 0x7f6eb1df4d30>
>>> d
[<androguard.core.bytecodes.dvm.DalvikVMFormat object at 0x7f6eaf2c72e8>]
>>> dx
<analysis.Analysis VMs: 1, Classes: 21, Strings: 12>

Androguard version 3.3.5 started
In [1]: 
```

2. From the interactive iphthon shell, we can extract the android version code, resources, path to app logo and the app name as follow:



```
n [3]: a.get_androidversion_code()
ut[3]: '1'

n [4]: a.get_android_resources()
ut[4]: <androguard.core.bytecodes.axml.ARSCParser at 0x7f6eacef4828>

n [5]: a.get_androidversion_code()
ut[5]: '1'

n [6]: a.get_app_icon()
ut[6]: 'res/drawable/logo.png'

n [7]: a.get_app_name()
ut[7]: '附近的人'
```

The app logo was:

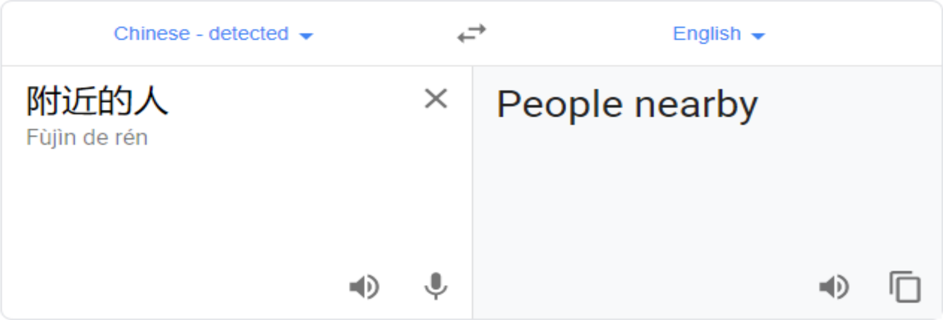


Size 512 × 512 pixels
Type PNG image
File Size 31.9 kB
Folder drawable

Aperture
Exposure
Focal Length
ISO
Metering
Camera

Date
Time

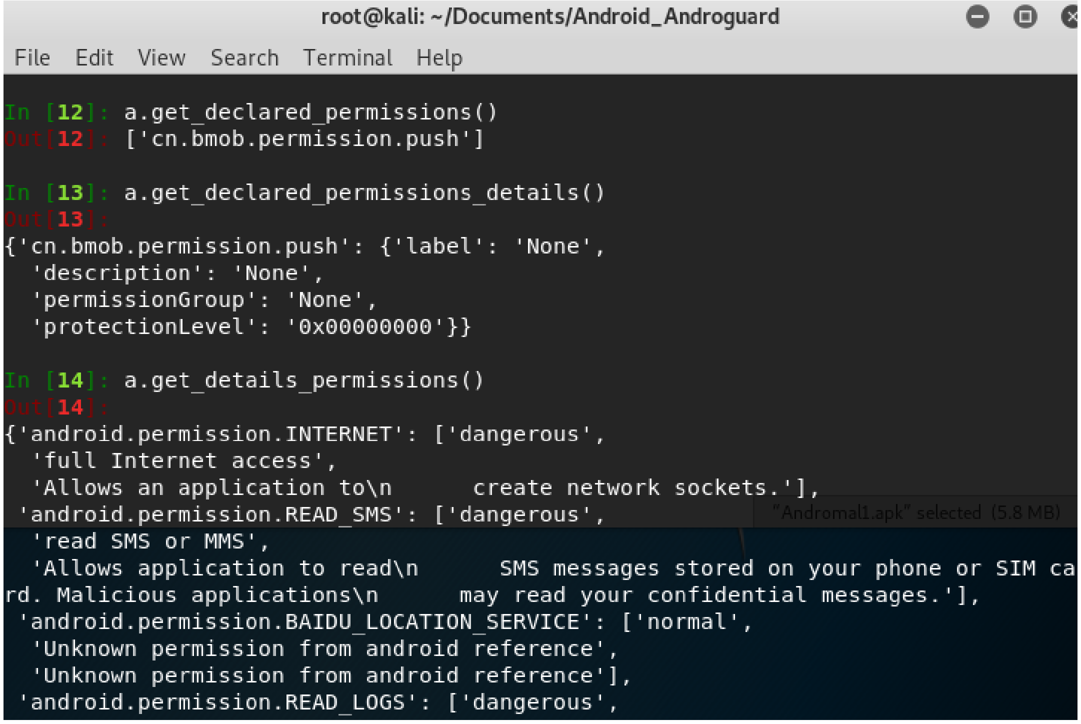
Using google translate we could understand the app name is in Chinese and the app name in English:



附近的人
Fùjìn de rén

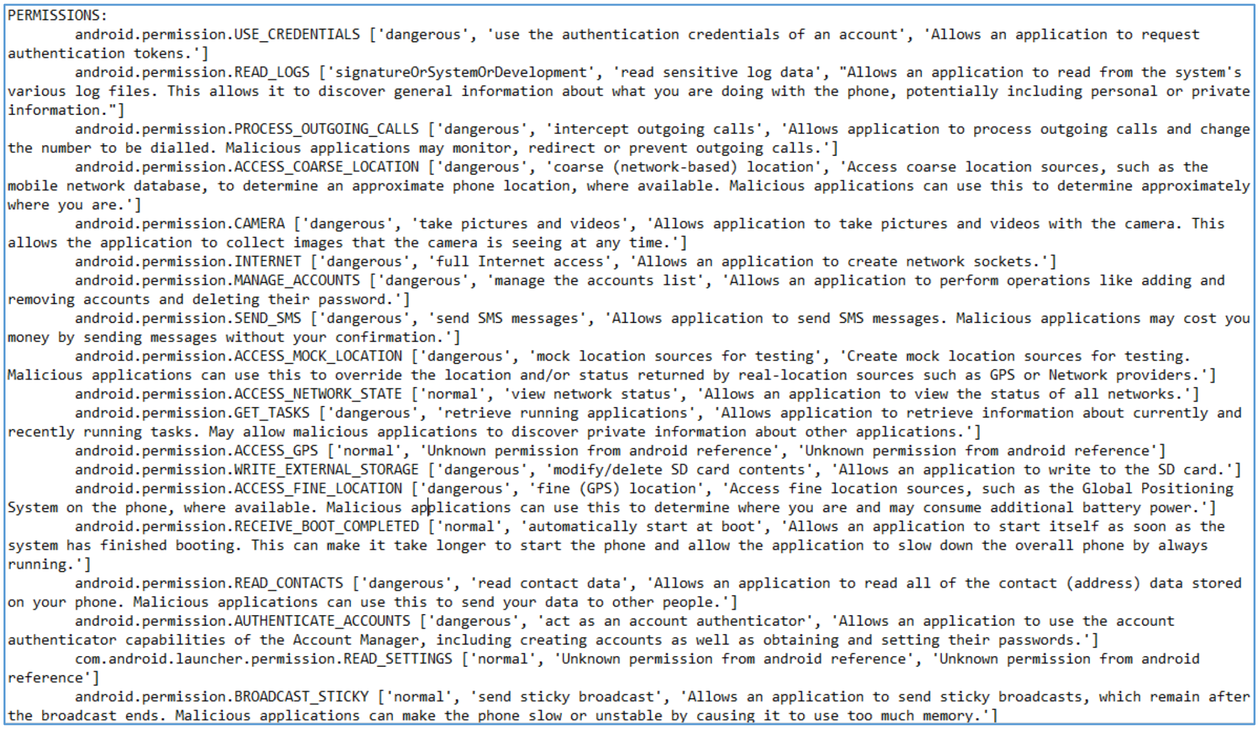
People nearby

3. Then we extracted the detailed permissions as:


root@kali: ~/Documents/Android_Androguard
File Edit View Search Terminal Help
"Andromal1.apk" selected (5.8 MB)

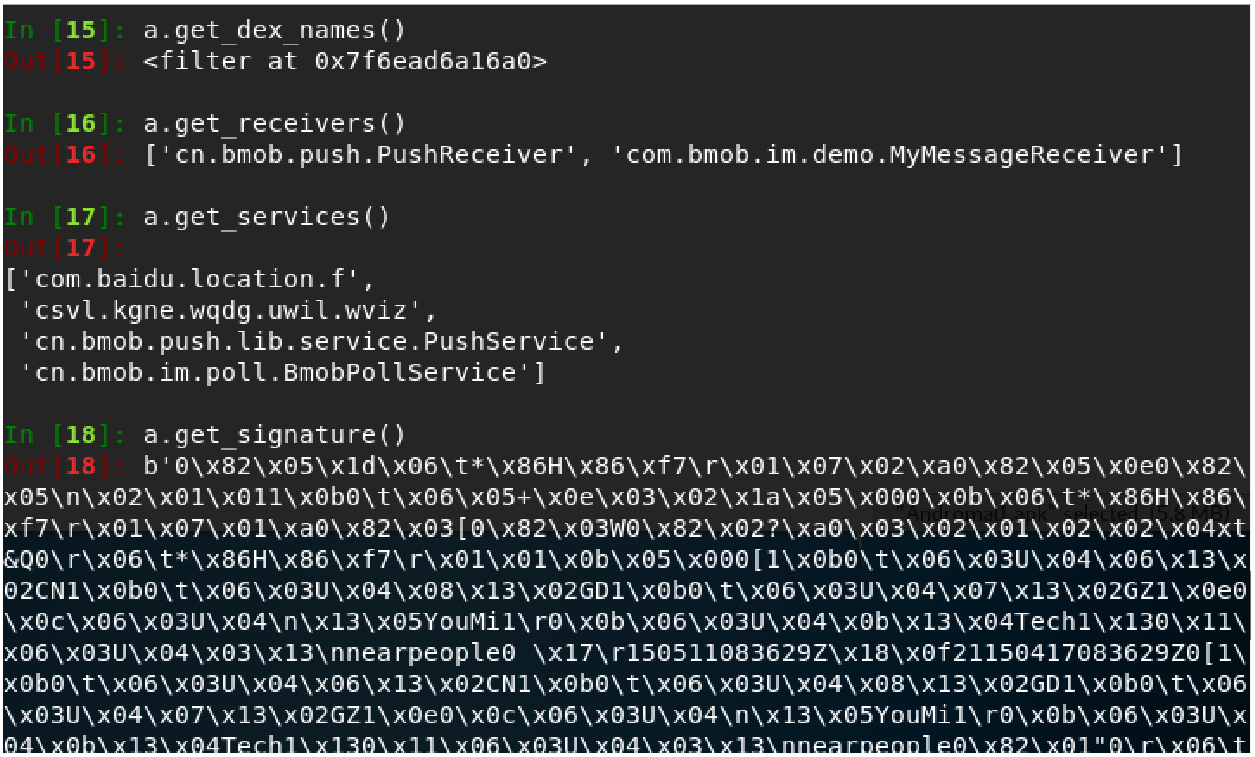
```
In [12]: a.get_declared_permissions()
Out[12]: ['cn.bmob.permission.push']

In [13]: a.get_declared_permissions_details()
Out[13]:
{'cn.bmob.permission.push': {'label': 'None',
  'description': 'None',
  'permissionGroup': 'None',
  'protectionLevel': '0x00000000'}}

In [14]: a.get_details_permissions()
Out[14]:
{'android.permission.INTERNET': ['dangerous',
  'full Internet access',
  'Allows an application to\n      create network sockets.'],
 'android.permission.READ_SMS': ['dangerous',
  'read SMS or MMS',
  'Allows application to read\n      SMS messages stored on your phone or SIM ca
rd. Malicious applications\n      may read your confidential messages.'],
 'android.permission.BAIDU_LOCATION_SERVICE': ['normal',
  'Unknown permission from android reference',
  'Unknown permission from android reference'],
 'android.permission.READ_LOGS': ['dangerous',
```

Androguard tags the permissions as normal or dangerous, and why it the app needs that permission as shown in the following text file:



```
PERMISSIONS:
	android.permission.USE_CREDENTIALS ['dangerous', 'use the authentication credentials of an account', 'Allows an application to request authentication tokens.']
	android.permission.READ_LOGS ['signatureOrSystemOrDevelopment', 'read sensitive log data', "Allows an application to read from the system's various log files. This allows it to discover general information about what you are doing with the phone, potentially including personal or private information."]
	android.permission.PROCESS_OUTGOING_CALLS ['dangerous', 'intercept outgoing calls', 'Allows application to process outgoing calls and change the number to be dialled. Malicious applications may monitor, redirect or prevent outgoing calls.']
	android.permission.ACCESS_COARSE_LOCATION ['dangerous', 'coarse (network-based) location', 'Access coarse location sources, such as the mobile network database, to determine an approximate phone location, where available. Malicious applications can use this to determine approximately where you are.']
	android.permission.CAMERA ['dangerous', 'take pictures and videos', 'Allows application to take pictures and videos with the camera. This allows the application to collect images that the camera is seeing at any time.']
	android.permission.INTERNET ['dangerous', 'full Internet access', 'Allows an application to create network sockets.']
	android.permission.MANAGE_ACCOUNTS ['dangerous', 'manage the accounts list', 'Allows an application to perform operations like adding and removing accounts and deleting their password.']
	android.permission.SEND_SMS ['dangerous', 'send SMS messages', 'Allows application to send SMS messages. Malicious applications may cost you money by sending messages without your confirmation.']
	android.permission.ACCESS_MOCK_LOCATION ['dangerous', 'mock location sources for testing', 'Create mock location sources for testing. Malicious applications can use this to override the location and/or status returned by real-location sources such as GPS or Network providers.']
	android.permission.ACCESS_NETWORK_STATE ['normal', 'view network status', 'Allows an application to view the status of all networks.']
	android.permission.GET_TASKS ['dangerous', 'retrieve running applications', 'Allows application to retrieve information about currently and recently running tasks. May allow malicious applications to discover private information about other applications.']
	android.permission.ACCESS_GPS ['normal', 'Unknown permission from android reference', 'Unknown permission from android reference']
	android.permission.WRITE_EXTERNAL_STORAGE ['dangerous', 'modify/delete SD card contents', 'Allows an application to write to the SD card.']
	android.permission.ACCESS_FINE_LOCATION ['dangerous', 'fine (GPS) location', 'Access fine location sources, such as the Global Positioning System on the phone, where available. Malicious applications can use this to determine where you are and may consume additional battery power.']
	android.permission.RECEIVE_BOOT_COMPLETED ['normal', 'automatically start at boot', 'Allows an application to start itself as soon as the system has finished booting. This can make it take longer to start the phone and allow the application to slow down the overall phone by always running.']
	android.permission.READ_CONTACTS ['dangerous', 'read contact data', 'Allows an application to read all of the contact (address) data stored on your phone. Malicious applications can use this to send your data to other people.']
	android.permission.AUTHENTICATE_ACCOUNTS ['dangerous', 'act as an account authenticator', 'Allows an application to use the account authenticator capabilities of the Account Manager, including creating accounts as well as obtaining and setting their passwords.']
	com.android.launcher.permission.READ_SETTINGS ['normal', 'Unknown permission from android reference', 'Unknown permission from android reference']
	android.permission.BROADCAST_STICKY ['normal', 'send sticky broadcast', 'Allows an application to send sticky broadcasts, which remain after the broadcast ends. Malicious applications can make the phone slow or unstable by causing it to use too much memory.']
```

4. Extract the dex file contained in the root directory of the APK and the receivers and services list as follow:

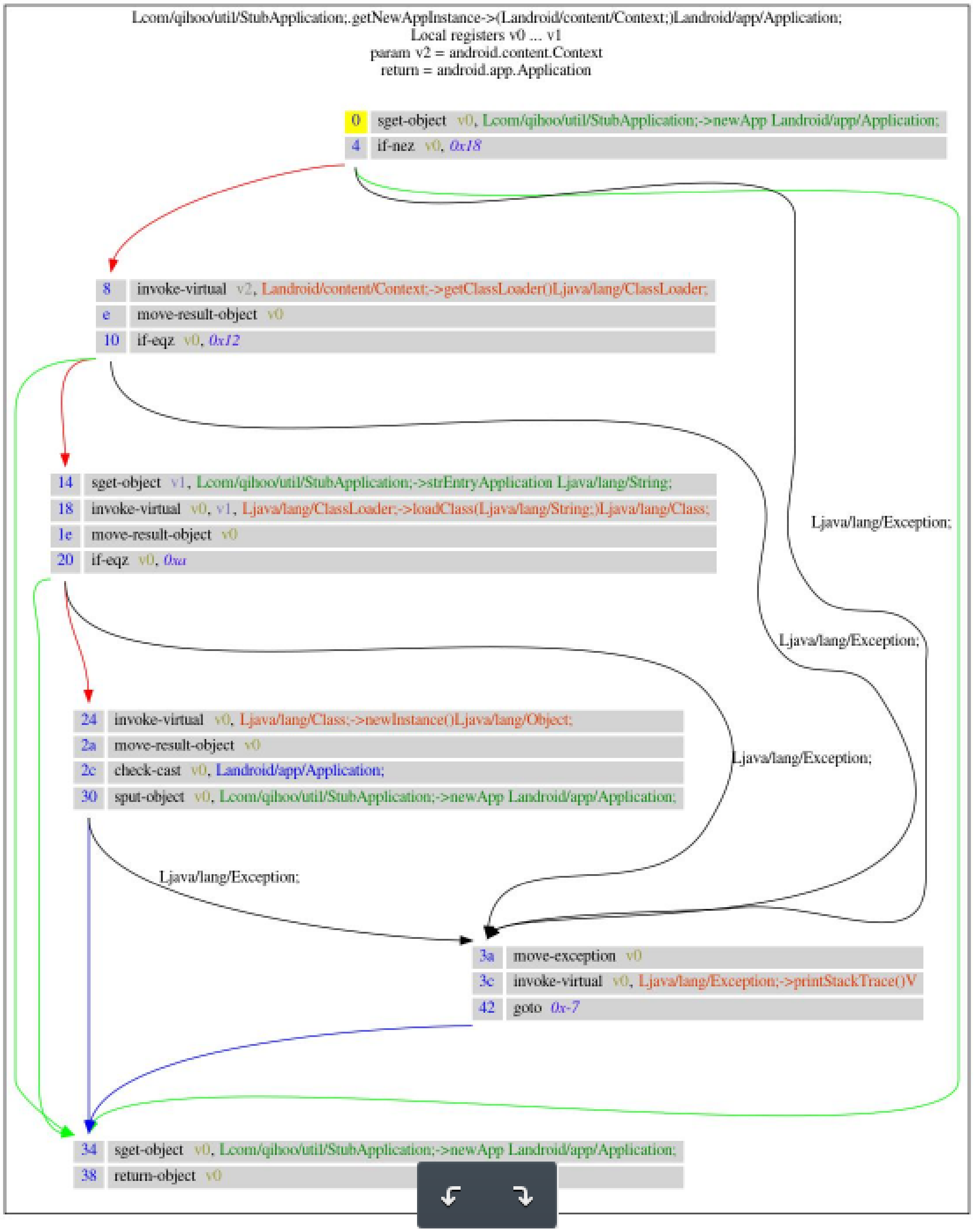


```
In [15]: a.get_dex_names()
Out[15]: <filter at 0x7f6ead6a16a0>

In [16]: a.get_receivers()
Out[16]: ['cn.bmob.push.PushReceiver', 'com.bmob.im.demo.MyMessageReceiver']

In [17]: a.get_services()
Out[17]:
['com.baidu.location.f',
 'csvl.kgne.wqdg.uwil.wviz',
 'cn.bmob.push.lib.service.PushService',
 'cn.bmob.im.poll.BmobPollService']

In [18]: a.get_signature()
Out[18]: b'0\x82\x05\x1d\x06\t*\x86H\x86\xf7\r\x01\x07\x02\xa0\x82\x05\x0e0\x82\
x05\n\x02\x01\x011\x0b0\t\x06\x05+\x0e\x03\x02\x1a\x05\x000\x0b\x06\t*\x86H\x86\
xf7\r\x01\x07\x01\xa0\x82\x03[0\x82\x03W0\x82\x02?\xa0\x03\x02\x01\x02\x02\x04xt
&Q0\r\x06\t*\x86H\x86\xf7\r\x01\x01\x0b\x05\x000[1\x0b0\t\x06\x03U\x04\x06\x13\x
02CN1\x0b0\t\x06\x03U\x04\x08\x13\x02GD1\x0b0\t\x06\x03U\x04\x07\x13\x02GZ1\x0e0
\x0c\x06\x03U\x04\n\x13\x05YouMi1\r0\x0b\x06\x03U\x04\x0b\x13\x04Tech1\x130\x11\
x06\x03U\x04\x03\x13\nnearpeople0 \x17\r150511083629Z\x18\x0f21150417083629Z0[1\
x0b0\t\x06\x03U\x04\x06\x13\x02CN1\x0b0\t\x06\x03U\x04\x08\x13\x02GD1\x0b0\t\x06
\x03U\x04\x07\x13\x02GZ1\x0e0\x0c\x06\x03U\x04\n\x13\x05YouMi1\r0\x0b\x06\x03U\x
04\x0b\x13\x04Tech1\x130\x11\x06\x03U\x04\x03\x13\nnearpeople0\x82\x01"0\r\x06\t
```

Reverse Engineering of Android Malware

5. Androguard provides commands, that generate the control flow graphs of the methods and it is a very useful way to visualize the app.

```
androguard decompile -d output_folder -f jpg Andromal1.jpg
```

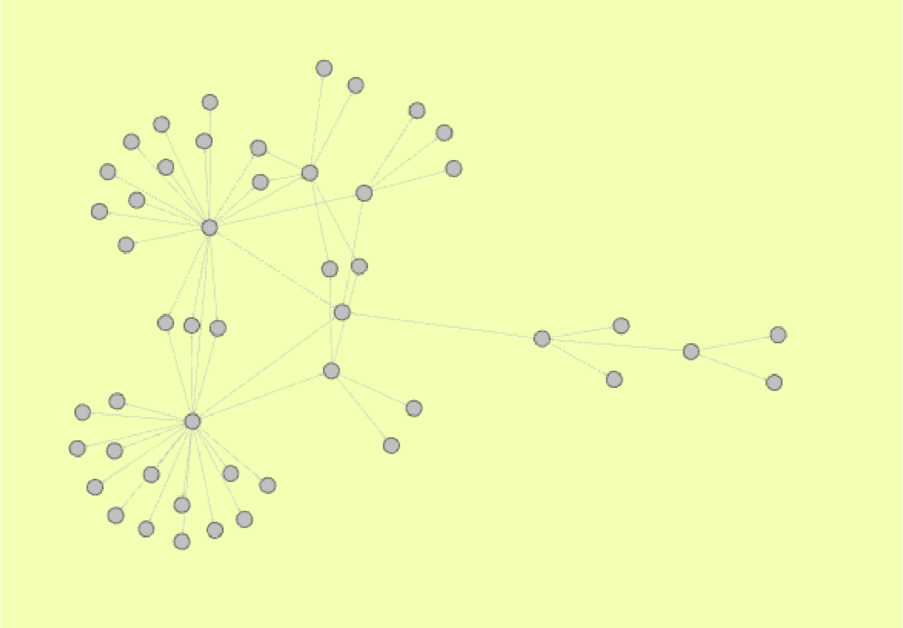


```
Lcom/qihoo/util/StubApplication;.getNewAppInstance->(Landroid/content/Context;)Landroid/app/Application;
Local registers v0 ... v1
param v2 = android.content.Context
return = android.app.Application

0   sget-object  v0, Lcom/qihoo/util/StubApplication;->newApp Landroid/app/Application;
4   if-nez  v0, 0x18

8   invoke-virtual  v2, Landroid/content/Context;->getClassLoader()Ljava/lang/ClassLoader;
e   move-result-object  v0
10  if-eqz  v0, 0x12

14  sget-object  v1, Lcom/qihoo/util/StubApplication;->strEntryApplication Ljava/lang/String;
18  invoke-virtual  v0, v1, Ljava/lang/ClassLoader;->loadClass(Ljava/lang/String;)Ljava/lang/Class;
1e  move-result-object  v0
20  if-eqz  v0, 0xa

24  invoke-virtual  v0, Ljava/lang/Class;->newInstance()Ljava/lang/Object;
2a  move-result-object  v0
2c  check-cast  v0, Landroid/app/Application;
30  sput-object  v0, Lcom/qihoo/util/StubApplication;->newApp Landroid/app/Application;

3a  move-exception  v0
3c  invoke-virtual  v0, Ljava/lang/Exception;->printStackTrace()V
42  goto  0x-7

34  sget-object  v0, Lcom/qihoo/util/StubApplication;->newApp Landroid/app/Application;
38  return-object  v0
```

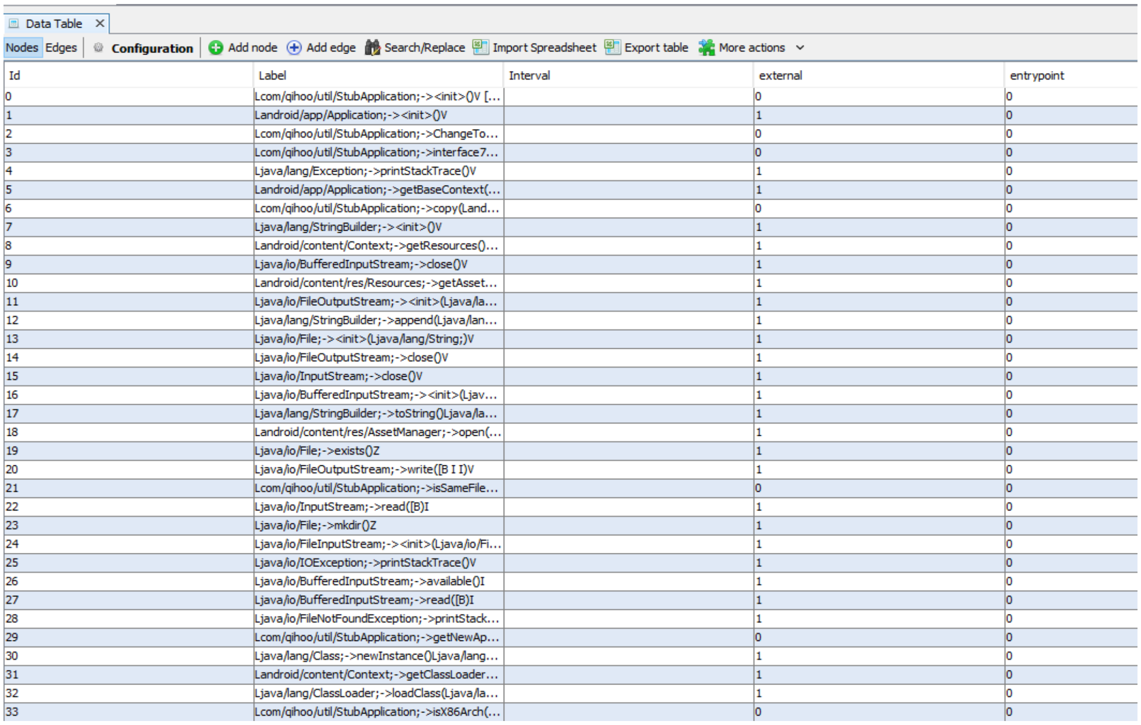
6. Call graph: Androguard provides functionality to present the methods as the node of a graph and the calls to the methods as the edge. The call graph can be generated using the following command:

```
androguard cg Andromal1.apk
```

This command will generate a .gml file which can further be visualized using a network visualization tool (e.g: Gephi):



The following table shows the snapshot of the methods label and their node id details:



Id	Label	Interval	external	entrypoint
0	Lcom/qihoo/util/StubApplication;-><init>()V [...		0	0
1	Landroid/app/Application;-><init>()V		1	0
2	Lcom/qihoo/util/StubApplication;->ChangeTo...		0	0
3	Lcom/qihoo/util/StubApplication;->interface7...		0	0
4	Ljava/lang/Exception;->printStackTrace()V		1	0
5	Landroid/app/Application;->getBaseContext(...		1	0
6	Lcom/qihoo/util/StubApplication;->copy(Land...		0	0
7	Ljava/lang/StringBuilder;-><init>()V		1	0
8	Landroid/content/Context;->getResources()...		1	0
9	Ljava/io/BufferedInputStream;->close()V		1	0
10	Landroid/content/res/Resources;->getAsset...		1	0
11	Ljava/io/FileOutputStream;-><init>(Ljava/la...		1	0
12	Ljava/lang/StringBuilder;->append(Ljava/lan...		1	0
13	Ljava/io/File;-><init>(Ljava/lang/String;)V		1	0
14	Ljava/io/FileOutputStream;->close()V		1	0
15	Ljava/io/InputStream;->close()V		1	0
16	Ljava/io/BufferedInputStream;-><init>(Ljav...		1	0
17	Ljava/lang/StringBuilder;->toString()Ljava/la...		1	0
18	Landroid/content/res/AssetManager;->open(...		1	0
19	Ljava/io/File;->exists()Z		1	0
20	Ljava/io/FileOutputStream;->write([B I I)V		1	0
21	Lcom/qihoo/util/StubApplication;->isSameFile...		0	0
22	Ljava/io/InputStream;->read([B)I		1	0
23	Ljava/io/File;->mkdir()Z		1	0
24	Ljava/io/FileInputStream;-><init>(Ljava/io/Fi...		1	0
25	Ljava/io/IOException;->printStackTrace()V		1	0
26	Ljava/io/BufferedInputStream;->available()I		1	0
27	Ljava/io/BufferedInputStream;->read([B)I		1	0
28	Ljava/io/FileNotFoundException;->printStack...		1	0
29	Lcom/qihoo/util/StubApplication;->getNewAp...		0	0
30	Ljava/lang/Class;->newInstance()Ljava/lang...		1	0
31	Landroid/content/Context;->getClassLoader...		1	0
32	Ljava/lang/ClassLoader;->loadClass(Ljava/la...		1	0
33	Lcom/qihoo/util/StubApplication;->isX86Arch(...		0	0

Reverse Engineering of Android Malware

References:

[1] Androguard Documentation. Release 3.4.0.
https://buildmedia.readthedocs.org/media/pdf/androguard/latest/androguard.pdf

[2] androguard - RE.wiki

https://code.google.com/archive/p/androguard/wikis/RE.wiki

[3] androguard - The swiss army knife

https://androguard.readthedocs.io/en/latest/tools/androguard.html