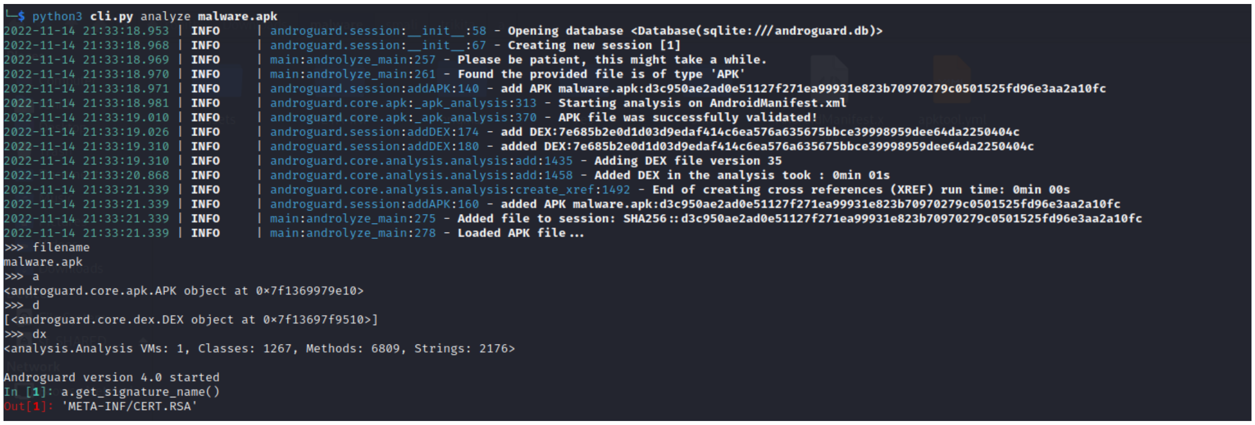
Malicious APK File Analysis
No. 16

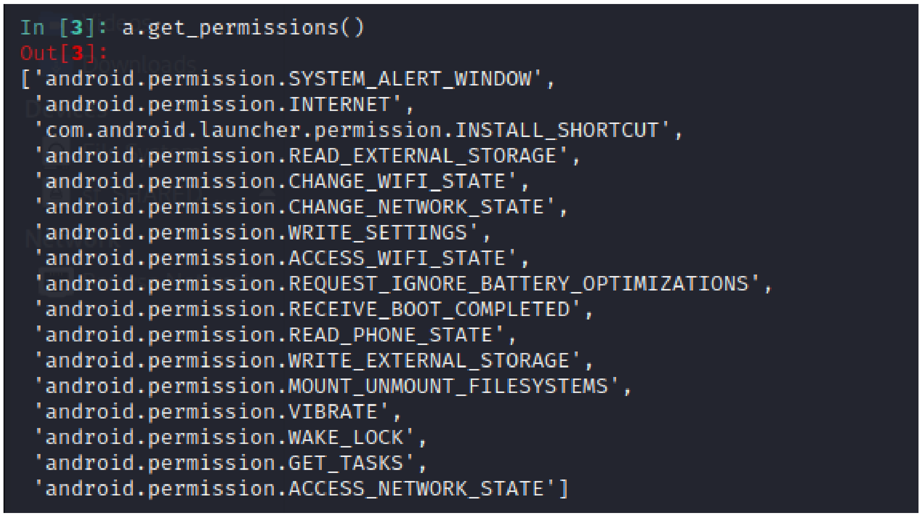
1. Analyzing the .apk file using androgaurd.



```
└─$ python3 cli.py analyze malware.apk
2022-11-14 21:33:18.953 | INFO     | androguard.session:__init__:58 - Opening database <Database(sqlite:///androguard.db)>
2022-11-14 21:33:18.968 | INFO     | androguard.session:__init__:67 - Creating new session [1]
2022-11-14 21:33:18.969 | INFO     | main:androlyze_main:257 - Please be patient, this might take a while.
2022-11-14 21:33:18.970 | INFO     | main:androlyze_main:261 - Found the provided file is of type 'APK'
2022-11-14 21:33:18.971 | INFO     | androguard.session:addAPK:140 - add APK malware.apk:d3c950ae2ad0e51127f271ea99931e823b70970279c0501525fd96e3aa2a10fc
2022-11-14 21:33:18.981 | INFO     | androguard.core.apk:_apk_analysis:313 - Starting analysis on AndroidManifest.xml
2022-11-14 21:33:19.010 | INFO     | androguard.core.apk:_apk_analysis:370 - APK file was successfully validated!
2022-11-14 21:33:19.026 | INFO     | androguard.session:addDEX:174 - add DEX:7e685b2e0d1d03d9edaf414c6ea576a635675bbce39998959dee64da2250404c
2022-11-14 21:33:19.310 | INFO     | androguard.session:addDEX:180 - added DEX:7e685b2e0d1d03d9edaf414c6ea576a635675bbce39998959dee64da2250404c
2022-11-14 21:33:19.310 | INFO     | androguard.core.analysis.analysis:add:1435 - Adding DEX file version 35
2022-11-14 21:33:20.868 | INFO     | androguard.core.analysis.analysis:add:1458 - Added DEX in the analysis took : 0min 01s
2022-11-14 21:33:21.339 | INFO     | androguard.core.analysis.analysis:create_xref:1492 - End of creating cross references (XREF) run time: 0min 00s
2022-11-14 21:33:21.339 | INFO     | androguard.session:addAPK:160 - added APK malware.apk:d3c950ae2ad0e51127f271ea99931e823b70970279c0501525fd96e3aa2a10fc
2022-11-14 21:33:21.339 | INFO     | main:androlyze_main:275 - Added file to session: SHA256::d3c950ae2ad0e51127f271ea99931e823b70970279c0501525fd96e3aa2a10fc
2022-11-14 21:33:21.339 | INFO     | main:androlyze_main:278 - Loaded APK file...
>>> filename
malware.apk
>>> a
<androguard.core.apk.APK object at 0×7f1369979e10>
>>> d
[<androguard.core.dex.DEX object at 0×7f13697f9510>]
>>> dx
<analysis.Analysis VMs: 1, Classes: 1267, Methods: 6809, Strings: 2176>

Androguard version 4.0 started
In [1]: a.get_signature_name()
Out[1]: 'META-INF/CERT.RSA'
```

Command: a.get_signature_name()
Signature name: ‘META-INF/CERT.RSA’

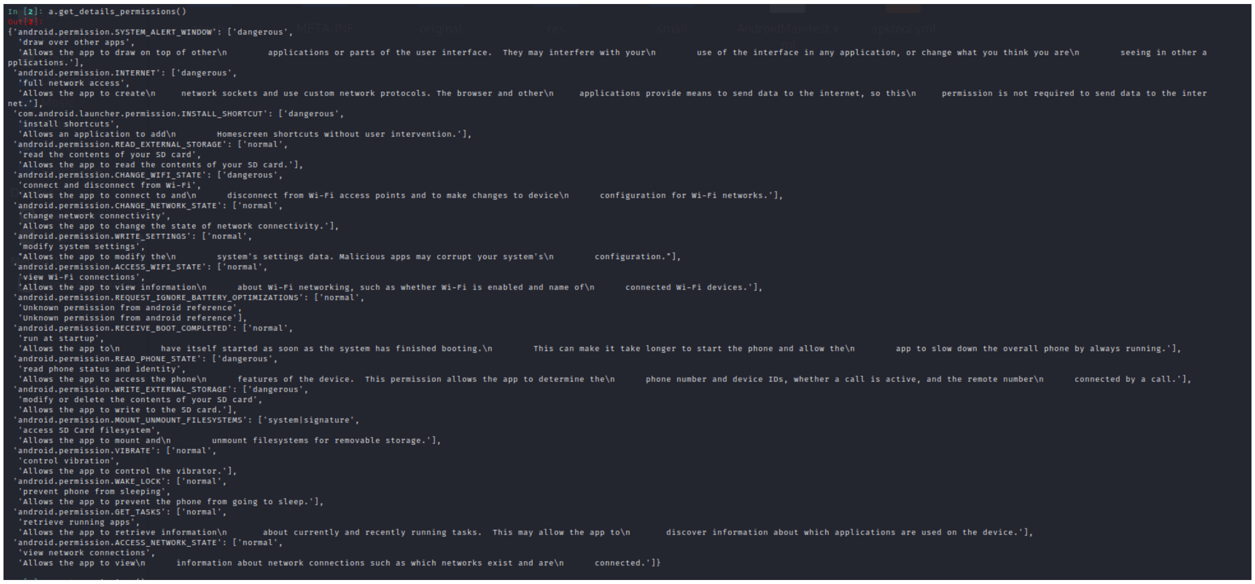
2. Permissions for the .apk



```
In [3]: a.get_permissions()
Out[3]:
['android.permission.SYSTEM_ALERT_WINDOW',
 'android.permission.INTERNET',
 'com.android.launcher.permission.INSTALL_SHORTCUT',
 'android.permission.READ_EXTERNAL_STORAGE',
 'android.permission.CHANGE_WIFI_STATE',
 'android.permission.CHANGE_NETWORK_STATE',
 'android.permission.WRITE_SETTINGS',
 'android.permission.ACCESS_WIFI_STATE',
 'android.permission.REQUEST_IGNORE_BATTERY_OPTIMIZATIONS',
 'android.permission.RECEIVE_BOOT_COMPLETED',
 'android.permission.READ_PHONE_STATE',
 'android.permission.WRITE_EXTERNAL_STORAGE',
 'android.permission.MOUNT_UNMOUNT_FILESYSTEMS',
 'android.permission.VIBRATE',
 'android.permission.WAKE_LOCK',
 'android.permission.GET_TASKS',
 'android.permission.ACCESS_NETWORK_STATE']
```

Command: a.get_permissions()

3. what app does (Normal part)



```
In [2]: a.get_details_permissions()
Out[2]:
{'android.permission.SYSTEM_ALERT_WINDOW': ['dangerous',
  'draw over other apps',
  'Allows the app to draw on top of other\n        applications or parts of the user interface.  They may interfere with your\n        use of the interface in any application, or change what you think you are\n        seeing in other a
pplications.'],
 'android.permission.INTERNET': ['dangerous',
  'full network access',
  'Allows the app to create\n     network sockets and use custom network protocols. The browser and other\n     applications provide means to send data to the internet, so this\n     permission is not required to send data to the inter
net.'],
 'com.android.launcher.permission.INSTALL_SHORTCUT': ['dangerous',
  'install shortcuts',
  'Allows an application to add\n        Homescreen shortcuts without user intervention.'],
 'android.permission.READ_EXTERNAL_STORAGE': ['normal',
  'read the contents of your SD card',
  'Allows the app to read the contents of your SD card.'],
 'android.permission.CHANGE_WIFI_STATE': ['dangerous',
  'connect and disconnect from Wi-Fi',
  'Allows the app to connect to and\n      disconnect from Wi-Fi access points and to make changes to device\n      configuration for Wi-Fi networks.'],
 'android.permission.CHANGE_NETWORK_STATE': ['normal',
  'change network connectivity',
  'Allows the app to change the state of network connectivity.'],
 'android.permission.WRITE_SETTINGS': ['normal',
  'modify system settings',
  "Allows the app to modify the\n        system's settings data. Malicious apps may corrupt your system's\n        configuration."],
 'android.permission.ACCESS_WIFI_STATE': ['normal',
  'view Wi-Fi connections',
  'Allows the app to view information\n      about Wi-Fi networking, such as whether Wi-Fi is enabled and name of\n      connected Wi-Fi devices.'],
 'android.permission.REQUEST_IGNORE_BATTERY_OPTIMIZATIONS': ['normal',
  'Unknown permission from android reference',
  'Unknown permission from android reference'],
 'android.permission.RECEIVE_BOOT_COMPLETED': ['normal',
  'run at startup',
  'Allows the app to\n        have itself started as soon as the system has finished booting.\n        This can make it take longer to start the phone and allow the\n        app to slow down the overall phone by always running.'],
 'android.permission.READ_PHONE_STATE': ['dangerous',
  'read phone status and identity',
  'Allows the app to access the phone\n      features of the device.  This permission allows the app to determine the\n      phone number and device IDs, whether a call is active, and the remote number\n      connected by a call.'],
 'android.permission.WRITE_EXTERNAL_STORAGE': ['dangerous',
  'modify or delete the contents of your SD card',
  'Allows the app to write to the SD card.'],
 'android.permission.MOUNT_UNMOUNT_FILESYSTEMS': ['system|signature',
  'access SD Card filesystem',
  'Allows the app to mount and\n        unmount filesystems for removable storage.'],
 'android.permission.VIBRATE': ['normal',
  'control vibration',
  'Allows the app to control the vibrator.'],
 'android.permission.WAKE_LOCK': ['normal',
  'prevent phone from sleeping',
  'Allows the app to prevent the phone from going to sleep.'],
 'android.permission.GET_TASKS': ['normal',
  'retrieve running apps',
  'Allows the app to retrieve information\n       about currently and recently running tasks.  This may allow the app to\n       discover information about which applications are used on the device.'],
 'android.permission.ACCESS_NETWORK_STATE': ['normal',
  'view network connections',
  'Allows the app to view\n      information about network connections such as which networks exist and are\n      connected.']}
```

As we can see the [‘normal’] in the screen shot. The permissions act normally and don’t steal the user data or access.

- Read external storage
- Access network location
- Write settings
- Battery optimization
- Phones vibrate
- Booting
- Wake lock
- Get tasks running on the mobile.

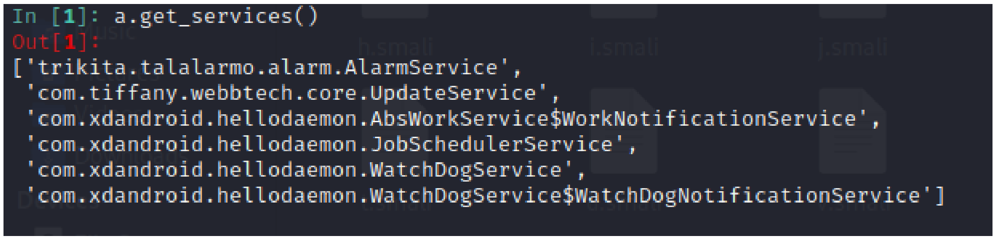
4. What app does (dangerous part)

In the above screenshot we can see [“Dangerous”]. The listed permissions can harm user device or can steal data or access.

- Alert window
- Internet – full access
- Change wifi state
- Read phone state
- Write external storage.

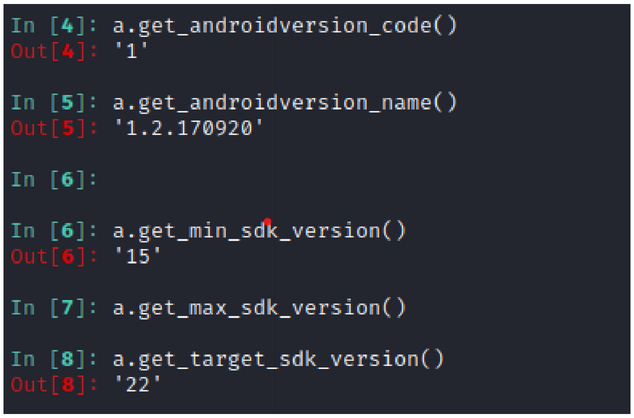
5. The picture tells about the services.

   Command: a.get_services()



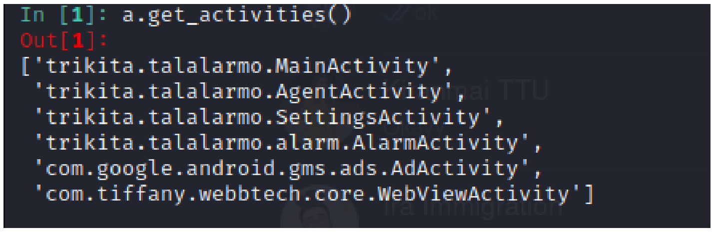
```
In [1]: a.get_services()
Out[1]:
['trikita.talalarmo.alarm.AlarmService',
 'com.tiffany.webbtech.core.UpdateService',
 'com.xdandroid.hellodaemon.AbsWorkService$WorkNotificationService',
 'com.xdandroid.hellodaemon.JobSchedulerService',
 'com.xdandroid.hellodaemon.WatchDogService',
 'com.xdandroid.hellodaemon.WatchDogService$WatchDogNotificationService']
```

6. The picture tells about the android version code, name and sdk version of min, max and target.



```
In [4]: a.get_androidversion_code()
Out[4]: '1'

In [5]: a.get_androidversion_name()
Out[5]: '1.2.170920'

In [6]:

In [6]: a.get_min_sdk_version()
Out[6]: '15'

In [7]: a.get_max_sdk_version()

In [8]: a.get_target_sdk_version()
Out[8]: '22'
```

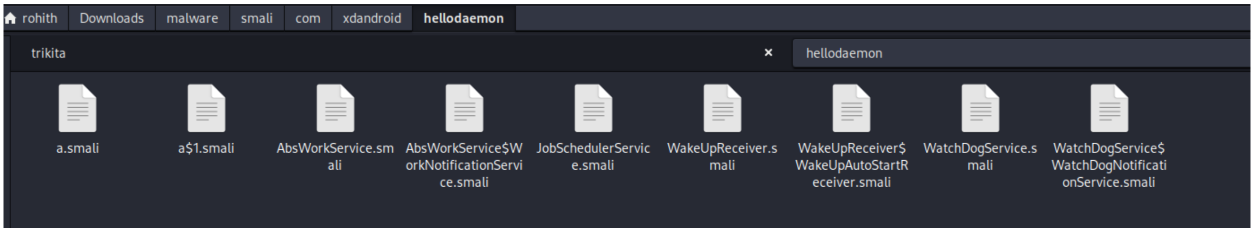
7. Activities performed by the app.



```
In [1]: a.get_activities()
Out[1]:
['trikita.talalarmo.MainActivity',
 'trikita.talalarmo.AgentActivity',
 'trikita.talalarmo.SettingsActivity',
 'trikita.talalarmo.alarm.AlarmActivity',
 'com.google.android.gms.ads.AdActivity',
 'com.tiffany.webbtech.core.WebViewActivity']
```

8. Decomplie the .apk file using apktool

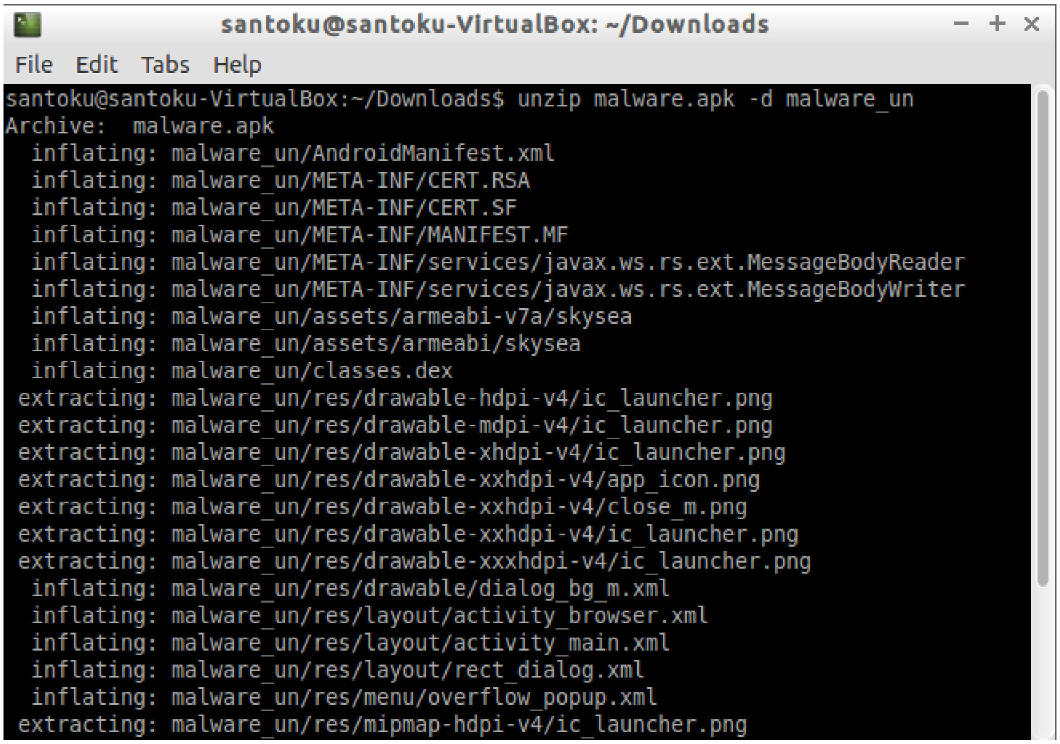
Command: apktool d malware.apk



After analyzing the decompiled .apk we can see WatchDogService files which allow malicious activites.

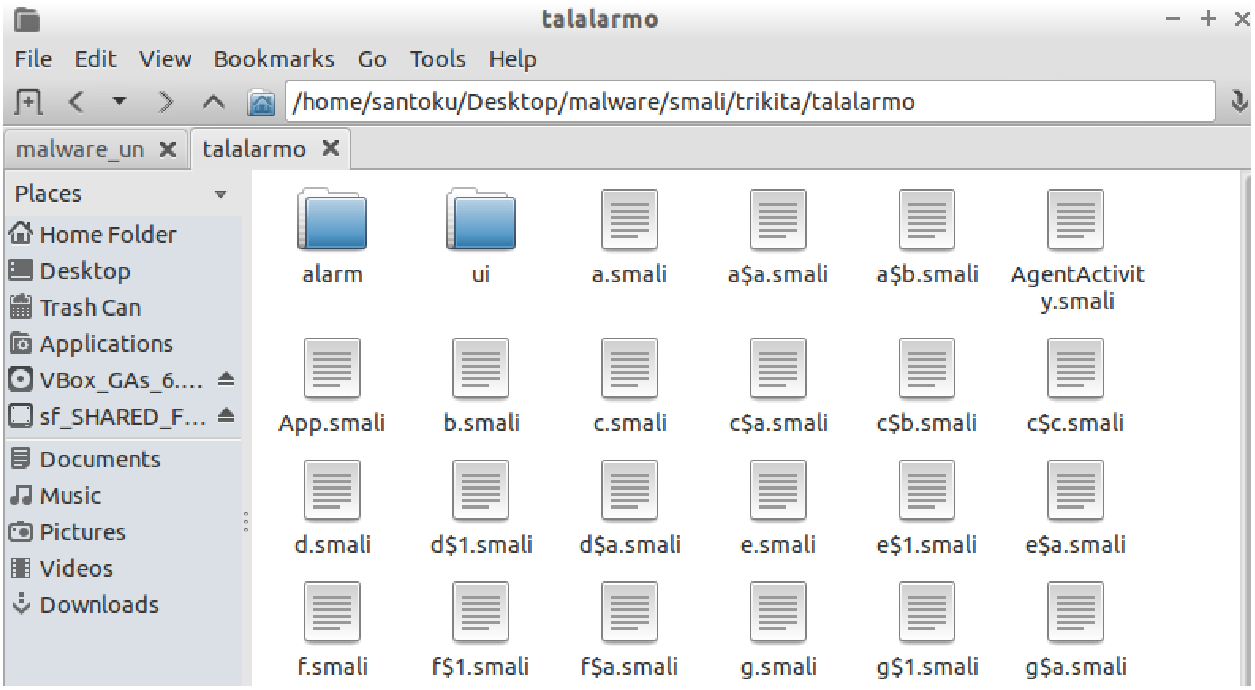
Watchdog is a rogue anti-spyware program from Rogue. FakeVimes family of computer infections.

9. Unzip the malware.apk using unzip command.



```
santoku@santoku-VirtualBox:~/Downloads$ unzip malware.apk -d malware_un
Archive:  malware.apk
  inflating: malware_un/AndroidManifest.xml
  inflating: malware_un/META-INF/CERT.RSA
  inflating: malware_un/META-INF/CERT.SF
  inflating: malware_un/META-INF/MANIFEST.MF
  inflating: malware_un/META-INF/services/javax.ws.rs.ext.MessageBodyReader
  inflating: malware_un/META-INF/services/javax.ws.rs.ext.MessageBodyWriter
  inflating: malware_un/assets/armeabi-v7a/skysea
  inflating: malware_un/assets/armeabi/skysea
  inflating: malware_un/classes.dex
 extracting: malware_un/res/drawable-hdpi-v4/ic_launcher.png
 extracting: malware_un/res/drawable-mdpi-v4/ic_launcher.png
 extracting: malware_un/res/drawable-xhdpi-v4/ic_launcher.png
 extracting: malware_un/res/drawable-xxhdpi-v4/app_icon.png
 extracting: malware_un/res/drawable-xxhdpi-v4/close_m.png
 extracting: malware_un/res/drawable-xxhdpi-v4/ic_launcher.png
 extracting: malware_un/res/drawable-xxxhdpi-v4/ic_launcher.png
  inflating: malware_un/res/drawable/dialog_bg_m.xml
  inflating: malware_un/res/layout/activity_browser.xml
  inflating: malware_un/res/layout/activity_main.xml
  inflating: malware_un/res/layout/rect_dialog.xml
  inflating: malware_un/res/menu/overflow_popup.xml
 extracting: malware_un/res/mipmap-hdpi-v4/ic_launcher.png
```

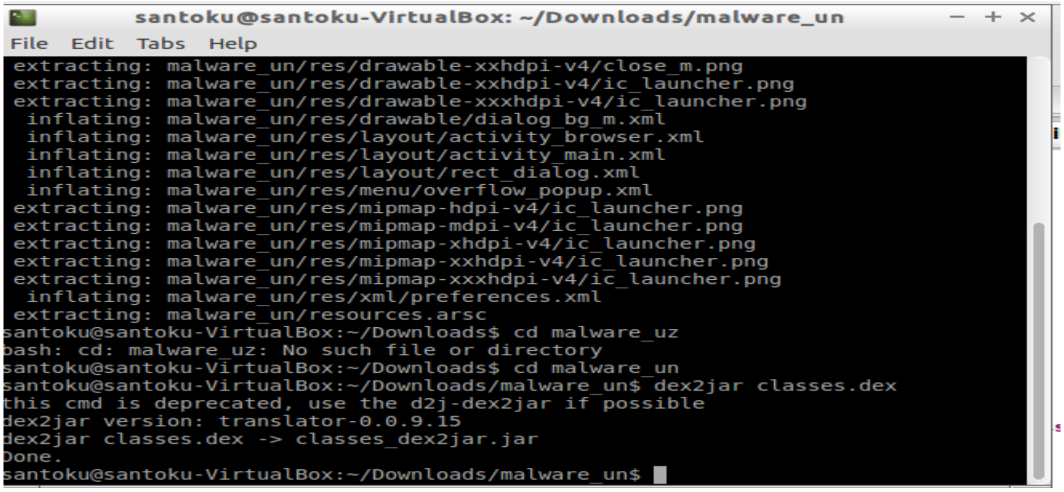
10. After unzipping, we can see some. smali files which are responsible for triggering malicious activities with the original functionality of the app.



11. Now, we can see classes.dex file in the malware_uz folder.

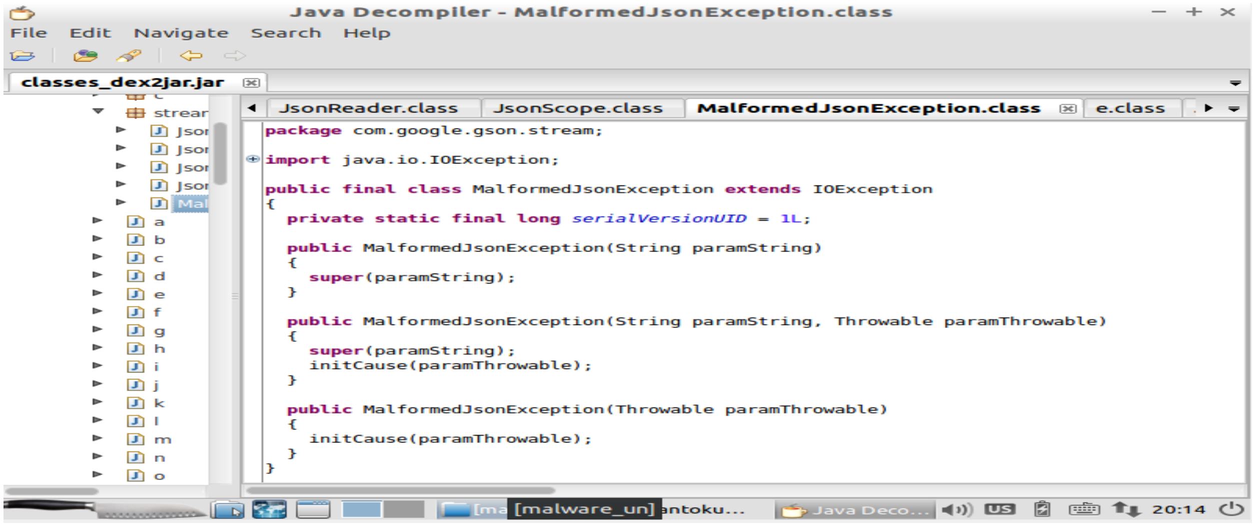
We can open the classes.dex2jar.jar file using JD-GUI.

Command: dex2jar classes.dex



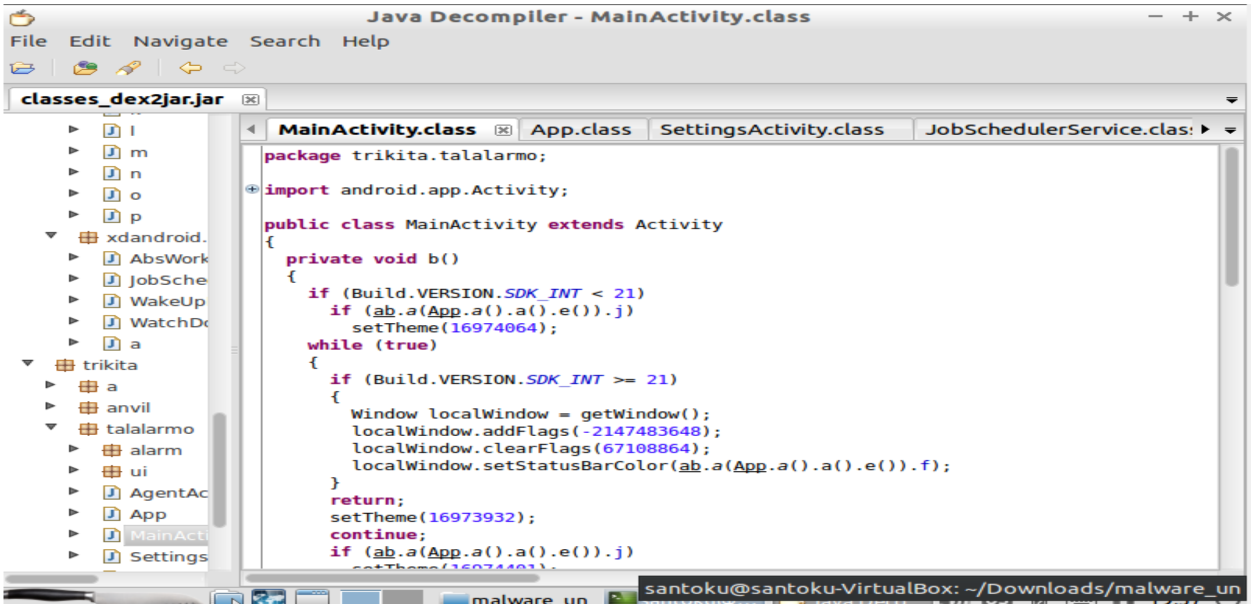
```
 extracting: malware_un/res/drawable-xxhdpi-v4/close_m.png
 extracting: malware_un/res/drawable-xxhdpi-v4/ic_launcher.png
 extracting: malware_un/res/drawable-xxxhdpi-v4/ic_launcher.png
  inflating: malware_un/res/drawable/dialog_bg_m.xml
  inflating: malware_un/res/layout/activity_browser.xml
  inflating: malware_un/res/layout/activity_main.xml
  inflating: malware_un/res/layout/rect_dialog.xml
  inflating: malware_un/res/menu/overflow_popup.xml
 extracting: malware_un/res/mipmap-hdpi-v4/ic_launcher.png
 extracting: malware_un/res/mipmap-mdpi-v4/ic_launcher.png
 extracting: malware_un/res/mipmap-xhdpi-v4/ic_launcher.png
 extracting: malware_un/res/mipmap-xxhdpi-v4/ic_launcher.png
 extracting: malware_un/res/mipmap-xxxhdpi-v4/ic_launcher.png
  inflating: malware_un/res/xml/preferences.xml
 extracting: malware_un/resources.arsc
santoku@santoku-VirtualBox:~/Downloads$ cd malware_uz
bash: cd: malware_uz: No such file or directory
santoku@santoku-VirtualBox:~/Downloads$ cd malware_un
santoku@santoku-VirtualBox:~/Downloads/malware_un$ dex2jar classes.dex
this cmd is deprecated, use the d2j-dex2jar if possible
dex2jar version: translator-0.0.9.15
dex2jar classes.dex -> classes_dex2jar.jar
Done.
santoku@santoku-VirtualBox:~/Downloads/malware_un$ 
```

12. We didn’t see any obfuscation in the code. Everything is clear but, some objects and classes are unclear like a,b and so on.



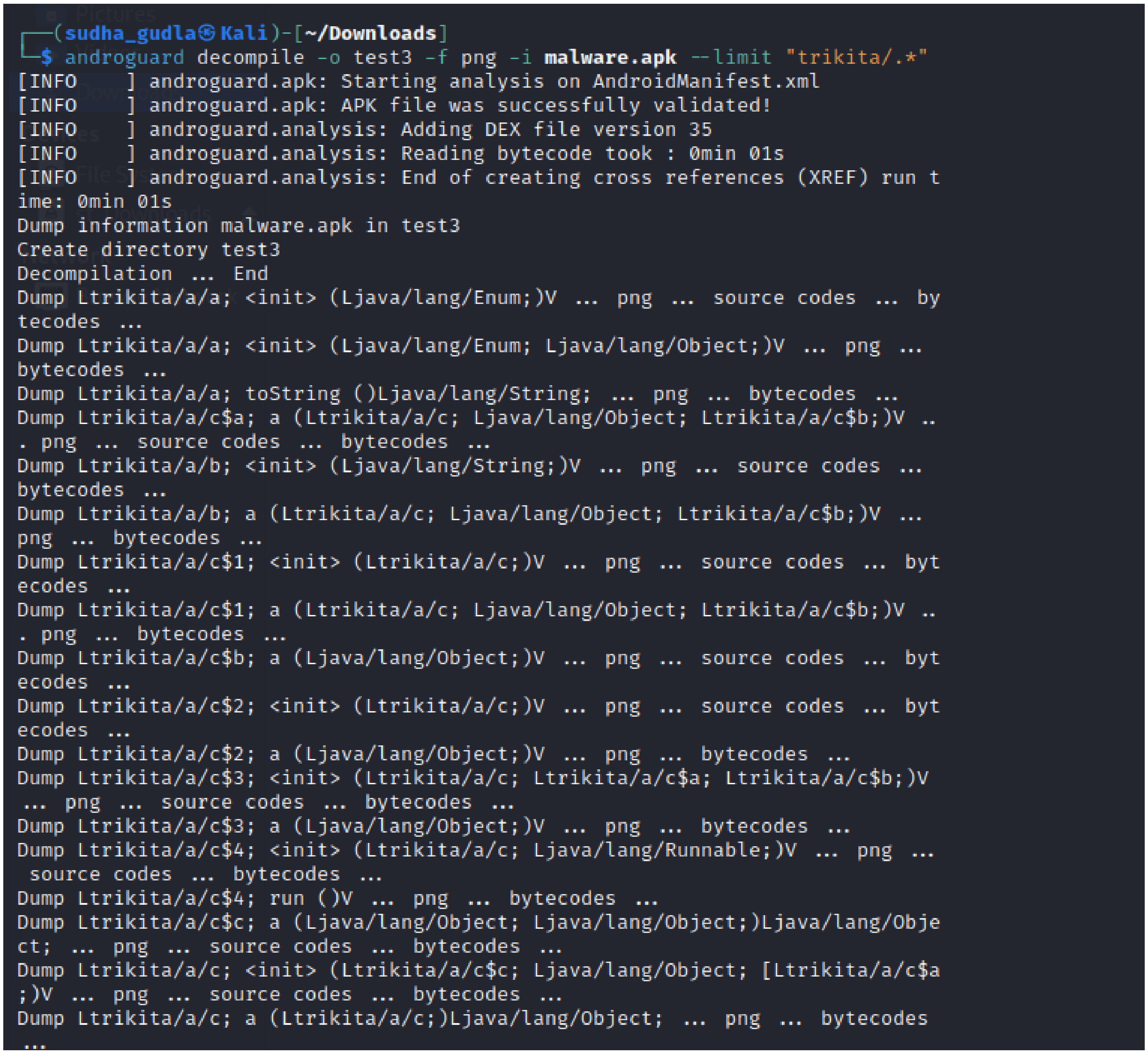
```
package com.google.gson.stream;

import java.io.IOException;

public final class MalformedJsonException extends IOException
{
  private static final long serialVersionUID = 1L;

  public MalformedJsonException(String paramString)
  {
    super(paramString);
  }

  public MalformedJsonException(String paramString, Throwable paramThrowable)
  {
    super(paramString);
    initCause(paramThrowable);
  }

  public MalformedJsonException(Throwable paramThrowable)
  {
    initCause(paramThrowable);
  }
}
```

13. We found the MainActivity file was in the .jar file where the program starts.



```
package trikita.talalarmo;

import android.app.Activity;

public class MainActivity extends Activity
{
  private void b()
  {
    if (Build.VERSION.SDK_INT < 21)
      if (ab.a(App.a().a().e()).j)
        setTheme(16974064);
    while (true)
    {
      if (Build.VERSION.SDK_INT >= 21)
      {
        Window localWindow = getWindow();
        localWindow.addFlags(-2147483648);
        localWindow.clearFlags(67108864);
        localWindow.setStatusBarColor(ab.a(App.a().a().e()).f);
      }
      return;
      setTheme(16973932);
      continue;
      if (ab.a(App.a().a().e()).j)
```

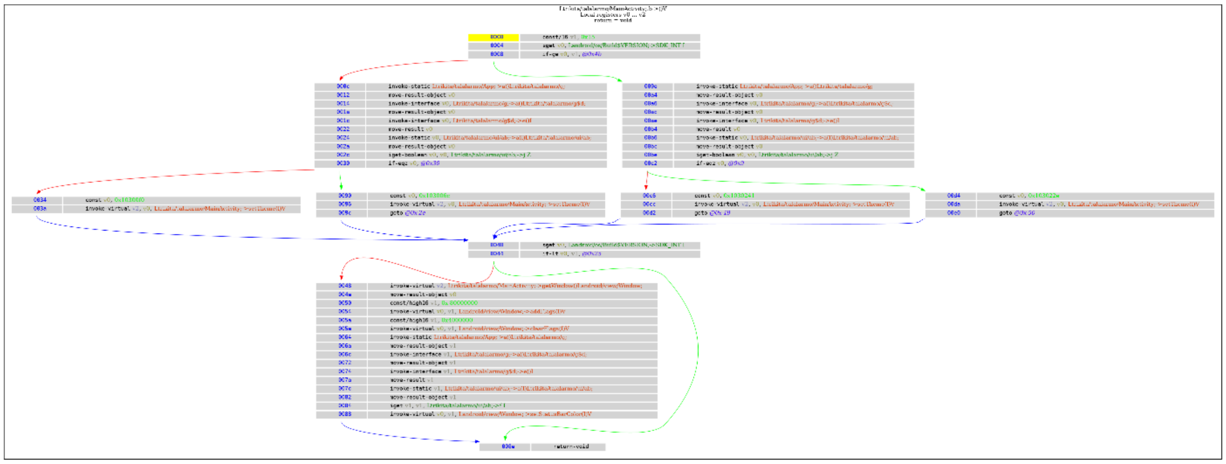
14. Created control graph of the .apk file using androgaurd.



```
┌──(sudha_gudla㉿Kali)-[~/Downloads]
└─$ androguard decompile -o test3 -f png -i malware.apk --limit "trikita/.*"
[INFO    ] androguard.apk: Starting analysis on AndroidManifest.xml
[INFO    ] androguard.apk: APK file was successfully validated!
[INFO    ] androguard.analysis: Adding DEX file version 35
[INFO    ] androguard.analysis: Reading bytecode took : 0min 01s
[INFO    ] androguard.analysis: End of creating cross references (XREF) run t
ime: 0min 01s
Dump information malware.apk in test3
Create directory test3
Decompilation ... End
Dump Ltrikita/a/a; <init> (Ljava/lang/Enum;)V ... png ... source codes ... by
tecodes ...
Dump Ltrikita/a/a; <init> (Ljava/lang/Enum; Ljava/lang/Object;)V ... png ...
bytecodes ...
Dump Ltrikita/a/a; toString ()Ljava/lang/String; ... png ... bytecodes ...
Dump Ltrikita/a/c$a; a (Ltrikita/a/c; Ljava/lang/Object; Ltrikita/a/c$b;)V ..
. png ... source codes ... bytecodes ...
Dump Ltrikita/a/b; <init> (Ljava/lang/String;)V ... png ... source codes ...
bytecodes ...
Dump Ltrikita/a/b; a (Ltrikita/a/c; Ljava/lang/Object; Ltrikita/a/c$b;)V ...
png ... bytecodes ...
Dump Ltrikita/a/c$1; <init> (Ltrikita/a/c;)V ... png ... source codes ... byt
ecodes ...
Dump Ltrikita/a/c$1; a (Ltrikita/a/c; Ljava/lang/Object; Ltrikita/a/c$b;)V ..
. png ... bytecodes ...
Dump Ltrikita/a/c$b; a (Ljava/lang/Object;)V ... png ... source codes ... byt
ecodes ...
Dump Ltrikita/a/c$2; <init> (Ltrikita/a/c;)V ... png ... source codes ... byt
ecodes ...
Dump Ltrikita/a/c$2; a (Ljava/lang/Object;)V ... png ... bytecodes ...
Dump Ltrikita/a/c$3; <init> (Ltrikita/a/c; Ltrikita/a/c$a; Ltrikita/a/c$b;)V
 ... png ... source codes ... bytecodes ...
Dump Ltrikita/a/c$3; a (Ljava/lang/Object;)V ... png ... bytecodes ...
Dump Ltrikita/a/c$4; <init> (Ltrikita/a/c; Ljava/lang/Runnable;)V ... png ...
 source codes ... bytecodes ...
Dump Ltrikita/a/c$4; run ()V ... png ... bytecodes ...
Dump Ltrikita/a/c$c; a (Ljava/lang/Object; Ljava/lang/Object;)Ljava/lang/Obje
ct; ... png ... source codes ... bytecodes ...
Dump Ltrikita/a/c; <init> (Ltrikita/a/c$c; Ljava/lang/Object; [Ltrikita/a/c$a
;)V ... png ... source codes ... bytecodes ...
Dump Ltrikita/a/c; a (Ltrikita/a/c;)Ljava/lang/Object; ... png ... bytecodes
 ...
```

Command: androgaurd decompile -o outputfolder -f format png -I .apk file –limit “file where you wanter the graph”

15. The graph of the main activity tells about the classes and objects which are accessible



```
Ltrikita/talalarmo/MainActivity;.b ()V
Local registers v0 ... v2
return = void

0000  const/16 v1, 0x15
0004  sget v0, Landroid/os/Build$VERSION;->SDK_INT I
0008  if-ge v0, v1, @0x4b

000c  invoke-static Ltrikita/talalarmo/App;->a()Ltrikita/talalarmo/g;
0012  move-result-object v0
0014  invoke-interface v0, Ltrikita/talalarmo/g;->a()Ltrikita/talalarmo/g$d;
001a  move-result-object v0
001c  invoke-interface v0, Ltrikita/talalarmo/g$d;->a()I
0022  move-result v0
0024  invoke-static v0, Ltrikita/talalarmo/ui/ab;->a(I)Ltrikita/talalarmo/ui/ab;
002a  move-result-object v0
002c  iget-boolean v0, v0, Ltrikita/talalarmo/ui/ab;->j Z
0030  if-eqz v0, @0x30

009e  invoke-static Ltrikita/talalarmo/App;->a()Ltrikita/talalarmo/g;
00a4  move-result-object v0
00a6  invoke-interface v0, Ltrikita/talalarmo/g;->a()Ltrikita/talalarmo/g$d;
00ac  move-result-object v0
00ae  invoke-interface v0, Ltrikita/talalarmo/g$d;->a()I
00b4  move-result v0
00b6  invoke-static v0, Ltrikita/talalarmo/ui/ab;->a(I)Ltrikita/talalarmo/ui/ab;
00bc  move-result-object v0
00be  iget-boolean v0, v0, Ltrikita/talalarmo/ui/ab;->j Z
00c2  if-eqz v0, @0x9

0034  const v0, 0x10300f0
003a  invoke-virtual v2, v0, Ltrikita/talalarmo/MainActivity;->setTheme(I)V

0090  const v0, 0x103006e
0096  invoke-virtual v2, v0, Ltrikita/talalarmo/MainActivity;->setTheme(I)V
009c  goto @0x-2e

00c6  const v0, 0x1030241
00cc  invoke-virtual v2, v0, Ltrikita/talalarmo/MainActivity;->setTheme(I)V
00d2  goto @0x-49

00d4  const v0, 0x103022e
00da  invoke-virtual v2, v0, Ltrikita/talalarmo/MainActivity;->setTheme(I)V
00e0  goto @0x-50

0040  sget v0, Landroid/os/Build$VERSION;->SDK_INT I
0044  if-lt v0, v1, @0x25

0048  invoke-virtual v2, Ltrikita/talalarmo/MainActivity;->getWindow()Landroid/view/Window;
004e  move-result-object v0
0050  const/high16 v1, 0x80000000
0054  invoke-virtual v0, v1, Landroid/view/Window;->addFlags(I)V
005a  const/high16 v1, 0x4000000
005e  invoke-virtual v0, v1, Landroid/view/Window;->clearFlags(I)V
0064  invoke-static Ltrikita/talalarmo/App;->a()Ltrikita/talalarmo/g;
006a  move-result-object v1
006c  invoke-interface v1, Ltrikita/talalarmo/g;->a()Ltrikita/talalarmo/g$d;
0072  move-result-object v1
0074  invoke-interface v1, Ltrikita/talalarmo/g$d;->a()I
007a  move-result v1
007c  invoke-static v1, Ltrikita/talalarmo/ui/ab;->a(I)Ltrikita/talalarmo/ui/ab;
0082  move-result-object v1
0084  iget v1, v1, Ltrikita/talalarmo/ui/ab;->f I
0088  invoke-virtual v0, v1, Landroid/view/Window;->setStatusBarColor(I)V

008e  return-void
```