Antiy Labs

A Hidden Way of Malware on Android

Antiy Labs

August 8, 2013

Contents

Backgroud

In Android operation system, APK is the ZIP format file that contains several normal files and executable files. In a normal APK file, the compressed root directory includes a DEX executable file named classes.dex, and it may contain a shared object file or several shared object files with ELF format. If there are other executable files or shared object files with the format of APK, DEX or ELF at different locations of the APK file, then we call it abnormal executable file.

When detecting malware, the security software would not only carry out feature matching detection among APK, classes.dex and relevant shared object files, but also detect the feature of abnormal executable file.

Here we make the sample of egdata family as an example to introduce how to hide the abnormal executable file in order to avoid the detection by security software against the relevant malware files and make the detection more difficult.

The Tampered Calender -- egdata.a

Sample egdata.a is a calendar application that has been tampered with and repacked by the attacker, which would prompt program updates when it is running; however, the updates would fail due to the different signatures.

Figure 1 Screenshot of running egdata.a

Comparing the sample APK file format with the official application format, we found the sample added one more eg.data file in /assets directory. After identifying the file head of eg.data, we discovered the beginning two bytes are PK and the root directory would contain AndroidManifest.xml and classes.dex after decompression, which meant it is the standard APK

file.

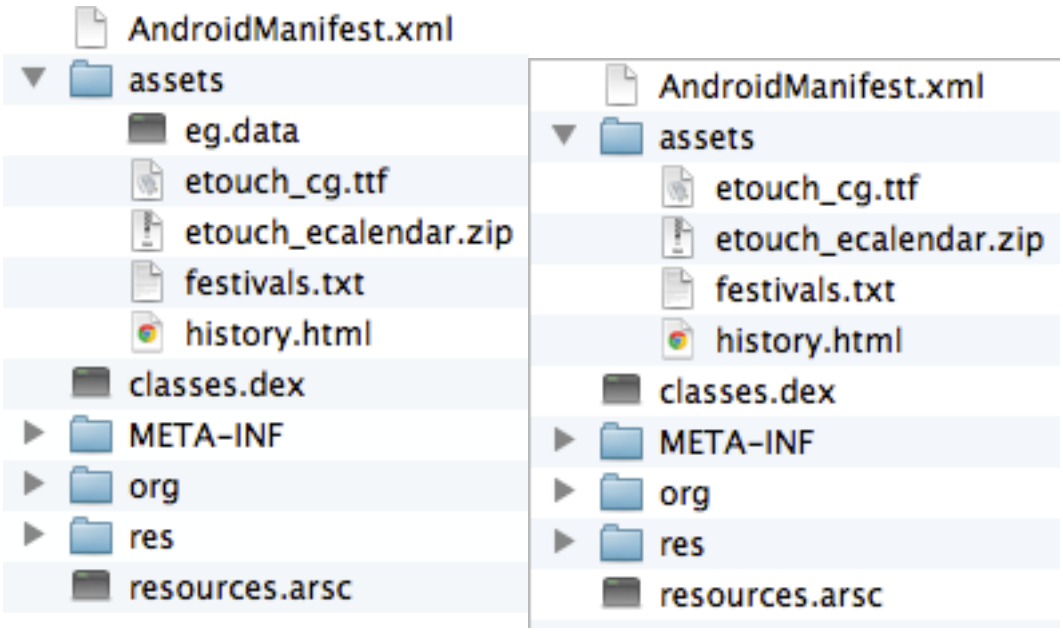

Figure 2 Format comparison between egdata.a (Left) and the official application (Right)

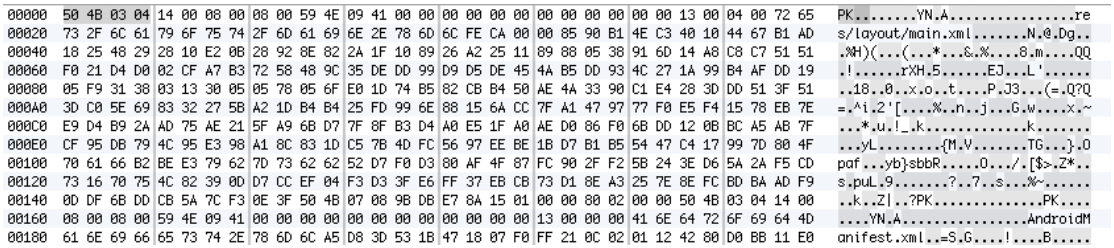

Figure3 eg.data file content

After analyzing sample APK, the method createSingleInstall() in the class com.android.commond.Egrecvol extracted eg.data from /assets.

```
InputStream v6 = Egrecvol.context.getAssets().open("eg.data");
FileOutputStream v7 = new FileOutputStream(this.fJar);
Egrecvol.Log("eg.data len=" + v6.available());
while (true) {
  v1 = new byte[1024];
  v9 = v6.read(v1);
  if (v9 > 0) {
    goto lable_142;
  }
    break;
lable_142:
    v7.write(v1, 0, v9);
}

v6.close();
```

```
v7.close();
```

After releasing eg.data, the way to dynamically load classes is as follows:

- Use DexClassLoader to dynamically load the released file eg.data, return ClassLoader.
- Call loadClass() to load specific class, here is the class name "com.suntu.engine3.engine.Main1".
- Get the Constructor.
- Call newInstance; the malware is completely called by now.

When dynamically loading and executing APK file eg.data, it will execute the method realeseFile() in the class com.suntu.engine3.engine.jni.JNIEngine to release .so local shared object file. The real content of the released file was stored in Java code as byte array. The following is a snippet of the array:

```
static
{
  byte[] arrayOfByte = new byte[5556];
  arrayOfByte[0] = 127;
  arrayOfByte[1] = 69;
  arrayOfByte[2] = 76;
  arrayOfByte[3] = 70;
  arrayOfByte[4] = 1;
  arrayOfByte[5] = 1;
  arrayOfByte[6] = 1;
  arrayOfByte[16] = 3;
  arrayOfByte[18] = 40;
  arrayOfByte[20] = 1;
  arrayOfByte[24] = -116;
  arrayOfByte[25] = 9;
  arrayOfByte[28] = 52;
  arrayOfByte[32] = 12;
  arrayOfByte[33] = 19;
  arrayOfByte[36] = 2;
  arrayOfByte[39] = 5;
  arrayOfByte[40] = 52;
  arrayOfByte[42] = 32;
  arrayOfByte[44] = 5;
  arrayOfByte[46] = 40;
  arrayOfByte[48] = 17;
  arrayOfByte[50] = 16;
  arrayOfByte[52] = 1;
  arrayOfByte[55] = 112;
```

```
arrayOfByte[56] = -72;
arrayOfByte[57] = 16;
arrayOfByte[60] = -72;
arrayOfByte[61] = 16;
arrayOfByte[64] = -72;
arrayOfByte[65] = 16;
arrayOfByte[68] = 72;
arrayOfByte[72] = 72;
arrayOfByte[76] = 4;
arrayOfByte[80] = 4;
arrayOfByte[84] = 1;
arrayOfByte[101] = 17;
```

In this sample, it still adopted the normal exception-added APK file and the dynamic load method [1] ,however, it hides the abnormal shared object file by way of storing .so shared object file content in the code.

The Infected Kuwo Music Player — Variant egdata.c

Variant egdata.c is a Kuwo Music Player application [2] that was tampered with by attackers, they made it more difficult for security software to extract and identify features by adopting a more covert method to hide the APK file that contains malware.

The APK file format is as follows:

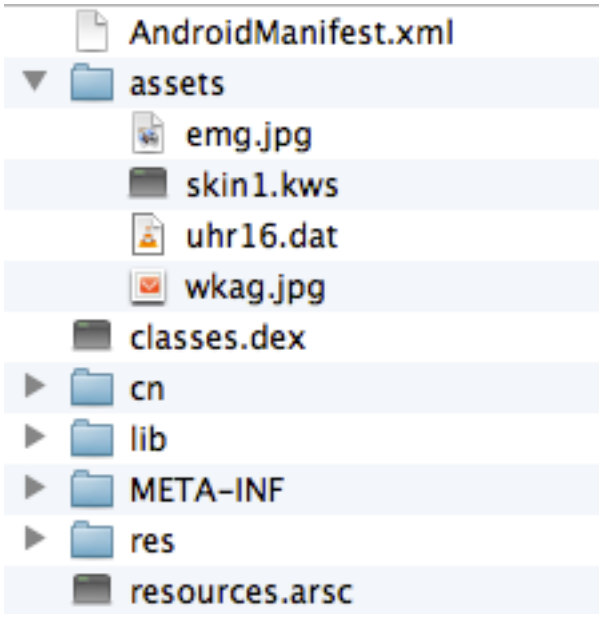

Figure 4 APK formate of egdata.c

There are two jpg image files named emg.jpg and wkag.jpg in /assets directory, of which image emg.jpg cannot present properly as an image. The method releaseClassData(), in the variant APK class cn.kuwo.player.MainActivityyb, extracting and loading APK file from file

wkag.jpg. The covert APK file content is from the 1024 byte offset in file wkag.jpg to the end of the file with one byte reduction of every byte.

```
InputStreamv0_1=
MainActivityyb.context.getAssets().open(MainActivityyb.eg1data);
FileOutputStream v1 = new FileOutputStream(MainActivityyb.fJar);
int v2 = Integer.parseInt(MainActivityyb.picLen);
int v3;
for (v3 = 0; v3 < v2; ++v3) {
  v0_1.read();
}

while (true) {
  byte[] v2_1 = new byte[1024];
  v3 = v0_1.read(v2_1);
  if (v3 > 0) {
    goto label_72;
  }

  break;
label_72:
  int v4;
  for (v4 = 0; v4 < v3; ++v4) {
    v2_1[v4] = ((byte)(v2_1[v4] - 1));
  }

  v1.write(v2_1, 0, v3);
}
v0_1.close();
v1.close();
```

The method createSingleInstall(), under the class of cn.kuwo.player.MainActivitygxwa, extracting the main APK that contains malware from image emg.jpg. The APK content here is from the 8 byte offset with one byte reduction of every byte.

```
v0_2 = MainActivitygxwa.context.getAssets().open(MainActivitygxwa.egdata);
v1_1 = new FileOutputStream(this.fJar);
MainActivitygxwa.Log(“eg.data len=” + v0_2.available();
v0_2.read(new byte[8]);
while (true) {
  v2 = new byte[1024];
  v3 = v0_2.read(v2);
  if (v3 > 0) {
    goto label_168;
  }
```

```
  goto label_148;
}

label_168:
  int v4 = 0;
  while (true) {
    if (v4 >= v3) {
      goto lable_177;
    }

    try {
      v2[v4] = ((byte)(v2[v4] - 1));
      ++v4;
      continue;
    } catch(Exception v0) {
    }
  }
```

When releasing the final malware, variant egdata.c experienced two steps in which it extracted and dynamically loaded APK file from image files. Instead of the normal exception-added executable file method, it chose to insert the malicious APK file into other normal type files and adopt encryption switch to hide the unique feature information of APK file so that it realized the goal of covering itself.

Summary

According to the analysis on the two samples of egdata family, the summary of the hidden way and detection difficulty can be shown in the following table:

	egdata.a	egdata.c
Hidden Way	1. Abnormal APK：eg.data 2. Byte array stores the file content of .so	1. Image files hide the malicious APK 2. Store after APK byte alternation.
Detection Difficulty	Easy for eg.data; hard for .so	Hard to extract features and detect.

Variant egdata.c maintains the malware functionality, but it changes greatly on the method of extracting and releasing the main file that contains malware in order to make it more difficult for security vendors to extract and identify the features.

Reference

[1] http://www.cnblogs.com/crazypebble/archive/2011/04/13/2014582.html

[2] http://blog.csdn.net/cqupt_chen/article/details/9012929

Any technical information that is made available by Antiy Labs is the copyrighted work of Antiy Labs and is owned by Antiy Labs. NO WARRANTY. Antiy Labs makes no warranty as to this document's accuracy or use. The information in this document may include typographical errors or inaccuracies, and may not reflect the most current developments; and Antiy Labs does not represent, warrant or guarantee that it is complete, accurate, or up-to-date, nor does Antiy Labs offer any certification or guarantee with respect to any opinions expressed herein or any references provided. Changing circumstances may change the accuracy of the content herein. Opinions presented in this document reflect judgment at the time of publication and are subject to change. Any use of the information contained in this document is at the risk of the user. Antiy Labs assumes no responsibility for errors, omissions, or damages resulting from the use of or reliance on the information herein. Antiy Labs reserves the right to make changes at any time without prior notice.

About Antiy Labs

Antiy Labs is an antivirus vendor which makes advanced research and technology contributions to the field. Currently, there are tens of thousands of firewalls, UTM and security devices deployed with our antivirus engine. More information is available at www.antiy.net.

Antiy Labs

Copyright ©2012 Antiy Labs. All rights reserved