MOBILE THREAT HUNTING: MALICIOUS APK DEFORMING ZIP FILE FORMAT FOUND UNDER EXPERIMENT IN THE WILD

INTRODUCTION

As smartphones has become widely popular these days, mobile malware is also seeping into our lives. A small hand-held device holds almost every information related to its owner. Except for a few users who freely use high-tech devices, we are still exposed to malware threats, as same as the PC.

Since 2017, the Financial Security Institute has been tracking and responding to Voice phishing, which is a type of financial fraud, impersonating financial institutes to mislead victims to install malicious apps which steal device information and results in actual financial loss via social engineering. Malicious apps are always involved in this crime process, while a new anti-analysis method based on the ZIP file format was observed in a sample collected in the wild at mid-August.

The anti-analysis app installs and runs normally without errors in real machines or emulators, but gives an error when static analysis tools try to analyze it. This sample was intended to hinder analysis, so we are to share the process and result of the root cause analysis.

ROOT CAUSE ANALYSIS

We first noticed that real machines and emulators had no problems, but static analysis tools failed with errors. In other words, execution was not a problem but tools could not process the sample, which led to the assumption that it was a file processing problem.

UNPACKING THE APK FILE

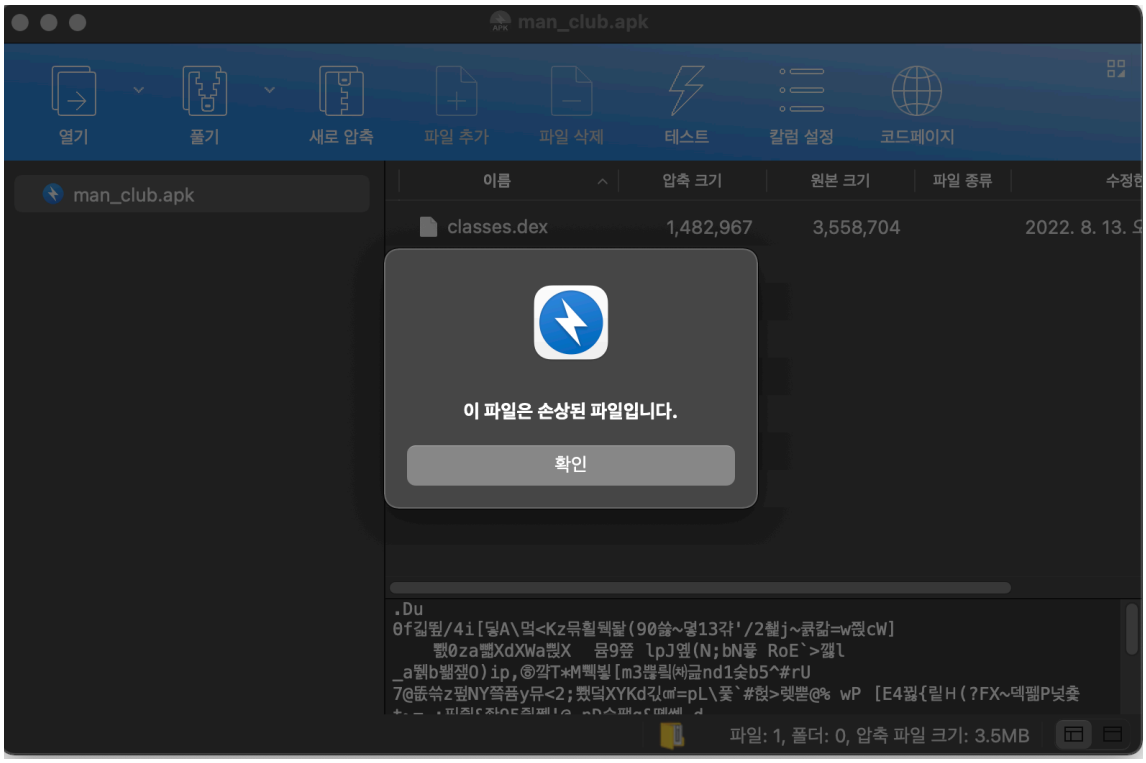
Upon trying to decompress the file using the unzip command, the command terminated without decompressing the file normally.

```
❯ unzip man_club.apk
Archive:  man_club.apk
.Du
   ??f?y??/?4i[?LA\??E<Kz??C????(9???~?v?^[?13?d'/2??j~?g???=^[w?scW?]  ?B0za?HXdXWa?zX ??9?l l?pJ???(N;bN
?F?  RoE`>??l

            _a?Zb?w?JO?)ip,??? T*M?R??[m3?b???âInd1?Gb?5^?#rU
        7@?e?3?z?jNY?j??y??<? 2?;?J?DXYKd?u??=pL?\??`?#?C>?????@% ?wP ?[E??{????(?7FX~???rP???nt~=???:???e&
??95?S?Y'@_nD???kg&?J??-dD_? ?X??t:?,`N??|? 1<@bw+"

                                                ??G'S{?5)G2??-?#zQ$/???Y?^lii??r*a ?       ??oG6{&?ZI}?
uH?"
                                                      ??????
```

We also tried a domestic-wide compressing utility for decompression, but we were not able to see the internal APK components, and only got a "This file is damaged" message.



Apktool, A well-known APK analysis tool also failed to proceed with errors.

```
fsi@fsiui-MacBookPro-2 apktool % java -jar apktool_2.6.1.jar d  man_club.apk
I: Using Apktool 2.6.1 on man_club.apk
I: Loading resource table...
W: Skipping package group: mt.work.service
I: Decoding AndroidManifest.xml with resources...
Exception in thread "main" brut.androlib.err.RawXmlEncounteredException: Could not decode XML
        at brut.androlib.res.decoder.XmlPullStreamDecoder.decode(XmlPullStreamDecoder.java:145)
        at brut.androlib.res.decoder.XmlPullStreamDecoder.decodeManifest(XmlPullStreamDecoder.java:151)
        at brut.androlib.res.decoder.ResFileDecoder.decodeManifest(ResFileDecoder.java:159)
        at brut.androlib.res.AndrolibResources.decodeManifestWithResources(AndrolibResources.java:193)
        at brut.androlib.Androlib.decodeManifestWithResources(Androlib.java:141)
        at brut.androlib.ApkDecoder.decode(ApkDecoder.java:109)
        at brut.apktool.Main.cmdDecode(Main.java:175)
        at brut.apktool.Main.main(Main.java:79)
Caused by: java.io.IOException: Expected: 0x00080003 or 0x00080001, got: 0x00080000
        at brut.util.ExtDataInput.skipCheckInt(ExtDataInput.java:45)
        at brut.androlib.res.decoder.AXmlResourceParser.doNext(AXmlResourceParser.java:808)
        at brut.androlib.res.decoder.AXmlResourceParser.next(AXmlResourceParser.java:98)
        at brut.androlib.res.decoder.AXmlResourceParser.nextToken(AXmlResourceParser.java:108)
        at org.xmlpull.v1.wrapper.classic.XmlPullParserDelegate.nextToken(XmlPullParserDelegate.java:105)
        at brut.androlib.res.decoder.XmlPullStreamDecoder.decode(XmlPullStreamDecoder.java:138)
        ... 7 more
fsi@fsiui-MacBookPro-2 apktool %
```

WHY IS THE APK NOT UNPACKING PROPERLY?

APK files are based on the ZIP file format. The ZIP file format is one of the lossless data compression schemes, using a sophisticated file format since 1 bit of error can cause a large effect on the original data.

Static analysis tools can be seen as a delicate parser which provides accurate information to the analyst. We assumed that the analysis tool failed to process and extract the internal APK components for unknown reasons, and continued to file format analysis.

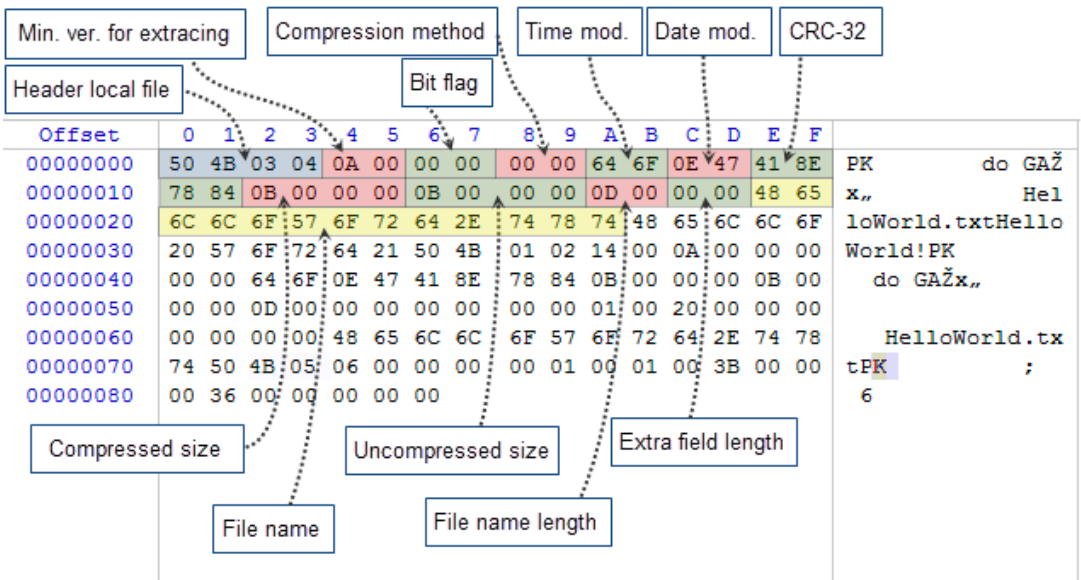
OVERALL ZIP FILE FORMAT

As APK Files are based on the ZIP file format, analysis tools should directly implement functions to process ZIP files, or indirectly, use external libraries. The ZIP file structure is divided into multiple areas as follows. The original data cannot be tampered to guarantee the functionality of the malicious app, so we decided to check areas beyond the file data.

```
[local file header 1]
[encryption header 1]
[file data 1]
[data descriptor 1]
.
.
.
[local file header n]
[encryption header n]
[file data n]
[data descriptor n]
[archive decryption header]
[archive extra data record]
[central directory header 1]
.
.
.
[central directory header n]
[zip64 end of central directory record]
[zip64 end of central directory locator]
[end of central directory record]
```

OVERVIEW OF THE LOCAL FILE HEADER

The local file header is constructed of sub-fields as follows. Some of these fields cannot be changed without losing the ZIP file's unique characteristics, which leads to a conclusion that the tampered fields should be unrelated to the ZIP functionality.



To confirm this, we compared the sample with a normal APK file which was able to be analyzed using existing tools.

Name	Value
struct ZIPFILERECORD record	META-INF/MANIFEST.MF
char frSignature[4]	PK
ushort frVersion	20
ushort frFlags	2056
enum COMPTYPE frCompression	COMP_DEFLATE (8)
DOSTIME frFileTime	14:09:42
DOSDATE frFileDate	06/06/2022
uint frCrc	0h
uint frCompressedSize	0
uint frUncompressedSize	0

Comparing the local file headers of a normal APK and our sample, we found out that the field values of frVersion, frFlags, frCompression were different, as shown in the image. To dive further into field byte units and ranges, check the official document[1].

struct ZIPFILERECORD record[0]	AndroidManifest.xml
char frSignature[4]	PK
ushort frVersion	28963
ushort frFlags	60495
enum COMPTYPE frCompression	-18527
DOSTIME frFileTime	
DOSDATE frFileDate	05/03/2077
uint frCrc	5A5312F5h
uint frCompressedSize	1242
uint frUncompressedSize	3700

* The file time is also different, but as file time does not have a significant effect on the execution process, we have left this field out for this analysis.

version needed to extract (frVersion)

The official document defines this field to have a value from 1.0 to 6.3, which is the version needed for extraction, while the malicious APK has given this field a value of 28963.

general purpose bit flag (frFlags)

Multiple flags are defined to check the current status, and the malicious APK has given this field a value of 60495, which is out of range.

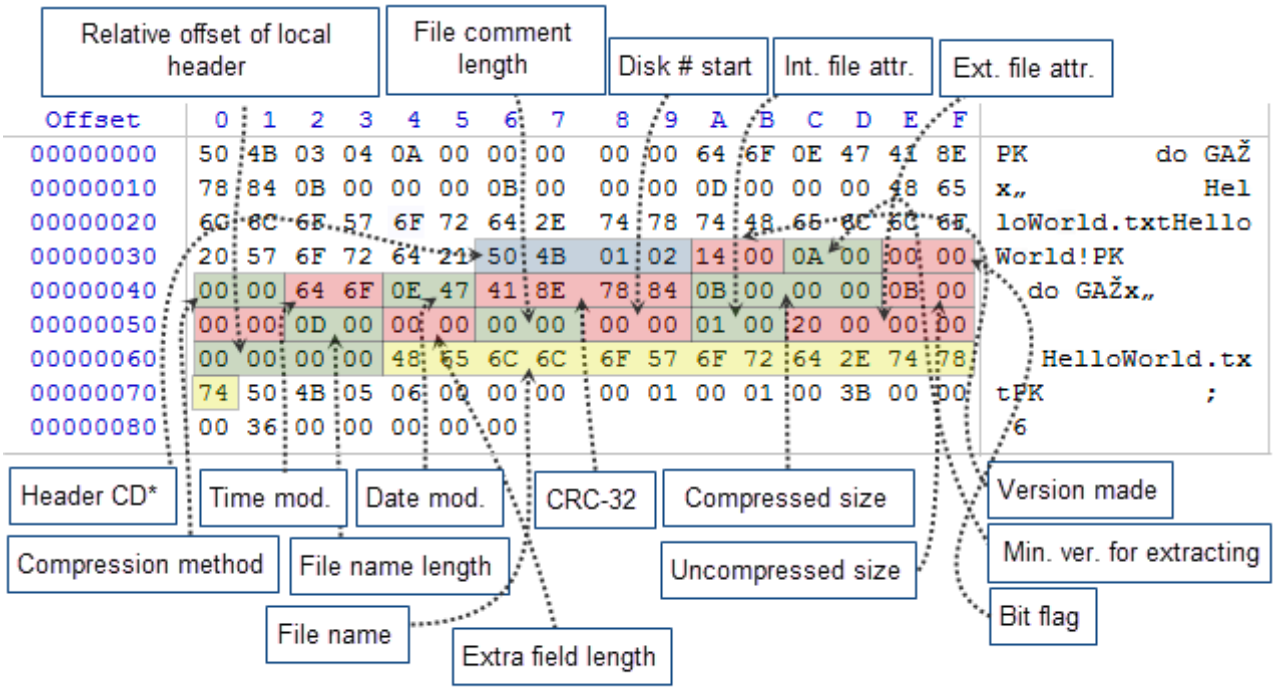
compression method (frCompression)

This defines the compression method of the ZIP file, and the malicious APK has given this field a value of -18527, which is out of range.

1) https://pkware.cachefly.net/webdocs/casestudies/APPNOTE.TXT

OVERVIEW OF THE CENTRAL DIRECTORY HEADER

The ZIP file has another Central Directory Header, which holds many fields which overlap with the local file header.

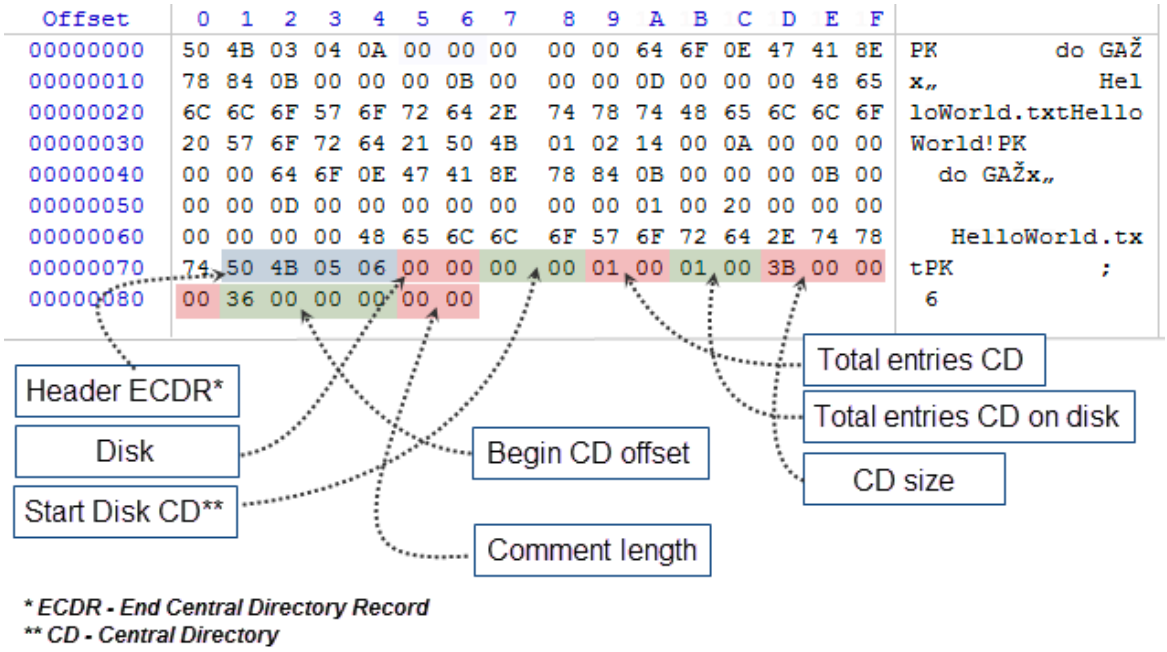


The field values of the malicious APK's central directory header have normal ranged values, unlike the local file header. There are also fields which both exist in the central directory header and the local file header, and these fields have the same values in normal APK files.

This implies that we can respond to the malicious APK by amending the local file header fields with the central directory header field values.

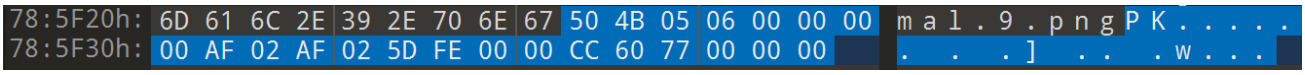
OVERVIEW OF THE END OF CENTRAL DIRECTORY RECORD

The End of central directory record is structured as follows.

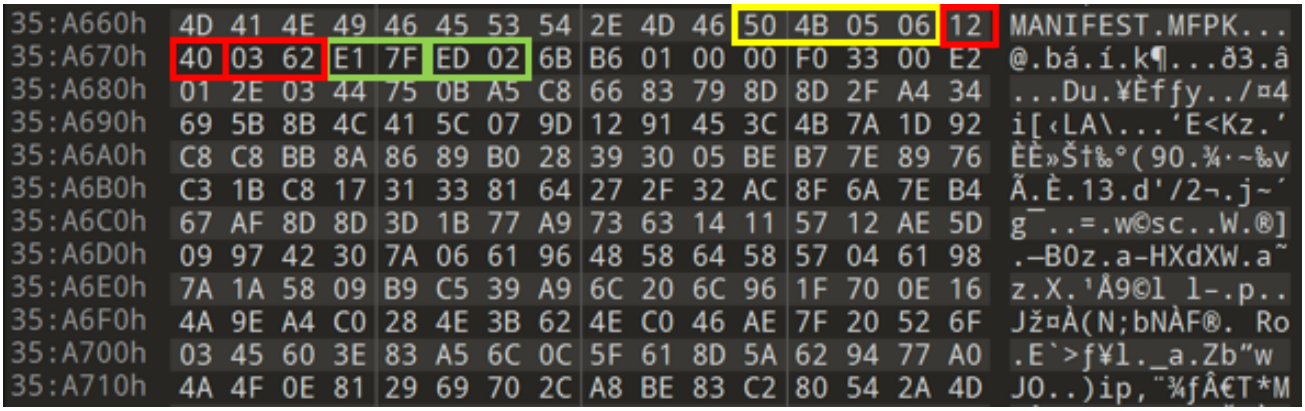


The End of central directory record area is the last area of a zip file, and is first referenced on opening a zip file. This area consists the information of central directories such as count, size, and starting offset.

By comparing a normal APK and the malicious sample, we found out that the 'number of this disk' field was tampered, and the central directory count was different in particular.



* End of central directory record of normal APK



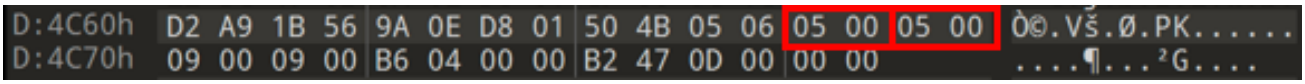
* End of central directory record of the sample APK

number of this disk

Disk numbers are used for split compression.

The disk number is 0 for a single compressed file, and changes by split count and position if it is a split compressed file.

The following image is the End of central directory record of a 6-split compressed file. This file is the last split of the whole group, so the number of this disk value is set to 5. The number of the disk with the start of the central directory is also set to 5, as the central directory also starts from this split.

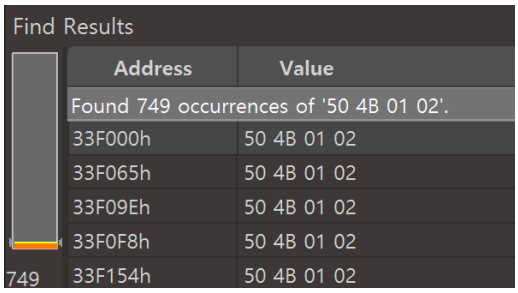


The tampered APK is a single compressed file, but the number of this disk field is set to 0x4012, 0x032E, so we can fix this value to 0.

total number of entries in the central directory

Unlike normal APK files, the malicious APK's 'total number of entries in the central directory on this disk' field has a value of 0x7FE1, which is different from the 'total number of entries in the central directory' field's value 0x02ED.

Actually searching the malicious APK using the central directory signature field's magic number 0x02014B50 returns 749(0x02ED) searchs, so we can confirm that this is the actual entry count.



So we can now fix the 'total number of entries in the central directory on this disk' and 'total number of entries in the central directory' field values to the actual entry count confirmed by our search.

YOU HAVEN'T SEEN IT ALL

We had thought all was well after fixing the deformed ZIP file format, but 2 more techniques were observed. One uses the file system, while the other modifies the format of 'androidmanifest.xml'.

FILENAME OVERFLOW

For popular OSs such as Windows, Linux, MacOS, the lengths of file names are not infinite and is limited for backwards compatibility. The malicious APK abuses this by containing multiple 0-byte files under the asset directory, which have file names longer than 255 characters, to induce errors to analytic tools. Since these files have no effect on analysis, we responded by deleting those files.

```
assets/cefwVSae330NVd9xNTGWzhmDp7IDjgwHUV0EKxVKClfSLGomHeqrBTKW2MhZjI6WGC3aoFXp9
7tnbF3TTw6f1ffbkdM8p7A5quliOo9cratFAxVxUzfYM05Bx20k300bGz8RuUfFy4Y0XLLYtpn11NtAz
UWXyYjCar9bAddauLzTcfPWqb2OMjl4fHsK8sgKUwZGMKypmnGpnKyufaGx7G02ff63WDcbFZ0RqjjQO
MBzN6QGj8F8XUOnG8AGPFDTkFVyLRe611TufBpjAzUY9fpVlXlsUObiiSohlr6C4J
...
assets/iLK2WvctHsIf2GPCRKHxRTlyUzzJxmA9ZCIB47QjaLyJZ9s4wYzbLKrnzmiw6E7v3KrA9nIU4
xwgvMJVhGnJXyyokvcSYuoHOdRpS36iqPinnl3XOypArxIElcpNl06Kj0Cr6tcpKethqkb9DpeAeYqrG
mdQNs8ATJexTL2uv01mdAobbrk7WGFvKlWTpxNrh4zSgOp2xUff8dnf9cPviaOneTYIZif8xJfkbH6xb
sJbAzyG8RUmTrjgplNx0SdoPhSy8ceChru0CMXPvsKiy3pgz0504JIupvxtS0I5Msf
```

OVERVIEW OF THE ANDROIDMANIFEST.XML

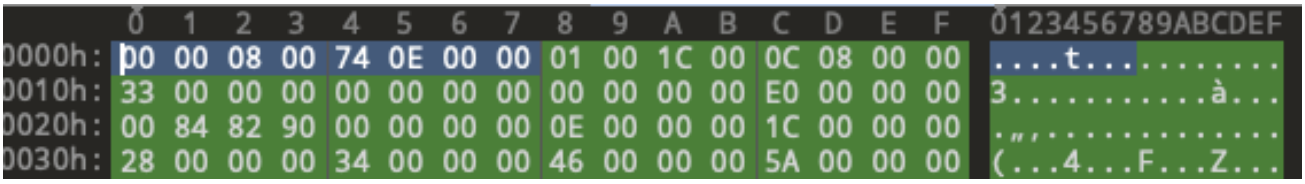
The AndroidManifest.xml is a core file which is essential to constructing an APK file. This file can be used to check required permissions and components such as Activity and Service, which makes it helpful for analyzing malicious apps. Another unknown anti-analysis technique was also applied to this file.

```
❯ java -jar ./AXMLPrinter2.jar ./AndroidManifest.xml
java.io.IOException: Invalid chunk type (6750319).
        at android.content.res.AXmlResourceParser.doNext(AXmlResourceParser.java:812)
        at android.content.res.AXmlResourceParser.next(AXmlResourceParser.java:72)
        at test.AXMLPrinter.main(AXMLPrinter.java:43)
```

For analysis, we referred to many articles on the Internet as there was no official file format document opened to the public.

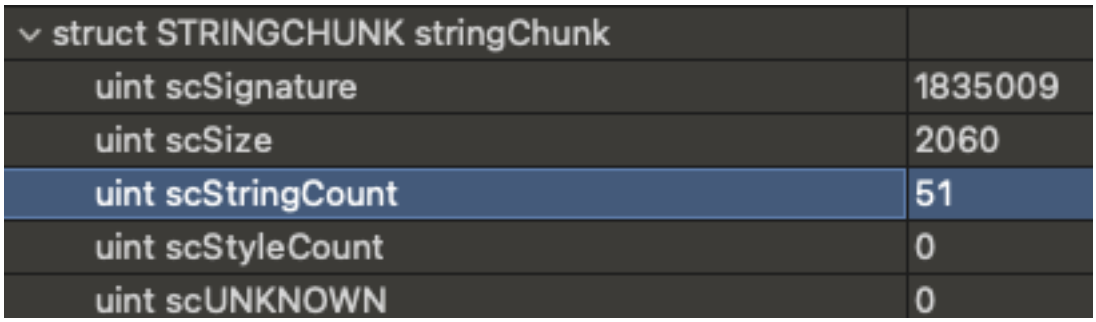
Magic number (AndroidManifest Signiture)

The AndroidManifset.xml has a 4-bytes magic number for its header (0x03000800), while the malicious APK modified this into 0x00000800). File signatures are crucial for file parsing, so we can fix this value back to normal.

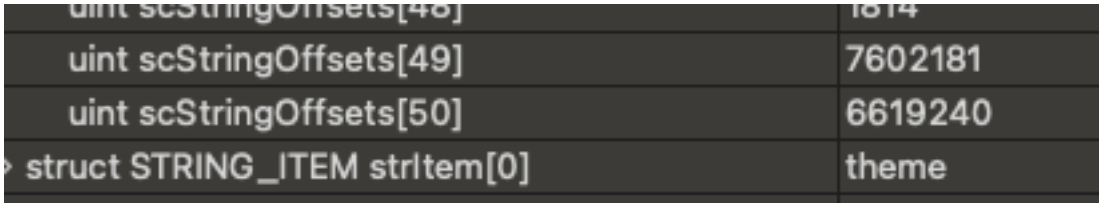


scStringCount

scStringCount is where the size of scStringOffsets array is stored, and the scStringOffsets array is referenced based on this value.



For this sample, scStringOffsets has 51 items in the array, from index 0 to 50. However, the last two indices 49 and 50 should not be referenced, so the value 51 for scStringCount invokes an error. Thus, scStringCount should be 49 instead of 51.



For a more exact procedure to calculate the scStringCount value, we can use the scStringPoolOffset value. As the scStringPoolOffset value is 224 in this sample, by calculating (224-0x1C) / 4, we get 49, which is the actual array size of scStringOffsets.

* 0x1C and 4 are constant numbers to calculate the offset.

CONCLUSIONS

The mobile malware designed for cyber crime targeting unspecified people are now being witnessed using sophisticated techniques to hinder analysis by carefully tampering part of the file format. These apps did not even care for source code obfuscation just a few years ago, but now they are applying multiple techniques and methods to protect themselves from analysis.

This again proves the continuous technical evolution of cyber criminals, which will not stop. The Computer Emergency Analysis Team of Financial Security Institute will keep tracking down these attempts and will respond to cyber crime and threats endangering financial consumers.

IOCS

Deformed APK
Man-Club.apk
(MD5)
1539A1EA9A718C8B7FA2903F02CF9713
(SHA256)
EE857C1D8051255B4F2D56636B0EB35D35C4429E4684041446239CA39650A485

REFERENCES

https://www.mql5.com/en/articles/1971
https://pkware.cachefly.net/webdocs/casestudies/APPNOTE.TXT

MOBILE THREAT HUNTING: MALICIOUS APK DEFORMING ZIP FILE FORMAT FOUND UNDER EXPERIMENT IN THE WILD

Published in October, 2022

Editor-in-chief Chulwoong Kim

Editor board member Financial Security Institute, Computer Emergency Analysis Team (Manager: Woonyoung Chang) **Minchang Jang**, **Heyji Heo**, Kuyju Kim, Jinuk Kim, Minhee lee, Hwang Byung Woo

Published by the Financial Security Institute
16881 132, Daeji-ro, Suji-gu, Yongin-si, Gyeonggi-do, Korea
TEL:+82-2-3495-9000