# MOBILE THREAT HUNTING:
# 분석 방해를 위해 ZIP 파일 포맷을 변조한 악성 APK 연구

# 개 요

스마트폰이 대중화 됨에 따라 모바일 악성코드 또한 우리 삶에 스며들고 있다. 한손에 잡히는 이 작은 기기에 한 사람의 거의 모든 정보가 담겨 있다고해도 과언이 아니다. PC 환경과 마찬가지로 첨단기기를 자유자재로 다루는 일부 사용자를 제외하고 우리는 여전히 악성코드 위협에 노출되어 있다.

금융보안원 침해위협분석팀은 2017년부터 금융회사를 사칭해 악성 앱을 설치하도록 유도한 후 사용자가 설치할 경우 기기의 정보를 탈취하고 소셜 엔지니어링으로 실제 금전적 피해까지 이어지는 보이스피싱이라고 불리는 일종의 금융사기범죄를 추적 및 대응해오고 있다. 이 범죄 과정에는 항상 안드로이드 기반 악성 앱이 수반되는 데, 지난 8월 중순쯤 실제 사례에서 수집한 한 악성 앱에서 ZIP 파일 포맷에 기반한 특이한 분석 방해 기법이 확인되었다.

분석 방해 기법이 적용된 악성 앱은 실물 기기나 에뮬레이터 환경에서는 아무런 에러 없이 설치 및 정상 동작이 가능하지만, 정적 분석 도구로 분석을 시도할 때 에러발생과 함께 분석이 불가능한 상황이 발생했다. 의도적으로 분석을 방해하려는 목적이 명확했기에 근본적인 원인을 확인하는데 주력했다. 이 보고서에서는 근본 원인을 확인 과정에 대해 상세히 설명하고, 발견된 여러 가지 분석 방해 기법의 대응 방법을 다룬다.

## 근본 원인 분석

실물 기기나 에뮬레이터 환경에서는 정상 동작하지만, 정적 분석 도구에서는 에러가 발생했다. 다르게 표현하면 동작에는 전혀 문제가 없는 파일이지만 정적 분석도구는 제대로 프로세싱하지 못한것이기 때문에 파일 처리 과정에서 문제가 있을 것으로 추정하고 분석을 시작했다.

## 샘플 APK 압축 풀기

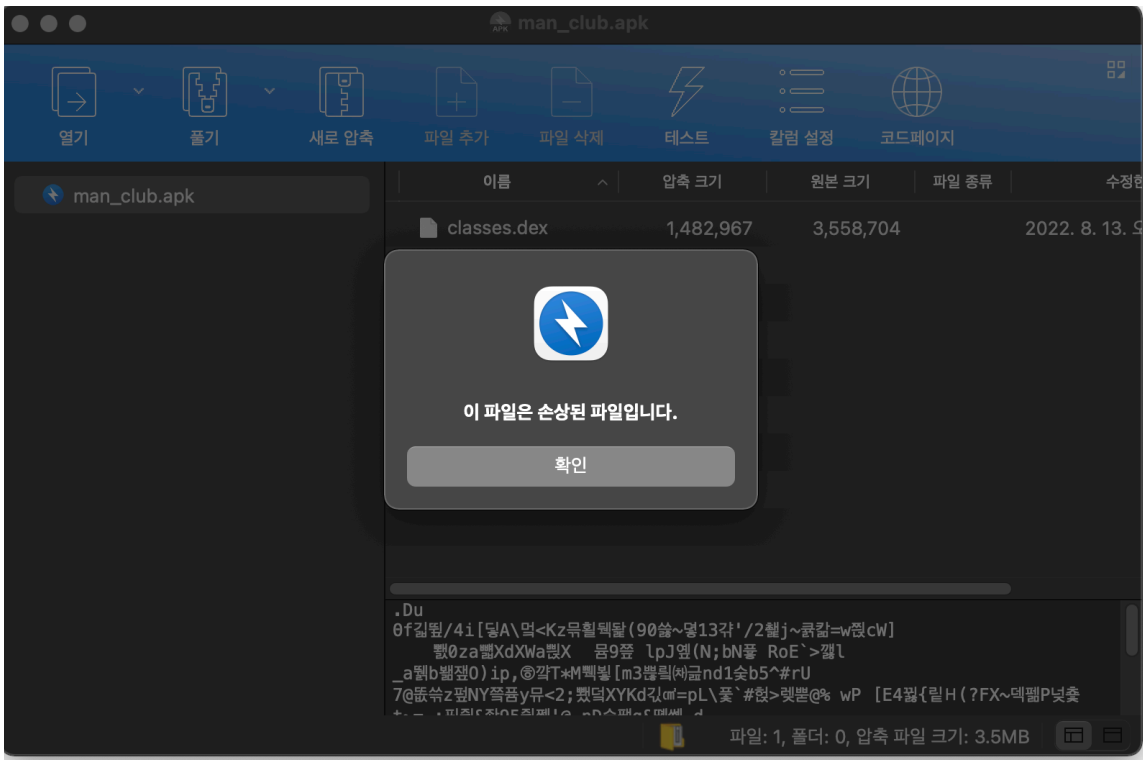
APK 파일은 ZIP 파일 포맷을 기반으로 하여 압축 해제가 가능하다. 그러나 unzip 명령을 사용해 압축 해제를 시도 했을 때 정상적으로 압축이 해제되지 않고 종료되었다.

```
❯ unzip man_club.apk
Archive:  man_club.apk
.Du
   ��f�y��/�4i[�LA\��E<Kz���ℭ����(9000~�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@@e3@z@jNY@j0@y@@<@ 2@;@J@DXYKd@u@@=pL@\@@`@#@C>▯▯▯▯▯@% @wP @[E@@{▯▯▯▯(@?FX~▯▯▯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@"
                                                  ▯▯▯▯▯▯
```

국내 유명 압축유틸리티를 사용해 압축 해제를 시도해봐도 “이 파일은 손상된 파일입니다.” 메시지와 함께 APK 파일 내부의 구성요소들도 제대로 확인되지 않았다.



잘 알려진 APK 분석 도구인 apktool 을 사용해봐도 에러와 함께 분석 진행이 불가능 했다.

```
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 %
```

## 왜 압축이 제대로 풀리지 않을까?

APK 파일은 ZIP 파일 포맷에 기반하고 있다. ZIP 파일 포맷은 여러 비손실 압축 포맷 중 하나이며, 1bit라도 잘못된 처리를 하게 된다면 원본 데이터에 큰 영향을 줄 수 있으므로 매우 정교한 파일 포맷을 가지고 있을 것이다.

정적 분석 도구는 분석가에게 정확한 데이터를 제공해야하므로 일종의 정교한 파서(paser)라고 볼 수 있다. 따라서 분석 도구가 APK 파일 내부의 데이터를 온전히 추출해야되는 과정에서 알 수 없는 이유로 인해 데이터 처리를 못했기 때문에 문제가 발생한것이라 가정하고 파일 포맷에 기반한 분석을 진행했다.

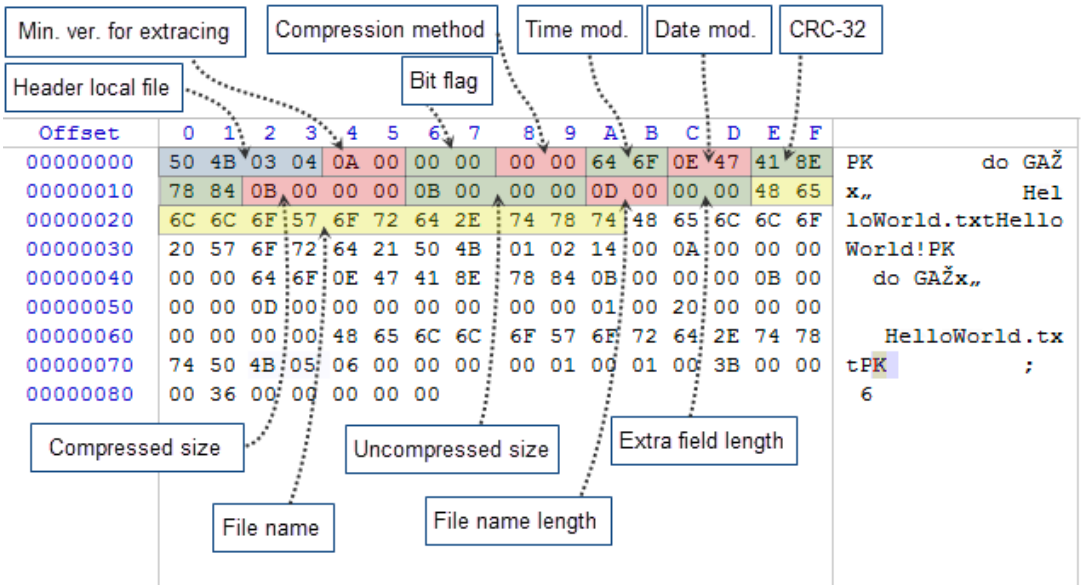
## ZIP 파일 포맷

APK 파일은 ZIP 파일 포맷에 기반하기에 여러 분석 도구는 ZIP 파일을 처리하는 기능을 직접 구현하거나 외부 라이브러리 등을 사용했을 것이다. ZIP 파일 구조는 다음과 같이 다양한 영역으로 나눠져 있다. 악성 앱의 정상 동작을 위해 원본 데이터는 변조되어서는 안될 것이기에 file data 영역을 제외한 영역들을 확인해보기로 했다.

```
[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]
```

## LOCAL FILE HEADER 개요

local file header는 다음과 같이 세부 필드로 구성되어 있다. 이 필드의 값 중 일부는 변경할 경우 ZIP 파일 고유 성격 자체를 상실하게 되므로, 수정되더라도 고유의 기능을 유지할 수 있는 필드의 값을 임의 조작했을 것으로 볼 수 있다.



이를 확인하기 위해 분석 도구로 정상 분석이 가능한 APK와 비교해 보기로 했다.

| 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 |

정상 APK와 비정상 APK의 local file header를 비교해보면 이미지 상에서 frVersion, frFlags, frCompression 필드의 값이 다른것을 볼 수 있다. 각 필드가 어떤 범위의 값을 몇바이트 단위로 사용하는지에 대해서는 공식 문서[1]에서 자세하게 확인할 수 있다.

| 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 |

* 물론 파일 시간 정보도 다르지만 일반적으로 파일의 시간이 동작 과정에 관여하는 부분은 미미하므로 분석에는 제외하였다.

**version needed to extract (frVersion)**
공식 문서에 따르면 이 필드는 압축 해제시 필요한 버전을 1.0 ~ 6.3 까지의 값으로 정의 할 수 있는데, 비정상 APK에는 엉뚱한 값인 28963이 할당 되어 있다.

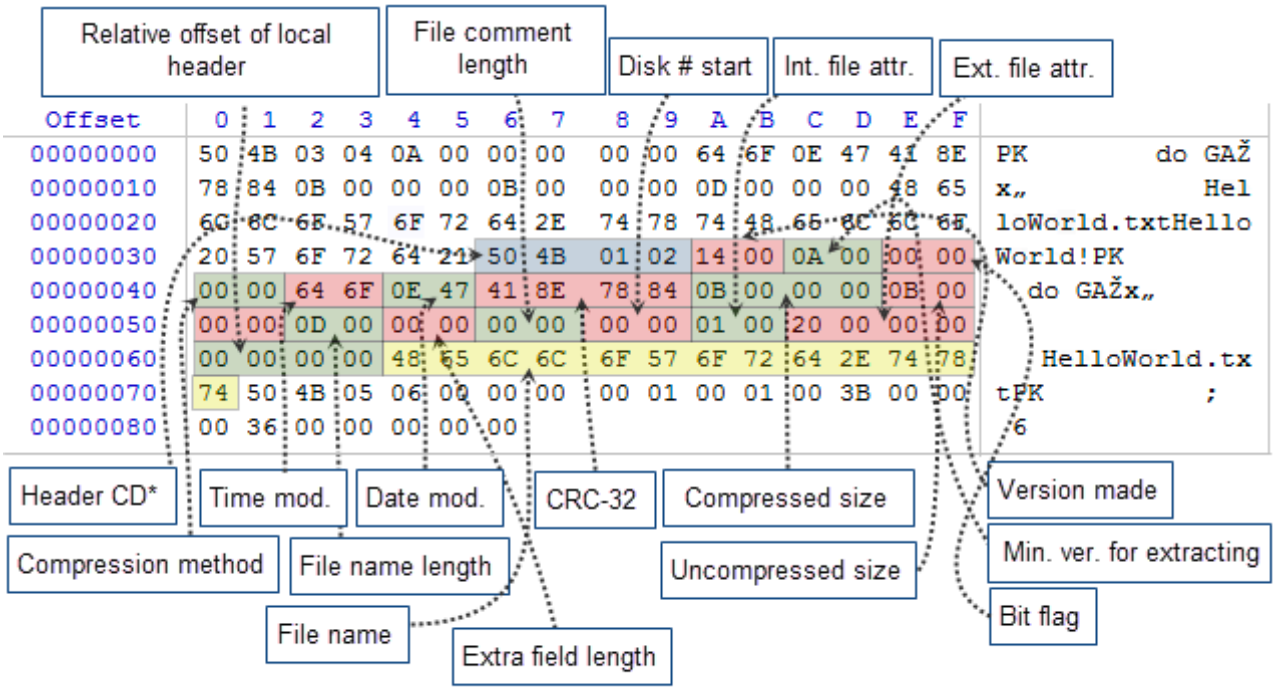
**general purpose bit flag (frFlags)**
특정 상태를 확인 할 수 있는 Flags를 정의할 수 있는데, 비정상 APK에는 공식 문서에서 허용하지 않는 범위에 해당하는 값인 60495가 할당 되어 있다.

**compression method (frCompression)**
compression method 유형이 정의되는 부분인데, 비정상 APK에는 허용되는 범위를 벗어난 -18527이 할당 되어 있다.

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

## CENTRAL DIRECTORY HEADER 개요

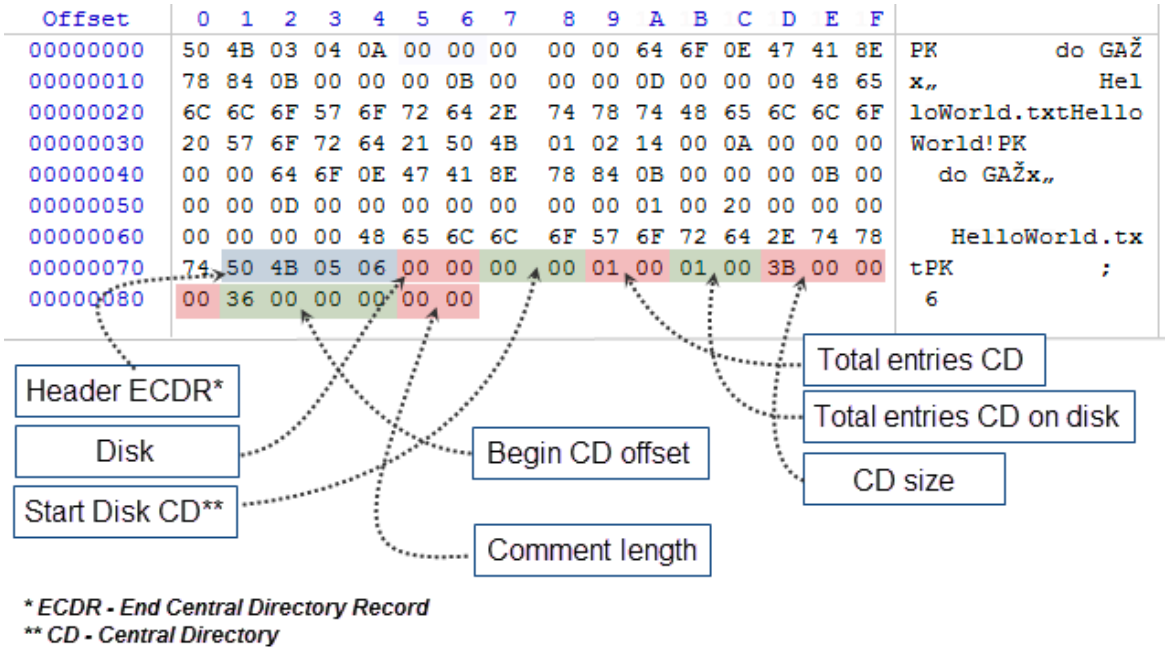
앞서 살펴본 local file header 에 존재하는 다양한 필드가 central directory header에도 동일하게 존재한다.




비정상 APK의 central directory header에 있는 필드의 값들은 local file header에서 확인했던것과 달리 모두 정상 범위에 속하는 값들로 확인된다. 그리고 central directory header에는 local file header와 동일한 필드가 여러개 존재하는데, 정상 APK의 경우 동일 필드의 값은 두개의 header 모두 동일하다. 따라서, 비정상 APK의 central directory header에 있는 필드의 값으로 local file header의 필드를 수정하는 형태로 대응이 가능할 것으로 보인다.

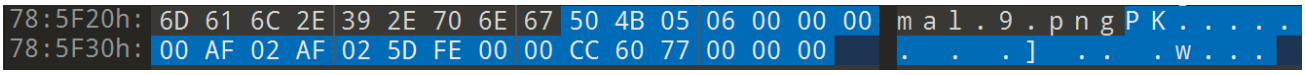
## THE END OF CENTRAL DIRECTORY RECORD 개요

the End of central directory record는 아래와 같은 세부 필드로 구성된다.

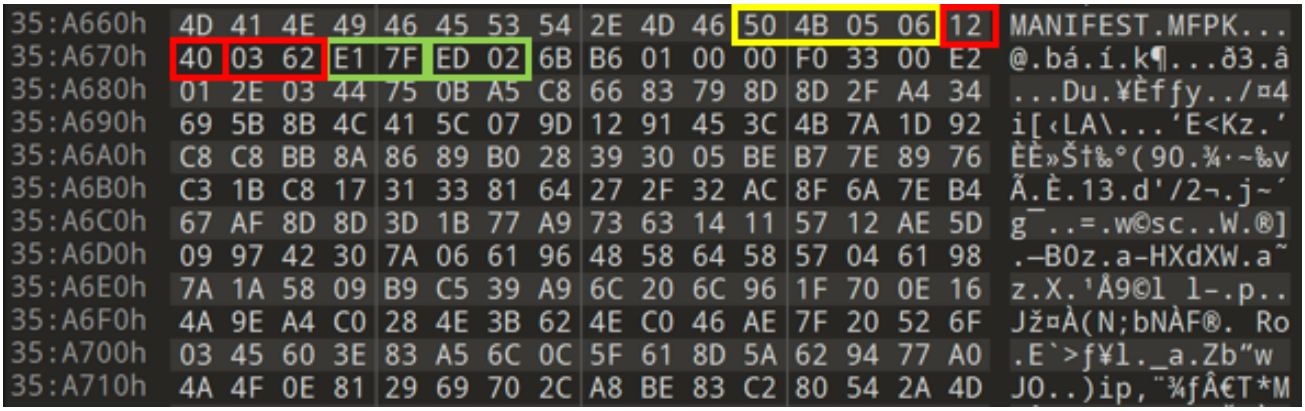



the End of central directory는 zip파일의 마지막에 위치하는 하나의 영역으로 zip 파일을 열었을 때 가장 먼저 참조하게된다. 이 영역은 central directory의 개수, 크기, 시작 오프셋 등의 정보를 포함하고 있다.

정상 APK와 비정상 APK를 비교한 결과 디스크 번호 필드의 값이 변조된 것으로 보이고, 특히 central directory 개수가 서로 다른것을 확인 할 수 있었다.




* 정상 APK의 End of central directory record

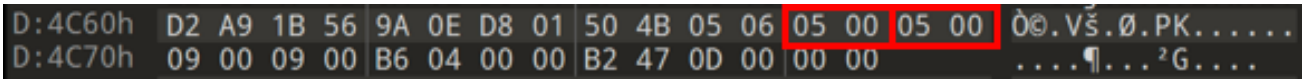



* 비정상 APK의 End of central directory record

**number of this disk**

디스크 번호는 분할 압축 시 활용된다. 만약 단일 압축 파일인 경우 디스크 번호는 모두 0이며, 분할 압축 파일인 경우 분할된 개수와 위치에 따라 번호가 달라진다.

아래 이미지는 6개로 분할된 압축파일의 end of central directory이다. 이 영역은 zip 파일의 마지막에 위치하므로 number of this disk 필드 값은 마지막 디스크 번호인 5가 설정된다. 또한 해당 파일의 경우 central directory가 end of central directory와 동일한 분할에서 시작하여 number of the disk with the start of the central directory 필드 값이 5로 설정되었다.

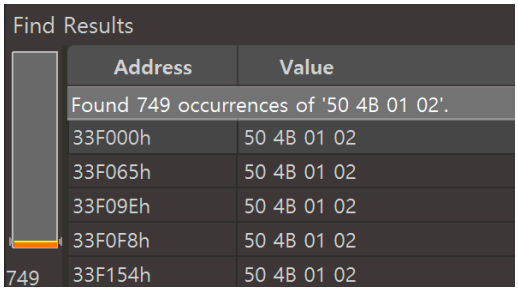



변조된 APK는 단일 압축파일임에도 디스크 번호가 0x4012, 0x6203으로 설정되어 있으므로, 해당 필드의 값을 0으로 수정해주면 된다.

**total number of entries in the central directory**

정상 APK와 달리 변조된 APK에서는 total number of entries in the central directory on this disk 필드의 값이 0x7FE1로 total number of entries in the central directory 필드의 값인 0x02ED와 같지 않은 것으로 확인된다.

실제 비정상 APK에서 central directory signature 인 0x02014B50을 검색하면 총 749(0x2ED)개가 검색되며, 이를 통해 해당 개수 만큼의 central directory가 존재함을 알 수 있다.




따라서 total number of entries in the central directory on this disk 필드의 값과 total number of entries in the central directory 필드의 값을 앞서 검색을 통해 확인한 실제 개수로 수정해주면 된다.

## 추가 기법 확인

앞서 확인한 비정상 ZIP 파일 포맷으로 모든걸 해결한 줄 알았지만 두가지 분석 방해 기법이 추가로 확인되었다. 하나는 파일 시스템의 특징을 이용한 방법이고, 나머지 하나는 AndroidManifest.xml 의 파일 포맷을 변조한 방법이다.

## 파일명 오버플로우

우리가 일반적으로 사용하는 OS인 윈도우, 리눅스, 맥 등에서 파일명의 길이는 무한하지 않으며 하위 호환성을 위해 파일시스템에서 제한하고 있다. 이 점을 악용해 비정상 APK는 assets 경로 하위에 255자리 이상의 파일명을 가진 0바이트 크기의 파일을 여러개 구성하는 방법으로 분석 도구의 오류를 유도한다. 해당 파일들은 분석에 영향을 미치지 않으므로 모두 삭제하는 방법으로 대응했다.

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

## ANDROIDMANIFEST.XML 개요

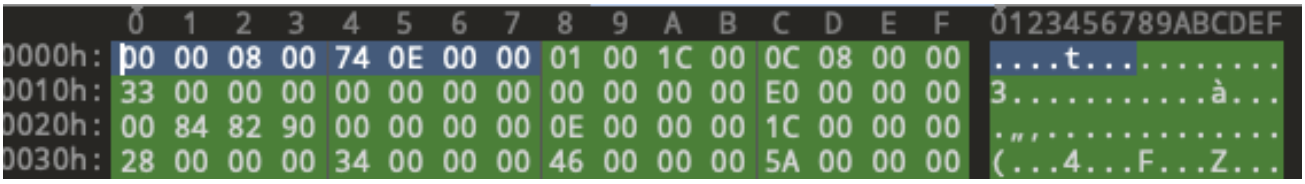
AndroidManifest.xml은 APK 파일 구성요소에서 반드시 존재해야되는 핵심적인 파일이다. 이 파일에는 앱 동작에 필요한 권한들과 Activity, Service와 같은 구성요소(components) 등을 확인할 수 있어 악성 앱 분석에 있어 많은 도움이 되는 파일이다. 이 파일에도 알 수 없는 분석 방해 기법이 적용되어 있었다.

```
❯ 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)
```

AndroidManifest.xml 파일의 경우 공식적으로 공개된 파일 포맷이 없었기 때문에 인터넷에 공개되어 있는 여러 자료를 참고하여 분석을 했다.

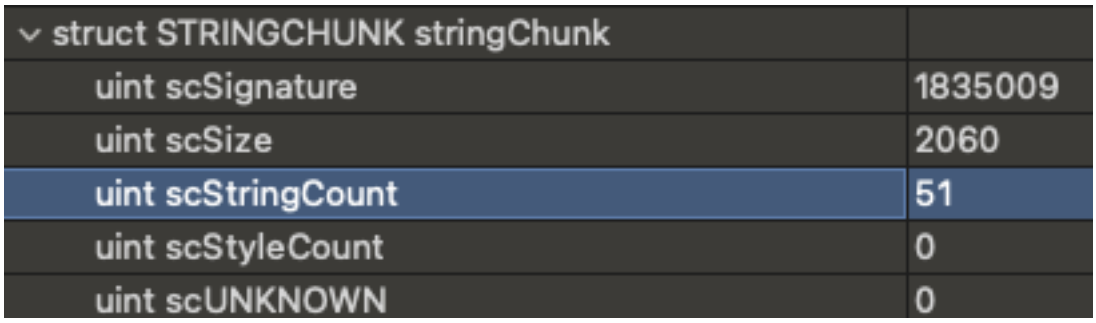
**Magic number (AndroidManifest Signiture)**

AndroidManifext.xml의 경우 헤더 시그니처로 매직넘버라고 불리는 4 바이트 크기의 데이터(03 00 08 00)가 있는데, 비정상 APK의 경우 이 매직넘버가 비정상적인 값(00 00 08 00)으로 확인되었다. 시그니처는 파일을 파싱 할 때 중요한 기준이 되므로 비정상적인 값을 올바르게 수정해주면 된다.

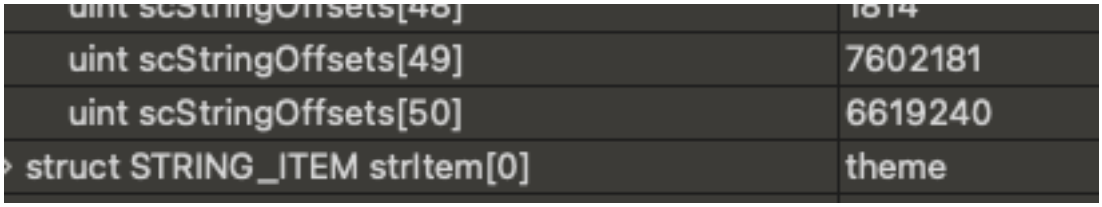



**scStringCount**

scStringCount는 scStringOffests 배열의 사이즈가 저장되는 공간이며, 해당 값을 기준으로 4바이트 오프셋을 가지는 scStringOffests 배열을 참조한다.




scStringOffsets 필드는 0번 인덱스부터 50번 인덱스까지 총 51개의 배열이 확인된다. 하지만 마지막 49번과 50번 인덱스는 참조되어서는 안되는 값이기 때문에 에러를 유발하게 된다. 즉 scStringCount의 값은 51이 아니라 49가 되도록 수정해야한다.




scStringCount 값을 계산하기 위한 보다 정확한 방법으로는 scStringPoolOffset 값을 이용할 수 있다. scStringPoolOffset의 값이 224일 경우 (224-0x1C) / 4를 통해 scStringOffsets 배열의 실제 크기인 49를 얻을 수 있다.

* 0x1C와 4는 오프셋 계산을 위한 상수 값이다.

## 결 론

불특정 다수를 대상으로 하는 사이버 범죄에 사용되는 악성코드가 분석을 방해하기 위한 목적으로 파일 포맷의 일부를 의도적으로 정교하게 변조한 기법이 적용된 것을 분명히 확인했다. 이런 유형의 악성 앱은 불과 몇년전까지만 해도 소스코드 난독화 조차 적용하지 않았지만 최근에는 자신들의 앱을 보호하기위해 여러가지 수단과 방법을 적용하고 있는 것으로 보인다.

이는 사이버 범죄자들이 기술적으로 계속 발전하고 있음을 다시 한번 증명하는것이며, 앞으로도 이런 시도는 계속 될 것이다. 금융보안원 침해위협분석팀은 악성 앱을 지속적으로 추적 관찰하여 금융소비자를 위협하는 사이버 범죄와 위협에 대응할 것이다.

## 침해지표

**샘플 APK**

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

## 참 조

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

# MOBILE THREAT HUNTING:
# 분석 방해를 위해 ZIP 파일 포맷을 변조한 악성 APK 연구

**발행일** 2022년 10월

**발행인** 김철웅

**작성자** 금융보안원 침해대응부 침해위협분석팀 (팀장 장운영)
**장민창**, **허혜지**(이후 가나다순), 김귀주, 김진욱, 이민희, 황병우

**발행처** 금융보안원
경기도 용인시 수지구 대지로 132
TEL 02-3495-9000