Android Malware Analysis : Dissecting Hydra Dropper

July 18, 2019 Ahmet Bilal Can Android

Hydra is another android bankbot variant. It uses overlay to steal information like Anubis . Its name comes from command and control panel. Through July 2018 to March 2019 there was atleast 8-10 sample on Google Play Store. Distribution of malware is similar to Anubis cases. Dropper apps are uploaded to Play Store. But unlike Anubis, Dropper apps extract dex file from png file with *kinda* stenography and downloads malicious app from command and control server with dropped dex. You can find the sample that I will go through in this post here : Dropper

ToC:

- Bypass checks that on the java side
- GDB Debug
- Ghidra shenanigans
- Understanding creation of the dex file
- Bonus

First of all, if the dropper app likes the environment it runs, it will load the dex file and connect to the command and control server. There are multiple checks on java and native side. We will debug the native side with gdb and use ghidra to help us to find checks and important functions.

Time Check

When we open the first app with jadx we can see time check in class `com.taxationtex.giristexation.qes.Hdvhepuwy.`

```java
public static boolean j() {
        return new Date().getTime() >= 1553655180000L && new Date().getTime()
<= 1554519180000L;
}
```

This function called in another class : com.taxationtex.giristexation.qes.Sctdsqres

```java
class Sctdsqres {
    private static boolean L = false;
    private static native  void fyndmmn(Object obj);
    Sctdsqres() {
    }
    static void j() {
        if (Hdvhepuwy.j()) {
            H();
        }
    }
    static void H() {
        if (!L) {
            System.loadLibrary("hoter");
            L = true;
        }
        fyndmmn(Hdvhepuwy.j());
    }
}
```

First, it checks the time and if the condition holds, the app will load the native library and call `fyndmmn(Hdvhepuwy.j());` which is native function. We need to bypass this check so app will always load the library.

I used `apktool` to disassemble apk to smali and changed `j()` to always return true.

- apktool d com.taxationtex.giristexation.apk
- cd com.taxationtex.giristexation/smali/com/taxationtext/giristexation/qes

- edit j()Z in Hdvhepeuwy.smali

```
.method public static j()Z
    .locals 1
    const/4 v0, 0x1
    return v0
.end method
```

rebuild apk with `apktool b com.taxationtex.giristexation -o hydra_time.apk` and sign it.

Now time control will always return true and after loading native library and `fyndmmn` native function is called. Even with this still app doesn't load dex file.

GDB Debug

Here is a great post explaining how to setup gdb to debug native libraries. Steps:

- Download android sdk with ndk
- adb push ~android-ndk-r20/prebuilt/android-TARGET-ARCH/gdbserver/gdbserver /data/local/tmp
- adb shell "chmod 777 /data/local/tmp/gdbserver"
- adb shell "ls -l /data/local/tmp/gdbserver"
- get process id, ps -A | grep com.tax
- /data/local/tmp/gdbserver :1337 –attach $pid
- adb forward tcp:1337 tcp:1337
- gdb
- target remote :1337
- b Java_com_tax\TAB

There is a small problem here. App will load the library and call the native function and exit. The app needs to wait for gdb connection. My first thought was putting sleep and then connect with gdb.

- apktool d hydra_time.apk
- vim hydra_time/com.taxationtex.giristexation/smali/com/taxationtex/giristexation/qes/Sctdsqres.smali

after following block:

```
.line 43
:cond_0
```

Add

```
const-wide/32 v0, 0xea60
invoke-static {v0, v1}, Landroid/os/SystemClock;->sleep(J)V
```

and since `locals` variable is 1 and we use an extra v1 variable, increment it to 2

```
.method static H()V
    .locals 2
```

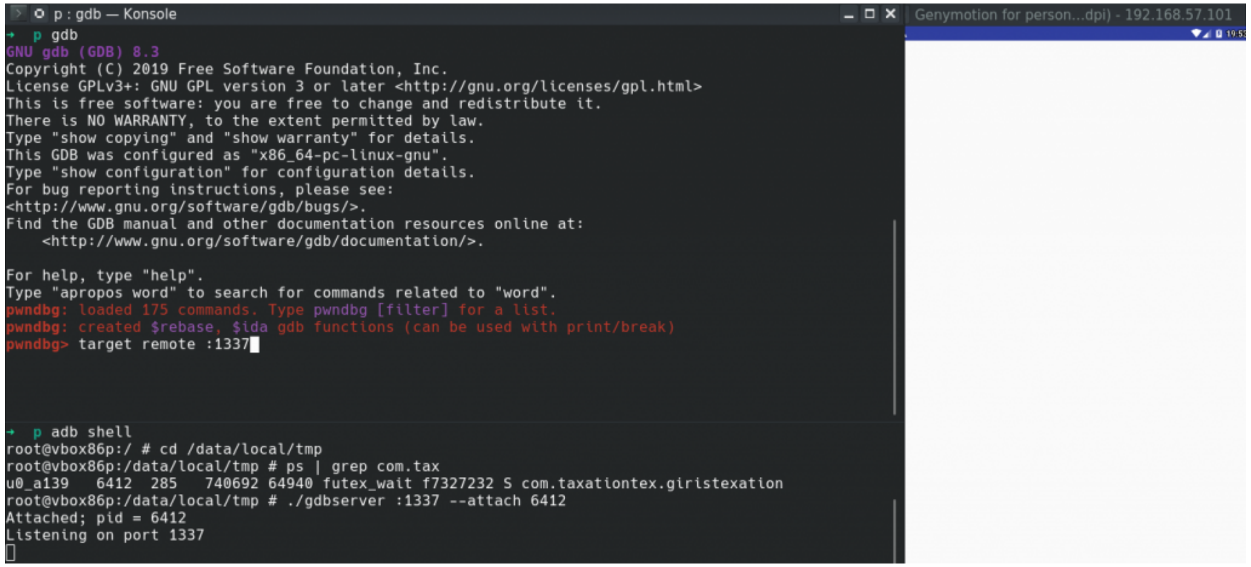
Again sign and install the app. If all goes well the app will wait 60 seconds in a white screen. Now we can connect with gdb.

```
ps | grep com.tax
/data/local/tmp/gdbserver :1337 --attach $pid
```

I use pwndbg for better gdb experience, you can try peda or whatever you want.

- adb forward tcp:1337 tcp:1337

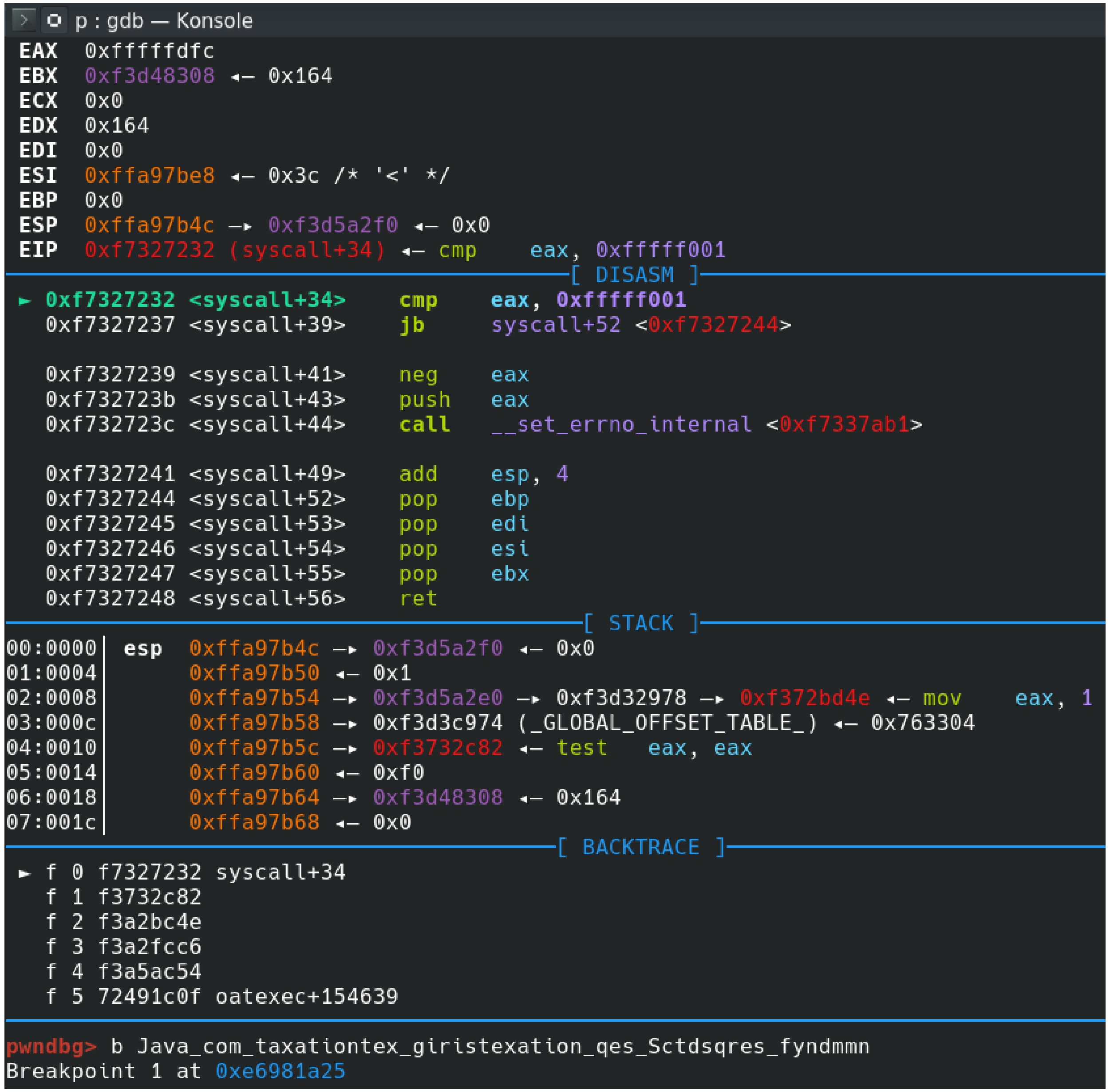
- gdb
- target remote :1337



```
p : gdb — Konsole                                    Genymotion for person...dpi) - 192.168.57.101
→  p gdb
GNU gdb (GDB) 8.3
Copyright (C) 2019 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-pc-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word".
pwndbg: loaded 175 commands. Type pwndbg [filter] for a list.
pwndbg: created $rebase, $ida gdb functions (can be used with print/break)
pwndbg> target remote :1337


→  p adb shell
root@vbox86p:/ # cd /data/local/tmp
root@vbox86p:/data/local/tmp # ps | grep com.tax
u0_a139   6412  285   740692 64940 futex_wait f7327232 S com.taxationtex.giristexation
root@vbox86p:/data/local/tmp # ./gdbserver :1337 --attach 6412
Attached; pid = 6412
Listening on port 1337
```

debug session

It takes some time to load all libraries. Put breakpoint to native function `fymdmmn`

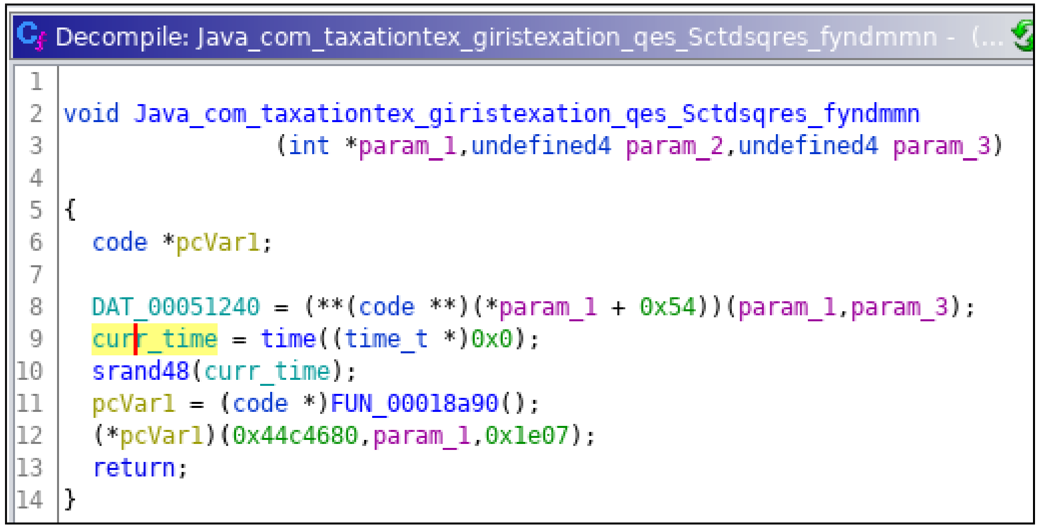


```
EAX  0xffffdfc
EBX  0xf3d48308 ◂— 0x164
ECX  0x0
EDX  0x164
EDI  0x0
ESI  0xffa97be8 ◂— 0x3c /* '<' */
EBP  0x0
ESP  0xffa97b4c —▸ 0xf3d5a2f0 ◂— 0x0
EIP  0xf7327232 (syscall+34) ◂— cmp    eax, 0xfffff001
───────────────────────────────────────[ DISASM ]───────
 ► 0xf7327232 <syscall+34>    cmp    eax, 0xfffff001
   0xf7327237 <syscall+39>    jb     syscall+52 <0xf7327244>

   0xf7327239 <syscall+41>    neg    eax
   0xf732723b <syscall+43>    push   eax
   0xf732723c <syscall+44>    call   __set_errno_internal <0xf7337ab1>

   0xf7327241 <syscall+49>    add    esp, 4
   0xf7327244 <syscall+52>    pop    ebp
   0xf7327245 <syscall+53>    pop    edi
   0xf7327246 <syscall+54>    pop    esi
   0xf7327247 <syscall+55>    pop    ebx
   0xf7327248 <syscall+56>    ret
────────────────────────────────────────[ STACK ]───────
00:0000│ esp  0xffa97b4c —▸ 0xf3d5a2f0 ◂— 0x0
01:0004│      0xffa97b50 ◂— 0x1
02:0008│      0xffa97b54 —▸ 0xf3d5a2e0 —▸ 0xf3d32978 —▸ 0xf372bd4e ◂— mov    eax, 1
03:000c│      0xffa97b58 —▸ 0xf3d3c974 (_GLOBAL_OFFSET_TABLE_) ◂— 0x763304
04:0010│      0xffa97b5c —▸ 0xf3732c82 ◂— test   eax, eax
05:0014│      0xffa97b60 ◂— 0xf0
06:0018│      0xffa97b64 —▸ 0xf3d48308 ◂— 0x164
07:001c│      0xffa97b68 ◂— 0x0
──────────────────────────────────────[ BACKTRACE ]─────
 ► f 0 f7327232 syscall+34
   f 1 f3732c82
   f 2 f3a2bc4e
   f 3 f3a2fcc6
   f 4 f3a5ac54
   f 5 72491c0f oatexec+154639
────────────────────────────────────────────────────────
pwndbg> b Java_com_taxationtex_giristexation_qes_Sctdsqres_fyndmmn
Breakpoint 1 at 0xe6981a25
```

set breakpoint

If you want to sync gdb and ghidra addresses, type vmmap at gdb and look for first entry of `libhoter.so` .
`0xe73be000 0xe73fc000 r-xp 3e000 0`
`/data/app/com.taxationtex.giristexation-1/lib/x86/libhoter.so`
So `0xe73be000` is my base address.
Go to `Window` -> `Memory Map` and press `Home` icon on the upper right. Put your base address and rebase the binary.

Look at the entry of native function in ghdira:


Decompile: Java_com_taxationtex_giristexation_qes_Sctdsqres_fyndmmn - (...

```

void Java_com_taxationtex_giristexation_qes_Sctdsqres_fyndmmn
               (int *param_1,undefined4 param_2,undefined4 param_3)

{
  code *pcVar1;

  DAT_00051240 = (**(code **)(*param_1 + 0x54))(param_1,param_3);
  curr_time = time((time_t *)0x0);
  srand48(curr_time);
  pcVar1 = (code *)FUN_00018a90();
  (*pcVar1)(0x44c4680,param_1,0x1e07);
  return;
}
```

fyndmmn function

Why call the time function ? Again time check ? Rename return value of time function (curr_time) and press `ctrl+shift+f` from assembly view and go to location that context is `READ`

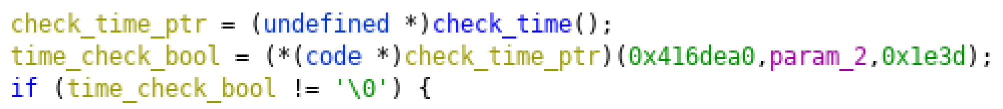
```
return (uint)(curr_time + 0xa3651a74U < 0xd2f00)
```

So we were right, again time check. Rename the current function to `check_time`. Calculate the epoch time:

```
>>> 0xffffffff-0xa3651a74+0xd2f00
>>> 1554519179
>>> (1554519179+ 0xa3651a74) & 0xffffffff < 0xd2f00
>>> True
```

convert epoch to time : Saturday, April 6, 2019 2:52:59 AM

Yep this was the time that app was on play store. Check how this boolean is used. Look for xrefs of `check_time` function.



```
check_time_ptr = (undefined *)check_time();
time_check_bool = (*(code *)check_time_ptr)(0x416dea0,param_2,0x1e3d);
if (time_check_bool != '\0') {
```

Yep, as we think it will exit if time doesn't hold.

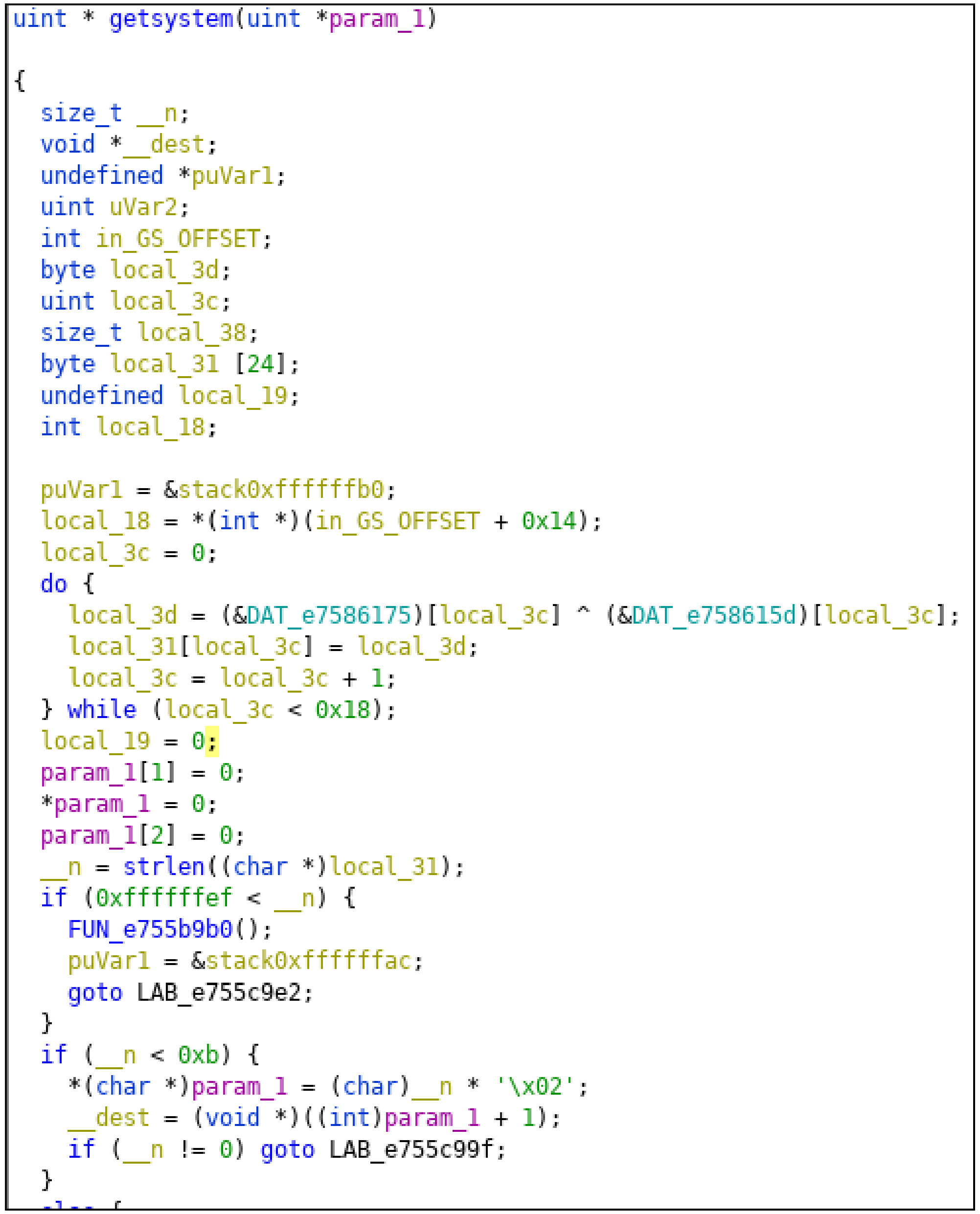
First breakpoint/binary patch point is here. Or we can change emulator/phone's time to April 5 2019.

`b *(base + 0x8ba8)`

But bypassing time check is not enough.

Ghidra Shenanigans

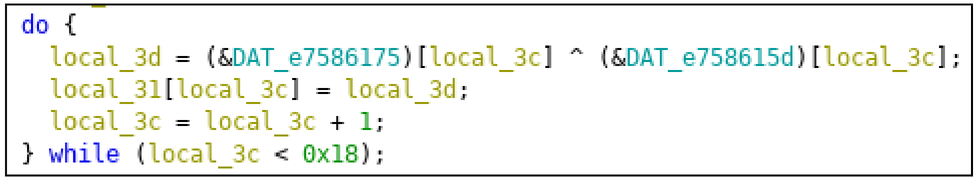
Now diving into binary file you will find multiple functions like this :



```
uint * getsystem(uint *param_1)

{
  size_t __n;
  void *__dest;
  undefined *puVar1;
  uint uVar2;
  int in_GS_OFFSET;
  byte local_3d;
  uint local_3c;
  size_t local_38;
  byte local_31 [24];
  undefined local_19;
  int local_18;

  puVar1 = &stack0xffffffb0;
  local_18 = *(int *)(in_GS_OFFSET + 0x14);
  local_3c = 0;
  do {
    local_3d = (&DAT_e7586175)[local_3c] ^ (&DAT_e758615d)[local_3c];
    local_31[local_3c] = local_3d;
    local_3c = local_3c + 1;
  } while (local_3c < 0x18);
  local_19 = 0;
  param_1[1] = 0;
  *param_1 = 0;
  param_1[2] = 0;
  __n = strlen((char *)local_31);
  if (0xffffffef < __n) {
    FUN_e755b9b0();
    puVar1 = &stack0xffffffac;
    goto LAB_e755c9e2;
  }
  if (__n < 0xb) {
    *(char *)param_1 = (char)__n * '\x02';
    __dest = (void *)((int)param_1 + 1);
    if (__n != 0) goto LAB_e755c99f;
  }
```

decryption blocks

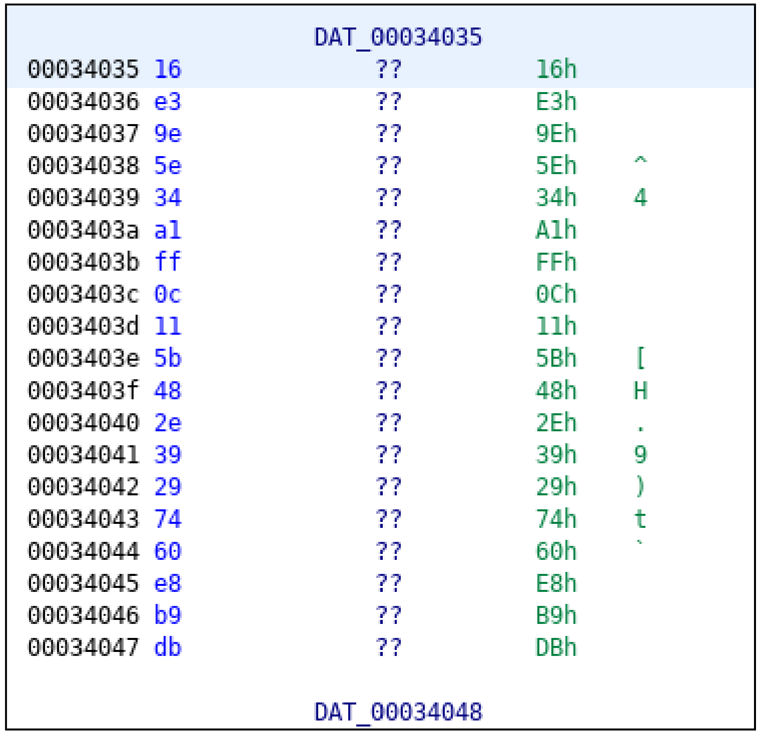
If you look at while loop.



```
do {
  local_3d = (&DAT_e7586175)[local_3c] ^ (&DAT_e758615d)[local_3c];
  local_31[local_3c] = local_3d;
  local_3c = local_3c + 1;
} while (local_3c < 0x18);
```

xor while loop

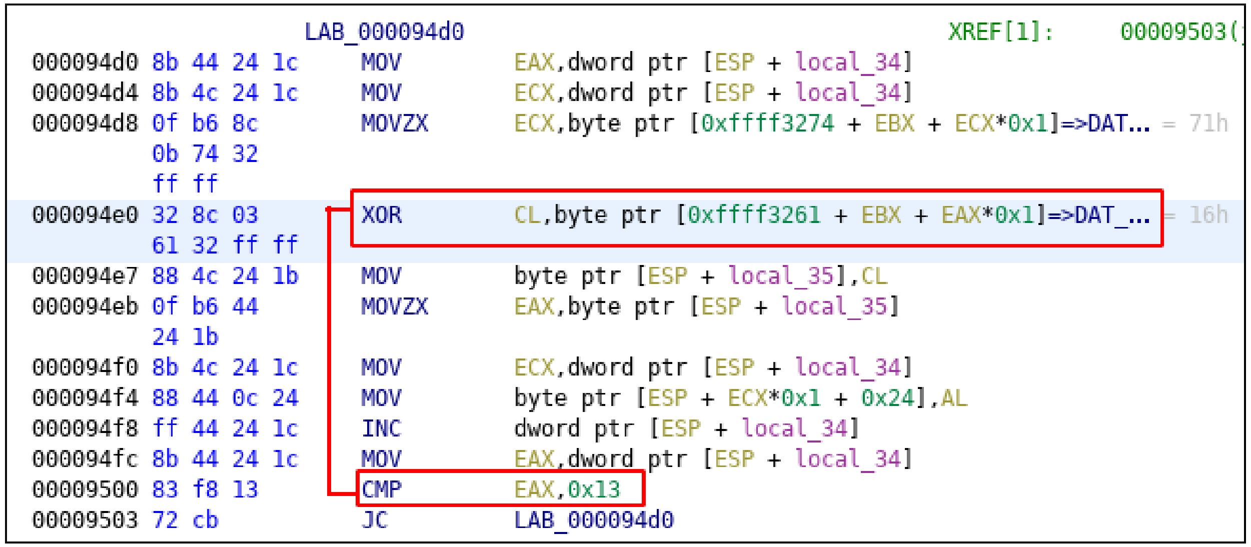
2 blocks of data are XORed. (Length 0x18) We can put breakpoint after do while but it will not be efficient solution. Let’s think a programmatic way to find decrypted strings.
These xor blocks are next to each other. If we can get length of blocks we can easily get decrypted string. Then find the function that use these xor blocks and rename it. Afterwards we can jump `2*length` and get next xor blocks. Repeat.
Starting xor block is at `0x34035`.
Get xrefs of block:



```
                        DAT_00034035
00034035 16             ??          16h
00034036 e3             ??          E3h
00034037 9e             ??          9Eh
00034038 5e             ??          5Eh    ^
00034039 34             ??          34h    4
0003403a a1             ??          A1h
0003403b ff             ??          FFh
0003403c 0c             ??          0Ch
0003403d 11             ??          11h
0003403e 5b             ??          5Bh    [
0003403f 48             ??          48h    H
00034040 2e             ??          2Eh    .
00034041 39             ??          39h    9
00034042 29             ??          29h    )
00034043 74             ??          74h    t
00034044 60             ??          60h    `
00034045 e8             ??          E8h
00034046 b9             ??          B9h
00034047 db             ??          DBh

                        DAT_00034048
```

xor block

go to function,



```
                      LAB_000094d0                              XREF[1]:     00009503(
000094d0 8b 44 24 1c     MOV        EAX,dword ptr [ESP + local_34]
000094d4 8b 4c 24 1c     MOV        ECX,dword ptr [ESP + local_34]
000094d8 0f b6 8c        MOVZX      ECX,byte ptr [0xffff3274 + EBX + ECX*0x1]=>DAT...  = 71h
         0b 74 32
         ff ff
000094e0 32 8c 03        XOR        CL,byte ptr [0xffff3261 + EBX + EAX*0x1]=>DAT_...  = 16h
         61 32 ff ff
000094e7 88 4c 24 1b     MOV        byte ptr [ESP + local_35],CL
000094eb 0f b6 44        MOVZX      EAX,byte ptr [ESP + local_35]
         24 1b
000094f0 8b 4c 24 1c     MOV        ECX,dword ptr [ESP + local_34]
000094f4 88 44 0c 24     MOV        byte ptr [ESP + ECX*0x1 + 0x24],AL
000094f8 ff 44 24 1c     INC        dword ptr [ESP + local_34]
000094fc 8b 44 24 1c     MOV        EAX,dword ptr [ESP + local_34]
00009500 83 f8 13        CMP        EAX,0x13
00009503 72 cb           JC         LAB_000094d0
```

get cmp value

get size from CMP instruction, since we know the address of first xor block, add size to first address and get the address of second xor block. XOR the blocks and rename the calling function.

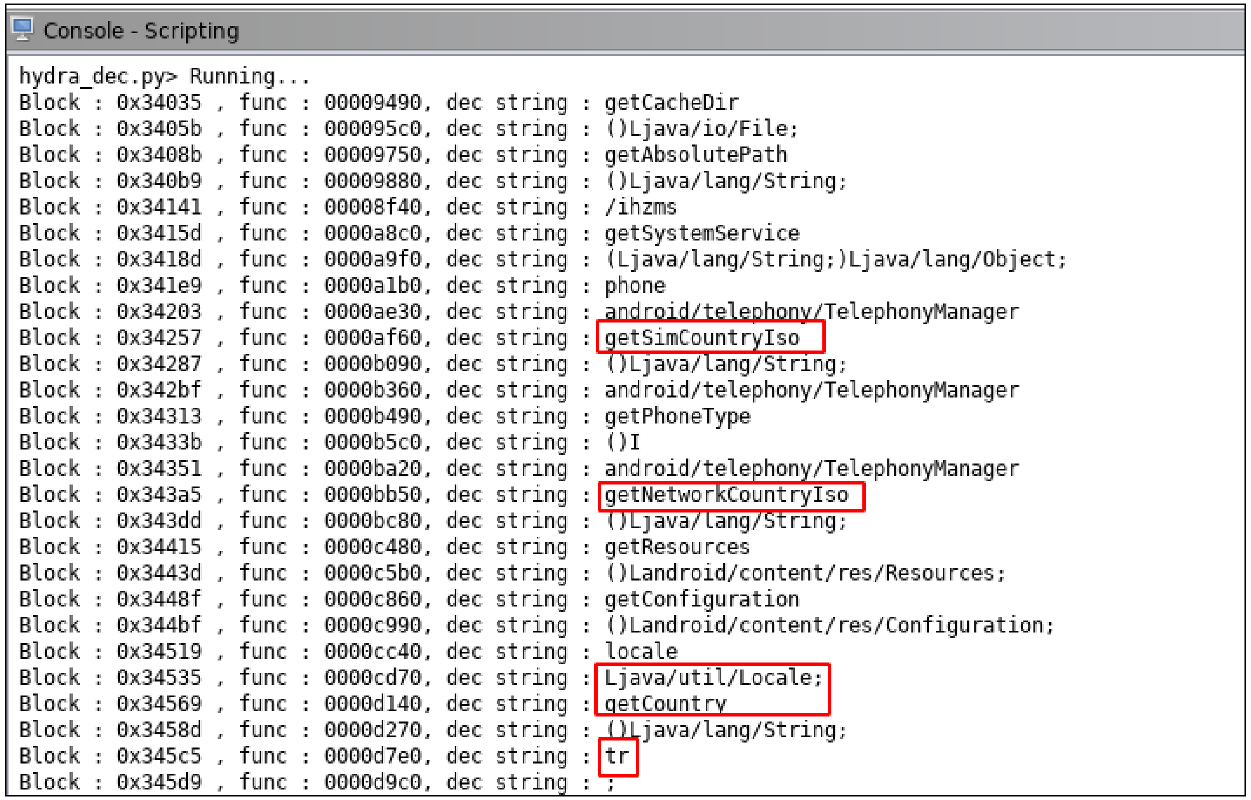
Ghidra : go to `Window` -> `Script Manager` -> `Create New Script` -> `Python`.
Set name for script and let's write our ghidra script.

```python
import ghidra.app.script.GhidraScript
import exceptions
from ghidra.program.model.address import AddressOutOfBoundsException
from ghidra.program.model.symbol import SourceType

def xor_block(addr,size):

  ## get byte list
  first_block = getBytes(toAddr(addr),size).tolist()
  second_block = getBytes(toAddr(addr+size),size).tolist()

  a = ""
  ## decrypt the block
  for i in range(len(first_block)):
    a += chr(first_block[i]^second_block[i])
        ## each string have trash value at the end, delete it
  trash = len("someval")
  return a[:-trash]

def block(addr):
   ## block that related to creation of dex file. pass itt
  if addr == 0x34755:
    return 0x0003494f
  ## get xrefs
  xrefs = getReferencesTo(toAddr(addr))
  if len(xrefs) ==0:
    ## no xrefs go to next byte
    return addr+1

  for xref in xrefs:
    ref_addr = xref.getFromAddress()
    try:
      inst = getInstructionAt(ref_addr.add(32))
    except AddressOutOfBoundsException as e:
      print("Found last xor block exiting..")
      exit()

    ## Get size of block with inst.getByte(2)
    block_size = inst.getByte(2)
    ## decrypt blocks
    dec_str = xor_block(addr,block_size)
    ## get function
    func = getFunctionBefore(ref_addr)
    new_name = "dec_"+dec_str[:-1]
    ## rename the function
    func.setName(new_name,SourceType.USER_DEFINED)
    ## log
    print("Block : {} , func : {}, dec string :
{}".format(hex(addr),func.getEntryPoint(),dec_str))
```

```python

    return addr+2*block_size

def extract_encrypted_str():

    ## starting block
    curr_block_location = 0x34035
    for i in range(200):
      curr_block_location = block(curr_block_location)

def run():
    extract_encrypted_str()

run()
```

To run the script, select created script in `Script Manager` and press Run.

Now look at the output.

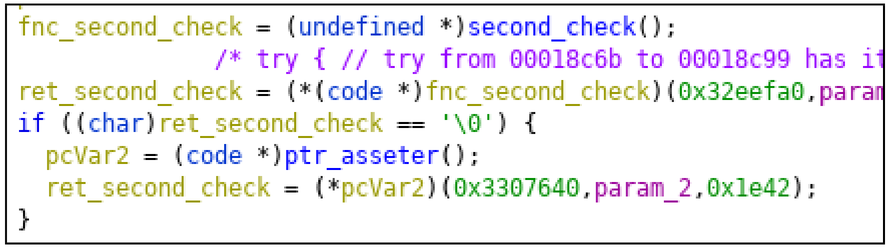


```
Console - Scripting
hydra_dec.py> Running...
Block : 0x34035 , func : 00009490, dec string : getCacheDir
Block : 0x3405b , func : 000095c0, dec string : ()Ljava/io/File;
Block : 0x3408b , func : 00009750, dec string : getAbsolutePath
Block : 0x340b9 , func : 00009880, dec string : ()Ljava/lang/String;
Block : 0x34141 , func : 00008f40, dec string : /ihzms
Block : 0x3415d , func : 0000a8c0, dec string : getSystemService
Block : 0x3418d , func : 0000a9f0, dec string : (Ljava/lang/String;)Ljava/lang/Object;
Block : 0x341e9 , func : 0000a1b0, dec string : phone
Block : 0x34203 , func : 0000ae30, dec string : android/telephony/TelephonyManager
Block : 0x34257 , func : 0000af60, dec string : getSimCountryIso
Block : 0x34287 , func : 0000b090, dec string : ()Ljava/lang/String;
Block : 0x342bf , func : 0000b360, dec string : android/telephony/TelephonyManager
Block : 0x34313 , func : 0000b490, dec string : getPhoneType
Block : 0x3433b , func : 0000b5c0, dec string : ()I
Block : 0x34351 , func : 0000ba20, dec string : android/telephony/TelephonyManager
Block : 0x343a5 , func : 0000bb50, dec string : getNetworkCountryIso
Block : 0x343dd , func : 0000bc80, dec string : ()Ljava/lang/String;
Block : 0x34415 , func : 0000c480, dec string : getResources
Block : 0x3443d , func : 0000c5b0, dec string : ()Landroid/content/res/Resources;
Block : 0x3448f , func : 0000c860, dec string : getConfiguration
Block : 0x344bf , func : 0000c990, dec string : ()Landroid/content/res/Configuration;
Block : 0x34519 , func : 0000cc40, dec string : locale
Block : 0x34535 , func : 0000cd70, dec string : Ljava/util/Locale;
Block : 0x34569 , func : 0000d140, dec string : getCountry
Block : 0x3458d , func : 0000d270, dec string : ()Ljava/lang/String;
Block : 0x345c5 , func : 0000d7e0, dec string : tr
Block : 0x345d9 , func : 0000d9c0, dec string : ;
```

ghidra script output

As you can see there are functions : `getSimCountryISO`, `getNetworkCountryIso`, `getCountry` and one suspicious string : `tr`. Without running we can assume code will check if these function's return values are equals to `tr`. I know this app targets Turkish people so this is reasonable to avoid sandbox and even manual analyze.

If you follow from these functions' xrefs to function `FUN_00018A90()` (called after time check) you can see this block :



```
fnc_second_check = (undefined *)second_check();
              /* try { // try from 00018c6b to 00018c99 has it
ret_second_check = (*(code *)fnc_second_check)(0x32eefa0,param
if ((char)ret_second_check == '\0') {
  pcVar2 = (code *)ptr_asseter();
  ret_second_check = (*pcVar2)(0x3307640,param_2,0x1e42);
}
```

country check

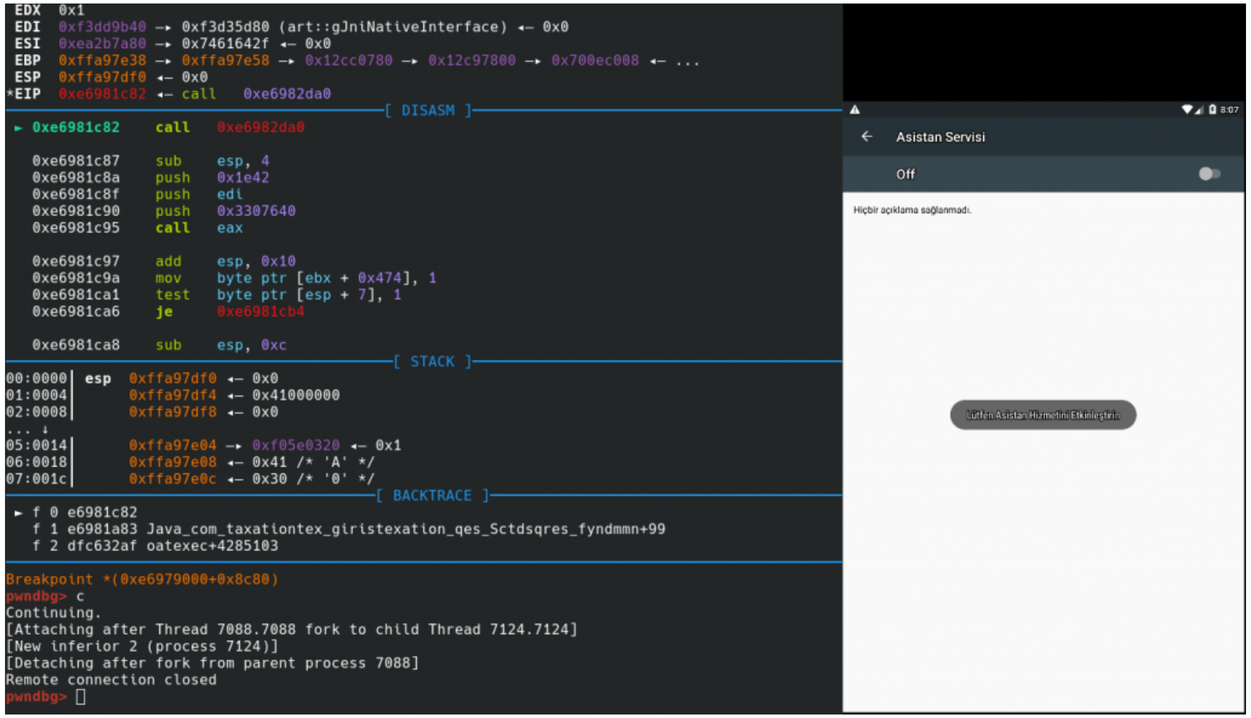
So next patch/breakpoint is this check :

`b *(base + 0x8c80)`

After these checks code will drop dex and load it. If you run without patch/breakpoints only `edevlet` page is shown and nothing happens. Get your base address and try bypassing checks :

```
b *(base + 0x8ba8)
b *(base + 0x8c80)
copy eip : .... a8 -> set $eip = .... aa
c
copy eip : .... 80 -> set $eip = .... 82
c
```

After these breakpoints, app will create dex file and load it. You will see Accessibility page pop-pup if you do it correctly.



```
 EDX  0x1
 EDI  0xf3dd9b40 —▸ 0xf3d35d80 (art::gJniNativeInterface) ◂— 0x0
 ESI  0xea2b7a80 —▸ 0x7461642f ◂— 0x0
 EBP  0xffa97e38 —▸ 0xffa97e58 —▸ 0x12cc0780 —▸ 0x12c97800 —▸ 0x700ec008 ◂— ...
 ESP  0xffa97df0 ◂— 0x0
*EIP  0xe6981c82 ◂— call   0xe6982da0
──────────────────────────────────────[ DISASM ]──────────────────────────────────────
 ► 0xe6981c82    call   0xe6982da0

   0xe6981c87    sub    esp, 4
   0xe6981c8a    push   0x1e42
   0xe6981c8f    push   edi
   0xe6981c90    push   0x3307640
   0xe6981c95    call   eax

   0xe6981c97    add    esp, 0x10
   0xe6981c9a    mov    byte ptr [ebx + 0x474], 1
   0xe6981ca1    test   byte ptr [esp + 7], 1
   0xe6981ca6    je     0xe6981cb4

   0xe6981ca8    sub    esp, 0xc
───────────────────────────────────────[ STACK ]──────────────────────────────────────
00:0000│ esp  0xffa97df0 ◂— 0x0
01:0004│      0xffa97df4 ◂— 0x41000000
02:0008│      0xffa97df8 ◂— 0x0
... ↓
05:0014│      0xffa97e04 —▸ 0xf05e0320 ◂— 0x1
06:0018│      0xffa97e08 ◂— 0x41 /* 'A' */
07:001c│      0xffa97e0c ◂— 0x30 /* '0' */
─────────────────────────────────────[ BACKTRACE ]────────────────────────────────────
 ► f 0 e6981c82
   f 1 e6981a83 Java_com_taxationtex_giristexation_qes_Sctdsqres_fyndmmn+99
   f 2 dfc632af oatexec+4285103
──────────────────────────────────────────────────────────────────────────────────────
Breakpoint *(0xe6979000+0x8c80)
pwndbg> c
Continuing.
[Attaching after Thread 7088.7088 fork to child Thread 7124.7124]
[New inferior 2 (process 7124)]
[Detaching after fork from parent process 7088]
Remote connection closed
pwndbg> 
```

8:07
Asistan Servisi
Off
Hiçbir açıklama sağlanmadı.
Lütfen Asistan Hizmetini Etkinleştirin

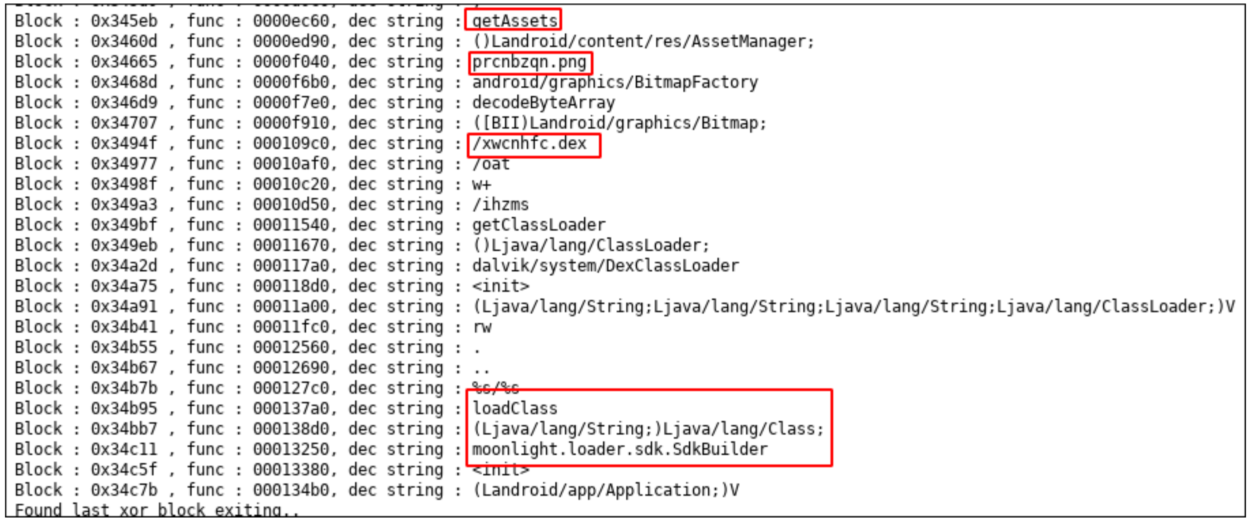
checks bypassed

Or we can patch `je` instructions to `jne` in native library and build apk again.

Understanding creation of the dex file

If you look for dropped file in filesystem, you won't see anything. File is removed with `remove`. We can attach frida and catch dropped file easily. But forget about it for now and find how png file is used to create dex file.

Look at the last parts of the ghidra script's output.

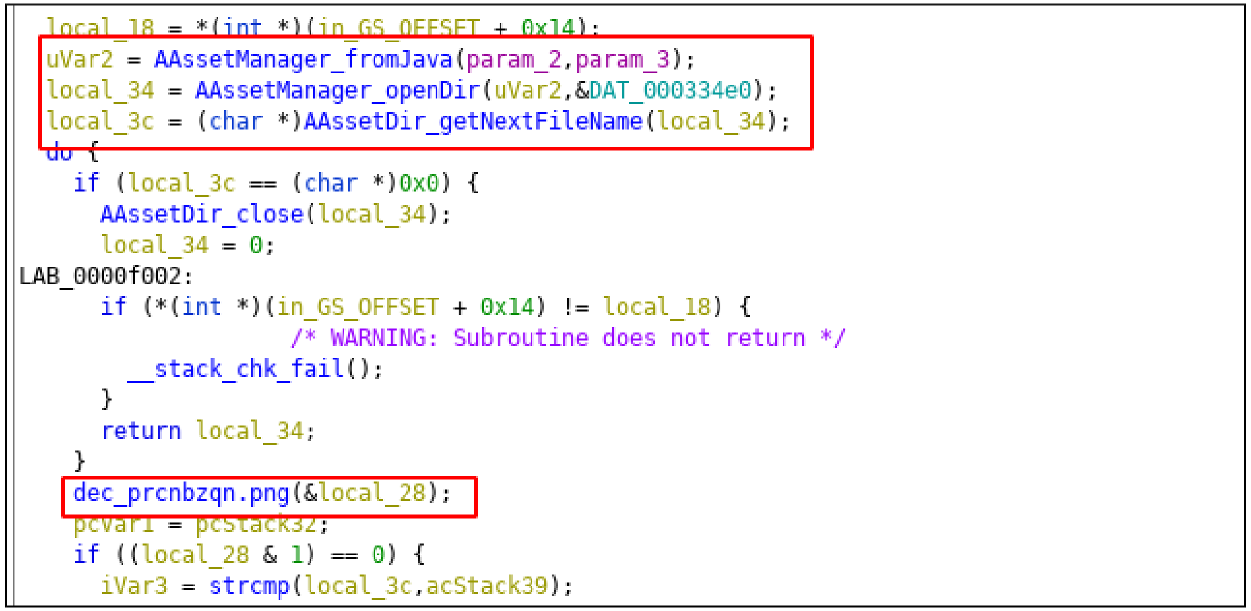


```
Block : 0x345eb , func : 0000ec60, dec string : getAssets
Block : 0x3460d , func : 0000ed90, dec string : ()Landroid/content/res/AssetManager;
Block : 0x34665 , func : 0000f040, dec string : prcnbzqn.png
Block : 0x3468d , func : 0000f6b0, dec string : android/graphics/BitmapFactory
Block : 0x346d9 , func : 0000f7e0, dec string : decodeByteArray
Block : 0x34707 , func : 0000f910, dec string : ([BII)Landroid/graphics/Bitmap;
Block : 0x3494f , func : 000109c0, dec string : /xwcnhfc.dex
Block : 0x34977 , func : 00010af0, dec string : /oat
Block : 0x3498f , func : 00010c20, dec string : w+
Block : 0x349a3 , func : 00010d50, dec string : /ihzms
Block : 0x349bf , func : 00011540, dec string : getClassLoader
Block : 0x349eb , func : 00011670, dec string : ()Ljava/lang/ClassLoader;
Block : 0x34a2d , func : 000117a0, dec string : dalvik/system/DexClassLoader
Block : 0x34a75 , func : 000118d0, dec string : <init>
Block : 0x34a91 , func : 00011a00, dec string : (Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/ClassLoader;)V
Block : 0x34b41 , func : 00011fc0, dec string : rw
Block : 0x34b55 , func : 00012560, dec string : .
Block : 0x34b67 , func : 00012690, dec string : ..
Block : 0x34b7b , func : 000127c0, dec string : %s/%s
Block : 0x34b95 , func : 000137a0, dec string : loadClass
Block : 0x34bb7 , func : 000138d0, dec string : (Ljava/lang/String;)Ljava/lang/Class;
Block : 0x34c11 , func : 00013250, dec string : moonlight.loader.sdk.SdkBuilder
Block : 0x34c5f , func : 00013380, dec string : <init>
Block : 0x34c7b , func : 000134b0, dec string : (Landroid/app/Application;)V
Found last xor block exiting..
```

ghidra script output

Somehow `prcnbzqn.png` is processed with `AndroidBitmap` and dex file is created with the name `xwchfc.dex`. Then with `ClassLoader` API dex file is loaded and `moonlight.loader.sdk.SdkBuilder` class is called.

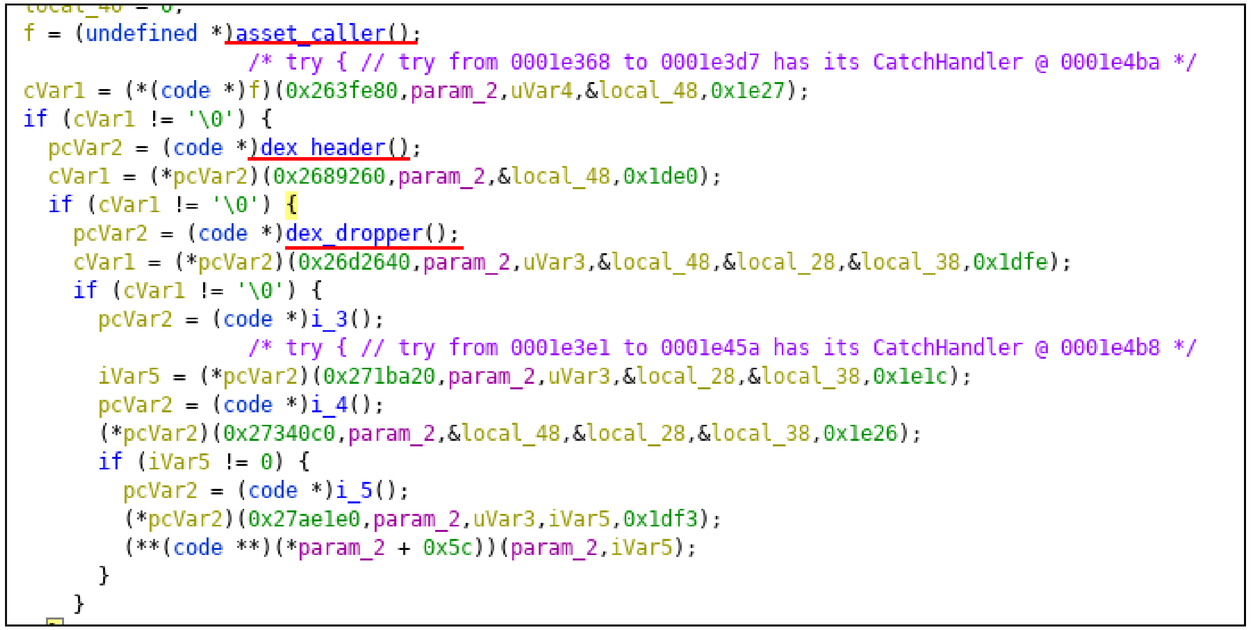
Check function : `0xeec0`



```
  local_18 = *(int *)(in_GS_OFFSET + 0x14);
  uVar2 = AAssetManager_fromJava(param_2,param_3);
  local_34 = AAssetManager_openDir(uVar2,&DAT_000334e0);
  local_3c = (char *)AAssetDir_getNextFileName(local_34);
  do {
    if (local_3c == (char *)0x0) {
      AAssetDir_close(local_34);
      local_34 = 0;
LAB_0000f002:
      if (*(int *)(in_GS_OFFSET + 0x14) != local_18) {
                    /* WARNING: Subroutine does not return */
        __stack_chk_fail();
      }
      return local_34;
    }
    dec_prcnbzqn.png(&local_28);
    pcVar1 = pcStack32;
    if ((local_28 & 1) == 0) {
      iVar3 = strcmp(local_3c,acStack39);
```

get png file from asset folder

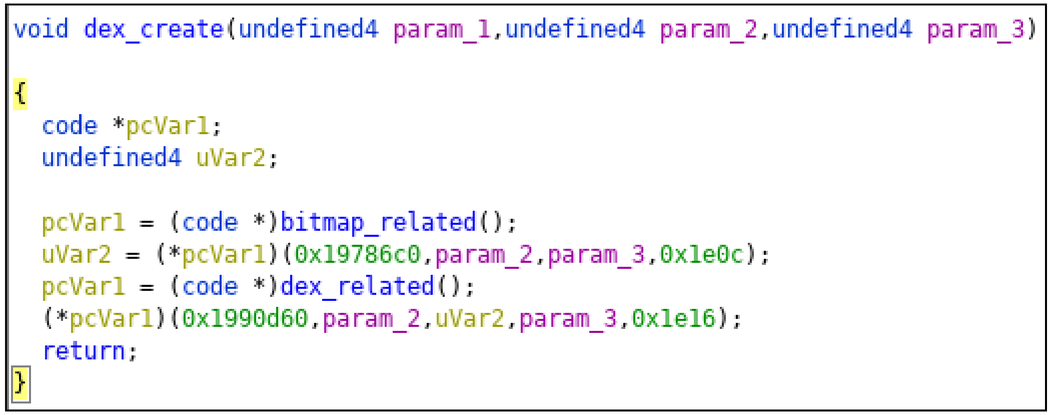
Iterates over assets and finds png file. Good. Rename this function `asset_caller`. Go to xrefs of this func and find `0xe2c0`. I renamed some of functions. `dex_header` creates dex file on memory. `dex_dropper` drops dex file to system and loads.



```
f = (undefined *)asset_caller();
                    /* try { // try from 0001e368 to 0001e3d7 has its CatchHandler @ 0001e4ba */
cVar1 = (*(code *)f)(0x263fe80,param_2,uVar4,&local_48,0x1e27);
if (cVar1 != '\0') {
  pcVar2 = (code *)dex_header();
  cVar1 = (*pcVar2)(0x2689260,param_2,&local_48,0x1de0);
  if (cVar1 != '\0') {
    pcVar2 = (code *)dex_dropper();
    cVar1 = (*pcVar2)(0x26d2640,param_2,uVar3,&local_48,&local_28,&local_38,0x1dfe);
    if (cVar1 != '\0') {
      pcVar2 = (code *)i_3();
                    /* try { // try from 0001e3e1 to 0001e45a has its CatchHandler @ 0001e4b8 */
      iVar5 = (*pcVar2)(0x271ba20,param_2,uVar3,&local_28,&local_38,0x1e1c);
      pcVar2 = (code *)i_4();
      (*pcVar2)(0x27340c0,param_2,&local_48,&local_28,&local_38,0x1e26);
      if (iVar5 != 0) {
        pcVar2 = (code *)i_5();
        (*pcVar2)(0x27ae1e0,param_2,uVar3,iVar5,0x1df3);
        (**(code **)(*param_2 + 0x5c))(param_2,iVar5);
      }
    }
```

hierarchy of functions

How dex_header creates dex file ? Go to function definition.

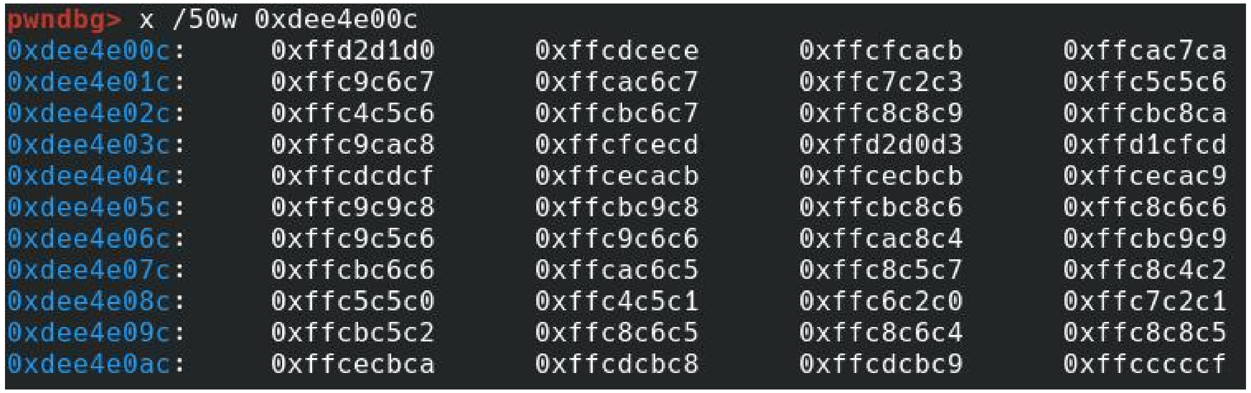


```
void dex_create(undefined4 param_1,undefined4 param_2,undefined4 param_3)

{
  code *pcVar1;
  undefined4 uVar2;

  pcVar1 = (code *)bitmap_related();
  uVar2 = (*pcVar1)(0x19786c0,param_2,param_3,0x1e0c);
  pcVar1 = (code *)dex_related();
  (*pcVar1)(0x1990d60,param_2,uVar2,param_3,0x1e16);
  return;
}
```

dex creator function

`bitmap_related` creates bitmap from png file. Bitmap object is passed to `dex_related` function. Bitmap ?

If you read png file byte byte you don't get color codes of pixels directly. You need to convert it to bitmap. So app first transfer png file to bitmap and read hex values of pixels. Fire up gimp/paint and look at the hex codes of first pixel of the image and compare with below picture 🙂

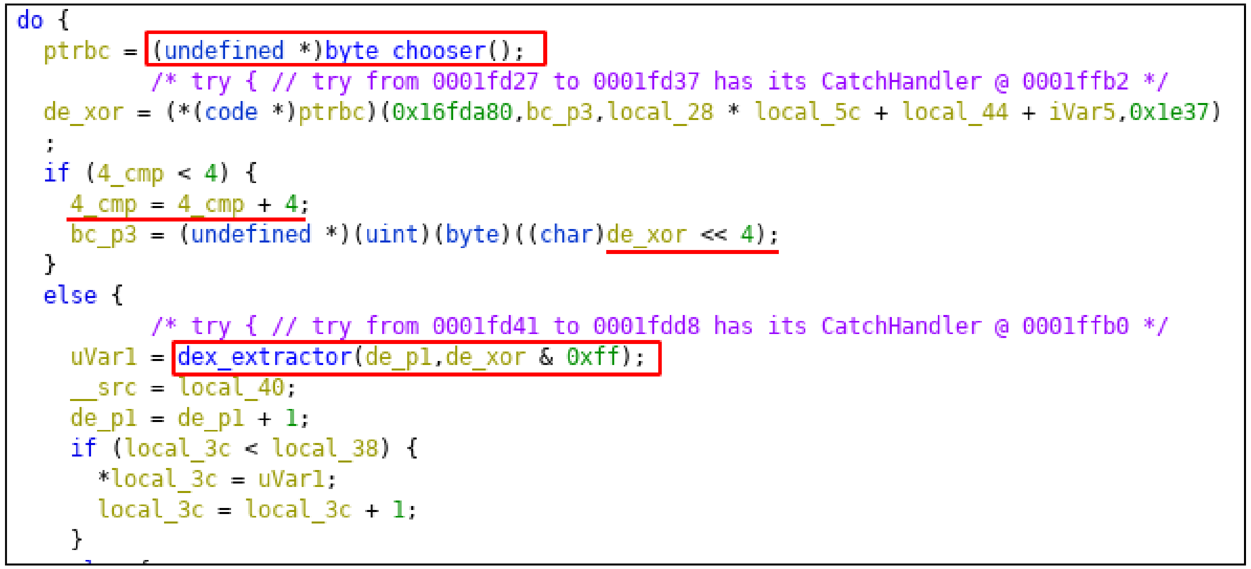


```
pwndbg> x /50w 0xdee4e00c
0xdee4e00c:     0xffd2d1d0      0xffcdcece      0xffcfcacb      0xffcac7ca
0xdee4e01c:     0xffc9c6c7      0xffcac6c7      0xffc7c2c3      0xffc5c5c6
0xdee4e02c:     0xffc4c5c6      0xffcbc6c7      0xffc8c8c9      0xffcbc8ca
0xdee4e03c:     0xffc9cac8      0xffcfcecd      0xffd2d0d3      0xffd1cfcd
0xdee4e04c:     0xffcdcdcf      0xffcecacb      0xffcecbcb      0xffcecac9
0xdee4e05c:     0xffc9c9c8      0xffcbc9c8      0xffcbc8c6      0xffc8c6c6
0xdee4e06c:     0xffc9c5c6      0xffc9c6c6      0xffcac8c4      0xffcbc9c9
0xdee4e07c:     0xffcbc6c6      0xffcac6c5      0xffc8c5c7      0xffc8c4c2
0xdee4e08c:     0xffc5c5c0      0xffc4c5c1      0xffc6c2c0      0xffc7c2c1
0xdee4e09c:     0xffcbc5c2      0xffc8c6c5      0xffc8c6c4      0xffc8c8c5
0xdee4e0ac:     0xffcecbca      0xffcdcbc8      0xffcdcbc9      0xffcccccf
```

rgb values of pixels

Now comes fun part. How these values are used. At `0xfbf0` you can find `dex_related` function.

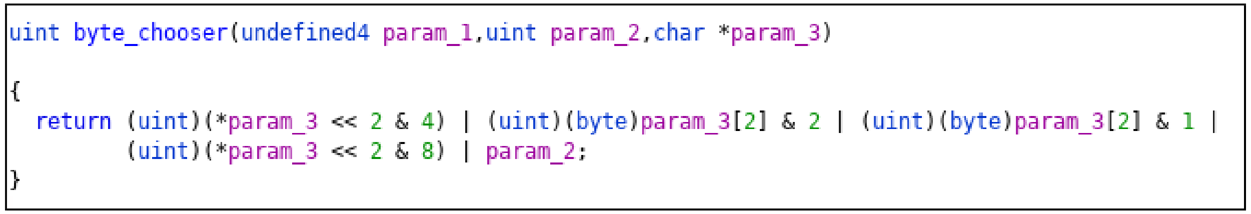
Bitmap object is passed to this function. Now there are 2 important functions here:



```
do {
  ptrbc = (undefined *)byte_chooser();
          /* try { // try from 0001fd27 to 0001fd37 has its CatchHandler @ 0001ffb2 */
  de_xor = (*(code *)ptrbc)(0x16fda80,bc_p3,local_28 * local_5c + local_44 + iVar5,0x1e37)
  ;
  if (4_cmp < 4) {
    4_cmp = 4_cmp + 4;
    bc_p3 = (undefined *)(uint)(byte)((char)de_xor << 4);
  }
  else {
          /* try { // try from 0001fd41 to 0001fdd8 has its CatchHandler @ 0001ffb0 */
    uVar1 = dex_extractor(de_p1,de_xor & 0xff);
    __src = local_40;
    de_p1 = de_p1 + 1;
    if (local_3c < local_38) {
      *local_3c = uVar1;
      local_3c = local_3c + 1;
    }
```

two important function

`byte_chooser` will return one byte and `dex_extractor` will use that byte to get final dex bytes. `4_cmp` variable is set to 0 at the beginning and will set to 0 at the end of else block. So flow will hit `byte_chooser` 2 times before entering `dex_extractor`. Here is `byte_chooser`

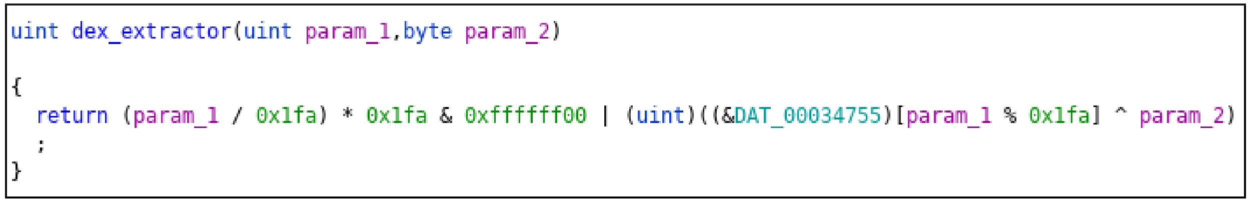


```
uint byte_chooser(undefined4 param_1,uint param_2,char *param_3)

{
  return (uint)(*param_3 << 2 & 4) | (uint)(byte)param_3[2] & 2 | (uint)(byte)param_3[2] & 1 |
         (uint)(*param_3 << 2 & 8) | param_2;
}
```

byte chooser function

param_3 is hex codes of pixels. param_2 is like seed. If its first call of byte_chooser it is set to 0. In second call of byte_chooser, param_2 will be return value of first call and left shifted by 4. Then its set to 0 at the end of else block.

After calculating the byte by calling byte_chooser twice, return value is passed to `dex_extractor`.



```
uint dex_extractor(uint param_1,byte param_2)

{
  return (param_1 / 0x1fa) * 0x1fa & 0xffffff00 | (uint)((&DAT_00034755)[param_1 % 0x1fa] ^ param_2)
  ;
}
```

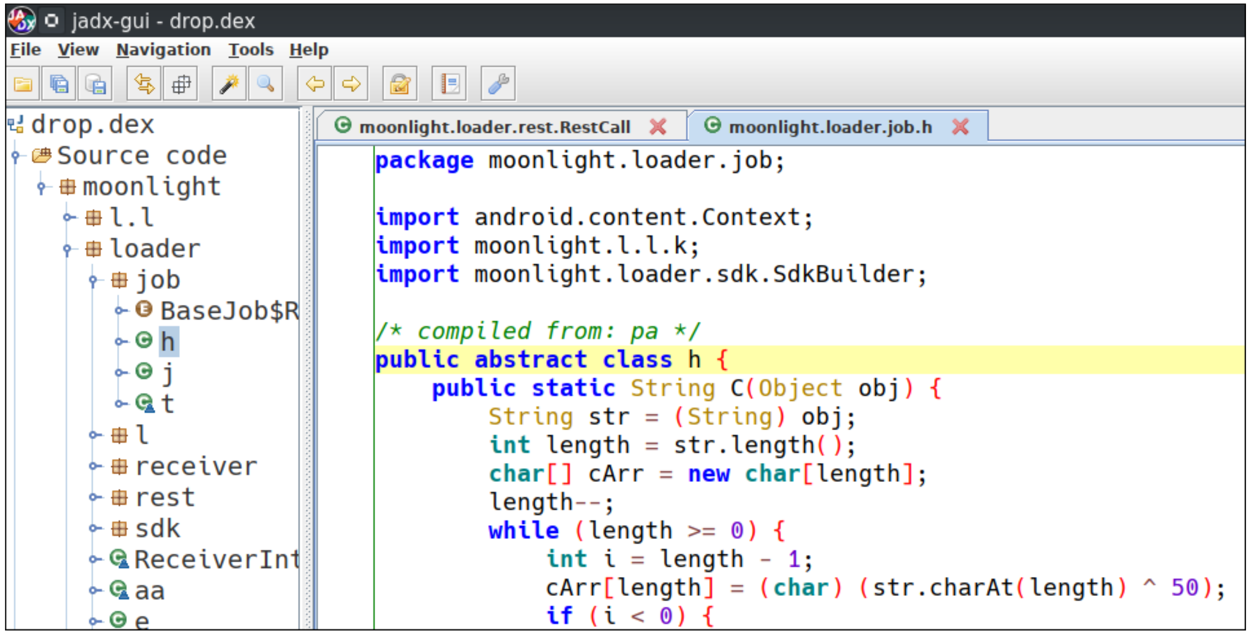
dex byte calculator function

param_2 is calculated byte param_1 is index.

Now we know how the dex file is created. Let’s do it with python

```python
from PIL import Image
import struct

image_file = "prcnbzqn.png"
so_file = "libhoter.so"
offset = 0x34755
size = 0x1fa
output_file = "drop.dex"


im = Image.open(image_file)
rgb_im = im.convert('RGB')
im_y = im.size[1]
im_x = im.size[0]

dex_size = im_y*im_x/2-255

f = open(so_file)
d = f.read()
d = d[offset:offset+size]

def create_magic(p1,p2,p3):
  return (p1<<2 &4 | p2 & 2 | p2 & 1 | p1 << 2 & 8 | p3)

def dex_extractor(p1,p2):
  return (p1/size)*size&0xffffff00| ord(d[p1%size]) ^ p2

count = 0
dex_file = open(output_file,"wb")
second = False
magic_byte = 0
for y in range(0,im.size[1]):
  for x in range(0,im.size[0]):
    r, g, b = rgb_im.getpixel((x, y))
    magic_byte = create_magic(r,b,magic_byte)
    if second:
      magic_byte = magic_byte & 0xff
      dex_byte = dex_extractor(count,magic_byte)
      dex_byte = dex_byte &0xff
      if count > 7 and count-8 < dex_size:
        dex_file.write(struct.pack("B",dex_byte))
      magic_byte = 0
      second = False
      count+=1
    else:
      magic_byte = magic_byte << 4
      second = True

dex_file.close()
```

Let’s look at the output file with jadx



```
package moonlight.loader.job;

import android.content.Context;
import moonlight.l.l.k;
import moonlight.loader.sdk.SdkBuilder;

/* compiled from: pa */
public abstract class h {
    public static String C(Object obj) {
        String str = (String) obj;
        int length = str.length();
        char[] cArr = new char[length];
        length--;
        while (length >= 0) {
            int i = length - 1;
            cArr[length] = (char) (str.charAt(length) ^ 50);
            if (i < 0) {
```

dropped dex file

Remember moonlight from output of ghidra script ? Yep this looks correct.

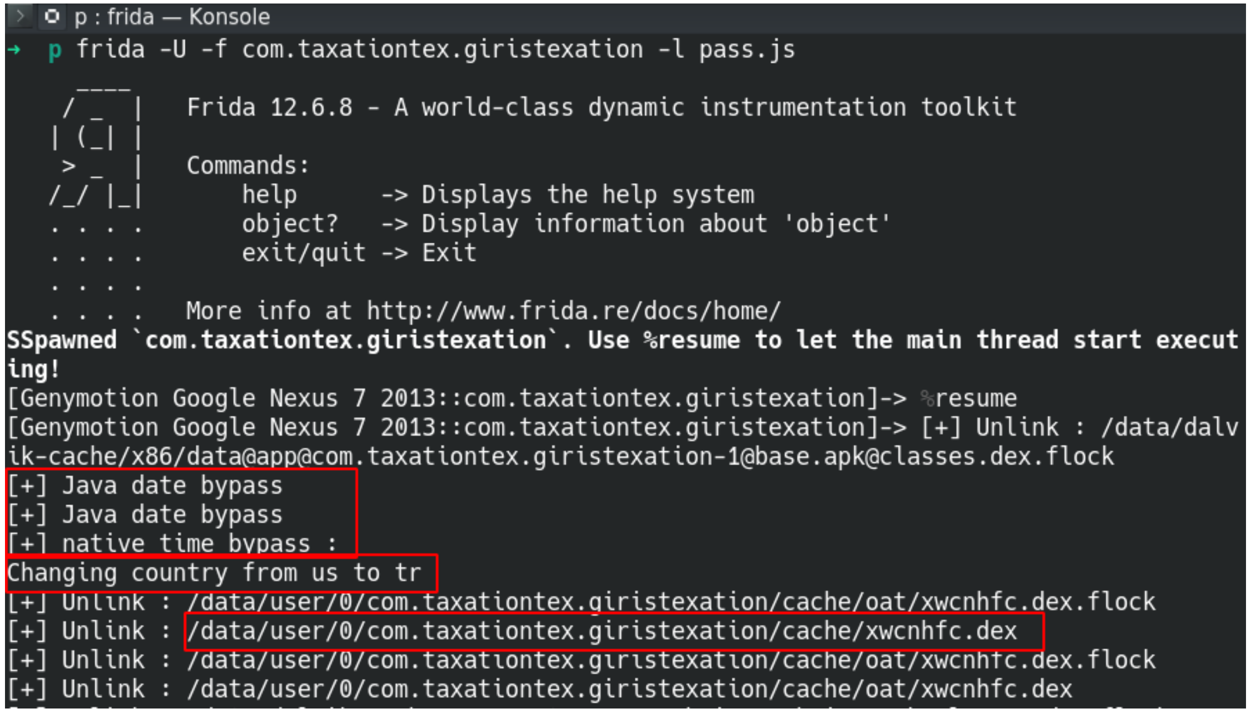
Frida <3

Well I cant write an article without mentioning frida. Bypass checks with frida.

- There are time checks on java and native side.
- Country check
- File is removed at native side.

```
var unlinkPtr = Module.findExportByName(null, 'unlink');
// remove bypass
Interceptor.replace(unlinkPtr, new NativeCallback( function (a){
     console.log("[+] Unlink : " +  Memory.readUtf8String(ptr(a)))

}, 'int', ['pointer']));
```

```
var timePtr = Module.findExportByName(null, 'time');
// time bypass
Interceptor.replace(timePtr, new NativeCallback( function (){
    console.log("[+] native time bypass : ")
    return 1554519179
},'long', ['long']));

Java.perform(function() {
    var f = Java.use("android.telephony.TelephonyManager")
    var t = Java.use('java.util.Date')
    //country bypass
    f.getSimCountryIso.overload().implementation = function(){
        console.log("Changing country from " + this.getSimCountryIso() + " to tr ")
        return "tr"
    }
    t.getTime.implementation = function(){
    console.log("[+] Java date bypass ")
    return 1554519179000
    }
 })
```



```
→  p frida -U -f com.taxationtex.giristexation -l pass.js
     ____
    / _  |   Frida 12.6.8 - A world-class dynamic instrumentation toolkit
   | (_| |
    > _  |   Commands:
   /_/ |_|       help      -> Displays the help system
   . . . .       object?   -> Display information about 'object'
   . . . .       exit/quit -> Exit
   . . . .
   . . . .   More info at http://www.frida.re/docs/home/
SSpawned `com.taxationtex.giristexation`. Use %resume to let the main thread start execut
ing!
[Genymotion Google Nexus 7 2013::com.taxationtex.giristexation]-> %resume
[Genymotion Google Nexus 7 2013::com.taxationtex.giristexation]-> [+] Unlink : /data/dalv
ik-cache/x86/data@app@com.taxationtex.giristexation-1@base.apk@classes.dex.flock
[+] Java date bypass
[+] Java date bypass
[+] native time bypass :
Changing country from us to tr
[+] Unlink : /data/user/0/com.taxationtex.giristexation/cache/oat/xwcnhfc.dex.flock
[+] Unlink : /data/user/0/com.taxationtex.giristexation/cache/xwcnhfc.dex
[+] Unlink : /data/user/0/com.taxationtex.giristexation/cache/oat/xwcnhfc.dex.flock
[+] Unlink : /data/user/0/com.taxationtex.giristexation/cache/oat/xwcnhfc.dex
```

output of frida session

Pull the dex file with `adb pull path/xwcnhfc.dex`.

Homework

This part is homework for reader 🙂 Next version of this malware only use native arm binaries. So we can't easily debug without having arm based device. But we can use our dex dropper python script. Malware sample. Load the arm binary to ghidra. Find the correct offset of the dex data block and the size of the block. dex_extractor function might look different but it does the same thing. So you need to only change the name of the files, offset and size variables at the python script. Hash of dropped dex file : 7ff02fb46009fc96c139c48c28fb61904cc3de60482663631272396c6c6c32ec

Conclusion

We attached gdb to debug native code and found certain checks. Wrote a ghidra script to automate decryption of strings and frida script to bypass checks. Also learned that png files needs to be converted with Bitmap to get pixel values. So next time you see png file and suspicious app, look for bitmap calls 😉

References

GDB Debug : https://packmad.github.io/gdb-android/

Featured image : https://www.deviantart.com/velinov/art/Hydra-monster-144496963

android malware reverse

What do you think?

14 Responses

Upvote Funny Love Surprised Angry Sad

0 Comments **Pentest Blog - Inn for security folks** 1 **Login**

Recommend Tweet Share **Sort by Best**

Start the discussion…

LOG IN WITH

OR SIGN UP WITH DISQUS ?

Name

Be the first to comment.

Subscribe Add Disqus to your site Disqus' Privacy Policy DISQUS

< N Ways to Unpack Mobile Malware

Art of Anti Detection 4 – Self-Defense >

Search...

INVICTUS CYBER SECURITY SERVICES

RECENT POSTS

Why Secure Design Matters ? Secure Approach to Session Validation on Modern Frameworks (Django Solution)

Art of Anti Detection 4 – Self-Defense

Android Malware Analysis : Dissecting Hydra Dropper

N Ways to Unpack Mobile Malware

Advisory | MailCleaner Community Edition Remote Code Execution CVE-2018-20323

LATEST COMMENTS

Ege Balci on Art of Anti Detection 3 – Shellcode Alchemy

Chase Run Taylor on Art of Anti Detection 1 – Introduction to AV & Detection Techniques

Mehmet İnce on Unexpected Journey #4 – Escaping from Restricted Shell and Gaining Root Access to SolarWinds Log & Event Manager (SIEM) Product

0x00 on Unexpected Journey #4 – Escaping from Restricted Shell and Gaining Root Access to SolarWinds Log & Event Manager (SIEM) Product

Mehmet İnce on Unexpected Journey #4 – Escaping from Restricted Shell and Gaining Root Access to SolarWinds Log & Event Manager (SIEM) Product

TAGS

0day 1day advisory antidetection backdoors binary burp bypass crypter ddos dns dos exploit flood fud icmp llmnr malware metasploit mitm netbios network office exploit packer patching phishing pivoting privilege escalation responder reversing routing shellcode sql injection sqlmap ssh tcp tunneling udp wep windows wireless word wpa wpa2 wpad

AWARDED TOP 15 PENTEST BLOG