[原创]bang加固简单分析 优

falconnnn
大侠

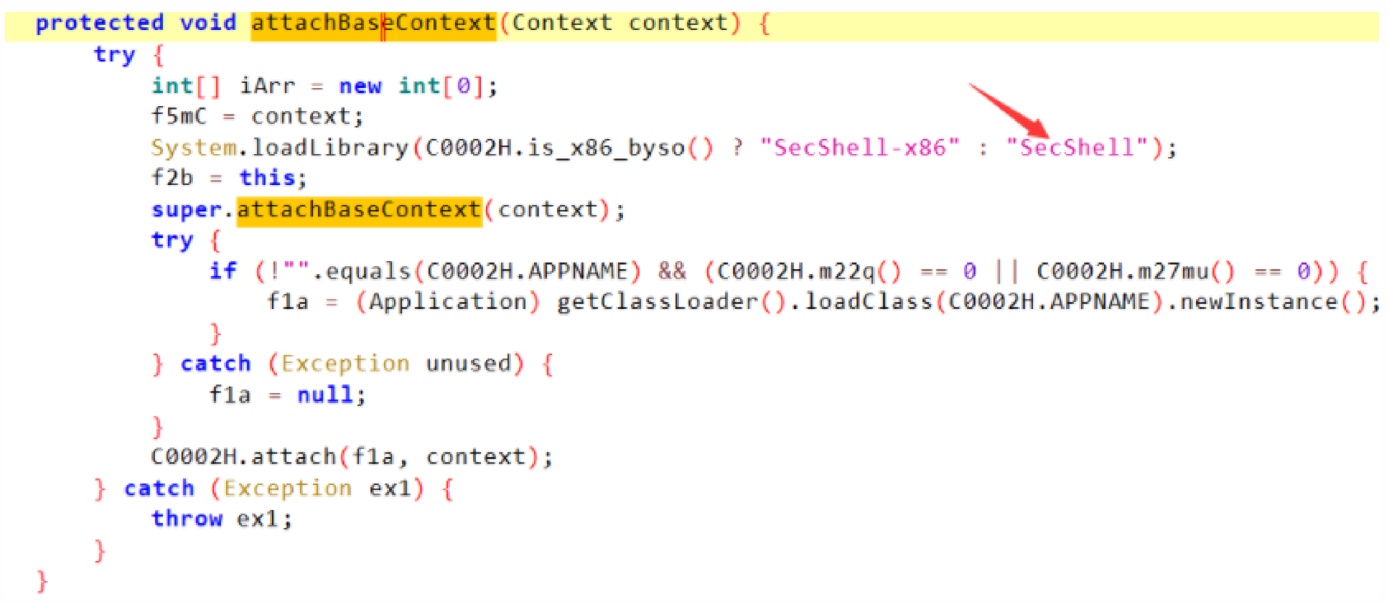
12 5 ¥ 举报

2022-7-13 21:15 6828

自己的一个demo随手就上传加固了一下，然后开始分析,是免费版的，应该不少人已经分析过了

dex

dex加固，可以使用frida-dexdump可以直接dump下来



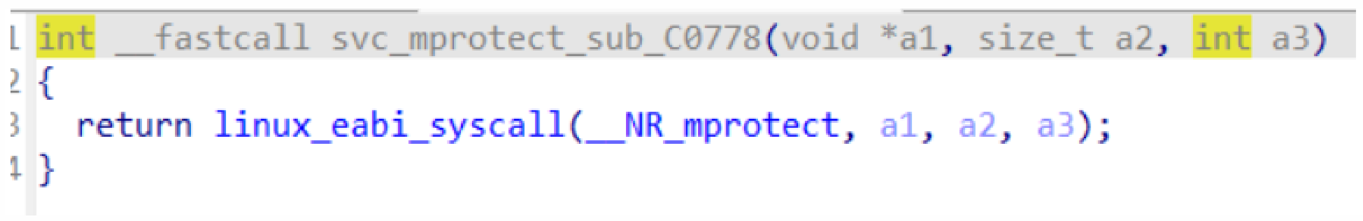
```java
protected void attachBaseContext(Context context) {
    try {
        int[] iArr = new int[0];
        f5mC = context;
        System.loadLibrary(C0002H.is_x86_byso() ? "SecShell-x86" : "SecShell");
        f2b = this;
        super.attachBaseContext(context);
        try {
            if (!"".equals(C0002H.APPNAME) && (C0002H.m22q() == 0 || C0002H.m27mu() == 0)) {
                f1a = (Application) getClassLoader().loadClass(C0002H.APPNAME).newInstance();
            }
        } catch (Exception unused) {
            f1a = null;
        }
        C0002H.attach(f1a, context);
    } catch (Exception ex1) {
        throw ex1;
    }
}
```

可以看到加载了SecShell进行脱壳调用，这个libSecShell.so是32位的

libSecShell.so

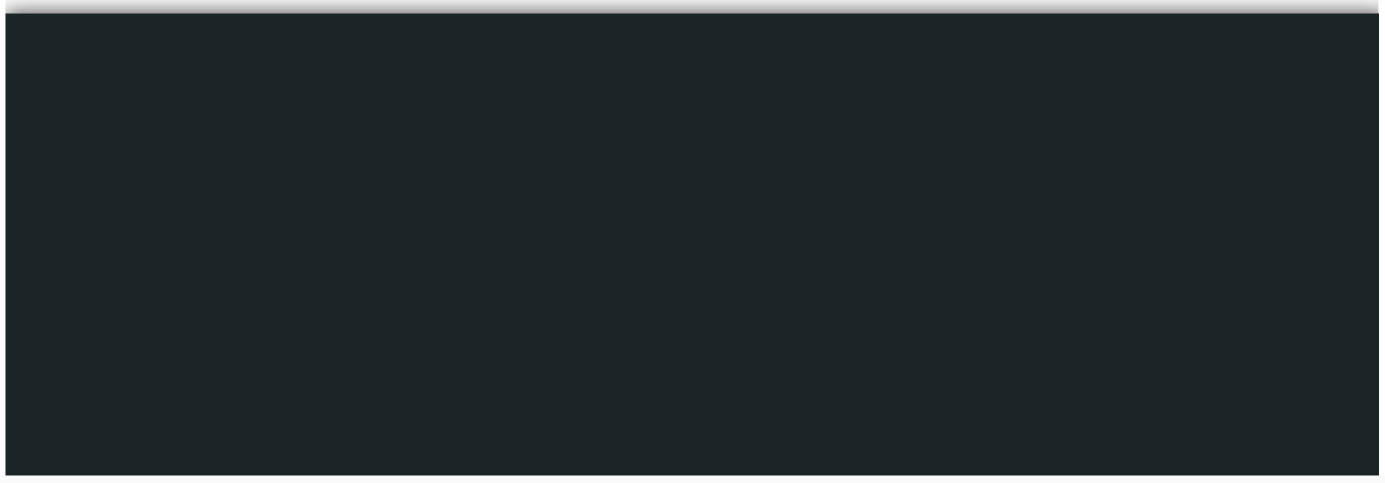
export列表中看到了JNI_Onload，但是是加密的，分析不出来，修改代码的话一定会调用mprotect，在mprotect处交叉引用，找不到调用，于是猜测可能是svc调用，用脚本跑了一下，发现了mprotect，脚本是之前论坛上看到的

```
system call : 7d 70
addr : c0783
Func Name : __NR_mprotect
 c0 70 a0 e3 00 00 00 ef
```



```c
int __fastcall svc_mprotect_sub_C0778(void *a1, size_t a2, int a3)
{
  return linux_eabi_syscall(__NR_mprotect, a1, a2, a3);
}
```

在这里交叉引用发现都在sub_C0C30里调用
用frida去hook这个函数



12

5

¥

```
var mprotect_cnt = 0
//frida -U --no-pause -f com.testlinker.ty -l hook.js
function sleep(delay) {
    var start = (new Date()).getTime();
    while ((new Date()).getTime() - start < delay) {
      continue;
    }
}

function hook_svc_mprotect() {
    let base_svc_mprotect = Module.findBaseAddress("libSecShell.so");
    if (base_svc_mprotect != null) {
        console.log("base_svc_mprotect : " + base_svc_mprotect)
    }else{
        return ;
    }
    let svc_mprotect = base_svc_mprotect.add(0xC0778);//32位
    Interceptor.attach(svc_mprotect, {
        onEnter: function(args) {
            console.log("==========================================")

            console.log("svc_mprotect: start = " + args[0] + " , len = " + args[1] + " , ATTRIBUTE
            mprotect_cnt += 1
            console.log(hexdump(base_svc_mprotect.add(0x281B4)))
        },
        onLeave: function(){
            console.log("svc_mprotect leave")
            console.log("==========================================")
        }
    })
}
function dis(address, number) {
    for (var i = 0; i < number; i++) {
        var ins = Instruction.parse(address);
        console.log("address:" + address + "--dis:" + ins.toString());
        address = ins.next;
    }
}
//libc->strstr()  从linker里面找到call_function的地址
function hook() {
//call_function("DT_INIT", init_func_, get_realpath());
    var linkermodule
    if (Process.pointerSize == 4) {
        linkermodule = Process.findModuleByName("linker");
    }else if (Process.pointerSize == 8) {
        linkermodule = Process.findModuleByName("linker64");
    }
    // var linkermodule = Process.getModuleByName("linker");
    var call_function_addr = null;
    var symbols = linkermodule.enumerateSymbols();
    for (var i = 0; i < symbols.length; i++) {
        var symbol = symbols[i];
        //LogPrint(linkername + "->" + symbol.name + "---" + symbol.address);
        if (symbol.name.indexOf("__dl__ZL13call_functionPKcPFviPPcS2_ES0_") != -1) {
            call_function_addr = symbol.address;
            //LogPrint("linker->" + symbol.name + "---" + symbol.address)
        }
    }
    Interceptor.attach(call_function_addr, {
        onEnter: function (args) {
            var type = ptr(args[0]).readUtf8String();
            var address = args[1];
            var sopath = ptr(args[2]).readUtf8String();
            console.log("loadso:" + sopath + "--addr:" + address + "--type:" + type);
            if (sopath.indexOf("libSecShell.so") != -1) {
                var libnativemodule = Process.getModuleByName("libSecShell.so");//call_function正在
                var base = libnativemodule.base;
                hook_svc_mprotect()
            }
        }
    })
}
function main() {
    hook();
}
setImmediate(main)
```

可以发现经过mprotect一次后，对应地址的值发生了变化

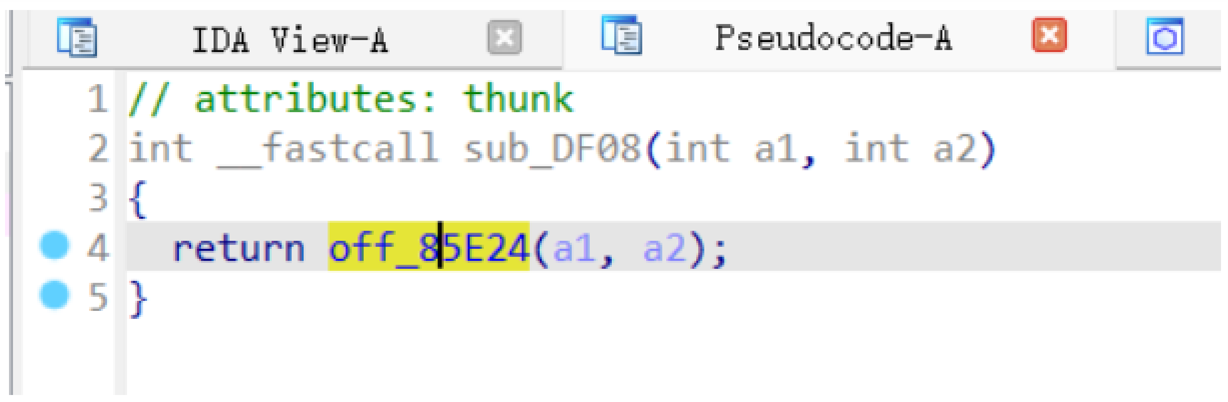
首页 论坛 课程 招聘 发现

```
[Pixel 3::com.example.cryptotest ]->
base_svc_mprotect : 0xcfaea000
========================================
svc_mprotect: start = 0xcfaea000 , len = 0xa1000 , ATTRIBUTES = 0x7
           0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F  0123456789ABCDEF
cfb121b4  9b 66 a6 75 82 ab ba fb 1a 80 e6 75 d7 0e 7f 1b  .f.u.......u....
svc_mprotect leave
========================================
========================================
svc_mprotect: start = 0xcfb8b000 , len = 0x1f000 , ATTRIBUTES = 0x3
           0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F  0123456789ABCDEF
cfb121b4  2d e9 f0 4f ad f6 ac 4d df f8 44 4e df f8 44 3e  -..O...M..DN..D>
svc_mprotect leave
========================================
========================================
svc_mprotect: start = 0xcfaea000 , len = 0xa1000 , ATTRIBUTES = 0x7
           0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F  0123456789ABCDEF
cfb121b4  2d e9 f0 4f ad f6 ac 4d df f8 44 4e df f8 44 3e  -..O...M..DN..D>
svc_mprotect leave
========================================
```

用memdumper64（github上有，速度挺快） dump出so，用Sofixer修复so文件
打开跳转到JNI_Onload（0x1E5DC）
发现ida没有自动创建函数，按p会报错The function has undefined instruction/data at the specified address
用idapython强制创建函数

```
ida_funcs.add_func(0x281B4,0x2A5CC)
```

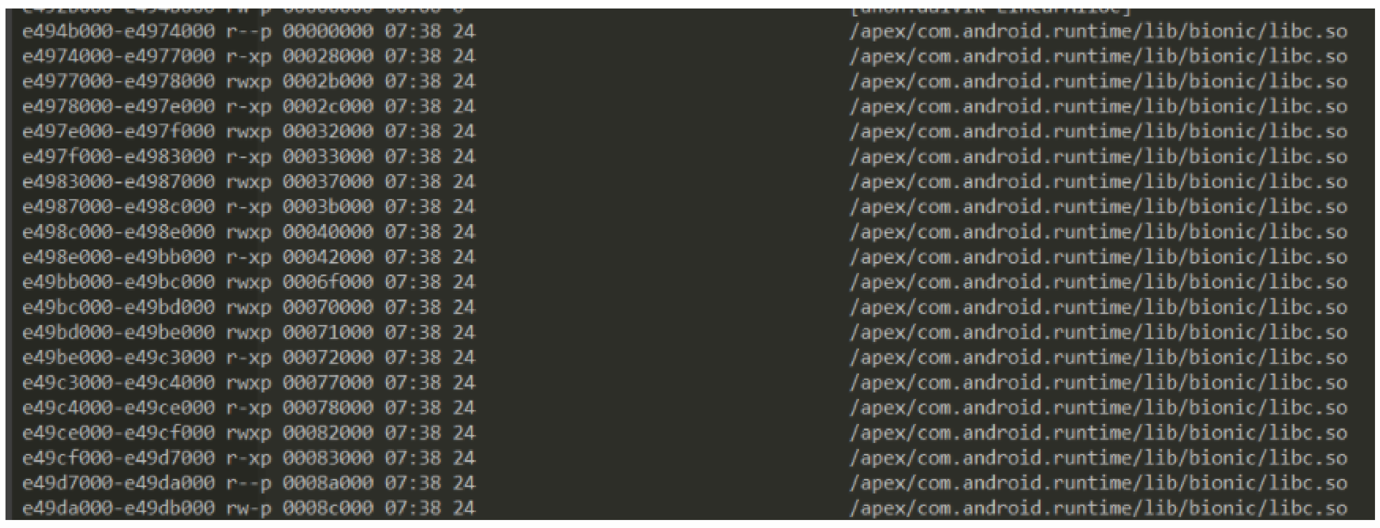
随便打开一个函数，发现是这样的



```
// attributes: thunk
int __fastcall sub_DF08(int a1, int a2)
{
  return off_85E24(a1, a2);
}
```

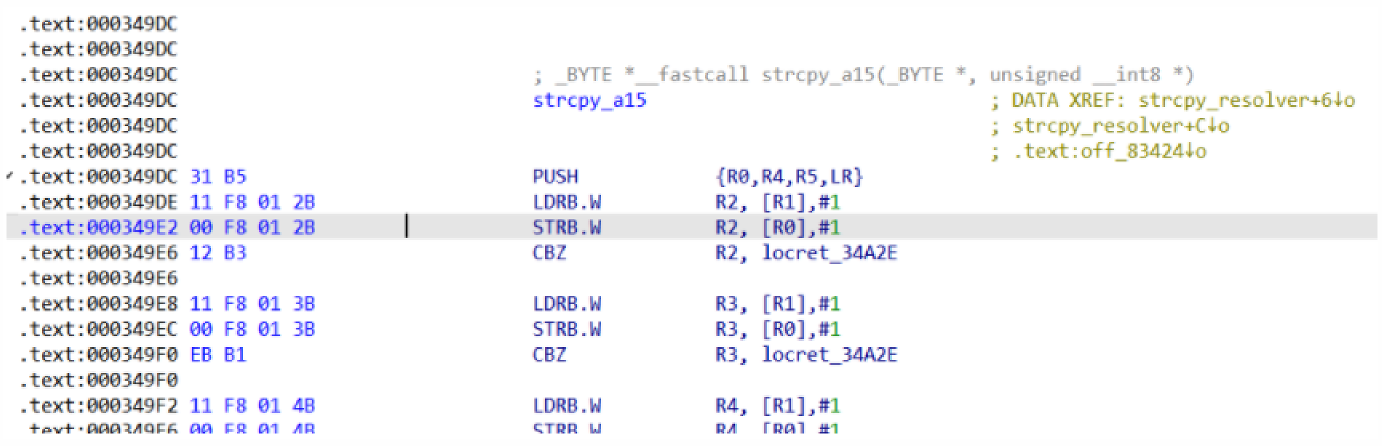
```
.data:00085E24 DD F9 97 E4 off_85E24 DCD 0xE497F9DD ; DATA XREF: sub_DF08+8↑r
```

`cat /proc/18395/maps | grep e49`看一下这个地址



```
e494b000-e4974000 r--p 00000000 07:38 24                  /apex/com.android.runtime/lib/bionic/libc.so
e4974000-e4977000 r-xp 00028000 07:38 24                  /apex/com.android.runtime/lib/bionic/libc.so
e4977000-e4978000 rwxp 0002b000 07:38 24                  /apex/com.android.runtime/lib/bionic/libc.so
e4978000-e497e000 r-xp 0002c000 07:38 24                  /apex/com.android.runtime/lib/bionic/libc.so
e497e000-e497f000 rwxp 00032000 07:38 24                  /apex/com.android.runtime/lib/bionic/libc.so
e497f000-e4983000 r-xp 00033000 07:38 24                  /apex/com.android.runtime/lib/bionic/libc.so
e4983000-e4987000 rwxp 00037000 07:38 24                  /apex/com.android.runtime/lib/bionic/libc.so
e4987000-e498c000 r-xp 0003b000 07:38 24                  /apex/com.android.runtime/lib/bionic/libc.so
e498c000-e498e000 rwxp 00040000 07:38 24                  /apex/com.android.runtime/lib/bionic/libc.so
e498e000-e49bb000 r-xp 00042000 07:38 24                  /apex/com.android.runtime/lib/bionic/libc.so
e49bb000-e49bc000 rwxp 0006f000 07:38 24                  /apex/com.android.runtime/lib/bionic/libc.so
e49bc000-e49bd000 rwxp 00070000 07:38 24                  /apex/com.android.runtime/lib/bionic/libc.so
e49bd000-e49be000 rwxp 00071000 07:38 24                  /apex/com.android.runtime/lib/bionic/libc.so
e49be000-e49c3000 r-xp 00072000 07:38 24                  /apex/com.android.runtime/lib/bionic/libc.so
e49c3000-e49c4000 rwxp 00077000 07:38 24                  /apex/com.android.runtime/lib/bionic/libc.so
e49c4000-e49ce000 r-xp 00078000 07:38 24                  /apex/com.android.runtime/lib/bionic/libc.so
e49ce000-e49cf000 rwxp 00082000 07:38 24                  /apex/com.android.runtime/lib/bionic/libc.so
e49cf000-e49d7000 r-xp 00083000 07:38 24                  /apex/com.android.runtime/lib/bionic/libc.so
e49d7000-e49da000 r--p 0008a000 07:38 24                  /apex/com.android.runtime/lib/bionic/libc.so
e49da000-e49db000 rw-p 0008c000 07:38 24                  /apex/com.android.runtime/lib/bionic/libc.so
```

发现是libc.so，把这个libc.so拖出来放到ida分析
计算一下0xE497F9DD-0xe494b000 = 0x349dd
看一下libc.so，所以这个函数就是strcpy



```
.text:000349DC
.text:000349DC
.text:000349DC                                  ; _BYTE *__fastcall strcpy_a15(_BYTE *, unsigned __int8 *)
.text:000349DC                                  strcpy_a15                              ; DATA XREF: strcpy_resolver+6↓o
.text:000349DC                                                                          ; strcpy_resolver+C↓o
.text:000349DC                                                                          ; .text:off_83424↓o
.text:000349DC 31 B5                            PUSH            {R0,R4,R5,LR}
.text:000349DE 11 F8 01 2B                      LDRB.W          R2, [R1],#1
.text:000349E2 00 F8 01 2B                      STRB.W          R2, [R0],#1
.text:000349E6 12 B3                            CBZ             R2, locret_34A2E
.text:000349E6
.text:000349E8 11 F8 01 3B                      LDRB.W          R3, [R1],#1
.text:000349EC 00 F8 01 3B                      STRB.W          R3, [R0],#1
.text:000349F0 EB B1                            CBZ             R3, locret_34A2E
.text:000349F0
.text:000349F2 11 F8 01 4B                      LDRB.W          R4, [R1],#1
.text:000349F6 00 F8 01 4B                      STRB.W          R4, [R0],#1
```

12

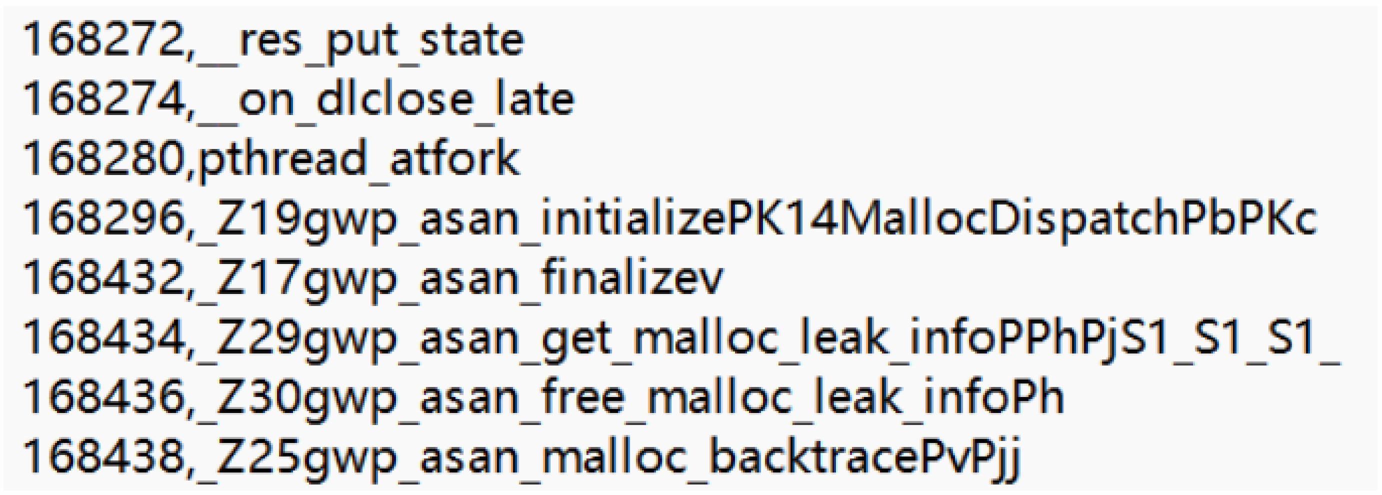
5

感觉可以写一个idapython脚本去修复一下

然后就写了一下，先从libc.so中提取函数地址和函数名

```python
from idautils import *
from idaapi import *
from idc import *
f = open("./func.txt",'w')
for func_addr in Functions(0,0x5B18BC):
    func_name = get_func_name(func_addr)
    print(func_addr , func_name)
    f.write(str(func_addr) + "," + func_name + "\n")
    # f.writelines()
f.close()
```

效果:

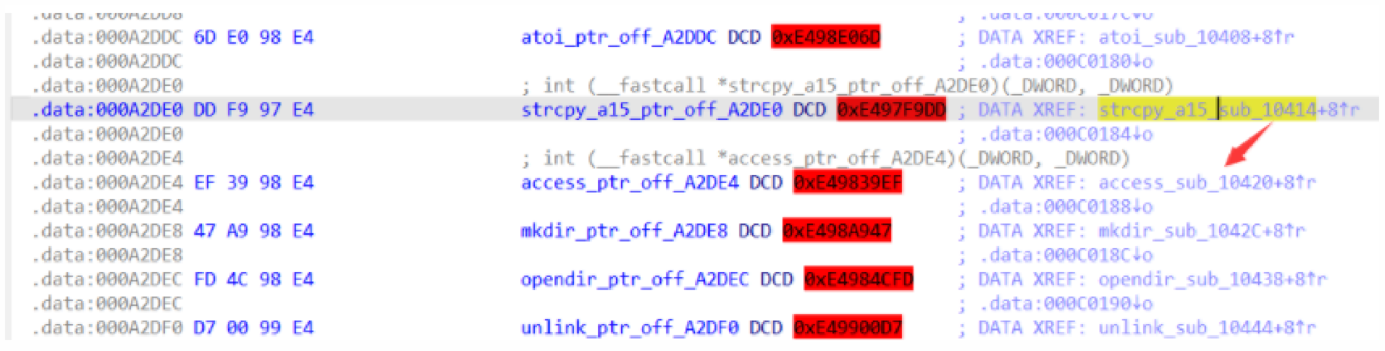


```
168272,__res_put_state
168274,__on_dlclose_late
168280,pthread_atfork
168296,_Z19gwp_asan_initializePK14MallocDispatchPbPKc
168432,_Z17gwp_asan_finalizev
168434,_Z29gwp_asan_get_malloc_leak_infoPPhPjS1_S1_S1_
168436,_Z30gwp_asan_free_malloc_leak_infoPh
168438,_Z25gwp_asan_malloc_backtracePvPjj
```

然后从.data段中找到相应地址，相减得到libc.so中地址的偏移，然后对应起来，去修改函数名

```python
from idautils import *
from idaapi import *
from idc import *
f = open(r"CryptoTest_32\CryptoTest\lib\func.txt",'r')
func_info = {}
while True:
    info = f.readline().strip('\n')
    if not info:
        break
    addr, func_name = info.split(',')
    # print(addr + func_name)
    func_info[int(addr,10)] = func_name
# print(func_info)
f.close()
textStart = 0xA2984
textEnd = 0xC2000
# textStart = 0xA2DE0
# textEnd = 0xA2E04
libc_dump_base = 0xe494b000
for i in range(textStart,textEnd,4):
    dword_ = get_dword(i)
    if dword_ > libc_dump_base:
        libc_func = dword_ - libc_dump_base
        # print(dword_,libc_func)
        func_name = func_info.get(libc_func)
        if not func_name:
            func_name = func_info.get(libc_func-1)  #thumb
        if not func_name:
            continue
        raw_name_off = get_name(i)
        patch_name_off = func_name + "_ptr_" + raw_name_off
        set_name(i,patch_name_off)
        xrefaddrs = XrefsTo(i, flags=0)
        for xrefaddr in xrefaddrs:
            raw_name = get_func_name(xrefaddr.frm)          #拿到函数原名称
            patch_fun_addr = get_name_ea_simple(raw_name)   #拿到函数地址
            # print(get_func_name(xrefaddr.frm))
            if raw_name and patch_fun_addr:
                break
        if raw_name and patch_fun_addr:
            patch_name = func_name + "_" + raw_name
            print("patch_name : ",patch_name)
            set_name(patch_fun_addr,patch_name)
        print(dword_,func_name)
```

效果如下：

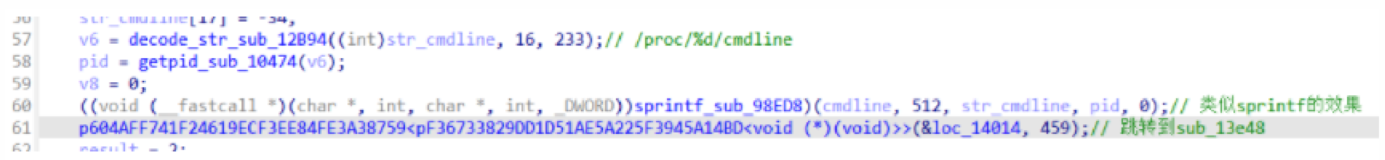


```
.data:000A2DDC 6D E0 98 E4            atoi_ptr_off_A2DDC DCD 0xE498E06D      ; DATA XREF: atoi_sub_10408+8↑r
.data:000A2DDC                                                               ; .data:000C0180↓o
.data:000A2DE0                        ; int (__fastcall *strcpy_a15_ptr_off_A2DE0)(_DWORD, _DWORD)
.data:000A2DE0 DD F9 97 E4            strcpy_a15_ptr_off_A2DE0 DCD 0xE497F9DD ; DATA XREF: strcpy_a15_sub_10414+8↑r
.data:000A2DE0                                                               ; .data:000C0184↓o
.data:000A2DE4                        ; int (__fastcall *access_ptr_off_A2DE4)(_DWORD, _DWORD)
.data:000A2DE4 EF 39 98 E4            access_ptr_off_A2DE4 DCD 0xE49839EF    ; DATA XREF: access_sub_10420+8↑r
.data:000A2DE4                                                               ; .data:000C0188↓o
.data:000A2DE8 47 A9 98 E4            mkdir_ptr_off_A2DE8 DCD 0xE498A947     ; DATA XREF: mkdir_sub_1042C+8↑r
.data:000A2DE8                                                               ; .data:000C018C↓o
.data:000A2DEC FD 4C 98 E4            opendir_ptr_off_A2DEC DCD 0xE4984CFD   ; DATA XREF: opendir_sub_10438+8↑r
.data:000A2DEC                                                               ; .data:000C0190↓o
.data:000A2DF0 D7 00 99 E4            unlink_ptr_off_A2DF0 DCD 0xE49900D7    ; DATA XREF: unlink_sub_10444+8↑r
```

这样就容易分析得多，其实不止libc.so，还有libdl.so等，不过这个函数少，就手动恢复了

init_array

地址：0x11720



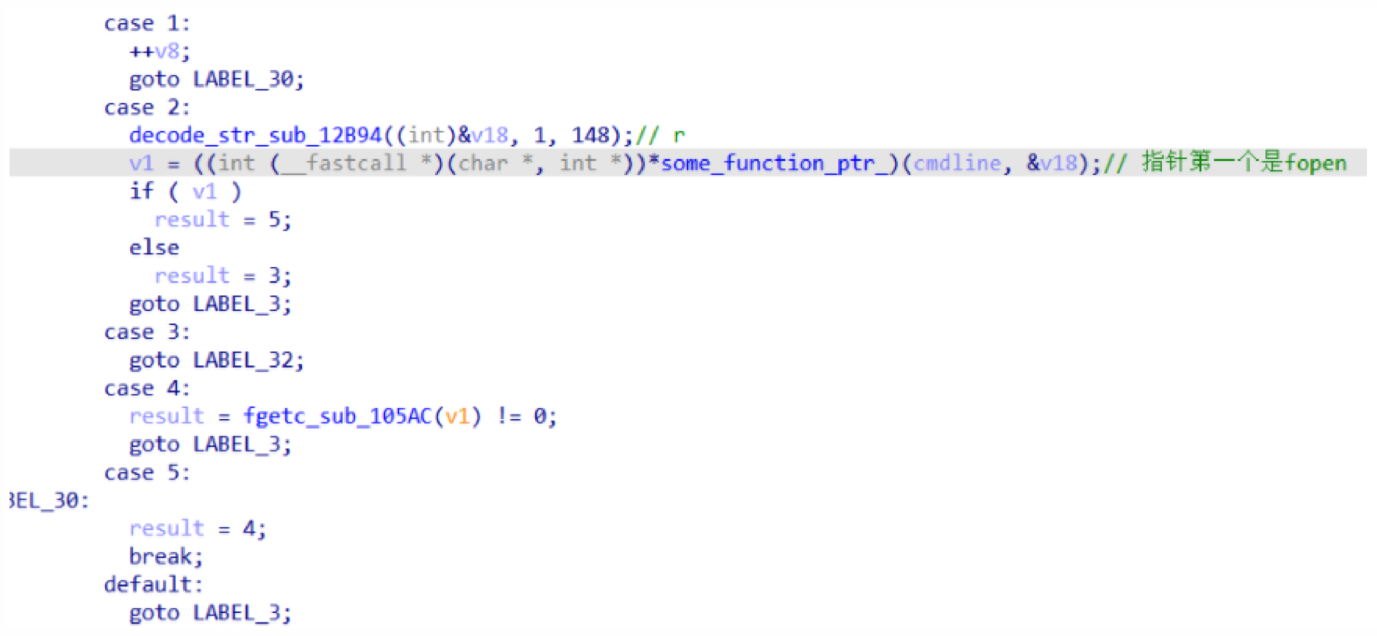
```
v6 = decode_str_sub_12B94((int)str_cmdline, 16, 233);// /proc/%d/cmdline
pid = getpid_sub_10474(v6);
v8 = 0;
((void (__fastcall *)(char *, int, char *, int, _DWORD))sprintf_sub_98ED8)(cmdline, 512, str_cmdline, pid, 0);// 类似sprintf的效果
p604AFF741F24619ECF3EE84FE3A38759<pF367338290D1D51AE5A225F3945A14BD<void (*)(void)>>(&loc_14014, 459);// 跳转到sub_13e48
```

sub_13E48：打开libc.so，通过dlsym获取了mprotect、mmap、munmap、fopen、fclose、fgets、fwrite、fread、sprintf、pthread_create函数指针

接着跟着frida的log，程序运行到了case 2

流程是case 2 -> case 5 -> case 4（读cmdline） -> case 1 -> case 5 -> case 4循环读取

这里主要是记录包名的长度，存在v8里



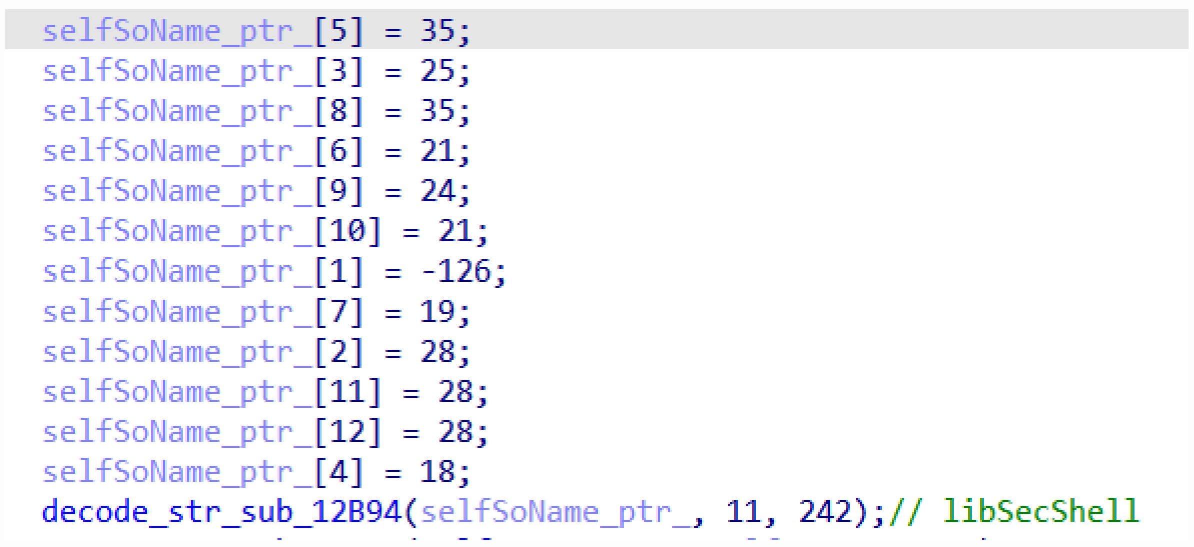
```
        case 1:
          ++v8;
          goto LABEL_30;
        case 2:
          decode_str_sub_12B94((int)&v18, 1, 148);// r
          v1 = ((int (__fastcall *)(char *, int *))*some_function_ptr_)(cmdline, &v18);// 指针第一个是fopen
          if ( v1 )
            result = 5;
          else
            result = 3;
          goto LABEL_3;
        case 3:
          goto LABEL_32;
        case 4:
          result = fgetc_sub_105AC(v1) != 0;
          goto LABEL_3;
        case 5:
3EL_30:
          result = 4;
          break;
        default:
          goto LABEL_3;
```

最终执行到case 0，读包名，然后和/system/bin/dex2oat对比，这里我包名和/system/bin/dex2oat不匹配，不进入下面的步骤（这个过程看不懂它要干啥）
然后进入到JNI_Onload

JNI_Onload

字符串解密

刚看到JNI_Onload，发现用了sub_12B94函数大量解密字符串，



```
  selfSoName_ptr_[5] = 35;
  selfSoName_ptr_[3] = 25;
  selfSoName_ptr_[8] = 35;
  selfSoName_ptr_[6] = 21;
  selfSoName_ptr_[9] = 24;
  selfSoName_ptr_[10] = 21;
  selfSoName_ptr_[1] = -126;
  selfSoName_ptr_[7] = 19;
  selfSoName_ptr_[2] = 28;
  selfSoName_ptr_[11] = 28;
  selfSoName_ptr_[12] = 28;
  selfSoName_ptr_[4] = 18;
  decode_str_sub_12B94(selfSoName_ptr_, 11, 242);// libSecShell
```

于是采用frida hook这个函数，打印出相应的信息（比如解密后的函数，返回地址），本来是只想解密字符串，但是字符串的解密顺序其实帮助了分析流程的过程，解密字符串的函数不止一个，具体的可以看看附件，写得很乱，需要注意的是这个hook的时机应该是在JNI_Onload解密之后，不然可能会出问题

```
function hook_decode_str(){
    let base_secShell = Module.findBaseAddress("libSecShell.so");
    let decode_str = base_secShell.add(0x12B94+1);
    Interceptor.attach(decode_str, {
        onEnter: function(args) {
            console.log("=======decode_str========="+ " size = " + args[1] + " op = " + args[2],"
)
            this.args0 = args[0]
            this.args1 = args[1]
        },
        onLeave: function(){
            console.log(hexdump(this.args0,{length:this.args1.toInt32()}))
            // console.log(hexdump(args[0],{length:0x10}))
        }
    })
}


function hook_svc_mprotect() {
    let base_secShell = Module.findBaseAddress("libSecShell.so");
    if (base_secShell != null) {
        console.log("base_secShell : " + base_secShell)
    }else{
        return ;
    }
    let svc_mprotect = base_secShell.add(0xC0778);//32位
    // let svc_mprotect = base_secShell.add(0x1541A0);//64位
    //private native void jniLoadScriptFromAssets(AssetManager assetManager, String assetURL, bool
    Interceptor.attach(svc_mprotect, {
        onEnter: function(args) {
            console.log("==========================================")

            console.log("svc_mprotect: start = " + args[0] + " , len = " + args[1] + " , ATTRIBUTE
            mprotect_cnt += 1
            console.log(hexdump(base_secShell.add(0x281B4)))
        },
        onLeave: function(){
            console.log("svc_mprotect leave")
            console.log("==========================================")

            if(mprotect_cnt == 2){
                hook_decode_str()
                // hook_elf_hook()
                // sleep(1000000)
            }
        }
    })
}
```

大概流程

先执行case 0：初始化JNIEnv，解密得到com/SecShell/SecShell/H字符串

然后case8（0x29e00）：

跳到sub_13E48，获取libc.so一些函数指针，从java类获取PKGNAME = "com.example.cryptotest"，后面在case8里的case分支干了一些不知道在干啥，好像是在配置环境

然后是case9：

调用android/app/ActivityThread类的currentActivityThread方法

调用ActivityThread对象的getSystemContext方法

调用ContextImpl的getPackageManager方法

调用PackageManager的getPackageInfo方法

获取PackageInfo对象的applicationInfo字段

获取ApplicationInfo对象的sourceDir字段

获取ApplicationInfo对象的nativeLibraryDir字段

拼接出/proc/%d/fd/%d，遍历fd找到base.apk路径

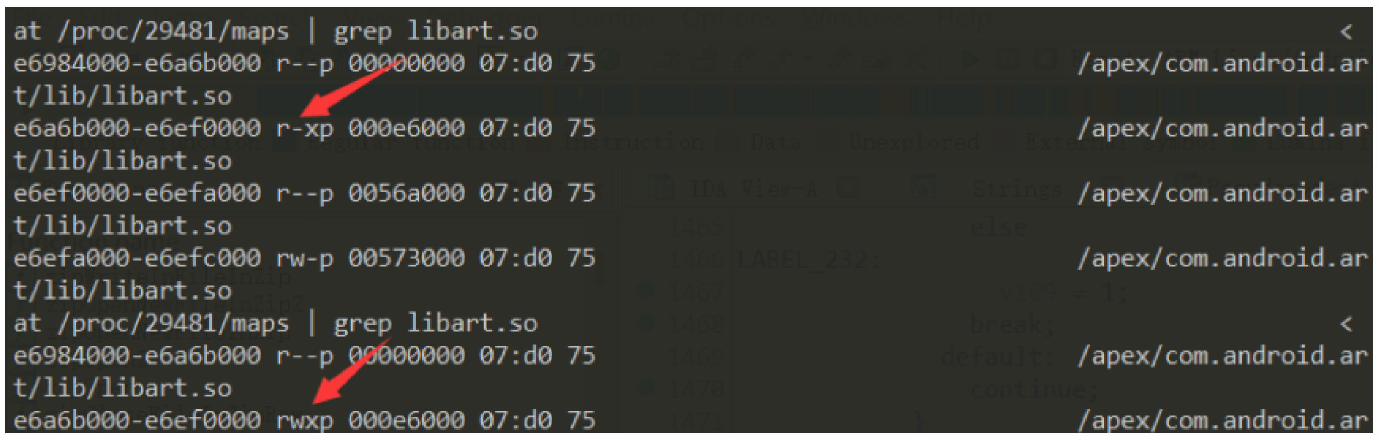
然后是case2：对小米手机进行适配

然后是case3：创建了线程（没执行到），验证了签名

case1->case10

case10：打开/proc/self/maps，找到lib/libart.so，比较是否是r-xp权限，通过格式化字符串%lx-%lx读取地址

case11：把libart.so改为可读写，两个箭头前后对比

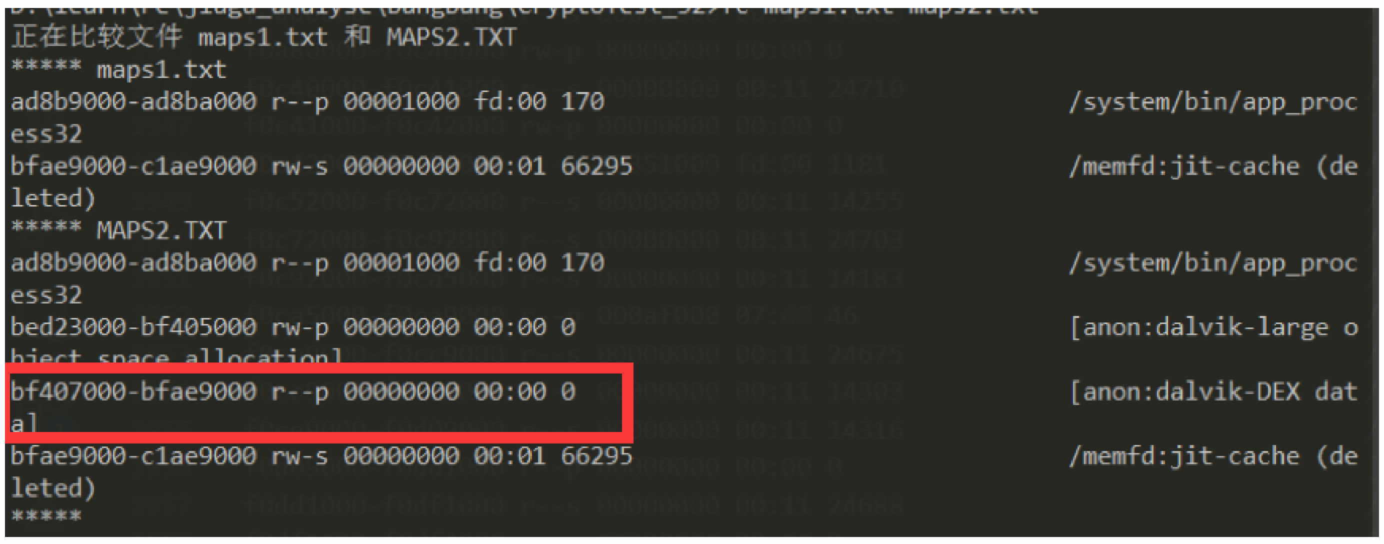


```
at /proc/29481/maps | grep libart.so                                      <
e6984000-e6a6b000 r--p 00000000 07:d0 75                        /apex/com.android.ar
t/lib/libart.so
e6a6b000-e6ef0000 r-xp 000e6000 07:d0 75                        /apex/com.android.ar
t/lib/libart.so
e6ef0000-e6efa000 r--p 0056a000 07:d0 75                        /apex/com.android.ar
t/lib/libart.so
e6efa000-e6efc000 rw-p 00573000 07:d0 75                        /apex/com.android.ar
t/lib/libart.so
at /proc/29481/maps | grep libart.so                                      <
e6984000-e6a6b000 r--p 00000000 07:d0 75                        /apex/com.android.ar
t/lib/libart.so
e6a6b000-e6ef0000 rwxp 000e6000 07:d0 75                        /apex/com.android.ar
```

case13：拼接出各种字符串，比如/data/app/~~yqfNRTFBNC4L6gA2oycp-g==/com.example.cryptotest-VtwyTKkuWOIQLYKSpK7Z5Q==/oat/arm/base.odex
然后到case13里面的case10，打开这个base.odex
会打开打开classes.dve进行校验
然后会执行sub_260BC，这里会调用0x4D7DC（hook_libc_so_func）,对hook部分说的那些函数进行hook，然后读classes.jar写到内存里的时候就调用这些函数进行解密（dex加载）
case4：把libart.so权限改回去
case12：弄了好多inlinehook，但是好像也没有执行（不知道是不是我系统版本过高）
具体见附件给的idb吧

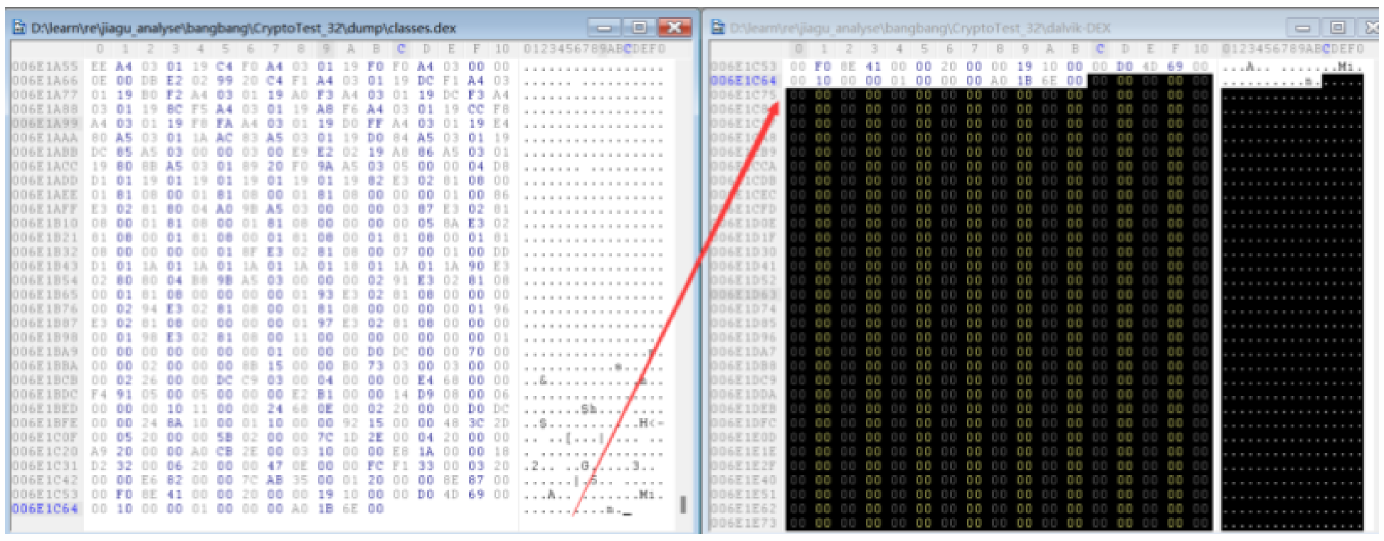
dex加载

sub_1DFB0调用了com/SecShell/SecShell/H的f方法加载/data/user/0/com.example.cryptotest/.cache/classes.jar
调用com/SecShell/SecShell/H的ff加载/data/user/0/com.example.cryptotest/.cache/v1filter.jar
通过dump maps来比较加载前和加载后的差异



```
正在比较文件 maps1.txt 和 MAPS2.TXT
***** maps1.txt
ad8b9000-ad8ba000 r--p 00001000 fd:00 170                    /system/bin/app_proc
ess32
bfae9000-c1ae9000 rw-s 00000000 00:01 66295                  /memfd:jit-cache (de
leted)
***** MAPS2.TXT
ad8b9000-ad8ba000 r--p 00001000 fd:00 170                    /system/bin/app_proc
ess32
bed23000-bf405000 rw-p 00000000 00:00 0                      [anon:dalvik-large o
bject space allocation]
bf407000-bfae9000 r--p 00000000 00:00 0                      [anon:dalvik-DEX dat
a]
bfae9000-c1ae9000 rw-s 00000000 00:01 66295                  /memfd:jit-cache (de
leted)
*****
```

可以直接把这个直接dump下来，发现解析不了，有点尴尬
于是比较一下frida-dexdump dump下来的文件，发现后面多了几百个字节，删掉就可以解析了

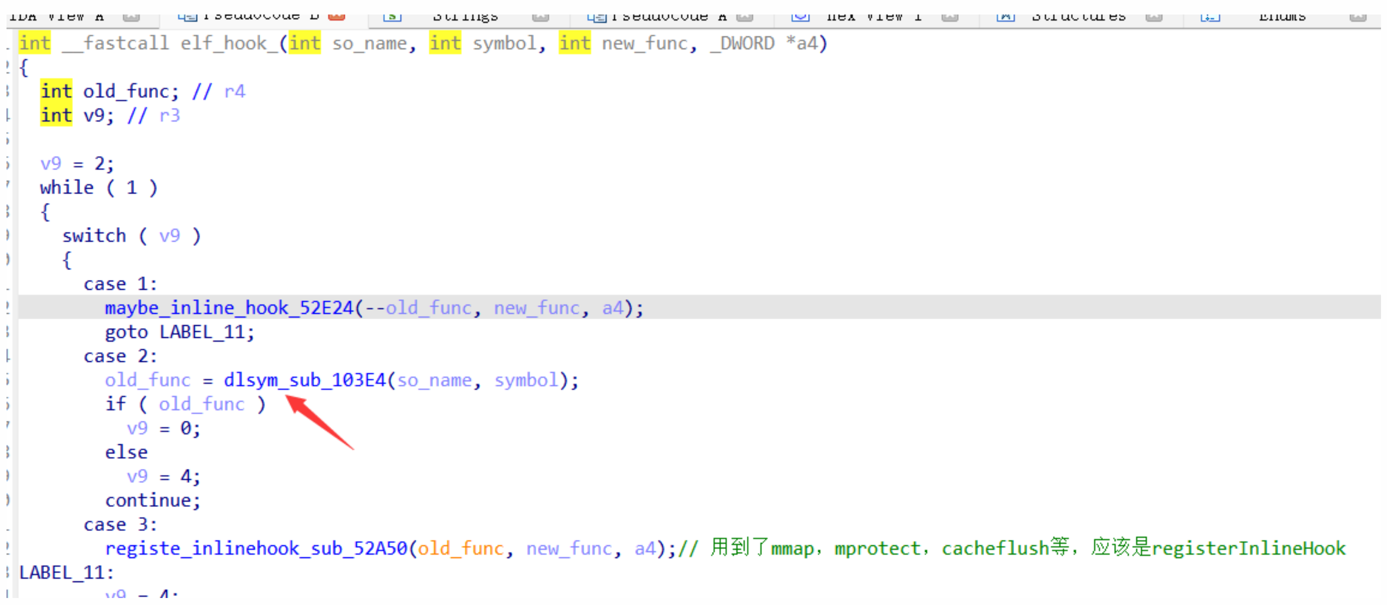


```
D:\learn\re\jiagu_analyse\bangbang\CryptoTest_32\dump\classes.dex
          0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F  10  0123456789ABCDEF0
006E1A55  EE A4 03 01 19 C4 F0 A4 03 01 19 F0 F0 A4 03 00 00  .................
006E1A66  0E 00 DB E2 02 99 20 C4 F1 A4 03 01 19 DC F1 A4 03  ...... ..........
006E1A77  01 19 B0 F2 A4 03 01 19 A0 F3 A4 03 01 19 DC F3 A4  .................
006E1A88  03 01 19 8C F5 A4 03 01 19 A8 F6 A4 03 01 19 CC F8  .................
006E1A99  A4 03 01 19 F8 FA A4 03 01 19 D0 FF A4 03 01 19 E4  .................
006E1AAA  80 A5 03 01 1A AC 83 A5 03 01 19 D0 84 A5 03 01 19  .................
006E1ABB  DC 85 A5 03 00 00 03 00 E9 E2 02 19 A8 86 A5 03 01  .................
006E1ACC  19 80 8B A5 03 01 89 20 F0 9A A5 03 05 00 00 04 D8  ....... .........
006E1ADD  D1 01 19 01 19 01 19 01 19 01 19 82 E3 02 81 08 00  .................
006E1AEE  01 81 08 00 01 81 08 00 01 81 08 00 00 00 01 00 86  .................
006E1AFF  E3 02 81 80 04 A0 98 A5 03 00 00 00 03 87 E3 02 81  .................
006E1B10  08 00 01 81 08 00 01 81 08 00 00 00 00 05 8A E3 02  .................
006E1B21  81 08 00 01 81 08 00 01 81 08 00 01 81 08 00 01 81  .................
006E1B32  08 00 00 00 00 01 8F E3 02 81 08 00 07 00 01 00 DD  .................
006E1B43  D1 01 1A 01 1A 01 1A 01 1A 01 18 01 1A 01 1A 90 E3  .................
006E1B54  02 80 80 04 B8 9B A5 03 00 00 00 02 91 E3 02 81 08  .................
006E1B65  00 01 81 08 00 00 00 00 01 93 E3 02 81 08 00 00 00  .................
006E1B76  00 02 94 E3 02 81 08 00 01 81 08 00 00 00 00 01 96  .................
006E1B87  E3 02 81 08 00 00 00 00 01 97 E3 02 81 08 00 00 00  .................
006E1B98  00 01 98 E3 02 81 08 00 11 00 00 00 00 00 00 00 01  .................
006E1BA9  00 00 00 00 00 00 00 01 00 00 00 D0 DC 00 00 70 00  ...............p.
006E1BBA  00 00 02 00 00 00 8B 15 00 00 B0 73 03 00 03 00 00  ...........s.....
006E1BCB  00 02 26 00 00 DC C9 03 00 04 00 00 00 E4 68 00 00  ..&..........h...
006E1BDC  F4 91 05 00 05 00 00 00 E2 B1 00 00 14 D9 08 00 06  .................
006E1BED  00 00 00 10 11 00 00 24 68 0E 00 02 20 00 00 D0 DC  .......$h... ....
006E1BFE  00 00 24 8A 10 00 01 10 00 00 92 15 00 00 48 3C 2D  ..$...........H<-
006E1C0F  00 05 20 00 00 5B 02 00 00 7C 1D 2E 00 04 20 00 00  .. ..[...|.... ..
006E1C20  A9 20 00 00 A0 CB 2E 00 03 10 00 00 E8 1A 00 00 18  . ...............
006E1C31  D2 32 00 06 20 00 00 47 0E 00 00 FC F1 33 00 03 20  .2.. ..G.....3.. 
006E1C42  00 00 E6 82 00 00 7C AB 35 00 01 20 00 00 8E 87 00  ......|.5.. .....
006E1C53  00 F0 8E 41 00 00 20 00 00 19 10 00 00 D0 4D 69 00  ...A.. .......Mi.
006E1C64  00 10 00 00 01 00 00 00 A0 1B 6E 00                 ............n.
```

```
D:\learn\re\jiagu_analyse\bangbang\CryptoTest_32\dalvik-DEX
          0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F  10  0123456789ABCDEF0
006E1C53  00 F0 8E 41 00 00 20 00 00 19 10 00 00 D0 4D 69 00  ...A.. .......Mi.
006E1C64  00 10 00 00 01 00 00 00 A0 1B 6E 00 00 00 00 00 00  ...........n.....
006E1C75  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .................
006E1C86  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .................
006E1C97  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .................
006E1CA8  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .................
006E1CB9  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .................
006E1CCA  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .................
006E1CDB  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .................
006E1CEC  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .................
006E1CFD  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .................
006E1D0E  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .................
006E1D1F  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .................
006E1D30  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .................
006E1D41  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .................
006E1D52  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .................
006E1D63  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .................
006E1D74  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .................
006E1D85  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .................
006E1D96  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .................
006E1DA7  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .................
006E1DB8  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .................
006E1DC9  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .................
006E1DDA  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .................
006E1DEB  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .................
006E1DFC  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .................
006E1E0D  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .................
006E1E1E  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .................
006E1E2F  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .................
006E1E40  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .................
006E1E51  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .................
006E1E62  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .................
006E1E73  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .................
```

inlineHook

地址：0x53E30是inline hook函数



```
int __fastcall elf_hook_(int so_name, int symbol, int new_func, _DWORD *a4)
{
  int old_func; // r4
  int v9; // r3

  v9 = 2;
  while ( 1 )
  {
    switch ( v9 )
    {
      case 1:
        maybe_inline_hook_52E24(--old_func, new_func, a4);
        goto LABEL_11;
      case 2:
        old_func = dlsym_sub_103E4(so_name, symbol);
        if ( old_func )
          v9 = 0;
        else
          v9 = 4;
        continue;
      case 3:
        registe_inlinehook_sub_52A50(old_func, new_func, a4);// 用到了mmap，mprotect，cacheflush等，应该是registerInlineHook
LABEL_11:
        v9 = 4;
```

12

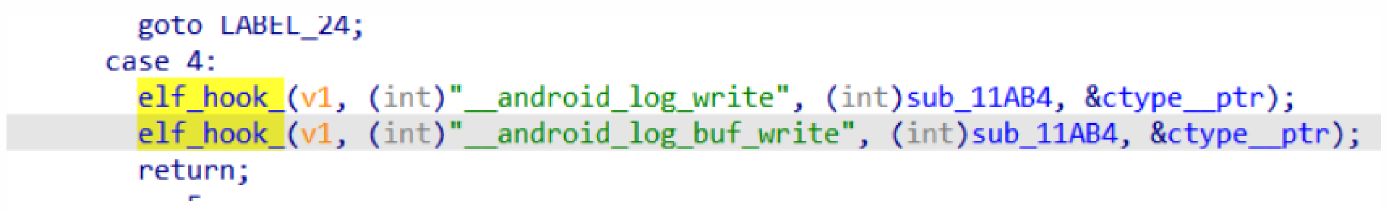
5

交叉引用可以看到很多hook的地方

比如hook了libc.so的pread64、ftruncate64、write、read、munmap、msync、__open、__openat、__mmap2

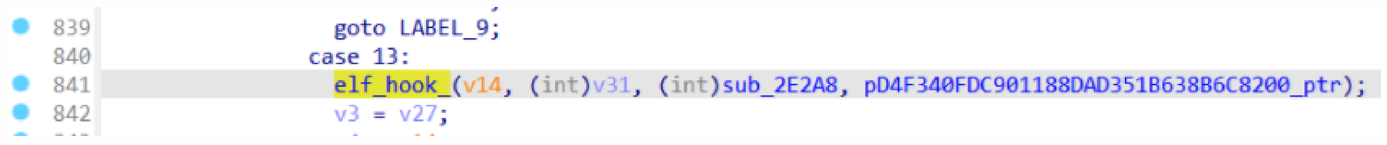
运行的时候发现其他的hook没有触发，之后用ida动态调试了一下确实是只hook了这些，其他地方不知道是不是有啥其他办法能让我断不下来

这些没有执行的地方就不过多分析了



```
        goto LABEL_24;
      case 4:
        elf_hook_(v1, (int)"__android_log_write", (int)sub_11AB4, &ctype__ptr);
        elf_hook_(v1, (int)"__android_log_buf_write", (int)sub_11AB4, &ctype__ptr);
        return;
```

函数p208CA25EFD02F087E334CA562B3F8423：

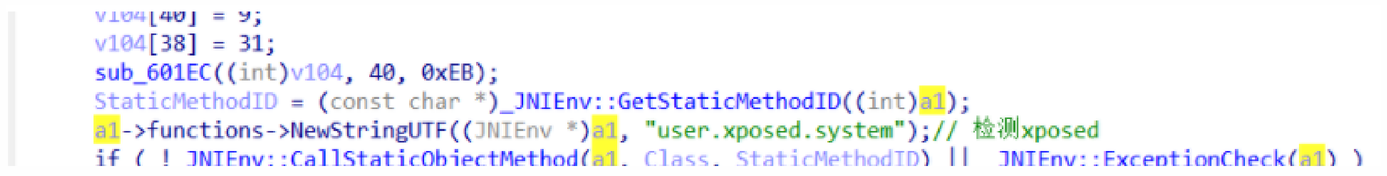


```
            goto LABEL_9;
          case 13:
            elf_hook_(v14, (int)v31, (int)sub_2E2A8, pD4F340FDC901188DAD351B638B6C8200_ptr);
            v3 = v27;
```

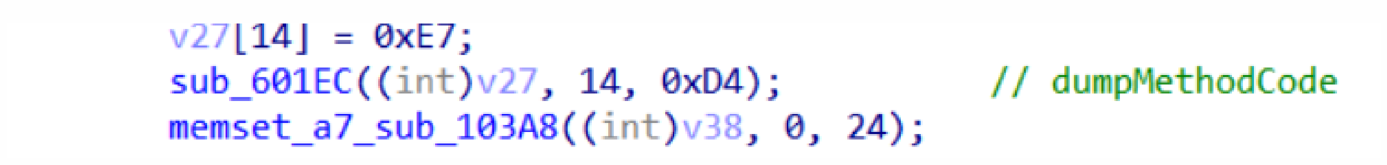
检测

地址0x60C5C：（似乎没有执行，发现这些check函数好像都没有执行）

xposed检测，fart检测等



```
v104[40] = 9;
v104[38] = 31;
sub_601EC((int)v104, 40, 0xEB);
StaticMethodID = (const char *)_JNIEnv::GetStaticMethodID((int)a1);
a1->functions->NewStringUTF((JNIEnv *)a1, "user.xposed.system");// 检测xposed
if ( ! JNIEnv::CallStaticObjectMethod(a1, Class, StaticMethodID) || JNIEnv::ExceptionCheck(a1) )
```



```
v27[14] = 0xE7;
sub_601EC((int)v27, 14, 0xD4);          // dumpMethodCode
memset_a7_sub_103A8((int)v38, 0, 24);
```

check_usb：0x25508

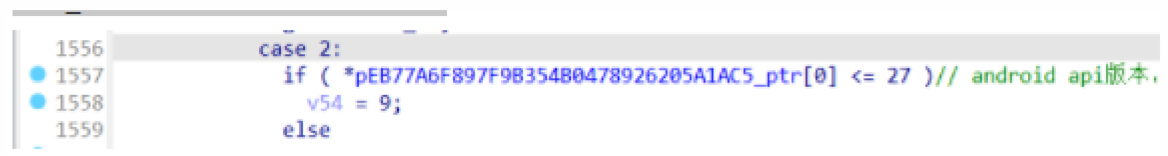
check_root：0x17D9C

其他

函数地址：0x53E30，对华为和荣耀手机进行适配

is_miuiinstaller_process对小米手机进行适配

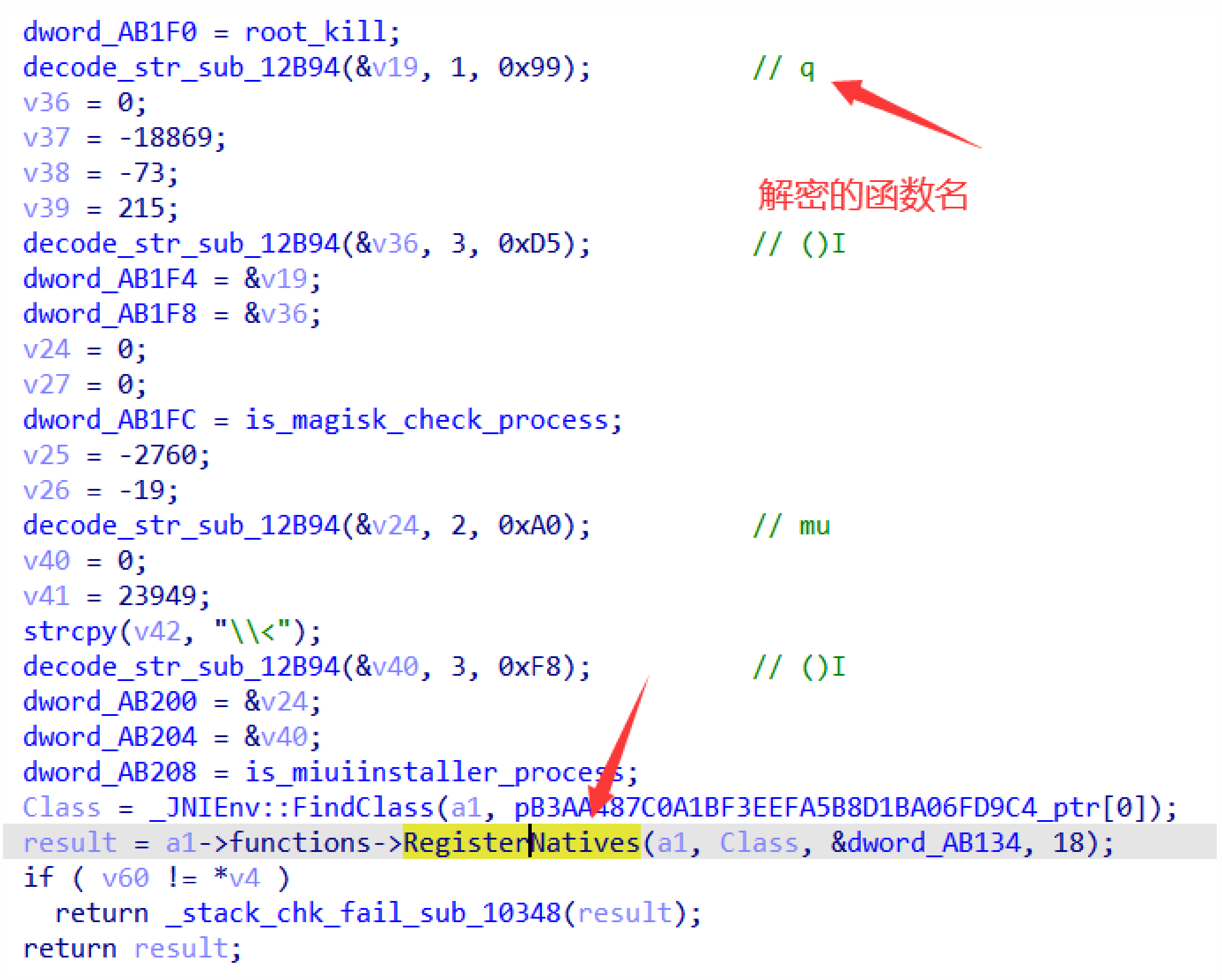
JNI_Onload里兼容性适配：



```
        case 2:
          if ( *pEB77A6F897F9B354B0478926205A1AC5_ptr[0] <= 27 )// android api版本，
            v54 = 9;
          else
```

JNI函数注册：sub_16028（通过字符串解密log很容易发现）

首页 论坛 课程 招聘 发现



```
dword_AB1F0 = root_kill;
decode_str_sub_12B94(&v19, 1, 0x99);          // q
v36 = 0;
v37 = -18869;
v38 = -73;
v39 = 215;
decode_str_sub_12B94(&v36, 3, 0xD5);          // ()I
dword_AB1F4 = &v19;
dword_AB1F8 = &v36;
v24 = 0;
v27 = 0;
dword_AB1FC = is_magisk_check_process;
v25 = -2760;
v26 = -19;
decode_str_sub_12B94(&v24, 2, 0xA0);          // mu
v40 = 0;
v41 = 23949;
strcpy(v42, "\\<");
decode_str_sub_12B94(&v40, 3, 0xF8);          // ()I
dword_AB200 = &v24;
dword_AB204 = &v40;
dword_AB208 = is_miuiinstaller_process;
Class = _JNIEnv::FindClass(a1, pB3AA487C0A1BF3EEFA5B8D1BA06FD9C4_ptr[0]);
result = a1->functions->RegisterNatives(a1, Class, &dword_AB134, 18);
if ( v60 != *v4 )
  return _stack_chk_fail_sub_10348(result);
return result;
```

用ida动态调试的时候，可能会遇到函数不会自动解析成函数，在下面框框输入这段脚本，然后用createFunction函数就可以创建函数了

```python
def createFunction(start,end):
    len_func = end - start
    begin = start
    del_items(start,0,len_func)         #先undefine
    while len_func:
        cnt = idc.create_insn(begin)
        if cnt == 0:
            break       #遇到比如off_31F40 DCD __stack_chk_guard_ptr - 0x31D78这种就不解析了，一般是
        begin += cnt
        len_func -= cnt
        print(len_func)
    #idc.create_insn(start)
    return idc.add_func(start,end)
```

【参考文献】分析一下梆x加固：https://bbs.pediy.com/thread-266247.htm

好像超过上传大小了，两个文件，apk传不上去，所有文件放在百度网盘：
链接：https://pan.baidu.com/s/1Wdjp431IhhoCbcICQCJRAg
提取码：kxuc

[2022夏季班]《安卓高级研修班(网课)》月薪三万班招生中～

最后于 2022-7-26 14:23 被falconnnn编辑，原因：

上传的附件：

打包.zip (6.33MB，23次下载)

收藏 · 12 点赞 · 5 打赏 分享

首页 论坛 课程 招聘 发现