

1. 强制挂接

ps | grep "xxx" 查找进程 id 值
kill -19 pid 暂停进程
ida 挂接, dump 出需要的数据
ida 脱离挂接
kill -18 pid 继续进程

2. 内存 dex 定位

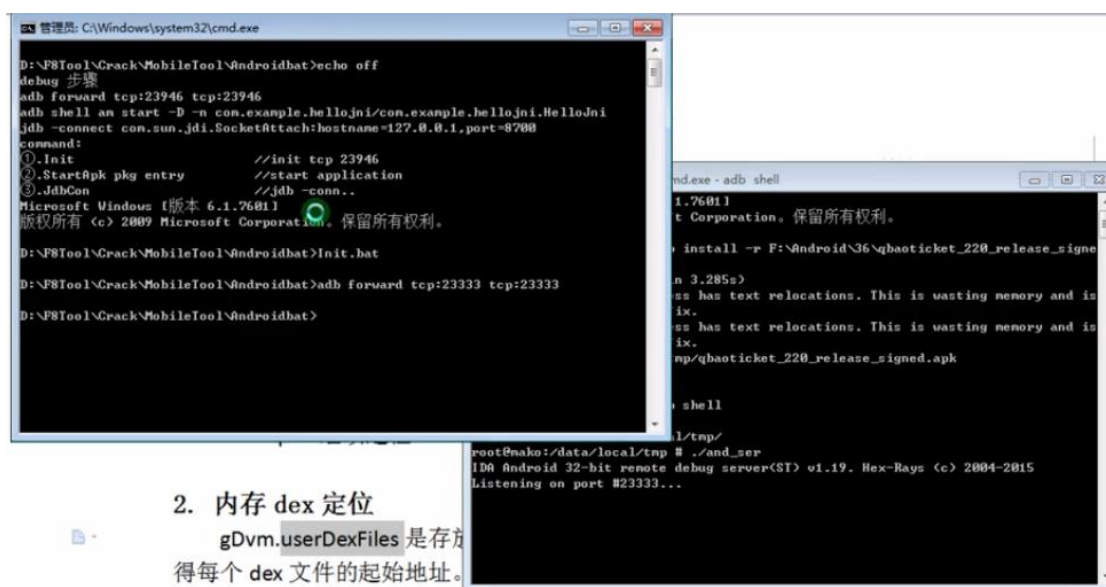
gDvm.userDexFiles 是存放 dex cookie(DexOrJar 结构)的地方, 因此可以通过遍历该结构获取每个 dex 文件的起始地址。

3. Dex 重构

通过分析内存中的 dex 存储结构, 完成对整个 dex 文件的 dump。

4. Demo 演示 (爱加密壳)

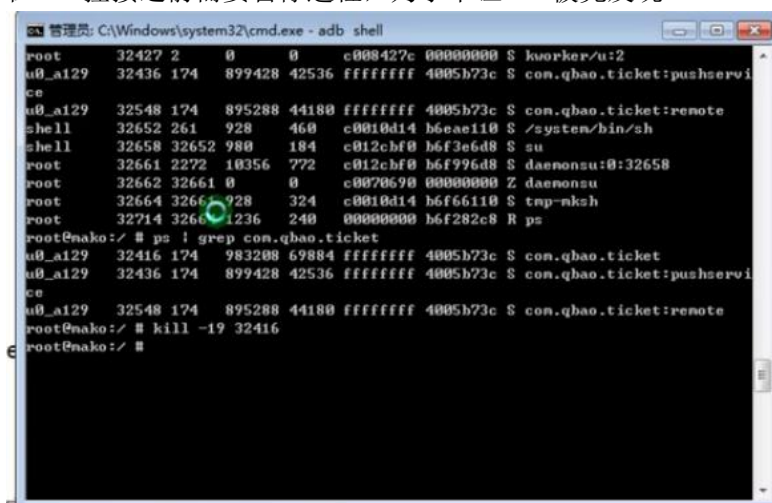
安装目标 app, 然后开启调试端口并进行端口转发



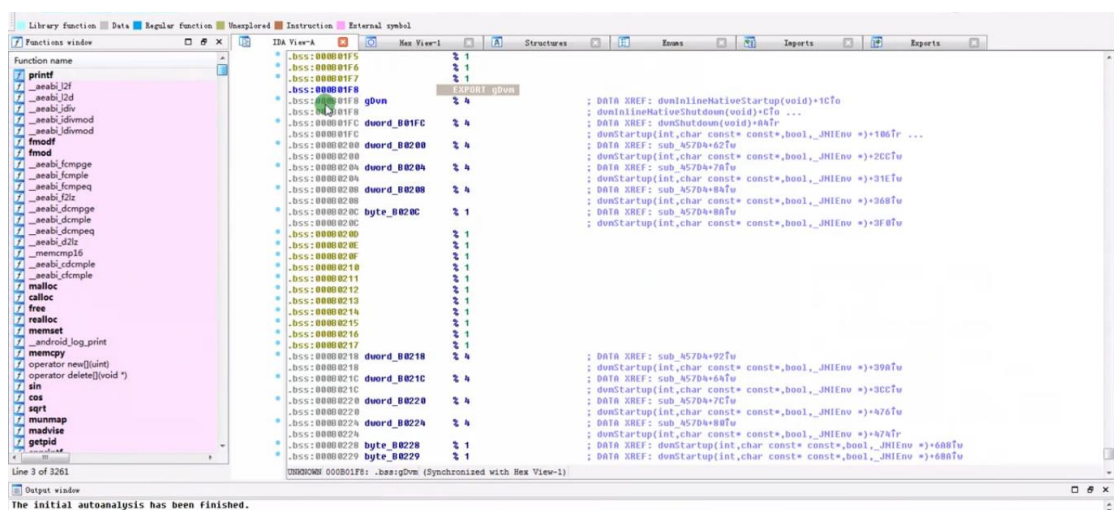
2. 内存 dex 定位

gDvm.userDexFiles 是存放
得每个 dex 文件的起始地址。

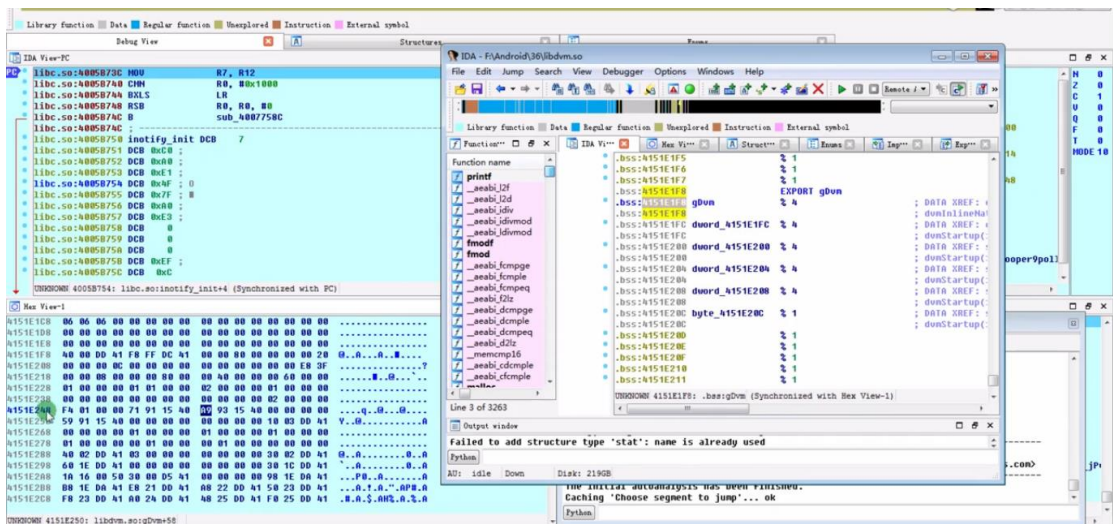
在 ida 挂接之前需要暂停进程, 为了不让 ida 被壳发现



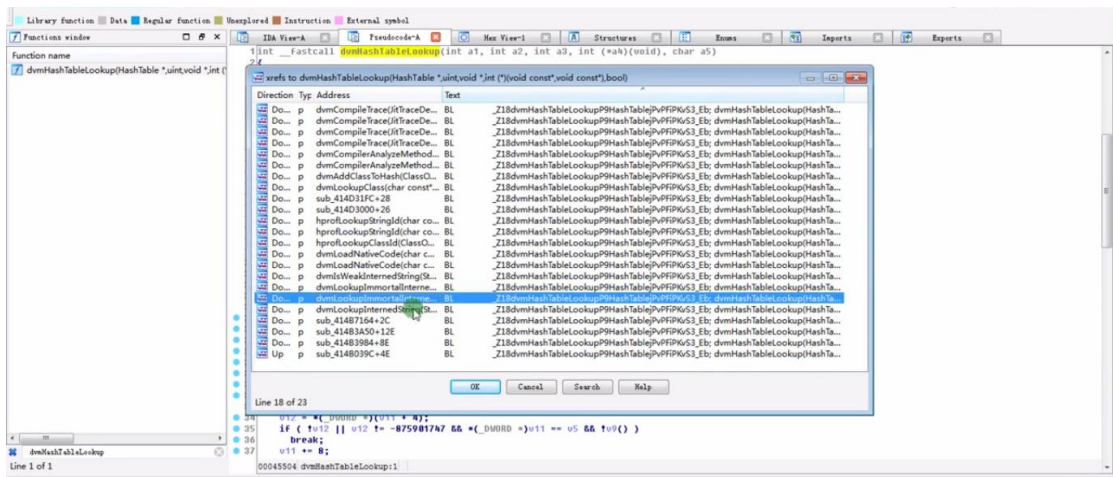
然后使用 ida 完成挂接



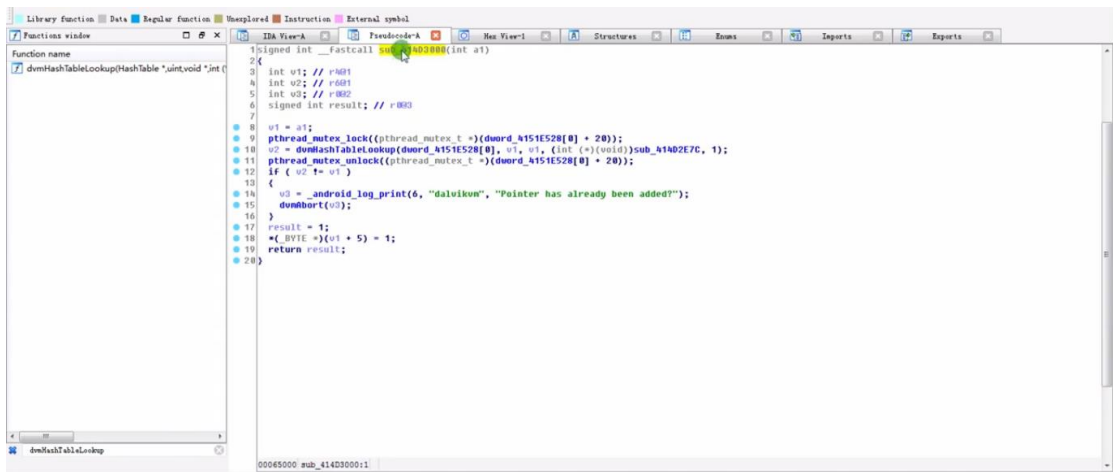
The screenshot shows the Immunity Debugger interface during the initial autoanalysis phase. The main window displays the assembly code for the 'libcrypto.so.1.0.2' library, specifically the 'libcrypto.so.1.0.2' function. The 'New View-1' window shows a hex dump of the memory at address 00000000. The 'Initial autoanalysis has been finished.' message is visible in the output window.



查找相关函数确定 gDvm 的调用位置



根据源代码中的相关信息，找到对应的函数位置

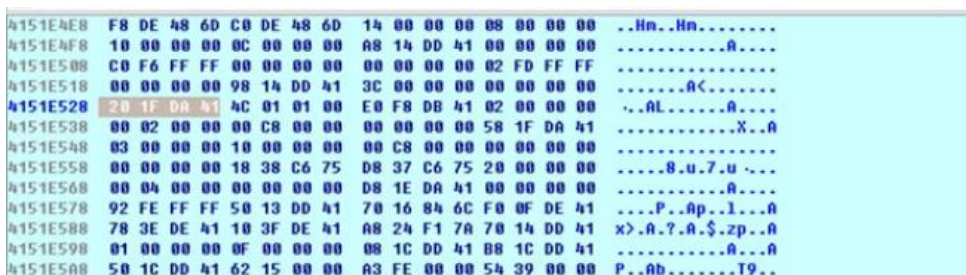


确定出 struct DvmGlobals gDvm; HashTable* userDexFiles;的地址

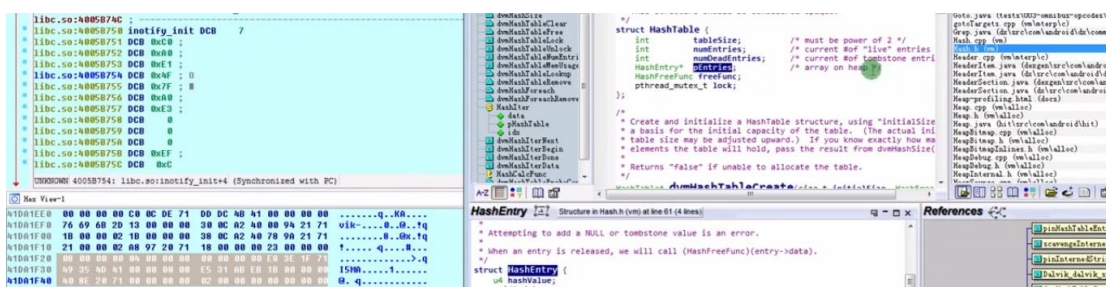
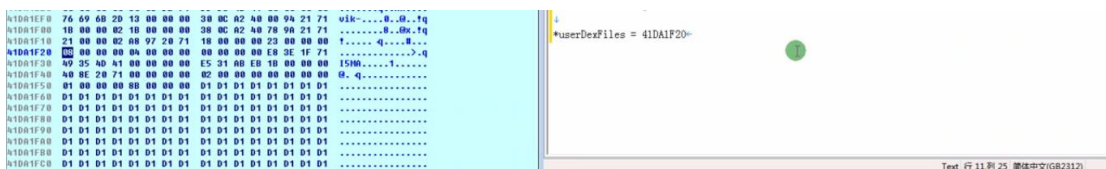
```
struct DvmGlobals gDvm; 4151E1F8
HashTable* userDexFiles; 4151E528
```

得到: userDexFiles = gDvm + 0x330

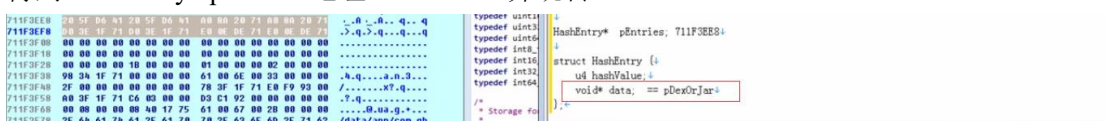
以此也确定了 userDexFiles 的位置



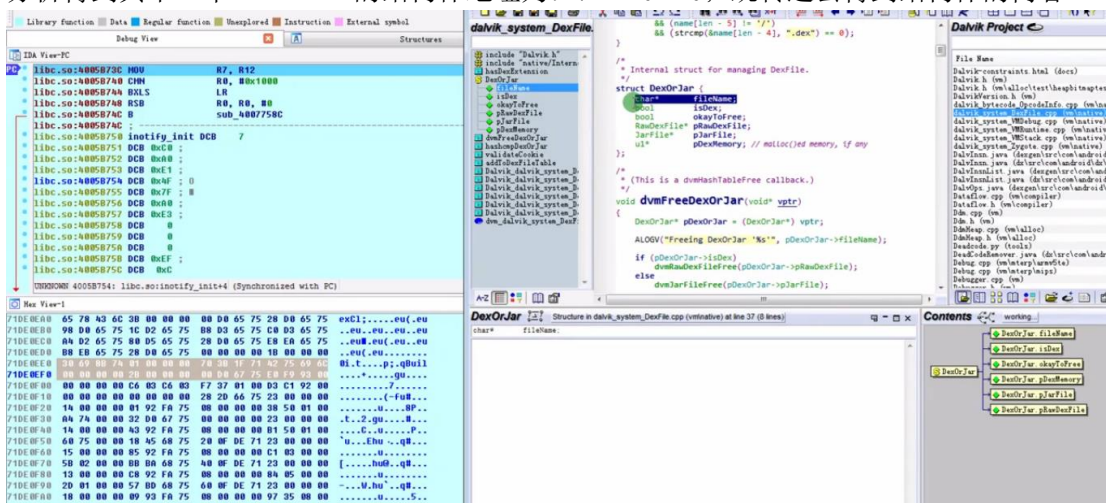
根据指针地址跳转过去，得到 userDexFiles 的 HashTable 结构体



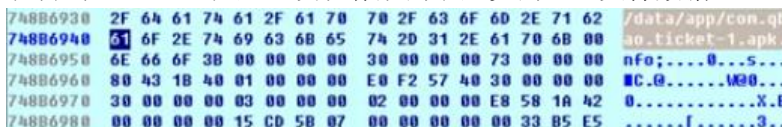
得到 HashEntry* pEntries 地址: 711F3EE8 并跳转



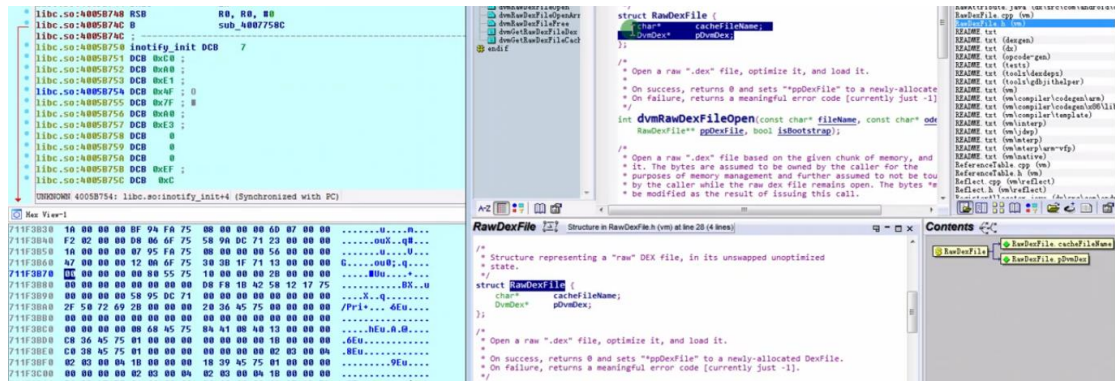
分析得到其中一个 DexOrJar 的结构体地址为: 71DE0EE0; 跳转过去得到结构体的内容



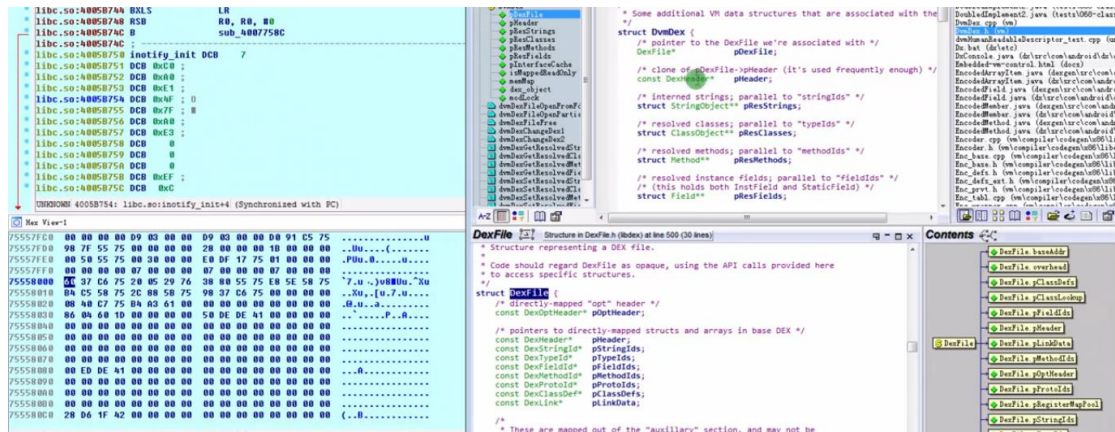
跳转到 fileName 位置，发现存放的就是壳的 dex 文件数据



回到刚才结构体的位置，确定 pRawDexFile: = 711f3b70, 跳转过去

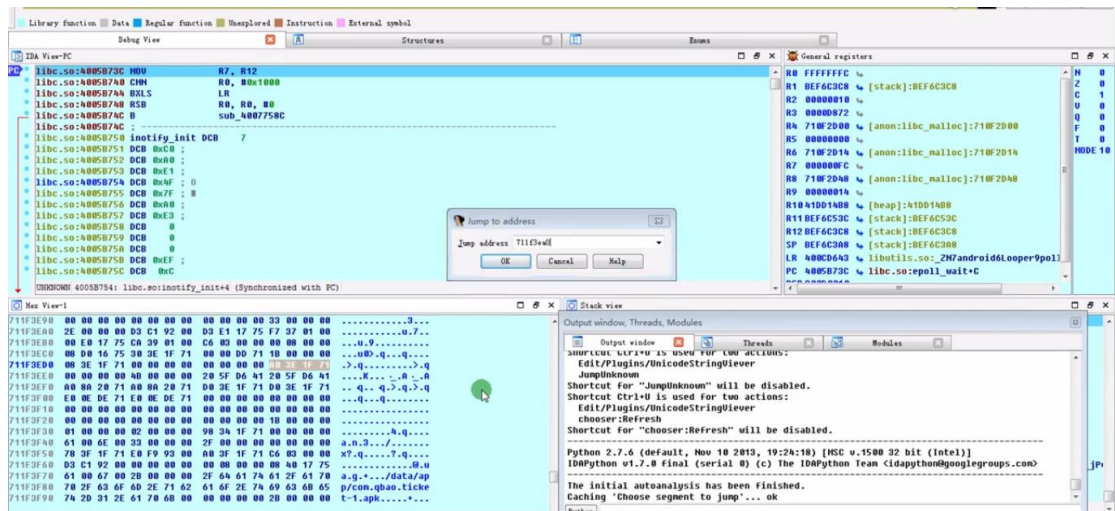


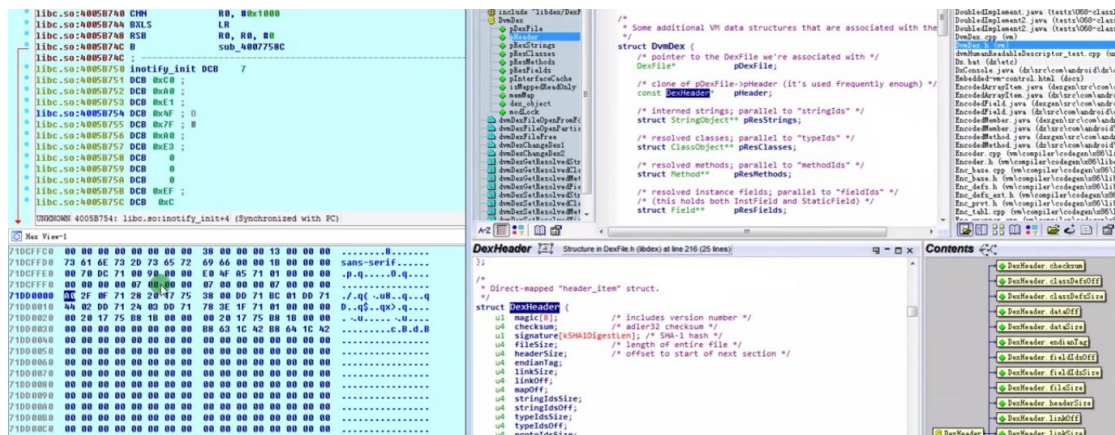
得到pDvmDex的指针，跳转过去



取 pHeader 数据地址，跳转发现 dex 头被抹去成为无效数据了，换到其他 DexEntries 按照相同的步骤进行

取倒数第二个，这次是 jar 结构，跳转过去





跳转之后找到 pHeader 位置并跳转，得到 dex 文件位置


```
Hex View-1
762904E0 07 19 05 00 00 17 05 00 2C 05 05 00 50 06 05 00 .....]...
762904F0 0E 0A 05 00 08 18 05 00 AA 1F 05 00 0C 00 05 00 .....
76290500 12 02 05 00 E1 01 05 00 22 1C 05 00 0A 10 05 00 .....
76290510 32 1A 05 00 95 0E 05 00 CA 18 05 00 35 18 05 00 2.....5...
76290520 03 0A 05 00 C2 1F 05 00 03 18 05 00 0F 0A 05 00 .....
76290530 49 12 05 00 A5 06 05 00 06 12 05 00 93 1C 05 00 1.....
76290540 D3 15 05 00 48 07 05 00 78 56 34 12 00 00 00 00 .....xU4....
76290550 00 00 00 00 CC 31 18 00 AC 87 00 00 70 00 00 00 .....P....
76290560 B3 19 00 00 20 0F 02 00 F8 23 00 00 1C 45 00 00 .....E...
76290570 EA 63 00 00 8C F5 04 00 9E 80 00 00 DC 14 08 00 .....C.....
76290580 C0 14 00 00 CC 99 00 00 E8 71 51 00 CC 31 10 00 .....Q0..1..
76290590 98 F3 46 00 9A F3 46 00 D0 F3 46 00 48 F4 46 00 ..F...F...F...F...
762905A0 4B FA 46 00 4F FA 46 00 52 FA 46 00 55 FA 46 00 K.F..F..F..F..F..
762905B0 58 FA 46 00 5B FA 46 00 5E FA 46 00 61 FA 46 00 X.F..F..F..F..F..
762905C0 64 FA 46 00 67 FA 46 00 70 FA 46 00 77 FA 46 00 d.F..g..f..p..f..w..f..
762905D0 7D FA 46 00 8D FA 46 00 A8 FA 46 00 88 FA 46 00 }.F...F...F...F...
762905E0 BE FA 46 00 CB FA 46 00 DC FA 46 00 EB FA 46 00 ..F...F...F...F...

DymDex:71DD0000
75C74008 04 68 78 9A 30 33 38 00 46 E4 C3 68 A3 41 14 E2 dex.035.F..h.A..
75C74018 6F D1 7D A8 C3 F2 59 6A 28 0F FF C4 A8 F4 38 D9 n.)...YjC.....
75C74028 B4 A3 61 00 70 00 00 78 56 34 12 00 00 00 00 .....xV4.....
75C74038 00 00 00 00 CC 31 10 00 AC B7 00 00 70 00 00 00 .....1.....p...
75C74048 B3 19 00 00 20 0F 02 00 F8 23 00 00 EC 45 03 00 .....#...E...
75C74058 EA 63 00 00 8C F5 04 00 9E 80 00 00 DC 14 08 00 .....C.....
75C74068 C0 14 00 00 CC 99 00 00 E8 71 51 00 CC 31 10 00 .....Q0..1..
75C74078 98 F3 46 00 9A F3 46 00 D0 F3 46 00 48 F4 46 00 ..F...F...F..H.F..

Text 行 30列 1
Caching 'Choose segment to jump'... ok
```

dump 脚本:

```
DesHelper (F:\CodeSrc\GnDx)
idea
python
pySide
init_py
idaapi.py
idautils.py
idc.py
python.cfg
python.p64
python.plw
DesDumper.py
DesHelper.ini
CODEXriver.py
Readme.md
External Libraries

def readmagiculeb128(addr):
    res = getByte(addr)
    len = 1
    if res < 0x7F:
        cur = getByte(addr + 1)
        res = (res << 0x7F) | ((cur <> 0x7F) << 7)
        len = 2
    if cur < 0x7F:
        cur = getByte(addr + 2)
        res |= (cur << 0x7F) << 14
        len = 3
    if cur < 0x7F:
        cur = getByte(addr + 3)
        res |= (cur << 0x7F) << 21
        len = 4
    if cur < 0x7F:
        cur = getByte(addr + 4)
        res |= cur << 28
        len = 5
    if (cur < 0x7F):
        raise TypeError("Invalid uleb128 integer encountered at offset: " + addr)
    return res, len

def readmagiculeb128(addr):
    res = getByte(addr)
    len = 1
    if res < 0x7F:
        res = rightshift((res << 25), 25)
    else:
        cur = getByte(addr + 1)
        res = (res << 0x7F) | ((cur <> 0x7F) << 7)
        len = 2
    if cur < 0x7F:
        res = rightshift((res << 18), 18)
    else:
        cur = getByte(addr + 2)
        res |= (cur << 0x7F) << 14
        len = 3
    if cur < 0x7F:
        res = rightshift((res << 11), 11)
    else:
        cur = getByte(addr + 3)
        res |= (cur << 0x7F) << 21
        len = 4
    if cur < 0x7F:
        return res, len

self.memmap = 0 #this may save the normal dexfile_ptr 0x20

def dump(self, addr):
    # print "Dexfile 0x" + hex(addr)
    self.pDexFile = getDword(addr)
    self.memmap = getDword(addr + 0x20)

def printf(self):
    print "Dexfile address: 0x" + hex(self.pDexFile)
    print "memmap address: 0x" + hex(self.memmap)

def getDexFileAddr(self):
    return self.pDexFile, self.memmap

class HashTable:
    def __init__(self):
        self.start = 0
        self.start_addr = getDword(addr)
        self.inn_count = getDword(addr + 4)
        self.handler_off = getDword(addr + 8)
        self.entries = []

    def dump(self, addr):
        self.start = 0
        self.start_addr = getDword(addr)
        self.inn_count = getDword(addr + 4)
        self.handler_off = getDword(addr + 8)
        self.entries = []

        print "HashTable 0x" + hex(addr), "Entries 0x" + hex(self.pDexFile)
        while 1:
            if getDword(self.pDexFile + 0x0) < 0:
                entry = HashEntry()
                entry.dump(self.pDexFile + 0x0)
                self.entries.append(entry)
                1entry += 1
            else:
                break

def printf(self):
    for i in range(0, len(self.entries)):
        print "cookie: ", i
        self.entries[i].printf()
```

```
class DexFile:
    def __init__(self):
        self.pOptHeader = 0
        self.pHeader = 0
        self.pStringIds = 0
        self.pTypeIds = 0
        self.pFieldIds = 0
        self.pMethodIds = 0
        self.pProtoIds = 0
        self.pClassDefs = 0
        self.pLinkData = 0
        self.pBaseAddr = 0
        self.pOptHeader = OptHeader()
        self.dexHeader = DexHeader()
        self.dump_dir = ""

    def dumpHeader(self, addr):
        global baseAddr
        sig = getWord(addr)
        if sig == 0x00000000:
            self.pOptHeader = addr
            self.pHeader = addr + 0x28
        else:
            self.pOptHeader = 0
            self.pHeader = addr

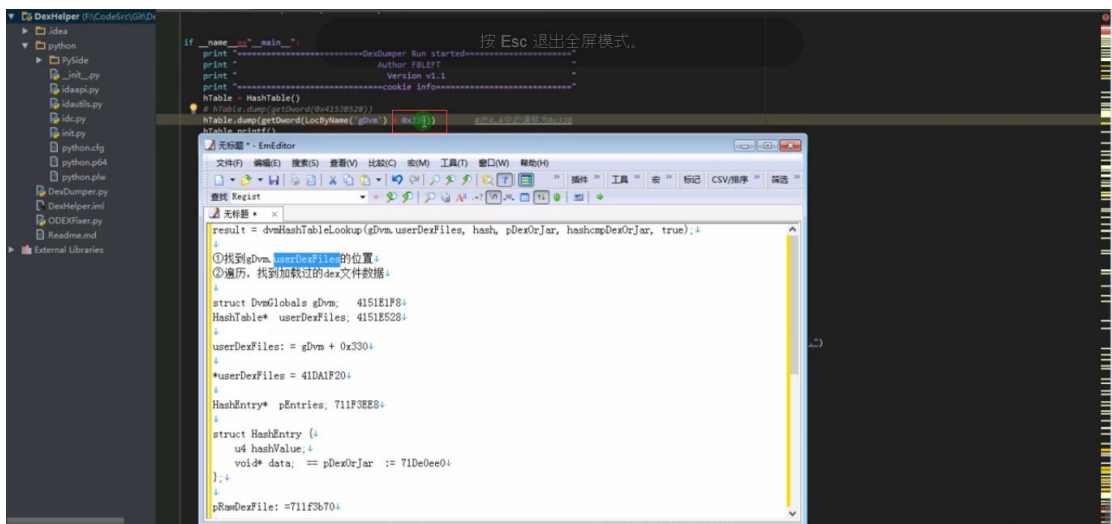
        self.OptHeader.dump(self.pOptHeader)
        self.dexHeader.dump(self.pHeader)

        self.baseAddr = self.pHeader
        baseAddr = self.baseAddr
        self.pStringIds = self.dexHeader.stringIdsOff + self.pHeader
        self.pTypeIds = self.dexHeader.typeIdsOff + self.pHeader
        self.pFieldIds = self.dexHeader.fieldIdsOff + self.pHeader
        self.pMethodIds = self.dexHeader.methodIdsOff + self.pHeader
        self.pProtoIds = self.dexHeader.protoIdsOff + self.pHeader
        self.pClassDefs = self.dexHeader.classDefsOff + self.pHeader
        self.pLinkData = self.dexHeader.linkOff + self.pHeader

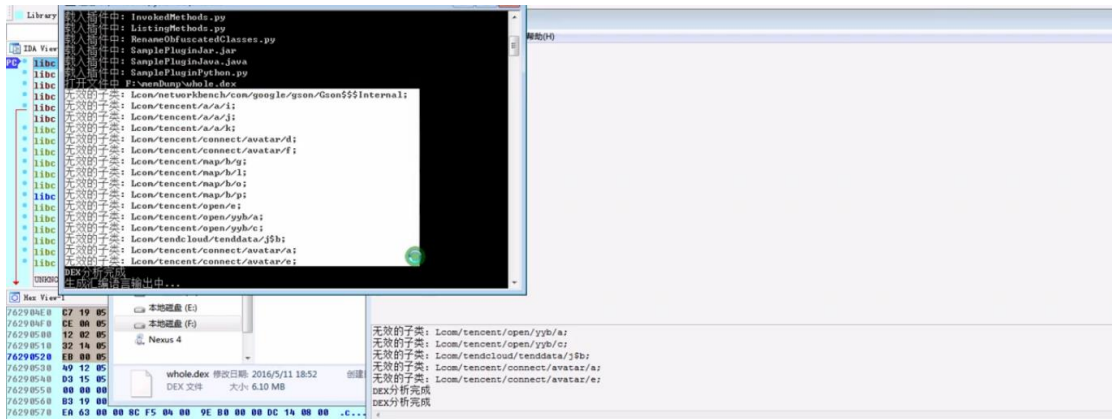
    def dump(self, addr):
        global baseAddr
        num_class_def = self.dexHeader.classDefSize
        total_point = self.dexHeader.dataOff + self.dexHeader.dataSize
        start = self.dexHeader.dataOff
        end = total_point
        while total_point < end:
            total_point += 1
            print "num class def", num_class_def
            logger.write("num class def %d\n" % num_class_def)
            for i in range(num_class_def):
                print "for class", i, "total", num_class_def
                logger.write("for class %d total %d\n" % (i, num_class_def))
                classDef = DexClassDef()
                classDef.dump(self.pClassDefs + i)
                descriptor = dexClassDescriptor(self, classDef)
                need_extra = False
                need_pass = False
                tap = lastMethod(descriptor)
                # If descriptor is not a method or classDef.classDataOff == 0, skip class invalid... maybe it can't be used...
                # need_pass = True
                # print "des", tap, "is passed"
                if classDef.classDataOff == 0:
                    need_pass = True
                    print "des", tap, "is passed"
                    logger.write("des %d is passed\n" % tap)
                else:
                    print "des", tap
                    logger.write("des %d\n" % tap)
                    if classDef.classDataOff < start or classDef.classDataOff > end:
                        need_extra = True
                        classData = ClassDataItem()
                        classData.dump(int(self.baseAddr - classDef.classDataOff))
                        if classData.direct_methods_size:
                            for j in range(classData.direct_methods_size):
                                method = classData.direct_methods[j]
                                if method.code_off == 0:
                                    continue
                                if method.access_flags & 0x100:
                                    # native func or ...
                                    need_extra = True
                                    method.code_off = 0
                                    continue
                                if method.code_off < start or method.code_off > end:
                                    need_extra = True
                                    codeItem = CodeItem()
                                    codeItem.dump(int(self.baseAddr - method.code_off))

if __name__ == '__main__':
    print "=====DexDumper Run started=====
    print "
    print "
    print "
    print "
    print "=====cookie info=====
    hTable = HashTable()
    hTable.dump(getWord(0x4130328))
    hTable.dump(getWord(LockyName("gDvm") + 0x330))
    hTable.print()
    print "=====cookie end=====
    # print "EX BY 8 hTable.dump(LockyName("gDvm")
    isel = AskLong(8, "please select a cookie to dump:")
    if isinstance(isel, int):
        pDexFile, pNewMap = hTable.getDefAddr(isel)
        # DexDvm - number online (cookies value)
        print "=====dump start=====
        print "=====dump DexFile=====
        dump_dir = "F:\gDvm\
        dexFile = DexFile()
        dexFile.dump(pDexFile)
        dexFile.dexHeader.print()
        dexFile.cpytofile(dump_dir)
        print "=====dump DexFile finished=====
        if dexFile.pOptHeader != pNewMap and dexFile.pHeader != pNewMap:
            print "=====dump DexFile mapping=====
            isDumpMem = AskYN(0, "extra message has been found, do you want to dump the memMapping? This may be help in some case.")
            dump_mem = "Y/n/memDump/"
            if pNewMap != 0 and isDumpMem == 1:
                dexMem = DexFile()
                dexMem.dumpHeader(pNewMap)
                dexMem.dexHeader.print()
                dexMem.cpytofile(dump_mem)
            print "=====dump MemMapping finished=====
            print "=====dump end=====
            print "=====DexDumper Run finished=====
```

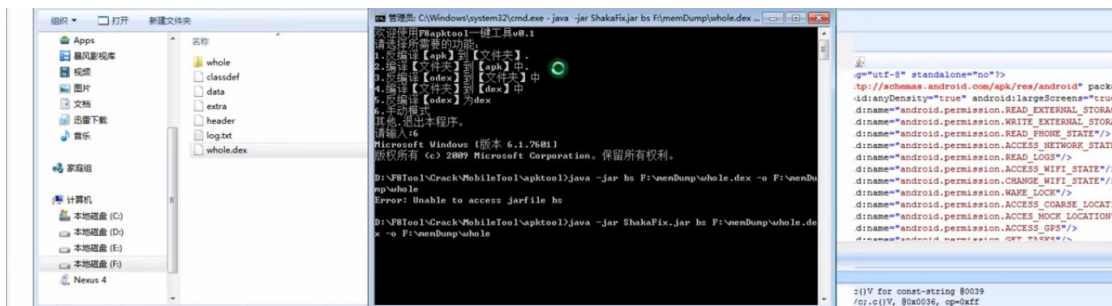
注意要修改 gDvm 地址数据



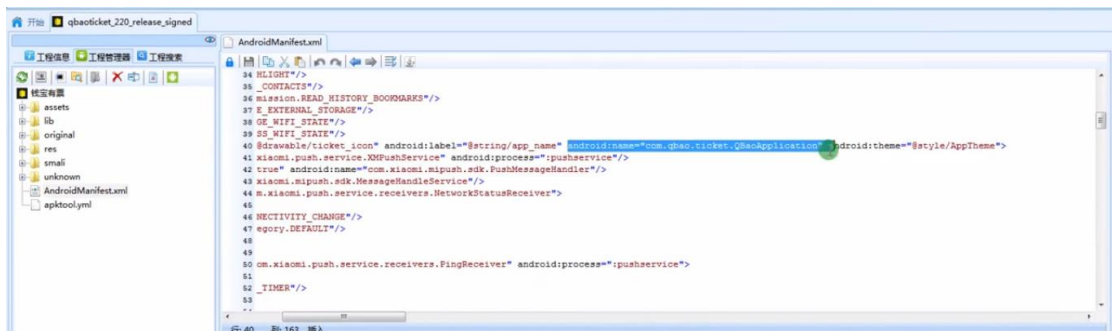
将原始 dex 文件 dump 下来后使用 jeb 打开



先还原为 smali 代码然后替换原项目中的 smali 文件



然后删除 AndroidManifest.xml 中的 android:name 字段



重新编译安装，运行后异常退出，原因是没有找到原始的 Application，解决方法：将

android:name 字段保留

