梆梆脱壳方法

Via WooYun知识库 by Michael23

0x00 前言

前段时间遇到一个app，解包后发现了libDexHekper.so，搜索了一下发现是梆梆的壳，于是花了一段时间研究了一下怎么脱。这篇博客文章在dexhunter的基础上做了修改。

0x01 dexhunter的适配

加固发展到今天，几乎很难通过动态调试的传统方式把壳脱下来了。由于android的开源特性，现在更好的脱壳办法就是修改dvm（或art）虚拟机 ，这种方法最好的代表就是dexhunter。但是dexhunter并不能直接用来脱梆梆的壳了，梆梆做了一些监测，监测到dexhunter后会闪退。分析后发现主要监测了一下两个点：dexname和fopen／fwrite函数。

1. dexname

在使用dexhunter之前，首先要将dexname这个文件push到android设备的/data/目录下，而梆梆就监测了/data/目录下是否有dexname这个文件，有的话就直接闪退。这个还是比较好绕过的，把这个dexname改成其它的名字就可以了，dexhunter里面的读取该文件的代码都做相应的修改即可。

2. fopen／fwrite函数

dexhunter中使用了大量的fopen／fwrite函数，将内存中的dex的文件的各部分写出来。但是梆梆对于这些函数进行了hook，根本就无法进行调用，一调用就直接退出。这种监测的绕过方法dexhunter的wiki中也有说明：If the "fwrite" and other libc functions fail, maybe these functions are hooked by hardening sevices. As a result, you cannot dump the memory via them. You can bypass this limitation by calling relevant system calls directly.就是你用别的功能类似的函数替代fopen和fwrite函数就行了。（这个地方折腾了好久，之前试过用log把内存中的数据打出来，看来用一个东西之前还是要多看wiki）。这里我使用了open和write函数。

0x02 代码还原

梆梆使用了dvmDexFileOpenPartial函数加载dex文件,在这个函数里把dex文件dump出来即可。

```
int dvmDexFileOpenPartial(const void* addr, int len, DvmD
{
    DvmDex* pDvmDex;
    DexFile* pDexFile;
    int parseFlags = kDexParseDefault;
    int result = -1;

    int fd=open(path,O_CREAT|O_RDWR,0666);
    if(fd==-1)
        ALOGI("open file falied");
    else{
        write(fd,addr,len);
        close(fd);
        }
    pDexFile = dexFileParse((u1*)addr, len, parseFlags);
    if (pDexFile == NULL) {
        ALOGE("DEX parse failed");
        goto bail;
    }
```

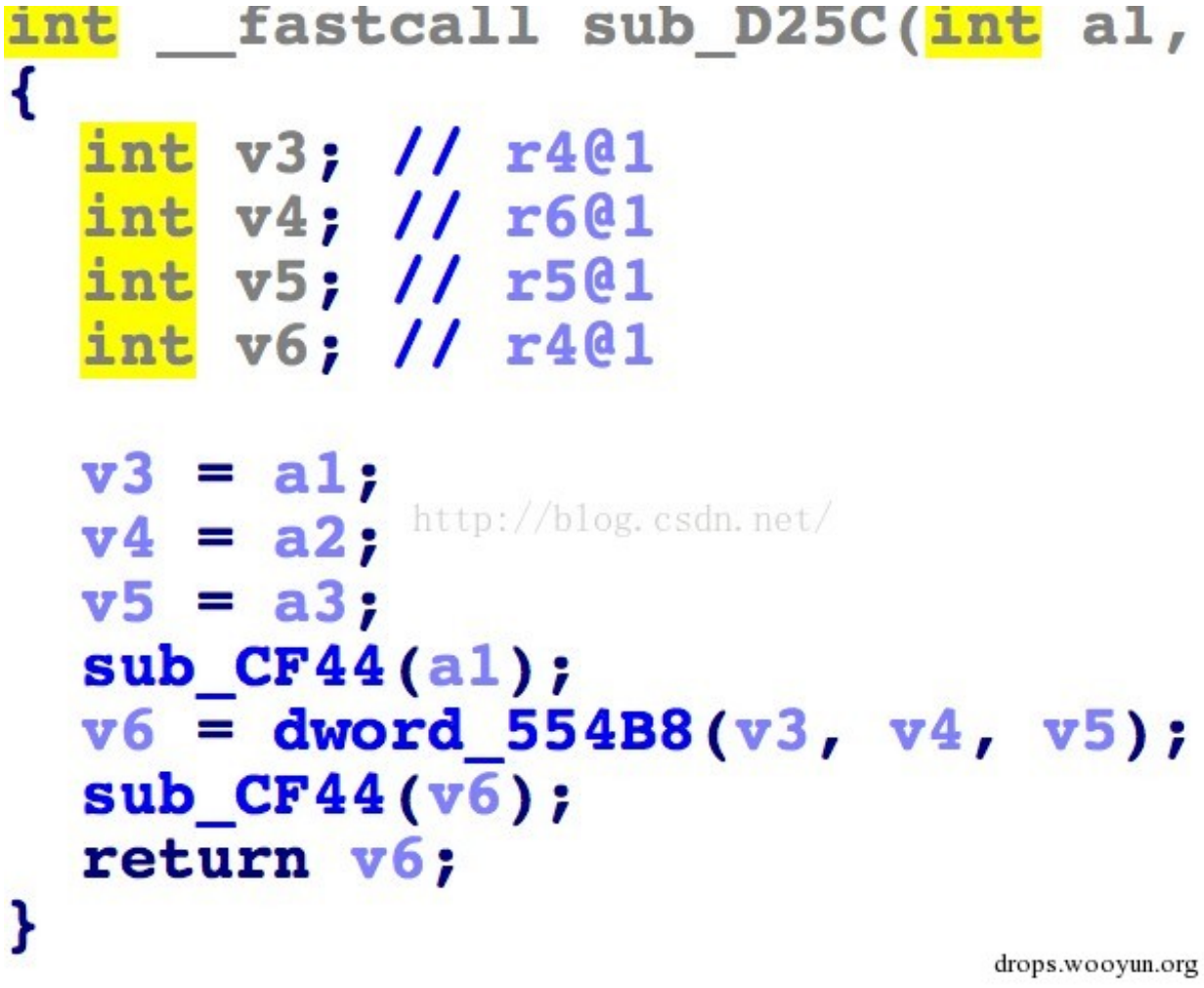

dump出来的dex经过dex2jar的转化后，放到jd-jui里发现很多代码被抽取掉了，如下图：

```
import android.os.Handler;

public final class g extends Handler
{
  public g(BaseDialog paramBaseDialog)
  {
  }

  public final void handleMessage(Message paramMessage)
  {
  }
}
```


看一下libDexHelper.so，发现其使用了o-llvm进行混淆，分析起来简直麻烦，不过还是找到了其hook libdvm.so的片段。

```
v11 = getpid();
sprintf(&v163, "/proc/%d/maps", v11);
v12 = fopen(&v163, "r");
if ( v12 )
{
  while ( 1 )
  {
    v13 = *((_WORD *)v12 + 6) & 0x20;
    if ( *((_WORD *)v12 + 6) & 0x20 )
      break;
    if ( fgets(&v187, 255, v12) )
    {
      if ( strstr(&v187, ".so") )
      {
        memset(&v209, v13, 0x100u);
        v16 = v13;
        if ( sscanf(&v187, "%08x-%08x %s %s %s %s\t%s", &v16, &v17,
        {
          if ( !strcmp(&v209, "/system/lib/libdvm.so") )
            break;
        }
      }
    }
  }
  fclose(v12);
  v2 = v16;
  if ( v16 )
    sub_1AC5E(&v10[v16 + 1], (int)sub_D25C, (int)&dword_554B8);
```


其中sub_1AC5E是hook的承载函数，sub_D25C替代了dvmResolveClass函数。这个函数首先将抽取掉的代码还原，然后调用dvmResolveClass函数执行代码，执行完代码后又将代码抽取掉。这个样子dexhunter是不能获取到原代码的，但是可以在dvmResolveClass函数里进行些修改，dump出代码。



在dvmResolveClass中进行修改的代码片段如下（这里以directMethod为例）：

```
for(int i=0;i<resClass->directMethodCount;i++)
{
        Method *method = &(resClass->directMethods[i]);
        const DexCode *pCode=dvmGetMethodCode(method);
        if(mprotect((void *)PAGE_ALIGN((int)pCode->insns),PAGE_SIZE,PROT_READ)==-1)
{               ALOGI("BANGBANG change code right failed");
                goto out;
        }
        ALOGI("BANGBANG  method name is %s,length is %d",method->name,dvmGetMethodInsnsSize(method));
        int fd=open(filename,O_CREAT|O_RDWR,0666);
        if(fd==-1)
        {
                ALOGI("BANGBANG open file failed,%s",strerror(errno));
                goto out;
}
        write(fd,pCode->insns,2*dvmGetMethodInsnsSize(method));
        close(fd);
}
```

将dump出来的代码写到dex对应的地方即可。（当然也可以直接在里面写,自动化的完成）。

还有一个问题就是dvmDefineClassNative去加载所有的类，这样才能还原所有的代码。在这个过程中，可能有一些类是加载不成功的（如okhttp之类的）。经过调研后，发现这是mutildex的缘故，有兴趣的可以研究一下mutildex的实现原理，但是更好的办法是用5.0以上的系统去做脱壳，这样就不存在mutlidex的问题了，但是同时要多研究一下art虚拟机。