基于Frida框架打造Art模式下的脱壳工具（OpenMemory）的原理分析

原创

Fly20141201 于 2018-07-08 12:11:59 发布 8729 收藏 9
分类专栏： Android逆向学习 Android系统安全和逆向分析研究

本文链接：https://blog.csdn.net/QQ1084283172/article/details/80956614

本文博客地址：https://blog.csdn.net/QQ1084283172/article/details/80956614

作者dstmath在看雪论坛公布一个Android的art模式下基于frida框架的脱壳脚本，详情见文章《基于frida的脱壳工具》的介绍，作者dstmath公布的frida脚本代码见github地址:https://github.com/dstmath/frida-unpack,脱壳原理比较简单，和我前面的博客《ART模式下基于Xposed Hook开发脱壳工具》中提到的作者的脱壳的思路是一致，这个脱壳思路也比较常见，在看雪论坛有多位楼主提到了这个基于Android的art模式下的OpenMemory拦截进行内存dump的脱壳思路，其实很多Xposed框架的Hook思路都可以移植到基于Frida框架的Hook上，顺便提一句，作者dstmath的简书博客写的不错。

具体参考的其他的文章：

《乐固libshella 2.10.1分析笔记》作者的帖子写的很不错。

《Android第二代加固（support 4.4-8.1）》

《修改源码dump某加固保的dex》

《legu 2.10.7.1 脱壳简明过程》

下面简要的说下，作者dstmath提供的frida脱壳脚本需要注意的几个点：

0x2 原理说明

利用frida hook libart.so中的OpenMemory方法，拿到内存中dex的地址，计算出dex文件的大小，从内存中将dex导出。

ps：查看OpenMemory的导出名称，可以将手机中的libart.so通过adb pull命令导出到电脑，然后利用：

`nm libart.so |grep OpenMemory`命令来查看到出名。

0x3 脚本用法

- 在手机上启动frida server端
- 执行脱壳脚本

```
./inject.sh 要脱壳的应用的包名 OpenMemory.js
```

- 脱壳后的dex保存在`/data/data/应用包名/`目录下

0x4 脚本测试环境

此脚本在以下环境测试通过

- android os: 7.1.2 32bit (64位可能要改OpenMemory的签名)
- legu: libshella-2.8.so
- 360: libjiagu.so

0x5 参考链接

- frida

0x06 python脚本支持

`python frida_unpack.py 应用包名`

源码见github链接：https://github.com/dstmath/frida-unpack

由于随着Android系统版本的升级，Art模式下libart.so库中OpenMemory函数的传入参数会有所不同，因此libart.so库文件中OpenMemory函数在编译时的名称粉粹也会不同，不同版本的Android系统的libart.so库文件导出函数OpenMemory的系统符号也会有差异，可用通过adb pull导出相应版本的Android系统的libart.so库文件，用IDA打开该libart.so库文件查看导出表中OpenMemory函数的导出系统符号或者使用 nm 等能查看ELF文件的导出系统符号的工具，查看libart.so库文件中OpenMemory函数的导出系统符号，相应的修改frida-unpack中的frida脱壳脚本文件frida_unpack.py和OpenMemory.js中OpenMemory函数的导出系统符号。

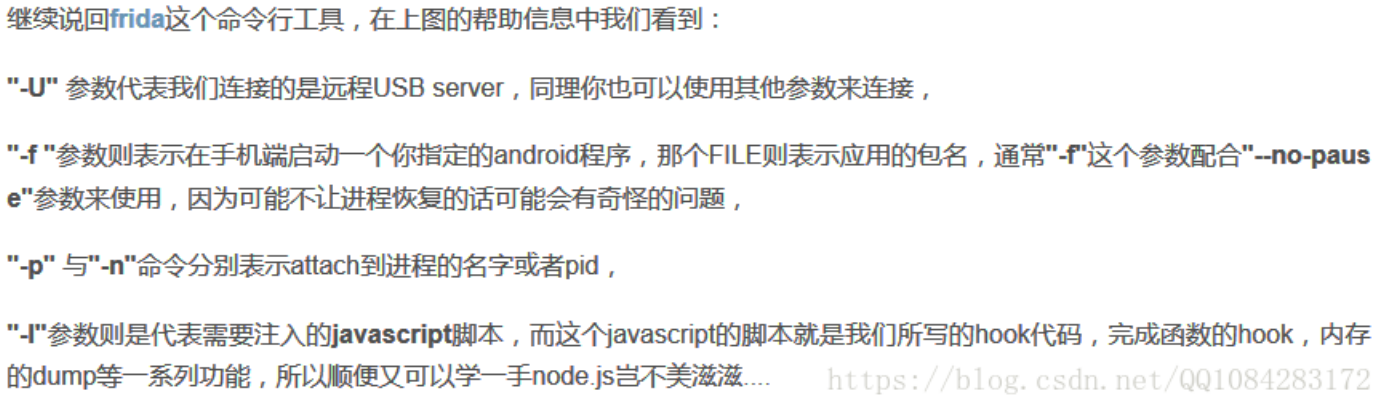
frida的命令行帮助：

```
>frida -h
Usage: frida [options] target

Options:
  --version             show program's version number and exit
  -h, --help            show this help message and exit
  -D ID, --device=ID    connect to device with the given ID
  -U, --usb             connect to USB device
  -R, --remote          connect to remote frida-server
  -H HOST, --host=HOST  connect to remote frida-server on HOST
  -f FILE, --file=FILE  spawn FILE
  -n NAME, --attach-name=NAME
                        attach to NAME
  -p PID, --attach-pid=PID
                        attach to PID
  --debug               enable the Node.js compatible script debugger
  --enable-jit          enable JIT
  -l SCRIPT, --load=SCRIPT
                        load SCRIPT
  -c CODESHARE_URI, --codeshare=CODESHARE_URI
                        load CODESHARE_URI
  -e CODE, --eval=CODE  evaluate CODE
  -q                    quiet mode (no prompt) and quit after -l and -e
  --no-pause            automatically start main thread after startup
  -o LOGFILE, --output=LOGFILE
                        output to log file
```

1.直接执行frida命令加载脱壳脚本OpenMemory.js文件进行Hook OpenMemory函数的操作， 内存dump dex文件。

```
frida -U -f $1 -l $2 --no-pause
# $1--为被脱壳的Android应用的包名
# $2--为被frida加载的脚本文件OpenMemory.js
```



被frida加载的脚本文件OpenMemory.js的代码如下（代码中使用的被脱壳的Android应用包名需要做相应的修改）：

```
'use strict';
/**
 * 此脚本在以下环境测试通过
 * android os: 7.1.2 32bit  (64位可能要改OpenMemory的签名)
 * legu: libshella-2.8.so
 * 360:libjiagu.so
 */
Interceptor.attach(Module.findExportByName("libart.so", "_ZN3art7DexFile10OpenMemoryEPKhjRKNSt3__112basic_s
    onEnter: function (args) {

        //dex起始位置
        var begin = args[1]
        //打印magic
        console.log("magic : " + Memory.readUtf8String(begin))
        //dex fileSize 地址
        var address = parseInt(begin,16) + 0x20
        //dex 大小
        var dex_size = Memory.readInt(ptr(address))

        console.log("dex_size :" + dex_size)
        //dump dex 到/data/data/pkg/目录下
        var file = new File("/data/data/xxx.xxx.xxx/" + dex_size + ".dex", "wb")
        file.write(Memory.readByteArray(begin, dex_size))
        file.flush()
        file.close()
    },
    onLeave: function (retval) {
        if (retval.toInt32() > 0) {
            /* do something */
        }
    }
});
```

2.使用frida框架的远程控制端（客户端）提供的python语言的控制接口调用frida程序执行Hook OpenMemory函数的操作 ， 内存dump dex文件，python的脱壳脚本frida_unpack.py文件的代码如下：

```
#-*- coding:utf-8 -*-
# coding=utf-8
import frida
import sys

def on_message(message, data):
    base = message['payload']['base']
    size = int(message['payload']['size'])
    print hex(base),size
    # print session
    # dex_bytes = session.read_bytes(base, size)
    # f = open("1.dex","wb")
    # f.write(dex_bytes)
    # f.close()

package = sys.argv[1]
print "dex 导出目录为: /data/data/%s"%(package)
device = frida.get_usb_device()
pid = device.spawn(package)
session = device.attach(pid)
src = """
```

```
Interceptor.attach(Module.findExportByName("libart.so", "_ZN3art7DexFile10OpenMemoryEPKhjRKNSt3__112basic_s
    onEnter: function (args) {

        var begin = args[1]

        console.log("magic : " + Memory.readUtf8String(begin))

        var address = parseInt(begin,16) + 0x20

        var dex_size = Memory.readInt(ptr(address))

        console.log("dex_size :" + dex_size)

        var file = new File("/data/data/%s/" + dex_size + ".dex", "wb")
        file.write(Memory.readByteArray(begin, dex_size))
        file.flush()
        file.close()

        var send_data = {}
        send_data.base = parseInt(begin,16)
        send_data.size = dex_size
        send(send_data)
    },
    onLeave: function (retval) {
        if (retval.toInt32() > 0) {
        }
    }
});
"""%(package)

script = session.create_script(src)

script.on("message" , on_message)

script.load()
device.resume(pid)
sys.stdin.read()
```

附上《修改源码dump某加固保的dex》这篇帖子的内容做个备份，本来打算在这篇博客中学习一下art模式下dex文件的加载流程，想想还是后面再研究吧，想看该帖子的内容建议去作者的原链接处看，谢谢原作者。

一般安卓加载代码都是通过classloader来装取本地代码到内存中去的。

classloader有两种加载方式：

1.通过路径寻找本地代码，然后载入内存:

基本的源码顺序是 BaseDexClassLoader->DexPathList->makeDexElements->loadDexFile->Dexfile.loadDex

最后DexFile中有一个native方法 **OpenDexFileNative** 对应了 **DexFile_openDexFileNative** 方法。其中的OpenDexFileFormOat便是最主要的方法了。

当然这里我们就跳过，需要了解classloader的机制可以自行去看xref，因为这里是路径加载的方式，一般加固不会选择这里的方法去加载的。

2.直接映射到内存中去：

在4.x版本的DexFile.java有

```
native private static int openDexFile(byte[] fileContents);
```

这个函数在5.x中的java方法已经摒弃，在native层中依旧存在于/art/runtime/dex_file.cc中。

其方法为**OpenMemory**函数。我们只需要在此从内存中扣出文件就好了。

```
const DexFile* DexFile::OpenMemory(const byte* base,
                                   size_t size,
                                   const std::string& location,
                                   uint32_t location_checksum,
                                   MemMap* mem_map,
                                   const OatFile* oat_file,
                                   std::string* error_msg) {
  CHECK_ALIGNED(base, 4);  // various dex file structures must be word aligned
  std::unique_ptr<DexFile> dex_file(
      new DexFile(base, size, location, location_checksum, mem_map, oat_file));
  if (!dex_file->Init(error_msg)) {
    return nullptr;
  } else {
    __android_log_print(ANDROID_LOG_DEBUG,"chason 's DexFile::OpenMemory",
                                "size is:%zu,location is:%s", size, location.c_str());
    if (!strcmp(location.c_str(),"/data/data/com.autohome.mycar/.jiagu/classes.dex"))
    {

        int fd = open("/data/data/com.autohome.mycar/classes.dex",O_CREAT|O_EXCL|O_WRONLY,S_IRWXU);
         __android_log_print(ANDROID_LOG_DEBUG,"chason's copy is starting!","hello");
        if (fd>0)
          write(fd,base,size);
        else
          __android_log_print(ANDROID_LOG_DEBUG,"chason's copy is failed!","codeis:%d",fd);
        close(fd);
    }
    return dex_file.release();
  }
}
```

最后在路径里找到dex就ok了。。。

PS：先看logcat里的location字符串是啥，然后再去写if判断字符串是否是这个~

分享apk以及脱完的dex。。。希望请大神们指点思路O(∩_∩)O

好了，后面继续学习~