抽取样本java实验报告_一个自定义classloader的函数抽取壳样本

原创

weixin_39610353 于 2021-02-26 02:01:33 发布 83 收藏
文章标签： 抽取样本java实验报告
版权声明：本文为博主原创文章，遵循 CC 4.0 BY-SA 版权协议，转载请附上原文出处链接和本声明。
本文链接：https://blog.csdn.net/weixin_39610353/article/details/114712869
版权
原标题：一个自定义classloader的函数抽取壳样本

本文为看雪论坛文章

看雪论坛作者ID：lemn

本文为 看雪安卓高研2w班(7月班)优秀学员作品。

下面先让我们来看看学员的学习心得吧！

学员感想

本题目出自2W班7月第三题。

题目要求：基于frida实现的fart的一个版本是通过对ClassLink类中的LoadMethod函数进行hook实现对函数粒度的脱壳的。请编写基于xposed实现的fart版本插件，能够实现对

函数粒度的脱壳。

根据题目要求，可以知道，我们通过hook LoadMethod即可得到DexFile进而得到base和size，即可dump出dex。如果有函数抽取，即遍历所有的类得到codeitem后还原即可。

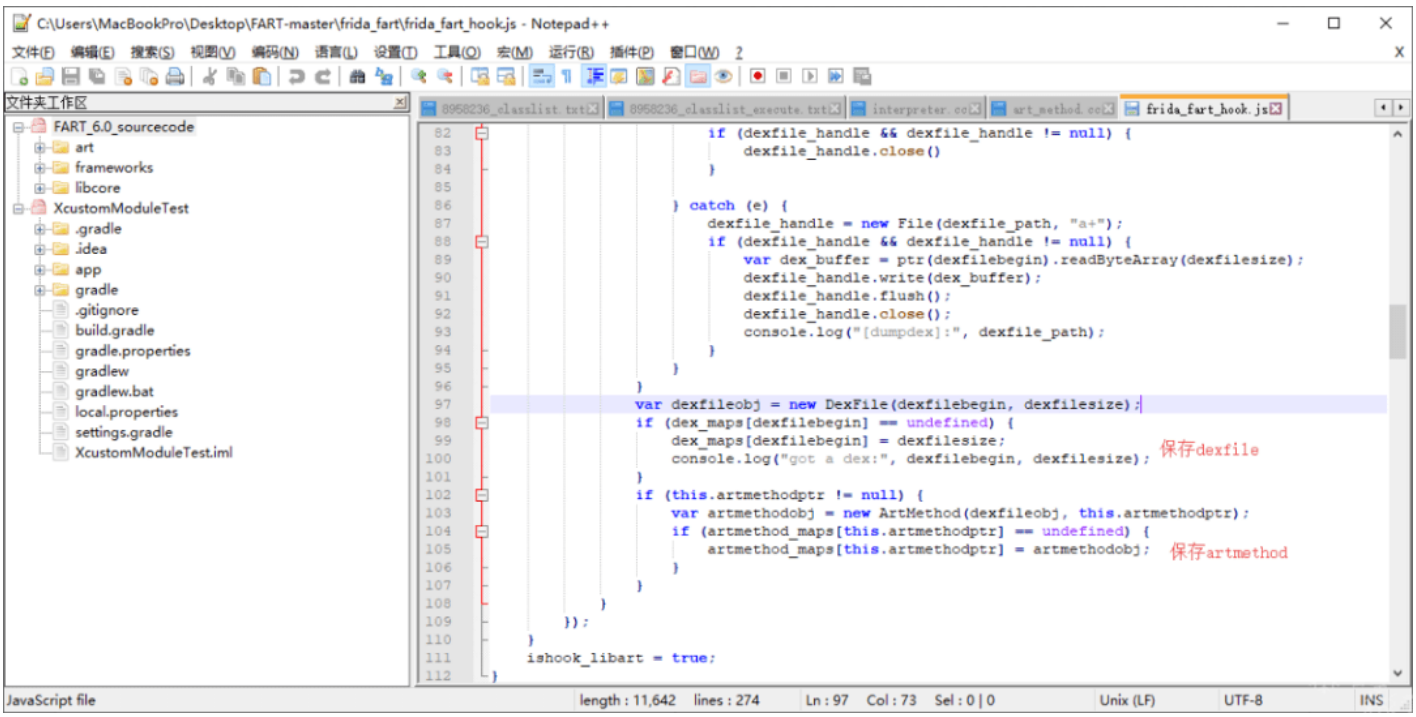
本次实践分为两个部分，一个即是原始思路，照葫芦画瓢参照FART的脱壳思路去编写代码，遇到困难暂时没法解决的时候，转变思路二，改用FART配合xposed去实现。

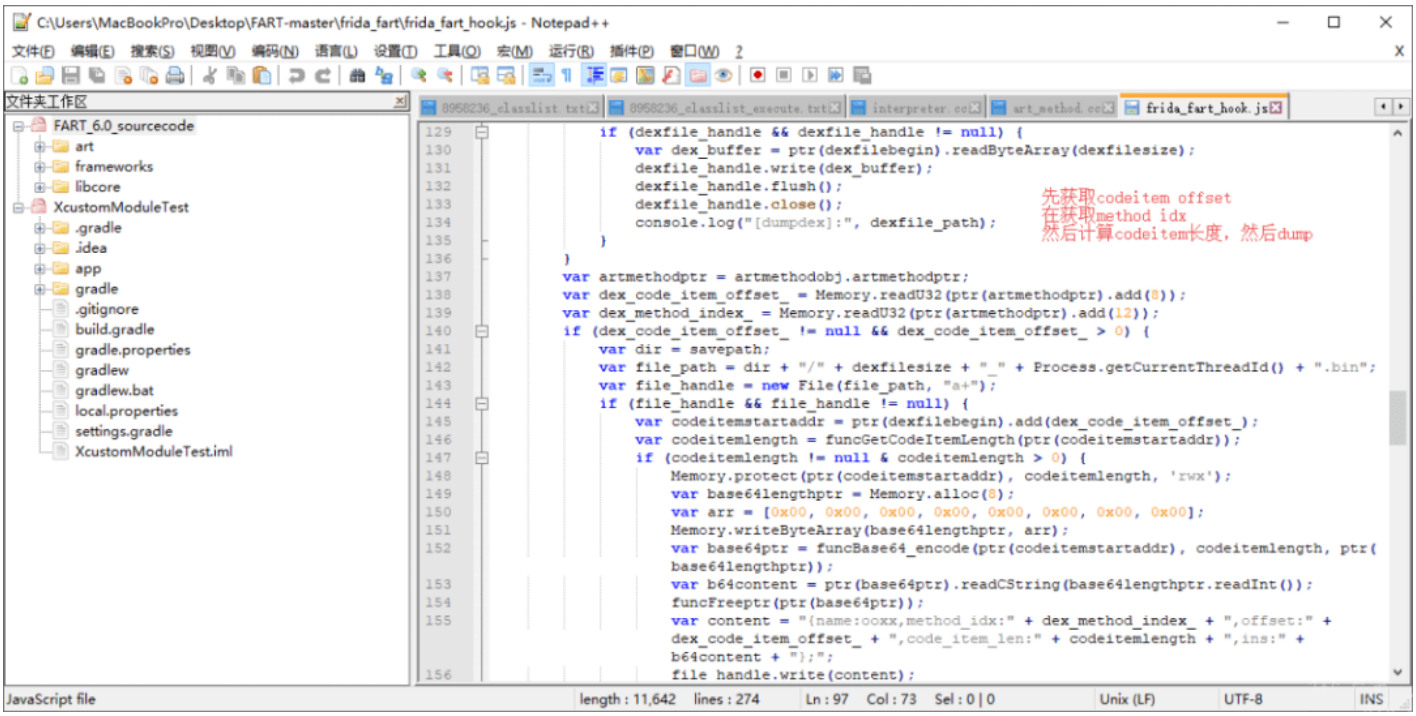
ps. 题目附件请点击“阅读原文”下载。

解题过程

首先我们可以分析frida-fart是如何实现demp dex的。


```
if (addrLoadMethod != null) {
    Interceptor.attach(addrLoadMethod, {
        onEnter: function (args) {
            this.dexfileptr = args[1];
            this.artmethodptr = args[4];
        },
        onLeave: function (retval) {
            var dexfilebegin = null;
            var dexfilesize = null;
            if (this.dexfileptr != null) {
                dexfilebegin = Memory.readPointer(ptr(this.dexfileptr).add(Process.pointerSize
                 * 1));
                dexfilesize = Memory.readU32(ptr(this.dexfileptr).add(Process.pointerSize * 2
                ));
                var dexfile_path = savepath + "/" + dexfilesize + "_loadMethod.dex";
                var dexfile_handle = null;
                try {
                    dexfile_handle = new File(dexfile_path, "r");
                    if (dexfile_handle && dexfile_handle != null) {
                        dexfile_handle.close()
                    }

                } catch (e) {
                    dexfile_handle = new File(dexfile_path, "a+");
                    if (dexfile_handle && dexfile_handle != null) {
                        var dex buffer = ptr(dexfilebegin).readByteArray(dexfilesize);
```





粗略分析一下就是：

1. 拿到dexfile 就可以拿到base size

2. 拿到artmethod 就可以拿到codeitem offset和method idx

3. 计算codeitem的长度

4. dump出来

那么我们在so层也可以画葫芦试试。测试安卓版本为8.1，部分代码如下：

void*pVoid = old_loadmethod3(thiz, thread, dex_file, it, klass, artmethod);

__android_log_print(5, "hookso", "pVoid ptr:%p", pVoid);

获取base和size

constDexHeader *base = dex_file.pHeader;

size_tsize = dex_file.pHeader->fileSize;

获取code item offset和method idx

uint32_tcodeItemOffset = artmethod->dex_code_item_offset_;

uint32_tidx = artmethod->dex_method_index_;

hook prettymethod方法 主动调用获得方法名

conststd:: string& string= prettyMethodFunction(artmethod, artmethod,

true);

通过偏移可以拿到codeitem

longcodeItemAddr = (long) base + codeItemOffset;

CodeItem *codeItem = (CodeItem *) codeItemAddr;

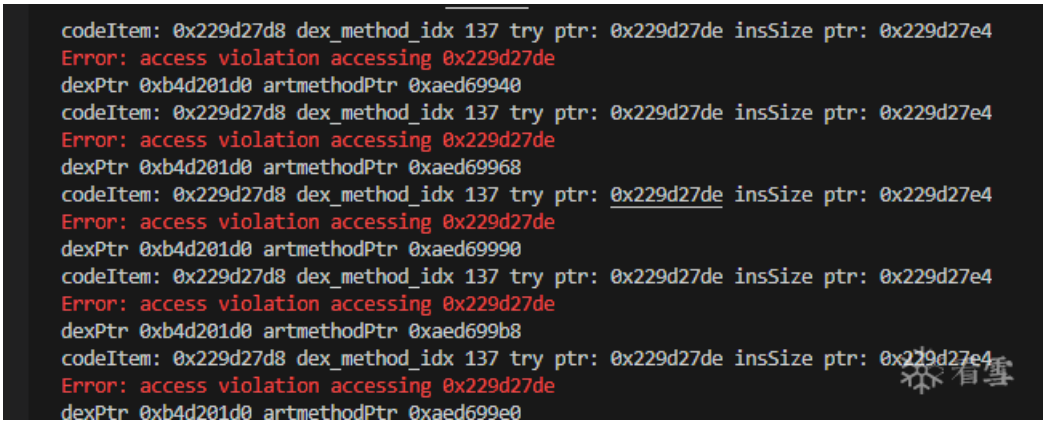
这部分代码可以直接dump dex出来

```
intpid = getpid;

chardexFilePath[ 100] = { 0};

sprintf(dexFilePath, "/sdcard/xxxxx/%p %d LoadMethod.dex", base, size);

mkdir( "/sdcard/xxxxx", 0777);

intfd = open(dexFilePath, O_CREAT | O_RDWR, 666);

if(fd > 0) {

ssize_ti = write(fd, base, size);

if(i > 0) {

close(fd);

}

}

...
```

上述看起来一步一步的确是可以拿得到codeitem，然后进一步拿得到ins的。

但是有一个问题我一直没法解决，就是某些方法会报access violation 异常。我通过搜索发现这个是底层发出来的异常，软件层貌似没法catch。

通过使用frida-fart我发现frida版本的也有这样的问题，但是frida这边可以通过try catch捕捉，让程序继续行走。



因为暂时无法解决这个问题，所以我放弃这个方向。尝试用fart进行dump。

这里偷懒，就直接用寒冰大佬的xp+fart rom继续dump，就没有自己编译源码了。这里环境android6.0。

那么这时候思路就转变了，通过hook loadMethod方法可以拿到artmethod，

而根据fart代码，dumpArtMethod这个方法，传入artmethod即可dump。

先执行原来的loadMethod逻辑

```
void*pVoid = old_loadmethod3(thiz, thread, dex_file, it, klass, artmethod);

__android_log_print( 5, "hookso", "pVoid ptr:%p", pVoid);
```

然后通过so层hook dumpArtmethod函数，并将参数传入

```
try{
dumpArtMethodFunction(artmethod);
} catch(...){
}
```

上述通过so层调用FART的dumpArtMethodFunction方法，可以dump出dex，以及classlist。这是FART自带的功能。

而在java层，我们需要遍历所有类。首先hook掉DexClassLoader和PathClassLoader的构造函数，并将classloader存起来。

因为是自定义的Xposed，而所以名字为XcustomBridge。其实这里就是Xposed，仅供参考，不可照抄：

```
XcustomBridge.hookAllConstructors(DexClassLoader.class, newXC_MethodHook {
@Override
protectedvoidafterHookedMethod(MethodHookParam param)throwsThrowable{
super.afterHookedMethod(param);
finalClassLoader classLoader = (ClassLoader) param.thisObject;
XcustomBridge.log( "DexClassLoader:"+ classLoader.toString);
mClassLoaders.add(classLoader);
}
});
XcustomBridge.hookAllConstructors(PathClassLoader.class, newXC_MethodHook {
@Override
protectedvoidafterHookedMethod(MethodHookParam param)throwsThrowable{
super.afterHookedMethod(param);
finalClassLoader classLoader = (ClassLoader) param.thisObject;
XcustomBridge.log( "PathClassLoader:"+ classLoader.toString);
mClassLoaders.add(classLoader);
}
});
```

紧接着加载我们的so，并遍历所有的classloader，执行loadClass操作：

```
XcustomHelpers.findAndHookMethod(Application.class, "attach", Context.class, newXC_MethodHook {
@Override
protectedvoidafterHookedMethod(MethodHookParam param)throwsThrowable{
super.afterHookedMethod(param);
```

```
XcustomBridge.log( "attach after");

mContext = (Context) param.args[ 0];

XcustomHelpers.callMethod(Runtime.getRuntime, "doLoad", "/system/lib/libnative-lib.so",
mContext.getClassLoader);

newThread( newRunnable {

@Override

publicvoidrun{

try{

Thread.sleep( 30* 1000);

} catch(InterruptedException e) {

e.printStackTrace;

Log.e( "hook1", Log.getStackTraceString(e));

}

for( inti = 0; i < mClassLoaders.size; i++) {

ClassLoader classLoader = mClassLoaders.get(i);

TestClassloader(classLoader);

}

fart;

}

}).start;

}

});
```

将dump下来后bin文件批量恢复后，即可查看到，函数已经恢复了。

```
import com.sup.superb.video.d.g;
import com.sup.superb.video.e;
import com.sup.superb.video.e.k;

public abstract class AbsVideoPresenter implements LifecycleObserver, WeakHandler.IHandler, a
    public static ChangeQuickRedirect a;
    protected Handler b = new WeakHandler(this);
    protected boolean c = false;
    protected Activity d;
    protected boolean e;
    boolean f;
    k g;
    int h = 2;
    private LifecycleOwner i;

    public AbsVideoPresenter(Activity activity, LifecycleOwner lifecycleOwner, k kVar) {
        lifecycleOwner.getLifecycle().addObserver(this);
        this.i = lifecycleOwner;
        this.d = activity;
        this.g = kVar;
    }

    public void a(Activity activity, VideoModel videoModel) {
        if (PatchProxy.isSupport(new Object[]{activity, videoModel}, this, a, false, 7206, ne
            PatchProxy.accessDispatch(new Object[]{activity, videoModel}, this, a, false, 720
        } else if (this.c && this.e) {
            g.a().a((Context) activity, videoModel);
        }
```

但是如果点多几个函数查看，会发现部分函数并没有复原：

```
12 public class HostManager {
       private Context context;
       private ConcurrentMap<String, DnsRecord> hostMap;
       private boolean isNeedRefetchOnExpire;
       private String lastBSSID;
       private int lastNet;
       private BroadcastReceiver receiver;
       private ConcurrentSkipListSet<String> resolvingSet;
       private ConcurrentMap<String, DnsRecord> wifiHostMap;

13     static /* synthetic */ ConcurrentMap access$000(HostManager hostMana
       }

13     static /* synthetic */ String access$100(HostManager hostManager) {
       }

13     static /* synthetic */ String access$102(HostManager hostManager, St
       }

13     static /* synthetic */ ConcurrentMap access$200(HostManager hostMana
       }

13     static /* synthetic */ int access$300(HostManager hostManager) {
       }

13     static /* synthetic */ int access$302(HostManager hostManager, int i
       }
```

那么通过回想，我们可以得知，还原的代码其实都是app启动过程中调用过的方法，所以dump下来后，是包含代码的。因此这个壳也是函数抽取壳，而且恢复后不会复原。

而查看FART dump出来的ins文件，发现也缺了很多数据，那么这里可以猜测是某个环节出了问题导致没有dump出来。

然后通过回溯error log可以发现是classloader的锅：

/**

* 01-06 00:31:12.925 12000-12060/com.sup.android.superb W/System.err: java.lang.IllegalArgumentException: Expected receiver of typedalvik.system.BaseDexClassLoader, but got com.bytedance.frameworks.plugin.core.DelegateClassLoader

* 01-06 00:31:12.925 12000-12060/com.sup.android.superb W/System.err: at java.lang.reflect.Field.get(Native Method)

* 01-06 00:31:12.925 12000-12060/com.sup.android.superb W/System.err: at com.l.sevenclasshook.hook.Hook3.getFieldOjbect(Hook3.java:206)

* 01-06 00:31:12.925 12000-12060/com.sup.android.superb W/System.err: at com.l.sevenclasshook.hook.Hook3.TestClassloader(Hook3.java:259)

* 01-06 00:31:12.925 12000-12060/com.sup.android.superb W/System.err: at com.l.sevenclasshook.hook.Hook3.fart(Hook3.java:240)

* 01-06 00:31:12.926 12000-12060/com.sup.android.superb W/System.err: at com.l.sevenclasshook.hook.Hook3 $3$1.run(Hook3.java:95)

* 01-06 00:31:12.926 12000-12060/com.sup.android.superb W/System.err: at java.lang.Thread.run(Thread.java:818)

* 01-06 00:31:12.926 12000-12060/com.sup.android.superb W/System.err: java.lang.NullPointerException: null receiver

* 01-06 00:31:12.926 12000-12060/com.sup.android.superb W/System.err: at java.lang.reflect.Field.get(Native Method)

* 01-06 00:31:12.926 12000-12060/com.sup.android.superb W/System.err: at com.l.sevenclasshook.hook.Hook3.getFieldOjbect(Hook3.java:206)

* 01-06 00:31:12.926 12000-12060/com.sup.android.superb W/System.err: at com.l.sevenclasshook.hook.Hook3.TestClassloader(Hook3.java:260)

* 01-06 00:31:12.926 12000-12060/com.sup.android.superb W/System.err: at com.l.sevenclasshook.hook.Hook3.fart(Hook3.java:240)

* 01-06 00:31:12.926 12000-12060/com.sup.android.superb W/System.err: at com.l.sevenclasshook.hook.Hook3 $3$1.run(Hook3.java:95)

* 01-06 00:31:12.926 12000-12060/com.sup.android.superb W/System.err: at java.lang.Thread.run(Thread.java:818)

那么这里我们知道了这个app有自定义的classloader。

```
port java.lang.reflect.Method;
port java.util.List;

blic class DelegateClassLoader extends ClassLoader {
  private Method findClassMethod = MethodUtils.getAccessibleMethod(ClassLoa
  private Method findLoadedClassMethod = MethodUtils.getAccessibleMethod(C
  private ClassLoader pathClassLoader;

  public DelegateClassLoader(ClassLoader classLoader, ClassLoader classLoa
      super(classLoader);
      this.pathClassLoader = classLoader2;
  }

  /* access modifiers changed from: protected */
  public Class<?> findClass(String str) throws ClassNotFoundException {
      Throwable th;
```

根据提示，我们可以发现这是一个直接继承于classloader的类。

那么来看看通常我们是如何获取classlist的：

```
Field pathList_Field = (Field) getClassField(appClassloader, "dalvik.system.BaseDexClassLoader", "pathList");

ObjectpathList_object = getFieldOjbect( "dalvik.system.BaseDexClassLoader", appClassloader, "pathList");

Object[] ElementsArray = ( Object[]) getFieldOjbect( "dalvik.system.DexPathList", pathList_object,
"dexElements");

Field dexFile_fileField = null;

try{

dexFile_fileField = (Field) getClassField(appClassloader, "dalvik.system.DexPathList$Element", "dexFile");

} catch(Exception e) {

e.printStackTrace;

} catch( Errore) {

e.printStackTrace;

}

.....
```

我们是通过获得BaseDexClassLoader的pathList字段从而进一步往下获取classList的。但是由于我们现在是直接继承于classloader，所以我们要想办法获取classlist。

那么这里就有两种方法可以实现了。

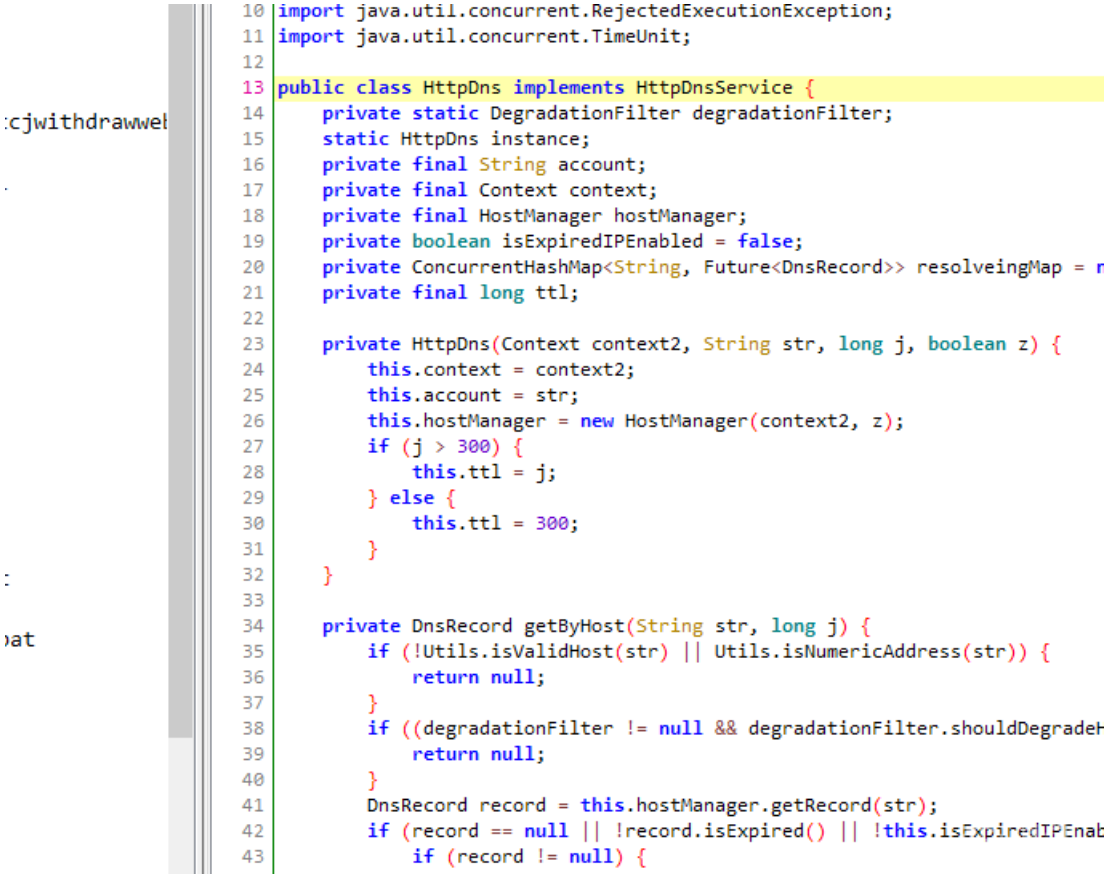
第一个方法， fart使用过程中使dump classlist出来：

```
if(appClassloader instanceofBaseDexClassLoader) {

} elseif(appClassloader instanceofClassLoader) {

List< String> nameList = newArrayList<>;

BufferedReader br = null;

try{

br = newBufferedReader( newInputStreamReader( newFileInputStream(
newFile(Environment.getExternalStorageDirectory + "/8958236_classlist.txt")),

"UTF-8"));

StringlineTxt = null;

while((lineTxt = br.readLine) != null) {

if(!TextUtils.isEmpty(lineTxt)) {

Stringname = lineTxt.replace( "L", "").replaceAll( "/", ".").replace( ";", "");

Log.e( "hook1", "after replace name:"+ name);

nameList.add(name);

}
```

```
}
br.close;
} catch(Exception e) {
Log.e( "hook1", Log.getStackTraceString(e));
}
for( Stringname : nameList) {
try{
Log.e( "Hook1", "==================>loadClass:"+ name);
appClassloader.loadClass(name);
} catch(Exception e) {
Log.e( "Hook1", Log.getStackTraceString(e));
}
}
return;
}
```

因此这里我们直接遍历文件，然后loadClass即可。

接下来我们对dex方法体进行批量恢复后，可以看到函数都恢复了。



```
import java.util.concurrent.RejectedExecutionException;
import java.util.concurrent.TimeUnit;

public class HttpDns implements HttpDnsService {
    private static DegradationFilter degradationFilter;
    static HttpDns instance;
    private final String account;
    private final Context context;
    private final HostManager hostManager;
    private boolean isExpiredIPEnabled = false;
    private ConcurrentHashMap<String, Future<DnsRecord>> resolveingMap = r
    private final long ttl;

    private HttpDns(Context context2, String str, long j, boolean z) {
        this.context = context2;
        this.account = str;
        this.hostManager = new HostManager(context2, z);
        if (j > 300) {
            this.ttl = j;
        } else {
            this.ttl = 300;
        }
    }

    private DnsRecord getByHost(String str, long j) {
        if (!Utils.isValidHost(str) || Utils.isNumericAddress(str)) {
            return null;
        }
        if ((degradationFilter != null && degradationFilter.shouldDegradeH
            return null;
        }
        DnsRecord record = this.hostManager.getRecord(str);
        if (record == null || !record.isExpired() || !this.isExpiredIPEnab
            if (record != null) {
```

少部分没有恢复的函数，是因为这个classloader没有找到类，报class not found。接下来我们可以将所有classloader都跑一遍这个list即可。

第二种方法：其实fart也是解析DexFile从而拿到classlist的。既然有了整个dexFile文件，那么有啥不能解析的呢。

```
if(size == 8958236) {

u4 classDefSize = dex_file.pHeader->classDefsSize;

u4 classDefsOff = dex_file.pHeader->classDefsOff;

__android_log_print( 5, "hookso", "base:%p", ( void*) base);

__android_log_print( 5, "hookso", "classDefSize:%ld classDefsOff:%p", classDefSize,

( void*) classDefsOff);

u4 typeIdsOff = dex_file.pHeader->typeIdsOff;

u4 stringIdsOff = dex_file.pHeader->stringIdsOff;

__android_log_print( 5, "hookso", "typeIdsOff:%p stringIdsOff:%p", ( void*) typeIdsOff,

( void*) stringIdsOff);

inti;

for(i = 0; i < classDefSize; i++) {

longcurrClassAddr = ( long) classDefsOff + i * 32+ ( long) base;

__android_log_print( 5, "hookso", "currClass ptr:%p", ( void*) currClassAddr);

int*idx = ( int*) (currClassAddr);

__android_log_print( 5, "hookso", "currClassIdx:%i", *idx);

inttmpIdx = *idx;

longcurrTypeIdAddr = (( long) typeIdsOff + 4* tmpIdx + ( long) base);

int*currTypeIdx = ( int*) currTypeIdAddr;

__android_log_print( 5, "hookso", "currTypeIdx:%ld", *currTypeIdx);

tmpIdx = *currTypeIdx;

longcurrStringOffAddr = (( long) stringIdsOff + 4* tmpIdx + ( long) base);

int*currStringOff = ( int*) currStringOffAddr;

__android_log_print( 5, "hookso", "currStringOff:%ld", *currStringOff);

tmpIdx = *currStringOff;

longoff = ( long) base+ tmpIdx;

__android_log_print( 5, "hookso", "string off:%p", ( void*) off);

constuint8_t *strPtr = (uint8_t *) off;
```

DecodeUnsignedLeb128(&strPtr);

char*classname = (char*) strPtr;

__android_log_print(5, "hookso", "classname:%s", classname);

}

}

在这里我们可以一层一层的不断获取不断遍历，就可以拿到我们想要的classname：

```
:o: pVoid ptr:0xf4cf6dce
:o: base:0x9c9bbdec
:o: classDefSize:6895   classDefsOff:0x13e3b4
:o: typeIdsOff:0x41a90   stringIdsOff:0x70
:o: currClass ptr:0x9cafa1a0
:o: currClassIdx:96
:o: currTypeIdx:11258
:o: currStringOff:2018841
:o: string off:0x9cba8c05
:o: classname:Landroid/arch/core/R;
:o: currClass ptr:0x9cafa1c0
:o: currClassIdx:97
:o: currTypeIdx:11259
:o: currStringOff:2018864
:o: string off:0x9cba8c1c
:o: classname:Landroid/arch/core/executor/ArchTaskExecutor$1;
:o: currClass ptr:0x9cafa1e0
:o: currClassIdx:98
:o: currTypeIdx:11260
```

拿到classname后我们就可以用这个去遍历loadClass后再dump dex即可。

可以看到这个题目最后想要说的就是自定义classloader如何进行函数抽取还原，那么这里我们需要熟悉Dexfile的文件结构以及了解classloader如何使用。

看雪ID：lemn

好消息！！现在看雪《安卓高级研修班》线下班 & 网课(12月班)开始同步招生 啦！以前没报上高研班的小伙伴赶快抓紧机会报名，升职加薪唾手可得！！

戳“阅读原文 ”一起来充电吧！返回搜狐，查看更多

责任编辑：