

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) {          首先拿到dexfile和artmethod
            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);          根据dexfile的偏移拿到base
                    dexfile_handle.write(dex_buffer);
                    dexfile_handle.flush();
                    dexfile_handle.close();
                    console.log("[dumpdex]:", dexfile_path);
                }
            }
        }
    });
}
```

```
82 if (dexfile_handle && dexfile_handle != null) {
83     dexfile_handle.close()
84 }
85
86 } catch (e) {
87     dexfile_handle = new File(dexfile_path, "a+");
88     if (dexfile_handle && dexfile_handle != null) {
89         var dex_buffer = ptr(dexfilebegin).readByteArray(dexfilesize);
90         dexfile_handle.write(dex_buffer);
91         dexfile_handle.flush();
92         dexfile_handle.close();
93         console.log("[dumpdex]:", dexfile_path);
94     }
95 }
96
97 var dexfileobj = new DexFile(dexfilebegin, dexfilesize);
98 if (dex_maps[dexfilebegin] == undefined) {
99     dex_maps[dexfilebegin] = dexfilesize;
100    console.log("got a dex:", dexfilebegin, dexfilesize);          保存dexfile
101 }
102 if (this.artmethodptr != null) {
103     var artmethodobj = new ArtMethod(dexfileobj, this.artmethodptr);
104     if (artmethod_maps[this.artmethodptr] == undefined) {
105         artmethod_maps[this.artmethodptr] = artmethodobj;          保存artmethod
106     }
107 }
108
109 });
110
111 ishook_libart = true;
112 }
```

```

C:\Users\MacBookPro\Desktop\FART-master\frida_fart\frida_fart_hook.js - Notepad++
文件(E) 编辑(B) 搜索(S) 视图(V) 编码(N) 语言(L) 设置(I) 工具(O) 宏(M) 运行(R) 插件(P) 窗口(W) ?
8958236_classlist.txt[1] 8958236_classlist_execute.txt[2] interpreter.c[3] art_method.c[4] frida_fart_hook.js[5]
文件夹工作区
FART_6.0_sourcecode
  |- art
  |- frameworks
  |- libcore
  |- XcustomModuleTest
    |- .gradle
    |- .idea
    |- app
      |- gradle
        |- .gitignore
        |- build.gradle
        |- gradle.properties
        |- gradlew
        |- gradlew.bat
        |- local.properties
        |- settings.gradle
        |- XcustomModuleTest.iml
129     if (dexfile_handle && dexfile_handle != null) {
130         var dex_buffer = ptr(dexfilebegin).readByteArray(dexfilesize);
131         dexfile_handle.write(dex_buffer);
132         dexfile_handle.flush();
133         dexfile_handle.close();
134         console.log("[dumpdex]:", dexfile_path);
135     }
136 }
137 var artmethodptr = artmethodobj.artmethodptr;
138 var dex_code_item_offset_ = Memory.readU32(ptr(artmethodptr).add(8));
139 var dex_method_index_ = Memory.readU32(ptr(artmethodptr).add(12));
140 if (dex_code_item_offset_ != null && dex_code_item_offset_ > 0) {
141     var dir = savepath;
142     var file_path = dir + "/" + dexfilesize + "_" + Process.getCurrentThreadId() + ".bin";
143     var file_handle = new File(file_path, "a+");
144     if (file_handle && file_handle != null) {
145         var codeitemstartaddr = ptr(dexfilebegin).add(dex_code_item_offset_);
146         var codeitemlength = funcGetCodeItemLength(ptr(codeitemstartaddr));
147         if (codeitemlength != null & codeitemlength > 0) {
148             Memory.protect(ptr(codeitemstartaddr), codeitemlength, 'rwx');
149             var base64lengthptr = Memory.alloc(8);
150             var arr = [0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00];
151             Memory.writeByteArray(base64lengthptr, arr);
152             var base64ptr = funcBase64_encode(ptr(codeitemstartaddr), codeitemlength, ptr(base64lengthptr));
153             var b64content = ptr(base64ptr).readCString(base64lengthptr.readInt());
154             funcFreeptr(ptr(base64ptr));
155             var content = "(name:0xxx,method_idx:" + dex_method_index_ + ",offset:" +
156             dex_code_item_offset_ + ",code_item_len:" + codeitemlength + ",ins:" +
157             b64content + ")";
158             file_handle.write(content);
159         }
160     }
}

```

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(this, thread, dex_file, it, klass, artmethod);

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

```

获取base和size

```
const DexHeader *base = dex_file.pHeader;
```

```
size_t size = dex_file.pHeader->fileSize;
```

获取code item offset和method idx

```
uint32_t codeItemOffset = artmethod->dex_code_item_offset_;
```

```
uint32_t idx = artmethod->dex_method_index_;
```

hook prettymethod方法 主动调用获得方法名

```
const std::string& string= prettyMethodFunction(artmethod, artmethod,
true);
```

通过偏移可以拿到codeitem

```
long codeItemAddr = (long) base + codeItemOffset;
```

```
Codelitem *codeItem = (Codelitem *) codeItemAddr;
```

这部分代码可以直接dump dex出来

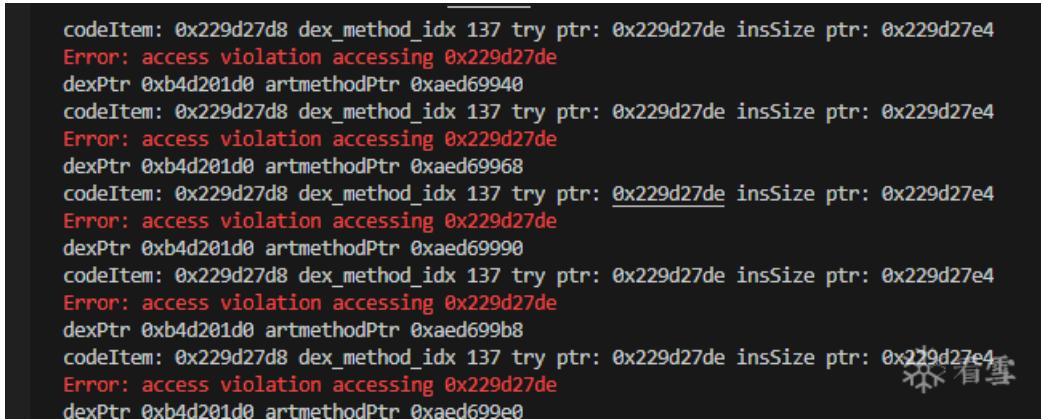
```
intpid = getpid;  
chardexFilePath[ 100] = { 0};  
  
sprintf(dexFilePath, "/sdcard/xxxxx/%p %d LoadMethod.dex", base, size);  
  
mkdir( "/sdcard/xxxxx", 0777);  
  
intfd = open(dexFilePath, O_CREAT | O_RDWR, 666);  
  
if(fd > 0) {  
  
    ssize_t i = write(fd, base, size);  
  
    if(i > 0) {  
  
        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。



```
codeItem: 0x229d27d8 dex_method_idx 137 try ptr: 0x229d27de insSize ptr: 0x229d27e4  
Error: access violation accessing 0x229d27de  
dexPtr 0xb4d201d0 artmethodPtr 0xaed69940  
codeItem: 0x229d27d8 dex_method_idx 137 try ptr: 0x229d27de insSize ptr: 0x229d27e4  
Error: access violation accessing 0x229d27de  
dexPtr 0xb4d201d0 artmethodPtr 0xaed69968  
codeItem: 0x229d27d8 dex_method_idx 137 try ptr: 0x229d27de insSize ptr: 0x229d27e4  
Error: access violation accessing 0x229d27de  
dexPtr 0xb4d201d0 artmethodPtr 0xaed69990  
codeItem: 0x229d27d8 dex_method_idx 137 try ptr: 0x229d27de insSize ptr: 0x229d27e4  
Error: access violation accessing 0x229d27de  
dexPtr 0xb4d201d0 artmethodPtr 0xaed699b8  
codeItem: 0x229d27d8 dex_method_idx 137 try ptr: 0x229d27de insSize ptr: 0x229d27e4  
Error: access violation accessing 0x229d27de  
dexPtr 0xb4d201d0 artmethodPtr 0xaed699e0
```

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(this, 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
    protected void afterHookedMethod(MethodHookParam param) throws Throwable {
        super.afterHookedMethod(param);
        final ClassLoader classLoader = (ClassLoader) param.thisObject;
        XcustomBridge.log("DexClassLoader:" + classLoader.toString());
        mClassLoaders.add(classLoader);
    }
});
XcustomBridge.hookAllConstructors(PathClassLoader.class, newXC_MethodHook {
    @Override
    protected void afterHookedMethod(MethodHookParam param) throws Throwable {
        super.afterHookedMethod(param);
        final ClassLoader 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
    protected void afterHookedMethod(MethodHookParam param) throws Throwable {
        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

public void run{

try{

Thread.sleep( 30* 1000);

} catch(InterruptedException e) {

e.printStackTrace;

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

}

for( int i = 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。

```

20 import com.sup.superb.video.u.g;
21 import com.sup.superb.video.e;
22 import com.sup.superb.video.e.k;
23
24 public abstract class AbsVideoPresenter implements LifecycleObserver, WeakHandler.IHandler, a
25     public static ChangeQuickRedirect a;
26     protected Handler b = new WeakHandler(this);
27     protected boolean c = false;
28     protected Activity d;
29     protected boolean e;
30     boolean f;
31     k g;
32     int h = 2;
33     private LifecycleOwner i;
34
35     public AbsVideoPresenter(Activity activity, LifecycleOwner lifecycleOwner, k kVar) {
36         lifecycleOwner.getLifecycle().addObserver(this);
37         this.i = lifecycleOwner;
38         this.d = activity;
39         this.g = kVar;
40     }
41
42     public void a(Activity activity, VideoModel videoModel) {
43         if (PatchProxy.isSupport(new Object[]{activity, videoModel}, this, a, false, 7206, ne
44             PatchProxy.accessDispatch(new Object[]{activity, videoModel}, this, a, false, 720
45         } else if (this.c && this.e) {
46             g.a().a((Context) activity, videoModel);
47         }

```

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 {
13     private Context context;
13     private ConcurrentHashMap<String, DnsRecord> hostMap;
13     private boolean isNeedRefetchOnExpire;
13     private String lastBSSID;
13     private int lastNet;
13     private BroadcastReceiver receiver;
13     private ConcurrentSkipListSet<String> resolvingSet;
13     private ConcurrentHashMap<String, DnsRecord> wifiHostMap;
13
13     static /* synthetic */ ConcurrentHashMap access$000(HostManager hostMa
13
13     static /* synthetic */ String access$100(HostManager hostManager) {
13
13     static /* synthetic */ String access$102(HostManager hostManager, St
13
13     static /* synthetic */ ConcurrentHashMap access$200(HostManager hostMana
13
13     static /* synthetic */ int access$300(HostManager hostManager) {
13
13     static /* synthetic */ int access$302(HostManager hostManager, int i
13

```

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 type dalvik.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.getFieldObject(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.getFieldObject(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;

public class DelegateClassLoader extends ClassLoader {
    private Method findClassMethod = MethodUtils.getAccessibleMethod(ClassLoader.class, "findClass", new Class[]{String.class}, new Type[]{String.class});
    private Method findLoadedClassMethod = MethodUtils.getAccessibleMethod(ClassLoader.class, "findLoadedClass", new Class[]{String.class}, new Type[]{String.class});
    private ClassLoader pathClassLoader;

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

    /* access modifiers changed from: protected */
    public Class<?> findClass(String str) throws ClassNotFoundException {
        Throwable th;
        //从自己的缓存中加载，如果失败，则从父类的缓存中加载
    }
}
```

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");

Object pathList_object = getFieldObject( "dalvik.system.BaseDexClassLoader", appClassLoader, "pathList");

Object[] ElementsArray = ( Object[]) getFieldObject( "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( Error e) {

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 instanceof BaseDexClassLoader) {

} elseif(appClassLoader instanceof ClassLoader) {

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。

```

10 import java.util.concurrent.RejectedExecutionException;
11 import java.util.concurrent.TimeUnit;
12
13 public class HttpDns implements HttpDnsService {
14     private static DegradationFilter degradationFilter;
15     static HttpDns instance;
16     private final String account;
17     private final Context context;
18     private final HostManager hostManager;
19     private boolean isExpiredIPEnabled = false;
20     private ConcurrentHashMap<String, Future<DnsRecord>> resolveingMap = r
21     private final long ttl;
22
23     private HttpDns(Context context2, String str, long j, boolean z) {
24         this.context = context2;
25         this.account = str;
26         this.hostManager = new HostManager(context2, z);
27         if (j > 300) {
28             this.ttl = j;
29         } else {
30             this.ttl = 300;
31         }
32     }
33
34     private DnsRecord getByHost(String str, long j) {
35         if (!Utils.isValidHost(str) || Utils.isNumericAddress(str)) {
36             return null;
37         }
38         if ((degradationFilter != null && degradationFilter.shouldDegradet
39             return null;
40         }
41         DnsRecord record = this.hostManager.getRecord(str);
42         if (record == null || !record.isExpired() || !this.isExpiredIPEnat
43             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;  
        longcurrTypeldAddr = (( long) typeIdsOff + 4* tmpldx + ( long) base);  
        int*currTypeldx = ( int*) currTypeldAddr;  
        __android_log_print( 5, "hookso", "currTypeldx:%d", *currTypeldx);  
        tmpldx = *currTypeldx;  
        longcurrStringOffAddr = (( long) stringIdsOff + 4* tmpldx + ( long) base);  
        int*currStringOff = ( int*) currStringOffAddr;  
        __android_log_print( 5, "hookso", "currStringOff:%d", *currStringOff);  
        tmpldx = *currStringOff;  
        longoff = ( long) base+ tmpldx;  
        __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:0x9cafaf1a0
:o: currClassIdx:96
:o: currTypeIdx:11258
:o: currStringOff:2018841
:o: string off:0x9cba8c05
:o: classname:Landroid/arch/core/R;
:o: currClass ptr:0x9cafaf1c0
:o: currClassIdx:97
:o: currTypeIdx:11259
:o: currStringOff:2018864
:o: string off:0x9cba8c1c
:o: classname:Landroid/arch/core/executor/ArchTaskExecutor$1;
:o: currClass ptr:0x9cafaf1e0
: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：