# [原创] 小红书 6.89.0.1版本 shield unidbg

贪吃虫 6

极客



2021-8-25 18:10



> **本文章仅供学习，如有侵权，请告知删除**

就在刚刚1629879304，用小红书练习unidbg成功了，解处了shield，趁着现在记忆力还好，赶紧写下，要不然肯定会忘了。这里严重感谢龙哥（你要是不知道龙哥是谁，可以加我我告诉你）

首先呢，shield我就默认大家都知道是什么，就是小红书请求中header的一个加密字段，具体生成位置在（位置怎么找网上都有，so用的libshield.so，本篇重点在unidbg使用，其实找这个位置是最难的）：

```
com.xingin.shield.http.XhsHttpInterceptor的这个native方法
public native Response intercept(Interceptor.Chain chain, long j2) throws IOException;
```

既然知道native方法了，那我们就开始用unidbg开发把（补环境吧），首先把最基本的unidbg框架搭建以下（别告诉我你不会搭建，我都把注释写好了，什么意思都告诉你了）这就是一个最基础的模板，开搞之前先把这个弄好

```
*/
public class XhsShieldTest extends AbstractJni {
    private final VM vm;
    private final Module module;
    private final AndroidEmulator emulator;

    public static void main(String[] args) {
        XhsShieldTest xhsShield = new XhsShieldTest();
    }

    XhsShieldTest() {
        emulator = AndroidEmulatorBuilder.for32Bit().build(); 定义模拟器一般都是32位的，如果so是64位那就是64
        final Memory memory = emulator.getMemory();
        memory.setLibraryResolver(new AndroidResolver( sdk: 23));  那内存拿出来设置以下sdk版本，23就是安卓6

        vm = emulator.createDalvikVM(new File( pathname: "unidbg-android/src/test/resources/apk/xhs_6_89_0_1.apk"));
        vm.setVerbose(true); //打印虚拟器日志  把虚拟机也设置以下        要加载apk，不管有没有用，加载准没错
        vm.setJni(this);

        DalvikModule dm = vm.loadLibrary(new File( pathname: "unidbg-android/src/test/resources/so/libshield.so"),  forceCallInit: true);
        module = dm.getModule();                    然后加载我们的目标so
        dm.callJNI_OnLoad(emulator);
    }                   onlaod以下，如果是静态注册，可以不用，但是写了没坏处，所以不管静态动态，都写上，准没错，饱和式救援嘛
}
```

下面开始正式调用这个native函数了

```
intercept(Interceptor.Chain chain, long j2)
```

它有两个参数，第一个是Interceptor.Chain对象，我们就传初始化一个对象传进取（其实我感觉应该hook一下看这个对象有没有什么特殊的），第二参数一个long字段，这个字段我们通过jadx反编译代码后，可以看出是另外一个native方法的返回值

```
public native long initialize(String str);
```

```
import okhttp3.Request;
import okhttp3.Response;

public class XhsHttpInterceptor implements Interceptor {
    public long cPtr;
    public a<Request> predicate;

    static {
        initializeNative();
    }

    public XhsHttpInterceptor(String str, a<Request> aVar) {
        this.cPtr = initialize(str);
        this.predicate = aVar;
    }

    public static native void initializeNative();

    public static XhsHttpInterceptor newInstance(String str) {
        return new XhsHttpInterceptor(str, null);
    }

    public native void destroy(long j2);

    public void finalize() throws Throwable {
        super.finalize();
        destroy(this.cPtr);
    }

    public native long initialize(String str);

    @Override // okhttp3.Interceptor
    public Response intercept(Interceptor.Chain chain) throws IOException {
        Response response;
        long currentTimeMillis = ContextHolder.sJavaLogAble ? System.currentTimeMillis() : 0;
        a<Request> aVar = this.predicate;
        if (aVar == null || aVar.test(chain.request())) {
            response = intercept(chain, this.cPtr);
        } else {
            response = chain.proceed(chain.request());
        }
        if (ContextHolder.sJavaLogAble) {
            Log.i("XhsHttpInterceptor", "cost:" + (System.currentTimeMillis() - currentTimeMillis));
        }
        return response;
    }

    public native Response intercept(Interceptor.Chain chain, long j2) throws IOException;

    public static XhsHttpInterceptor newInstance(String str, a<Request> aVar) {
        return new XhsHttpInterceptor(str, aVar);
    }
}
```

而这个方法需要一个字符串参数，这个参数我们通过jadx找一下调用位置，可以看到是"main"（这个简单就不截图了）。

到目前为之我们需要的东西都有了，把调用代码逻辑写上，然后只要运行代码，把报错的环境给补上就可以了。在调用目标函数之前，需要调用一个初始化native函数，如果没有可以不调，但是有的话必须调用。

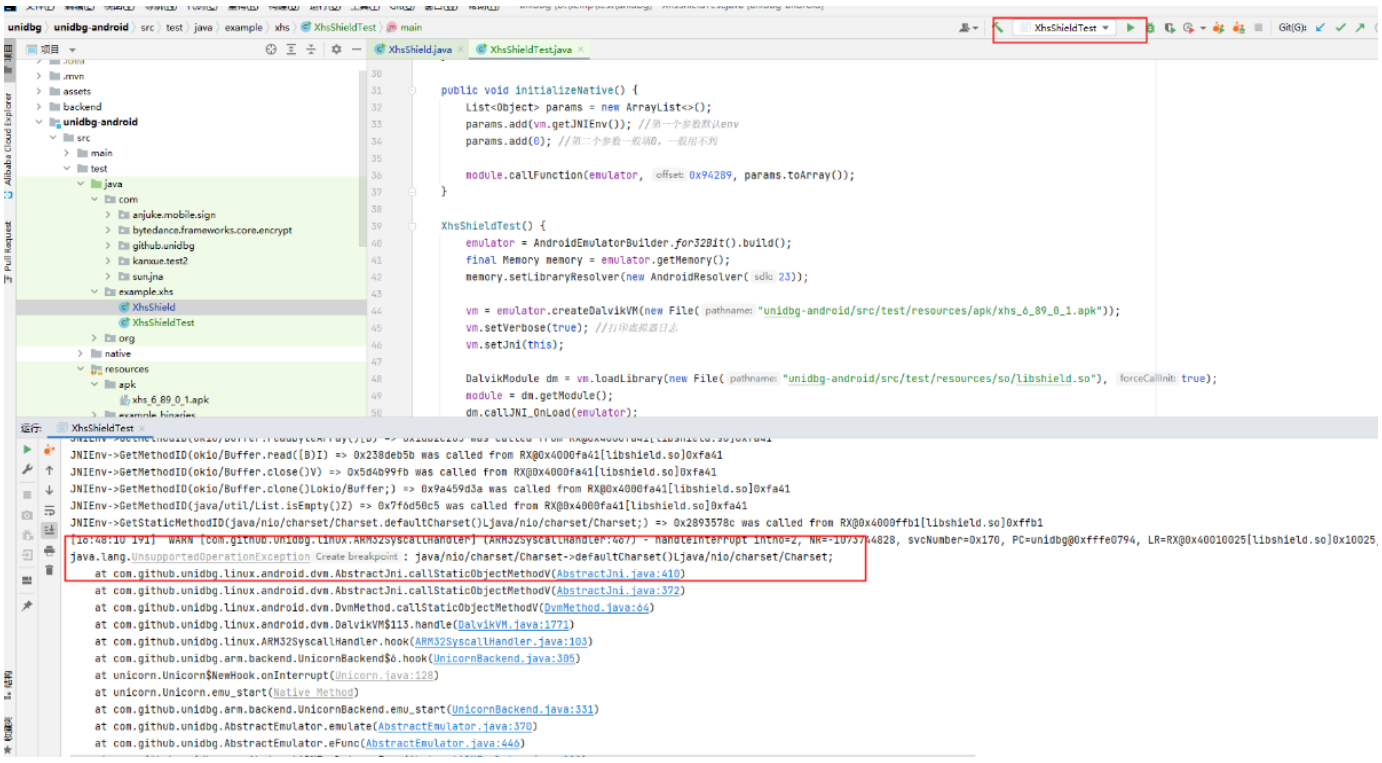
具体步骤：

> 1：调用初始化native函数（调用的时候把报错的环境补完）
> 2：调用public native long initialize(String str);方法（调用的时候吧报错的环境补完）
> 3：调用最后的目标函数intercept(Interceptor.Chain chain, long j2)（调用的时候吧报错的环境补完）
> 最后就可以了，注意一个一个来，初始化native环境补完了，就把调用第二个函数的逻辑写上继续运行，不要一次性把所有调用逻辑写完，这样很可能出问题

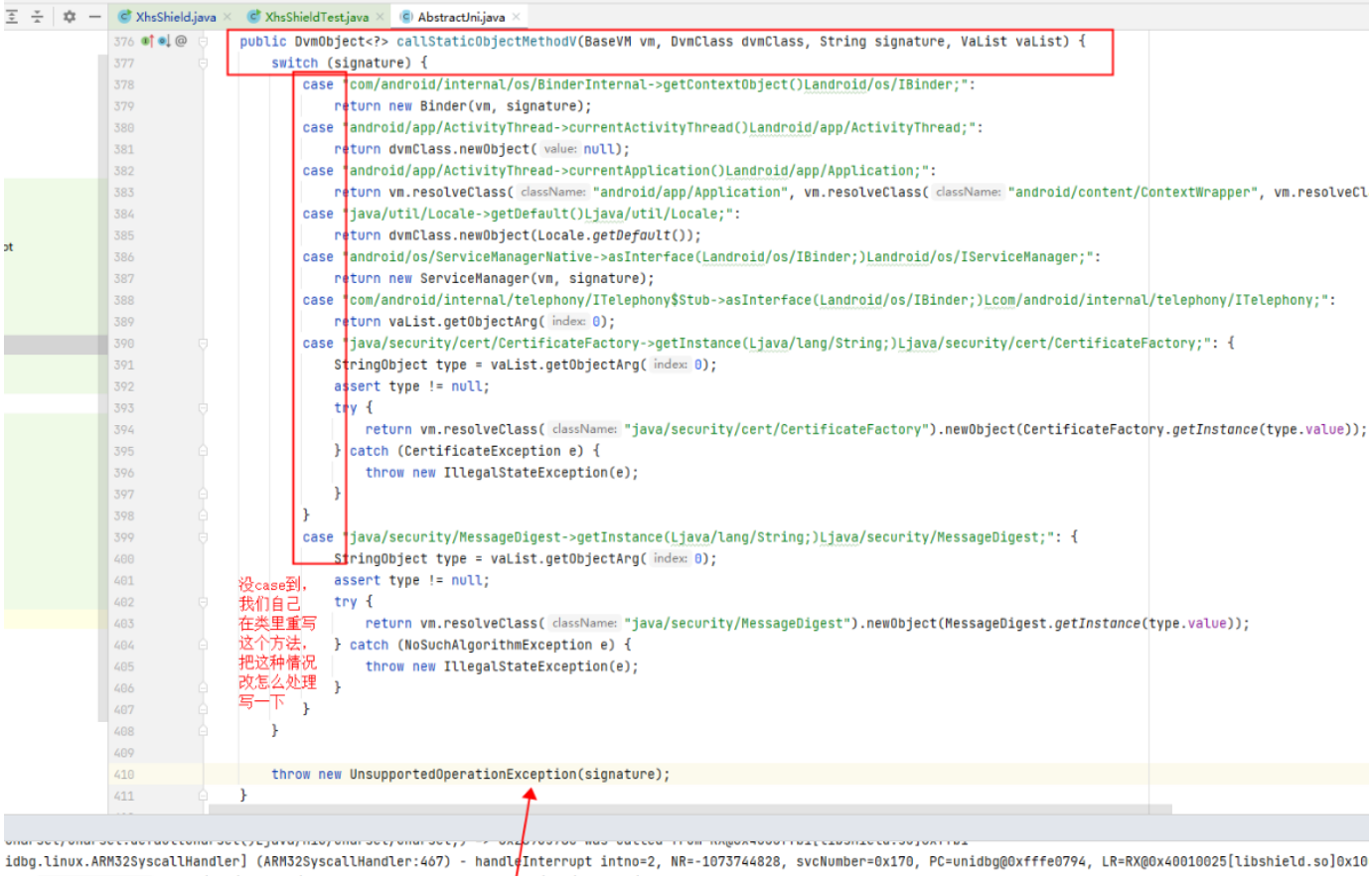
先把初始化函数的代码逻辑写上

```
22    private final AndroidEmulator emulator;
23
24    public static void main(String[] args) {
25        XhsShieldTest xhsShield = new XhsShieldTest();
26
27        xhsShield.initializeNative();
28
29    }
30
31    public void initializeNative() {          初始化
32        List<Object> params = new ArrayList<>();
33        params.add(vm.getJNIEnv()); //第一个参数默认env
34        params.add(0); //第二个参数一般填0，一般用不到
35
36        module.callFunction(emulator, offset: 0x94289, params.toArray());
37    }
38
39    XhsShieldTest() {
40        emulator = AndroidEmulatorBuilder.for32Bit().build();
41        final Memory memory = emulator.getMemory();
42        memory.setLibraryResolver(new AndroidResolver( sdk: 23));
43
44        vm = emulator.createDalvikVM(new File( pathname: "unidbg-android/src/test/resources/apk/xhs_6_89_0_1.apk"));
45        vm.setVerbose(true); //打印虚拟器日志
46        vm.setJni(this);
47
48        DalvikModule dm = vm.loadLibrary(new File( pathname: "unidbg-android/src/test/resources/so/libshield.so"), forceCallInit: true);
49        module = dm.getModule();
50        dm.callJNI_OnLoad(emulator);
51    }
52}
53
```

运行一下




报错了，看到这个错误一定要笑出来（内心(^_^)），因为这个错误最好处理了，就是我们说的补环境，我解释下什么意思啊这个错误：就是说java/nio/charset/Charset这个类实例调用defaultCharset()方法，返回java/nio/charset/Charset，没找到处理的逻辑，因为unidbg不可能把所有的情况都给我们写好，如果遇到这种比较个性的话情况，我们就自己写处理逻辑，我们点击第一个错误的位置看看



我们复制这个函数到我们的类，自己重写一下，如果遇到这个signatur，因该怎么做

```
        params.add(vm.getJNIEnv()); //第一个参数默认env
        params.add(0); //第二个参数一般填0，一般用不到

        module.callFunction(emulator, offset: 0x94289, params.toArray());
    }

    XhsShieldTest() {
        emulator = AndroidEmulatorBuilder.for32Bit().build();
        final Memory memory = emulator.getMemory();
        memory.setLibraryResolver(new AndroidResolver( sdk: 23));

        vm = emulator.createDalvikVM(new File( pathname: "unidbg-android/src/test/resources/apk/xhs_6_89_0_1.apk"));
        vm.setVerbose(true); //打印虚拟器日志
        vm.setJni(this);

        DalvikModule dm = vm.loadLibrary(new File( pathname: "unidbg-android/src/test/resources/so/libshield.so"), forceCallInit: true);
        module = dm.getModule();
        dm.callJNI_OnLoad(emulator);
    }

    @Override
    public DvmObject<?> callStaticObjectMethodV(BaseVM vm, DvmClass dvmClass, String signature, VaList vaList) {
        switch (signature) {
            case "java/nio/charset/Charset->defaultCharset()Ljava/nio/charset/Charset;": {
                return vm.resolveClass( className: "java/nio/charset/Charset").newObject(Charset.defaultCharset());
            }

        }                                                    把刚才那个方法复制过来，写一个出现这种情况的处理代码

        return super.callStaticObjectMethodV(vm, dvmClass, signature, vaList);
    }
}
```

现在你们可能疑问，为什么这样补，为什么要return这个，而不是return别的

下面这个图来解释一下

```
        module = dm.getModule();
        dm.callJNI_OnLoad(emulator);
    }

    @Override
    public DvmObject<?> callStaticObjectMethodV(BaseVM vm, DvmClass dvmClass, String signature, VaList vaList) {
        switch (signature) {
            case "java/nio/charset/Charset->defaultCharset()Ljava/nio/charset/Charset;": {
                return vm.resolveClass( className: "java/nio/charset/Charset").newObject(Charset.defaultCharset());
            }

        }   意思是最终要返回Chaset这个对象，这个对象怎么生成呢，用Charset.defaultChaset()函数生成，
            所以我们就这样return了，当然返回基本类型可能跟这个写发不太一样，但是道理是一样的
        return super.callStaticObjectMethodV(vm, dvmClass, signature, vaList);
    }
}
```

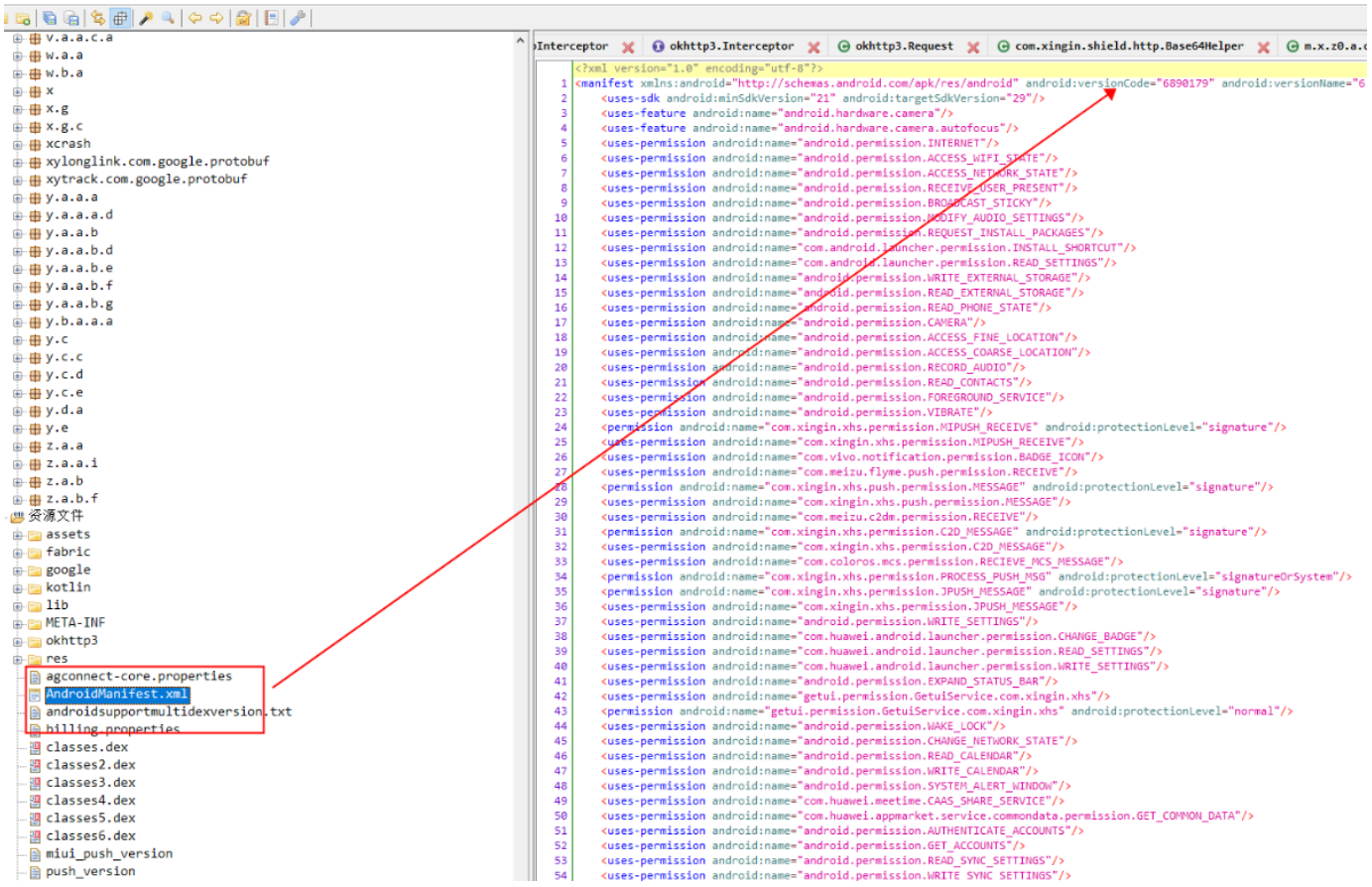
这个就是Charset对象，有时候为空就要用jintrace来看返回值

存储参数，如果defaultCharset()有参数，这里面会存参数

补好后，继续点运行

```
JNIEnv->CallObjectMethodV(android.content.pm.PackageManager@6bedbc4d, getPackageInfo("com.xingin.xhs", 0x20000) => a
JNIEnv->FindClass(android/content/pm/PackageInfo) was called from RX@0x4000ff41[libshield.so]0xff41
JNIEnv->GetFieldID(android/content/pm/PackageInfo.versionCodeI) => 0x41c3e70b was called from RX@0x4000fbb1[libshiel
[17:02:07 670]  WARN [com.github.unidbg.linux.ARM32SyscallHandler] (ARM32SyscallHandler:467) - handleInterrupt intno
java.lang.UnsupportedOperationException Create breakpoint : android/content/pm/PackageInfo->versionCode:I
    at com.github.unidbg.linux.android.dvm.AbstractJni.getIntField(AbstractJni.java:534)
    at com.github.unidbg.linux.android.dvm.AbstractJni.getIntField(AbstractJni.java:529)
    at com.github.unidbg.linux.android.dvm.DvmField.getIntField(DvmField.java:64)
    at com.github.unidbg.linux.android.dvm.DalvikVM$97.handle(DalvikVM.java:1433)
    at com.github.unidbg.linux.ARM32SyscallHandler.hook(ARM32SyscallHandler.java:103)
    at com.github.unidbg.arm.backend.UnicornBackend$6.hook(UnicornBackend.java:305)
    at unicorn.Unicorn$NewHook.onInterrupt(Unicorn.java:128)
    at unicorn.Unicorn.emu_start(Native Method)
```

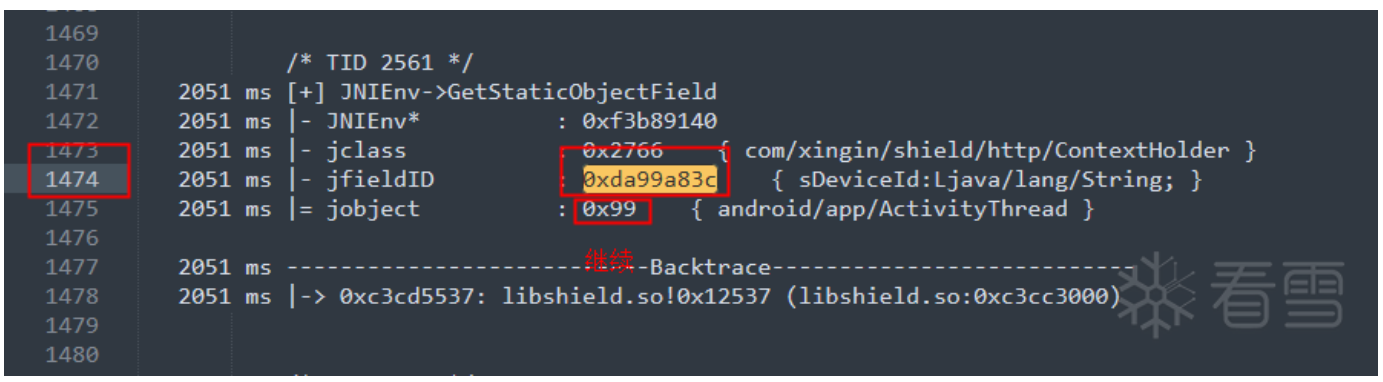
这个一看是android文件的某个类的某个参数，网上查了下，在这个地方

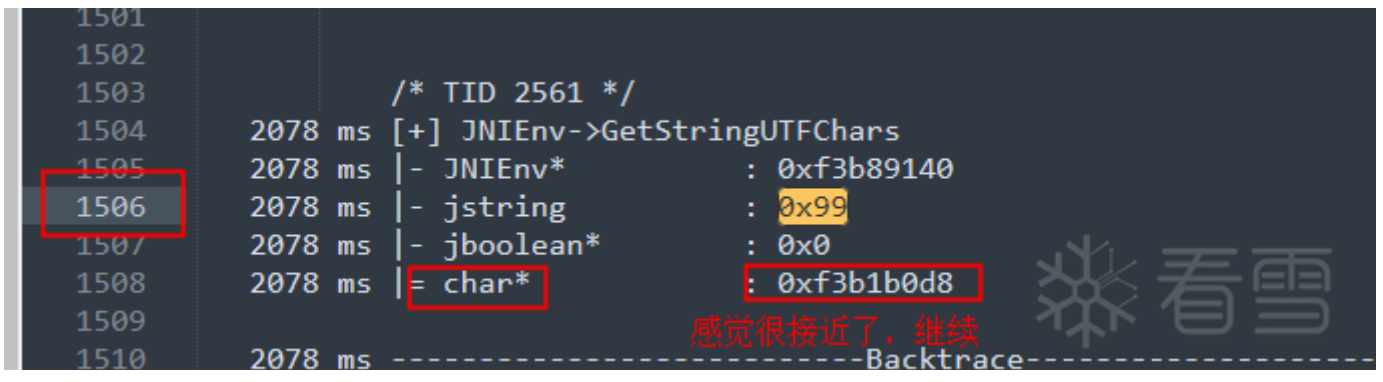



继续运行，继续看


**这次要某个类的sDeviceId,这里是重点，刚因为jnitrace分析错误导致shield短了，我的信誉在群里全毁了，又退回到傻傻的初学者职位了，经历了社会性死亡瞬间。**

jnitrace怎么把日志那出来就不说了，直接上分析的路径图，关键点就是**保障TID一样**，因为有多线程，如果分析到其他线程就错了，还有就是**按顺序来**，别刚刚在1800行，找下一个位置找到了1600行，得往下找，别往上找。

```
1555            /* TID 2561 */
1556     2122 ms [+] JNIEnv->ReleaseStringUTFChars
1557     2122 ms |- JNIEnv*          : 0xf3b89140
1558     2122 ms |- jstring          : 0xf3b1b0d8
1559     2122 ms |- char*            : 0xf3b1b0d8
1560     2122 ms |:     81e30ab8-2b81-33dd-8435-f9404554b4b5
1561
1562     2122 ms ------------------------ Backtrace ------------------------
1563     2122 ms |-> 0xc3cd3cbf: libshield.so!0x10cbf (libshield.so:0xc3cc3
```

冰狗，找到了

找到了这个字符串，就是我们的sDevicesid啦，把他返回就可以了

就这样往复几十次，把所有报错修复了就可以了，是不是很简单?

我就不把所有补的都截图了，因为太多了，大多数原理都一样，看返回值是什么，按照要求弄就可以了。

```
@Override
public DvmObject<?> getStaticObjectField(BaseVM vm, DvmClass dvmClass, String signature) {
    switch (signature) {
        case "com/xingin/shield/http/ContextHolder->sDeviceId:Ljava/lang/String;": {
            return new StringObject(vm, value: "81e30ab8-2b81-33dd-8435-f9404554b4b5");
        }
    }
    return super.getStaticObjectField(vm, dvmClass, signature);
}
```

初始化native这个报错全都补完后，再把调用获取第二个参数的方法逻辑写上，继续运行，继续补环境，接着是目标函数，知道最终不报错了。

```
public void initializeNative() {
    List<Object> params = new ArrayList<>();
    params.add(vm.getJNIEnv()); //第一个参数默认env
    params.add(0); //第二个参数一般填0，一般用不到

    module.callFunction(emulator, offset: 0x94289, params.toArray());
}

public long initialize() {
    List<Object> params = new ArrayList<>();
    params.add(vm.getJNIEnv()); //第一个参数默认env
    params.add(0); //第二个参数一般填0，一般用不到

    params.add(vm.addLocalObject(new StringObject(vm, value: "main")));
    Number number = module.callFunction(emulator, offset: 0x937b1, params.toArray())[0];
    return number.longValue();
}

public void intercept(long initialize) {        目标函数
    List<Object> params = new ArrayList<>();
    params.add(vm.getJNIEnv()); //第一个参数默认env
    params.add(0); //第二个参数一般填0，一般用不到

    DvmObject<?> chain = vm.resolveClass( className: "okhttp3/Interceptor$Chain").newObject( value: null);
    params.add(vm.addLocalObject(chain));
    params.add(initialize);
    Number[] numbers = module.callFunction(emulator, offset: 0x939d9, params.toArray());
    Object result = vm.getObject(numbers[0].intValue()).getValue();
    System.out.println(result);
}
```

最终你会在这里得到shield值，因为shield值header的一个参数，所以补环境的时候，你会还想这个地

方可能是shield生产的位置，打印一下确认就是了

```
    }
    case "okhttp3/Request$Builder->header(Ljava/lang/String;Ljava/lang/String;)Lokhttp3/Request$Builder;": {
        Request.Builder builder = (Request.Builder) dvmObject.getValue();
        String param1 = (String) vaList.getObjectArg( index: 0).getValue();
        String param2 = (String) vaList.getObjectArg( index: 1).getValue();
        System.out.println("**********************************");
        System.out.println("key:" + param1);
        System.out.println("value:" + param2);
        if ("shield".equals(param1)) {
            if (param2 != null) {
                shield = param2;
            }
        }
        System.out.println("**********************************");
        return vm.resolveClass( className: "okhttp3/Request$Builder").newObject(builder.header(param1, param2));
    }
```

```
[17:36:29 044] DEBUG [com.github.unidbg.linux.android.dvm.DalvikVM] (DalvikVM$32:518) - callObjectMethodV object=unidbg@0x4b8ee4de, jmethodID=unidbg@0x4ad64af0, va_list=unidbg@0xbffff534, lr=RX@0x4000fab5[libshield.so]0xfab5
**********************************
key:shield
value:XYAAAAAQAAAAEAAABTAAAAUzUWEe0xG1IbD9/c+qCLOlKGmTtFa+lG43AGdeFXQ6RKxNGznuc1RJ3örehez8MpiJl+2aYxFQwYGGGIYL7x2Hlk1OHCfósDBEBBtqPfFNN7izhq
**********************************
JNIEnv->CallObjectMethodV(okhttp3.Request$Builder@4b8ee4de, header("shield", "XYAAAAAQAAAAEAAABTAAAAUzUWEe0xG1IbD9/c+qCLOlKGmTtFa+lG43AGdeFXQ6RKxNGznuc1RJ3örehez8MpiJl+2aYxFQwYGGGIYL7x2Hlk1OHCfósDBEBBtqPfFNN7izhq") => okhttp3
[17:36:29 044] DEBUG [com.github.unidbg.linux.android.dvm.BaseVM] (BaseVM:131) - addObject hash=0x27f981c6, global=false
```

以上就是小红书shield，用unidbg的调用方式。你可能会决定这样太简单了，是的，就是很简单。
只要你会用工具，你就超越了绝大多数猴子--出自《2001太空漫游》
-
有兴趣的加个微信呦，带你认识大佬，带你赶龙超肉
wechat:zhoutianxing_





**最新回复** (6)

**陈世已忘** 2021-8-26 23:34 2 楼 0

最新版7.6版本shield zhao我 eGhzIGtzIG**********IA== base64解码以后看

禁止

最后于 2021-8-27 09:42 被kanxue编辑， 原因： 广告

**贪吃虫** 2021-8-27 08:38 3 楼 0

> 陈世已忘 最新版7.6版本shield zhao我 eG**********A= ...

极客

以学习为目的的可以加我进群和龙哥学习最新版的解密方法呀（little wink）

最后于 2021-8-27 09:42 被kanxue编辑， 原因：

**sicssss** 2021-8-27 09:32 4 楼 0

> 贪吃虫 以学习为目的的可以加我进群和龙哥学习最新版的解密方法呀（little wink）

临时

已经在群里了，等新版解密方法

**最新回复** (6)

临时

**wx_我住隔壁我姓张** 0 2021-9-8 15:00

5 楼 0 ...

等

最后于 *2021-9-8 15:20* 被*wx_我住隔壁我姓张*编辑，原因：

15

3

¥

临时

**mb_eroobxld** 0 2021-9-12 16:42

6 楼 0 ...

> sicssss 已经在群里了，等新版解密方法

怎么加群

极客

**贪吃虫** 6 2021-9-12 21:11

7 楼 0 ...

> mb_eroobxld 怎么加群

可以加我微信（文章底部）拉你加群呦（wink）

看雪高研

内容

回帖 表情 高级回复

返回