

百度和网页 <https://www.52pojie.cn/thread-674074-1-1.html> 的作者无关，不对其内容负责。百度快照仅为网络故障时之索引，不代表被搜索网站的即时页面。

【网络诊断修复工具】

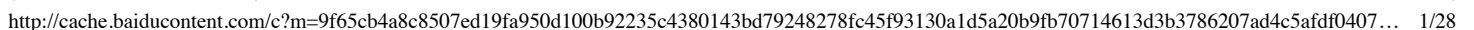


站点帮助
站务处理

热搜: [新手](#) [脱壳](#) [内购](#) [ctf](#) [去广告](#) [破解版](#) [绿色版](#) [破解教程](#)

[下一页](#)

不多说了，赶紧来分析吧，不过本文不在利用粗暴的静态方式去破解了，应广大同学要求，就介绍IDA动态调试so来进行破解，这样也能给大家带来IDA的使用技巧。毕竟我写文章技术都是为了你们。这种分析请求数据的突破口一般都是抓包，这不用多说了。不过都是用了https请求，所以需要手动在设备中安装Fiddler证书，才能抓到正确的数据信息：



log.snssdk.com /service/2/app_log/?id=167158639

Tunnel to frontier.snssdk.com:443

is.snssdk.com /service/1/z_app_stats/?id=167158639

frontier.snssdk.com /v1/z_app_stats/?id=167158639

is.snssdk.com /service/settings/v2/?app=1&retry_type=1

is.snssdk.com /feedback/2/list/?appkey=aweme-a

Tunnel to is.snssdk.com:443

api.amemv.com /aweme/v1/screen/ad/?unused=0

is.snssdk.com /service/2/app_alert/?has_market=1

alog.umeng.com /app_logs

Tunnel to uop.umeng.com:443

aweme.snssdk.com /aweme/v1/feed/?type=0&max_cursor=1

uop.umeng.com /

openbox.mobiledm.360.cn /service/pluginStatus?os=19&vc=3

api.amemv.com /aweme/v1/nearby/feed/?max_cursor=1

alog.umeng.com /app_logs

Tunnel to uop.umeng.com:443

uop.umeng.com /

openbox.mobiledm.360.cn /channel/getPlugInfoByPnames?&p

api.amemv.com /aweme/v1/crawl/sdk/log/?retry_type=1

Tunnel to api.amemv.com:443

p.s.360.cn /pstat/plog.php

api.amemv.com /aweme/v1/settings/?retry_type=1

sp.adpushonline.com:7088 /sdk_p/d

Tunnel to security.snssdk.com:443

security.snssdk.com /api/plugin/config/v1/?id=167158639

p3.pstatp.com /large/446e0005f2d99bf43bc65.jpeg

p3.pstatp.com /

p3.pstatp.com /aweme/100x100/3ee700098dfb41dcee55.jpeg

p3.pstatp.com /live/100x100/3ee700098dfb41dcee55.jpeg

p1.pstatp.com /

p1.pstatp.com /aweme/100x100/3ee4001749fe6b.jpeg

p3.pstatp.com /large/3fa00006d20ef0dc8360.jpeg

Tunnel to aweme.snssdk.com:443

aweme.snssdk.com /aweme/v1/play/?video_id=3a5f6c

v7.pstatp.com /3f134beb6032610998c588682263

api.amemv.com /aweme/v1/user/?user_id=604970

api.amemv.com /aweme/v1/aweme/stats/?id=1671

p3.pstatp.com /aweme/1080x1080/3ee700098dfb41dcee55.jpeg

beacon.tingyun.com /xhr?av=1.6.2&v=1.6.0&key=heTr

101.199.124.155 /index.html

m.shouji.360tpcdn.com /KtVBlqw1yoVIRPD9ZHIXcvXRJvhv

m.shouji.360tpcdn.com /KtVBlqw1yoVIRPD9ZHIXcvXRJvhv

m.shouji.360tpcdn.com /KtVBlqw1yoVIRPD9ZHIXcvXRJvhv

Tunnel to security.snssdk.com:443

m.shouji.360tpcdn.com /KtVBlqw1yoVIRPD9ZHIXcvXRJvhv

m.shouji.360tpcdn.com /KtVBlqw1yoVIRPD9ZHIXcvXRJvhv

Headers TextView SyntaxView WebForms HexView Auth Cookies Raw

QueryString

Name

type

max_cursor

min_cursor

count

retry_type

id

device_id

ac

channel

aid

app_name

version_code

version_name

device_platform

ssmix

device_type

device_brand

Transformer Headers TextView SyntaxView ImageView HexView WebView Au

JSON

aweme_list

author

avatar_larger

uri=3ee700098dfb41dcee55

url_list

https://p3.pstatp.com/aweme/1080x1080/3ee700098dfb41dcee55.jpeg

https://pb9.pstatp.com/aweme/1080x1080/3ee700098dfb41dcee55.jpeg

https://pb3.pstatp.com/aweme/1080x1080/3ee700098dfb41dcee55.jpeg

avatar_medium

uri=3ee700098dfb41dcee55

url_list

https://p3.pstatp.com/aweme/720x720/3ee700098dfb41dcee55.jpeg

https://pb9.pstatp.com/aweme/720x720/3ee700098dfb41dcee55.jpeg

https://pb3.pstatp.com/aweme/720x720/3ee700098dfb41dcee55.jpeg

avatar_thumb

uri=3ee700098dfb41dcee55

url_list

https://p3.pstatp.com/aweme/100x100/3ee700098dfb41dcee55.jpeg

https://pb9.pstatp.com/aweme/100x100/3ee700098dfb41dcee55.jpeg

https://pb3.pstatp.com/aweme/100x100/3ee700098dfb41dcee55.jpeg

https请求想要抓包，得先安装Fiddler证书

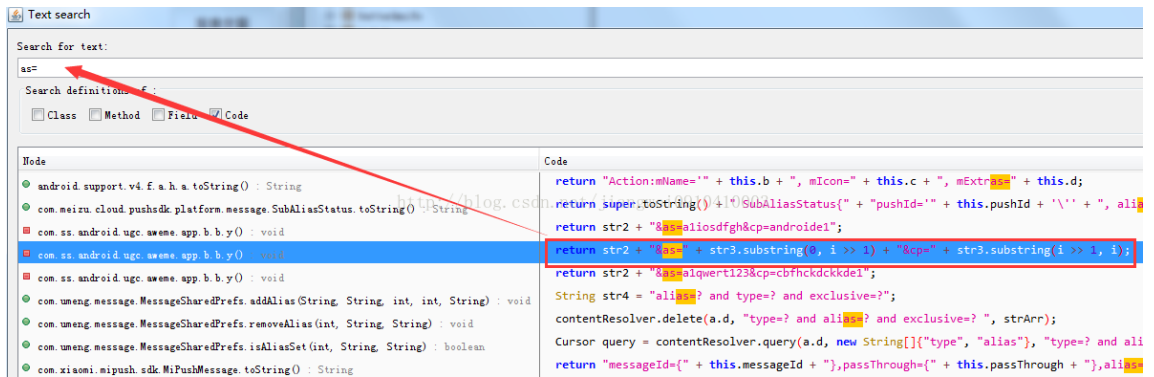
这三个字段是用来进行分页请求功能的

打开抖音之后，看到数据刷的很快，发现一个feed的接口是返回的首页的数据，在分析它的请求参数中有三个字段是minCursor,maxCursor,count其实这三个字段就是后面他进行数据分页请求的关键，到后面再详细说。不过这里看到还没有什么问题，不过问题往下看他的更多请求参数，会发现两个字段：

QueryString	
Name	Value
version_name	1.5.9
device_platform	android
ssmix	a
device_type	MI 3
device_brand	Xiaomi
os_api	19
os_version	4.4.4
uuid	863970029764198
openudid	b39d9675ee6af5b2
manifest_version_code	159
resolution	1080*1920
dpi	480
update_version_code	1592
ts	1509843697
app_type	normal
as	a175067f912fe9a21e
cp	61fc965f18eff822e1

这时候会发现其他参数都和本地设备有关或者直接写死的，唯独这两个参数信息是始终变化的。所以猜想这两个字段是F+，请求协议数据加密和服务端进行校验的。那么如果我们想单独构造信息去请求，这两个字段的信息一定要正确，不然请求不到正确的数据的。好了，这里简单了，使用Jadx打开抖音app即可，直接全局搜索字段的信息"as="：

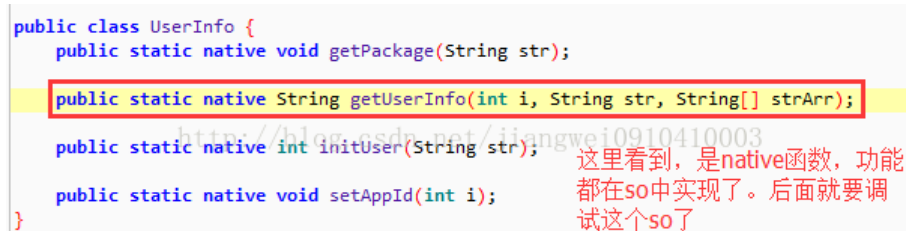
http://cache.baiducontent.com/c?m=9f65cb4a8c8507ed19fa950d100b92235c4380143bd79248278fc45f93130a1d5a20b9fb70714613d3b3786207ad4c5afdf0407... 2/28



看到这里就是构造了as和cp两个字段的值，直接点击进入即可：



这里大致的逻辑也比较简单，利用UserInfo.getUserInfo函数获取字符串，然后对半开给as和cp两个字段，右移操作就是除以2的意思。这里不分析代码来看看参数怎么来的，直接用Xposed进行hook这个函数打印参数看结果，粗暴快捷，以后其实这都是快速解决问题的一种方式，hook大法是最无敌的：



先来看一下这个加密方法的参数信息，看到是native的，也就是说加密操作是在so中做的，后面需要调试的也是这个so了。也不管了，hook这个方法然后打印信息看参数构造：

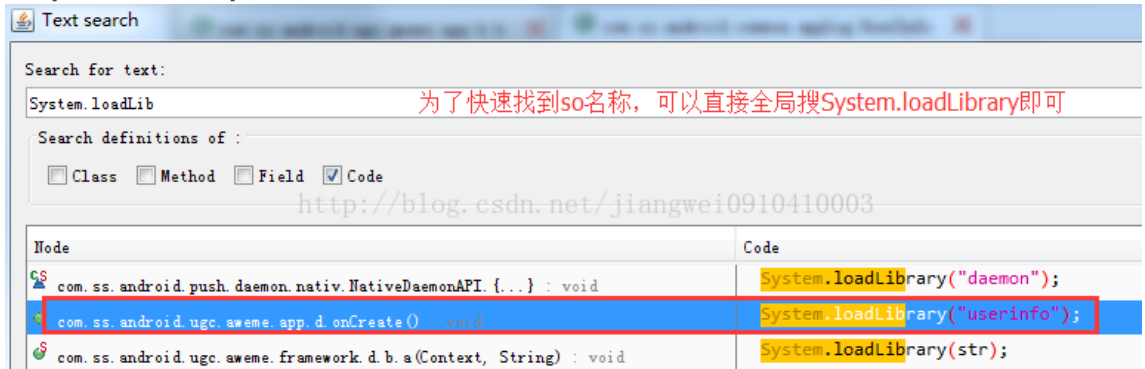


直接运行模块，打印日志看信息：

```
com.ss.android.ugc.aweme    jw    args0:1509845330
com.ss.android.ugc.aweme    jw    args1:https://api.amemv.com/aweme/v1/feed/?type=0&max_cursor=0&min_cursor=0&c
count=6&retry_type=no_retry&iid=16715863991&device_id=40545321430&ac=wifi&chan
nel=360&aid=1128&app_name=aweme&version_code=159&version_name=1.5.9&device_pl
atform=android&ssmix=a&device_type=MI+3&device_brand=Xiaomisos_api=19&os_vers
ion=4.4.4&uuid=863970029764198&openudid=b39d9675ee6af5b2&manifest_version_cod
e=159&resolution=1080*1920&dpi=480&update_version_code=1592&ts=1509845330&app
type=normal
com.ss.android.ugc.aweme    jw    args2:iid:16715863991:device_id:40545321430:ac:wifi:channel:360:aid:1128:app_
name:aweme:version_code:159:version_name:1.5.9:device_platform:android:ssmix:
a:device_type:MI 3:device_brand:Xiaomi:os_api:19:os_version:4.4.4:uuid:863970
029764198:openudid:b39d9675ee6af5b2:manifest_version_code:159:resolution:1080
*1920:dpi:480:update_version_code:1592:app_type:normal:
com.ss.android.ugc.aweme    jw    result:a1f566bf52e509095e615b945e26e0f59ee1
```

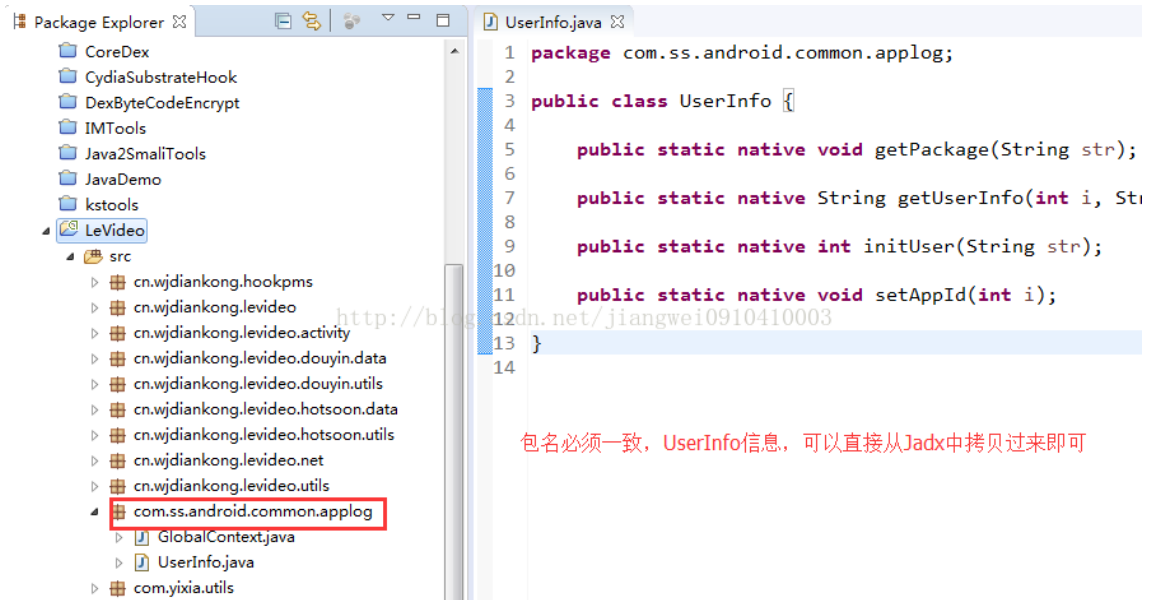
参数一应该是当前时间戳
参数二应该是请求url
参数三是请求的参数拆分字段
数组信息

看到三个参数打印的值，发现大致三个参数的意义是：当前时间戳，请求url，请求参数的数组信息。好了，我们现在可以新建一个Android工程，然后构造这三个参数信息，然后调用它的so函数，为了找到这个so名称，全局搜索字符串信息"System.loadLibrary"即可：



为了快速找到so名称，可以直接全局搜System.loadLibrary即可

这样就找到了这个so名称了，到libs目录下把这个so拷贝出来到我们自己构建的demo工程中，这里先不着急调试去分析native的加密函数功能，还是老规矩，拿来主义，直接上层构造一个和他一样的包名的native函数，调用它的so获取结果即可：



包名必须一致，UserInfo信息，可以直接从Jadx中拷贝过来即可

然后就开始构造参数信息了，这里为了简单，先把那些公共的参数信息写死，比如设备的sid，aid，版本号等信息：

```

public class DouyinUtils {

    private final static String GETDATA_JSON_URL = "https://api.amemv.com/aweme/v1/feed/";

    public static String getEncryptUrl(){
        String url = null;
        int time = (int)(System.currentTimeMillis()/1000);
        try{
            HashMap<String,String> paramsMap = getCommonParams();
            String[] paramsAry = new String[paramsMap.size()*2];
            int i = 0;
            for(Entry<String,String> entry : paramsMap.entrySet()){
                paramsAry[i] = entry.getKey();
                i++;
                paramsAry[i] = entry.getValue();
                i++;
            }

            paramsMap.put("count", "10");
            paramsMap.put("type", "0");
            paramsMap.put("retry_type", "no_retry");
            paramsMap.put("ts", ""+time);

            //这里需要注意这两个字段是进行分页请求功能，大致规则如下：
            /**
             * 第一次请求，这两个字段都是0
             * 第二次请求取第一次请求返回的json数据中的min_cursor字段，max_cursor不需要携带。
             * 第三次以及后面所有的请求都只带max_cursor字段，值为第一次请求返回的json数据中的max_cursor字段。
             */
            paramsMap.put("max_cursor", "0");
            paramsMap.put("min_cursor", "0");
        }
    }
}

```

这里是服务器的时间戳，需要截断后面三位，直接除以1000即可。

这里需要构造请求字段的数组信息，字段和字段值一次存储即可

注意这后面的五个参数是不参与上面的参数字段数组中的，可以通过之前的打印值分析即可。

这里看到他的大致请求分页数据功能实现了，不过这里不是本文的重点。后面文章会介绍

这里我们利用公众参数信息，构造请求字段数组信息，然后还有单独的几个字段值不能参与操作，可以通过之前的打印日志分析出来。当然时间戳也是服务器格式，和本地是少三位的，直接除以1000即可。下面为了简单直接把公众参数写死即可，当然后续优化就是把这些参数值进行动态获取填充即可：

```

/**
 * 公共参数
 * @return
 */
public static HashMap<String, String> getCommonParams(){
    HashMap<String, String> params = new HashMap<String,String>();
    params.put("iid", "16715863991");
    params.put("device_id", "40545321430");
    params.put("ac", "wifi");
    params.put("channel", "360");
    params.put("aid", "1128");
    params.put("app_name", "aweme");
    params.put("version_code", "159");
    params.put("version_name", "1.5.9");
    params.put("device_platform", "android");
    params.put("ssmix", "a");
    params.put("device_type", "MI+3");
    params.put("device_brand", "Xiaomi");
    params.put("os_api", "19");
    params.put("os_version", "4.4.4");
    params.put("uuid", "863970029764198");
    params.put("openudid", "b39d9675ee6af5b2");
    params.put("manifest_version_code", "159");
    params.put("resolution", "1080*1920");
    params.put("dpi", "480");
    params.put("update_version_code", "1592");
    params.put("app_type", "normal");
    return params;
}

```

这里为了方便操作，直接把这些公众参数都写死，后面需要动态获取设备信息和应用信息的。

构造好参数之后，然后就开始调用native函数，然后获取返回结果即可：


```

StringBuilder paramsSb = new StringBuilder();
for(String key : paramsMap.keySet()){
    paramsSb.append(key+"="+paramsMap.get(key)+"&");
}
String urlStr = GETDATA_JSON_URL + "?" + paramsSb.toString();
if(urlStr.endsWith("&")){
    urlStr = urlStr.substring(0, urlStr.length()-1);
}

Logger.Log("get data url:"+urlStr+",len:"+paramsAry.length);

String as_cp = UserInfo.getUserInfo(time, urlStr, paramsAry);

Logger.Log("get as_cp:"+as_cp);

String asStr = as_cp.substring(0, as_cp.length()/2);
String cpStr = as_cp.substring(as_cp.length()/2, as_cp.length());

url = urlStr + "&as="+asStr+"&cp="+cpStr;

```

参数构造
完了,就
开始调用
native函
数了

到这里,我们构造的工作就完成了,下面就开始直接运行吧,不过可惜的是,没有这么顺利,直接运行闪退了:

```

cn.wjdiankong.... dalvikvm Trying to load lib /data/app-lib/cn.wjdiankong.levideo-1/libuserinfo.so 0x41a
88770
cn.wjdiankong.... dalvikvm Added shared lib /data/app-lib/cn.wjdiankong.levideo-1/libuserinfo.so 0x41a88
770
system_process ActivityMan... Process cn.wjdiankong.levideo (pid 3940) has died.

```

加载so出现问题了

这里应该进行加载so出现问题了,那么我在回过头看看我们是不是有些环境没初始化,看看UserInfo类中是不是还有一些初始化方法没调用:

```

public class UserInfo {
    public static native void getPackage(String str);

    public static native String getUserInfo(int i, String str, String[] strArr);

    public static native int initUser(String str);
    public static native void setAppId(int i);
}

```

这两个函数得调用一下

这里看到的确有两个方法有点可疑,全局查看这两个方法的调用地方:

```

public static native void setAppId(int i);

```

Usage search

Usage for: com.ss.android.common.appllog.UserInfo.setAppId(int) : void

Node	Code
com.ss.android.common.appllog.UserInfo.setAppId(int) : void	public static native void setAppId(int i);
com.ss.android.ugc.aweme.app.d.onCreate() : void	UserInfo.setAppId(2);

这里的appid直接写死的是2

看到全局中就一个地方调用了setAppId方法,而且值就是写死的为2,另一个方法initUser就有点麻烦了,不过还是万能的hook大法,直接hook这个方法打印他的参数信息即可:

```

XposedHelpers.findAndHookMethod("com.ss.android.common.appllog.UserInfo",
    loadPackageParam.classLoader, "initUser", String.class, new XC_MethodHook()
    @Override
    protected void afterHookedMethod(MethodHookParam param) throws Throwable {
        super.afterHookedMethod(param);
        try{
            Log.i("jw", "initUser args:"+param.args[0]);
            Log.i("jw", "initUser result:"+param.getResult());

            throw new NullPointerException();

        }catch(Exception e){
            Log.i("jw", "hook err:"+Log.getStackTraceString(e));
        }
    }
});

```

运行模块，查看打印的日志信息：

```
com.ss.android... jw load class:com.ss.android.common.applog.UserInfo
com.ss.android... jw initUser args:a3668f0afac72ca3f6c1697d29e0e1bb1fef4ab0285319b95ac39fa42c38d05f
com.ss.android... jw initUser result:0
```

<http://blog.csdn.net/jiangwei0910410003> 直接拷贝这个值到demo工程中即可

看到了，参数信息一致都是这个字符串信息，直接拷贝出来赋值调用即可：

```
try{
    System.loadLibrary("userinfo");
}catch(Exception e){
    Logger.log("load so err:"+Log.getStackTraceString(e));
}

UserInfo.setAppId(2);

int result = UserInfo.initUser("a3668f0afac72ca3f6c1697d29e0e1bb1fef4ab0285319b95ac39fa42c38d05f");
Logger.log("init user result:"+result);
```

<http://blog.csdn.net/jiangwei0910410003>

三、IDA调试so文件

在次运行，发现可惜了，还是报错，而且诡异的是日志没打印，也就是说setAppId和initUser函数没有调用就退出了，那么就会想到？是不是so中的JNI_OnLoad函数中做了一些判断逻辑呢？直接用IDA打开这个so文件即可：

```
1 signed int __fastcall JNI_OnLoad(int a1, int a2)
2 {
3     int v2; // r5@22
4     int v3; // r4@25
5     int v4; // r4@25
6     int v5; // r4@25
7     int v6; // r4@25
8     int v7; // r4@25
9     int v8; // r4@25
10    int v10; // [sp+0h] [bp-60h]@4
11    int v11; // [sp+8h] [bp-58h]@1
12    _JavaVM *v12; // [sp+Ch] [bp-54h]@1
13    int v13; // [sp+14h] [bp-4Ch]@10
14    int v14; // [sp+18h] [bp-48h]@22
15    int v15; // [sp+1Ch] [bp-44h]@25
16    char v16; // [sp+20h] [bp-40h]@1
17    char v17; // [sp+28h] [bp-38h]@1
18    int v18; // [sp+30h] [bp-30h]@1
19    int v19; // [sp+34h] [bp-2Ch]@1
20    char v20; // [sp+38h] [bp-28h]@7
21    int v21; // [sp+40h] [bp-20h]@1
22    int v22; // [sp+44h] [bp-1Ch]@7
23    char v23; // [sp+48h] [bp-18h]@1
24
25    v12 = (_JavaVM *)a1;
26    v11 = a2;
27    std::unique_lock<std::mutex>::unique_lock(&v23, &unk_63C8C);
28    sub_38FC0(&v16);
29    g_vm = v12;
30    sub_38FC0(&v17);
31    std::chrono::operator-<std::chrono::_V2::system_clock,std::chr
32        &v21,
33        &v17,
34        &v16);
35    std::chrono::duration<double,std::ratio<111,111>>::duration<lor
36    if ( (unsigned __int8)sub_142B0(v18, v19) ^ 1 )
37    {
38        std::unique_lock<std::mutex>::unlock(&v23);
39        exit(1);
40    }
41    sub_38FC0(&v10);
42    *(QWORD *)&v16 = *(QWORD *)&v10;
43    if ( _JavaVM::GetEnv(g_vm, &g_env, 65540) != 0 )
44    {
45        std::unique_lock<std::mutex>::unlock(&v23);
46        exit(1);
47    }
48 }
```

<http://blog.csdn.net/jiangwei0910410003>

F5查看对应的C代码，会发现他内部虽然9个判断，然后直接exit退出了。所以我们要想调用这个so，得先过了这几个判断了。

打开之后找到JNI_OnLoad函数，F5查看他的C代码，发现果然内部有很多判断，然后直接调用exit函数退出了。所以如果成功的调用它的so方法，得先过了他的这些判断了，这里就要开始进行调试操作了，其实这里可以直接静态分析有一个t...的方法，但是为了给大家介绍动态调试so技巧，就多走点弯路吧，后面再说一下粗暴简单的技巧。

下面就开始进行调试so文件了，关于IDA调试so文件，我之前已经写过一篇非常详细的文章了：[Android中IDA动态调试文件详解](#)；这里不会在详细介绍具体步骤了，直接上手干：**第一步：拷贝IDA安装目录下的android_server文件到设备目录下**

第二步：运行android_server命令

```

D:\>adb shell
*7*[r*[999;999H*[6n
*8shell@pisces:/ $ su
su
*7*[r*[999;999H*[6n
*8root@pisces:/ # cd /data/local/tmp
cd /data/local/tmp
root@pisces:/data/local/tmp # ./android_server
./android_server
IDA Android 32-bit remote debug server(ST) v1.19. Hex-Rays (c) 2004-2015
Listening on port #23946...

```

第三步：转发端口和用debug模式打开应用

```
D:\>adb forward tcp:23946 tcp:23946
```

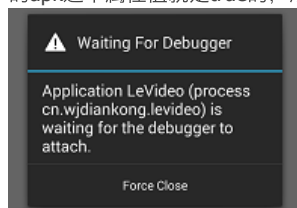
```

D:\>adb shell am start -D -n cn.wjdiankong.levideo/.activity.MainActivity
Starting: Intent { cmp=cn.wjdiankong.levideo/.activity.MainActivity }

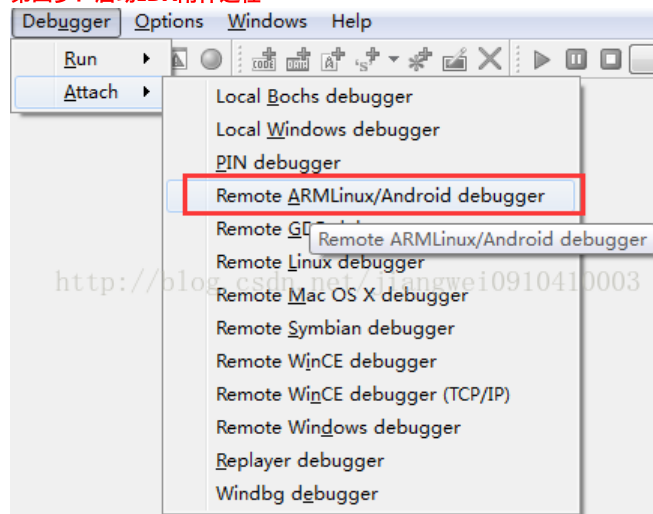
```

```
D:\>
```

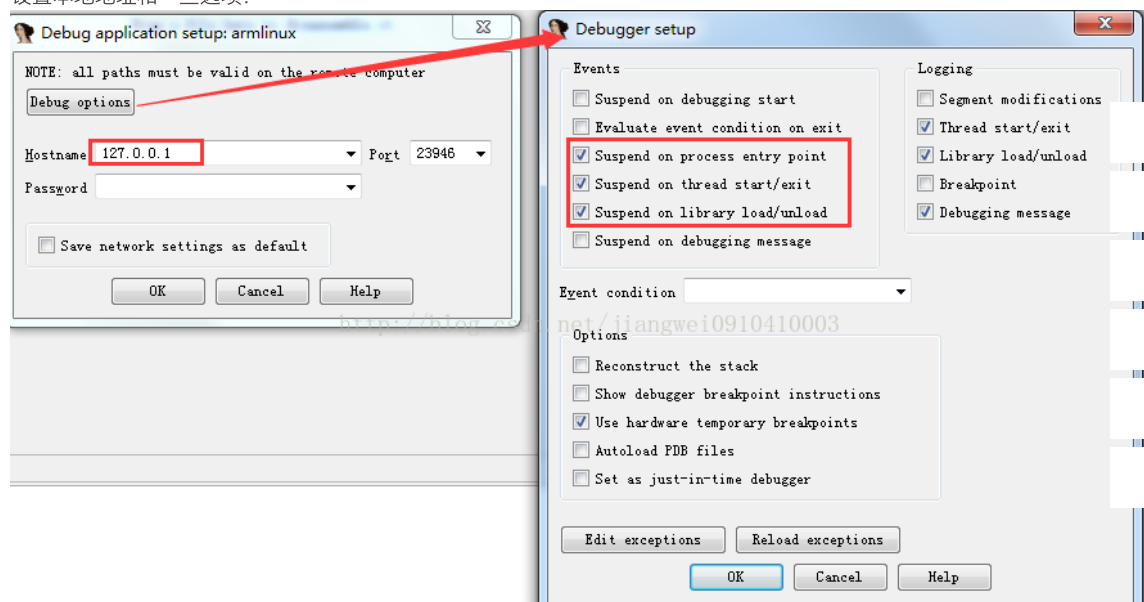
这里有同学好奇为什么不需要修改xml中的debug属性呢？因为我调试的是我自己的demo工程，而Eclipse默认签名打包出来的apk这个属性值就是true的，所以不需要进行修改了。



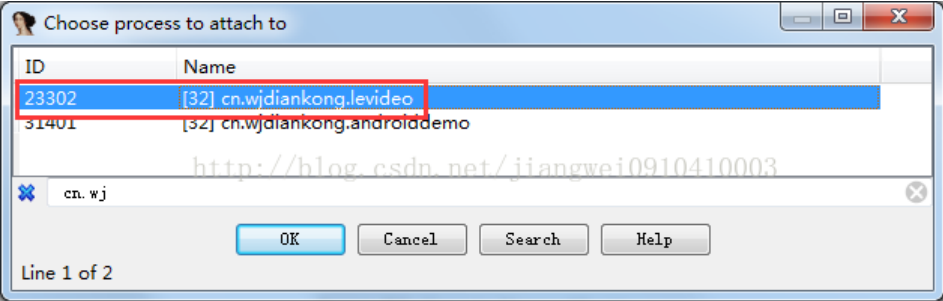
第四步：启动IDA附件进程



设置本地地址和一些选项：



因为现在已经知道需要调试JNI_OnLoad函数，所以需要设置挂起load函数处，然后选择调试的应用进程：



第五步：查看调试端口连接调试器

com.cyanogenmod.trebuchet	27310		8645
cn.wjdiankong.levideo	25680		8647 / 8700
com.ss.android.ugc.live.pushservice	28375		8648
?	31283		8649

在Eclipse中的DDMS中查看有红色小蜘蛛的调试应用端口号是8647，然后连接调试器：

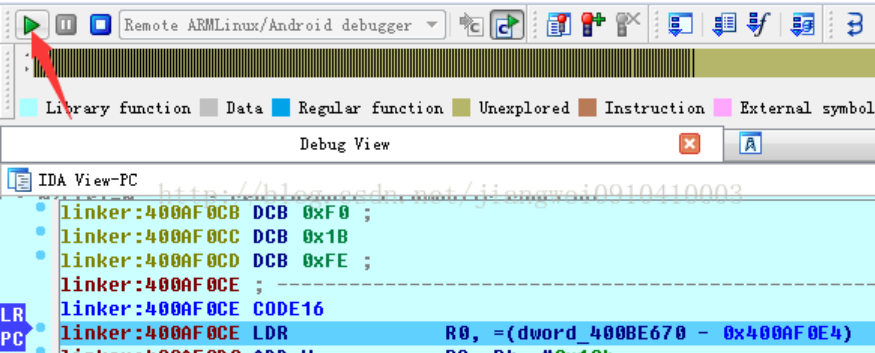
```
C:\Users\jiangwei1-g>adb forward tcp:23946 tcp:23946

C:\Users\jiangwei1-g>adb shell am start -D -n cn.wjdiankong.levideo/.activity.MainActivity
Starting: Intent { cmp=cn.wjdiankong.levideo/.activity.MainActivity }

C:\Users\jiangwei1-g>jdb -connect com.sun.jdi.SocketAttach:hostname=127.0.0.1,port=8647
设置未捕获的java.lang.Throwable
设置延迟的未捕获的java.lang.Throwable
正在初始化jdb...
>
>
```

这里一定要注意端口正确，不然链接操作的。

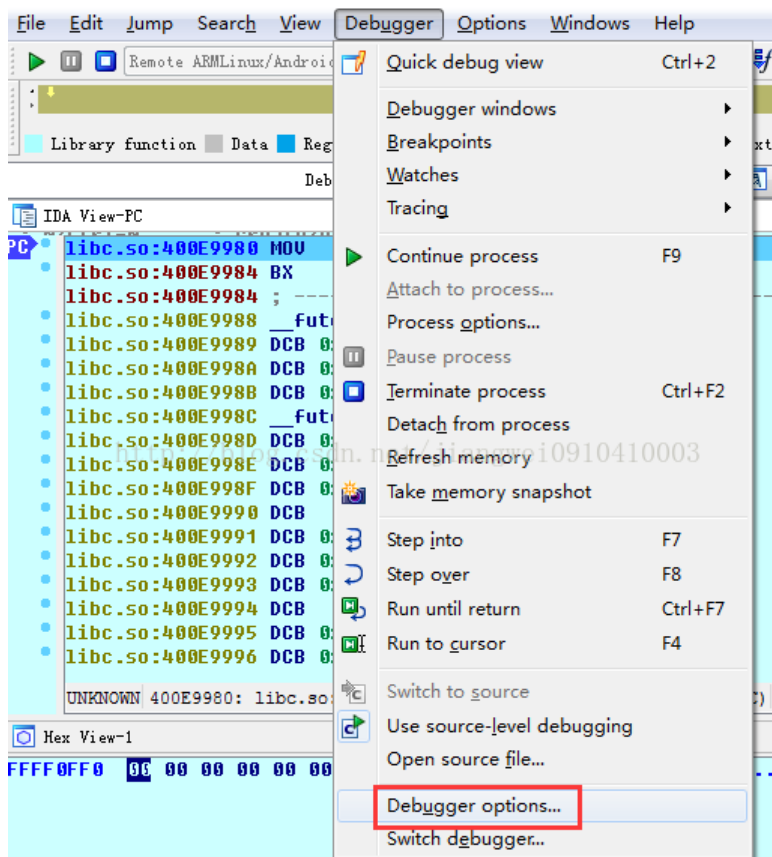
第六步：点击IDA中的运行按钮，或者F9快捷键



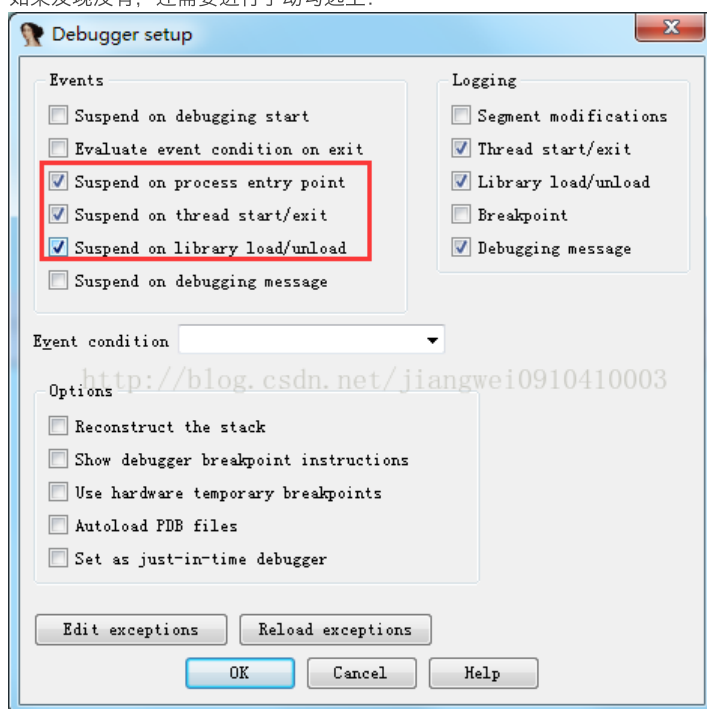
这时候发现红色蜘蛛变成绿色了，而且调试对话框也没有了。这时候就进入调试页面了：

com.android.deskclock	22286		8644
com.cyanogenmod.trebuchet	27310		8645
cn.wjdiankong.levideo	25680		8647 / 8700
com.ss.android.ugc.live.pushservice	28375		8648
?	31283		8649

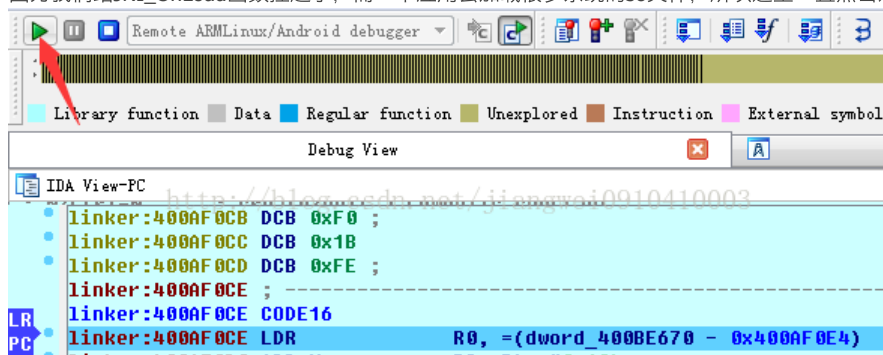
为了安全起见，再一次查看debug选项有没有挂起load函数：



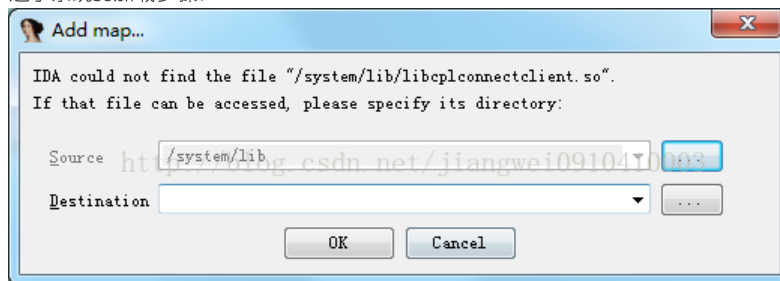
如果发现没有，还需要进行手动勾选上：



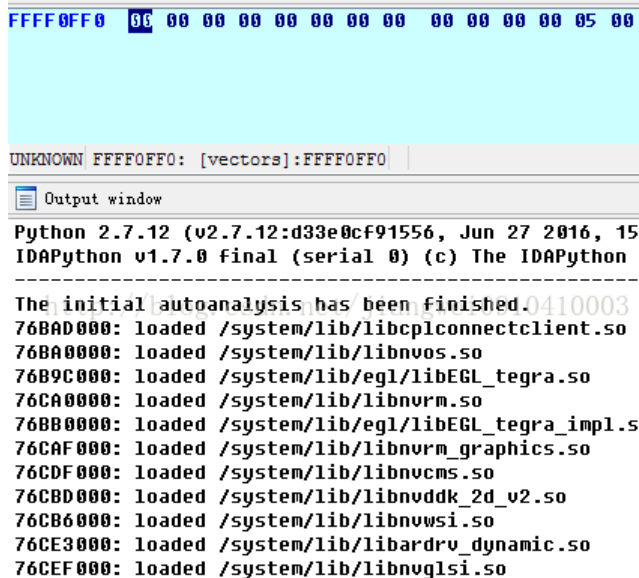
因为我们给JNI_OnLoad函数挂起了，而一个应用会加载很多系统的so文件，所以这里一直点击运行或者F9：



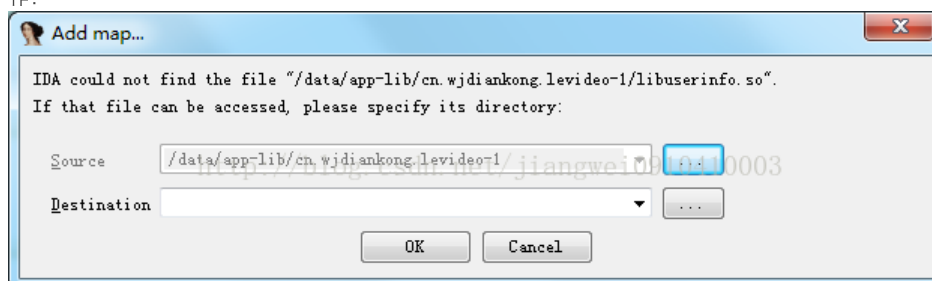
过了系统so加载步骤:



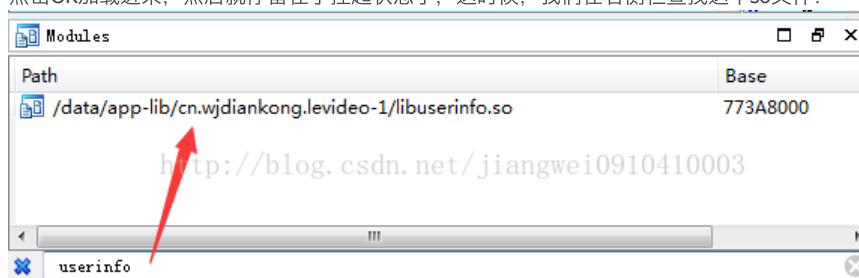
这些都是系统的so文件加载, 所以一路往下都直接运行越过调试即可:



可以看到这里会加载很多系统的so文件, 当我们点击应用的按钮, 加载自己的libuserinfo.so文件的时候才开始进行调试工作:



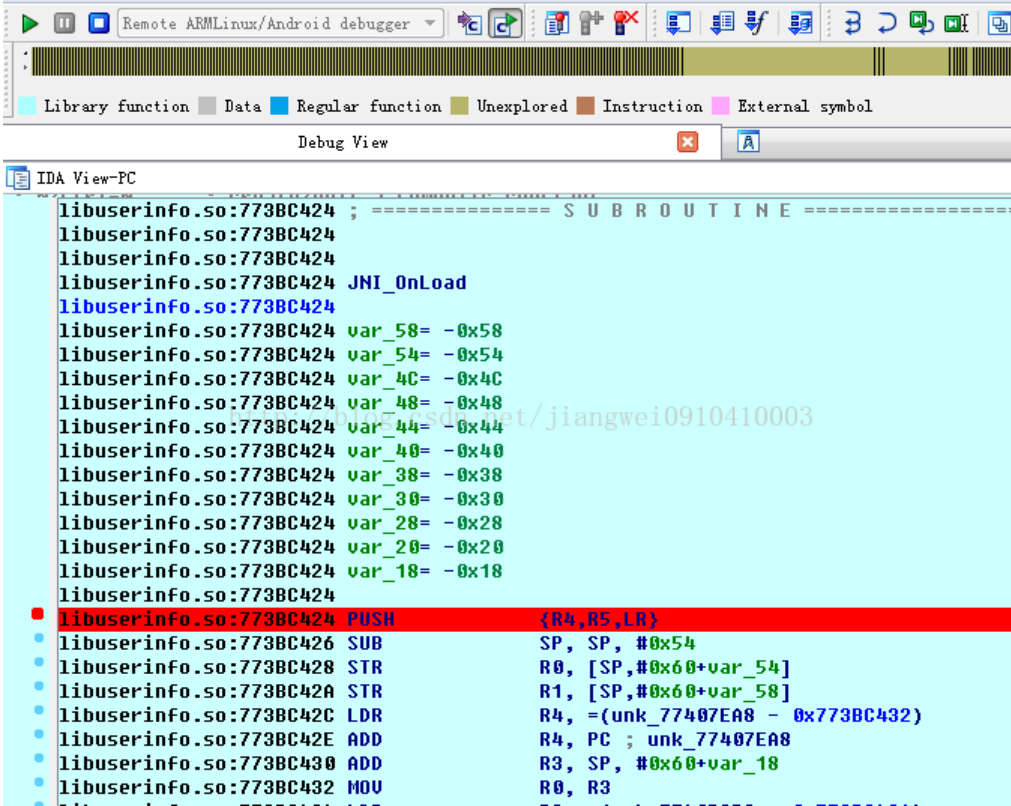
点击OK加载进来, 然后就停留在了挂起状态了, 这时候, 我们在右侧栏查找这个so文件:



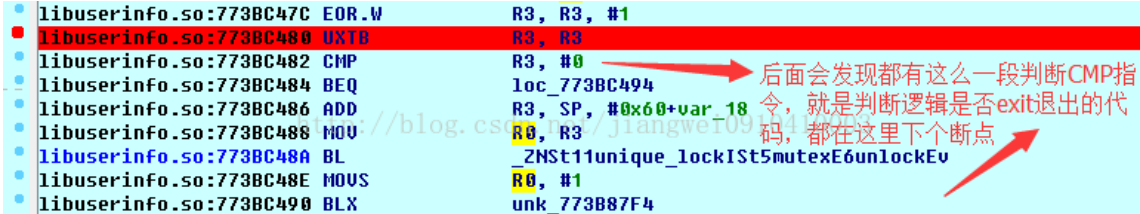
然后点击, 继续查找他的JNI_OnLoad函数:



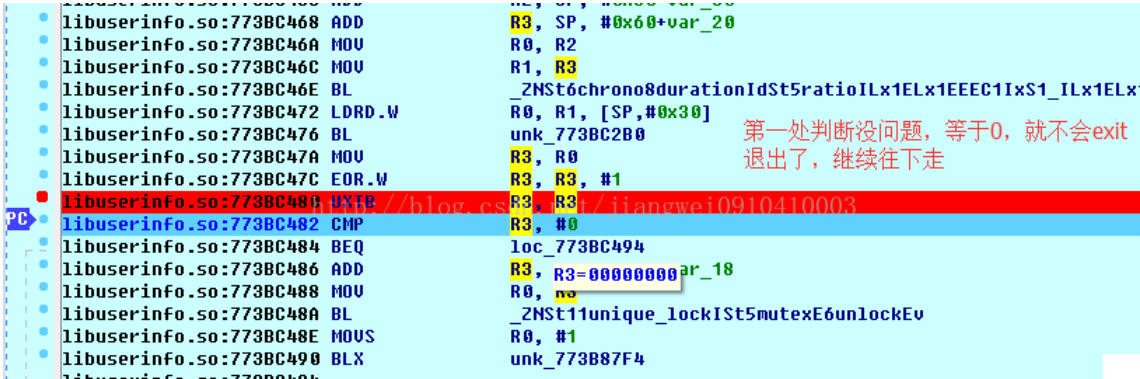
点击进入JNI_OnLoad函数处, 下个断点:



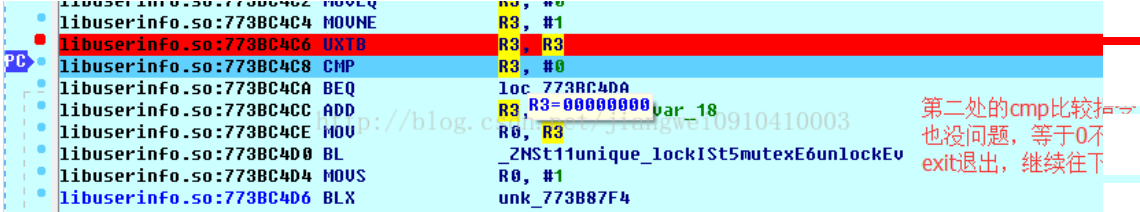
因为我们在之前静态分析了这个函数内部有很多个exit函数，为了好定位是哪个地方exit了，所以在每个exit函数之前下个断点，来判定退出逻辑，这里下断点有技巧，因为是if语句，所以在arm指令中肯定就是CMP指令之后的BEQ跳转，所以在每个CMP指令下个断点即可，这里发现了9个地方，所以下了断点也很多，慢慢分析：



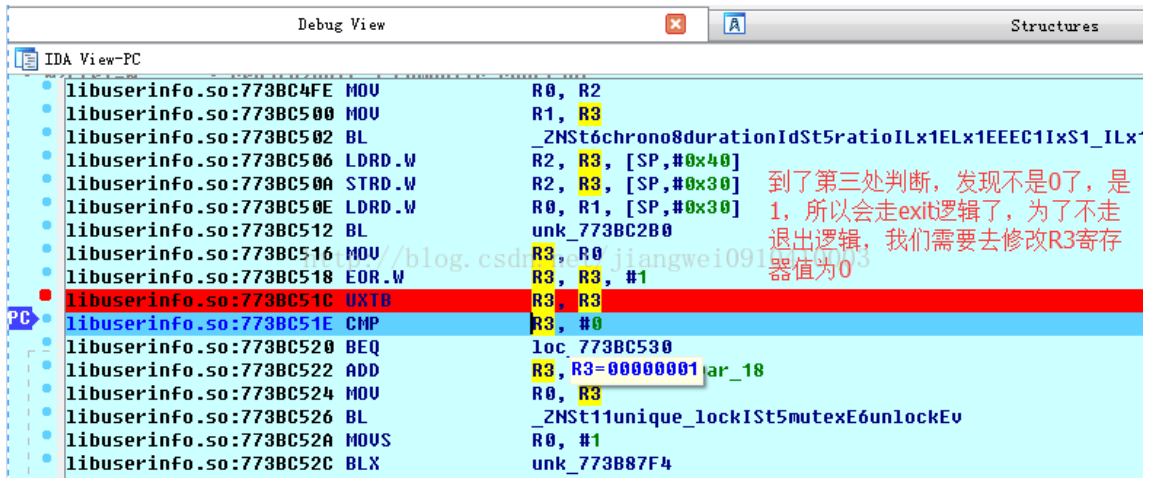
第一处的CMP指令下个断点，然后运行到此处，查看R3寄存器值是否为0：



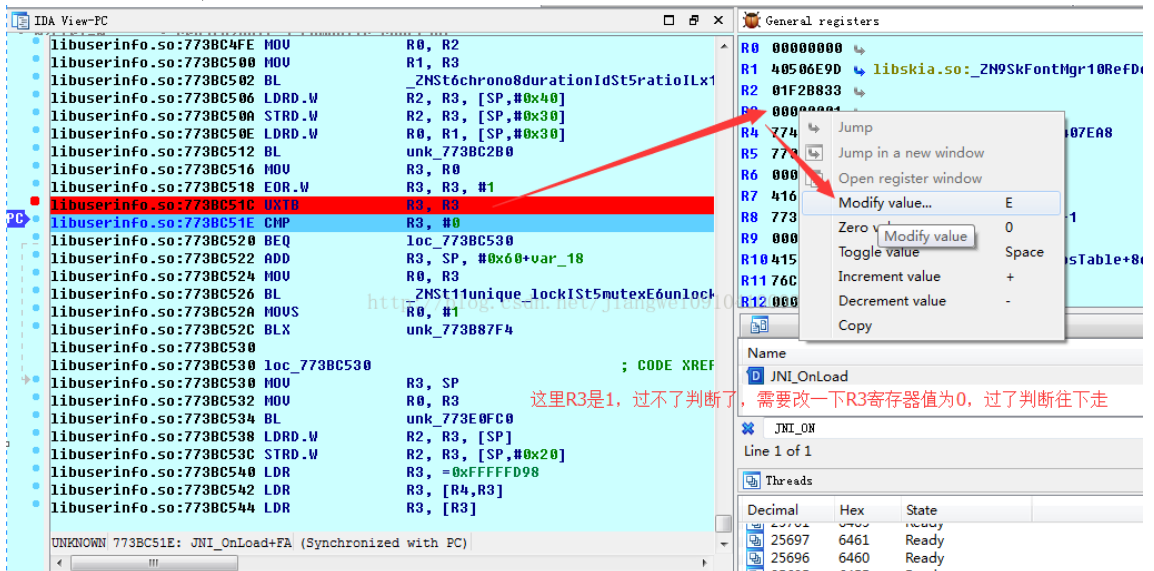
是0，那么第一处exit就没问题，接着往下走，直接按F9到下一个CMP判断指令断点处：



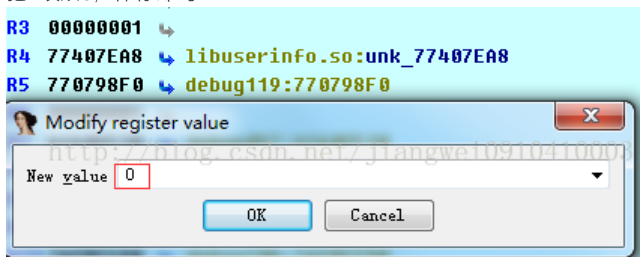
发现第二处的CMP中的R3寄存器值也是0，所以也没问题，直接F9到下一个断点：



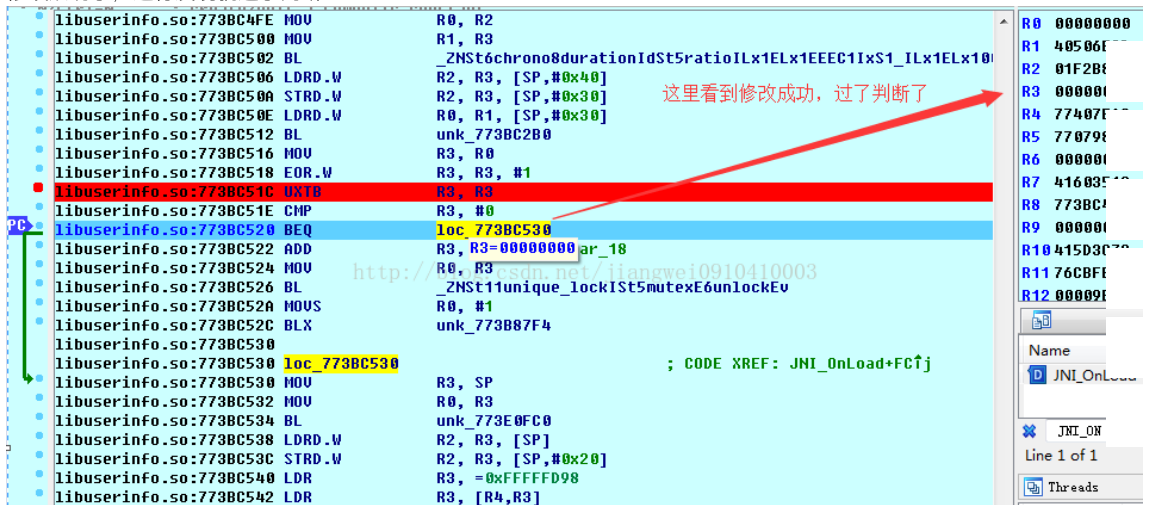
到了第三处判断发现R3寄出去你的值不是0了, 而是1, 所以为了继续能够往下走, 就修改R3寄存器值即可, 在右侧栏的寄存器中右击R3寄存器, 然后点击修改值:



把1改成0, 保存即可:



修改成功了, 运行发现就过了判断:



就继续往下走, 到下一个断点:


```

libuserinfo.so:773BC542 LDR      R3, [R4,R3]
libuserinfo.so:773BC544 LDR      R3, [R3]
libuserinfo.so:773BC546 MOV      R0, R3
libuserinfo.so:773BC548 BL       unk_773BC91C
libuserinfo.so:773BC54C MOV      R3, R0
libuserinfo.so:773BC54E STR      R3, [SP,#0x60+var_4C]
libuserinfo.so:773BC550 MOV      R3, SP
libuserinfo.so:773BC552 MOV      R0, R3
libuserinfo.so:773BC554 BL       loc_773E0FC0

```

悲剧来了，到这里就出现问题了，最终判断这个BL的函数有问题

这里有问题了，我们如果直接F9到第四处CMP的话，发现直接退出调试了，说明在3和4处判断中间有问题了，最终发现是这个跳转指令出现的问题，这里需要多次单步调试F7键，定位出现问题的地方了，我们进入这个跳转地址：

```

libuserinfo.so:773BC91C loc_773BC91C ; CODE XREF:
libuserinfo.so:773BC91C PUSH     {R4,R5,LR}
libuserinfo.so:773BC91E SUB      SP, SP, #0xFC
libuserinfo.so:773BC920 STR      R0, [SP,#4]
libuserinfo.so:773BC922 LDR      R4, =(unk_77407EA8 - 0x773BC928)
libuserinfo.so:773BC924 ADD      R4, PC, unk_77407EA8
libuserinfo.so:773BC926 LDR      R3, =0xFFFFFD78
libuserinfo.so:773BC928 LDR      R3, [R4,R3]
libuserinfo.so:773BC92A LDR      R3, [R3]
libuserinfo.so:773BC92C STR      R3, [SP,#0xF4]
libuserinfo.so:773BC92E LDR      R0, [SP,#4]
libuserinfo.so:773BC930 BL       unk_773BC894
libuserinfo.so:773BC934 MOV      R3, R0
libuserinfo.so:773BC936 STR      R3, [SP,#0x10]
libuserinfo.so:773BC938 LDR      R3, [SP,#0x10]
libuserinfo.so:773BC93A CMP      R3, #0
libuserinfo.so:773BC93C BNE      loc_773BC944
libuserinfo.so:773BC93E MOV      R3, #0xFFFFFFFF
libuserinfo.so:773BC942 B       loc_773BCA2E

```

这个BL出现问题

然后发现内部还有BL指令，出现问题了，继续深入查看：

```

libuserinfo.so:773BC894 loc_773BC894 ; CODE XREF: libuserinfo.so:JNI_OnLoad+50C1p
libuserinfo.so:773BC896 PUSH     {LR}
libuserinfo.so:773BC898 SUB      SP, SP, #0x1C
libuserinfo.so:773BC89A STR      R0, [SP,#4]
libuserinfo.so:773BC89C LDR      R0, [SP,#4]
libuserinfo.so:773BC89E LDR      R3, =(aConSsAndroidCo - 0x773BC8A2)
libuserinfo.so:773BC89E ADD      R3, PC, "com/ss/android/common/applog/GlobalCont"...
libuserinfo.so:773BC8A0 MOV      R1, R3
libuserinfo.so:773BC8A2 BL       _ZN7R3=libuserinfo.so:aConSsAndroidCo
libuserinfo.so:773BC8A6 MOV      R3, aConSsAndroidCo
libuserinfo.so:773BC8A8 STR      R3, DCB "com/ss/android/common/applog/GlobalContext",0
libuserinfo.so:773BC8AA LDR      R3,
libuserinfo.so:773BC8AC CMP      R3, #0
libuserinfo.so:773BC8AE BNE      loc_773BC8B4
libuserinfo.so:773BC8B0 MOVUS   R3, #0
libuserinfo.so:773BC8B2 B       loc_773BC906
libuserinfo.so:773BC8B4 ;
libuserinfo.so:773BC8B4 loc_773BC8B4 ; CODE XREF: libuserinfo.so:JNI_OnLoad+40A7j
libuserinfo.so:773BC8B4 LDR      R0, [SP,#4]
libuserinfo.so:773BC8B6 LDR      R1, [SP,#0xC]
libuserinfo.so:773BC8B8 LDR      R3, =(aGetContext - 0x773BC8BE)
libuserinfo.so:773BC8BA ADD      R3, PC, "getContext"
libuserinfo.so:773BC8BC MOV      R2, R3
libuserinfo.so:773BC8BE LDR      R3, =(aLandroidConten - 0x773BC8C4)
libuserinfo.so:773BC8C0 ADD      R3, PC, "()Landroid/content/Context;"
libuserinfo.so:773BC8C2 BL       _ZN7_JNIEnv17GetStaticMethodIDEP7_jclassPKcS3_
libuserinfo.so:773BC8C6 MOV      R3, R0
libuserinfo.so:773BC8C8 STR      R3, [SP,#0x10]
libuserinfo.so:773BC8CA LDR      R3, [SP,#0x10]
libuserinfo.so:773BC8CC CMP      R3, #0
libuserinfo.so:773BC8CE BNE      loc_773BC8DC
libuserinfo.so:773BC8D0 LDR      R0, [SP,#4]
libuserinfo.so:773BC8D2 LDR      R1, [SP,#0xC]
libuserinfo.so:773BC8D4 BL       _ZN7_JNIEnv14DeleteLocalRefEP8_jobject

```

这个函数出问题了，不过在这里发现问题的原因了，可以看到这里大致逻辑应该是底层用反射访问Java层的com.ss.android.common.applog.GlobalContext类的getContext方法获取全局Context变量，但是我们Java层没有这个类，所以报null错误，直接退出了。

看到了，这里发现问题的原因了，有一个GlobalContext类，这个类应该是Java层的，native层应该用反射机制获取全局的Context变量，我们直接去Jadx中搜索这个类：

```

package com.ss.android.common.applog;

import android.content.Context;

public class GlobalContext {
    private static Context mContext;

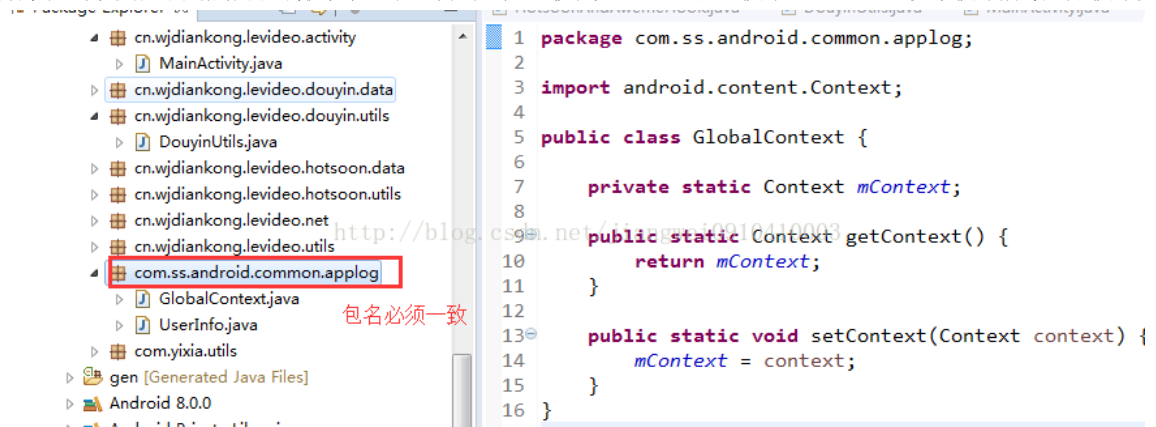
    public static Context getContext() {
        return mContext;
    }

    public static void setContext(Context context) {
        mContext = context;
    }
}

```

这里是用来管理全局的Context变量的

那么问题就清楚了，因为我们的demo工程中压根没这个类，所以native中获取全局context变量就失败了，所以就exit失败退出了，解决办法也简单，直接在demo工程中构造这个类，然后在Application中初始化context即可：



构造的时候一定要注意包名一致，然后在demo中的Application类进行设置context即可：

```
public class MainApplication extends Application{

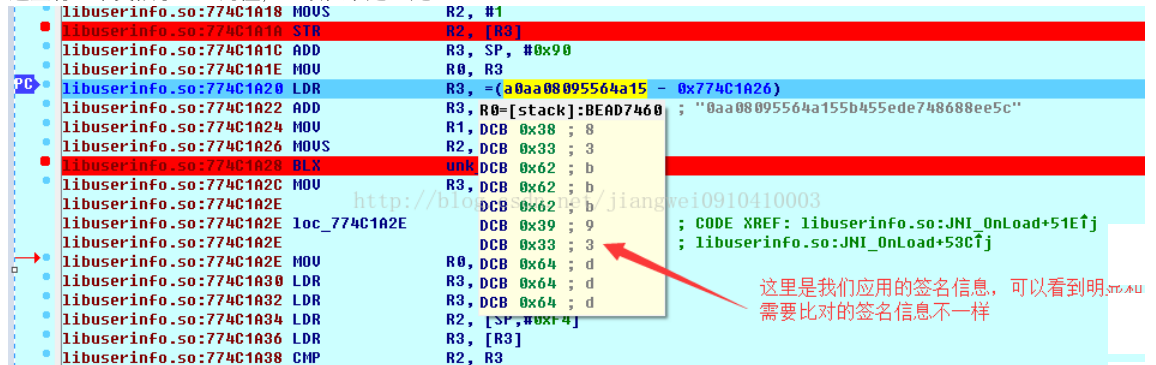
    @Override
    public void onCreate() {
        super.onCreate();
        GlobalContext.setContext(getApplicationContext());
    }
}
```

在demo中的Application中进行设置Context值

然后再次运行项目，可惜还是不行，所以还得进行调试JNI_OnLoad函数了，方法步骤和上面类似，不多说了，不过这里应该是过了上面的Context获取失败的问题了，继续往下走，发现了重要信息了：



这里有一个类似于MD5的值，继续往下走BLX处：



看到R0寄存器中的值，发现是demo应用的签名信息，继续往下走BLX看看：

```
libuserinfo.so:774C4714 loc_774C4714 ; CODE XREF: libuserinfo.so:JNI_OnLoad+520␣p
libuserinfo.so:774C4714 PUSH {LR}
libuserinfo.so:774C4716 SUB SP, SP, #0x3C
libuserinfo.so:774C4718 STR R0, [SP, #0xC]
libuserinfo.so:774C471A STR R1, [SP, #8]
libuserinfo.so:774C471C LDR R0, [SP, #0xC]
libuserinfo.so:774C471E LDR R1, [SP, #8]
libuserinfo.so:774C4720 BL _ZN7_JNIEnv14GetObjectClassEP8_jobject
libuserinfo.so:774C4724 MOV R3, R0
libuserinfo.so:774C4726 STR R3, [SP, #0x14]
libuserinfo.so:774C4728 LDR R0, [SP, #0xC]
libuserinfo.so:774C472A LDR R1, [SP, #0x14]
libuserinfo.so:774C472C LDR R3, =(aGetPackageName - 0x774C4732)
libuserinfo.so:774C472E ADD R3, PC ; "getPackageName"
libuserinfo.so:774C4730 MOV R2, R3
libuserinfo.so:774C4732 LDR R3, =(aLandroidConten - 0x774C4738)
libuserinfo.so:774C4734 ADD R3, PC ; "()Landroid/content/pm/PackageManager;"
libuserinfo.so:774C4736 BL _ZN7_JNIEnv11GetMethodIDEP7_jclassPKcS3_
libuserinfo.so:774C473A MOV R3, R0
libuserinfo.so:774C473C STR R3, [SP, #0x18]
libuserinfo.so:774C473E LDR R0, [SP, #0xC]
libuserinfo.so:774C4740 LDR R1, [SP, #8]
libuserinfo.so:774C4742 LDR R2, [SP, #0x18]
libuserinfo.so:774C4744 BL _ZN7_JNIEnv16CallObjectMethodEP8_jobjectP10_jmethodIdz
libuserinfo.so:774C4746 MOV R0, R0
```

这个方法中会通过上次Java代码的Context实例获取签名信息

这里看到大致清楚了，是获取应用签名信息，也就是签名校验了：

```
libuserinfo.so:774C47C6 LDR R3, =(aSignatures - 0x774C47CC)
libuserinfo.so:774C47C8 ADD R3, PC ; "signatures"
libuserinfo.so:774C47CA MOV R2, R3
libuserinfo.so:774C47CC LDR R3, =(aLandroidCont_0 - 0x774C47D2)
libuserinfo.so:774C47CE ADD R3, PC ; "[Landroid/content/pm/Signature;"
libuserinfo.so:774C47D0 BL _ZN7_JNIEnv10GetFieldIDEP7_jclassPKcS3_
libuserinfo.so:774C47D4 MOV R3, R0
libuserinfo.so:774C47D6 STR R3, [SP, #0x28]
libuserinfo.so:774C47D8 LDR R0, [SP, #0xC]
libuserinfo.so:774C47DA LDR R1, [SP, #0x24]
libuserinfo.so:774C47DC LDR R2, [SP, #0x28]
libuserinfo.so:774C47DE BL _ZN7_JNIEnv14GetObjectFieldEP8_jobjectP9_jfieldID
libuserinfo.so:774C47E2 MOV R3, R0
libuserinfo.so:774C47E4 STR R3, [SP, #0x2C]
libuserinfo.so:774C47E6 LDR R0, [SP, #0xC]
libuserinfo.so:774C47E8 LDR R1, [SP, #0x2C]
libuserinfo.so:774C47EA MOVS R2, #0
libuserinfo.so:774C47EC BL _ZN7_JNIEnv21GetObjectArrayElementEP13_jobjectArrayi
libuserinfo.so:774C47F0 MOV R3, R0
libuserinfo.so:774C47F2 STR R3, [SP, #0x30]
libuserinfo.so:774C47F4 LDR R0, [SP, #0xC]
libuserinfo.so:774C47F6 LDR R1, [SP, #0x14]
libuserinfo.so:774C47F8 BL _ZN7_JNIEnv14DeleteLocalRefEP8_jobject
libuserinfo.so:774C47FC LDR R0, [SP, #0xC]
libuserinfo.so:774C47FE LDR R1, [SP, #0x30]
libuserinfo.so:774C4800 BL _ZN7_JNIEnv14GetObjectClassEP8_jobject
libuserinfo.so:774C4804 MOV R3, R0
libuserinfo.so:774C4806 STR R3, [SP, #0x14]
libuserinfo.so:774C4808 LDR R0, [SP, #0xC]
libuserinfo.so:774C480A LDR R1, [SP, #0x14]
libuserinfo.so:774C480C LDR R3, =(aToCharsstring - 0x774C4812) |
libuserinfo.so:774C480E ADD R3, PC ; "toCharsString"
libuserinfo.so:774C4810 MOV R2, R3
libuserinfo.so:774C4812 LDR R3, =(aLjavaLangStrin - 0x774C4818)
libuserinfo.so:774C4814 ADD R3, PC ; "()Ljava/lang/String;"
libuserinfo.so:774C4816 BL _ZN7_JNIEnv11GetMethodIDEP7_jclassPKcS3_
```

好吧，在这一处判断exit函数中，应该是通过反射获取Java层的context值，然后在获取应用的签名信息和已经保存的原始抖音签名信息做对比，如果不对就退出了。那么过签名校验也很方便，直接利用我之前的kstools原理，把hook代码代码拷贝到工程中，拦截获取签名信息，然后返回正确的签名信息即可：

```
public class MainApplication extends Application{

    @Override
    public void onCreate() {
        super.onCreate();
        ServiceManagerWrapper.hookPMS(this.getApplicationContext());

        GlobalContext.setContext(getApplicationContext());
    }
}
```

一定要在第一行调用hook代码，这样才能有效果

具体的hook代码去看我的kstools实现原理即可：[Android中自动爆破签名信息工具kstools](#)；然后hook代码一定要在Application的第一行代码调用，不然没有效果的。这样操作完成之后，其实已经出数据了，不过为了能够进入加密函数进行调试分析具体的加密算法，我们继续调试JNI_OnLoad函数：

```

libuserinfo.so:774C45BA loc_774C45BA ; CODE XREF: JNI_OnLoad+186↑j
libuserinfo.so:774C45BA MOV R3, SP
libuserinfo.so:774C45BC MOV R0, R3
libuserinfo.so:774C45BE BL unk_774E8FC0
libuserinfo.so:774C45C2 LDRD.W R2, R3, [SP]
libuserinfo.so:774C45C6 STRD.W R2, R3, [SP, #0x20]
libuserinfo.so:774C45CA LDR R3, =(unk_77513CB8 - 0x774C45D0)
libuserinfo.so:774C45CC ADD R3, PC ; unk_77513CB8
libuserinfo.so:774C45CE MOV R0, R3
libuserinfo.so:774C45D0 BL 217be_attached_checkR1
libuserinfo.so:774C45D4 MOV R3, R0
libuserinfo.so:774C45D6 STR R3, [SP, #0x60+var_4C]
libuserinfo.so:774C45D8 MOV R3, SP
libuserinfo.so:774C45DA MOV R0, R3
libuserinfo.so:774C45DC BL unk_774E8FC0

```

这里有个函数很可疑，貌似在检查什么，下个断点，进去看看

继续往下走，会发现在第五个exit判断之后，有一个函数出问题了，就是这个BL，进入内部查看：

```

libuserinfo.so:774C2E0C 217be_attached_checkR1 ; CODE XREF: JNI_OnLoad+1AC↓p
libuserinfo.so:774C2E0C PUSH {R4,R5,LR}
libuserinfo.so:774C2E0E SUBW SP, SP, #0x824
libuserinfo.so:774C2E12 ADD R3, SP, #4
libuserinfo.so:774C2E14 STR R0, [R3]
libuserinfo.so:774C2E16 LDR R4, =(unk_7750FEA8 - 0x774C2E1C)
libuserinfo.so:774C2E18 ADD R4, PC ; unk_7750FEA8
libuserinfo.so:774C2E1A LDR R3, =0xFFFFFD78
libuserinfo.so:774C2E1C LDR R3, [R4,R3]
libuserinfo.so:774C2E1E LDR R3, [R3]
libuserinfo.so:774C2E20 STR.W R3, [SP, #0x81C]
libuserinfo.so:774C2E24 ADD R3, SP, #0xC
libuserinfo.so:774C2E26 MOV.W R2, #0x400
libuserinfo.so:774C2E2A STR R2, [R3]
libuserinfo.so:774C2E2C BLX unk_774C07A0
libuserinfo.so:774C2E30 MOV R2, R0
libuserinfo.so:774C2E32 ADD R3, SP, #0x10
libuserinfo.so:774C2E34 STR R2, [R3]
libuserinfo.so:774C2E36 ADD R2, SP, #0x1C
libuserinfo.so:774C2E38 ADD R3, SP, #0x10
libuserinfo.so:774C2E3A MOV R0, R2
libuserinfo.so:774C2E3C LDR R2, =(aProcDStatus - 0x774C2F42)
libuserinfo.so:774C2E3E ADD R2, PC ; "/proc/%d/status"
libuserinfo.so:774C2E40 MOV R1, R2
libuserinfo.so:774C2E42 LDR R2, [R3]
libuserinfo.so:774C2E44 BLX unk_774C074C
libuserinfo.so:774C2E48 ADD R3, SP, #0x1C
libuserinfo.so:774C2E4A MOV R0, R3
libuserinfo.so:774C2E4C LDR R3, =(unk_7750975C - 0x774C2E52)
libuserinfo.so:774C2E4E ADD R3, PC ; unk_7750975C
libuserinfo.so:774C2E50 MOV R1, R3
libuserinfo.so:774C2E52 BLX unk_774C074C
libuserinfo.so:774C2E56 MOV R2, R0
libuserinfo.so:774C2E58 ADD R3, SP, #0x14
libuserinfo.so:774C2E5A STR R2, [R3]
libuserinfo.so:774C2E5C ADD R3, SP, #0x14

```

看到这个字符串想到什么了？没错，这里应该是反调试的

哈哈，发现了这个字符串信息，弄过调试的同学大致都猜到了，这个是反调试操作，继续往下走：

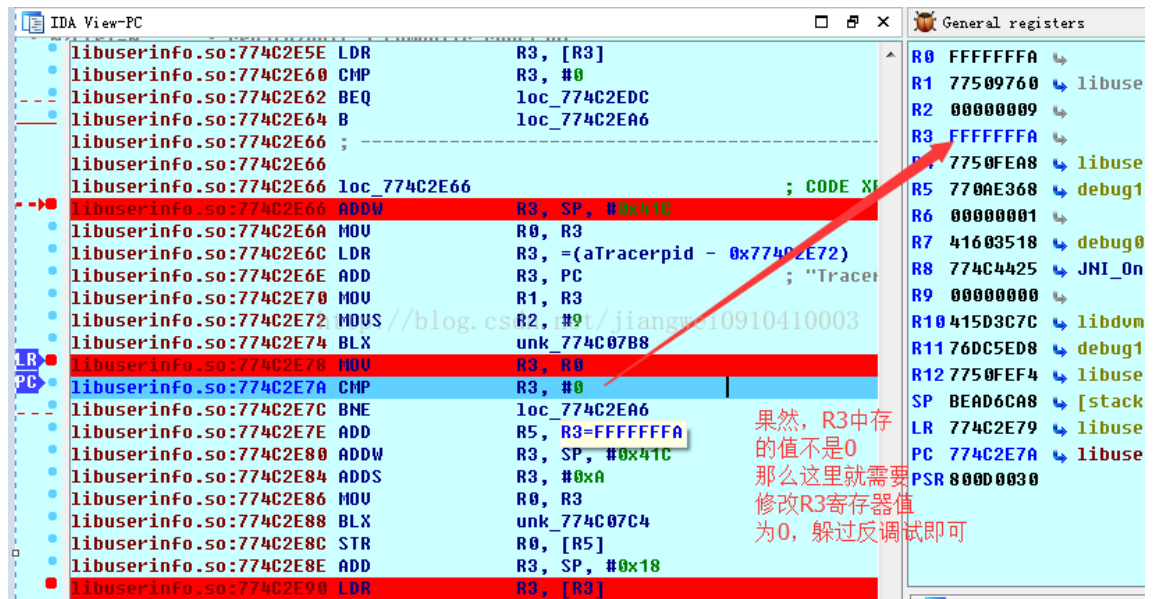
```

libuserinfo.so:774C2E64 B LOC_774C2E60
libuserinfo.so:774C2E66 ;
libuserinfo.so:774C2E66 ;
libuserinfo.so:774C2E66 loc_774C2E66 ; CODE XREF: libuserinfo.so:217be_attached_checkR1
libuserinfo.so:774C2E66 ADDW R3, SP, #0x41C
libuserinfo.so:774C2E6A MOV R0, R3
libuserinfo.so:774C2E6C LDR R3, =(aTracerpid - 0x774C2E72)
libuserinfo.so:774C2E6E ADD R3, PC ; "TracerPid"
libuserinfo.so:774C2E70 MOV R1, R3
libuserinfo.so:774C2E72 MOV R2, #9
libuserinfo.so:774C2E74 BLX unk_774C07B8
libuserinfo.so:774C2E78 MOV R3, R0
libuserinfo.so:774C2E7A CMP R3, #0
libuserinfo.so:774C2E7C BNE loc_774C2EA6
libuserinfo.so:774C2E7E ADD R5, SP, #0x18
libuserinfo.so:774C2E80 ADDW R3, SP, #0x41C
libuserinfo.so:774C2E84 ADDS R3, #0xA
libuserinfo.so:774C2E86 MOV R0, R3
libuserinfo.so:774C2E88 BLX unk_774C07C4
libuserinfo.so:774C2E8C STR R0, [R5]
libuserinfo.so:774C2E8E ADD R3, SP, #0x18
libuserinfo.so:774C2E90 LDR R3, [R3]
libuserinfo.so:774C2E92 CMP R3, #0
libuserinfo.so:774C2E94 BEQ loc_774C2EA4
libuserinfo.so:774C2E96 ADD R3, SP, #0x14
libuserinfo.so:774C2E98 LDR R0, [R3]
libuserinfo.so:774C2E9A BLX unk_774C07D0
libuserinfo.so:774C2E9E MOV.W R3, #0xFFFFFFFF
libuserinfo.so:774C2EA2 B loc_774C2E8

```

读取status文件中的TracerPid字段值判断是否为0

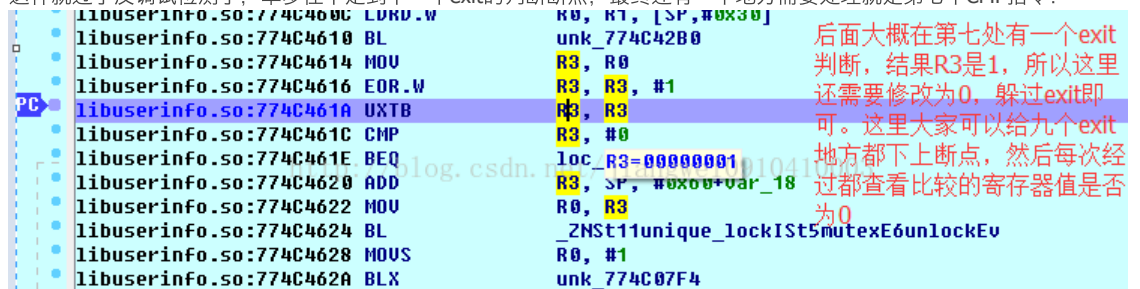
这里可以百分百确定是读取status文件中的TracerPid字段值来进行反调试检测了，给下面的两个CMP指令下个断点：



发现R3寄存器中的确不是0，所以为了过了反调试，直接修改R3寄存器值为0即可：



这样就过了反调试检测了，单步往下走到下一个exit的判断断点，最终还有一个地方需要处理就是第七个CMP指令：



处理方法直接修改R3寄存器中的值为0即可，就这样我们成功的过了JNI_OnLoad中的所有判断exit的地方了：


```
libuserinfo.so:774C466E UXTB R3, R3
libuserinfo.so:774C4670 CMP R3, #0
libuserinfo.so:774C4672 BEQ loc_774C4682
libuserinfo.so:774C4674 ADD R3, SP, #0x60+var_18
libuserinfo.so:774C4676 MOV R0, R3
libuserinfo.so:774C4678 BL _ZNSt11unique_lockISt5mutexE6unlockEv
libuserinfo.so:774C467C MOVS R0, #1
libuserinfo.so:774C467E BLX unk_774C07F4
libuserinfo.so:774C4682
libuserinfo.so:774C4682 loc_774C4682 ; CODE XREF: JNI
libuserinfo.so:774C4682 BL 219check_qemu_propertyv
libuserinfo.so:774C4686 MOV R4, R0
libuserinfo.so:774C4688 BL 215have_propertiesv
libuserinfo.so:774C468C MOV R3, R0
libuserinfo.so:774C468E ADD R4, R3
libuserinfo.so:774C4690 BL 216have_known_pipesv
libuserinfo.so:774C4694 MOV R3, R0
libuserinfo.so:774C4696 ADD R4, R3
libuserinfo.so:774C4698 BL 216have_known_filesu
libuserinfo.so:774C469C MOV R3, R0
libuserinfo.so:774C469E ADD R4, R3
libuserinfo.so:774C46A0 BL 221have_known_geny_filesu
libuserinfo.so:774C46A4 MOV R3, R0
libuserinfo.so:774C46A6 ADD R4, R3
libuserinfo.so:774C46A8 BL 217have_qemu_driversv
libuserinfo.so:774C46AC MOV R3, R0
libuserinfo.so:774C46AE ADD R4, R3
libuserinfo.so:774C46B0 BL 210check_vboxv
libuserinfo.so:774C46B4 MOV R3, R0
libuserinfo.so:774C46B6 ADD R3, R4
libuserinfo.so:774C46B8 STR R3, [SP, #0x60+var_44]
libuserinfo.so:774C46BA LDR R3, [SP, #0x60+var_44]
libuserinfo.so:774C46BC CMP R3, #3
libuserinfo.so:774C46BE BLE loc_774C46CE
```

到这里基本就结束了JNI_OnLoad函数的调试了，也过了几层校验，下面就直接进入加密函数了，我们也把so校验的工作都做完了

然后给加密函数getUserInfo下个断点：

Name	Address
D _Z6getUrliSt3mapISsSsSt4lessISsESaISt4pairIKSsSsEEE	774C14AC
D _Z7getUTF8P7_JNIEnvP7_jclassP8_jstring	774C3BE0
D Java_com_ss_android_common_applog_UserInfo_getUser...	774C3E30

getU

Line 3 of 3

直接点击进入即可：

```
libuserinfo.so:774C3E30 Java_com_ss_android_common_applog_UserInfo_getUserInfo
libuserinfo.so:774C3E30
libuserinfo.so:774C3E30 var_98= -0x98
libuserinfo.so:774C3E30 var_94= -0x94
libuserinfo.so:774C3E30 var_90= -0x90
libuserinfo.so:774C3E30 var_8C= -0x8C
libuserinfo.so:774C3E30 var_84= -0x84
libuserinfo.so:774C3E30 var_80= -0x80
libuserinfo.so:774C3E30 var_7C= -0x7C
libuserinfo.so:774C3E30 var_78= -0x78
libuserinfo.so:774C3E30 var_74= -0x74
libuserinfo.so:774C3E30 var_70= -0x70
libuserinfo.so:774C3E30 var_6C= -0x6C
libuserinfo.so:774C3E30 var_68= -0x68
libuserinfo.so:774C3E30 var_64= -0x64
libuserinfo.so:774C3E30 var_60= -0x60
libuserinfo.so:774C3E30 var_5C= -0x5C
libuserinfo.so:774C3E30 var_58= -0x58
libuserinfo.so:774C3E30 var_54= -0x54
libuserinfo.so:774C3E30 var_50= -0x50
libuserinfo.so:774C3E30 var_4C= -0x4C
libuserinfo.so:774C3E30 var_48= -0x48
libuserinfo.so:774C3E30 var_40= -0x40
libuserinfo.so:774C3E30 var_34= -0x34
libuserinfo.so:774C3E30 var_28= -0x28
libuserinfo.so:774C3E30 arg_0= 0
libuserinfo.so:774C3E30
R12 PC libuserinfo.so:774C3E30 PUSH {R4,R5,LR}
libuserinfo.so:774C3E32 SUB SP, SP, #0x8C
libuserinfo.so:774C3E34 STR R0, [SP, #0x98+var_8C]
libuserinfo.so:774C3E36 STR R1, [SP, #0x98+var_90]
libuserinfo.so:774C3E38 STR R2, [SP, #0x98+var_94]
libuserinfo.so:774C3E3A STR R3, [SP, #0x98+var_98]
libuserinfo.so:774C3E3C ADD R3, SP, #0x98+var_48
```

到这里，我们就顺利的进入了获取加密信息的函数了，不过这里因为这个函数代码太多了，我们其实没必要继续调试，解出算法了。直接去上次解决JNI_OnLoad中的校验工作，直接调用这个native方法获取加密内容即可。到这里我们也算是成功了。不过感兴趣的同学可以继续调试，这个函数已经没有任何校验工作了。可以大胆F7单步走了

四、加密数据结果输出

也成功到达了加密函数断点处，这里已经没有任何判断逻辑了，就是一个单纯的加密函数了，不过这里不在进行调试分析了，感兴趣的同学可以自己操作了，因为我们的目的达到了，就是成功的获取到了加密之后的数据了：

```
cn.wjdiankong.... levideo get as_cp:alb5876f421d497c7e7fd69f5e2cebf2cfe1
cn.wjdiankong.... levideo get data url:https://api.amemv.com/aweme/v1/feed/?version_name=1.5.9&ts=150980
5032&count=10&device_type=MI+3&iid=16715863991&app_type=normal&resolution=10
80*1920&aid=1128&type=0&app_name=aweme&max_cursor=0&device_platform=android&v
ersion_code=159&dpi=480&min_cursor=0&retry_type=no_retry&openudid=b39d9675ee6
af5b2&ssmix=asos_api=19&device_id=40545321430&device_brand=Xiaomi&manifest_ve
rsion_code=159&os_version=4.4.4&update_version_code=1592&ac=wifi&uuid=8639700
29764198&channel=360&as=alb5876f421d497c7e7fd69f5e2cebf2cfe1
cn.wjdiankong.... levideo list video data:maxcursor=-1978164542,mincursor=-1978164532,videolist:[videot
itle=,videoplayurl=https://aweme.snssdk.com/aweme/v1/play/?video_id=c5bc99764
23c4e959a1641287a89ef75&line=0&ratio=720p&media_type=4&vr_type=0,videodownloa
durl=https://aweme.snssdk.com/aweme/v1/play/?video_id=c5bc9976423c4e959a16412
87a89ef75&line=0&ratio=720p&watermark=1&media_type=4&vr_type=0,width=0,height
=0,coverimgurl=http://p1.pstatp.com/large/445500024c9c63344009.jpeg,musicname
=@李欣蕊! 创作的原声,musicimgurl=http://p3.pstatp.com/live/100x100/42dd00041d0bab8
e.jpeg,musicauthorname=李欣蕊!,authorname=,authorimgurl=https://p1.pstatp.com/aw
eme/100x100/3b5e0024342e010455be.jpeg,playcount=0, videotitle=,videoplayurl=h
ttps://aweme.snssdk.com/aweme/v1/play/?video_id=c5bc9976423c4e959a1641287a89e
f75&line=0&ratio=720p&media_type=4&vr_type=0,videodownloadurl=https://aweme.s
nssdk.com/aweme/v1/play/?video_id=c5bc9976423c4e959a1641287a89ef75&line=0&rat
io=720p&watermark=1&media_type=4&vr_type=0,width=0,height=0,coverimgurl=http:
//p1.pstatp.com/large/445500024c9c63344009.jpeg,musicname=@李欣蕊! 创作的原声,musi
cimgurl=http://p3.pstatp.com/live/100x100/42dd00041d0bab8fe28e.jpeg,musicauthorna
me=李欣蕊!,authorname=,authorimgurl=https://p1.pstatp.com/aweme/100x100/3b5e0024
342e010455be.jpeg,playcount=0, videotitle=,videoplayurl=https://aweme.snssdk.
com/aweme/v1/play/?video_id=c5bc9976423c4e959a1641287a89ef75&line=0&ratio=720
p&media_type=4&vr_type=0,videodownloadurl=https://aweme.snssdk.com/aweme/v1/p
lay/?video_id=c5bc9976423c4e959a1641287a89ef75&line=0&ratio=720p&watermark=1&
```

as和cp字段值成功获取
然后进行对半拆分即可

这个就是最终的
数据请求url了

这里我们也成功的拿到了
返回数据，这里解析了json
数据，后面会继续用到

看到了，我们成功的获取到了as和cp值，然后构造到请求url中，也成功的拿到了返回数据。我们这里多了一步解析json数据而已，原始的json数据是这样的：

```
cover_medium
url_list :
http://p3.pstatp.com/live/200x200/42de00071fdc884af1f4.jpeg
id_st 884af1f4
sche 8639886
owne 361
vr_type
label_top
url_list :
https://p3.pstatp.com/obj/c150000f34767e2cb56
https://pb9.pstatp.com/obj/c150000f34767e2cb56
https://pb3.pstatp.com/obj/c150000f34767e2cb56
uri : c150000f34767e2cb56
cha_list :
0
schema : aweme://aweme/challenge/detail?cid=1557765331532802
user_count : 0
cid : 1557765331532802
cha_name : 跳舞机
author
desc : 不知是单单的跳舞哦而是有挑战性的
share_info
share_weibo_desc : #抖音上瘾# @na na 发了一个抖音短视频，你尽管点开，不好看算我输！戳这里>>
share_title : @na na 发了一个抖音短视频，你尽管点开，不好看算我输！
share_url : https://www.douyin.com/share/video/6484033166217579789/?region=CN&mid=6484033415598639886
share_desc : 别人那里发我的视频都是四十多万赞 自己亲自发抖音君敢不敢让我上次精选口@抖音小助手
```

之所以要解析，也是为了后面的项目准备的，到时候我会公开项目的开发进程。不管怎么样，到这里我们就成功的获取抖音的加密信息了。主要通过动态调试JNI_OnLoad函数来解决so调用闪退问题，在文章开头的时候说到了，其实本文可以直接用简单粗暴的方式解决，就是万能大法：全局搜索字符串信息，这里包括Jadx中查找和IDA中查看，因为字符串信息能给我

们带来的信息非常多，有时候靠猜一下就可以定位到破解口了，比如这里，我们在IDA中使用快捷键Shift+F12打开字符串窗口：

Function name	Address	Length	Type	String
JNINEnv::FindClass(char const*)	.text:00050A18	00000031	C	AES for ARMv4, CRYPTOGAMS by <appro@openssl.org>
JNINEnv::ExceptionClear(void)	.ARM.exidx:00...	00000005	C	\bSDn
JNINEnv::DeleteLocalRef(_jobject *)	.rodata:000596...	00000011	C	its huoshanzhibo
JNINEnv::GetObjectClass(_jobject *)	.rodata:000596F0	00000011	C	its awemedouyin1
JNINEnv::GetMethodID(jclass *,char const*,char const*)	.rodata:000597...	00000011	C	its jokeessay123
JNINEnv::CallObjectMethod(_jobject *,_jmethodID)	.rodata:000597...	00000005	C	%08x
JNINEnv::GetFieldID(jclass *,char const*,_jfieldID)	.rodata:000597...	00000005	C	rstr
JNINEnv::GetStaticMethodID(jclass *,char const*,_jmethodID)	.rodata:000597...	00000010	C	/proc/%d/status
JNINEnv::CallStaticMethod(jclass *,_jmethodID)	.rodata:000597...	0000000A	C	TracerPid
JNINEnv::NewStringUTF(char const*)	.rodata:000597...	0000001C	C	vector::M_emplace_back_aux
JNINEnv::GetStringUTFChars(jstring *,uchar *)	.rodata:000597...	00000009	C	getBytes
JNINEnv::ReleaseStringUTFChars(jstring *,char *)	.rodata:000597...	00000017	C	(Ljava/lang/String;)B
JNINEnv::ReleaseStringUTFChars(jstring *,char *)	.rodata:000597...	00000006	C	UTF-8
JNINEnv::GetArrayLength(_jarray *)	.rodata:000597...	00000005	C	rstr
JNINEnv::GetObjectArrayElement(_jobjectArray *)	.rodata:000597...	00000013	C	csGib, nica13c4d98555d694efe10410465
JNINEnv::GetByteElements(_jbyteArray *,uc *)	.rodata:000598...	00000012	C	getPackageManager;
JNINEnv::ReleaseByteElements(_jbyteArray *,uc *)	.rodata:000598...	00000026	C	()Landroid/content/pm/PackageManager;
JNINEnv::ExceptionCheck(void)	.rodata:000598...	0000000F	C	getPackageName
getUTF8(JNINEnv *,_jclass *,_jstring *)	.rodata:000598...	00000015	C	()Ljava/lang/String;
JNIN_OnLoad	.rodata:000598...	0000000F	C	getPackageInfo
check_package_installed(JNINEnv *,std::vector<...>)	.rodata:000598...	00000036	C	(Ljava/lang/String;)Landroid/content/pm/PackageInfo;
	.rodata:000598...	0000000B	C	signatures
	.rodata:000598...	00000020	C	()Landroid/content/pm/Signature;
	.rodata:000598...	0000000E	C	toCharsString
	.rodata:000598...	0000002B	C	com/ss/android/common/applog/GlobalContext
	.rodata:000598...	0000000B	C	getContext
	.rodata:000598...	0000001C	C	()Landroid/content/Context;
	.rodata:000598...	00000021	C	0aa08095564a155b455ede748688ee5c
	.rodata:000598...	00000012	C	/proc/tty/drivers

凭着这些关键字字符串信息就能断定so中做了哪些操作。记住这些敏感的字符串信息，对日后的逆向非常关键。

五、技术总结

上面就解决了抖音的请求数据加密信息问题了，下面来总结一下本次逆向学习到的技术：

- 第一、看到在native层用反射去调用Java层的方法获取信息也是一种防护so被恶意调用的方式。比如本文的context变量获取。
- 第二、签名校验永远都不过时，其实本文当时没想到他有签名校验，因为看到native函数中都没有传递context变量，谁知道他是用反射调用Java层方法获取的，长知识和经验了。
- 第三、反调试也是永远不过时的，不过他这里的反调试检测有点简单了，就一处而且就一个进程，如果高级点应该启多个进程，循环检查tracepid值进行校验，会增大难度。
- 第四、在逆向中有时候在Java层没必要去花时间分析一个方法的参数和返回值构造情况，直接利用Xposed进行hook大法打印方法的参数信息靠猜也就出来了。
- 第五、静态方式分析永远都不会过时，全局搜索字符串也是最基本法则，靠猜就可以快速获取结果。

六、火山小视频加密分析

上面解决了抖音的加密问题，下面再来看一下火山小视频的加密信息，突破口依然是使用Fiddler进行抓包查看数据：

依然是https的
安装证书即可

和抖音异曲同工，利用时间戳和count字段来进行分页请求，而会很神奇的发现，公共字段都和抖音一样，真是自家兄弟

```

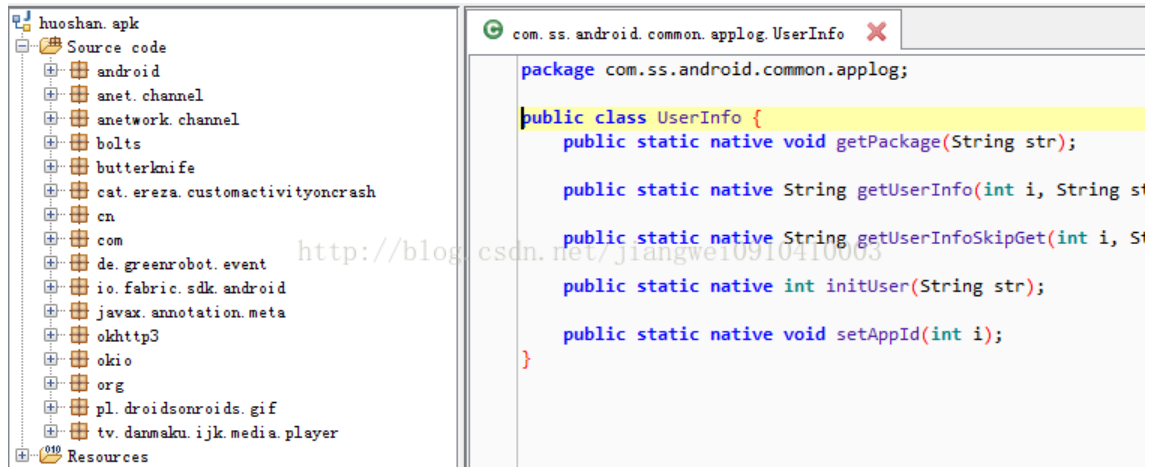
{
  "JSON": {
    "data": {
      "data": {
        "allow_comment=True",
        "allow_dislike=True",
        "allow_share=True",
        "at_users": {
          "author": {
            "allow_be_located=True",
            "allow_find_by_contacts=True",
            "allow_others_download_video=True",
            "allow_show_in_gossip=True",
            "allow_strange_comment=True",
            "allow_sync_to_other_platform=True",
            "avatar_jpg": {
              "uri=720x720/3ffc0004c0ccdbb76c3e",
              "url_list": [
                "http://p1.pstatp.com/live/720x720/3ffc0004c0ccdbb76c3e.jpg",
                "http://pb3.pstatp.com/live/720x720/3ffc0004c0ccdbb76c3e.jpg",
                "http://pb3.pstatp.com/live/720x720/3ffc0004c0ccdbb76c3e.jpg"
              ]
            },
            "avatar_large": {
              "uri=1080x1080/3ffc0004c0ccdbb76c3e"
            }
          }
        }
      }
    }
  }
}

```

通过抓包会很神奇的发现，和抖音的数据结构字段几乎异曲同工，果然是自家兄弟，继续查看他的加密字段：

Name	Value
version_code	272
version_name	2.7.2
device_platform	android
ssmix	a
device_type	MI 3
device_brand	Xiaomi
os_api	19
os_version	4.4.4
uuid	863970029764198
openudid	b39d9675ee6af5b2
manifest_version_code	272
resolution	1080*1920
dpi	480
update_version_code	2722
ts	1509851148
as	a2f5983fdc9029008e
cp	89059b5dc9e0f207e2

不想多说了，既然加密的字段都是一样的。那么我们直接不多说用Jadx打开火山小视频，看看他有没有那个native类UserInfo信息：



哈哈，看来不用多一次分析了，完全一样，那么直接用同一个so，同一个加密即可，在demo工程中运行查看日志：

```
try{
    System.loadLibrary("userinfo");
}catch(Exception e){
    Logger.log("load so err:"+Log.getStackTraceString(e));
}

UserInfo.setAppId(2);

int result = UserInfo.initUser("a3668f0afac72ca3f6c1697d29e0e1bb1fef4ab0285319b95ac39fa42c38d05f");
Logger.log("init user result:"+result);

//getDouyinVideoData();
getHotsoonVideoData();
```

初始化都是一样的，直接运行：

```
levideo    get as_cp:a115b81f499d8951be8dd29a559be7f91ee1
levideo    get data url:https://hotsoon.snssdk.com/hotsoon/feed/?version_name=1.5.9&ts=1 0
509851609&count=10&device_type=MI+3&iid=16715863991&app_type=normal&resolutio 0
n=1080*1920&aid=1128&type=video&app_name=aweme&device_platform=android&versio 0
n_code=159&dpi=480&openudid=b39d9675ee6af5b2&live_sdk_version=272&ssmix=a&os_ 0
api=19&device_id=40545321430&device_brand=Xiaomi&min_time=0&manifest_version_ 0
code=159&os_version=4.4.4&req_from=enter_auto&update_version_code=1592&max_ti 0
me=0&ac=wifi&uuid=863970029764198&channel=360&as=a115b81f499d8951be&cp=8dd29a 0
559be7f91ee1
levideo    list video data:maxtime=-1976878295,minitime=-1976878286,videolist:[videotitle 0
=贼贼贼，让贼偷走啦@三观正的乔小乔,videoplayurl=https://api.huoshan.com/hotsoon/it...
layback/?video_id=e5dd5146689245c888c357afdfd12efe&line=0&app_id=1128&vqualit 0
y=normal,videodownloadurl=https://api.huoshan.com/hotsoon/item/video/_playbac 0
k/?video_id=e5dd5146689245c888c357afdfd12efe&line=0&app_id=1128&vquality=norm 0
al&watermark=1,width=540,height=960,coverimgurl=http://p1.pstatp.com/large/42 0
830013393a444bc51b.webp,authorname=,authorimgurl=http://p3.pstatp.com/live/72 0
0x720/42a90001b0b73d6480dd.jpg,authorcity=辽源,authorage=90后,videoplaycount=189 0
9767,videoduration=15.015, videotitle=贼贼贼，让贼偷走啦@三观正的乔小乔,videoplayu...
```

看到了，数据也回来了，看看他的原始json数据：


```
data
{
  display_style : 3
  weibo_share_title : #玩视频上火山#贝一亮在火山上了分享了视频，快来围观！传送门戳我>>https://www.huoshan.com/share/hsvi
  create_time : 1509625991
  video
  {
    video_id : 066a491d58f745a69ffe1f0fd4414784
    url_list :
    {
      https://api.huoshan.com/hotsoon/item/video/_playback?video_id=066a491d58f745a69ffe1f0fd4414784&line=0&app_id=112
      https://api.huoshan.com/hotsoon/item/video/_playback?video_id=066a491d58f745a69ffe1f0fd4414784&line=1&app_id=112
    }
    allow_cache : true
    cover
    {
      uri : 066a491d58f745a69ffe1f0fd4414784
      download_url :
      {
        https://api.huoshan.com/hotsoon/item/video/_playback?video_id=066a491d58f745a69ffe1f0fd4414784&line=0&app_id=112
        https://api.huoshan.com/hotsoon/item/video/_playback?video_id=066a491d58f745a69ffe1f0fd4414784&line=1&app_id=112
      }
      height : 960
      width : 540
      duration : 9.734
    }
    cover_medium
    {
      url_list :
      {
        http://p3.pstatp.com/medium/427e00096f46848cf2e4.webp
        http://pb9.pstatp.com/medium/427e00096f46848cf2e4.webp
        http://pb3.pstatp.com/medium/427e00096f46848cf2e4.webp
      }
      uri : medium/427e00096f46848cf2e4
    }
    cover_thumb
    {
      url_list :
      {
        http://p3.pstatp.com/live/100x100/427e00096f46848cf2e4.webp
        http://pb9.pstatp.com/live/100x100/427e00096f46848cf2e4.webp
        http://pb3.pstatp.com/live/100x100/427e00096f46848cf2e4.webp
      }
      uri : live/100x100/427e00096f46848cf2e4
    }
    id : 6483794239581850894
    user_bury : 0
  }
}
```

到这里，我们就成功的破解了抖音和火山小视频的数据请求协议了，有了这两个短视频数据，后面我们开发app就简单了，当然在文章开始的时候也说了，现阶段短视频四小龙：抖音，火山，秒拍，快手，那么已经干掉了前面两个，下一个是谁呢？猜对有奖。争取在年底把这四个app全部爆破成功，能够请求到他的数据。为我们明年的app作为基础

严重声明

本文的意图只有一个就是通过分析app学习更多的逆向技术，如果有人利用本文知识和技术进行非法操作进行牟利，带来的任何法律责任都将由操作者本人承担，和本文作者无任何关系，最终还是希望大家能够秉着学习的心态阅读此文。鉴于安全问题，样本和源码都去编码美丽小密圈自取！

七、总结

通过本文可以发现，我们其实没有真正意义上的破解它的算法，但是结果却是我们想要的，这就够了。而对于现在很多app把加密算法放到so中，在对so做一些防护，这样就很难利用本文的技术去调用app的so了。不过so再怎么防护就是那么几种方法，我们依然可以用动态调试来解决。有人在文中很好奇，那些判断校验不能直接修改so指令做到吗？比如签名校验，没必要在Java层进hook呀，直接修改CMP和BEQ指令呗？的确可以这么做，但是这是下一篇介绍的内容，因为我们下一篇要破解下一个短视频app，就用到这种方式过校验。敬请期待！

点评

 hupengpeng 四哥老牛逼了！！！！点赞！ 发表于 2017-12-11 10:07

免费评分

参与人数	15	威望	+2	吾爱币	+28	热心值	+15	理由	收藏
<hr/>									
h080294				+ 1		+ 1		我很赞同！	
<hr/>									
 陈小胖				+ 1		+ 1		我很赞同！	
<hr/>									
 金神主				+ 1		+ 1		谢谢@Thanks！	
<hr/>									

	maya369	+ 1	+ 1	大神，天书，打基础！
	我叫初恋	+ 1	+ 1	我很赞同！
	赛浩	+ 2	+ 1	膜拜大神，不知道大神有没有0基础视频，您这个图文我看不懂，hook怎
	sec0ces	+ 1	+ 1	大神
	WYWZ	+ 1	+ 1	用心讨论，共获提升！
	xkwdm	+ 1	+ 1	感谢发布原创作品，吾爱破解论坛因你更精彩！
	Zhenwu1080	+ 1	+ 1	感谢发布原创作品，吾爱破解论坛因你更精彩！
	愚无尽	+ 1	+ 1	膜拜
	qtreet00	+ 2	+ 12	+ 1 感谢发布原创作品，吾爱破解论坛因你更精彩！
	跳跃的灵魂	+ 2	+ 1	用心讨论，共获提升！
	sunnylds7	+ 1	+ 1	热心回复！
	supperlitt	+ 1	+ 1	niu...BBBBBBBBBBBBBBBBBBB

查看全部评分

本帖被以下淘专辑推荐：

· 学习及教程 | 主题: 845, 订阅: 411

发帖求助前要善用【论坛搜索】功能，那里可能会有你要找的答案；
如果你在论坛求助问题，并且已经从坛友或者管理的回复中解决了问题，请把帖子分类或者标题加上【已解决】；
如何回报帮助你解决问题的坛友，一个好办法就是给对方加【热心】和【CB】，加分不会扣除自己的积分，做一个热心并受欢迎的人！
论坛附件只能通过单线程下载，迅雷等多线程工具不能正常下载！

回复 使用道具 举报

jiangwei212



楼主 | 发表于 2017-12-11 08:41 | 显示全部楼层

吾爱这边不给弄二维码，这边的编辑格式也不太方便查看，感兴趣的可以去我的website: <http://www.wjdiankong>

免费评分




参与人数 1 吾爱币 +2 热心值 +1 理由 收录

 鱼儿飞 + 2 + 1 你的网址输入错了

查看全部评分

发帖求助前要善用【论坛搜索】功能，那里可能会有你要找的答案；
如果你在论坛求助问题，并且已经从坛友或者管理的回复中解决了问题，请把帖子分类或者标题加上【已解决】；
如何回报帮助你解决问题的坛友，一个好办法就是给对方加【热心】和【CB】，加分不会扣除自己的积分，做一个热心并受欢迎的人！
【吾爱破解论坛总版规】 - [让你充分了解吾爱破解论坛行为规则]

<div data-bbox="97 152 221 174" data-label="Text"><p>jiangwei212</p></div> <div data-bbox="105 210 288 392" data-label="Image"></div>	<div data-bbox="386 89 1484 112" data-label="Text"><p>回复 支持 反对 使用道具 举报</p></div> <div data-bbox="347 150 826 174" data-label="Text"><p> 楼主 发表于 2017-12-11 09:21 显示全部楼层</p></div> <div data-bbox="347 208 1238 271" data-label="Text"><p>吾爱破解论坛没有任何官方QQ群，禁止留联系方式，禁止任何商业交易。 吾爱这边不能贴二维码，文章排版有点乱，可以去我的website查看：http://www.wjdiankong.cn</p></div> <div data-bbox="347 443 1474 665" data-label="Text"><p>发帖求助前要善用【论坛搜索】功能，那里可能会有你要找的答案； 如果你在论坛求助问题，并且已经从坛友或者管理的回复中解决了问题，请把帖子分类或者标题加上【已解决】； 如何回报帮助你解决问题的坛友，一个好办法就是给对方加【热心】和【CB】，加分不会扣除自己的积分，做一个热心并受欢迎的人！ 论坛账号被盗严重，教你如何保障社区帐号的安全！</p></div> <div data-bbox="386 692 1484 714" data-label="Text"><p>回复 支持 反对 使用道具 举报</p></div>
<div data-bbox="97 757 194 779" data-label="Text"><p>supperlitt</p></div> <div data-bbox="105 815 288 996" data-label="Image"></div>	<div data-bbox="347 757 753 779" data-label="Text"><p> 发表于 2017-12-11 09:11 显示全部楼层</p></div> <div data-bbox="347 813 948 875" data-label="Text"><p>《站点帮助文档》有什么问题来这里看看吧，这里有你想知道的内容！ 厉害了，我的哥。</p></div> <div data-bbox="347 1048 1474 1270" data-label="Text"><p>发帖求助前要善用【论坛搜索】功能，那里可能会有你要找的答案； 如果你在论坛求助问题，并且已经从坛友或者管理的回复中解决了问题，请把帖子分类或者标题加上【已解决】； 如何回报帮助你解决问题的坛友，一个好办法就是给对方加【热心】和【CB】，加分不会扣除自己的积分，做一个热心并受欢迎的人！ 如何升级？如何获得积分？积分对应解释说明！</p></div> <div data-bbox="386 1296 1484 1319" data-label="Text"><p>回复 支持 反对 使用道具 举报</p></div>
<div data-bbox="97 1361 194 1384" data-label="Text"><p>supperlitt</p></div> <div data-bbox="105 1420 288 1601" data-label="Image"></div>	<div data-bbox="347 1361 753 1384" data-label="Text"><p> 发表于 2017-12-11 09:11 显示全部楼层</p></div> <div data-bbox="347 1417 596 1440" data-label="Text"><p>你太，厉害了，老哥。。。</p></div> <div data-bbox="347 1612 1455 1834" data-label="Text"><p>发帖求助前要善用【论坛搜索】功能，那里可能会有你要找的答案； 如果你在论坛求助问题，并且已经从坛友或者管理的回复中解决了问题，请把帖子分类或者标题加上【已解决】； 如何回报帮助你解决问题的坛友，一个好办法就是给对方加【热心】和【CB】，加分不会扣除自己的积分，做一个热心并受欢迎的人！ 如何快速判断一个文件是否为病毒！</p></div> <div data-bbox="386 1861 1458 1883" data-label="Text"><p>回复 支持 反对 使用道具 当</p></div>
<div data-bbox="97 1926 193 1948" data-label="Text"><p>mcloveok</p></div> <div data-bbox="118 1984 288 2132" data-label="Image"></div>	<div data-bbox="347 1926 753 1948" data-label="Text"><p> 发表于 2017-12-11 10:00 显示全部楼层</p></div> <div data-bbox="347 1982 434 2004" data-label="Text"><p>前排沙发</p></div>

	<div>发帖求助前要善用【论坛搜索】功能，那里可能会有你要找的答案；😄</div> <div>如果你在论坛求助问题，并且已经从坛友或者管理的回复中解决了问题，请把帖子分类或者标题加上【已解决】；👉</div> <div>如何回报帮助你解决问题的坛友，一个好办法就是给对方加【热心】和【CB】，加分不会扣除自己的积分，做一个热心并受欢迎的人！😊</div> <div>呼吁大家发布原创作品添加吾爱破解论坛标示！</div> <div><div>回复支持反对</div><div>使用道具举报</div></div>
xkwdm	<div>👤 发表于 2017-12-11 10:04 显示全部楼层</div> <div></div> <div>膜拜大神👍</div> <div>发帖求助前要善用【论坛搜索】功能，那里可能会有你要找的答案；😄</div> <div>如果你在论坛求助问题，并且已经从坛友或者管理的回复中解决了问题，请把帖子分类或者标题加上【已解决】；👉</div> <div>如何回报帮助你解决问题的坛友，一个好办法就是给对方加【热心】和【CB】，加分不会扣除自己的积分，做一个热心并受欢迎的人！😊</div> <div>如何快速赚到 CB，而且不会被关进小黑屋！</div> <div><div>回复支持反对</div><div>使用道具举报</div></div>
hupengpeng	<div>👤 发表于 2017-12-11 10:06 显示全部楼层</div> <div></div> <div>给四哥点赞！！！！</div> <div>发帖求助前要善用【论坛搜索】功能，那里可能会有你要找的答案；😄</div> <div>如果你在论坛求助问题，并且已经从坛友或者管理的回复中解决了问题，请把帖子分类或者标题加上【已解决】；👉</div> <div>如何回报帮助你解决问题的坛友，一个好办法就是给对方加【热心】和【CB】，加分不会扣除自己的积分，做一个热心并受欢迎的人！😊</div> <div><div>回复支持反对</div><div>使用道具举报</div></div>
yuan71058	<div>👤 发表于 2017-12-11 10:21 显示全部楼层</div> <div></div> <div>感觉你的项目就是获取他们几个视频的真实地址，然后加以利用</div> <div>发帖求助前要善用【论坛搜索】功能，那里可能会有你要找的答案；😄</div> <div>如果你在论坛求助问题，并且已经从坛友或者管理的回复中解决了问题，请把帖子分类或者标题加上【已解决】；👉</div> <div>如何回报帮助你解决问题的坛友，一个好办法就是给对方加【热心】和【CB】，加分不会扣除自己的积分，做一个热心并受欢迎的人！😊</div> <div><div>回复支持反对</div><div>使用道具举报</div></div>
ly847846556	<div>👤 发表于 2017-12-11 10:28 显示全部楼层</div> <div></div> <div>面对大神 我们都是仰望着.....</div>



发帖求助前要善用【论坛搜索】功能，那里可能会有你要找的答案； 😊

如果你在论坛求助问题，并且已经从坛友或者管理的回复中解决了问题，请把帖子分类或者标题加上【已解决】； 🙌

如何回报帮助你解决问题的坛友，一个好办法就是给对方加【热心】和【CB】，加分不会扣除自己的积分，做一个热心并受欢迎的人！ 😊

回复

支持

反对

使用道具

举报

下 一 页 »

发帖 ▾

返回列表

1

2

1 / 2 页

下一页

高级模式

您需要登录后才可以回帖 登录 | 注册[Register]  用QQ帐号登录

发表回复

☐ 回帖并转播

☐ 回帖后跳转到最后一页

本版积分规则

免责声明：
吾爱破解所发布的一切破解补丁、注册机和注册信息及软件的解密分析文章仅限于学习和研究目的；不得将上述内容用于商业或者非法用途，否则，一切后果请用户自负。本站信息来自网络，版权争议与本站无关。您必须在下载后的24个小时之内，从您的电脑中彻底删除上述内容。如果您喜欢该程序，请支持正版软件，购买注册，得到更好的正版服务。如有侵权请邮件与我们联系处理。

Powered by **Discuz!**

RSS订阅 | 手机版 | 小黑屋 | 联系我们 | 吾爱破解 - LCG - LSG (京ICP备16042023号 | 京公网安备 11010502030087号)

© 2001-2017 Comsenz Inc.

GMT+8, 2017-12-11 11:48

Mail To:Service@52PoJie.Cn

http://cache.baiducontent.com/c?m=9f65cb4a8c8507ed19fa950d100b92235c4380143bd79248278fc45f93130a1d5a20b9fb70714613d3b3786207ad4c5afdf040... 28/28