包名：com.achievo.vipshop

搜索接口

抓包分析

直接在app中搜索商品，然后在charles中搜索app中搜到的商品关键字，进而定位是哪个接口

请求网址：https://mapi.appvipshop.com/vips-mobile/rest/shopping/search/product/list/v1
请求方式：post
请求头：

```
    authorization   OAuth api_sign=e5099c585c02e81d4ce4ff8ca2b862067e8e7eda
    x-vip-host  mapi.appvipshop.com
    content-type    application/x-www-form-urlencoded
    content-length  2487
    accept-encoding gzip
    user-agent  okhttp/4.9.1
```

请求体：

```
    api_key 23e7f28019e8407b98b84cd05b5aef2c
    app_name    shop_android
    app_version 7.83.3
    bigSaleTagIds
    brandIds
    brandStoreSns
    categoryId
    channelId   1
    channel_flag    0_1
    client  android
    client_type android
    darkmode    0
    deeplink_cps
    device_model    Google Pixel
    did 0.0.d2e4dbe854df78dc59011b56282bdb21.f89f14
    elder   0
    extParams
{"priceVer":"2","mclabel":"1","cmpStyle":"1","statusVer":"2","ic2label":"1","vid
eo":"2","uiVer":"2","preheatTipsVer":"4","floatwin":"1","superHot":"1","exclusiv
ePrice":"1","router":"1","coupons":"1","needVideoExplain":"1","rank":"2","needVi
deoGive":"1","bigBrand":"1","couponVer":"v2","videoExplainUrl":"1","live":"1","s
ellpoint":"1","reco":"1","vreimg":"1","search_tag":"2","tpl":"1","stdSizeVids":"
","labelVer":"2"}
    fdc_area_id 101102101
    functions
RTRecomm,flagshipInfo,feedback,otdAds,zoneCode,slotOp,survey,hasTabs,floaterPara
ms
    harmony_app 0
    harmony_os  0
    headTabType all
    height  1794
    isMultiTab  0
```

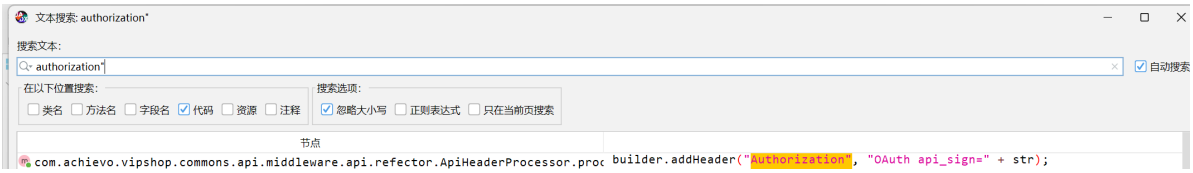
```
    keyword 香水
    lastPageProperty    {"isBgToFront":"0","suggest_text":"香
水","scene_entry_id":"-99","refer_page_id":"page_te_globle_classify_search_174804
9002631","text":"香
水","tag":"1","module_name":"com.achievo.vipshop.search","type":"all","typename":
"全部","is_back_page":"0"}
    maker   GOOGLE
    mars_cid    c4acd008-4f30-3599-b32c-7f7dbc2135e9
    mobile_channel  oziq7dxw:::
    mobile_platform 3
    net WIFI
    operator
    os  Android
    osv 10
    otddid
    other_cps
    page_id page_te_commodity_search_1748049007784
    phone_model pixel
    priceMax
    priceMin
    props
    province_id 101102
    referer com.achievo.vipshop.search.activity.TabSearchProductListActivity
    rom Dalvik/2.1.0 (Linux; U; Android 10; Pixel Build/QP1A.191005.007.A3)
    sd_tuijian  0
    service_provider
    session_id  _shop_android_1748048965624
    skey    6692c461c3810ab150c9a980d0c275ec
    sort    0
    source  app
    source_app  android
    standby_id  oziq7dxw:::
    sys_version 29
    timestamp   1748049007
    union_mark  blank&_&blank&_&oziq7dxw:::&_&blank&_&blank
    vipService
    warehouse   VIP_BJ
    width   1080
```

请求头

```
    authorization       OAuth api_sign=e5099c585c02e81d4ce4ff8ca2b862067e8e7eda
    x-vip-host          mapi.appvipshop.com
    content-type        application/x-www-form-urlencoded
    content-length      2487
    accept-encoding     gzip
    user-agent          okhttp/4.9.1
```

authorization

```
//请求头中两个参数需要去分析：authorization、x-vip-host
//x-vip-host是明文 应该是主机名
//authorization是搞出来一个sign然后拼接的


//我们直接去搜索关键字：authorization或者OAuth api_sign试试
```




```
//这就很像了，首先是在添加请求头，其次key和value都是我们想看到的==>进去看看
public boolean process() {
    ......
if (str != null) {
    Request.Builder builder = apiProccessModel5.request;
    builder.addHeader("Authorization", "OAuth api_sign=" + str);
}
    ......
}

//str就是OAuth api_sign=对应的值；authorization的OAuth api_sign=是固定的
//str在process函数中有三种赋值方式：
String str = "";
str = b.b(context, treeMap2, apiProccessModel2.tokenSecret,
apiProccessModel2.url);
str = b.b(context2, e10, apiProccessModel4.tokenSecret, apiProccessModel4.url);

//str肯定不是空，所以就是通过b.b得到的,我们去hook一下b.b
```

```
//在注入frida脚本的时候发现闪退了，应该是有frida检测，我们去hook一下dlopen
[dlopen_ext:] /data/app/com.achievo.vipshop-
3kpyBAS3nQgtTRZD8KzORw==/lib/arm/libmsaoaidsec.so
Process terminated
//我们试着去直接跳过，不加载这个so看看能行不==>跳过libmsaoaidsec.so不影响app的运行，所以我们直接去把它删掉，免得每次spawn都得跳过
```

```
//继续分析，我们去hook一下b.b看看入参和返回值，这样就能知道OAuth api_sign=是怎么来的了

b.b is called:
//参数:
context=com.achievo.vipshop.common.VipApplicationLike@cdcaa7b,

treeMap={api_key=23e7f28019e8407b98b84cd05b5aef2c, app_name=shop_androi
d, app_version=7.83.3, bigSaleTagIds=, brandIds=, brandStoreSns=, categoryId=,
channelId=1, channel_flag=0_1, client=android, client_type=android, da
rkmode=0, deeplink_cps=, device_model=Google Pixel,
did=0.0.d2e4dbe854df78dc59011b56282bdb21.f89f14, elder=0, extParams=
{"priceVer":"2","mclabel":"1"
```

,"cmpStyle":"1","statusVer":"2","ic2label":"1","video":"2","uiVer":"2","preheatTipsVer":"4","floatwin":"1","superHot":"1","exclusivePrice":"1","router":"1","coupons":"1","needVideoExplain":"1","rank":"2","needVideoGive":"1","bigBrand":"1","couponVer":"v2","videoExplainUrl":"1","live":"1","sellpoint":"1","reco":"1","vreimg":"1","search_tag":"2","tpl":"1","stdSizeVids":"","labelVer":"2"}, fdc_area_id=101102101, functions=RTRecomm,flagshipInfo,feedback,otdAds,zoneCode,slotOp,survey,hasTabs,floaterParams, harmony_app=0, harmony_os=0, headTabType=all, height=1794, isMultiTab=0, keyword=口红, lastPageProperty={"isBgToFront":"0","suggest_text":"口红","scene_entry_id":"-99","refer_page_id":"page_te_globle_classify_search_1748052436903","text":"口红","tag":"1","module_name":"com.achievo.vipshop.search","type":"all","typename":"全部","is_back_page":"0"}, maker=GOOGLE, mars_cid=c4acd008-4f30-3599-b32c-7f7dbc2135e9, mobile_channel=oziq7dxw:::, mobile_platform=3, net=WIFI, operator=, os=Android, osv=10, otddid=, other_cps=, page_id=page_te_commodity_search_1748052449703, phone_model=pixel, priceMax=, priceMin=, props=, province_id=101102, referer=com.achievo.vipshop.search.activity.TabSearchProductListActivity, rom=Dalvik/2.1.0 (Linux; U; Android 10; Pixel Build/QP1A.191005.007.A3), sd_tuijian=0, service_provider=, session_id=c4acd008-4f30-3599-b32c-7f7dbc2135e9_shop_android_1748052403503, skey=6692c461c3810ab150c9a980d0c275ec, sort=0, source=app, source_app=android, standby_id=oziq7dxw:::, sys_version=29, timestamp=1748052449, union_mark=blank&_&blank&_&oziq7dxw:::&_&blank&_&blank, vipService=, warehouse=VIP_BJ, width=1080},

str=null,

str2=https://mapi.appvipshop.com/vips-mobile/rest/shopping/search/product/list/v1

返回值：

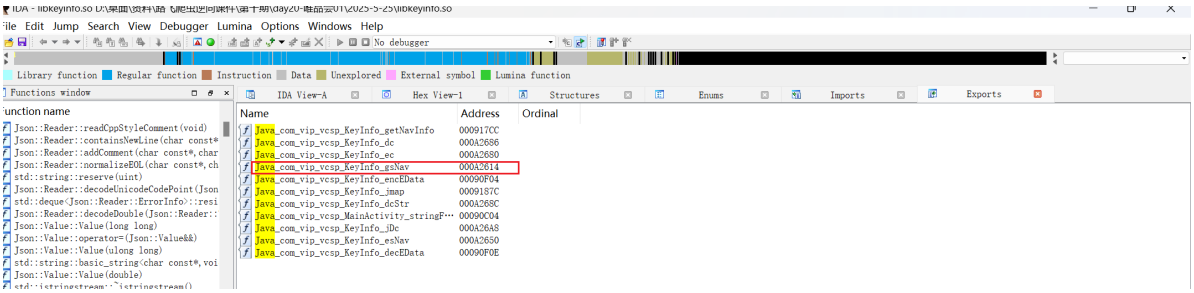
b.b result=24357f9b8530d4b41648afa25e2d486347edb5fc

```
//参数1是context、参数2是一个treeMap、参数3是空、参数4是我们的搜索的url
//再仔细对比，我们发现参数2的treeMap就是我们的请求体
//现在我们知道参数是什么了，我们去看b.b的逻辑，看看是如何得到OAuth api_sign=的
b.b:
public static String b(Context context, TreeMap<String, String> treeMap, String str, String str2) {
    if (treeMap != null && TextUtils.isEmpty(treeMap.get("skey"))) {
        treeMap.put("skey", f(context, new String[0]));
    }
    return a(context, treeMap, str);
}
//进来后先判断treeMap是否为空，进而判断treeMap中得到"skey"的值是否为空，二者都不为空就调用a(context, treeMap, str)，若"skey"的值的为空，就先put进去一个。在对b.b进行hook的时候，我们可以看到在treeMap中"skey"的值是不为空的，所以我们去看a方法

a(context, treeMap, str):
private static String a(Context context, TreeMap<String, String> treeMap, String str) {
    try {
```

```
        if (VCSPCommonsConfig.getContext() == null) {
            VCSPCommonsConfig.setContext(context);
        }
        String apiSign = VCSPSecurityBasicService.apiSign(context, treeMap,
str);
        if (TextUtils.isEmpty(apiSign)) {
            String a10 = com.achievo.vipshop.commons.c.a();
            return "p: " + a10 + ", vcsp return empty sign :" + apiSign;
        }
        return apiSign;
    } catch (Exception e10) {
        e10.printStackTrace();
        String a11 = com.achievo.vipshop.commons.c.a();
        return "p: " + a11 + ", Exception:" + e10.getMessage();
    } catch (Throwable th2) {
        th2.printStackTrace();
        String a12 = com.achievo.vipshop.commons.c.a();
        return "p: " + a12 + ", Throwable:" + th2.getMessage();
    }
}
//这里传过来的str是调用b.b的时候的第三个参数，传进去的是null
//在a方法中，主要的逻辑就是try里边的，拿到一个apiSign，然后返回，如果apiSign为空就返回"p: "
+ a10 + ", vcsp return empty sign :" + apiSign; 显然这里的apiSign不为空，所以我们去看
apiSign的生成逻辑:
String apiSign = VCSPSecurityBasicService.apiSign(context, treeMap, str);
//进而调用了VCSPSecurityBasicService.apiSign(context, treeMap, str)得到的apiSign
//所以我们去看VCSPSecurityBasicService.apiSign(context, treeMap, str)
public static String apiSign(Context context, TreeMap<String, String> treeMap,
String str) throws Exception {
    if (context == null) {
        context = VCSPCommonsConfig.getContext();
    }
    return VCSPSecurityConfig.getMapParamsSign(context, treeMap, str, false);
}
//里边又调用了VCSPSecurityConfig.getMapParamsSign(context, treeMap, str, false)
public static String getMapParamsSign(Context context, TreeMap<String, String>
treeMap, String str, boolean z10)
{
    String str2 = null;
    if (treeMap != null) {
        boolean z11 = false;
        Set<Map.Entry<String, String>> entrySet = treeMap.entrySet();
        if (entrySet != null) {
            Iterator<Map.Entry<String, String>> it = entrySet.iterator();
            while (true) {
                if (it == null || !it.hasNext()) {
                    break;
                }
                Map.Entry<String, String> next = it.next();
                if (next != null && next.getKey() != null &&
ApiConfig.USER_TOKEN.equals(next.getKey()) &&
!TextUtils.isEmpty(next.getValue())) {
                    z11 = true;
                    break;
                }
```

```
            }
        }
        if (z11) {
            if (TextUtils.isEmpty(str)) {
                str = VCSPCommonsConfig.getTokenSecret();
            }
            str2 = str;
        }
        return getSignHash(context, treeMap, str2, z10);
    }
    return null;
    }
//这个方法最终调用的是getSignHash(context, treeMap, str2, z10);
//在这个方法中，主要是对str2进行赋值，进来后str2是null，然后对treeMap里边的值进行判断，看是否存在一个key为"user_token"的值，若存在，会调用VCSPCommonsConfig.getTokenSecret()生成一个字符串，然后赋值给str2，在调用getSignHash的时候传进去
//通过对我们抓到的信息进行分析是没有key为"user_token"的值的，因此这里str2应该是null。只通过静态分析是不能判断的，我们去hook一下getSignHash(context, treeMap, str2, z10)，看看传过来的str2是什么值
//VCSPSecurityConfig.getSignHash is called:
context=com.achievo.vipshop.common.VipApplicationLike@cdcaa7b, map=[object Object], str=null, z10=false
//经过hook验证，str确实为null，和我们之前分析的一致

 //接着去看getSignHash(context, treeMap, str2, z10)的逻辑：
public static String getSignHash(Context context, Map<String, String> map, String str, boolean z10) {
    try {
        return gs(context.getApplicationContext(), map, str, z10);
    } catch (Throwable th2) {
        VCSPMyLog.error(clazz, th2);
        return "error! params invalid";
    }
}
//里边直接调用了gs(context.getApplicationContext(), map, str, z10)
private static String gs(Context context, Map<String, String> map, String str, boolean z10) {
    try {
        if (clazz == null || object == null) {
            synchronized (lock) {
                initInstance();
            }
        }
        if (gsMethod == null) {
            gsMethod = clazz.getMethod("gs", Context.class, Map.class, String.class, Boolean.TYPE);
        }
        return (String) gsMethod.invoke(object, context, map, str, Boolean.valueOf(z10));
    } catch (Exception e10) {
        e10.printStackTrace();
        return "Exception gs: " + e10.getMessage();
    } catch (Throwable th2) {
        th2.printStackTrace();
        return "Throwable gs: " + th2.getMessage();
```

```
    }
}
//这里是进行了反射调用，首先判断类和对象是否为空，为空就去调用initInstance();
private static void initInstance() {
    if (clazz == null || object == null) {
        try {
            int i10 = KeyInfo.f69594a;
            clazz = KeyInfo.class;
            object = KeyInfo.class.newInstance();
        } catch (Exception e10) {
            e10.printStackTrace();
        }
    }
}
//这里主要是给clazz和object赋值了，赋值的是KeyInfo类的类和对象
//再回到gs中，然后就去获取方法id： clazz.getMethod("gs", Context.class, Map.class,
String.class, Boolean.TYPE)
//所以这里拿到的就是KeyInfo类的gs方法的id
//然后去invoke这个KeyInfo类下的gs，传过去的是context, map, str, Boolean.valueOf(z10),
也就是之前传进去的参数
//接下来我们去看个KeyInfo类下的gs：
public static String gs(Context context, Map<String, String> map, String str,
boolean z10) {
    try {
        try {
            return gsNav(context, map, str, z10);
        } catch (Throwable th2) {
            return "KI gs: " + th2.getMessage();
        }
    } catch (Throwable unused) {
        SoLoader.load(context, LibName);
        return gsNav(context, map, str, z10);
    }
}
//进来后又调用了gsNav(context, map, str, z10)
private static native String gsNav(Context context, Map<String, String> map,
String str, boolean z10);

//gsNav是一个native方法，我们去看看在这个类中有没有加载so==>有一个静态代码块
private static final String LibName = "keyinfo";
static {
    try {
        System.loadLibrary(LibName);
    } catch (Throwable th2) {
        th2.printStackTrace();
    }
}
//我们先对这个native方法进行一下hook，拿到参数，方便一会主动调用==>invoke_gsnav()

//我们找keyinfo这个so看看：
```



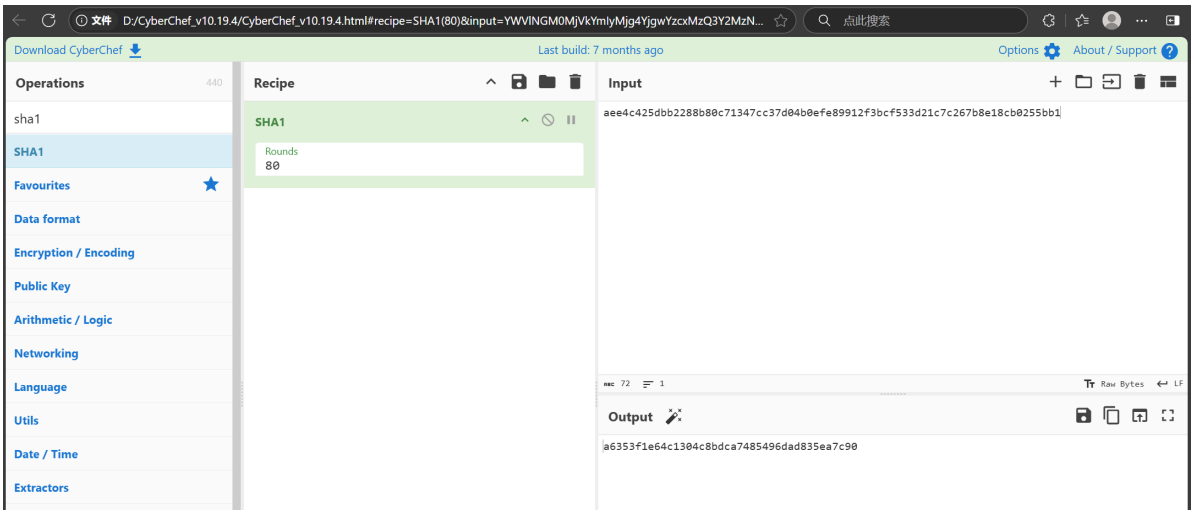

```
//静态注册的==>
int __fastcall Java_com_vip_vcsp_KeyInfo_gsNav(JNIEnv *env, jclass cla, jobject
context, jobject map, jobject str, jobject z10)
{
  int v9; // r5
  if ( j_Utils_ima(env, cla, context) )
    v9 = j_Functions_gs(env, cla, map, str, z10);
  else
    v9 = 0;
  j_Utils_checkJniException(env);
  return v9;
}

//进来后先去判断了j_Utils_ima(env, cla, context)，然后就执行v9 = j_Functions_gs(env,
cla, map, str, z10);
//这里应该是在检测context是否为空啊什么的，不为空就执行v9 = j_Functions_gs(env, cla,
map, str, z10);
//否则就v9 = 0;然后返回。 显然这里是执行的v9 = j_Functions_gs(env, cla, map, str,
z10);

//j_Functions_gs(env, cla, map, str, z10):
int __fastcall j_Functions_gs(int env, int cls, int map, int str, int z10)
{
  return Functions_gs(env, cls, map, str, z10);
}
//又调用了：Functions_gs(env, cls, map, str, z10)
//进来Functions_gs(env, cls, map, str, z10)后我们从后往前看，由果溯因：
//先看return==>整篇代码有很多return 0，是在进行其他操作的时候若不符合预判就return 0；只有一
个return v53是我们要的
return v53;
//而v53是通过newStringUTF得到的：
v53 = (*env)->NewStringUTF(env, v59);
//所以我们去看v59怎么来的
const char *v59;
memset(v80, 0, sizeof(v80));
memset(v79, 0, sizeof(v79));
v55 = j_getByteHash(env, jclass, v30, v16, v80, 256);
if (
    v55 && (
        v56 = (const char *)v55,
        v57 = strcpy(v79, dest),
        strcat(v57, v56),
        memset(v80, 0, sizeof(v80)),
        v58 = strlen(v79),
        (v59 = (const char *)j_getByteHash(env, jclass, v79, v58, v80, 256)) !=
0
            )
```

```
    )
//v59是在if判断表达式中进行赋值的 v55&&(逗号表达式)
//我们可以先不分析这些，可以先去hook j_getByteHash，因为v59是j_getByteHash的返回值，我们先去hook一下，并且我们发现v55也是通过调用j_getByteHash得到的



//在hook j_getByteHash的时候发现被调用了三次，再ida中查看j_getByteHash的交叉引用可以确定j_getByteHash是被调用了三次：
//第一次是在Utils_ima函数中，有获取包名，包信息等等，应该是对包进行了一个计算，看有没有被冲打包
//第二次是拿到v55，
//第三次另一次是拿到v59.v59会返回到java层，我们去看返回的是v59的那次，也就是和我们主动调用java层函数gsnav时结果一致的那一次
```

```
//我们把hook得到的结果拿过来：
gsNav invoke is called!
=============第一次开始=============
args[0] is env!
args[1] is jclass!
[2] *H>> 0_0ηNw0
args[3] is  ==> 611
args[4] is  ==>
args[5] is  ==> 128
retval is  ==> 1ed562e1e90b23ae3f9a40f8b2a65382b95a4752
=============第一次结束=============

=============第二次开始=============
args[0] is env!
args[1] is jclass!
args[2] is ==>
aee4c425dbb2288b80c71347cc37d04bapi_key=23e7f28019e8407b98b84cd05b5aef2c&app_name=shop_android&app_version=7.83.3&bigSaleTagIds=&brandIds=&categoryId=&channelId=1&channel_flag=0_1&client=android&client_type=android&darkmode=0&deeplink_cps=&device_model=Google Pixel&did=0.0.d2e4dbe854df78dc59011b56282bdb21.f89f14&elder=0&extParams=
{"priceVer":"2","mclabel":"1","cmpStyle":"1","statusVer":"2","ic2label":"1","video":"2","uiVer":"2",
"preheatTipsVer":"4","floatwin":"1","superHot":"1","exclusivePrice":"1","router":"1","coupons":"1","needVideoExplain":"1","rank":"2","needVideoGive":"1","bigBrand":"1","couponVer":"v2","videoExplainUrl":"1","live":"1","sellpoint":"1","reco":"1","vreimg":"1","search_tag":"2","tpl":"1","stdSizeVids":"","labelVer":"2"}&fdc_area_id=101102101&functions=RTRecomm,flagshipInfo,feedback,otdAds,zoneCode,slotOp,survey,hasTabs,floaterParams&harmony_app=0&harmony_os=0&headTabType=all&height=1794&isMultiTab=0&keyword=运动裤&lastPageProperty={"isBgToFront":"0","suggest_text":"运动裤","scene_entry_id":"-99","refer_page_id":"page_te_globle_classify_search_1748059729222","text":"运动裤","tag":"1","module_name":"com.achievo.vipshop.search","type":"all","typename":"全部","is_back_page":"0"}&maker=GOOGLE&mars_cid=c4acd008-4f30-3599-b32c-7f7dbc2135e9&mobile_channel=oziq7dxw:::&mobile_platform=3&net=WIFI&os=Android&osv=10&otddid=&other_cps=&page_id=page_te_commodity_search_1748059741674&phone_model=pixel&priceMax=&priceMin=&props=&province_id=101102&referer=com.achievo.vipshop.search.activity.TabSearchProductListActivity&rom=Dalvik/2.1.0 (Linux; U; Android 10; Pixel Build/QP1A.191005.007.A3)&sd_t
```

```
uijian=0&service_provider=&session_id=c4acd008-4f30-3599-b32c-
7f7dbc2135e9_shop_android_1748059681783&skey=6692c461c3810ab150c9a980d0c275ec&so
rt=0&so
urce=app&source_app=android&standby_id=oziq7dxw:::&sys_version=29&timestamp=1748
059741&union_mark=blank&_&blank&_&oziq7dxw:::&_&blank&_&blank&vipService=&wareho
use=VIP_BJ&width=1080
args[3] is  ==> 1965
args[4] is  ==>
args[5] is  ==> 256
retval is  ==> 0efe89912f3bcf533d21c7c267b8e18cb0255bb1
=============第二次结束=============

=============第三次开始=============
args[0] is env!
args[1] is jclass!
args[2] is ==>
aee4c425dbb2288b80c71347cc37d04b0efe89912f3bcf533d21c7c267b8e18cb0255bb1
args[3] is  ==> 72
args[4] is  ==>
args[5] is  ==> 256
retval is  ==> a6353f1e64c1304c8bdca7485496dad835ea7c90
=============第三次结束=============
gsNav invoke result is ==> a6353f1e64c1304c8bdca7485496dad835ea7c90



//可以看出来第三次的结果就是我们的sign，也就是返回到java层的值，那我们去看明文，发现也是加密过
的。
//我们再去看第二次加密的结果，发现是我们第三次明文的后半部分：
//第三次明文:aee4c425dbb2288b80c71347cc37d04b
0efe89912f3bcf533d21c7c267b8e18cb0255bb1
//第二次结果：
0efe89912f3bcf533d21c7c267b8e18cb0255bb1

//我们去再去看第二次的明文，发现再开头的地方也有aee4c425dbb2288b80c71347cc37d04b，那么它很
有可能就是盐，换些参数重新调用，发现还是这个，那它就是盐。再看除了盐的其他部分，就是把我们java层
传过来的treeMap进行了toString，转为了key=vaule&key=value..的形式。然后再开头拼上盐，调用了
j_getByteHash，然后对它的结果再拼上盐又调用了一次j_getByteHash，然后就返回到java层了。我们
去看看j_getByteHash里边的逻辑，看看是什么加密

j_getByteHash：

char *__fastcall getByteHash(JNIEnv *env, jclass cla, int a3, int a4, char *a5)
{
  char *v7; // r10
  int i; // r4
  int v9; // r2
  _QWORD v11[8]; // [sp+0h] [bp-E0h] BYREF
  _DWORD v12[26]; // [sp+44h] [bp-9Ch] BYREF

  if ( !a3 )
    return 0;
  v7 = a5;
  j_SHA1Reset(v12);
  j_SHA1Input(v12, a3, a4);
```

```
  if ( j_SHA1Result(v12) )
  {
    for ( i = 0; i != 5; ++i )
    {
      v9 = v12[i];
      v11[0] = 0LL;
      v11[1] = 0LL;
      v11[6] = 0LL;
      v11[7] = 0LL;
      v11[4] = 0LL;
      v11[5] = 0LL;
      v11[2] = 0LL;
      v11[3] = 0LL;
      sprintf((char *)v11, "%08x", v9);
      strcat(a5, (const char *)v11);
    }
  }
  return v7;
}


//进来后发现有函数名为j_SHA1Input，所以很可能是sha1加密，我们拿j_getByteHash第三次的参数去试试看看是不是标准的sha1：        //参数：aee4c425dbb2288b80c71347cc37d04b0efe89912f3bcf533d21c7c267b8e18cb0255bb1
//结果：a6353f1e64c1304c8bdca7485496dad835ea7c90
```




```
//验证完毕j_getByteHash是标准的sha1，然后转16进制


//那么api_sign就可以确定了：
        //1、就是把请求体的数据进行key=value&key=value...进行拼接
        //2、盐aee4c425dbb2288b80c71347cc37d04b+treeMap拼接后的字符串
        //3、进行一次sha1加密==>hex编码
        //4、盐盐aee4c425dbb2288b80c71347cc37d04b+刚sha1的结果拼接起来再进行一次sha1==>hex编码
        //通过上述4步就能到了api_sign的值，也就是两次sha1加密
```

```
//通过上边的hook，我们知道了是进行了sha1加密，那么我们回过头再去分析一下这个native函数
//书接上回，白发染了将军眉哈哈哈哈哈，忽然想到了的一句歌词===>歌名：千秋月
```

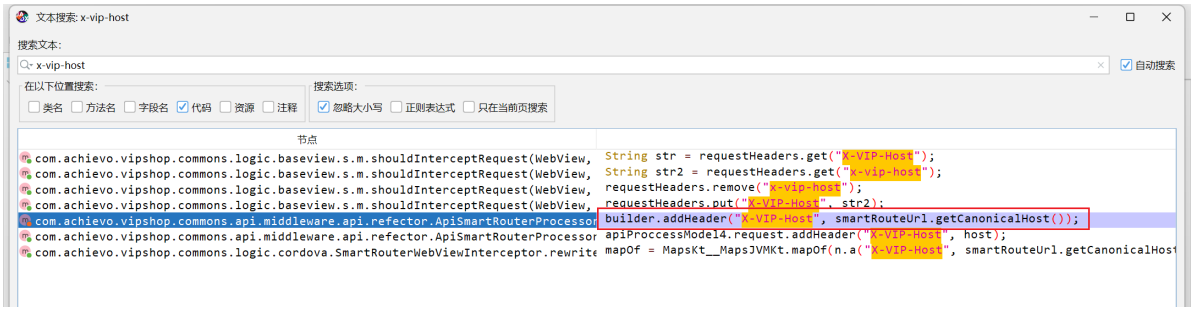
```
//书接上回，我们发现这个函数是静态注册的：
//静态注册的==>
int __fastcall Java_com_vip_vcsp_KeyInfo_gsNav(JNIEnv *env, jclass cla, jobject
context, jobject map, jobject str, jobject z10)
{
  int v9; // r5
  if ( j_Utils_ima(env, cla, context) )
    v9 = j_Functions_gs(env, cla, map, str, z10);
  else
    v9 = 0;
  j_Utils_checkJniException(env);
  return v9;
}

//进来后先去判断了j_Utils_ima(env, cla, context)，然后就执行v9 = j_Functions_gs(env,
cla, map, str, z10);
//这里应该是在检测context是否为空啊什么的，不为空就执行v9 = j_Functions_gs(env, cla,
map, str, z10);
//否则就v9 = 0;然后返回。 显然这里是执行的v9 = j_Functions_gs(env, cla, map, str,
z10);

//j_Functions_gs(env, cla, map, str, z10)：
int __fastcall j_Functions_gs(int env, int cls, int map, int str, int z10)
{
  return Functions_gs(env, cls, map, str, z10);
}
//又调用了：Functions_gs(env, cls, map, str, z10)
//进来Functions_gs(env, cls, map, str, z10)后我们从后往前看，由果溯因：
//先看return==>整篇代码有很多return 0，是在进行其他操作的时候若不符合预判就return 0；只有一
个return v53是我们要的
return v53;
//而v53是通过newStringUTF得到的：
v53 = (*env)->NewStringUTF(env, v59);
//所以我们去看v59怎么来的
const char *v59;
memset(v80, 0, sizeof(v80));
memset(v79, 0, sizeof(v79));
v55 = j_getByteHash(env, jclass, v30, v16, v80, 256);
if (
    v55 && (
        v56 = (const char *)v55,
        v57 = strcpy(v79, dest),
        strcat(v57, v56),
        memset(v80, 0, sizeof(v80)),
        v58 = strlen(v79),
        (v59 = (const char *)j_getByteHash(env, jclass, v79, v58, v80, 256)) !=
0
            )
    )

//我们先看这个if判断表达式做了什么
    //整体来看，是做了一个逻辑与，v55&&(逗号表达式)
    //v55是j_getByteHash的返回值
    //再看逗号表达式：
```

```
    //逗号1：v56 = (const char *)v55 把v55的值赋值给了v56，或者说v56指向了v55所指向的内存区域
    //逗号2：v57 = strcpy(v79, dest)，把dest里边的东西copy到了v79中，dest是啥？跟到最上边通过hook验证为盐，所以这里就是先把盐copy到v79中，而v79是刚刚清空的一个数组
    //逗号3：strcat(v57, v56)：把v56拼接到v57后边，通过刚刚的hook，我们知道v55就是我们java层传来的数据sha1后的结果，然后给了v56.这里又把v56拼接到v57后边了
    //逗号4：memset(v80, 0, sizeof(v80)),给v80清空，看着像存储密文的，刚刚没在意，再去hook一下==>验证完了确实是存密文的
    //逗号5：v58 = strlen(v79),经典操作，取明文长度
    //逗号6：(v59 = (const char *)j_getByteHash(env, jclass, v79, v58, v80, 256)) != 0调用j_getByteHash进行加密


//通过这个if表达式，就对明文进行了加密了。我们再往前追，看看盐怎么来的以及treeMap是怎么进行拼接的
 //先看这个盐，也就是dest，去找到所有使用过dest的地方，他们都是可能对dest做出改变的地方：
memset(v82, 0, sizeof(v82));
memset(dest, 0, sizeof(dest));
v9 = (const char *)j_get_strData(z10_false);
v10 = strlen(v9);
v11 = (const char *)j_Utils_gsigds(v82, v9, v10);
if ( !v11 )
  return 0;
strcpy(dest, v11);
if ( str )
 {
     v12 = j_Utils_jstringtochar(env, str);
     if ( v12 )
     {
         v13 = (char *)v12;
         *(_WORD *)&dest[strlen(dest)] = 38;
         strcat(dest, v13);
         free(v13);
     }
v14 = j_Utils_getDataSize((int)env, (int)cla, (int)map, (int)dest);
v67 = strlen(dest);
qmemcpy(v75, dest, v67);
v57 = strcpy(v79, dest)



//从最后开始看：
    //首先v57 = strcpy(v79, dest)是把dest copy到v79中，不会改变dest的值
    //然后是qmemcpy(v75, dest, v67);是把dest的内容copy到v75中，v67是dest的长度，也不影响
    //接着是v14 = j_Utils_getDataSize((int)env, (int)cla, (int)map, (int)dest); 经过hook验证onEnter和onLeave时dest都是盐，没有发生改变
    //然后是一个if判断，if ( str )，str是java传来的null，这里就是0，所以没有进这个if表达式
    //再往上就是strcpy(dest, v11);    把v11 copy到dest中
    //v11是v11 = (const char *)j_Utils_gsigds(v82, v9, v10);得到的，经过hook验证，其返回值就是v9的值
    //所以v9就是盐，再看v9是v9 = (const char *)j_get_strData(z10_false);得到的，z10_false是java传过来的false，经过hook验证，如果传true那盐就是另外一个。
//在j_get_strData是取了一个bss段的值，这个值是在运行中进行赋值的
```

```
//然后map就是通过调用java层的方法， 然后拼接的
```

x-vip-host

```
//再去看看x-vip-host，直接去搜索关键字把==>找到5条
```




```
//我们直接去看builder.addHeader("X-VIP-Host", smartRouteUrl.getCanonicalHost());
//因为它是builder.addHeader添加请求头的
builder.addHeader("X-VIP-Host", smartRouteUrl.getCanonicalHost());
//它是通过smartRouteUrl.getCanonicalHost()
public String getCanonicalHost() {
    try {
        String host = this.mCanonicalUri.getHost();
        return host == null ? "" : host;
    } catch (Throwable th2) {
        MyLog.error(getClass(), "getCanonicalHost", th2);
        return "";
    }
}
//这里就是对URI.getHost()的安全封装，返回的主机名
```

请求体

```
api_key 23e7f28019e8407b98b84cd05b5aef2c
app_name    shop_android
app_version 7.83.3
bigSaleTagIds
brandIds
brandStoreSns
categoryId
channelId   1
channel_flag    0_1
client  android
client_type android
darkmode    0
deeplink_cps
device_model    Google Pixel
did 0.0.d2e4dbe854df78dc59011b56282bdb21.f89f14
elder   0
```

```
    extParams
{"priceVer":"2","mclabel":"1","cmpStyle":"1","statusVer":"2","ic2label":"1","vid
eo":"2","uiVer":"2","preheatTipsVer":"4","floatwin":"1","superHot":"1","exclusiv
ePrice":"1","router":"1","coupons":"1","needVideoExplain":"1","rank":"2","needVi
deoGive":"1","bigBrand":"1","couponVer":"v2","videoExplainUrl":"1","live":"1","s
ellpoint":"1","reco":"1","vreimg":"1","search_tag":"2","tpl":"1","stdSizeVids":"
","labelVer":"2"}
    fdc_area_id 101102101
    functions
RTRecomm,flagshipInfo,feedback,otdAds,zoneCode,slotOp,survey,hasTabs,floaterPara
ms
    harmony_app 0
    harmony_os  0
    headTabType all
    height  1794
    isMultiTab  0
    keyword 香水
    lastPageProperty    {"isBgToFront":"0","suggest_text":"香
水","scene_entry_id":"-99","refer_page_id":"page_te_globle_classify_search_174804
9002631","text":"香
水","tag":"1","module_name":"com.achievo.vipshop.search","type":"all","typename":
"全部","is_back_page":"0"}
    maker   GOOGLE
    mars_cid    c4acd008-4f30-3599-b32c-7f7dbc2135e9
    mobile_channel  oziq7dxw:::
    mobile_platform 3
    net WIFI
    operator
    os  Android
    osv 10
    otddid
    other_cps
    page_id page_te_commodity_search_1748049007784
    phone_model pixel
    priceMax
    priceMin
    props
    province_id 101102
    referer com.achievo.vipshop.search.activity.TabSearchProductListActivity
    rom Dalvik/2.1.0 (Linux; U; Android 10; Pixel Build/QP1A.191005.007.A3)
    sd_tuijian  0
    service_provider
    session_id  _shop_android_1748048965624
    skey    6692c461c3810ab150c9a980d0c275ec
    sort    0
    source  app
    source_app  android
    standby_id  oziq7dxw:::
    sys_version 29
    timestamp   1748049007
    union_mark  blank&_&blank&_&oziq7dxw:::&_&blank&_&blank
    vipService
    warehouse   VIP_BJ
    width   1080
```

大致看看只有这些是加密的：api_key、did、fdc_area_id、mars_cid、session_id、skey我们先去看这些

api_key

我们多次抓包发现是固定的值，清除数据后也还是这个值，我们先去搜索url看看

```
//搜索url定位到了：
public ApiResponseObj<ProductListBaseResult> getProductList(Context context,
String str) throws Exception {
    Object obj;
    UrlFactory urlFactory = new UrlFactory(true, true, true);
    urlFactory.setService("/shopping/search/product/list/v1");
    urlFactory.setParam("keyword", this.keyword);
    if (SDKUtils.notNull(this.scene)) {
        urlFactory.setParam("scene", this.scene);
    }
    if (!TextUtils.isEmpty(str)) {
        urlFactory.setParam("pageToken", str);
    }
    urlFactory.setParam("brandIds", this.brandIds);
    urlFactory.setParam(VCSPUrlRouterConstants.UriActionArgs.categoryId,
this.lv3CatIds);
    urlFactory.setParam("brandStoreSns", this.brandStoreSn);
    urlFactory.setParam(VCSPUrlRouterConstants.UrlRouterUrlArgs.PROPS,
this.props);
    urlFactory.setParam("priceMin", this.priceMin);
    urlFactory.setParam("priceMax", this.priceMax);
    urlFactory.setParam("vipService", this.vipService);
    urlFactory.setParam("bigSaleTagIds", this.bigSaleTagIds);
    if (!TextUtils.isEmpty(this.selfSupport)) {
        urlFactory.setParam("selfSupport", this.selfSupport);
    }
    if (!TextUtils.isEmpty(this.tabContext)) {
        urlFactory.setParam("tabContext", this.tabContext);
    }
    if (!TextUtils.isEmpty(this.catTabContext)) {
        urlFactory.setParam("catTabContext", this.catTabContext);
    }
    if (!TextUtils.isEmpty(this.imgTabContext)) {
        urlFactory.setParam("imgTabContext", this.imgTabContext);
    }
    urlFactory.setParam("sort", this.sort.intValue());
    if (this.sortTipsType) {
        urlFactory.setParam("sortTipsType", "price");
    }
    if (!TextUtils.isEmpty(this.researchParams)) {
        urlFactory.setParam("researchParams", this.researchParams);
    }
    if (!TextUtils.isEmpty(this.productIds)) {
        urlFactory.setParam("productIds", this.productIds);
    }
    if (SDKUtils.notNull(this.activeType)) {
```

```
        urlFactory.setParam(VCSPUrlRouterConstants.UriActionArgs.activeType,
this.activeType);
    }
    if (SDKUtils.notNull(this.addonPrice)) {
        urlFactory.setParam("addonPrice", this.addonPrice);
    }
    if (SDKUtils.notNull(this.activeNos)) {
        urlFactory.setParam("activeNos", this.activeNos);
    }
    if (SDKUtils.notNull(this.addonProductIds)) {
        urlFactory.setParam("addonProductIds", this.addonProductIds);
    }
    urlFactory.setParam(RemoteMessageConst.Notification.CHANNEL_ID,
this.channelId);
    if (!TextUtils.isEmpty(this.slotOpDoorIds)) {
        urlFactory.setParam("slotOpDoorIds", this.slotOpDoorIds);
    }
    if (SDKUtils.notNull(this.clickedProducts)) {
        urlFactory.setParam("clickedProducts", this.clickedProducts);
    }
    if (!TextUtils.isEmpty(this.isMultiTab)) {
        urlFactory.setParam("isMultiTab", this.isMultiTab);
    }
    if (!TextUtils.isEmpty(this.headTabType)) {
        urlFactory.setParam("headTabType", this.headTabType);
    }
    if (!TextUtils.isEmpty(this.headTabContext)) {
        urlFactory.setParam("headTabContext", this.headTabContext);
    }
    urlFactory.setParam("functions", this.functions);
    Map<String, Object> extParams = getExtParams(context);
    if (SDKUtils.notEmpty(extParams)) {
        urlFactory.setParam(VCSPUrlRouterConstants.UrlRouterUrlArgs.EXT_PARAMS,
JsonUtils.mapToJSON(extParams).toString());
    }
    urlFactory.setParam(ApiConfig.DEVICE_MODEL, this.deviceModel);
    urlFactory.setParam("operator", this.operator);
    if (!TextUtils.isEmpty(c.Q().k())) {
        urlFactory.setParam("extProductIds", c.Q().k());
    }
    if (!TextUtils.isEmpty(this.bsFav)) {
        urlFactory.setParam("bsFav", this.bsFav);
    }
    if (!TextUtils.isEmpty(this.priceTabContext)) {
        urlFactory.setParam("priceTabContext", this.priceTabContext);
    }
    try {
        CpPage cpPage = CpPage.lastRecord;
        if (cpPage != null && (obj = cpPage.pageProperty) != null) {
            urlFactory.setParam("lastPageProperty", obj.toString());
        }
    } catch (Exception e10) {
        MyLog.error(SearchProductListApi.class, e10);
    }
    if (!TextUtils.isEmpty(this.extData)) {
```

```
        urlFactory.setParam("extData", this.extData);
    }
    ApiResponseObj<ProductListBaseResult> apiResponseObj = (ApiResponseObj)
ApiRequest.postHttpResponseType(context, urlFactory, new
TypeToken<ApiResponseObj<ProductListBaseResult>>() {
    }.getType());
    ProductListBaseResult productListBaseResult = apiResponseObj.data;
    if (productListBaseResult != null && productListBaseResult.slotOpData !=
null && SDKUtils.notNull(productListBaseResult.slotOpData.slots) &&
!apiResponseObj.data.slotOpData.slots.isEmpty() && this.slotOpDataListener !=
null) {
        apiResponseObj.data.slotOpData.requestId =
apiResponseObj.getRequestId();

 this.slotOpDataListener.onGetProductSlotOpData(apiResponseObj.data.slotOpData);
    }
    ProductListBaseResult productListBaseResult2 = apiResponseObj.data;
    if (productListBaseResult2 != null &&
productListBaseResult2.nativeSlotOpData != null &&
SDKUtils.notNull(productListBaseResult2.nativeSlotOpData.slots) &&
!apiResponseObj.data.nativeSlotOpData.slots.isEmpty()) {
        apiResponseObj.data.nativeSlotOpData.requestId =
apiResponseObj.getRequestId();
        for (ProductIdsResult.SlotOpNative slotOpNative :
apiResponseObj.data.nativeSlotOpData.slots) {
            slotOpNative.requestId = apiResponseObj.getRequestId();
        }
        IProductSlotOpDataListener iProductSlotOpDataListener =
this.slotOpDataListener;
        if (iProductSlotOpDataListener != null) {

 iProductSlotOpDataListener.onGetProductSlotOpData(apiResponseObj.data.nativeSlo
tOpData);
        }
    }
    if (apiResponseObj.isSuccess()) {
        ProductListBaseResult productListBaseResult3 = apiResponseObj.data;
        if (productListBaseResult3 != null) {
            productListBaseResult3.requestId = apiResponseObj.getRequestId();
        }
        return apiResponseObj;
    }
    throw new DataException();
}
```

```
//可以看到这里边有很多urlFactory.setParam(key, value);
//这都是在设置请求体，但是这里并没有我们的api_key
//我们去全局搜索一下这个关键字api_key==>找到了很多条，但是仔细看大部分都是通过
ApiConfig.getInstance().getApi_key()得到的值，所以我们去hook一下这个参数，打印一下调用栈

//打印完调用栈，我们可以看到它的上一层调用者都是：
com.achievo.vipshop.commons.api.middleware.BaseParamsBuilder.getBaseParams(Sourc
eFile:16)
```

```
//再往上一层就是com.achievo.vipshop.commons.api.middleware.UrlFactory.<init>
(SourceFile:5)

//我们再回来看，在搜到url的这一个方法中，都是通过urlFactory.setParam来设置请求体的，我们看
urlFactory就是一个UrlFactory类的对象。然后我们还可以看到urlFactory是通过UrlFactory
urlFactory = new UrlFactory(true, true, true);得到的

//和我们刚刚打印的堆栈就对上了，我们去看看这个UrlFactory类的构造方法，三个布尔类型参数的构造方
法:
public UrlFactory(boolean z10, boolean z11, boolean z12) {
    this.params = new TreeMap<>(new Comparator<Object>() {
        @Override
        public int compare(Object obj, Object obj2) {
            if (obj == null || obj2 == null) {
                return 0;
            }
            return String.valueOf(obj).compareTo(String.valueOf(obj2));
        }
    });
    this.params.putAll(BaseParamsBuilder.getBaseParams(z10, z11));
    if (z12) {
        BaseParamsBuilder.addOtdParams(CommonsConfig.getInstance().getContext(),
this.params);
    }
}
//进来后先给这个对象的 this.params进行了赋值，直接new了一个treeMap，并且重写了compare方法：
按照key的大小进行排序，a、b、c这样
//然后执行 this.params.putAll(BaseParamsBuilder.getBaseParams(z10, z11));
//也就是把BaseParamsBuilder.getBaseParams(z10, z11)返回的treeMap全部put到this.params
中
//我们去看BaseParamsBuilder.getBaseParams(z10, z11)干了什么，z10、z11、z12是new
UrlFactory(true, true, true)的时候传进来的，都是true，那么在
BaseParamsBuilder.getBaseParams(z10, z11)时传入的就是两个true

//BaseParamsBuilder.getBaseParams(z10, z11):
public static Map<String, String> getBaseParams(boolean z10, boolean z11) {
    TreeMap treeMap = new TreeMap();
    if (z10) {
        treeMap.put("mobile_platform", "3");
        treeMap.put("client", "android");
        treeMap.put(ApiConfig.CLIENT_type, "android");
        treeMap.put("app_name", "shop_android");
        treeMap.put(ApiConfig.SOURCE_APP, "android");
        treeMap.put("app_version", ApiConfig.getInstance().getApp_version());
        treeMap.put(ApiConfig.STANDBY_ID,
ApiConfig.getInstance().getStandbyId());
        treeMap.put(ApiConfig.SYS_VERSION,
ApiConfig.getInstance().getSysVersion());
        treeMap.put("mobile_channel", ApiConfig.getInstance().getStandbyId());
        treeMap.put(ApiConfig.DEEPLINK_CPS,
ApiConfig.getInstance().getDeeplink_standbyId());
        treeMap.put(ApiConfig.OTHER_CPS,
ApiConfig.getInstance().getOther_standbyId());
```

```
        treeMap.put("timestamp",
String.valueOf((ApiConfig.getInstance().getServer_time() +
System.currentTimeMillis()) / 1000));
        treeMap.put("warehouse", ApiConfig.getInstance().getWarehouse());
        treeMap.put(ApiConfig.FDC_AREA_ID,
ApiConfig.getInstance().getFdcAreaId());
        treeMap.put("api_key", ApiConfig.getInstance().getApi_key());
        treeMap.put(ApiConfig.CHANNEL_FLAG, "0_1");
        treeMap.put(ApiConfig.PHONE_MODEL, SDKUtils.getModel());
        String secureKey =
GobalConfig.getSecureKey(CommonsConfig.getInstance().getContext());
        if (!TextUtils.isEmpty(secureKey)) {
            treeMap.put(ApiConfig.SKEY, secureKey);
        }
        treeMap.put("mars_cid", ApiConfig.getInstance().getMid());
        treeMap.put(ApiConfig.PROVINCE_ID,
ApiConfig.getInstance().getProvince_id());
        treeMap.put("page_id", CommonsConfig.getInstance().getPage_id());
        treeMap.put("session_id", CommonsConfig.getInstance().getSession_id());
        treeMap.put(ApiConfig.ROM, SDKUtils.getRom());
        if (ApiConfig.getInstance().isDebug()) {
            treeMap.put(ApiConfig.LOGGINGMODEL, "1");
        }
        if (!TextUtils.isEmpty(ApiConfig.getInstance().getDid())) {
            treeMap.put(ApiConfig.DID, ApiConfig.getInstance().getDid());
        }
        if (SDKUtils.enablePreviewMode) {
            treeMap.put("previewMode", "1");
        }
        if (SDKUtils.enablePreviewMode) {
            long j10 = SDKUtils.previewTime;
            if (j10 > 0) {
                treeMap.put("previewTime", String.valueOf(j10));
            }
        }
        treeMap.put(ApiConfig.SD_TUIJIAN,
String.valueOf(!CommonsConfig.getInstance().isRecommendSwitch()));
        treeMap.put(ApiConfig.DARKMODE,
String.valueOf(CommonsConfig.getInstance().getDarkMode()));
        treeMap.put(ApiConfig.ELDERMODE,
String.valueOf(CommonsConfig.getInstance().isElderMode() ? 1 : 0));
        treeMap.put(ApiConfig.HARMONY_OS,
ApiConfig.getInstance().getHarmonyOSFlag());
        treeMap.put(ApiConfig.HARMONY_APP, "0");
        treeMap.put("referer", CurrentActivityNameHolder.VALUE);
        treeMap.put(ApiConfig.UNION_MARK, UnionMarkUtil.getUnionMark());
    }
    if (z11) {
        Context context = CommonsConfig.getInstance().getContext();
        treeMap.put("width",
String.valueOf(CommonsConfig.getInstance().getScreenWidth()));
        treeMap.put("height",
String.valueOf(CommonsConfig.getInstance().getScreenHeight()));
        treeMap.put("net", SDKUtils.getNetWorkType(context));
        treeMap.put("source", "app");
```

```
        treeMap.put(ApiConfig.SERVICE_PROVIDER,
SDKUtils.getSimOperator(context));
    }
    return treeMap;
}

//仔细看可以发现这里都是我们的请求体，再结合参数名，可以看出这是base请求体。
```

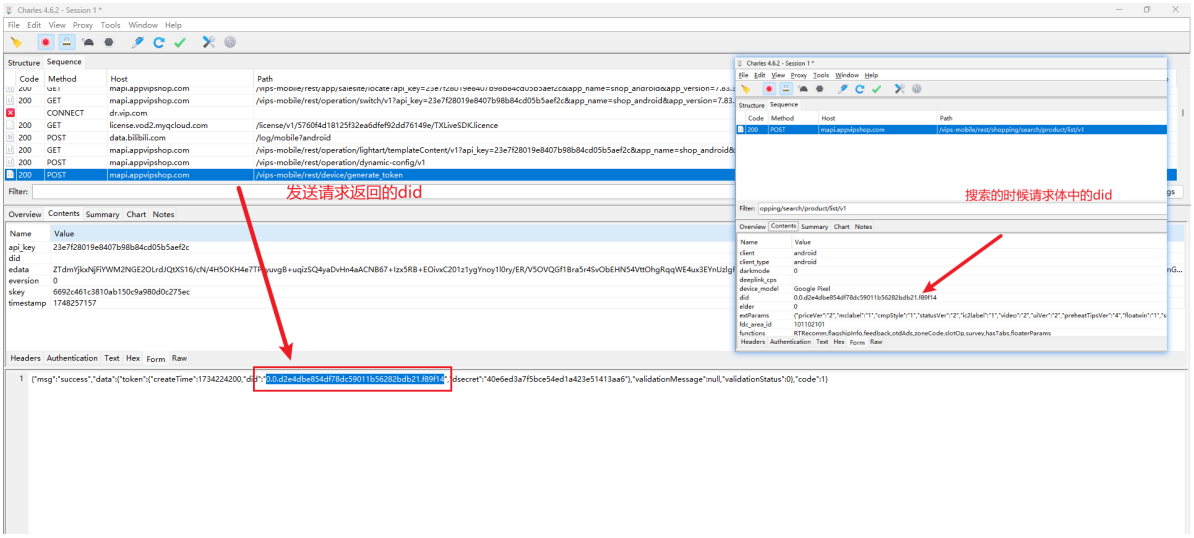
```
//接下来我们去看看api_key是怎么来的：
treeMap.put("api_key", ApiConfig.getInstance().getApi_key());
//再去看ApiConfig.getInstance().getApi_key()
public String getApi_key() {
    if (!TextUtils.isEmpty(this.mApiKey)) {
        return this.mApiKey;
    }
    throw new RuntimeException("You must set api_key param!");
}
//返回的是this.mApiKey
public ApiConfig setApi_key(String str) {
    this.mApiKey = str;
    return this;
}

//我们去看一下谁调用的setApi_key==>发现只有一个用例就是Y方法
public c Y(String str) {
    this.f73028v = str;
    ApiConfig.getInstance().setApi_key(str);
    LogConfig.self().setApi_key(str);
    return Q();
}
//发现参数是调用Y的时候传过来的，所以再看谁调用的Y==>发现有三条用例，去hook确认一下==>有两条调用栈：
//1、com.achievo.vipshop.common.BaseApplication.initAll(SourceFile:8)
//2、com.achievo.vipshop.common.b.p(SourceFile:8)
//我们分别去看一下：
//BaseApplication.initAll：
Y(Config.apikey_vipshop)
public static final String apikey_vipshop = "23e7f28019e8407b98b84cd05b5aef2c";
//发现就是写死的

//再看另一个：
//b.p：
Y(Config.apikey_vipshop)
public static final String apikey_vipshop = "23e7f28019e8407b98b84cd05b5aef2c";
//发现是同样的方式，那就可以说这个api_key就是固定的"23e7f28019e8407b98b84cd05b5aef2c"
```

did-generate_token接口

```
//找了半天发现也在getBaseParams中的：
public static final String DID = "did";
if (!TextUtils.isEmpty(ApiConfig.getInstance().getDid())) {
    treeMap.put(ApiConfig.DID, ApiConfig.getInstance().getDid());
}
//先去判断了.getDid()的返回值，不为空就put进去。去看看getDid()
```

```
public String getDid() {
    return this.mDid;
}

private String mDid;

public void setDid(String str) {
    this.mDid = str;
}

//发现setDid有很多个用例，去hook一下==>找到两条：
//1、 h6.a.b(SourceFile:3)
//2、 hk.c.m0(SourceFile:2)


//先看第一个：h6.a.b
public static void b(Context context, String str) {
    p5.a.c().d(context);
    String a10 = p5.a.c().a();
    ApiConfig.getInstance().setDid(a10);
    b8.b.c(context, str, a10, new b(a10));
}
//a10是did，a10又是通过p5.a.c().a()返回的
public String a() {
    File b10;
    String str = (String) CommonPreferencesUtils.getValueByKey(this.f81906b,
VCSPUrlRouterConstants.UrlRouterUrlArgs.PREFERENCE_DID, String.class);
    if (TextUtils.isEmpty(str) && (b10 = b()) != null) {
        String readFileData = FileHelper.readFileData(b10);
        if (!TextUtils.isEmpty(readFileData)) {
            str = Des3Helper.des3DecodeECB(readFileData, 0);
            if (!TextUtils.isEmpty(str)) {
                ApiConfig.getInstance().setDid(str);
                CommonPreferencesUtils.addConfigInfo(this.f81906b,
VCSPUrlRouterConstants.UrlRouterUrlArgs.PREFERENCE_DID, str);
            }
        }
    }
    return str;
}
//经过hook验证走了String str = (String) CommonPreferencesUtils.getValueByKey(
//最终跟到了：
public static String getString(Context context, String str, String str2, String
str3) {
    Cursor cursor = null;
    try {
        cursor =
context.getContentResolver().query(PreferenceProvider.buildUri(str, str2, 2),
null, null, null, null);
        if (cursor != null && cursor.moveToFirst()) {
            str3 = cursor.getString(cursor.getColumnIndex("value"));
        }
    } catch (Exception e10) {
        MyLog.error(PrefAccessor.class, "getString error", e10);
    }
```

```
    IOUtils.closeQuietly(cursor);
    return str3;
}
//但是jadx反编译不出来了，GDA也不行

//再看另一条调用栈：hk.c.m0(SourceFile:2)
public c m0(String str) {
    this.U = str;
    ApiConfig.getInstance().setDid(str);
    CommonsConfig.getInstance().setDid(str);
    LogConfig.self().setDid(str);
    return this;
}
//然后看调用m0时传进去的是什么：
try {

 hk.c.Q().m0(CommonPreferencesUtils.getStringByKey(BaseApplication.getContextObj
ect(), VCSPUrlRouterConstants.UrlRouterUrlArgs.PREFERENCE_DID));
}

//调用的getStringByKey 也是和走的那个，最终也不行

//后边我去charles中直接搜索了一下，发现是服务器返回的！
```



抓包分析

```
请求网址:https://mapi.appvipshop.com/vips-mobile/rest/device/generate_token
请求方式：POST
请求头：
    authorization   OAuth api_sign=f6982ac2af558e2bfcf8471900afa44be45e7495
    content-type    application/x-www-form-urlencoded
    content-length  1414
    accept-encoding gzip
    user-agent  okhttp/4.9.1
请求体：
    api_key 23e7f28019e8407b98b84cd05b5aef2c
    did
```

```
    edata
ZTdmYjkxNjFiYWM2NGE2OLrdJQtXS16/cN/4H5OKH4e7TPcyuvgB+uqizSQ4yaDvHn4aACNB67+Izx5R
B+EOivxC2O1z1ygYnoy1lOry/ER/V5OVQGf1Bra5r4SvObEHN54VttOhgRqqWE4ux3EYnUzlgFCVn4aq
Wg9oTae2SmOxhKUO3OXrK949lOED7iCpeTPhOOGIic2/2hJtqSAh7hhFvOkYO2xThOPiECkPjF/U41fa
3J9oD9j1anaKBnmG1mVClskFELamAnnn7LsRE2D4m1d/xNxrsFrN5pgxOXVyktlV4UC+egRmjBSZmn4b
bztxI/OurWJutKEzEZgyH/fwbFbsBjfwjDVFVqd55ErbWVCg39o1n5NxvbqdBrTDOhohwh6lqCM4dOL/
qx4Q2ppKEAmjIeKSlCEbIZoQFCZhuZaqfAXVVWeBBpw+bUbUkBQEg8gzGhDU/tOIsKqTc+St7XehUCe7
SbpKlNjdH1TOTg/P2z9BuFICVAfO8jZn4ofkh5IIx+pG6G7ZrCO1WNyLE8KmTISI9Wkp49L+96Usyhx/
gnGUpTDGHMkWHcp78OylPfjFnS7g1qoF9vdW/BImdq8RGBXrz1jvO6yWUF13m1DGalq8f7YjWnPgb7gX
8KWOWIHJP9J3I35oJTbNOkejaYjRAIxWUUc9sxObZYQ2ZOKUxtjuABQZ76u2id53Ulk44lNmapP2CyPz
TP+5Wpbr1OO85UJyhKKOAU7/+oN3RYFG75Z1MokgJvyiwZNsgyOMZyjs94itM6iX3NTgx9U3Nraf/3w2
qJvrZgZeROPNCgJb2UzvqwluabjBIa4sLX5jDRqno6SoCYF21ehB9O/PrlOJ9qml7i1j/OHib5AOMIif
CWWNkXs/YFIerRzfwkDNU8yhFXD3FR4VBN5xh5XUofi13HhQsT7MNSDeqD9XXDvkccuugMp3OgT7FRTs
3UmXw2+/T3tkJHFmGsRTzkTuueH5YvXXq4u7Rsm078fHk9NULEgqPSw6infQ2Y9w1+blDjVXRgIcGFDX
+LUjzmMLkj5ek+CXoGYcrjCvCr5QnP99SMAOMoYEwO1/lt9ILrUpkqYkgtc9DG+1KHMhlXl6r2m0EEwa
IRGkOOBdYXxTf9fdQ8+aVyal4nKbBE1FMPyBLQmnGbZlBOcWTNp9/JMVxVfcv1BvdMEz648vMm02smL6
MHgEhufFskpuay7C
    eversion    0
    skey    6692c461c3810ab150c9a980d0c275ec
    timestamp   1748257157
```

请求头

只有authorization是加密的，这个之前分析过了，不需要再分析了，直接验证一下是不是和之前的加密逻辑一样

明文：

```
api_key=23e7f28019e8407b98b84cd05b5aef2c&did=&edata=ZTdmYjkxNjFiYWM2NGE2OLrdJQtX
S16/cN/4H5OKH4e7TPcyuvgB+uqizSQ4yaDvHn4aACNB67+Izx5RB+EOivxC2O1z1ygYnoy1lOry/ER/
V5OVQGf1Bra5r4SvObEHN54VttOhgRqqWE4ux3EYnUzlgFCVn4aqWg9oTae2SmOxhKUO3OXrK949lOED
7iCpeTPhOOGIic2/2hJtqSAh7hhFvOkYO2xThOPiECkPjF/U41fa3J9oD9j1anaKBnmG1mVClskFELam
Annn7LsRE2D4m1d/xNxrsFrN5pgxOXVyktlV4UC+egRmjBSZmn4bbztxI/OurWJutKEzEZgyH/fwbFbs
BjfwjDVFVqd55ErbWVCg39o1n5NxvbqdBrTDOhohwh6lqCM4dOL/qx4Q2ppKEAmjIeKSlCEbIZoQFCZh
uZaqfAXVVWeBBpw+bUbUkBQEg8gzGhDU/tOIsKqTc+St7XehUCe7SbpKlNjdH1TOTg/P2z9BuFICVAfO
8jZn4ofkh5IIx+pG6G7ZrCO1WNyLE8KmTISI9Wkp49L+96Usyhx/gnGUpTDGHMkWHcp78OylPfjFnS7g
1qoF9vdW/BImdq8RGBXrz1jvO6yWUF13m1DGalq8f7YjWnPgb7gX8KWOWIHJP9J3I35oJTbNOkejaYjR
AIxWUUc9sxObZYQ2ZOKUxtjuABQZ76u2id53Ulk44lNmapP2CyPzTP+5Wpbr1OO85UJyhKKOAU7/+oN3
RYFG75Z1MokgJvyiwZNsgyOMZyjs94itM6iX3NTgx9U3Nraf/3w2qJvrZgZeROPNCgJb2UzvqwluabjB
Ia4sLX5jDRqno6SoCYF21ehB9O/PrlOJ9qml7i1j/OHib5AOMIifCWWNkXs/YFIerRzfwkDNU8yhFXD3
FR4VBN5xh5XUofi13HhQsT7MNSDeqD9XXDvkccuugMp3OgT7FRTs3UmXw2+/T3tkJHFmGsRTzkTuueH5
YvXXq4u7Rsm078fHk9NULEgqPSw6infQ2Y9w1+blDjVXRgIcGFDX+LUjzmMLkj5ek+CXoGYcrjCvCr5Q
nP99SMAOMoYEwO1/lt9ILrUpkqYkgtc9DG+1KHMhlXl6r2m0EEwaIRGkOOBdYXxTf9fdQ8+aVyal4nKb
BE1FMPyBLQmnGbZlBOcWTNp9/JMVxVfcv1BvdMEz648vMm02smL6MHgEhufFskpuay7C&eversion=0&
skey=6692c461c3810ab150c9a980d0c275ec&timestamp=1748257157
```

之前说过是两次sha1，都是在开头加盐:aee4c425dbb2288b80c71347cc37d04b

第一次sha1：

明文：

aee4c425dbb2288b80c71347cc37d04bapi_key=23e7f28019e8407b98b84cd05b5aef2c&did=&edata=ZTdmYjkxNjFiYWM2NGE2OLrdJQtXS16/cN/4H5OKH4e7TPcyuvgB+uqizSQ4yaDvHn4aACNB67+Izx5RB+EOivxC2O1z1ygYnoy1l0ry/ER/V5OVQGf1Bra5r4SvObEHN54VttOhgRqqWE4ux3EYnUzlgFCVn4aqWg9oTae2Sm0xhKU03OXrK949lOED7iCpeTPhOOGIic2/2hJtqSAh7hhFv0kY02xTh0PiECkPjF/U41fa3J9oD9j1anaKBnmG1mVClskFELamAnnn7LsRE2D4m1d/xNxrsFrN5pgxOXVyktlV4UC+egRmjBSZmn4bbztxI/OurWJutKEzEZgyH/fwbFbsBjfwjDVFVqd55ErbWVCg39o1n5NxvbqdBrTDOhohwh6lqCM4dOL/qx4Q2ppKEAmjIeKSlCEbIZoQFCZhuZaqfAXVVWeBBpw+bUbUkBQEg8gzGhDU/t0IsKqTc+St7XehUCe7SbpKlNjdH1TOTg/P2z9BuFICVAfO8jZn4ofkh5IIx+pG6G7ZrC01WNyLE8KmTISI9Wkp49L+96Usyhx/gnGUpTDGHMkWHcp78OylPfjFnS7g1qoF9vdW/BImdq8RGBXrz1jvO6yWUF13m1DGalq8f7YjWnPgb7gX8KWOWIHJP9J3I35oJTbNOkejaYjRAIxWUUc9sxObZYQ2ZOKUxtjuABQZ76u2id53Ulk44lNmapP2CyPzTP+5Wpbr1OO85UJyhKKOAU7/+oN3RYFG75Z1MokgJvyiwZNsgyOMZyjs94itM6iX3NTgx9U3Nraf/3w2qJvrZgZeROPNCgJb2UzvqwluabjBIa4sLX5jDRqno6SoCYF21ehB9O/PrlOJ9qml7i1j/OHib5AOMIifCWWNkXs/YFIerRzfwkDNU8yhFXD3FR4VBN5xh5XUofi13HhQsT7MNSDeqD9XXDvkccuugMp3OgT7FRTs3UmXw2+/T3tkJHFmGsRTzkTuueH5YvXXq4u7Rsm078fHk9NULEgqPSw6infQ2Y9w1+blDjVXRgIcGFDX+LUjzmMLkj5ek+CXoGYcrjCvCr5QnP99SMAOMoYEwO1/lt9ILrUpkqYkgtc9DG+1KHMhlXl6r2m0EEwaIRGkOOBdYXxTf9fdQ8+aVyal4nKbBE1FMPyBLQmnGbZlBOcWTNp9/JMVxVfcv1BvdMEz648vMm02smL6MHgEhufFskpuay7C&eversion=0&skey=6692c461c3810ab150c9a980d0c275ec&timestamp=1748257157
结果：d24a5c3c58fa95e2d3d9fecde30a97d7fffdd6a9

第二次sha1：
明文:aee4c425dbb2288b80c71347cc37d04bd24a5c3c58fa95e2d3d9fecde30a97d7fffdd6a9
结果：f6982ac2af558e2bfcf8471900afa44be45e7495

可以看到和我们抓包拿到的是一致的，没有问题

请求体

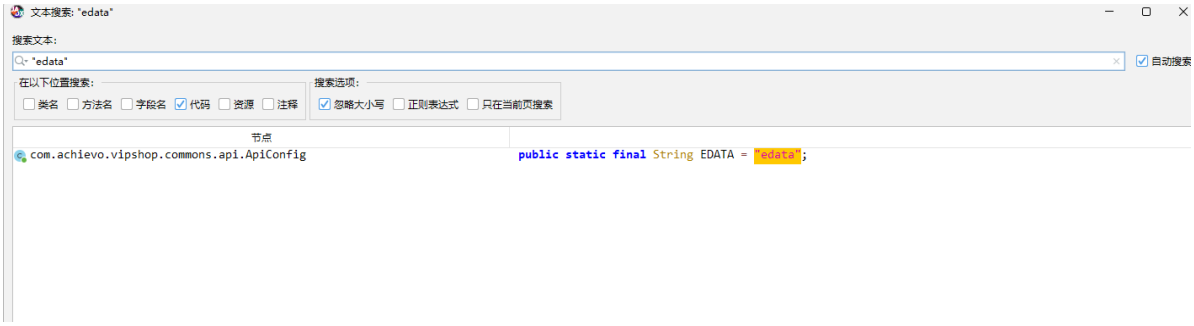
```
    api_key     23e7f28019e8407b98b84cd05b5aef2c
    did
    edata
ZTdmYjkxNjFiYWM2NGE2OLrdJQtXS16/cN/4H5OKH4e7TPcyuvgB+uqizSQ4yaDvHn4aACNB67+Izx5RB+EOivxC2O1z1ygYnoy1l0ry/ER/V5OVQGf1Bra5r4SvObEHN54VttOhgRqqWE4ux3EYnUzlgFCVn4aqWg9oTae2Sm0xhKU03OXrK949lOED7iCpeTPhOOGIic2/2hJtqSAh7hhFv0kY02xTh0PiECkPjF/U41fa3J9oD9j1anaKBnmG1mVClskFELamAnnn7LsRE2D4m1d/xNxrsFrN5pgxOXVyktlV4UC+egRmjBSZmn4bbztxI/OurWJutKEzEZgyH/fwbFbsBjfwjDVFVqd55ErbWVCg39o1n5NxvbqdBrTDOhohwh6lqCM4dOL/qx4Q2ppKEAmjIeKSlCEbIZoQFCZhuZaqfAXVVWeBBpw+bUbUkBQEg8gzGhDU/t0IsKqTc+St7XehUCe7SbpKlNjdH1TOTg/P2z9BuFICVAfO8jZn4ofkh5IIx+pG6G7ZrC01WNyLE8KmTISI9Wkp49L+96Usyhx/gnGUpTDGHMkWHcp78OylPfjFnS7g1qoF9vdW/BImdq8RGBXrz1jvO6yWUF13m1DGalq8f7YjWnPgb7gX8KWOWIHJP9J3I35oJTbNOkejaYjRAIxWUUc9sxObZYQ2ZOKUxtjuABQZ76u2id53Ulk44lNmapP2CyPzTP+5Wpbr1OO85UJyhKKOAU7/+oN3RYFG75Z1MokgJvyiwZNsgyOMZyjs94itM6iX3NTgx9U3Nraf/3w2qJvrZgZeROPNCgJb2UzvqwluabjBIa4sLX5jDRqno6SoCYF21ehB9O/PrlOJ9qml7i1j/OHib5AOMIifCWWNkXs/YFIerRzfwkDNU8yhFXD3FR4VBN5xh5XUofi13HhQsT7MNSDeqD9XXDvkccuugMp3OgT7FRTs3UmXw2+/T3tkJHFmGsRTzkTuueH5YvXXq4u7Rsm078fHk9NULEgqPSw6infQ2Y9w1+blDjVXRgIcGFDX+LUjzmMLkj5ek+CXoGYcrjCvCr5QnP99SMAOMoYEwO1/lt9ILrUpkqYkgtc9DG+1KHMhlXl6r2m0EEwaIRGkOOBdYXxTf9fdQ8+aVyal4nKbBE1FMPyBLQmnGbZlBOcWTNp9/JMVxVfcv1BvdMEz648vMm02smL6MHgEhufFskpuay7C
    eversion     0
    skey        6692c461c3810ab150c9a980d0c275ec
    timestamp   1748257157
```

请求体中一共6个参数 api_key、did、edata、eversion、skey、timestamp

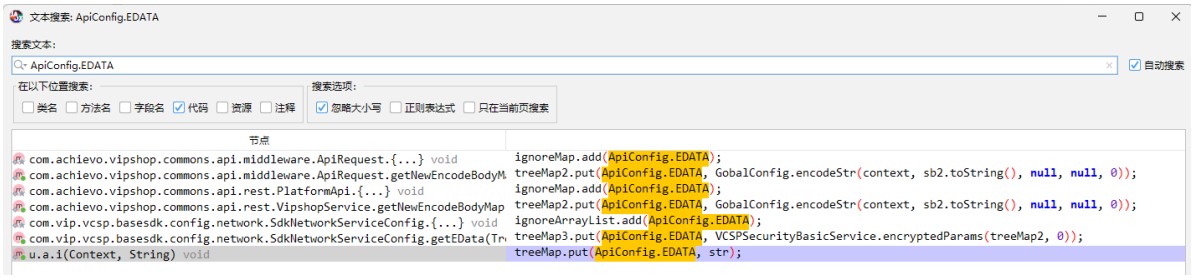
其中api_key分析过，对比了一下和之前的是一样的，不需要在分析了
发送这个请求就是为了拿did，所以在发送请求的时候did还么拿到，所以直接是空
edata需要去逆一下
eversion也去看看
skey也去看看==>32位可能是MD5
timestamp就是秒级时间戳吧
接下来去分析一下

edata

```
//先去看edata，搜索关键字看看

//搜索完发现是定义了个常量，然后调用常量使用的
```




```
//然后我们去搜索ApiConfig.EDATA或者直接查找用例，我这里直接查找用例就会卡死，所以选择去搜索ApiConfig.EDATA
```




```
//可以看到是通过treeMap put进去的，我们直接hook put方法，当key为"edata"时，我们打印他的value以及堆栈
//也可以直接去hook这个两个方法，encodeStr或者encryptedParams 看调用哪个，然后打印堆栈。毕竟只有这两个都去hook一下也不麻烦，jadx还可以生成frida脚本

//这里我直接hook了treeMap的put方法
```

```
key is ==> edata
```

```
value is ==>
MTc1MDM5NGRkYjNhZTM1NRu/+32aui+3Y8OTFuOrbyCsk25IFm6Q4JrZbp21UoQ8+SryauWDKDVkjflD
59sTSW4G/pkaH6ZnXwLOcvZGK5NzP1uD4E+wzDOkw5tmRnhpOnCsUkNNZ4bjEV7xx1SsApps2FTMBrFy
ighVB8LNmmvX//KcpYJZIyz4kSIWWxYVT7f0g8K0xt94zLzAV+6TxCNxM8s4lZvSg/d+0bIVWSFCKcHE
rORLf25S+43n7/pOs6kOOmOHHLCRMKP9iUAONsfbA+1m93Grrrvp4Ypavon3DfSHxK26Ao8OX/JP1Mpq
A32jBeRn9FHULHkdbksRXbzT46vzi1mcAo7dk4LZcX22EIfWOqoWAGQd32+73/vEr4AtuKNzfjLj+PwR
Sct/IphInRWOOBA8cTOiHUADAkky2nUQ5rZ24kF9eMtwYuARtPwsqb5Xly8hU45vjmxxWZQfE5frn2S2
e8NyaSqT4IFWMsilNpJX78rQhWzFlfEOetD+eq4umNsrIyFP2qze+7NZ2SC41EPD54YiW+oA4P68PNCz
dP5pOe38emVciJ9ULe49PBxkuaK4V4sk43v8u8lS1naZnHDVB2OnfcaSTu7T9D/k/7+6LksrMIodbS2n
vImCmtrqS5Lz+bSTA4H+5WcMie9rBseBAm4fDlnEXxPYKP7sOPkrOtM+hahY/8agL7RvxPHQVjvOj/LX
wnrXXcd+mjqXiB2zKnsIiErZEzCrxzjGo/dSpsZCUwd9CPvGIN+m+rpmC7pozwNrhtGSsIVEshOv3M1y
pD2D1NNQOGsqjftG4uYac1CJQdrQlwe5+Z48brTYuRX4rqhbfJ+pNpNk7CwrGqo4d2Xoy4RHZo1bZ4lZ
pTVx7JeQ7kwB5Hnke4mL85dH2kjBuHBdQLBKgbubhkRRpIfNvJrOuW1csYsGlb933ytTI7FCK6+Uzw53
gTXTNCWtGpPQz/sKr8Lo3NdaO5zu8CrffsL+OFh4USaQzBO2bNGUs8ffF/gmwksIiYelOIi3vFRBvPbq
CkIXAs/ogoOkJnGcOm3iNNi5tX8wIAinXGBKklr6RF4vCRAgSHKz5gGv8UV1QsVbp2CnrKLNwbBO8EWU
r9w7H/5rkhgs1vAD2A3/jxzinb6B+nH194LN2Q/Tp4ui/Y4r3VCb9LKrDcvWHw82V/nOOrkUufs2nkwl
XcT8OMl13QiyQheJ
java.lang.Throwable
        at java.util.TreeMap.put(Native Method)
        at
com.vip.vcsp.basesdk.config.network.SdkNetworkServiceConfig.getEData(SourceFile:
12)
        at
com.vip.vcsp.network.refector.VCSPApiBodyProcessor.process(SourceFile:13)
        at
com.vip.vcsp.network.refector.VCSPApiStepProcessor.execute(SourceFile:6)
        at com.vip.vcsp.network.api.VCSPNetworkService.netReq(SourceFile:14)
        at com.vip.vcsp.network.api.VCSPNetworkService.doPost(SourceFile:8)
        at
com.vip.vcsp.basesdk.config.security.SecurityServiceConfigProxy.request(SourceFi
le:8)
        at
com.vip.vcsp.security.model.VCSPSecurityNetRequest.getDid(SourceFile:6)
        at
com.vip.vcsp.security.api.VCSPSecurityBasicService$4.call(SourceFile:4)
        at
com.vip.vcsp.security.api.VCSPSecurityBasicService$4.call(SourceFile:1)
        at c.g$i.run(SourceFile:3)
        at
java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1167)
        at
java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:641)
        at java.lang.Thread.run(Thread.java:919)
```

```java
//对比了一下和charles抓到的是一致的，直接去看调用栈，去找getEData方法==>
public Map<String, String> getEData(TreeMap<String, String> treeMap) {
    if (treeMap == null || treeMap.isEmpty()) {
        return null;
    }
    TreeMap treeMap2 = new TreeMap();
    TreeMap treeMap3 = new TreeMap();
    for (Map.Entry<String, String> entry : treeMap.entrySet()) {
        if (entry != null) {
            if (ignoreArrayList.contains(entry.getKey())) {
```

```
                treeMap3.put(entry.getKey(), entry.getValue());
            } else {
                treeMap2.put(entry.getKey(), entry.getValue());
            }
        }
    }
    treeMap3.put(ApiConfig.EVERSION, "0");
    try {
        BaseSDK.getSecurityBasicService();
        treeMap3.put(ApiConfig.EDATA,
VCSPSecurityBasicService.encryptedParams(treeMap2, 0));
    } catch (Exception e10) {
        VCSPMyLog.error(SdkNetworkServiceConfig.class, e10);
    }
    return treeMap3;
}
//去hook一下getEdata方法， 看看入参和返回值：

//入参:
SdkNetworkServiceConfig.getEData is called: treeMap=
{api_key=23e7f28019e8407b98b84cd05b5aef2c, app_name=shop_android,
app_version=7.83.3, client_type=android, did=, dinfo=
{"ah1":"","ah2":"","ah3":"","ah4":"wifi","ah5":"1080_1794","ah6":1593600,"ah7":4
,"ah8":3948359680,"ah9":"Pixel","ah10":"","ah11":"","ah12":"","ah13":"","as1":"1
0","as2":"","as3":"","as4":"87cd3a74b10f74a1","as5":"","as6":"","as7":"29","ac1"
:"c4acd008-4f30-3599-b32c-7f7dbc2135e9"}, mars_cid=c4acd008-4f30-3599-b32c-
7f7dbc2135e9, phone_model=Pixel, session_id=c4acd008-4f30-3599-b32c-
7f7dbc2135e9_shop_android_1748336917850, skey=6692c461c3810ab150c9a980d0c275ec,
sys_version=29, timestamp=1748336918, vcspKey=4d9e524ad536c03ff203787cf0dfcd29,
vcspToken=NGQ5ZTUyNGFkNTM2YzAzZmYyMDM3ODdjZjBkZmNkMjl8fHwxNzUwOTI4OTQ1fHx8.e3d5f
5ca7bcf3c4009f946faed1bd38b}
//返回值:
SdkNetworkServiceConfig.getEData result=
{api_key=23e7f28019e8407b98b84cd05b5aef2c, did=,
edata=ZDMxY2Y1MDk5N2Y2YWYxMc+r8rrllO+CinxuOjcXmLqOKAAbnWcmvqATMdT8DULYJV6bOP9Q7J
QPcO6kQo7FExnOTJn9bKvzq7keobsFUBN4c5z5StRdrDmLiOurDI9ytWuvj5EryzoXiw+O1IfToxMiTT
J/xtODOE3jjPPfrDhJM1jfRzWjlOUFCbbrVnA1QRbRkfbqGf/F7nOPW17Ek/fLddLG3fD5OyLi4oTJMO
j2L/e8Ozee3Dlu1B7eRYRQMzQ8STOM6rNOjxqvi6VeO9IjV+FEIBlzN3geC25fsY22eK7cRMBsn
ecHAICqfQECZ+XEsMwxD/gE/pz6fb4387BOcAcgG98BsEyYR7cDySo/Qe/WZ143+/F47QcGmhax+4p8z
RHVHOHgZmUVPco1VrZJd6ph/aBz11iTIsGUXdKZsaajQQxV2N84AR6tz4nD8XjhTLh9//43/MZhOluf6
GeA1ZpTtt6y1njrmOzTA2whelfgcUC1XwQFWuKZNHn
dJPx8mwYTd/1U5X18QFRH2KKBy1rO/GiIGUt2PQa3nBXefCSncwGjaQsmcZbWeNZkbt75pZR7PO+lIKd
yxWwOgZBoDVz92R2Z7OIE/G3XfrhL9pepe8FMZvt/tGkWiE2OylSyhl76afZxIjfSgm0WVIuE97Aa+nz
rVI5ZvOVCSWk+1ouZJI5ckPD3pa9LXVrGcOlR+aati
/MfJFOrhv9ikDYrMgDSIZikZeR6KmftSDdcEiAXdJdeZjEZWk1ijpBLGLTj5bf1UlFrHK88ev1uI2I+c
hAsHaCSgd/S6BLb/V8gdx5yl6OEuOC2d58a8vCOQjjM/+2OawHa6LrUqPSoqyVAgszuSWy0tCZNogFc/
YvlKWz/HLkP7Cov+ry5vcCWp5MNo8apUn+3mv3jvGz
h1z+gSt19+BpwbPFcj4vzUdkSszAP9G/c5sAneb/rlGdwjxhQZuN86jyChLOhqLEfGa33lk7sCvf2m1U
244jopAD4+tItuhVEwFzBZQaHCHHSMLiWHzkZD7iJdET82DTMZKsfcVelZv3oRivYrhGLK9RGwqvQURY
fsBpdVdoOVXT6hmQ/UDpxJG4GpnHyArZmW/K9k5hIU0taQ946dlHxJR49yJylVTXfnH5EL8MCsPESC+t
vLDkQSSV+BhfhhzBmIFHf+VAj1u5GPPKact5FDKgDXRelRCRWHqkzKwjpRuNb, eversion=0,
skey=6692c461c3810ab150c9a980d0c275ec, timestamp=1748336918}
```

```
//可以看到，在入参的时候是一个treeMap，里边有很多东西。在执行完getEData后返回的treeMap3中，
就只有我们发送请求时的参数了

//我们去看下getEData的逻辑，分析一下edata是在哪里put的，其值又是从何如来!
public Map<String, String> getEData(TreeMap<String, String> treeMap) {
    if (treeMap == null || treeMap.isEmpty()) {
        return null;
    }
    TreeMap treeMap2 = new TreeMap();
    TreeMap treeMap3 = new TreeMap();
    for (Map.Entry<String, String> entry : treeMap.entrySet()) {
        if (entry != null) {
            if (ignoreArrayList.contains(entry.getKey())) {
                treeMap3.put(entry.getKey(), entry.getValue());
            } else {
                treeMap2.put(entry.getKey(), entry.getValue());
            }
        }
    }
    treeMap3.put(ApiConfig.EVERSION, "0");
    try {
        BaseSDK.getSecurityBasicService();
        treeMap3.put(ApiConfig.EDATA,
VCSPSecurityBasicService.encryptedParams(treeMap2, 0));
    } catch (Exception e10) {
        VCSPMyLog.error(SdkNetworkServiceConfig.class, e10);
    }
    return treeMap3;
}
//首先是new了两个TreeMap，其名为:treeMap2与treeMap3。==>就是把传进来的treeMap分成两个
treeMap
//最终返回的是treeMap3，treeMap3中的数据就是发送请求时携带的数据
//然后会判断传进来的这个treeMap中的key是否在ignoreArrayList列表中，若在就把key和value put
到treeMap3中，不在就put进treeMap2中
//然后再给treeMap3 put("eversion", "0");
//然后再给treeMap3 put("edata", VCSPSecurityBasicService.encryptedParams(treeMap2,
0));==>edata就是对treeMap2进行加密后put进去
//然后我们去看是如何对treeMap2加密得到的edata的:
public static String encryptedParams(Map map, int i10) throws Exception {
    StringBuilder sb2 = new StringBuilder();
    for (Map.Entry entry : map.entrySet()) {
        if (entry != null) {
            if (sb2.length() > 0) {
                sb2.append("&");
            }
            sb2.append((String) entry.getKey());
            sb2.append("=");
            sb2.append(VCSPSDKUtils.urlEncode((String) entry.getValue()));
        }
    }
    return sb2.length() > 0 ?
VCSPSecurityConfig.encodeStr(VCSPCommonsConfig.getContext(), sb2.toString(),
i10) : "";
}
//进来后会先对treeMap2中的value进行url编码然后将key=value&key=value...拼到sb2中，.
```

```
//然后调用VCSPSecurityConfig.encodeStr(VCSPCommonsConfig.getContext(),
sb2.toString(), i10)
public static String encodeStr(Context context, String str, int i10) {
    try {
        VCSPMyLog.info(VCSPSecurityConfig.class, "VCSPSecurityConfig
encodeStr");
        return es(context.getApplicationContext(), str, null, null, i10);
    } catch (Throwable th2) {
        VCSPMyLog.error(VCSPSecurityConfig.class, th2);
        return null;
    }
}
//进来后调用es(context.getApplicationContext(), str, null, null, i10);
private static String es(Context context, String str, String str2, String str3,
int i10) {
    try {
        if (clazz == null || object == null) {
            synchronized (lock) {
                initInstance();
            }
        }
        if (esMethod == null) {
            esMethod = clazz.getMethod("es", Context.class, String.class,
String.class, String.class, Integer.TYPE);
        }
        return (String) esMethod.invoke(object, context, str, str2, str3,
Integer.valueOf(i10));
    } catch (Exception e10) {
        e10.printStackTrace();
        return "Exception es: " + e10.getMessage();
    } catch (Throwable th2) {
        th2.printStackTrace();
        return "Throwable es: " + th2.getMessage();
    }
}
//进入es后又是反射调用了KeyInfo下的es，把Context、拼接后的treeMap2、null、null、0传进去了
//接着去看KeyInfo下的es
public static String es(Context context, String str, String str2, String str3,
int i10) {
    try {
        try {
            return esNav(context, str, str2, str3, i10);
        } catch (Throwable th2) {
            return "KI es: " + th2.getMessage();
        }
    } catch (Throwable unused) {
        SoLoader.load(context, LibName);
        return esNav(context, str, str2, str3, i10);
    }
}
//进来后又调用了esNav(context, str, str2, str3, i10); 传进去的是Context、拼接后的
treeMap2、null、null、0
//去看esNav
private static native String esNav(Context context, String str, String str2,
String str3, int i10);
```

```
//是一个native方法

//总的来说，就是把treeMap2拼接后，调用esNav这个native方法，传进去的就是Context、拼接后的treeMap2、null、null、0
//这就是getEData大体逻辑


//到这里，我们要想搞定edata这个参数就需要去看esNav这个native方法，以及treeMap2中的值是怎么来的。
//我们先去搞清楚这个esNav是如何进行加密的把
//先hook一下esNav，打印一下参数，方便主动调用
```

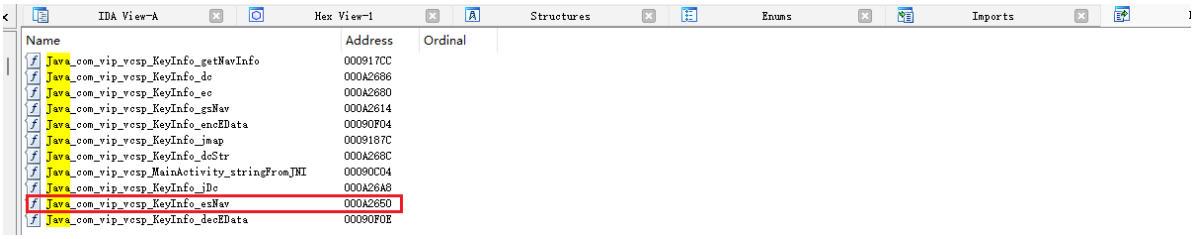
```
 KeyInfo.esNav is called:
//参数1:
 context=com.achievo.vipshop.common.VipApplicationLike@6a5144c,
//参数2:
str=app_name=shop_android&app_version=7.83.3&client_type=android&dinfo=%7B%22ah1%22%3A%22%22%2C%22ah2%22%3A%22%22%2C%22ah3%22%3A%22%22%2C%22ah4%22%3A%22wifi%22%2C%22ah5%22%3A%221080_1794%22%2C%22ah6%22%3A1593600%2C%22ah7%22%3A4%2C%22ah8%22%3A3948359680%2C%22ah9%22%3A%22Pixel%22%2C%22ah10%22%3A%22%22%2C%22ah11%22%3A%22%22%2C%22ah12%22%3A%22%22%2C%22ah13%22%3A%22%22%2C%22as1%22%3A%2210%22%2C%22as2%22%3A%22%22%2C%22as3%22%3A%22%22%2C%22as4%22%3A%2287cd3a74b10f74a1%22%2C%22as5%22%3A%22%22%2C%22as6%22%3A%22%22%2C%22as7%22%3A%2229%22%2C%22ac1%22%3A%22c4acd008-4f30-3599-b32c-7f7dbc2135e9%22%7D&mars_cid=c4acd008-4f30-3599-b32c-7f7dbc2135e9&phone_model=Pixel&session_id=c4acd008-4f30-3599-b32c-7f7dbc2135e9_shop_android_1748338429027&sys_version=29&vcspKey=4d9e524ad536c03ff203787cf0dfcd29&vcspToken=NGQ5ZTUyNGFkNTM2YzAzZmYyMDM3ODdjZjBkZmNkMjl8fHwxNzUwOTMwNDU2fHx8.cdf5e1e7fc60cbd19f154929191d32ce,
//参数3:
str2=null,
//参数4:
str3=null,
//参数5:
i10=0

//返回值
YTNiNzVkMmQ3NjgyMDYyNDDYOd0UGZl+Vwz9VHWKA4VF18fGIcItwTA/7q9eEd/aVCEQpTDGQeCdXEb+j2IyiT9VXbpkVQMP3IFqKD3zmn3971hOne9D5XNF5xYvDY1ODxkWAK32uOSTxkWSt5F1jQBFtVEcttdnlC1CUfnJe4zN+SGEPnj3n
7N/EcogjRA2s9TRCeOs1KFhs8Eja+0azYI+cp7nET3b/sdFrDU9kET2sk8N2cAqcwu1M17j+VNumWip0OCAY5CGSZC5jLXYbwQXZO+ZBmQg2uQy78Qd8YRD3imeeE1K8snrR4Vy0ABPBozWsGXjAGMTeovksFpW+8KpEeN28yAwvn6CkKnFiziEoPs+NW4SAYWKB2+5/8Z
nLt+ra7NM6Opz3dxUCkJT+plTrvDoB6jP0KNJpn1OFR/53EBNi8sl36blR1XfXibKjluFuXvr7936m2vqZE6COOJOqj/Ng2DaA9urT5lZCq/3MyCd5oWAuf4IjfwlMKyvCZFBLFaky8YD6lEedMbqL3iZS7qp5ODkEElLu1KCKiww9AihbZHwpU856Kck1oUg1916WtjLx
cD7zG6LBKGYsif28C2RMd2t8Ag7fMgmpxeUTzr19ouIFZWlbeV58elUiXYDaTUs5UedmYVxkPDKpktWE5cXCj3qTY7oQmDqxqJieLQFCjXvLR7MDdiAixxlmX6p7GG4bYwDDeG7LhJ6uMRnuXikrJFC6tK0zIW47hLXIg6uhw/L9heUD9YwHtIWseAeOTEDnlwbKaVXd6M
LhesV9kIlOF8HJ9836zdFxBDTftjolWT+plAKc4Dtz+pUN75vwwQkyaMVlG+iDwINT4jrY1MUNdgfMItSuBy2OdlVUsyPlHZ5u3YGs+gF5nJrqfnUFlEClkV5B/sroW5syvijCznOSjqJZ/KH/filqeywUxFFSic79fF45zYMh+LlInfYKF/gmMsY8AZPvC6aIpAOxVhlx
eLM5GpjkYuwFL7NT8+EkjZzO382kjIUn+bqWfAnnMQiowa7xIFfHMtcx1LG2VO2+pqZd4N0EkaKn9u2xSNOL3ssd6iYhg32MJWuOtePlR4H22jPeMBGUPdW7mozPgYIrwEEMVdJt5Ga2zshzQPhhDeto67yhRY7VOjOcOvwHQJo7gGt3rCdgVhgXdYPlQMOyrOmpur1Fdthr/TC8OlURpupqp1g65rWTjex
```

//现在去写一个主动调用esNav的脚本==>主动调用后发现，每次返回的值不一样，说明加了变化的值，可能是时间戳我们直接用for循环多hook几次，看看1秒内的值是否一样，这样可以判断是否是加了秒级时间戳

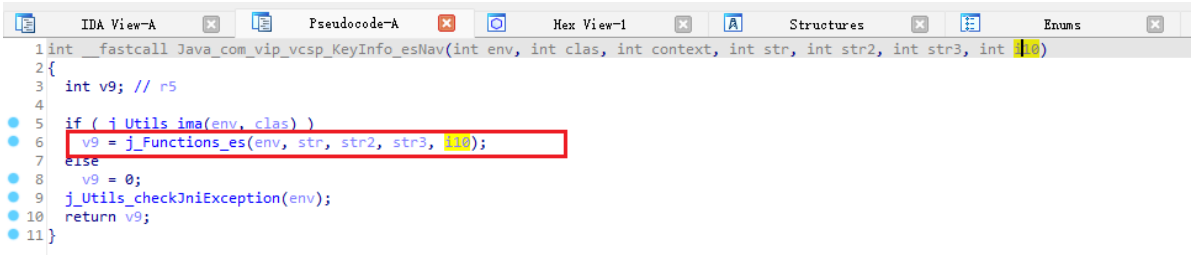
//可以看到在1秒内，其返回值是相同的，超出1秒后就变了，这就说明是只加了一个秒级时间戳然后进行加密的

//接下来我们去找esNav是在哪个so中，先看卡这个类中有加载so文件吗

```
static {
    try {
        System.loadLibrary("keyinfo");
    } catch (Throwable th2) {
        th2.printStackTrace();
    }
}
//我们直接去找这个so，libkeyinfo.so看看里边有没有对esNav进行注册
private static native String esNav(Context context, String str, String str2,
String str3, int i10);
```



//进来后看到是静态注册的，跳进去看看



//进来后看到又调用了v9 = j_Functions_es(env, str, str2, str3, i10);

//去看看j_Functions_es的逻辑：

//最终调用了 return Functions_es：(env, str, str2, str3, i10);

//直接去分析Functions_es==>由果溯因，先看所有的return

```
return 0;
return v45;
//显然是return的v45==>看v45怎么来的

if ( v52 )
{
    v45 = (*env)->NewStringUTF(env, v52);
    free(v52);
}
//v45是把v52转为了jstring

v52 = (char *)j_base64_encode((char *)v39, v53); //其中v53是v39的长度
//v52是把v39进行base64编码，看下j_base64_encode是不是标准的base64==>是一个标准的base64
//那v39就是密文，追v39
v46 = strlen((const char *)v60);
qmemcpy(v39, v60, v46);
qmemcpy((char *)v39 + v46, v48, v47);
v45 = 0;
*((_BYTE *)v39 + v53) = 0;
//两次拷贝，第一次把v60拷贝到v39这里，其中v46是v60的长度；第二次拷贝，把v48拷贝进v39+v46的地方，相当于拼接到v39后边，v47就是v48的长度
//也就是说v39中存的是v60以及v48
//接下来的任务就是找到v60和v48是如何来的
//先看v60:
__int64 v60[4];
v60[0] = 0LL;
v60[1] = 0LL;
v60[2] = 0LL;
v60[3] = 0LL;
v46 = strlen((const char *)v60);
qmemcpy(v39, v60, v46);
//可以看到v60就是一个长度为4的空数组，那么它就相当于没有拼接，因为strlen(v60)的时候是0，qmemcpy了长度为0个单位。所以v39中就只有v48

//接下来去看v48：
v48 = (*env)->GetByteArrayElements(env, v44, 0);
//是把取的v44这个字节数组中的元素，去看v44
v44 = (*env)->CallObjectMethod(env, v29, v43, v41);
//v44是调用的java层的方法拿到的
```

esNav的核心：固定key：cdd17ab29b84b32552ddcfbb4abf0225(hex编码)随机iv 的AES/CBC/PKCS5Padding加密。先对传进去的明文进行AES/CBC/PKCS5Padding加密。加密完后把iv拼到密文前边然后再进行base64编码，至于为什么1秒内的结果相同，那就是随机iv是通过时间戳搞出来的咯。分析过程请转战B站==>爬尽天下虫

```
//到这里就是知道是在so中如何进行加密的了，接下来就是要搞清楚他的明文是什么了，接下来我们回到getEdata中回顾一下：
public Map<String, String> getEData(TreeMap<String, String> treeMap) {
    if (treeMap == null || treeMap.isEmpty()) {
        return null;
    }
    TreeMap treeMap2 = new TreeMap();
    TreeMap treeMap3 = new TreeMap();
    for (Map.Entry<String, String> entry : treeMap.entrySet()) {
```

```
        if (entry != null) {
            if (ignoreArrayList.contains(entry.getKey())) {
                treeMap3.put(entry.getKey(), entry.getValue());
            } else {
                treeMap2.put(entry.getKey(), entry.getValue());
            }
        }
    }
    treeMap3.put(ApiConfig.EVERSION, "0");
    try {
        BaseSDK.getSecurityBasicService();
        treeMap3.put(ApiConfig.EDATA,
VCSPSecurityBasicService.encryptedParams(treeMap2, 0));
    } catch (Exception e10) {
        VCSPMyLog.error(SdkNetworkServiceConfig.class, e10);
    }
    return treeMap3;
}

//edata的明文是先对treeMap2中的value进行url编码然后进行key=value&key=value...拼接
//它的明文是从上层传进来的treeMap中取出来的，其中传进来的treeMap中有这些参数：
{
//api_key=23e7f28019e8407b98b84cd05b5aef2c,
app_name=shop_android,
app_version=7.83.3,
client_type=android,
//did=,
dinfo=
{"ah1":"","ah2":"","ah3":"","ah4":"wifi","ah5":"1080_1794","ah6":1593600,"ah7":4
,"ah8":3948359680,"ah9":"Pixel","ah10":"","ah11":"","ah12":"","ah13":"","as1":"1
0","as2":"","as3":"","as4":"87cd3a74b10f74a1","as5":"","as6":"","as7":"29","ac1"
:"c4acd008-4f30-3599-b32c-7f7dbc2135e9"},
mars_cid=c4acd008-4f30-3599-b32c-7f7dbc2135e9,
phone_model=Pixel,
session_id=c4acd008-4f30-3599-b32c-7f7dbc2135e9_shop_android_1748336917850,
//skey=6692c461c3810ab150c9a980d0c275ec,
sys_version=29,
timestamp=1748336918,
vcspKey=4d9e524ad536c03ff203787cf0dfcd29,
vcspToken=NGQ5ZTUyNGFkNTM2YzAzZmYyMDM3ODdjZjBkZmNkMjl8fHwxNzUwOTI4OTQ1fHx8.e3d5f
5ca7bcf3c4009f946faed1bd38b
}
//把api_key、did、skey、timestamp put进了treeMap3.其余的都put进了 treeMap2. treeMap2
就是edata的明文
//接着去分析treeMap2中的参数：app_name、app_version、client_type、dinfo、mars_cid、
phone_model、session_id、sys_version、vcspKey、vcspToken
```

edata明文

```
//这些是需要解决的app_name、app_version、client_type、dinfo、mars_cid、phone_model、
session_id、sys_version、vcspKey、vcspToken
```

//直接去看getEdat的上一层，看看调用getEdata时如何构造的参数treeMap

```
    public boolean process() {
        TreeMap<String, String> treeMap;
        Map<String, String> vcspToken;
        if (checkParamValidate()) {
            if (this.processParam.isPost) {
                TreeMap<String, String> treeMap2 = new TreeMap<>();
                VCSPNetworkServiceConfig.IVcspTokenConfig iVcspTokenConfig = this.networkServiceConfig.iVcspTokenConfig;
                if (iVcspTokenConfig != null && (vcspToken = iVcspTokenConfig.getVcspToken()) != null) {
                    treeMap2.putAll(vcspToken);
                }
                TreeMap<String, String> treeMap3 = this.networkServiceConfig.getNetworkParam().commonMap;
                if (treeMap3 != null) {
                    treeMap2.putAll(treeMap3);
                }
                TreeMap<String, String> treeMap4 = this.processParam.commonMap;
                if (treeMap4 != null) {
                    treeMap2.putAll(treeMap4);
                }
                if (this.networkServiceConfig.iEDataConfig != null && (treeMap = this.processParam.eDataMap) != null && !treeMap.isEmpty()) {
                    treeMap2.putAll(this.processParam.eDataMap);
                    Map<String, String> eData = this.networkServiceConfig.iEDataConfig.getEData(treeMap2);
                    if (eData != null) {
                        this.processParam.bodyDataMap.clear();
                        this.processParam.bodyDataMap.putAll(eData);
                        return true;
                    }
                }
                TreeMap<String, String> treeMap5 = this.networkServiceConfig.getNetworkParam().bodyDataMap;
                if (treeMap5 != null) {
                    treeMap2.putAll(treeMap5);
                }
                this.processParam.bodyDataMap.putAll(treeMap2);
            }
            return true;
        }
        return false;
    }
}
```

```
//是这样调用的getEdata:
 Map<String, String> eData =
this.networkServiceConfig.iEDataConfig.getEData(treeMap2);
if (eData != null) {
    this.processParam.bodyDataMap.clear();
    this.processParam.bodyDataMap.putAll(eData);
    return true;
}
//可以看到通过调用getEdata拿到结果后，会判断这个结果eData是否为null，不为null就把拿到的eDat
putAll进bodyDataMap中，这个eData就是最终的请求体

//其中这个treeMap2就是传进getEdata的参数，我们去看它怎么构建的
//treeMap2是通过几次putAll得到的，直接去hook 这个putAll方法，打印堆栈，发现在process方法
中，调用了8次，其中在调用getEdata之前的一共5次，，如下：
```

在process方法的第8、10、12行调用了putAll==>12行之前都是没调用getEdata的

第8行：

```
===== Hook TreeMap.putAll =====
params is ==> {api_key=23e7f28019e8407b98b84cd05b5aef2c, app_name=shop_android,
app_version=7.83.3, client_type=android, mars_cid=c4acd008-4f30-3599-b32c-
7f7dbc2135e9, phone_model=Pixel, session_id=c4acd008-4f30-3599-b32c-
7f7dbc2135e9_shop_android_1748479822213, skey=6692c461c3810ab150c9a980d0c275ec,
sys_version=29, timestamp=1748479822, vcspKey=4d9e524ad536c03ff203787cf0dfcd29}
java.lang.Throwable
    at java.util.TreeMap.putAll(Native Method)
    at com.vip.vcsp.network.refector.VCSPApiBodyProcessor.process(SourceFile:8)
```

第10行：

```
===== Hook TreeMap.putAll =====
params is ==> {}
java.lang.Throwable
```

```
    at java.util.TreeMap.putAll(Native Method)
    at com.vip.vcsp.network.refector.VCSPApiBodyProcessor.process(SourceFile:10)


第8行：
===== Hook TreeMap.putAll =====
params is ==> {api_key=23e7f28019e8407b98b84cd05b5aef2c, app_name=shop_android,
app_version=7.83.3, client_type=android, mars_cid=c4acd008-4f30-3599-b32c-
7f7dbc2135e9, phone_model=Pixel, session_id=c4acd008-4f30-3599-b32c-
7f7dbc2135e9_shop_android_1748479822213, skey=6692c461c3810ab150c9a980d0c275ec,
sys_version=29, timestamp=1748479823, vcspKey=4d9e524ad536c03ff203787cf0dfcd29}
java.lang.Throwable
    at java.util.TreeMap.putAll(Native Method)
    at com.vip.vcsp.network.refector.VCSPApiBodyProcessor.process(SourceFile:8)

第10行：
===== Hook TreeMap.putAll =====
params is ==> {}
java.lang.Throwable
    at java.util.TreeMap.putAll(Native Method)
    at com.vip.vcsp.network.refector.VCSPApiBodyProcessor.process(SourceFile:10)


第12行：
===== Hook TreeMap.putAll =====
params is ==> {did=, dinfo=
{"ah1":"","ah2":"","ah3":"","ah4":"wifi","ah5":"1080_1794","ah6":1593600,"ah7":4
,"ah8":3948359680,"ah9":"Pixel","ah10":"","ah11":"","ah12":"","ah13":"","as1":"1
0","as2":"","as3":"","as4":"87cd3a74b10f74a1","as5":"","as6":"","as7":"29","ac1"
:"c4acd008-4f30-3599-b32c-7f7dbc2135e9"}}
java.lang.Throwable
    at java.util.TreeMap.putAll(Native Method)
    at com.vip.vcsp.network.refector.VCSPApiBodyProcessor.process(SourceFile:12)
```

```
//可以看到在第8、10行都是putAll了两次，但是我们仔细对比，它的时间戳是不一样的，我们put进
treeMap2的是timestamp=1748479823和我们发送请求的是同一个，而第一次在第8行调用的时候传进去的
是timestamp=1748479822。所以后边三次的putAll是真正往treeMap2中put的。

//因此在第8行put进的是：api_key、app_name、app_version、client_type、mars_cid、
phone_model、session_id、skey、sys_version、timestamp、vcspKey
//在第10行put的是：空
//在第12行put的是：did、dinfo

//对比发现没有put vcspToken。一会再看vcspToken，先看其他的
```

```
//那我们就去看第8行和第12行：
```

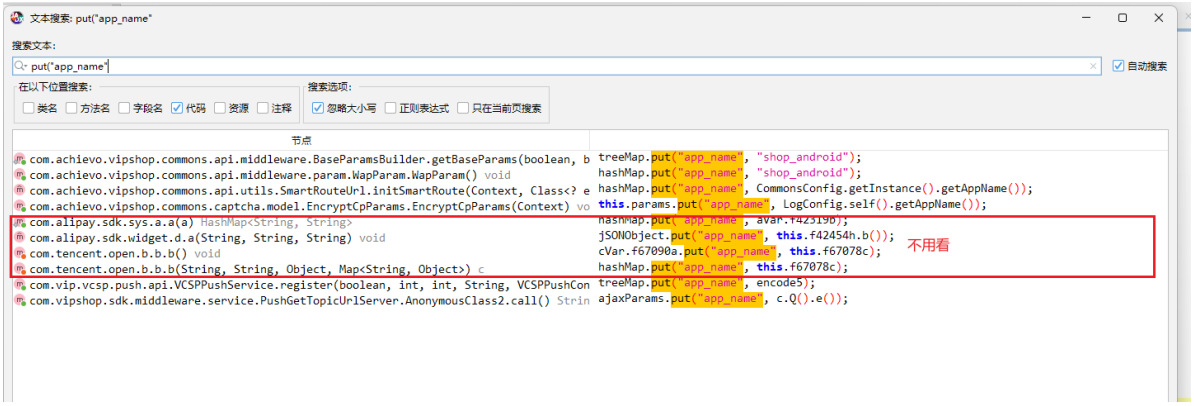
```
7               TreeMap<String, String> treeMap3 = this.networkServiceConfig.getNetworkParam().commonMap;
                if (treeMap3 != null) {
8                   treeMap2.putAll(treeMap3);
                }
```

```
11              if (this.networkServiceConfig.iEDataConfig != null && (treeMap = this.processParam.eDataMap) != null && !treeMap.isEmpty()) {
12                  treeMap2.putAll(this.processParam.eDataMap);
13                  Map<String, String> eData = this.networkServiceConfig.iEDataConfig.getEData(treeMap2);
```

```
//往treeMap2中putAll的数据，很多都是在init的时候创建的map，不容易定位，我们换个思路。直接去定位关键字
```

app_name

```
//直接去搜索关键字put("app_name"==> 搜到6条
```




```
//有两条是直接放的shop_android
//其他几条跟进去看后也都是放的固定值：shop_android
//所以app_name就是固定值
```

app_version

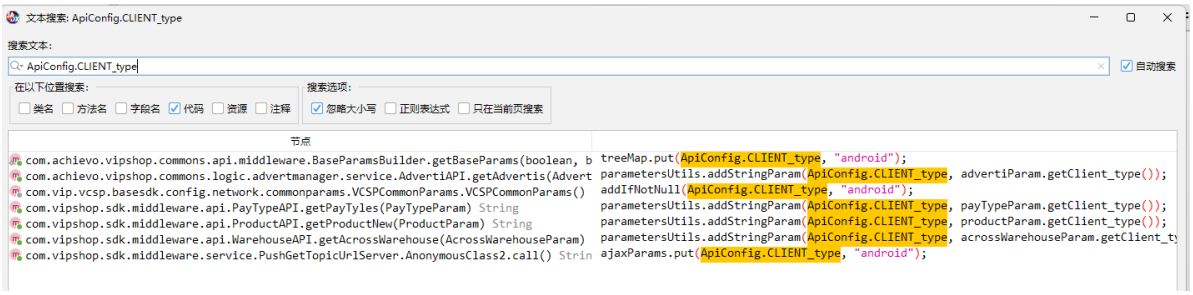
```
//这个就是app的版本把，这里抓包得到的是：app_version=7.83.3
```


```
//这里就是版本号
```

client_type

```
//client_type=android
//我们去搜索一下关键字==>找到一条结果
```

//去查找一下用例：==>找到六七条记录

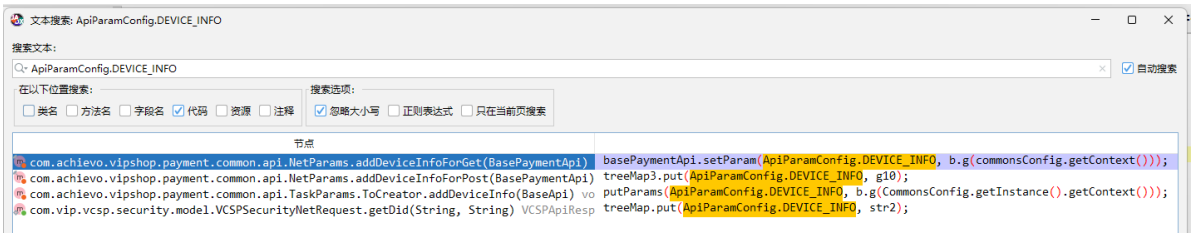


//这一看就是客户端类型把，咱们是安卓端所以就是android

dinfo

//直接去搜索关键词==>找到一条信息，是定义了一个常量

//我们直接去看它的用例



//发现有4个用例，我们直接去hook这四个方法，然后看看走的是哪个==>经过hook验证是走的getDid这个方法，并且我们同时hook了getEdata方法，发现和传进来的treeMap中的dinfo是一致的。我们去看getDid的逻辑：

```
public static VCSPApiResponseObj<VCSPReportModel> getDid(String str, String
str2) {
    TreeMap<String, String> treeMap = new TreeMap<>();
    treeMap.put("did", str);
    treeMap.put("dinfo", str2);
    return (VCSPApiResponseObj)
VCSPSecurityBasicService.getSecurityServiceConfig().request("https://mapi.appvip
shop.com/vips-mobile/rest/device/generate_token", true, true, treeMap, new
TypeToken<VCSPApiResponseObj<VCSPReportModel>>() {}.getType());
}
```

```
//是把str2的值给了dinfo这个参数，str2是调用getDid的时候传过来的，我们去上一层看看，传的是什么：
public static void getDID(final Context context, final String str, final
CallBackRequestData callBackRequestData, final String str2) {
     public String call() {
        String deviceInfo = VCSPSecurityUtils.getDeviceInfo(context, str);
        VCSPApiResponseObj<VCSPReportModel> did =
VCSPSecurityNetRequest.getDid(str2, deviceInfo);
     }
}

//是调用VCSPSecurityUtils.getDeviceInfo(context, str)得到的，传进去一个str==>我们去看看getDeviceInfo的逻辑：
public static String getDeviceInfo(Context context, String str) {
    JsonObject jsonObject = new JsonObject();
    try {
        jsonObject.addProperty("ah1", "");
        jsonObject.addProperty("ah2", "");
        jsonObject.addProperty("ah3", "");
        jsonObject.addProperty("ah4",
toResult(VCSPDeviceUtil.getNetworkType(context)));
        jsonObject.addProperty("ah5",
toResult(VCSPDeviceUtil.getScreenWidthHeight(context)));
        jsonObject.addProperty("ah6",
Integer.valueOf(VCSPDeviceUtil.getCpuFrequence()));
        jsonObject.addProperty("ah7",
Integer.valueOf(VCSPDeviceUtil.getCpuCoreNum()));
        jsonObject.addProperty("ah8",
Long.valueOf(VCSPDeviceUtil.getMemorySize(context)));
        jsonObject.addProperty("ah9", toResult(VCSPDeviceUtil.getPhoneModel()));
        jsonObject.addProperty("ah10", "");
        jsonObject.addProperty("ah11", "");
        jsonObject.addProperty("ah12", "");
        jsonObject.addProperty("ah13", "");
        jsonObject.addProperty("as1", toResult(VCSPDeviceUtil.getRomVersion()));
        jsonObject.addProperty("as2",
toResult(VCSPDeviceUtil.getCoreVersion()));
        jsonObject.addProperty("as3",
toResult(VCSPDeviceUtil.getBandVersion()));
        jsonObject.addProperty("as4",
toResult(VCSPDeviceUtil.getAndroidID(context)));
        jsonObject.addProperty("as5", "");
        jsonObject.addProperty("as6", "");
        jsonObject.addProperty("as7", toResult(VCSPDeviceUtil.getSdkVersion()));
        jsonObject.addProperty("ac1", toResult(str));
    } catch (Exception e10) {
        VCSPMyLog.error(SecurityManager.class, "getSecurityDid error", e10);
    }
    return jsonObject.toString();
}
//这里就是new了一个Json，然后放进去后toString的，我们去hook一下这个getDeviceInfo方法

VCSPSecurityUtils.getDeviceInfo is called:
context=com.achievo.vipshop.common.VipApplicationLike@762215d, str=c4acd008-
4f30-3599-b32c-7f7dbc2135e9
```

```
VCSPSecurityUtils.getDeviceInfo result=
{"ah1":"","ah2":"","ah3":"","ah4":"wifi","ah5":"1080_1794","ah6":1593600,"ah7":4
,"ah8":3948359680,"ah9":"Pixel","ah10":"","ah11":"","ah12":"","ah13":"","as1":"1
0","as2":"","as3":"","as4":"87cd3a74b10f74a1","as5":"","as6":"","as7":"29","ac1"
:"c4acd008-4f30-3599-b32c-7f7dbc2135e9"}
java.lang.Throwable
        at com.vip.vcsp.security.utils.VCSPSecurityUtils.getDeviceInfo(Native
Method)
        at
com.vip.vcsp.security.api.VCSPSecurityBasicService$4.call(SourceFile:2)
        at
com.vip.vcsp.security.api.VCSPSecurityBasicService$4.call(SourceFile:1)
        at c.g$i.run(SourceFile:3)
        at
java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1167)
        at
java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:641)
        at java.lang.Thread.run(Thread.java:919)


//传进来的str是4acd008-4f30-3599-b32c-7f7dbc2135e9 和 mars_cid、session_id的值是一样
的，我们去分析一下这个str是怎么来的：
//根据调用栈去看上一层：com.vip.vcsp.security.api.VCSPSecurityBasicService$4.call
//在call中，str是调用getDID时传进来的，我们去hook一下getDID，因为它有两个用例
VCSPSecurityBasicService.getDID is called:
context=com.achievo.vipshop.common.VipApplicationLike@762215d, str=c4acd008-
4f30-3599-b32c-7f7dbc2135e9, callBackRequestData=[object Object], str2=
java.lang.Throwable
        at com.vip.vcsp.security.api.VCSPSecurityBasicService.getDID(Native
Method)
        at b8.b.c(SourceFile:2)
        at h6.a.b(SourceFile:4)
        at h6.a.a(SourceFile:1)
        at h6.a$a$a.onResult(SourceFile:3)
//我们去看b8.b.c(SourceFile:2)
public static void c(Context context, String str, String str2, d dVar) {
    BaseSDK.getSecurityBasicService();
    VCSPSecurityBasicService.getDID(context, str, new C0024b(dVar), str2);
}
//可以看到str也是从上一层传过来的，继续看:h6.a.b(SourceFile:4)
public static void b(Context context, String str) {
    p5.a.c().d(context);
    String a10 = p5.a.c().a();
    ApiConfig.getInstance().setDid(a10);
    b8.b.c(context, str, a10, new b(a10));
}
//str还说从上一层传过来的，继续跟：h6.a.a(SourceFile:1)
//跟到了h6.a$a$a.onResult
public void onResult(boolean z10) {
    if (z10) {
        b8.b.k(true);
    }
    String mid = ApiConfig.getInstance().getMid();
    a.b(CallableC0846a.this.f72577b, mid);
    //调用了b方法，传去了mid，mid就是我们一直跟的结果
```

```
    if (CommonsConfig.getInstance().isAgreePrivacy()) {
        a.c(CallableC0846a.this.f72577b, mid);
    }
}
//里边调用了a.b(CallableC0846a.this.f72577b, mid);
//可以看到这个mid是通过String mid = ApiConfig.getInstance().getMid();得到的，去看看
public String getMid() {
    if (TextUtils.isEmpty(this.mid)) {
        String stringByKey =
CommonPreferencesUtils.getStringByKey(CommonsConfig.getInstance().getContext(),
CommonsConfig.VIP_MID_KEY);
        this.mid = stringByKey;
        if (SDKUtils.isNull(stringByKey) ||
DeviceUuidFactory.ANDROIDID_000000000_MID.equals(this.mid)) {
            String uuid =
DeviceUuidFactory.getDeviceUuid(CommonsConfig.getInstance().getContext()).toStri
ng();
            this.mid = uuid;
            if (SDKUtils.isNull(uuid)) {
                this.mid = UUID.randomUUID().toString();
            }

 CommonPreferencesUtils.addConfigInfo(CommonsConfig.getInstance().getContext(),
CommonsConfig.VIP_MID_KEY, this.mid);
        }
    }
    return this.mid;
}
//首先会去检查本地缓存中是否已经有了，若有了直接拿，没有就去调用
DeviceUuidFactory.getDeviceUuid(CommonsConfig.getInstance().getContext()).toStri
ng();生成一个：
public static UUID getDeviceUuid(Context context) {
    UUID randomUUID;
    String a10 = com.achievo.vipshop.commons.a.a(context);
    mAndroidID = a10;
    try {
        if (!"9774d56d682e549c".equals(a10) && !"0000000000000000".equals(a10)
&& !TextUtils.isEmpty(a10)) {
            randomUUID = UUID.nameUUIDFromBytes(a10.getBytes("utf8"));
            CommonPreferencesUtils.addConfigInfo(context,
CommonsConfig.MID_TYPE_KEY, "1");
        } else {
            randomUUID = UUID.randomUUID();
            CommonPreferencesUtils.addConfigInfo(context,
CommonsConfig.MID_TYPE_KEY, "3");
        }
    } catch (Exception e10) {
        randomUUID = UUID.randomUUID();
        CommonPreferencesUtils.addConfigInfo(context,
CommonsConfig.MID_TYPE_KEY, "3");
    }
    if (SDKUtils.isNull(randomUUID)) {
        randomUUID = UUID.randomUUID();
        CommonPreferencesUtils.addConfigInfo(context,
CommonsConfig.MID_TYPE_KEY, "3");
```

```
    }
    return randomUUID;
}
//进来后先判断a10，是否为9774d56d682e549c或者0000000000000000，若为这二者之一，则直接去生成一个UUID，若不为则把a10传进去做一个固定的UUid



//总的来说，在生成dinfo时是调用getDeviceInfo(Context context, String str)生成的，其中str是一个UUID，这个UUID首先会通过获取手机的一个值，然后拿一个固定的uuid，若没拿到手机这个值，则直接取一个uuid。接下来去看getDeviceInfo的逻辑：
```

```
public class VCSPSecurityUtils {
    public static String getDeviceInfo(Context context, String str) {
        JsonObject jsonObject = new JsonObject();
        try {
            jsonObject.addProperty("ah1", "");
            jsonObject.addProperty("ah2", "");
            jsonObject.addProperty("ah3", "");
            jsonObject.addProperty("ah4", toResult(VCSPDeviceUtil.getNetworkType(context)));
            jsonObject.addProperty("ah5", toResult(VCSPDeviceUtil.getScreenWidthHeight(context)));
            jsonObject.addProperty("ah6", Integer.valueOf(VCSPDeviceUtil.getCpuFrequence()));
            jsonObject.addProperty("ah7", Integer.valueOf(VCSPDeviceUtil.getCpuCoreNum()));
            jsonObject.addProperty("ah8", Long.valueOf(VCSPDeviceUtil.getMemorySize(context)));
            jsonObject.addProperty("ah9", toResult(VCSPDeviceUtil.getPhoneModel()));
            jsonObject.addProperty("ah10", "");
            jsonObject.addProperty("ah11", "");
            jsonObject.addProperty("ah12", "");
            jsonObject.addProperty("ah13", "");
            jsonObject.addProperty("as1", toResult(VCSPDeviceUtil.getRomVersion()));
            jsonObject.addProperty("as2", toResult(VCSPDeviceUtil.getCoreVersion()));
            jsonObject.addProperty("as3", toResult(VCSPDeviceUtil.getBandVersion()));
            jsonObject.addProperty("as4", toResult(VCSPDeviceUtil.getAndroidID(context)));
            jsonObject.addProperty("as5", "");
            jsonObject.addProperty("as6", "");
            jsonObject.addProperty("as7", toResult(VCSPDeviceUtil.getSdkVersion()));
            jsonObject.addProperty("ac1", toResult(str));
        } catch (Exception e10) {
            VCSPMyLog.error(SecurityManager.class, "getSecurityDid error", e10);
        }
        return jsonObject.toString();
    }

    private static String toResult(String str) {
        return str != null ? str : "";
    }
}
```

```
//可以看到，ah1、ah2、ah3、ah10、ah11、ah12、ah13、as5、as6是直接给的空。其他的值要先调用toResult判断一下是否为null，不为null了再放进去，为null直接放空值

//接下来我们去看它不为空的几个值ah4、ah5、ah6、ah7、ah8、ah9、as1、as2、as3、as4、as7、ac1


ah4:
//ah4的值是通过VCSPDeviceUtil.getNetworkType(context)得到的==>这个是判断网络类型，type==1就是wifi
public static String getNetworkType(Context context) {
    try {
        ConnectivityManager connectivityManager = (ConnectivityManager) context.getSystemService("connectivity");
        NetworkInfo activeNetworkInfo = connectivityManager != null ? connectivityManager.getActiveNetworkInfo() : null;
        if (activeNetworkInfo != null && activeNetworkInfo.isConnected()) {
            int type = activeNetworkInfo.getType();
            if (type == 1) {
                return "wifi";
            }
            if (type == 0) {
                switch (activeNetworkInfo.getSubtype()) {
                    case 1:
```

```
                        return "gprs";
                    case 2:
                        return "edge";
                    case 3:
                        return "umts";
                    case 4:
                        return "cdma";
                    case 5:
                        return "evdo_o";
                    case 6:
                        return "evdo_a";
                    case 7:
                        return "1xrtt";
                    case 8:
                        return "hsdpa";
                    case 9:
                        return "hsupa";
                    case 10:
                        return "hspa";
                    case 11:
                        return "iden";
                    case 12:
                        return "evdo_b";
                    case 13:
                        return "lte";
                    case 14:
                        return "ehrpd";
                    case 15:
                        return "hspap";
                    case 16:
                        return "gsm";
                    case 17:
                        return "scdma";
                    case 18:
                        return "iwlan";
                    default:
                        return "other";
                }
            }
        }
    } catch (Exception e10) {
        VCSPMyLog.error(VCSPDeviceUtil.class, "get getNetworkType error", e10);
    }
    return "other";
}
//我们抓包得到的值是："ah4":"wifi"  没问题
```

ah5：

```
//ah5是通过VCSPDeviceUtil.getScreenWidthHeight(context)得到的==>目测是屏幕的宽和高
public static String getScreenWidthHeight(Context context) {
    WindowManager windowManager = (WindowManager)
context.getSystemService("window");
```

```
    return "" + windowManager.getDefaultDisplay().getWidth() + "_" +
windowManager.getDefaultDisplay().getHeight();
}
//是 宽_高，我们抓到的是："ah5":"1080_1794"
```

ah6：

```
//ah6是通过Integer.valueOf(VCSPDeviceUtil.getCpuFrequence()))得到的：
public static int getCpuFrequence() {
    try {
        return VCSPNumberUtils.stringToInteger(new BufferedReader(new
InputStreamReader(new ProcessBuilder("/system/bin/cat",
"/sys/devices/system/cpu/cpu0/cpufreq/cpuinfo_max_freq").start().getInputStream(
))).readLine());
    } catch (IOException e10) {
        VCSPMyLog.error(VCSPDeviceUtil.class, "getCpuFrequence error", e10);
        return 0;
    }
}
//stringToInteger：
public static int stringToInteger(String str) {
    try {
        if (TextUtils.isEmpty(str)) {
            return 0;
        }
        return Integer.parseInt(str.trim());
    } catch (Exception e10) {
        VCSPMyLog.error(VCSPNumberUtils.class, "stringToInteger error", e10);
        return 0;
    }
}
//是先获取设备CPU的最大频率，单位为赫兹。这里抓到的是"ah6":1593600，此设备赫兹在1593600hz左
右也就是约1.6GHz。正常范围为：1-3.5GHz
```

ah7：

```
//ah7是通过：Integer.valueOf(VCSPDeviceUtil.getCpuCoreNum()));
//这是获取cpu的数量把
public static int getCpuCoreNum() {
    try {
        return new File("/sys/devices/system/cpu/").listFiles(new FileFilter() {
            @Override // java.io.FileFilter
            public boolean accept(File file) {
                return Pattern.matches("cpu[0-9]", file.getName());
            }
        }).length;
    } catch (Exception e10) {
        VCSPMyLog.error(VCSPDeviceUtil.class, "getNumCores error", e10);
        return 1;
    }
}
//是的 cpu的数量从 0-9，我的是4核的，所以抓到的是："ah7":4
```

ah8：

```
//ah8是通过：Long.valueOf(VCSPDeviceUtil.getMemorySize(context)))得到的，这是获取最大内存?
public static long getMemorySize(Context context) {
    return getTotalMemory(context);
}
//进来后获取总的内存：
//我的设别是4G内存，抓到的是"ah8":3948359680，在命令窗口执行命令：adb shell cat /proc/meminfo可以看到自己手机的内存信息，我这里返回的是：
C:\Users\yangsiteng>adb shell cat /proc/meminfo
MemTotal:        3855820 kB
MemFree:          151304 kB
MemAvailable:    1875188 kB
//总空间3855820 kB，然后乘以1024转为B，就是3948359680，也就是抓包得到的这个值。所以这个ah8是内存的总数，单位为B
```

ah9：

```
toResult(VCSPDeviceUtil.getPhoneModel()))
//见名知意，这是获取手机型号把，直接去看我们抓包得到的信息："ah9":"Pixel"  ==>毋庸置疑
```

as1：

```
jsonObject.addProperty("as1", toResult(VCSPDeviceUtil.getRomVersion()));
//这是获得rom的版本？内核版本呗？但是这里返回的10，我是安卓10的系统，去看下getRomVersion逻辑把：
public static String getRomVersion() {
    return Build.VERSION.RELEASE;
}
//确实就是获取的系统版本号
```

as2：

```
jsonObject.addProperty("as2", toResult(VCSPDeviceUtil.getCoreVersion()));
//我这里抓到的是空的，jadx和GDA反编译后看不到东西
```

as3：

```
jsonObject.addProperty("as3", toResult(VCSPDeviceUtil.getBandVersion()));

public static String getBandVersion() {
    try {
        Class<?> cls = Class.forName("android.os.SystemProperties");
        return (String) cls.getMethod(NetParams.get, String.class,
String.class).invoke(cls.newInstance(), "gsm.version.baseband", "no message");
    } catch (Exception unused) {
        return "";
    }
```

```
}

//这里是获取基带版本，获取不到就返回空，这里抓包抓到的为空。高通一般为MOLY.LR11.AC.05、三星一般为G9900XXU3CUH1



as4:
jsonObject.addProperty("as4", toResult(VCSPDeviceUtil.getAndroidID(context)));

public static String getAndroidID(Context context) {
    try {
        return AIDGenHelper.getAndroidID(context);
    } catch (Exception e10) {
        VCSPMyLog.error(VCSPDeviceUtil.class, "getAndroidID error", e10);
        return null;
    }
}

public static String getAndroidID(Context context) {
    if (TextUtils.isEmpty(ANDROID_ID) && CAN_GET_ANDROID_ID) {
        try {
            ANDROID_ID = Settings.Secure.getString(context.getContentResolver(),
"android_id");
            VCSPMyLog.info(AIDGenHelper.class, "ANDROID_ID = " + ANDROID_ID);
        } catch (Exception e10) {
            VCSPMyLog.error(AIDGenHelper.class, e10);
        }
    }
    return ANDROID_ID;
}

//这里是获取的android_id
//Android ID 是一个 64 位十六进制字符串（16 个字符），例如 9774d56d682e549c



as7:
jsonObject.addProperty("as7", toResult(VCSPDeviceUtil.getSdkVersion()));
//这里就是sdk的版本了，安卓10对应29，这里抓到的数据也是29==>"as7":"29"
public static String getSdkVersion() {
    return "" + Build.VERSION.SDK_INT;
}


ac1:
jsonObject.addProperty("ac1", toResult(str));
//这里就是把传过来的这个根据手机信息得到的固定uuid放进去了




//以上就是对dinfo的分析
```

mars_cid

```
//在dinfo中分析了，他和传进dinfo中参数的是相同的，都是通过手机信息去获取一个固定的UUid
```

session_id

```
//我们去看看我们抓到的
mars_cid  =c4acd008-4f30-3599-b32c-7f7dbc2135e9,
session_id=c4acd008-4f30-3599-b32c-
7f7dbc2135e9_shop_android_1748582534479,timestamp=1748582535,

//可以看到它和mars_cid多了写多西，使用_拼接上了app_name和timestamp的值，但是timestamp是毫秒级时间戳
```

phone_model

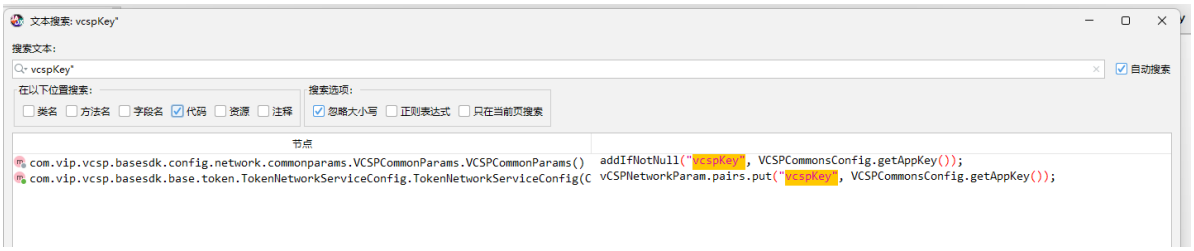
```
//这里就是手机型号了==》phone_model=Pixel,
```

sys_version

```
//这里是系统版本对应的sdk版本==》sys_version=29,
```

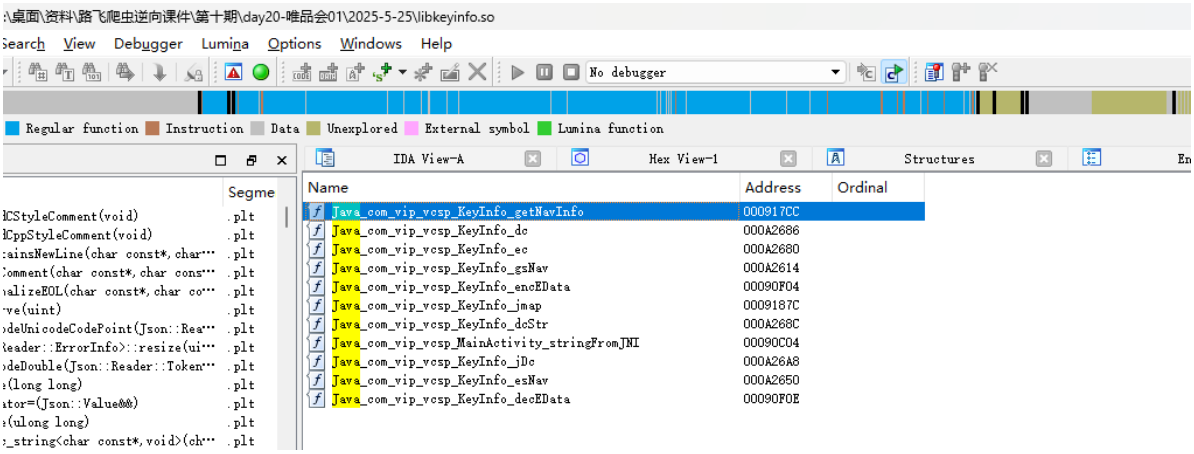
vcspKey

```
//我们去搜索一下关键字：
```




```
//找到两条，可以发现都是走的同一个逻辑去拿到的值：VCSPCommonsConfig.getAppKey()。我们直接去看getAppKey()：
public static String getAppKey() {
    return appKey;
}

private static String appKey = null;

//返回的是appKey，我们去看看在哪里给appKey赋值的：
public static void setAppKey(String str) {
    appKey = str;
}
//只有这一个地方赋值，再去看谁调用了setAppKey(String str)==>也只有一个地方调用
private static void initKeyInfo(String str) {
    ...

 VCSPCommonsConfig.setAppKey(KeyInfoFetcher.getInfo(VCSPCommonsConfig.getContext(), "vcsp_key"));
```

```
    ...
}

//其值通过KeyInfoFetcher.getInfo(VCSPCommonsConfig.getContext(), "vcsp_key")得到的：
public static String getInfo(Context context, String str) {
    try {
        if (clazz == null || object == null || method == null) {
            int i10 = KeyInfo.f69594a;
            clazz = KeyInfo.class;
            object = KeyInfo.class.newInstance();
            method = clazz.getMethod("getInfo", Context.class, String.class);
        }
        return (String) method.invoke(object, context, str);
    } catch (Exception e10) {
        VCSPMyLog.error(KeyInfoFetcher.class, e10);
        return "";
    }
}
//这里是反射调用了KeyInfo下的getInfo方法，把参数传进去了，我们去看KeyInfo下的getInfo方法：
public static String getInfo(Context context, String str) {
    try {
        try {
            return getNavInfo(context, str);
        } catch (Throwable th2) {
            return "KI gi: " + th2.getMessage();
        }
    } catch (Throwable unused) {
        SoLoader.load(context, LibName);
        return getNavInfo(context, str);
    }
}
//返回了getNavInfo(context, str);

private static native String getNavInfo(Context context, String str);
//getNavInfo(context, str)是一个native方法，我们对它进行一下hook，看看是不是走的这里==>发现打印了很多，但是我们知道，在获取vcspKey时传进去的 参数是:"vcsp_key",所以我们直接去搜"vcsp_key"
KeyInfo.getNavInfo is called:
context=com.achievo.vipshop.common.VipApplicationLike@762215d, str=vcsp_key
KeyInfo.getNavInfo result=4d9e524ad536c03ff203787cf0dfcd29
java.lang.Throwable
        at com.vip.vcsp.KeyInfo.getNavInfo(Native Method)
        at com.vip.vcsp.KeyInfo.getInfo(SourceFile:1)
        at java.lang.reflect.Method.invoke(Native Method)
        at
com.vip.vcsp.basesdk.base.keyinfo.KeyInfoFetcher.getInfo(SourceFile:5)
        at com.vip.vcsp.basesdk.base.BaseSDK.initKeyInfo(SourceFile:3)
        at com.vip.vcsp.basesdk.base.BaseSDK.startInner(SourceFile:5)
        at com.vip.vcsp.basesdk.base.BaseSDK.start(SourceFile:1)
//确实是走的这里，我们去so层看看：这个so很熟悉了，分析两次了，咱们直接去看导出函数：
```



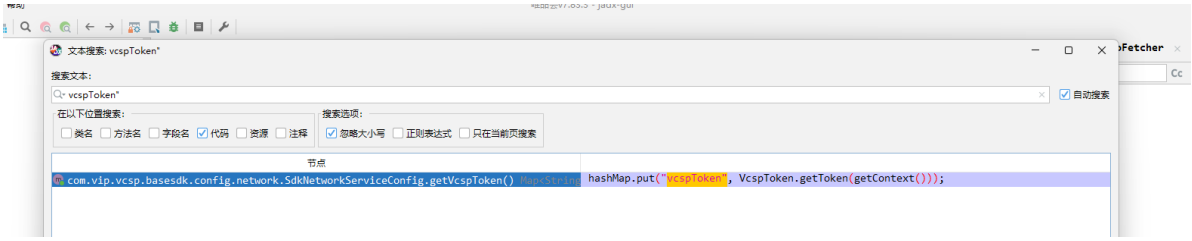

```
//进来后，修改JNIEnv 和各个参数
jstring __fastcall Java_com_vip_vcsp_KeyInfo_getNavInfo(JNIEnv *env, jclass
clas, jobject context, jobject str)
{
  const char *v7; // r6
  int v8; // r3
  jstring (*v9)(JNIEnv *, const char *); // r2
  void *v10; // r5
  int v11; // r4
  int v12; // [sp+0h] [bp-18h] BYREF
  int v13[5]; // [sp+4h] [bp-14h] BYREF

  if ( !str || !(*env)->GetStringUTFLength(env, str) || isInBlackList(env, str)
)
    return (*env)->NewStringUTF(env, &byte_1DCB68);
  v7 = (*env)->GetStringUTFChars(env, str, 0);
  v12 = 0;
  v13[0] = 0;
  j_getInfo((int)v7, (int)v13, (int)&v12, v8);
  (*env)->ReleaseStringUTFChars(env, str, v7);
  v9 = (*env)->NewStringUTF;
  if ( v12 < 1 )
    return v9(env, &byte_1DCB68);
  v10 = (void *)v13[0];
  if ( !v13[0] )
    return v9(env, &byte_1DCB68);
  v11 = (int)v9(env, (const char *)v13[0]);
  operator delete(v10);
  v13[0] = 0;
  return (jstring)v11;
}

//可以看到进来后先对str进行了判断，若为空或者是长度为0或者是在黑名单，都直接return
&byte_1DCB68，  &byte_1DCB68是0，这里不为空，所以继续向下执行
//然后通过GetStringUTFChars把传进来的jstr转为了cstr
//然后给v12赋0，v13[0]赋0，然后调用和  j_getInfo((int)v7, (int)v13, (int)&v12, v8);方
法
//执行完  j_getInfo((int)v7, (int)v13, (int)&v12, v8);之后会去判断v12小于1或者是
v13[0]w为0的都会返回0.否则就会把v13[0]转为jstr返回回去。
//然后我去hook了j_getInfo((int)v7, (int)v13, (int)&v12, v8);方法，发现在进入这个函数的
时候，v13[0]为空，执行完这个函数后，v13[0]就是我们的返回值了，所以核心是在j_getInfo：函数中
//我们去看下这个j_getInfo：
```

```
int __fastcall getInfo(int a1, _DWORD *a2, int *a3, int a4)
{
  int *v7; // r0
  int v8; // r6
  int v9; // r4
  _BYTE *v10; // r0
  _BYTE *v11; // r1
  _BYTE *v12; // r5
  _DWORD v14[3]; // [sp+0h] [bp-38h] BYREF
  unsigned __int8 v15; // [sp+Ch] [bp-2Ch] BYREF
  _BYTE v16[7]; // [sp+Dh] [bp-2Bh] BYREF
  _BYTE *v17; // [sp+14h] [bp-24h]

  v14[0] = a2;
  v14[1] = a3;
  v14[2] = a4;
  if ( !dword_2347C8 )
  {
    std::mutex::lock((std::mutex *)&mutex);
    initMap(&infoMap);
    std::mutex::unlock((std::mutex *)&mutex);
  }
  sub_91120(&v15);
  std::string::basic_string<decltype(nullptr)>((int)v14, (char *)a1);
  v7 = (int
*)std::__tree<std::__value_type<std::string,std::string>,std::__map_value_compar
e<std::string,std::__value_type<std::string,std::string>,std::less<std::string>,
true>,std::allocator<std::__value_type<std::string,std::string>>>::find<std::str
ing>(
                &infoMap,
                v14);
  if ( v7 == &dword_2347C4 )
  {
    v8 = 0;
  }
  else
  {
    std::string::operator=(&v15, v7 + 7);
    v8 = *(_DWORD *)&v16[3];
    v9 = v15 & 1;
    if ( !v9 )
      v8 = v15 >> 1;
    if ( v8 > 0 )
    {
      v10 = (_BYTE *)operator new[](v8 + 1);
      *a2 = v10;
      v11 = v17;
      if ( !v9 )
        v11 = v16;
      v12 = v10;
      qmemcpy(v10, v11, v8);
      v12[v8] = 0;
      *a3 = v8;
    }
  }
```

```
  std::string::~string(v14);
  std::string::~string(&v15);
  return v8;
}

//它的主要逻辑就是去infoMap这个映射表中查找值为a1的键所对应的值，然后放到a2中，其中a1就是我们从java层传进来的值，a2就是最终返回给java层的值
//所以这里就是固定的这个值：4d9e524ad536c03ff203787cf0dfcd29
```

vcspToken-getTokenByFP接口

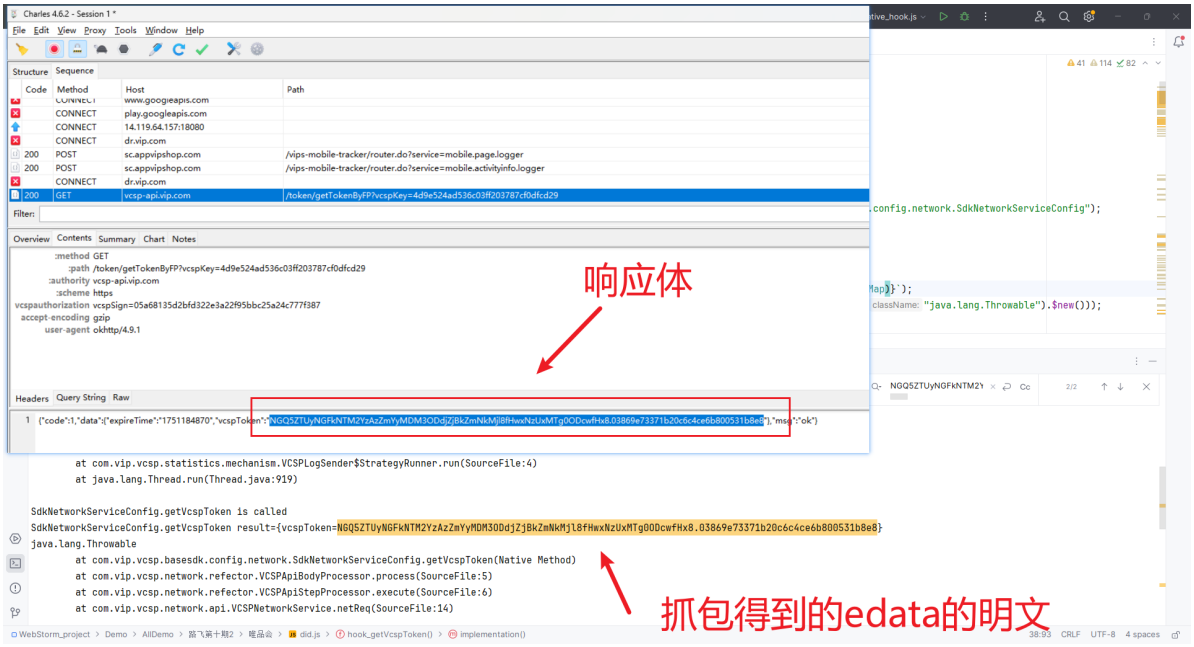
```
//我们直接去搜索关键字==>搜到一条结果
```




```
//我们进去看看==>hook一下，看看是不是走了这里
SdkNetworkServiceConfig.getVcspToken is called
SdkNetworkServiceConfig.getVcspToken result=
{vcspToken=NGQ5ZTUyNGFkNTM2YzAzZmYyMDM3ODdjZjBkZmNkMjl8fHwxNzUxMTg0MDg1fHx8.023678393b10b22b62bc926e0e2968d1}
java.lang.Throwable
        at
com.vip.vcsp.basesdk.config.network.SdkNetworkServiceConfig.getVcspToken(Native Method)
        at
com.vip.vcsp.network.refector.VCSPApiBodyProcessor.process(SourceFile:5)
        at
com.vip.vcsp.network.refector.VCSPApiStepProcessor.execute(SourceFile:6)
        at com.vip.vcsp.network.api.VCSPNetworkService.netReq(SourceFile:14)
        at com.vip.vcsp.network.api.VCSPNetworkService.doPost(SourceFile:8)
        at
com.vip.vcsp.basesdk.config.security.SecurityServiceConfigProxy.request(SourceFile:8)
        at
com.vip.vcsp.security.model.VCSPSecurityNetRequest.getDid(SourceFile:6)
        at
com.vip.vcsp.security.api.VCSPSecurityBasicService$4.call(SourceFile:4)
        at
com.vip.vcsp.security.api.VCSPSecurityBasicService$4.call(SourceFile:1)
        at c.g$i.run(SourceFile:3)
        at
java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1167)
        at
java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:641)

//确实走了这里，并且和edata的明文是一致的，我们去看看getVcspToken的逻辑：
public Map<String, String> getVcspToken() {
    HashMap hashMap = new HashMap(1);
    hashMap.put("vcspToken", VcspToken.getToken(getContext()));
```

```
    VCSPMyLog.debug(SdkNetworkServiceConfig.class, "vcspToken:" +
VcspToken.getToken(getContext()));
    return hashMap;
}
//进来后new 了一个hashMap 大小为1，就放了一个vcspToken，是通过
VcspToken.getToken(getContext()):得到的
VcspToken.getToken(getContext()):
public static String getToken(Context context) {
    if (TextUtils.isEmpty(sToken)) {
        try {
            sToken = (String) storageService.getValueByKey(context,
"vcsp_token", String.class);
        } catch (Exception e10) {
            VCSPMyLog.error(VcspToken.class, e10.toString());
        }
    }
    String str = sToken;
    return str == null ? "" : str;
}
//首先会判断sToken是否为空，不为空就直接返回，若为空就去通过getValueByKey得到一个，我们去看看
getValueByKey：
public <T> T getValueByKey(Context context, String str, Class<T> cls) {
    return (T) this.storage.getValueByKey(context, str, cls);
}

public <T> T getValueByKey(Context context, String str, Class<T> cls) {
    ConcurrentHashMap concurrentHashMap = this.mKVMap;
    if (concurrentHashMap != null && concurrentHashMap.containsKey(str)) {
        return (T) this.mKVMap.get(str);
    }
    return (T) getValueByKey(context, str, cls, "");
}


private static <T> T getValueByKey(Context context, String str, Class<T> cls,
String str2) {
    if (!TextUtils.isEmpty(str) && context != null) {
        VCSPVipPreference vCSPVipPreference = new VCSPVipPreference(context,
context.getPackageName() + str2);
        if (cls.equals(String.class)) {
            return (T) vCSPVipPreference.getPrefString(str, "");
        }
        if (cls.equals(Long.class)) {
            return (T) Long.valueOf(vCSPVipPreference.getPrefLong(str, 0L));
        }
        if (cls.equals(Boolean.class)) {
            return (T) Boolean.valueOf(vCSPVipPreference.getPrefBoolean(str,
false));
        }
        if (cls.equals(Integer.class)) {
            return (T) Integer.valueOf(vCSPVipPreference.getPrefInt(str, 0));
        }
    }
    return null;
}
```

//跟了一路发现最终是去缓存中取，那肯定是有地方把这个值放进缓存中去的，那很用可能就是动态拿到的，我们去响应体中搜一下：

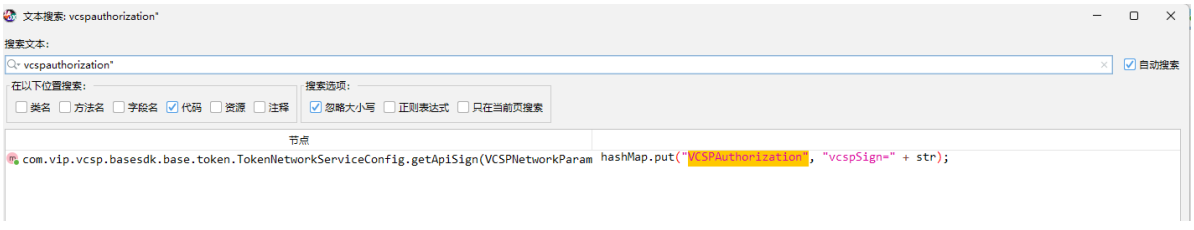


//果然是发送请求拿到的，我们再去看看是怎么拿的

######

请求网址：https://vcsp-api.vip.com/token/getTokenByFP
请求方式：GET
请求头：
    vcspauthorization vcspSign=05a68135d2bfd322e3a22f95bbc25a24c777f387
    accept-encoding gzip
    user-agent okhttp/4.9.1
请求参数：
    vcspKey 4d9e524ad536c03ff203787cf0dfcd29

//我们先去看请求头中的vcspauthorization
//还是去搜索关键字==>很友好，并且只有一条，格式也是我们想要的格式：

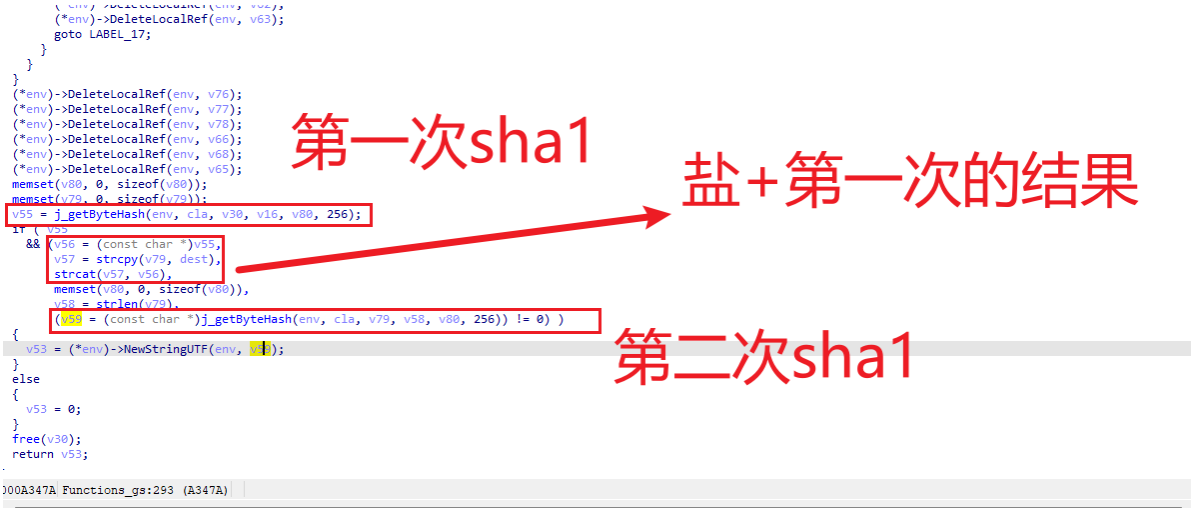


vcspauthorization

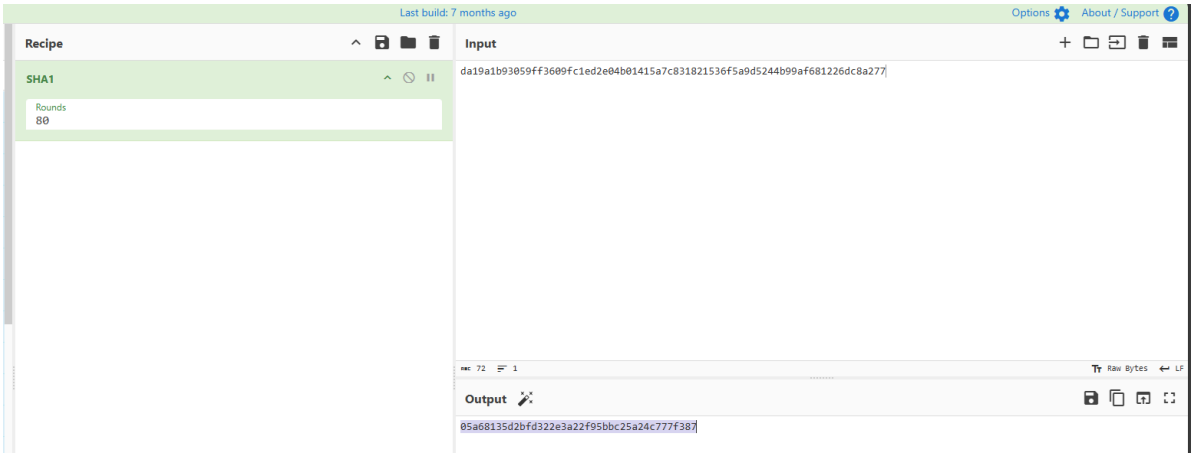
```
//public Map<String, String> getApiSign(VCSPNetworkParam vCSPNetworkParam) {
    String str;
    try {
        BaseSDK.getSecurityBasicService();
```

```
        str =
VCSPSecurityBasicService.apiSignVcspToken(VCSPCommonsUtils.getUrlParams(getConte
xt(), vCSPNetworkParam.url),
VCSPCommonsConfig.getIAppInfo().getUserTokenSecret());
    } catch (Exception e10) {
        VCSPMyLog.error(TokenNetworkServiceConfig.class, e10);
        str = "";
    }
    HashMap hashMap = new HashMap(1);
    hashMap.put("VCSPAuthorization", "vcspSign=" + str);
    return hashMap;
}

//可以看到是把str拼接到 "vcspSign=" 后边的，str是通过调用其他函数得到的：
str =
VCSPSecurityBasicService.apiSignVcspToken(VCSPCommonsUtils.getUrlParams(getConte
xt(), vCSPNetworkParam.url),
VCSPCommonsConfig.getIAppInfo().getUserTokenSecret());
//我们去hook一下这个apiSignVcspToken方法：
VCSPSecurityBasicService.apiSignVcspToken is called: treeMap=
{vcspKey=4d9e524ad536c03ff203787cf0dfcd29}, str=null
VCSPSecurityBasicService.apiSignVcspToken
result=05a68135d2bfd322e3a22f95bbc25a24c777f387
//传进去的参数1是一个treeMap，参数2是一个null，并且可以卡到这个treeMap就是我们getTokenByFP
的请求参数，我们去看看apiSignVcspToken的逻辑：
public static String apiSignVcspToken(TreeMap<String, String> treeMap, String
str) throws Exception {
    return VCSPSecurityConfig.getMapParamsSign(VCSPCommonsConfig.getContext(),
treeMap, str, true);
}

public static String getMapParamsSign(Context context, TreeMap<String, String>
treeMap, String str, boolean z10) {
    String str2 = null;
    if (treeMap != null) {
        boolean z11 = false;
        Set<Map.Entry<String, String>> entrySet = treeMap.entrySet();
        if (entrySet != null) {
            Iterator<Map.Entry<String, String>> it = entrySet.iterator();
            while (true) {
                if (it == null || !it.hasNext()) {
                    break;
                }
                Map.Entry<String, String> next = it.next();
                if (next != null && next.getKey() != null &&
ApiConfig.USER_TOKEN.equals(next.getKey()) &&
!TextUtils.isEmpty(next.getValue())) {
                    z11 = true;
                    break;
                }
            }
        }
        if (z11) {
            if (TextUtils.isEmpty(str)) {
                str = VCSPCommonsConfig.getTokenSecret();
```

```
            }
            str2 = str;
        }
        return getSignHash(context, treeMap, str2, z10);
    }
    return null;
}


//然后调用了getSignHash(context, treeMap, str2, z10);
....

//最后调用了private static native String gsNav(Context context, Map<String,
String> map, String str, boolean z10);
//是一个native方法，我们对gsNav进行了hook，拿到了入参和返回值：
KeyInfo.gsNav is called:
context=com.achievo.vipshop.common.VipApplicationLike@5960662, map=
{vcspKey=4d9e524ad536c03ff203787cf0dfcd29}, str=null, z10=true

KeyInfo.gsNav result=05a68135d2bfd322e3a22f95bbc25a24c777f387
//我们去看看是如何得到的     vcspauthorization
//在看了下好像和authorization走的是同一个方法，都是gsNav，都是sha1哈希算法：  第一次：盐+map
转字符串==>sha1、第二次：盐+第一次的结果==>sha1
//但是我们之前分析的时候说过，如果z10为true那么它的盐就是另一个，z10是来决定盐是哪个的，所以我
们需要去hook一下里边的j_get_strData(z10);
//经hook确定  当z10为true时，盐为：da19a1b93059ff3609fc1ed2e04b0141
```




```
//我们直接拿刚刚抓到的入参和返回值去验证
//盐：da19a1b93059ff3609fc1ed2e04b0141

//刚刚hook gsNav时的参数:map={vcspKey=4d9e524ad536c03ff203787cf0dfcd29} 返回值：
05a68135d2bfd322e3a22f95bbc25a24c777f387
//进gsNav后会拼成：vcspKey=4d9e524ad536c03ff203787cf0dfcd29
//然后把盐拼到前边：
da19a1b93059ff3609fc1ed2e04b0141vcspKey=4d9e524ad536c03ff203787cf0dfcd29
//进行第一次sha1后hex编码后的结果：5a7c831821536f5a9d5244b99af681226dc8a277
//再用盐拼上第一次的结果：
da19a1b93059ff3609fc1ed2e04b01415a7c831821536f5a9d5244b99af681226dc8a277
//第二次sha1后hex编码后的结果：05a68135d2bfd322e3a22f95bbc25a24c777f387
```



到这里这个vcspauthorization就解决了，是把请求参数的进行两次sha1+盐得到的

盐：da19a1b93059ff3609fc1ed2e04b0141

每次把盐加在最前边

第一次的明文为传进去的请求参数，key=value这样拼一下，再加上盐
第二次的明文为：第一次sha1的结果

两次sha1就得到的了我们的　　vcspauthorization

```
//接下来去看请求参数：vcspKey 4d9e524ad536c03ff203787cf0dfcd29


//这个很熟悉了，刚刚分析过，是在so层，根据传进去的key来映射的，就是固定的
```

vcspToken总结：
//到此vcspToken就解决了
　　向服务端发送请求拿来的

edata总结

edata为generate_token接口请求体中的一个参数，在getEData方法中通过传入一个treeMap然后对treeMap中的值进行key=value&key=value..的拼接，但是在拼接前会先对value进行一个url编码

拼接完后最终会走到调用native方法esNav

在esNav中会对拼接的treeMap进行一个固定key:cdd17ab29b84b32552ddcfbb4abf0225(hex编码),随机iv(iv为16位,由0-9,a-f组成)的AES/CBC/PKCS5Padding加密。加密后会把iv拼接到密文前边然后再进行一个base64编码，然后就得到的了edata

edata的明文，也就是拼接的treeMap如下：
app_name=shop_android　==>固定的
app_version=7.83.3　　==>固定的，版本号
client_type=android　　==>固定的，客户端类型

```
dinfo=
{"ah1":"","ah2":"","ah3":"","ah4":"wifi","ah5":"1080_1794","ah6":1593600,"ah7":4
,"ah8":3948359680,"ah9":"Pixel","ah10":"","ah11":"","ah12":"","ah13":"","as1":"1
0","as2":"","as3":"","as4":"87cd3a74b10f74a1","as5":"","as6":"","as7":"29","ac1"
:"c4acd008-4f30-3599-b32c-7f7dbc2135e9"}  ==>设备信息
mars_cid=c4acd008-4f30-3599-b32c-7f7dbc2135e9  ==>由设备信息去获得的固定uuid
phone_model=Pixel
session_id=c4acd008-4f30-3599-b32c-7f7dbc2135e9_shop_android_1748338429027 ==>由
设备信息去获得的固定uuid+app_name+毫秒级时间戳
sys_version=29              ===>sdk版本号
vcspKey=4d9e524ad536c03ff203787cf0dfcd29   ==>固定的
vcspToken=NGQ5ZTUyNGFkNTM2YzAzZmYyMDM3ODdjZjBkZmNkMjl8fHwxNzUwOTMwNDU2fHx8.cdf5e
1e7fc60cbd19f154929191d32ce ==>向服务器发送请求拿到的


这就是edata的生成
```

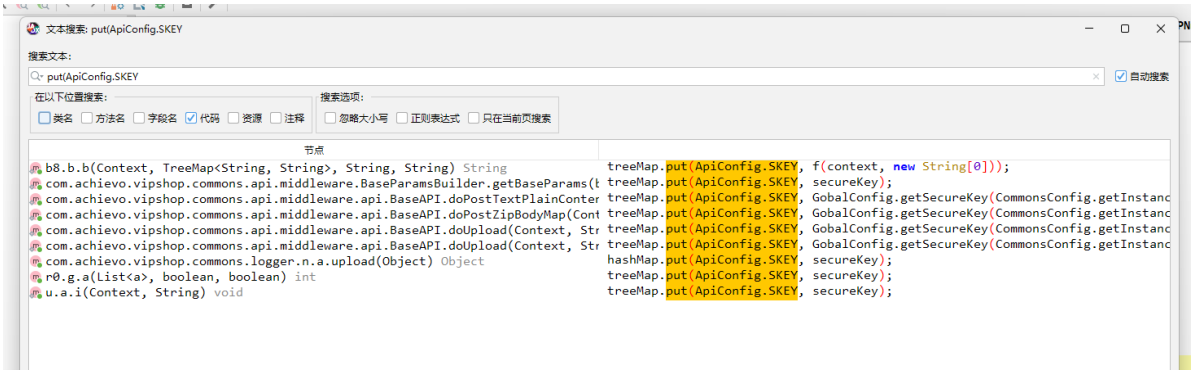
eversion

```
//在getEData函数中，我们主要到了，有往treeMap中put了一个eversion
public Map<String, String> getEData(TreeMap<String, String> treeMap) {
    if (treeMap == null || treeMap.isEmpty()) {
        return null;
    }
    TreeMap treeMap2 = new TreeMap();
    TreeMap treeMap3 = new TreeMap();
    for (Map.Entry<String, String> entry : treeMap.entrySet()) {
        if (entry != null) {
            if (ignoreArrayList.contains(entry.getKey())) {
                treeMap3.put(entry.getKey(), entry.getValue());
            } else {
                treeMap2.put(entry.getKey(), entry.getValue());
            }
        }
    }
    treeMap3.put(ApiConfig.EVERSION, "0");
    try {
        BaseSDK.getSecurityBasicService();
        treeMap3.put(ApiConfig.EDATA,
VCSPSecurityBasicService.encryptedParams(treeMap2, 0));
    } catch (Exception e10) {
        VCSPMyLog.error(SdkNetworkServiceConfig.class, e10);
    }
    return treeMap3;
}



treeMap3.put(ApiConfig.EVERSION, "0");


//直接put的0
```

skey

```
//我们直接去搜索关键字==>找到一条定义的常量
```


```
//查找用例看看==>找到好多条，我又hook了一下treeMap 的put方法，发现值都是一样的，我们随便进去一个看看，
```




```
//发现有几条都是走的
GobalConfig.getSecureKey(CommonsConfig.getInstance().getContext())
//所以我们去看看这个：”
public static String getSecureKey(Context context) {
    try {
        return b.f(context, new String[0]);
    } catch (Throwable th2) {
        d.d(GobalConfig.class, th2);
        d.f(GobalConfig.class, "AppConfig getSecureKey");
        return null;
    }
}
//进来后发现走的是f(context, new String[0])和我们查找用例的第一条对上了，再去看看那三条put的是secureKey的
String secureKey =
GobalConfig.getSecureKey(CommonsConfig.getInstance().getContext());
if (!TextUtils.isEmpty(secureKey)) {
    treeMap.put(ApiConfig.SKEY, secureKey);
}
//进来后发现其值也是通过调用getSecureKey得到的，getSecureKey又是调用的b.f(context, new String[0])所以源头还是b.f(context, new String[0])
//去看看b.f(context, new String[0])：
public static String f(Context context, String... strArr) {
    if (TextUtils.isEmpty(f2017b)) {
        String info = KeyInfoFetcher.getInfo(context, ApiConfig.SKEY);
        f2017b = info;
        if (TextUtils.isEmpty(info) || f2017b.startsWith("KI ")) {
            KeyInfoFetcher.loadKeyInfoSoWarp((strArr == null || strArr.length <= 0) ? "" : strArr[0]);
            f2017b = KeyInfoFetcher.getInfo(context, ApiConfig.SKEY);
```

```
        }
    }
    return f2017b;
}

//最后返回的f2017b，f2017b是通过调用KeyInfoFetcher.getInfo(context, ApiConfig.SKEY)得到的
//去看KeyInfoFetcher.getInfo(context, ApiConfig.SKEY)：
public static String getInfo(Context context, String str) {
    try {
        if (clazz == null || object == null || method == null) {
            int i10 = KeyInfo.f69594a;
            clazz = KeyInfo.class;
            object = KeyInfo.class.newInstance();
            method = clazz.getMethod("getInfo", Context.class, String.class);
        }
        return (String) method.invoke(object, context, str);
    } catch (Exception e10) {
        VCSPMyLog.error(KeyInfoFetcher.class, e10);
        return "";
    }
}
//反射调用了KeyInfo下的getInfo
public static String getInfo(Context context, String str) {
    try {
        try {
            return getNavInfo(context, str);
        } catch (Throwable th2) {
            return "KI gi: " + th2.getMessage();
        }
    } catch (Throwable unused) {
        SoLoader.load(context, LibName);
        return getNavInfo(context, str);
    }
}
//里边调用了getNavInfo(context, str);
private static native String getNavInfo(Context context, String str);
//这个我们之前分析过了，是根据传进去的str映射返回的，也就是固定的
```

timestamp

```
//秒级时间戳
```

did-总结

到这里did就搞定了

did是通过向服务器发送请求得到的

请求网址:https://mapi.appvipshop.com/vips-mobile/rest/device/generate_token
请求方式：POST
请求头：

```
    authorization   OAuth api_sign=f6982ac2af558e2bfcf8471900afa44be45e7495
    content-type    application/x-www-form-urlencoded
    content-length  1414
    accept-encoding gzip
    user-agent  okhttp/4.9.1
```

请求体：

```
    api_key 23e7f28019e8407b98b84cd05b5aef2c
    did
    edata
    eversion    0
    skey    6692c461c3810ab150c9a980d0c275ec
    timestamp   1748257157
```

这些参数在上边都分析过了

fdc_area_id

```
    fdc_area_id 101102101
```

```
//我们回顾一下之前怎么分析的
        //通过搜索url定位到了一个方法==>getProductList==>看名字是获取商品列表
        //在这个方法中首先new了一个 UrlFactory urlFactory = new UrlFactory(true,
true, true);
        //然后调用UrlFactory下的setParam方法，设置了一些参数
        //在new UrlFactory(true, true, true);的时候，也初始化了一些参数



public UrlFactory(boolean z10, boolean z11, boolean z12) {
        this.params = new TreeMap<>(new Comparator<Object>() {
            @Override // java.util.Comparator
            public int compare(Object obj, Object obj2) {
                if (obj == null || obj2 == null) {
                    return 0;
                }
                return String.valueOf(obj).compareTo(String.valueOf(obj2));
            }
        });
        this.params.putAll(BaseParamsBuilder.getBaseParams(z10, z11));
        if (z12) {
```

```
 BaseParamsBuilder.addOtdParams(CommonsConfig.getInstance().getContext(),
this.params);
        }
    }


//它会通过BaseParamsBuilder.getBaseParams(z10, z11)获取到一个treeMap放进params这个
treeMap中，params就是放的请求体。并且z10，z11，z12是在new UrlFactory的时候传入的，都是
true。因为z12是true所以它会再调用一个addOtdParams方法，往params中放一些数据
public static void addOtdParams(Context context, Map<String, String> map) {
    if (map == null) {
        return;
    }
    map.put("width",
String.valueOf(CommonsConfig.getInstance().getScreenWidth()));
    map.put("height",
String.valueOf(CommonsConfig.getInstance().getScreenHeight()));
    map.put("net", SDKUtils.getNetWorkType(context));
    map.put(ApiConfig.MAKER, Build.BRAND.toUpperCase());
    map.put(ApiConfig.OS, "Android");
    map.put(ApiConfig.OSV, Build.VERSION.RELEASE);
    if (context != null) {
        map.put(ApiConfig.OTDDID, SDKUtils.getImeiOrOaid(context));
    }
}


//所以就是三个地方都有设置请求体：getProductList方法中、new UrlFactory(true, true,
true)的时候在BaseParamsBuilder.getBaseParams(z10, z11)中，还有就是addOtdParams方法中


//现在我们去找fdc_area_id==>在getBaseParams方法中：
treeMap.put("fdc_area_id", ApiConfig.getInstance().getFdcAreaId());
//是通过ApiConfig.getInstance().getFdcAreaId()获取的。我们去hook一下这个
getFdcAreaId()方法 ，确实走了这里，我们去看下它的逻辑：
public String getFdcAreaId() {
    String fdcAreaId = CommonsConfig.getInstance().getFdcAreaId();
    return TextUtils.isEmpty(fdcAreaId) ? "104104" : fdcAreaId;
}

//进来后先通过CommonsConfig.getInstance().getFdcAreaId()拿到一个值，如果为空就返
回"104104"，不为空就返回拿到的这个
//我们去看下怎么拿到的
public String getFdcAreaId() {
    if (TextUtils.isEmpty(this.fdcAreaId)) {
        return getProvince_id();
    }
    return this.fdcAreaId;
}
//好像是获取省份的信息？Province_id
//进来后先判断fdcAreaId是否为空，不为空就返回，为空就返回getProvince_id()

//这里就是省份id
```

mars_cid

这个在edata中分析过，是通过设备信息获取到的固定的uuid

session_id

session_id在搜索接口中就是固定值+毫秒级时间戳 session_id _shop_android_1748048965624

skey

是把字符串"sket"传到so层，然后通过映射得到的，可以直接理解为固定值

搜索总结

到这里就结束了，其他的参数多次请求后发现是固定的，只有搜索的关键字那几个参数需要换一换，其他的 参数都是固定的!