包名：com.che168.autotradercloud

主要任务

逆向登录接口，使用python实现登录，拿到token

过强制更新

方法1

直接断网打开APP

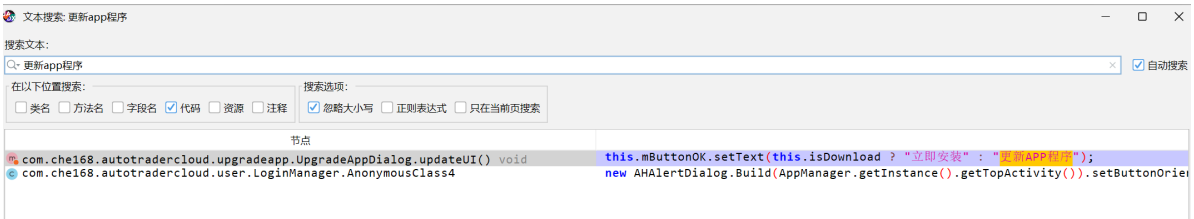
方法2

为了学习，我们去反编译apk，找一找强制更新的逻辑，然后使用`frida`去`hook`掉，从而达到去除强制更新的目的

打开后会有强制更新的弹窗，我们去`jadx`中搜一下关键字，`有新版了`、`更新APP程序`，那个具体版本和更新功能就不搜了，一般都是动态从服务器获取的，我们也可以抓包，然后去charles中搜索版本号和更新功能

jadx搜索关键字

搜索`有新版了`关键字没找到，搜索`更新APP程序`找到了一条，大致看了下是设置按钮名字的，如下：



```
96          if (this.mUpgradeBean.isForce()) {
97              setCancelable(false);
98              setCanceledOnTouchOutside(false);
99              this.mButtonCancel.setVisibility(8);
            } else {
102             setCancelable(true);
103             setCanceledOnTouchOutside(true);
104             this.mButtonCancel.setVisibility(0);
            }
107         if (this.mUpgradeBean.getUpdateType() == 1) {
108             textView.setVisibility(0);
109             textView2.setVisibility(8);
110             if (this.isForceSwitchUpdate) {
111                 updatetext = this.mContext.getString(R.string.force_update_app_tip, this.mUpgradeBean.getUpdatetext());
                } else {
112                 updatetext = this.mUpgradeBean.getUpdatetext();
                }
110             textView.setText(updatetext);
114             this.mButtonOK.setText(this.isDownload ? "立即安装" : "更新APP程序");
            } else {
117             textView.setVisibility(8);
118             textView2.setVisibility(0);
119             textView2.setText(this.mUpgradeBean.getUpdatetext());
120             this.mButtonOK.setText("我知道了");
            }
123         this.mButtonOK.setOnClickListener(new View.OnClickListener() { // from class: com.che168.autotradercloud.upgradeapp.Upgr
                @Override // android.view.View.OnClickListener
127             public void onClick(View view) {
```

这里是通过判断是否已经下载了安卓包，若下载了，按钮设置为立即安装，若没下载设置为更新APP程序，我们去阅读一下大体逻辑

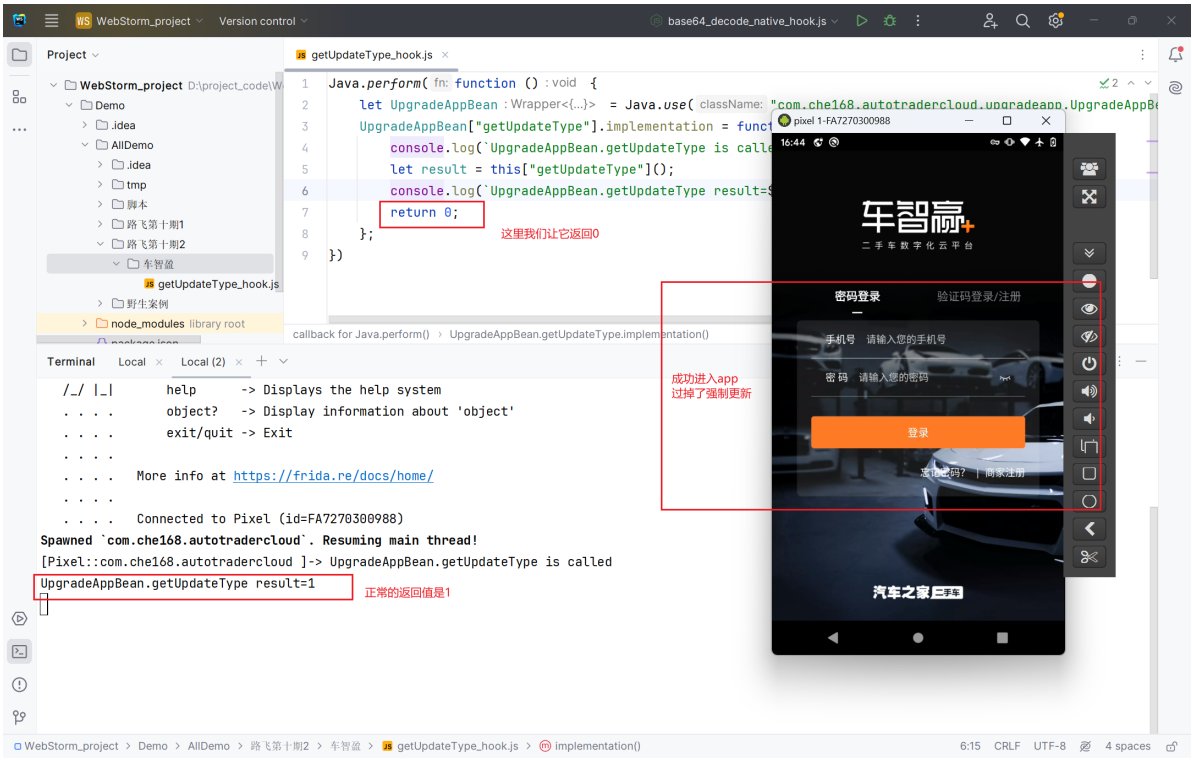
```
107         if (this.mUpgradeBean.getUpdateType() == 1) {
108             textView.setVisibility(0);
109             textView2.setVisibility(8);
110             if (this.isForceSwitchUpdate) {
111                 updatetext = this.mContext.getString(R.string.force_update_app_tip, this.mUpgradeBean.getUpdatetext());
                } else {
112                 updatetext = this.mUpgradeBean.getUpdatetext();
                }
110             textView.setText(updatetext);
114             this.mButtonOK.setText(this.isDownload ? "立即安装" : "更新APP程序");
            } else {
117             textView.setVisibility(8);
118             textView2.setVisibility(0);
119             textView2.setText(this.mUpgradeBean.getUpdatetext());
120             this.mButtonOK.setText("我知道了");
            }
```

可以看到他是通过判断this.mUpgradeBean.getUpdateType() 是否等于1，如果等于1，就会进入这个这个分支，进而就会执行`this.mButtonOK.setText(this.isDownload ? "立即安装" : "更新APP程序");`

我们可以看到，在`this.mButtonOK.setText(this.isDownload ? "立即安装" : "更新APP程序");`之前有一个`textView.setText(updatetext);`,这条语句是给textView设置一个信息的，并且可以看到参数是`updatetext`,盲猜应该是要更新的内容，这个信息应该是从服务器获取到的

所以我们过掉更新的第一个入口可以是不走这个if分支，进而去走else；若想不走这个if分支，只能是将`this.mUpgradeBean.getUpdateType()`的返回值不为1，所以我们直接去hook这个`this.mUpgradeBean.getUpdateType()`

让它的返回值为非1的数字，使用spwan的方式hook



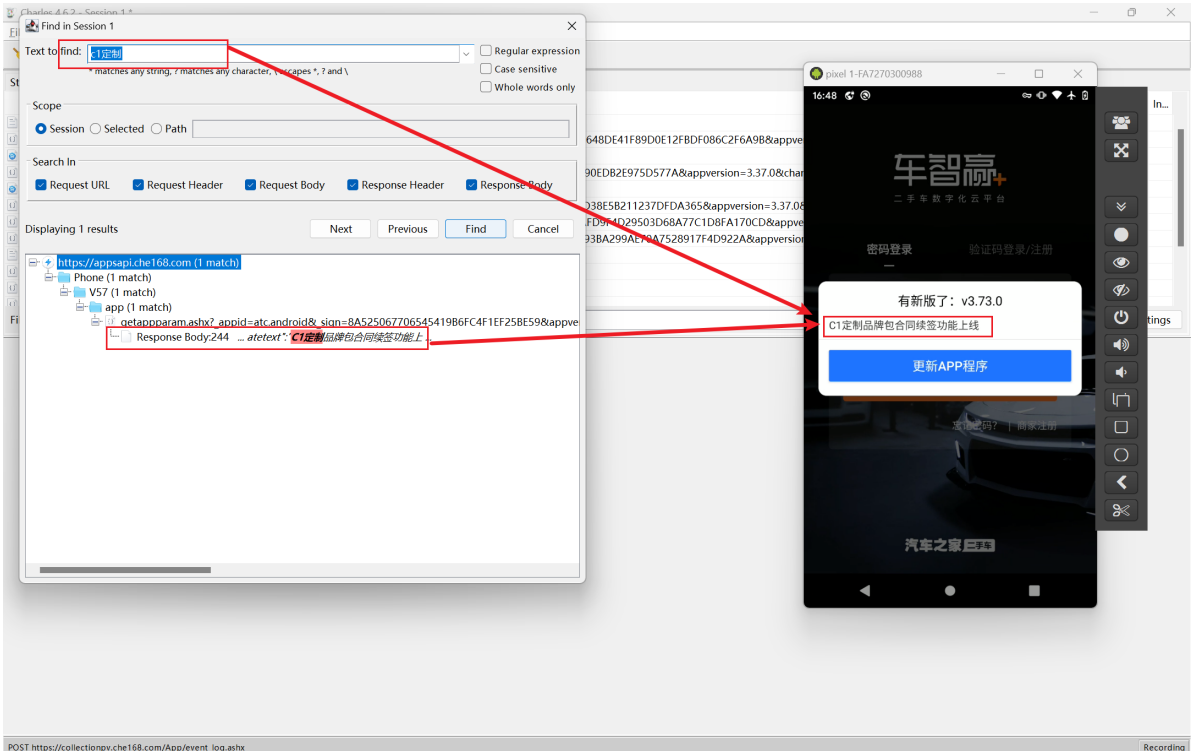
抓包搜索关键字

这里我们说过，这个版本号：v3.73.0 以及更新的内容：C1定制品牌包合同....

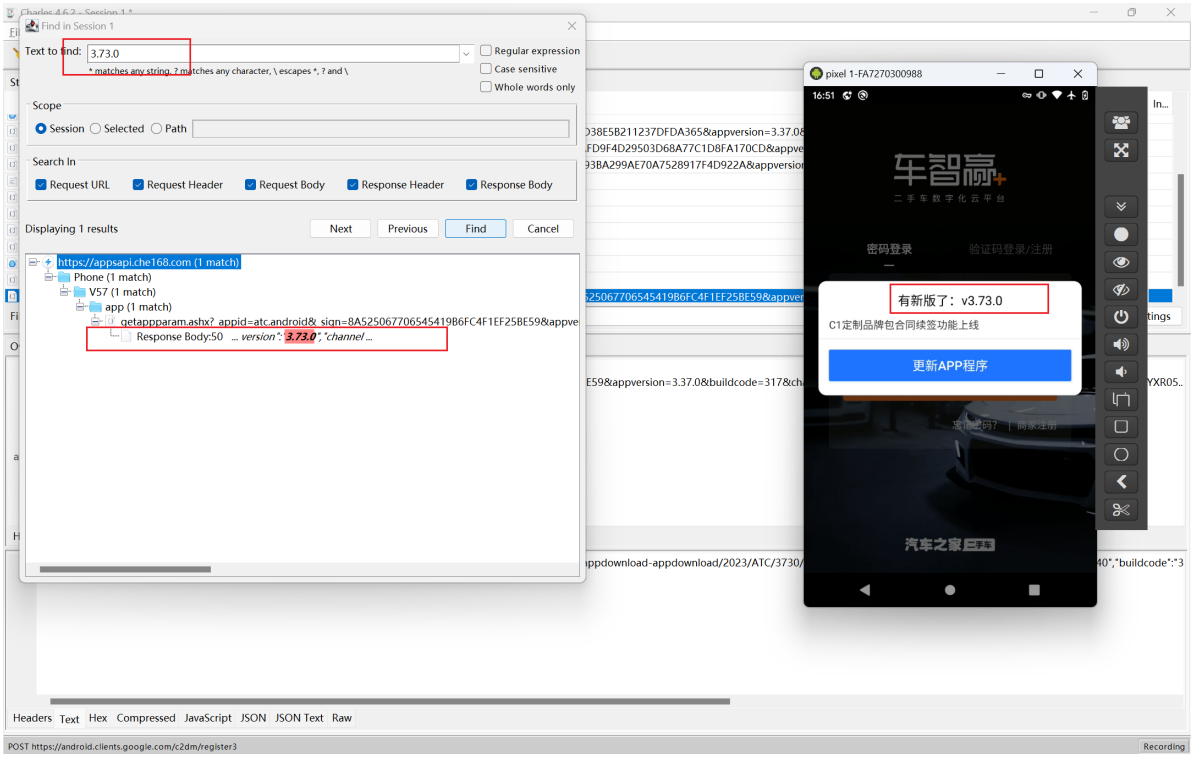
这两块内容肯定是动态从服务器获取的，不可能写死在程序里，所以我们在打开app时进行抓包，然后去搜索这些动态的关键字，进而找到请求网址，从而在APP内找到发送请求的地方吗，进而去过掉强制更新

所以我们去重启APP，抓包，搜索关键字：

搜索更新内容：



直接搜索版本号没有搜到，这里不带v只搜索3.73.0搜到了一条结果，和直接搜索更新内容搜到的是同一个位置：



请求信息：

```
请求网址：https://appsapi.che168.com/Phone/V57/app/getappparam.ashx
请求方式：GET
```

这里重点看请求网址和请求方式就行了，请求参数和请求头没必要看，我们主要是通过请求网址去找检测版本的逻辑，现在带着请求网址的后半部分去jadx中搜索一下：

精准找到一条信息，进去看看：

```
/* loaded from: classes3.dex */
public class UpgradeAppModel extends BaseModel {
    private static final String UPDATE_URL = HostHelp.HOST_APPS_API + "/Phone/V57/app/getappparam.ashx";

    public static void checkUpdate(Context context, ResponseCallback<UpgradeAppBean> responseCallback) {
        HttpUtil.Builder builder = new HttpUtil.Builder();
        HttpUtil.Builder method = builder.tag(context).method(HttpUtil.Method.GET);
        method.param("plaform", "atc.android." + AppUtils.getChannelId(context)).param(UmsConstants.KEY_VERSION_DEBUG, SystemUtil.
        doRequest(builder, responseCallback, new TypeToken<BaseResult<UpgradeAppBean>>() { // from class: com.che168.autotraderclo
        }.getType());
    }
}
```

这里是定义了一个常量，我们去查找一下用例，看看在哪里调用了它：

只有一条调用的地方，很好！我们点进去看看：

```
/* loaded from: classes3.dex */
public class UpgradeAppModel extends BaseModel {
    private static final String UPDATE_URL = HostHelp.HOST_APPS_API + "/Phone/V57/app/getappparam.ashx";

    public static void checkUpdate(Context context, ResponseCallback<UpgradeAppBean> responseCallback) {
        HttpUtil.Builder builder = new HttpUtil.Builder();
        HttpUtil.Builder method = builder.tag(context).method(HttpUtil.Method.GET);
        method.param("plaform", "atc.android." + AppUtils.getChannelId(context)).param(UmsConstants.KEY_VERSION_DEBUG, SystemUtil.
        doRequest(builder, responseCallback, new TypeToken<BaseResult<UpgradeAppBean>>() { // from class: com.che168.autotraderclo
        }.getType());
    }
}
```

其实就是在这个`checkUpdate`方法里调用的，不过在最后边，代码太长了，挡住了！

这里就是发送了检查更新的请求，并且将响应通过`ResponseCallback`回调接口传回去，我们去看看谁调用了这个`checkUpdate`方法

有两个地方调用了`checkUpdate`方法,所以我们去hook一下这个`checkUpdate`方法，打印个堆栈看看：

来到了`checkVersions`方法里边，看到调用了`checkUpdate`方法，

```
51      public void checkVersions(final Context context, final boolean z, UpgradeAppCheckListener upgradeAppCheckListener) {
            if (context == null) {
                return;
            }
52          this.mUpgradeAppCheckListener = upgradeAppCheckListener;
53          UpgradeAppModel.checkUpdate(context, new ResponseCallback<UpgradeAppBean>() { // from class: com.che168.autotradercloud.
                @Override // com.che168.autotradercloud.base.httpNew.ResponseCallback
56              public void success(UpgradeAppBean upgradeAppBean) {
57                  if (z && upgradeAppBean != null) {
58                      upgradeAppBean.setForceSwitchUpdate(true);
                    }
60                  UpgradeAppAgent.this.mUpgradeBean = upgradeAppBean;
62                  if (UpgradeAppAgent.this.mUpgradeBean != null) {
64                      if (UpgradeAppAgent.this.isWifiActiveDownload && ATCNetworkUtil.isWifi(context) && !UpgradeAppAgent.this.mUp
65                          UpgradeAppAgent upgradeAppAgent = UpgradeAppAgent.this;
                            upgradeAppAgent.startDownLoad(upgradeAppAgent.mUpgradeBean);
68                      } else if (UpgradeAppAgent.this.mUpgradeAppCheckListener != null) {
69                          UpgradeAppAgent.this.mUpgradeAppCheckListener.onSuccess(UpgradeAppAgent.this.mUpgradeBean);
                        }
74                  } else if (UpgradeAppAgent.this.mUpgradeAppCheckListener != null) {
75                      UpgradeAppAgent.this.mUpgradeAppCheckListener.onSuccess(UpgradeAppAgent.this.mUpgradeBean);
                    }
                }

                @Override // com.che168.autotradercloud.base.httpNew.ResponseCallback
82              public void failed(int i, ApiException apiException) {
83                  if (UpgradeAppAgent.this.mUpgradeAppCheckListener != null) {
84                      UpgradeAppAgent.this.mUpgradeAppCheckListener.onFailure(apiException);
                    }
                }
            });
```

然后在回调中，首先是判断`z && upgradeAppBean != null`然后设置是否强制更新，然以后再去判断其他的，

其中有一个是判断完了走向`UpgradeAppAgent.this.mUpgradeAppCheckListener.onSuccess(UpgradeAppAgent.this.mUpgradeBean);`

这里是通过外部监听器把服务器返回的信息传回去，进而去执行一些其他操作

并且在刚进入这个方法时，先对`this.mUpgradeAppCheckListener`进行了赋值，把从上层传过来的`upgradeAppCheckListener`赋值上去了。然后调用的它下边的onSuccess方法，其实就是调用了上层传过来的`upgradeAppCheckListener.onSuccess`，所以我们去看在调用的时候，是如何重写的这个`onSuccess`方法：

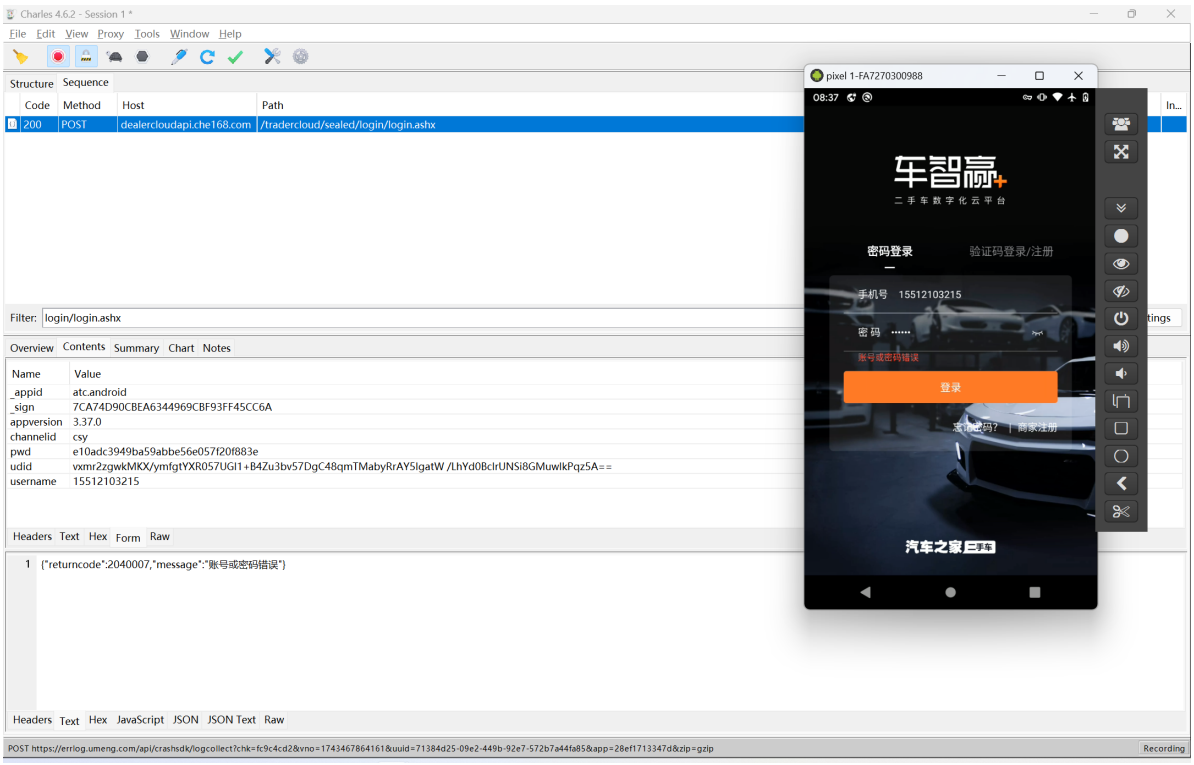
```
this.mUpgradeAppAgent.checkVersions(
    this.mBulder.activity,
    this.mBulder.isForceSwitchUpdate,
    new UpgradeAppAgent.UpgradeAppCheckListener() {
          public void onSuccess(UpgradeAppBean upgradeAppBean) {
                UpgradeAppManager.this.mUploadDataBean = upgradeAppBean;
                if (UpgradeAppManager.this.mBulder.checkListener != null) {

 UpgradeAppManager.this.mBulder.checkListener.onSuccess(UpgradeAppManager.this.m
UploadDataBean);
          }
                if (UpgradeAppManager.this.mBulder.isShowDialog &&
UpgradeAppManager.this.mUploadDataBean != null) {
         UpgradeAppManager upgradeAppManager2 = UpgradeAppManager.this;

upgradeAppManager2.showUpdateDialog(upgradeAppManager2.mBulder.activity,
UpgradeAppManager.this.mUploadDataBean, null);
                } else if (UpgradeAppManager.this.mBulder.mIAppUpdateListener !=
null) {
                    UpgradeAppManager.this.mBulder.mIAppUpdateListener.onNext();
                }
            }
```

这个onSuccess方法中主要逻辑就是两个if表达式，我们需要hook判断走了哪一个。

跟到这里，发现这个方法不太容易去做，所以对于这种我们还是直接搜关键字或者是断网进app更好一些

登录接口

我们去做账号密码登录的接口，直接hook上java层加解密算法，去探探路：



抓到包了：

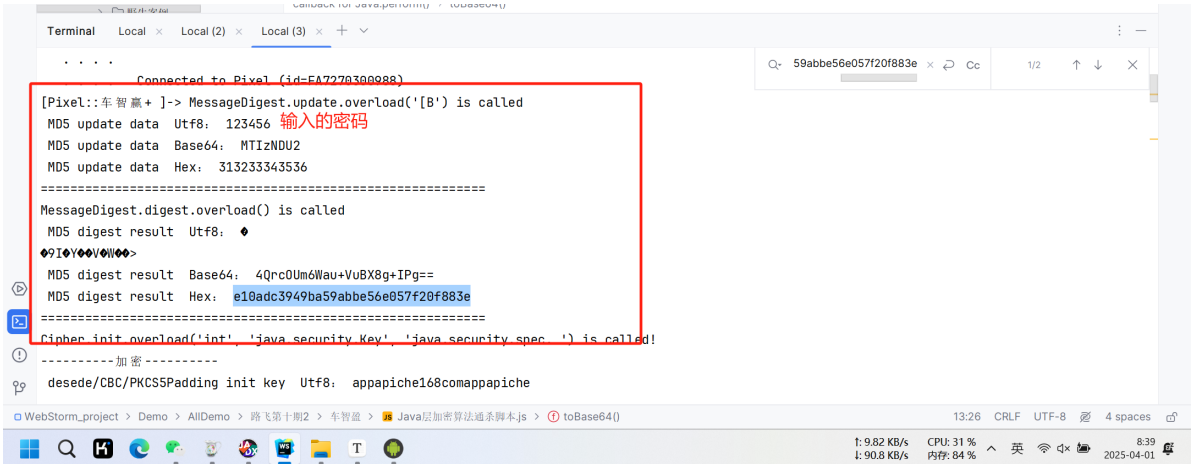
```
请求地址:https://dealercloudapi.che168.com/tradercloud/sealed/login/login.ashx
请求方式：POST
请求头：
    cache-control       public, max-age=0
    traceid             atc.android_349d8c4f-2cee-43ef-a684-0e4d2d4b4f54
    content-type        application/x-www-form-urlencoded
    content-length      254
    accept-encoding     gzip
    user-agent          okhttp/3.14.9
请求体：
    _appid              atc.android
    _sign               7CA74D90CBEA6344969CBF93FF45CC6A
    appversion          3.37.0
    channelid           csy
    pwd                 e10adc3949ba59abbe56e057f20f883e
    udid
vxmr2zgwkMKX/ymfgtYXR057UGl1+B4Zu3bv57DgC48qmTMabyRrAY5IgatW
/LhYd0BclrUNSi8GMuwlkPqz5A==
```

```
username              15512103215
```

请求体

pwd

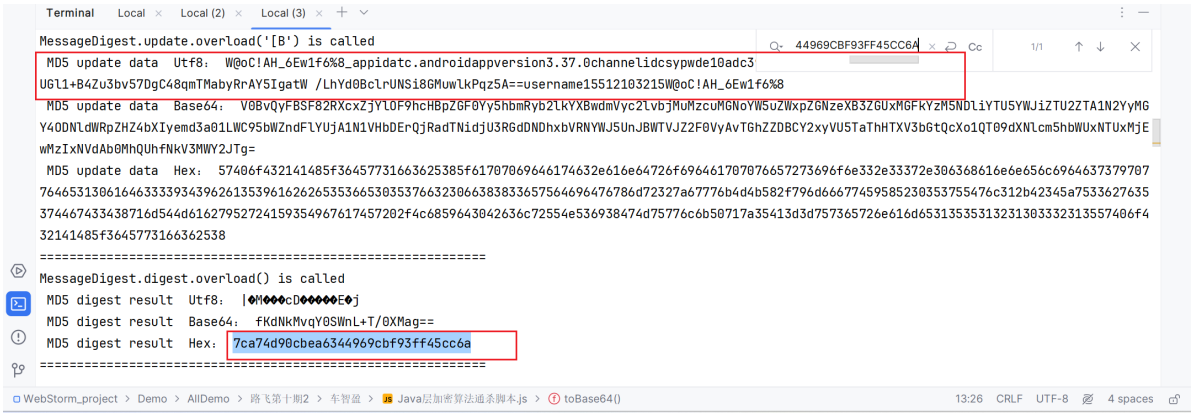
我们拿pwd的值去hook得到的信息中搜一下：



这里是直接用了一个MD5然后进行了hex编码

_sign

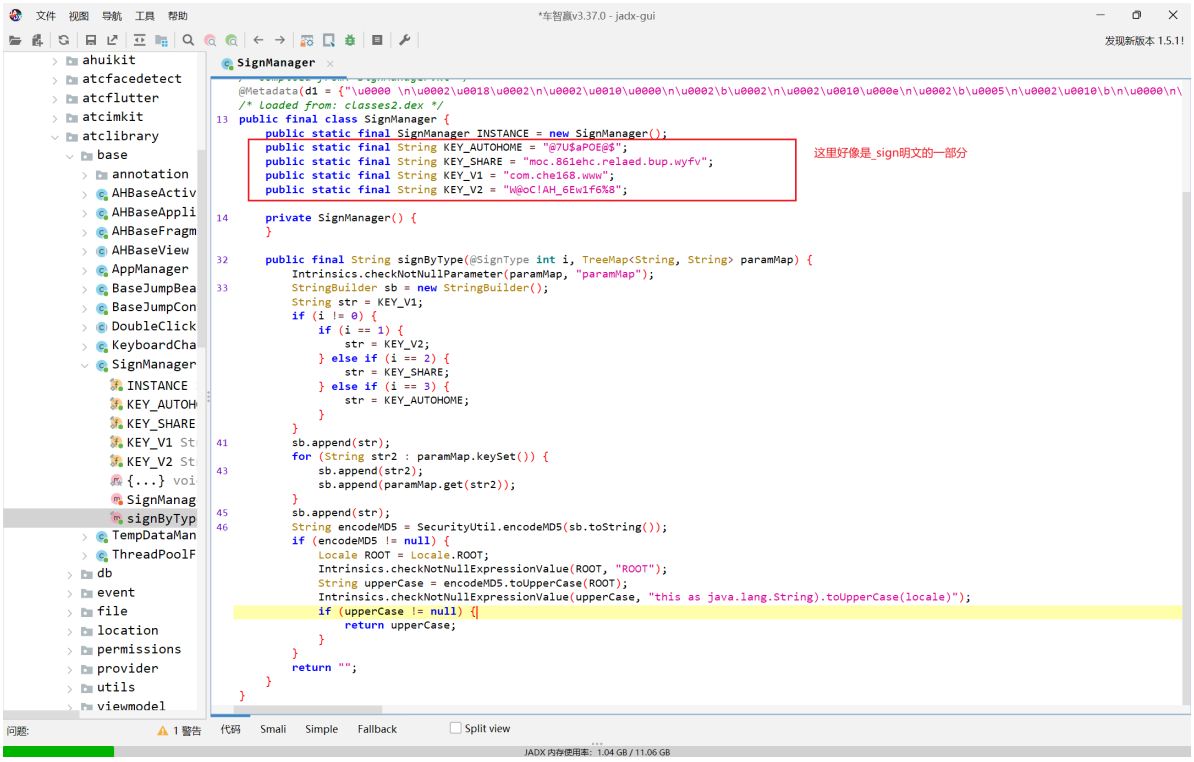
再去搜索一下_sign：



这里也是做了一个MD5加密，但是明文不知道怎么来的，我们去打印一下MD5的调用栈，然后去jadx中看看逻辑：

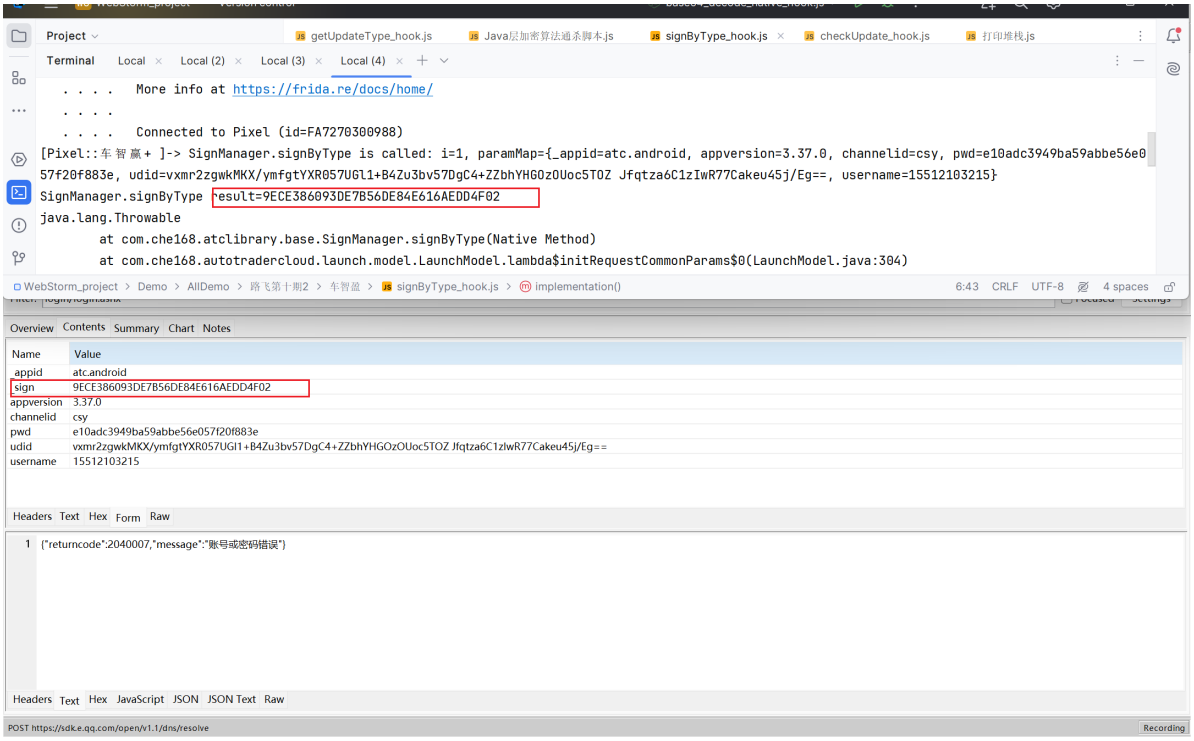
```
java.lang.Throwable
        at java.security.MessageDigest.update(Native Method)
        at com.autohome.ahkit.utils.SecurityUtil.encodeMD5(SecurityUtil.java:106)
        at com.che168.atclibrary.base.SignManager.signByType(SignManager.kt:46)
        at com.che168.autotradercloud.launch.model.LaunchModel.lambda$initRequestCommonParams$0(LaunchModel.java:304)
        at com.che168.autotradercloud.launch.model.LaunchModel$$ExternalSyntheticLambda0.checkParams(Unknown Source:0)
        at com.che168.ahnetwork.http.HttpUtil$Builder.checkParams(HttpUtil.java:554)
        at com.che168.ahnetwork.http.HttpUtil$Builder.doRequest(HttpUtil.java:490)
        at com.che168.ahnetwork.http.HttpUtil$Builder.doRequest(HttpUtil.java:428)
        at com.che168.autotradercloud.base.httpNew.BaseModel.doRequest(BaseModel.java:104)
        at com.che168.autotradercloud.user.model.UserModel.loginByPassword(UserModel.java:273)
        at com.che168.autotradercloud.user.model.UserModel.login(UserModel.java:1473)
        at com.che168.autotradercloud.user.LoginActivity.login(LoginActivity.java:156)
        at com.che168.autotradercloud.user.view.LoginView$1.onClick(LoginView.java:150)
```

我们先去看这个`at com.che168.atclibrary.base.SignManager.signByType(SignManager.kt:46)`吧



、

我们去hook一下这个`signByType`方法，看看有没有走这里：



没问题，确实是在这里生成的_sign，看下刚刚hook的信息：

```
SignManager.signByType is called:
i=1,
paramMap={_appid=atc.android, appversion=3.37.0, channelid=csy,
pwd=e10adc3949ba59abbe56e057f20f883e,
udid=vxmr2zgwkMKX/ymfgtYXR057UGl1+B4Zu3bv57DgC4+ZZbhYHGOzOUoc5TOZ
Jfqtza6C1zIwR77Cakeu45j/Eg==, username=15512103215}

SignManager.signByType result=9ECE386093DE7B56DE84E616AEDD4F02
```

```
java.lang.Throwable
        at com.che168.atclibrary.base.SignManager.signByType(Native Method)
        at
com.che168.autotradercloud.launch.model.LaunchModel.lambda$initRequestCommonPara
ms$0(LaunchModel.java:304)
        at
com.che168.autotradercloud.launch.model.LaunchModel$$ExternalSyntheticLambda0.ch
eckParams(Unknown Source:0)
        at
com.che168.ahnetwork.http.HttpUtil$Builder.checkParams(HttpUtil.java:554)
        at
com.che168.ahnetwork.http.HttpUtil$Builder.doRequest(HttpUtil.java:490)
        at
com.che168.ahnetwork.http.HttpUtil$Builder.doRequest(HttpUtil.java:428)
        at
com.che168.autotradercloud.base.httpNew.BaseModel.doRequest(BaseModel.java:104)
        at
com.che168.autotradercloud.user.model.UserModel.loginByPassword(UserModel.java:2
73)
        at
com.che168.autotradercloud.user.model.UserModel.login(UserModel.java:1473)
        at
com.che168.autotradercloud.user.LoginActivity.login(LoginActivity.java:156)
        at
com.che168.autotradercloud.user.view.LoginView$1.onClick(LoginView.java:150)
        at android.view.View.performClick(View.java:7140)
        at android.view.View.performClickInternal(View.java:7117)
        at android.view.View.access$3500(View.java:801)
        at android.view.View$PerformClick.run(View.java:27351)
        at android.os.Handler.handleCallback(Handler.java:883)
        at android.os.Handler.dispatchMessage(Handler.java:100)
        at android.os.Looper.loop(Looper.java:214)
        at android.app.ActivityThread.main(ActivityThread.java:7356)
        at java.lang.reflect.Method.invoke(Native Method)
        at
com.android.internal.os.RuntimeInit$MethodAndArgsCaller.run(RuntimeInit.java:492)
```

可以看到，这个方法的参数是一个int，一个treeMap。

这个treeMap中放的就是我们请求体中的所有信息，唯独没有 *sign*,sign是在这个方法中生成的。

我们先去刚刚hook MD5得信息拿过来，看看传进去的参数是什么：

```
_sign的明文：
W@oC!AH_6Ew1f6%8_appidatc.androidappversion3.37.0channelidcsypwde10adc3949ba59ab
be56e057f20f883eudidvxmr2zgwkMKX/ymfgtYXR057UGl1+B4Zu3bv57DgC48qmTMabyRrAY5IgatW
/LhYdOBclrUNSi8GMuwlkPqz5A==username15512103215W@oC!AH_6Ew1f6%8


生成_sign时传入的treeMap：
_appid=atc.android, appversion=3.37.0, channelid=csy,
pwd=e10adc3949ba59abbe56e057f20f883e,
udid=vxmr2zgwkMKX/ymfgtYXR057UGl1+B4Zu3bv57DgC4+ZZbhYHGOzOUoc5TOZ
Jfqtza6C1zIwR77Cakeu45j/Eg==, username=15512103215
```

对比完可以看到，首先是把treeMap中的key value拿出来拼到一起，然后再在前边加上W@oC!AH_6Ew1f6%8后边加上W@oC!AH_6Ew1f6%8然后进行了MD5，MD5后进行hex编码，然后转为大写，就得到了_sign

所以我们把其他的参数搞出来，_sign也就出来了

我们通过signByType的调用栈往回追，看看其他参数怎么生成的

```
public static  TreeMap lambda$initRequestCommonParams$0(int i, TreeMap treeMap)
{
        if (!treeMap.containsKey("_appid")) {
            treeMap.put("_appid", "atc.android");
        }
        if (!treeMap.containsKey("channelid")) {
            treeMap.put("channelid",
AppUtils.getChannelId(ContextProvider.getContext()));
        }
        if (!treeMap.containsKey("appversion")) {
            treeMap.put("appversion",
SystemUtil.getAppVersionName(ContextProvider.getContext()));
        }
        if (!treeMap.containsKey("udid")) {
            treeMap.put("udid", AppUtils.getUDID(ContextProvider.getContext()));
        }
        String userKey = UserModel.getUserKey();
        if (!ATCEmptyUtil.isEmpty((CharSequence) userKey)) {
            treeMap.put("userkey", userKey);
        }
        checkNullParams(treeMap);
        treeMap.put("_sign", SignManager.INSTANCE.signByType(i, treeMap));
        return treeMap;
    }
```

这里可以看到是put了所有请求体的参数，放进里treeMap中，其中还有一个userkey，这个是先UserModel.getUserKey()，如果有值，就放进去，没有就不放，登录场景下应该是没有的，所以直接看其他的参数：

_appid

固定的："atc.android"

channelid

这里固定是：csy

appversion

这个版本下是3.37.0

udid

```
treeMap.put("udid", AppUtils.getUDID(ContextProvider.getContext()));

 public static String getUDID(Context context) {
        return SecurityUtil.encode3Des(context, getIMEI(context) +"|"+
System.nanoTime() + "|"+ SPUtils.getDeviceId());
    }
```

这个udid是加密了的，我们可以看到是3des加密，并且我尝试删除udid再发包，发现会签名失败

我们去hook了一下这个encodeDes方法，打印了参数和返回值：

```
参数：26c6d00d-cf45-302e-8f9d-77e4f49a5b49|392833184283903|424789
返回值：vxmr2zgwkMKX/ymfgtYXRO57UGl1+B4Zu3bv57DgC490krgZ0hu1bVT/WzyH
gFmKa9C2iIL2nXujVhcWkPSrTA==
```

encode3Des方法：

```
private static final String encoding = "UTF-8";
    private static final String iv = "appapich";
    private static final char[] legalChars =
"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/".toCharArray()
;


public static String encode3Des(Context context, String str) {
        String desKey = AHAPIHelper.getDesKey(context);
        byte[] bArr = null;
        if (TextUtils.isEmpty(desKey)) {
            return null;
        }
        try {
            SecretKey generateSecret =
SecretKeyFactory.getInstance("desede").generateSecret(new
DESedeKeySpec(desKey.getBytes()));
            Cipher cipher = Cipher.getInstance("desede/CBC/PKCS5Padding");
            cipher.init(1, generateSecret, new IvParameterSpec(iv.getBytes()));
            bArr = cipher.doFinal(str.getBytes("UTF-8"));
        } catch (Exception unused) {
```

```
        }
        return encode(bArr).toString();
    }
```

可以看到是desede/CBC/PKCS5Padding加密

key为： String desKey = AHAPIHelper.getDesKey(context);

iv为：appapich

这里3des的key是通过AHAPIHelper.getDesKey(context);得到的：

```
    public static String getDesKey(Context context) {
        if (TextUtils.isEmpty(mDesKey)) {
            getSignDesKey(context);
        }
        return mDesKey;
    }
```

进来后又是调用了getSignDesKey给mDesKey赋值的

```
    private static void getSignDesKey(Context context) {
        mDesKey = CheckSignUtil.get3desKey(context);
    }
```

getSignDesKey里边又调用了CheckSignUtil.get3desKey(context);

```
  public static native String get3desKey(Context context);
```

可以看到这是一个native方法，我们去hook一下，看看是不是固定的：

我们发现直接注入没反应，是因为在第一次调用之后就把值给了mDesKey，所以再使用43des加密时直接取mDesKey就可以了

这里我们去重新启动一下：

重新启动之后再注入也没有hook到，可能时机不对也可能是xml持久化存储了。我们先去换一下hook时机，使用spwan方式hook一下

使用spwan方式就hook住了，拿到了返回值：appapiche168comappapiche168comap

这个应该就是3des的key，我们去清除一下数据，然后再hook试试，还是这个值，那应该就是固定的，稍后去分析一下这个native函数

所以3des加密的key和iv都拿到了：

desede/CBC/PKCS5Padding加密

key为：appapiche168comappapiche168comap

i v为：appapich

接下里就要去看看这个明文是怎么来的，确定了明文后，密文也就有了：

hook 3des得到的明文：26c6d00d-cf45-302e-8f9d-77e4f49a5b49|392833184283903|424789

```
    getIMEI(context) +"|"+ System.nanoTime() + "|"+ SPUtils.getDeviceId()
```

是由这三部分得到的：

getIMEI(context)　　设备IMEI号

System.nanoTime()　　高精度计时，一般可以记录一段代码执行所用时间，这里应该就像是时间戳

SPUtils.getDeviceId()　　设备id

这三部分都可以改！

最后将加密的密文进行base64编码，就得到了我们最终的udid

username pwd

跟着堆栈，一路向上。找到了这里：

```
   public static void loginByPassword(String str, String str2, String str3,
ResponseCallback<UserBean> responseCallback) {
        HttpUtil.Builder builder = new HttpUtil.Builder();

 builder.tag(str).method(HttpUtil.Method.POST).signType(1).url(LOGIN_URL).param(
"username", str2).param("pwd", SecurityUtil.encodeMD5(str3));
        doRequest(builder, responseCallback, new TypeToken<BaseResult<UserBean>>
() {
        }.getType());
    }
```

这里是username是明文，pwd是MD5加密了的，然后进行了hex编码

请求头

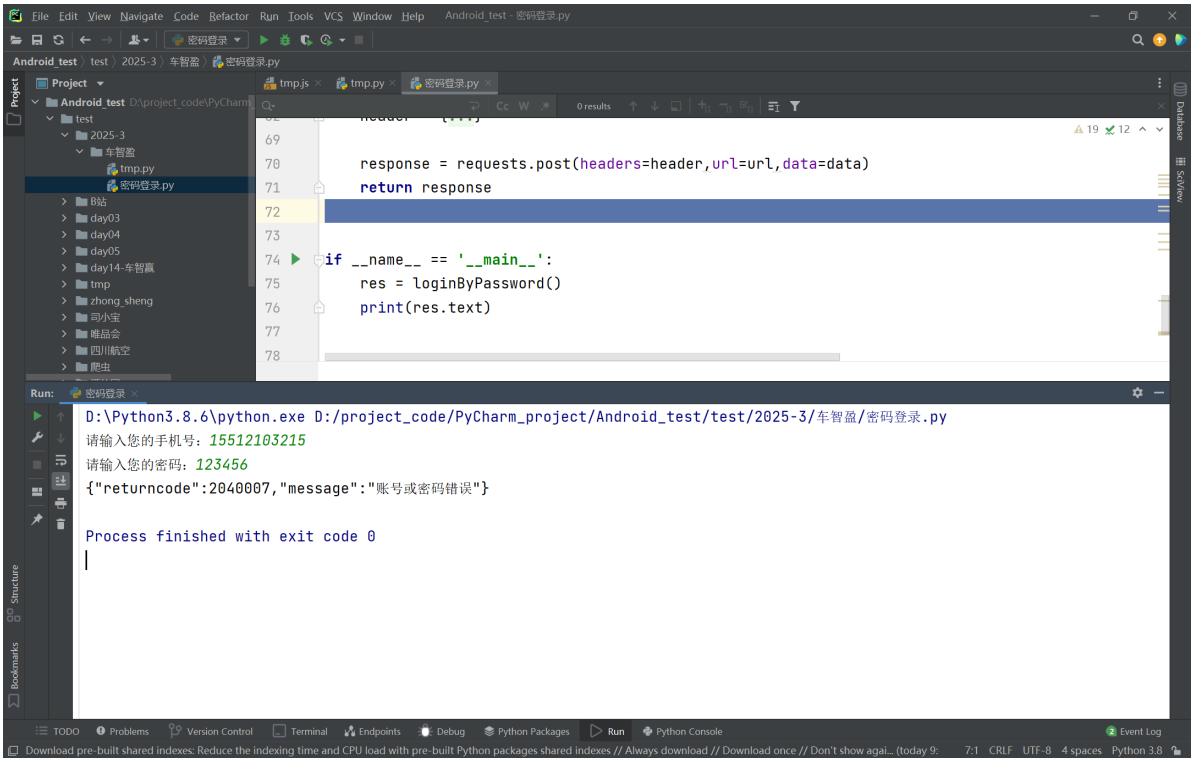
```
    cache-control       public, max-age=0
    traceid             atc.android_349d8c4f-2cee-43ef-a684-0e4d2d4b4f54
    content-type        application/x-www-form-urlencoded
    content-length      254
    accept-encoding     gzip
    user-agent          okhttp/3.14.9
```

只有traceid需要逆向

我们去搜索一下关键字

这个直接就是放的uuid，那请求的时候不带也可以，验证了一下，不带也可以发包

模拟登录




```python
import base64
import hashlib
import random
import uuid

import requests
from Crypto.Cipher import DES3
from Crypto.Util.Padding import pad

def getMd5(data):
    md5 = hashlib.md5()
    md5.update(data.encode('utf-8'))
    result = md5.hexdigest()
    return result
def desede_encrypt(plaintext, key, iv):
    # 确保 key 是 24 字节（192 位）
    key = key[:24].encode('utf-8')
    # 确保 iv 是 8 字节（64 位）
    iv = iv[:8].encode('utf-8')
    # 创建 DES3 加密器
    cipher = DES3.new(key, DES3.MODE_CBC, iv)
    # 填充明文
    padded_plaintext = pad(plaintext.encode('utf-8'), DES3.block_size)
    # 加密
    ciphertext = cipher.encrypt(padded_plaintext)
    # 进行 Base64 编码
    return base64_encode(ciphertext)
def getSign(data):
    result = getMd5(data)
    return  result.upper()
def base64_encode(data):
    if isinstance(data, str):
        data = data.encode('utf-8')
    encoded_bytes = base64.b64encode(data)
```

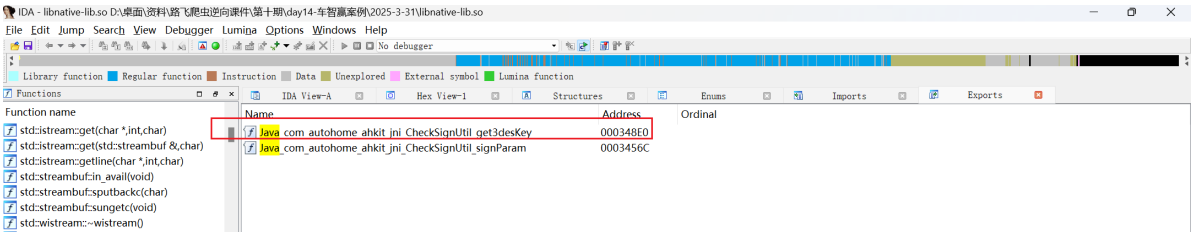
```
    return encoded_bytes.decode('utf-8')

def loginByPassword():
    key = "appapiche168comappapiche168comap"
    iv = "appapich"
    nanoTime = random.randint(388250309813510,513606633577399)
    udidPlaintext = f"26c6d00d-cf45-302e-8f9d-77e4f49a5b49|{nanoTime}|"
    _appid = "atc.android"
    appversion = "3.37.0"
    channelid = "csy"
    username = input("请输入您的手机号：")
    pwd = input("请输入您的密码：")
    pwdMd5 = getMd5(pwd)
    udid = desede_encrypt(udidPlaintext, key, iv)[0:60]+"
"+desede_encrypt(udidPlaintext, key, iv)[60:]
    signPlaintext =
f"W@oC!AH_6Ew1f6%8_appid{_appid}appversion{appversion}channelid{channelid}pwd{pw
dMd5}udid{udid}username{username}W@oC!AH_6Ew1f6%8"
    _sign = getSign(signPlaintext)
    headerUdid = "atc.android_" + str(uuid.uuid4())
    url =
"https://dealercloudapi.che168.com/tradercloud/sealed/login/login.ashx"
    data = {
        "_appid": "atc.android",
        "_sign": _sign,
        "appversion": appversion,
        "channelid": channelid,
        "pwd": pwdMd5.lower(),
        "udid": udid,
        "username": username
    }
    header = {
        "cache-control": "public, max-age=0",
        "traceid": headerUdid,
        "content-type": "application/x-www-form-urlencoded",
        "accept-encoding": "gzip",
        "user-agent": "okhttp/3.14.9"
    }

    response = requests.post(headers=header,url=url,data=data)
    return response


if __name__ == '__main__':
    res = loginByPassword()
    print(res.text)
```

获取3des key分析

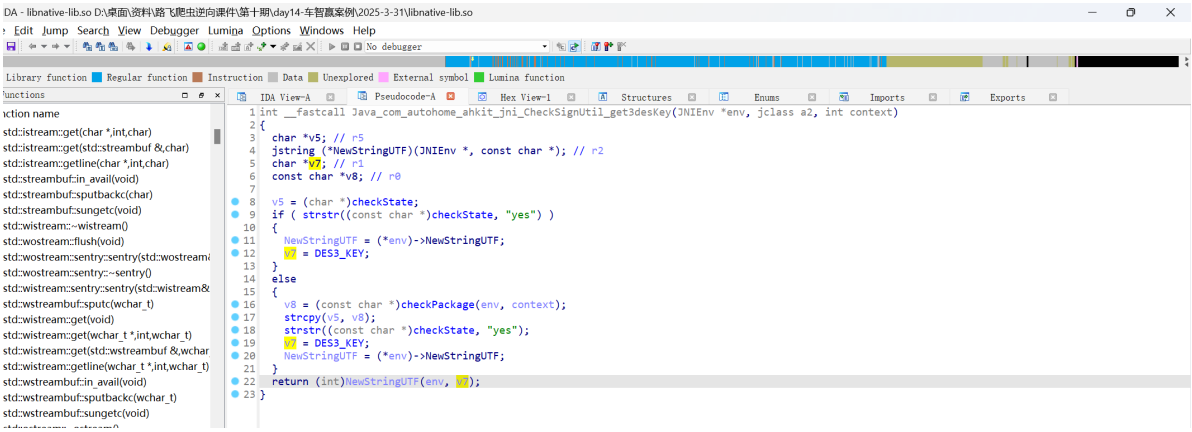
在破udid这个参数时，我们知道是用了3des加密，其中这个iv是直接给出的，而key是通过动态获取的，并且是在app启动的时候获取的，是通过一个native方法得到的：

```
public class CheckSignUtil {
    public static native String get3desKey(Context context);

    static {
        System.loadLibrary("native-lib");
    }
}
```

通过调用这个`get3desKey`得到的，我们去so层分析一下这个函数



静态注册的



进来F5反编译下，先看return，只有一个，说明没有其他分支

看到返回的是v7， 再去看v7是怎么来的

里边核心逻辑就是一个if-else判断，判断表达式是strstr，这个strstr是判断子串首次出现的位置，在这里就是主字符串是否包含子串，包含就返回地址，c语言中 非0就是真

在 C 语言中， `strstr` 是一个标准库函数，用于在字符串中查找子字符串的 **首次出现位置**。以下是其核心要点：

函数原型

c 复制

```c
#include <string.h>  // 必须包含头文件

char *strstr(const char *haystack, const char *needle);
```

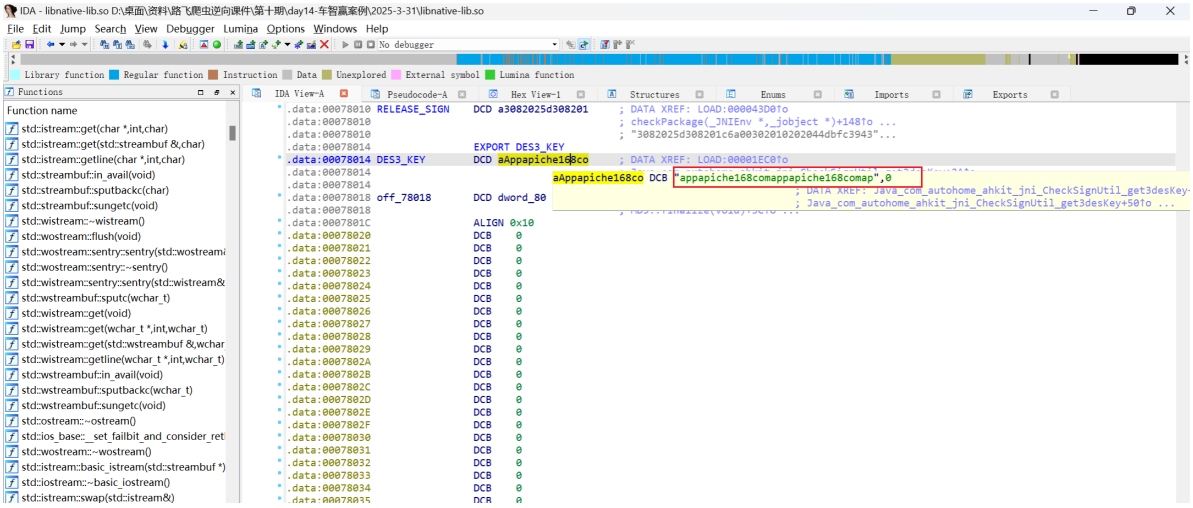
参数	说明
`haystack`	被搜索的主字符串
`needle`	要查找的目标子字符串

返回值：

- 找到时 → 返回指向首次出现位置的指针
- 未找到时 → 返回 `NULL`

但是我们可以看到，无论如何v7都是通过 v7 = DES3_KEY;得到的

我们再去看这个 DES3_KEY



他就是我们hook得到的那个key

这里也不是动态获取的，就是写死的，只不过是写到了so层