抓取登陆请求

POST [https://appapi.huazhu.com:8443//local/guest/Login/android/7@5@1/zh HTTP/1.1](https://appapi.huazhu.com:8443//local/guest/Login/android/7@5@1/zh)

X-Tingyun-Id: QFaFiyYOAwQ;c=2;r=1107668796;

X-Tingyun-Lib-Type-N-ST: 2;1531900932417

Content-Length: 1103

Content-Type: application/x-www-form-urlencoded

Host: appapi.huazhu.com:8443

Connection: Keep-Alive

User-Agent: HUAZHU/6.0 (Linux; Android LGE; Build/MRA58K)

Accept-Encoding: gzip

Accept-Charset: utf-8

data=vr%2BVqcFYQ%2BFuUy6FjwMaUX02ZYHnHTGQqT2LNs6q8lnU02ymMrLGGTZJ5hukj2r84JtL3aWvpXeOuD4YwrTViyCE0FEbWst0RGGZSfVITB95keW7gU6LRNg%2FlcXbASwJTe85ghqxhPr3NFVvbuog0zxqieIJy9UyM7%2BBLc8icH3BBa4fVO4vdpZUfAnxuQcBUQufSfuuBi2sameLWpoMZk%2BhCXuQI9sCutRdhr9fC9WplfYnWRGEpLS5XAg4Rfq7AGKvaNcK%2BhJrELwmR9redk61dAunxtc5dOUokO8Tomxy44FWyC2b3uuuhZ7I70z2AnYknFo2uyPeiA7jZs8tzKzmHQv4FqaIPZZxOm900ez%2BS%2BExq3W1AJFscuRC6t44D2vhz8L4TgFc5405pCy2XI0y16o7gf2la7hsEyMdmBLBfmpaTdDV3hDuhGX0Qi1EaedRlkiiXG%2BS2Rl%2B2tgirOMDSkRIMcpXRO%2B4T25Y6SOgniKaXw1cAVQGWkWNoSoH2vdHQ7MXUUTDgRk5VOxE6f0dg5vobO4eBICQ%2Fqq9zwPr9HxhJ%2FmPa4vAeGGpUJChlo%2F40yLXeIL1OK3vnIi3v3Xl8kMIpwpFUl9ni9FWjY5nEeG%2B8r%2FpWImGk%2FHMjNEfKCGscgoyPi9jAarN0eofROhPk456Z0UOeJxTbN6P0nHvR6Xsg1vLHV%2FZG4RuuZ4AgngQ1uAv8mlv2%2B3ZUud3KtRw3%2FKvJJJ3VP5dKOPmc2zGc2i3UPM42unRHAjRAF%2Bto%2Fw5mFe3dspZAkQ4nPM%3D&sign=LUKJeLdXM4IYGqaJ%2FxVNIQ%3D%3D&APPSIGN=AAjA8gAE4BEACZQxAArJaQAB1PUACMDyAAKx5AAG0AsABSjTAAEIqgAChJgABj8tAASmWgAGDcAA%0ABuKPAArJaQAIwcYABstVAAlpCQAFPh8AAoSYAAoaVgAK3TcACt03AAWOewAE6LcABoFJAAB5hgAH%0AjAMAA1BzAAOgdQAJI4wACSOMAAX%2FZQ%3D%3D%0A&time=20180718160212

HTTP/1.1 200 OK

Cache-Control: private

Content-Length: 131

Content-Type: application/octet-stream

Server: Microsoft-IIS/7.5

X-AspNetMvc-Version: 4.0

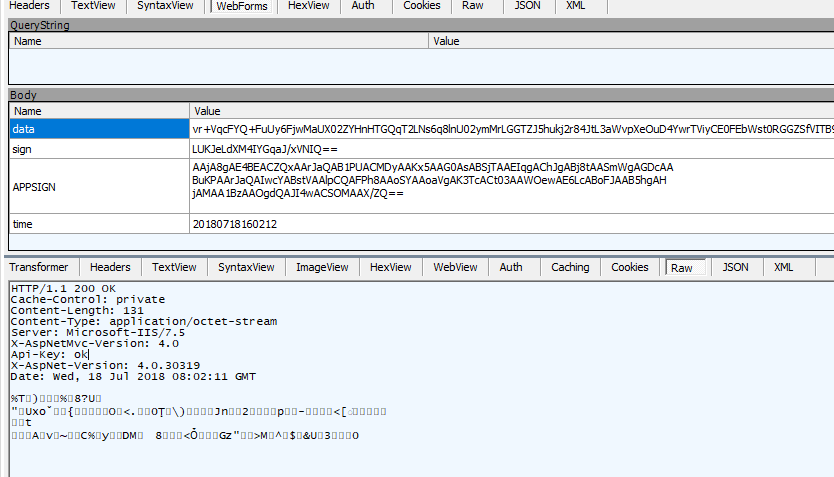
Api-Key: ok

X-AspNet-Version: 4.0.30319

Date: Wed, 18 Jul 2018 08:02:11 GMT

%T)%8?U\_  
"Uxoˇ{O<.0Ţ\)Jn2p-<[݁\_t

Av\_~C%yDM 8<ȰGz">M^$\_&U3\_0

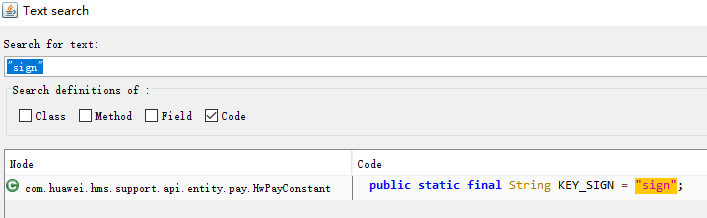


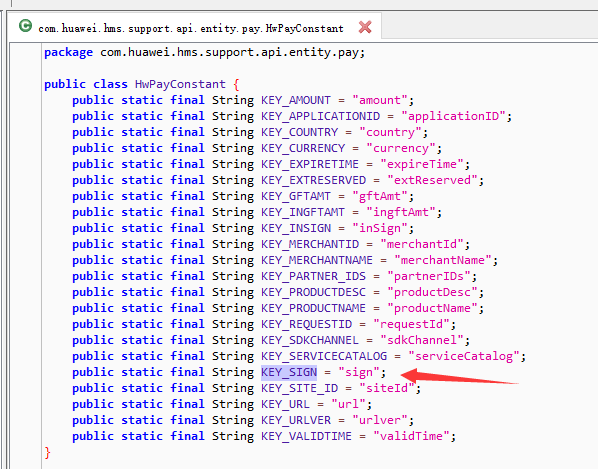
请求正文中，一共4个字段

Data, sign, APPSIGN, time

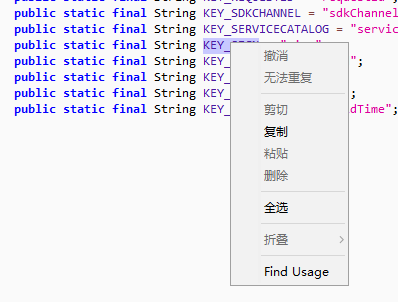
App没壳，直接jadx反编译

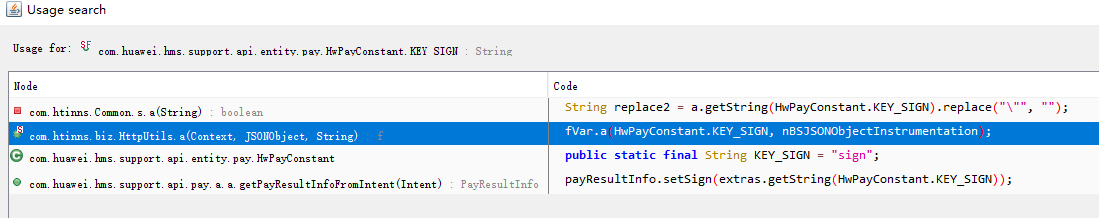
就搜下sign吧





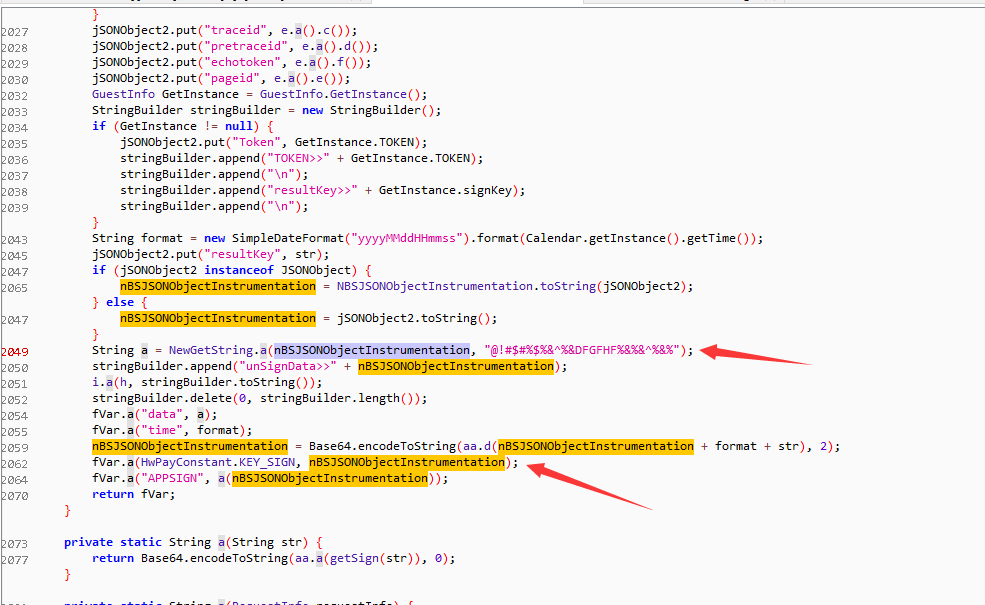
右击，选择查看使用





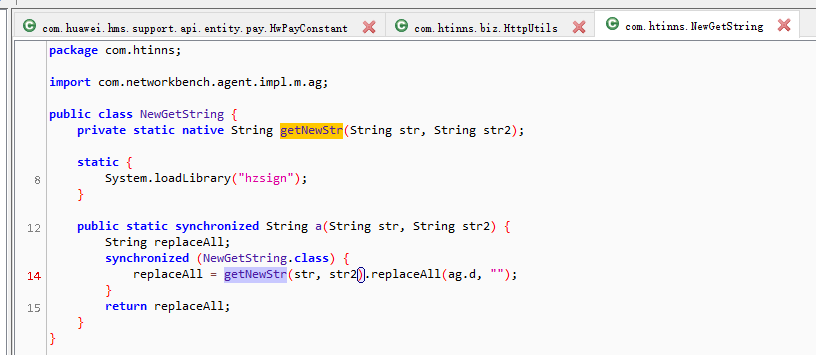
一共有4处引用，由于使用的是http请求，选第二个看看

直接来到了这里



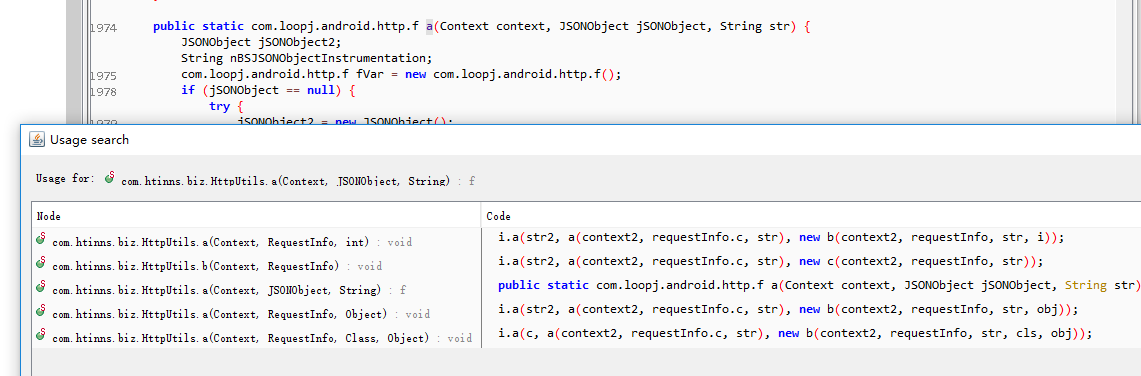
可以看到，下面箭头处是构建map

上面箭头处NewGetString.a 是调用了生成sign密文的方法，跟过去看一眼



可以看出，它进入了这个so库hzsign，的getNewStr方法

好了，这里要求是不去具体分析它的加密流程，我们回到上上图的方法处，查看它的引用



一共有4处引用，随便挑第一个过去看看



来到了这个类的上面这个a方法中，红框处就是调用的构建map的方法

可见，这个i.a方法，一共有三个参数

Str2,构建map的a方法返回值，后面还有一个b对象

双击str2,可看到它是在哪些地方构造的



下面有一句提示没有网络，说明它在这里不会去做网络请求，那么，上面的if语句代码块中，return上面一句，猜测就应该做网络请求的



到这里，根据安卓的编写习惯，猜测new b(context2, requestInfo, str, i)就是请求之后的回调函数了



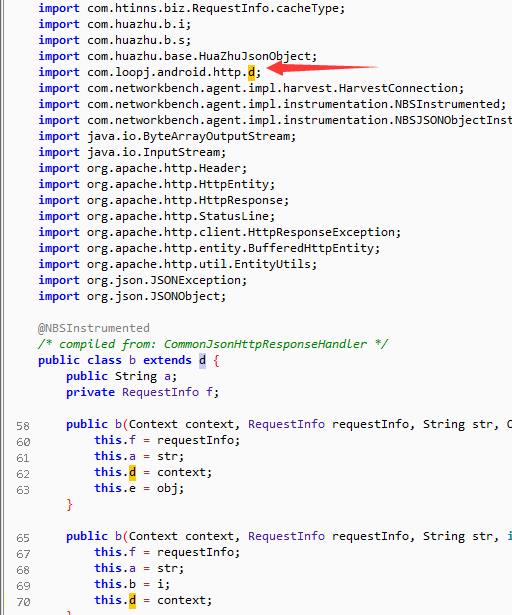
按住Ctrl键，鼠标左键点b,跟过去看看b这个类



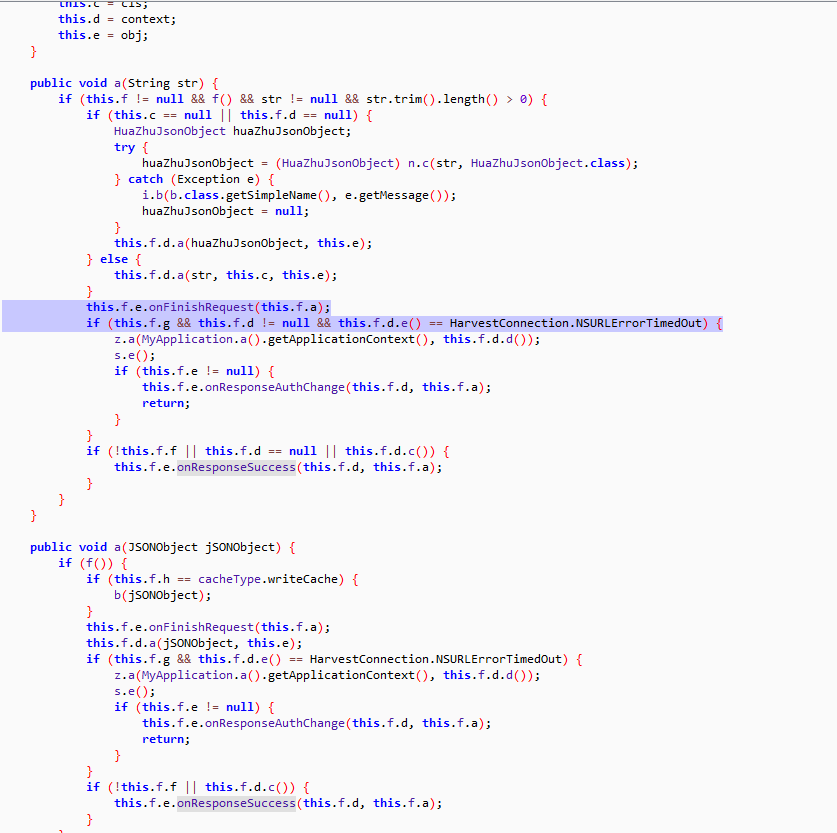
根据上上图，new b 时候带入的参数，可以判断它调用的是第二个构造函数

把相应的数据写入了类成员变量中

看到这个b类是继承自d类，看看d类，就更加坚定，这是http请求的回调函数



好了，既然这个b类是回调方法所在的类，那我们就继续往下看看



好吧，这方法，都是对请求状态的判断，以及相应的处理方法

继续往下，到了这个方法，看到它带入的是一个HttpResponse，就是http请求返回对象，这个对象，封装了，http请求返回的所有内容，而我们抓请求的时候，看到它返回的是

Content-Type: application/octet-stream

octet-stream数据流的形式，看来这里有料，



StatusLine statusLine = httpResponse.getStatusLine();读取返回的状态码

HttpEntity entity = httpResponse.getEntity();得到返回的Entity

Header firstHeader = httpResponse.getFirstHeader("Api-Key");读取apiKey

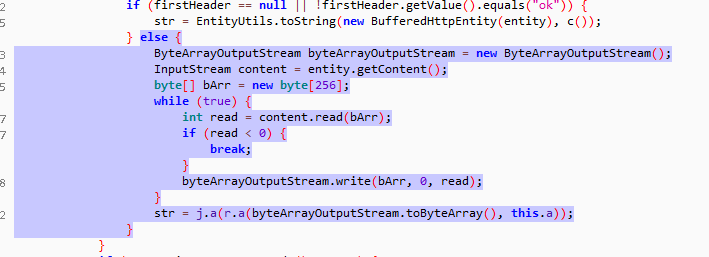
httpResponse.getFirstHeader("Content-Encoding");读取编码方式

if (firstHeader == null || !firstHeader.getValue().equals("ok")) {

str = EntityUtils.toString(new BufferedHttpEntity(entity), c());｝

如果返回的协议头为空，或者状态码不为OK,执行后面这一句代码

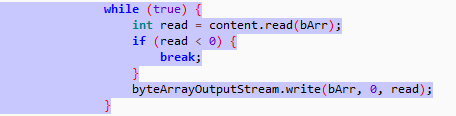
那么else语句中就是重头戏了？我们继续看



ByteArrayOutputStream byteArrayOutputStream = new ByteArrayOutputStream();new 一个output字节流

InputStream content = entity.getContent();拿到上下文

byte[] bArr = new byte[256];构建一个256长度的缓冲字节数组



读取数据，存放到byteArrayOutputStream中

最后，byteArrayOutputStream进入了这里

str = j.a(r.a(byteArrayOutputStream.toByteArray(), this.a));

先看看r.a方法所在的类

package com.htinns.Common;

import com.umetrip.umesdk.helper.ConstNet;

/\* compiled from: RC4Factory \*/

public class r {

public static byte[] a(byte[] bArr, String str) {

if (bArr == null || str == null) {

return null;

}

return b(bArr, str);

}

public static byte[] a(String str, String str2) {

if (str == null || str2 == null) {

return null;

}

return b(str.getBytes(), str2);

}

public static String b(String str, String str2) {

if (str == null || str2 == null) {

return null;

}

return c.b(a(str, str2), false);

}

private static byte[] a(String str) {

int i;

int i2 = 0;

byte[] bytes = str.getBytes();

byte[] bArr = new byte[256];

for (i = 0; i < 256; i++) {

bArr[i] = (byte) i;

}

if (bytes == null || bytes.length == 0) {

return null;

}

i = 0;

int i3 = 0;

while (i2 < 256) {

i = (i + ((bytes[i3] & ConstNet.REQ\_GetCheckinURL) + (bArr[i2] & ConstNet.REQ\_GetCheckinURL))) & ConstNet.REQ\_GetCheckinURL;

byte b = bArr[i2];

bArr[i2] = bArr[i];

bArr[i] = b;

i3 = (i3 + 1) % bytes.length;

i2++;

}

return bArr;

}

private static byte[] b(byte[] bArr, String str) {

if (bArr == null) {

return null;

}

byte[] a = a(str);

byte[] bArr2 = new byte[bArr.length];

int i = 0;

int i2 = 0;

for (int i3 = 0; i3 < bArr.length; i3++) {

int i4;

byte b;

i2 = (i2 + 1) & ConstNet.REQ\_GetCheckinURL;

if (a != null) {

i4 = a[i2];

} else {

i4 = 0;

}

i = (i + (i4 & ConstNet.REQ\_GetCheckinURL)) & ConstNet.REQ\_GetCheckinURL;

if (a != null) {

b = a[i2];

} else {

b = (byte) 0;

}

if (a != null) {

a[i2] = a[i];

a[i] = b;

i4 = ((a[i2] & ConstNet.REQ\_GetCheckinURL) + (a[i] & ConstNet.REQ\_GetCheckinURL)) & ConstNet.REQ\_GetCheckinURL;

bArr2[i3] = (byte) (a[i4] ^ bArr[i3]);

}

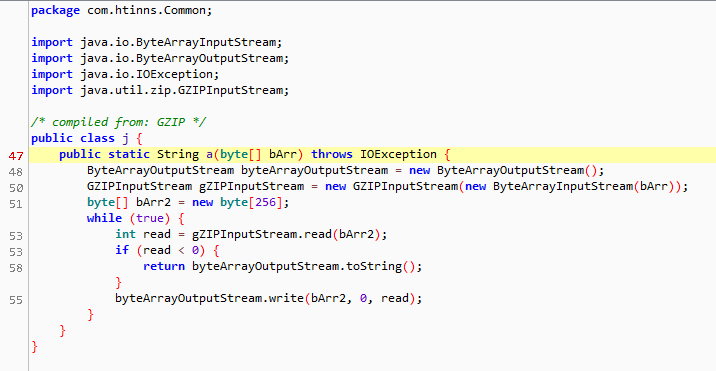
}

return bArr2;

}

}

再看看外层的j.a方法

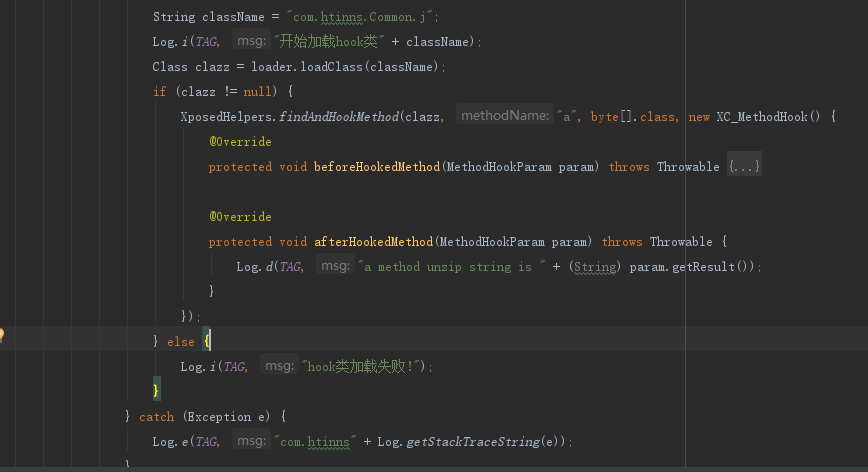


可以看出内层的r.a对byteArrayOutputStream进行了一系列处理，然后把结果带入了j.a方法

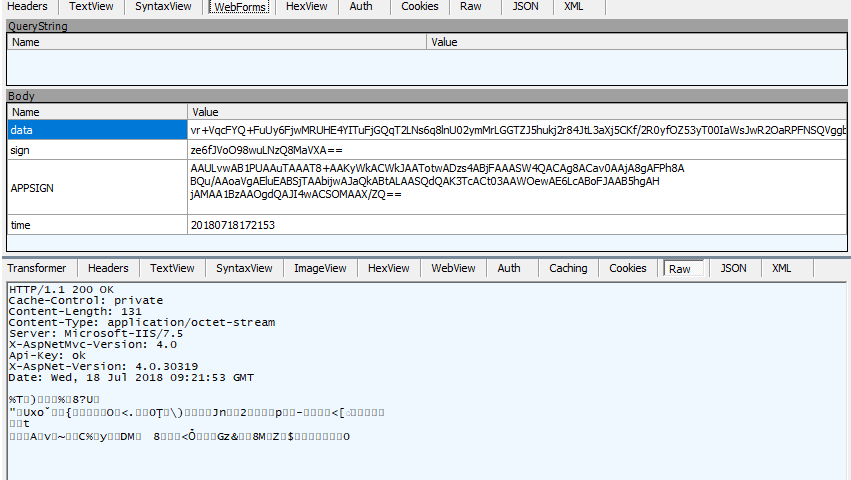
这个j.a静态方法，可以看出是gzip算法

通过这个算法出来的数据就是String类型的明文了，

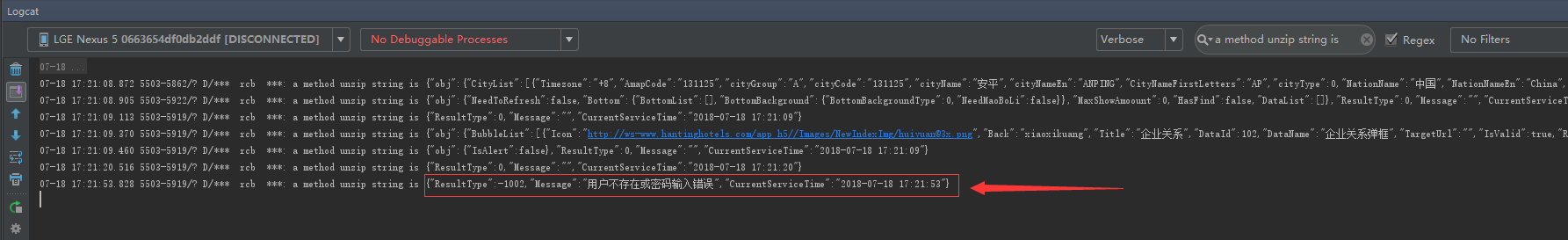
它再通过其它方法，对这些消息进行提示或者什么的，就不管它了

这里就不去调试j.a方法了，直接写插件hook吧，检验一下分析是否正确

生成apk，安装重启手机，打开app，随意输入一个帐号密码登陆，



这时候，用as查看一下日志



可见分析正确，除了登陆这条请求返回的提示信息，打开app其它请求的返回信息也一并打印出来了。至此，分析完毕。

总结下它返回的字节流解密流程

1. 把字节流带入了com.htinns.Common.r.a方法
2. 把第一步的结果带入gzip(com.htinns.Common.j.a方法)解压

调用处在com.htinns.biz.b类的a方法中

