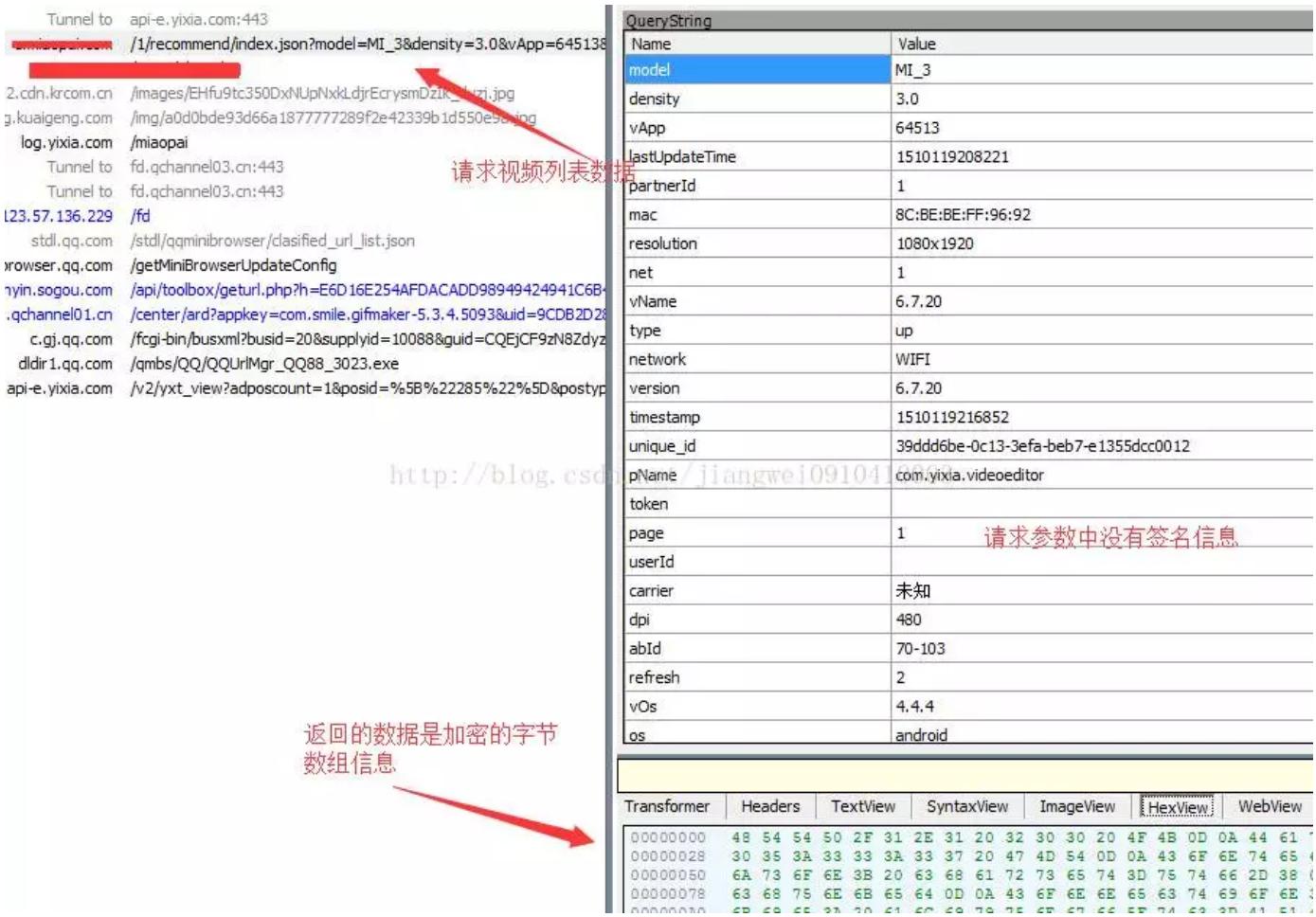
Android逆向之旅---Android中分析某短视频的数据请求加密协议(IDA动态调试SO)第二篇

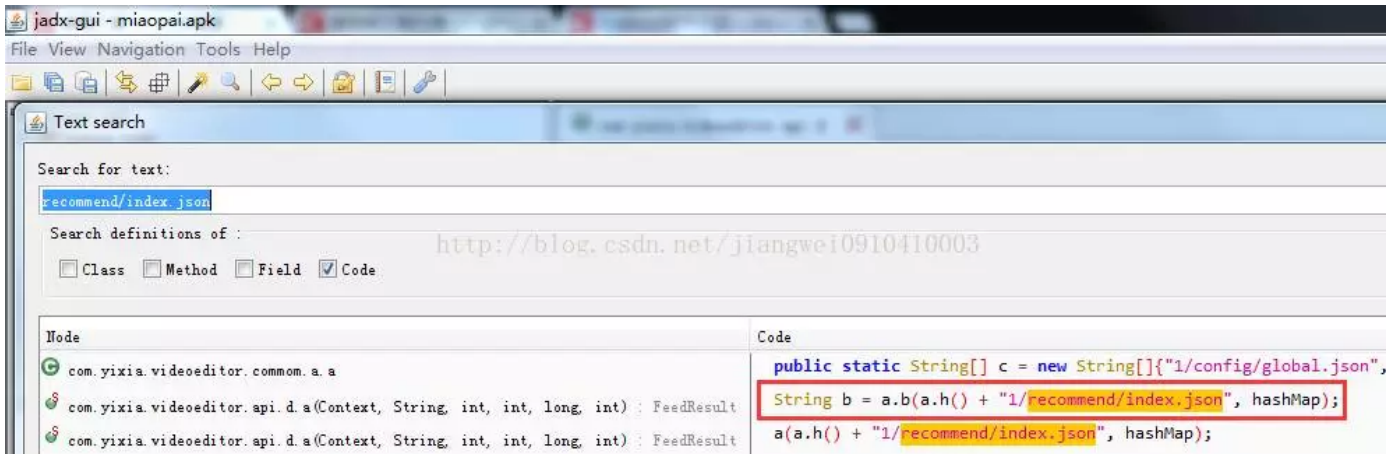
原创 2018-01-02 赵四 编码美丽

一、逆向分析

年前必须搞定短视频四小龙，之前已经搞定某音和某山小视频了，那么今天继续来搞下一个，在之前一篇文章中已经详细分析了不了解的同学，可以点击详情：某音和某山小视频的数据请求加密协议，通过IDA动态调试so来解决一些问题，本文我们继续来看另外一个短视频的数据请求加密协议，这个就是传说中的某拍视频，不多说找到突破口还是抓包：



我们通过下拉一次数据，看到这个请求url发现请求中的参数信息没有携带签名信息，但是返回的数据确实加密的字节数组了。所以得先搞定这个字节数组了。直接用Jadx打开应用，然后全局搜索url字符串信息：



很容易就找到了这个，直接点击进入即可：

这里看到调用一个方法之后拿到字符串就开始直接解析json了，看看这个返回字符串方法：

看到，Jadx中解析失败，不过没关系，还是smali代码，大致能看懂，继续往下看：

这里需要你用过okhttp框架了，一看就知道这里用到这个框架，而且最终通过bytes来获取字节数组，我们看到这里又调用了一个方法，然后直接返回字符串了，去查看这个方法实现：

继续往下看，应该是个解密方法：

看到，这里依然把加密工作放到了native层了，我们操作依然很简单，直接把这个libte.so拷贝到我们的demo工程中，然后构造这个native类，直接调用native方法进行解密即可：

我们通过之前抓包看到，他的请求参数都是一些常规的信息，没有签名信息，为了简单，这里直接把这些参数拷贝出来写死利用okhttp框架进行数据请求：

当然这些参数后续肯定需要优化，实现动态获取最靠谱：

拿到请求之后的字节数据之后，然后调用native方法进行解密，我们直接运行看看日志信息：

可惜的是，发现了尽然解密失败。那么就要怀疑so中是否有判断逻辑了，直接使用IDA打开so查看：

直接F5查看对应的C代码，看到一个tinydecode函数的返回值，然后有一个判断，进入这个函数看看：

这里调用了strstr系统函数，这个函数主要用来判断第二个参数是不是第一个参数的字串，如果是就返回字串的指针，如果不是就返回空指针NULL；看到这里有个包名字段值，感觉应该和包名有关系，双击这个g_packagename字段，然后点击X键，查看调用的地方：

在JNI_OnLoad中进行赋值的，依然查看JNI_OnLoad函数代码：

这里开始进行赋值了，看看上面这里的sub_43B0函数怎么获取字符串信息的：

好吧，尽然是通过读取系统的这个文件来获取包名值，而不是通过全局的Context变量了。这个文件是很奇特的，只要在本应用中读这个文件就是当前应用包名，而是用命令行去查看这个文件是没任何内容的。这个知识点大家就记住一下就好了。

那么这个到底用当前应用包名和哪个包名进行比较呢？看上面的strstr函数的第二个参数是：

依然双击这个变量，然后点击X键查看赋值地方：

还是在JNI_OnLoad函数中，点击进入赋值代码：

看到一个特别的字符串信息和一个循环指令，可以猜想应该是通过这个字符串信息来获取最后的信息赋值给g_me变量，其实这里我们可以才想到这个值应该是应用的包名：com.yixia.xxxxxx；这样就可以理解为只有当前应用的包名正确才能正确调用逻辑，这个是一层简单的防护而已。就是为了防止自己的so文件被其他恶意程序调用的。

二、IDA调试so

不过这里都是猜想，为了验证，可以动态调试so来验证。而调试so步骤这里不在详细多说了，在我写的大黄书**Android应用安全防护和逆向分析**一书中已经很详细介绍了，这里直接上手操作：

第一步：运行android_server开始监听

第二步：端口转发，以debug模式启动应用

第三步：打开IDA进行进程附加，记得勾选上JNI_OnLoad函数挂起状态

第四步：使用DDMS查看调试端口，然后启动调试器链接

一定要记得正确的端口号，不然链接失败报错的：

第五步：IDA中点击运行开始调试

不过这里为了保险起见，在运行之前，再去查看有么有成功挂起JNI_OnLoad函数设置：

如果没有，就在此勾选上：

然后在点击运行，当然也可以直接使用F9键：

运行成功之后，会发现，jdb也连接成功：

而且DDMS中的红色小蜘蛛变成绿色的了：

运行成功之后，因为程序会家在很多系统的so文件，而我们又在JNI_OnLoad挂起了，所以会弹出很多确定对话框，都不用管直接点击OK即可，直到把系统so文件全部加载完毕：

然后，这时候就会出现我们demo中的按钮界面了，我们直接点击按钮进行加载libte.so文件：

点击运行，加载需要调试的so文件，然后在右侧栏找到JNI_OnLoad和解密函数下断点：

先找到指定so文件，然后双击在查找函数：

点击进入函数处下断点：

接着给加密函数下断点：

点击进入函数代码处即可：

然后这时候，就可以开始运行了，运行到断点处，依次往下走，这里发现JNI_OnLoad中并没有反调试逻辑，直接略过，来到解密函数，F7单步往下走，来到了tinydecode函数处，下个断点：

然后进入函数，之前静态分析这个函数，他内部有一个字符串字串判断函数：

一般都会在CMP处下个断点，继续往下走，来到strstr函数内部，看看两个字符串比较的值是多少：

看到，获取到本应用的包名就是我们的demo应用，继续看另外一个字符串信息：

看到了这个包名的确是应用的"com.yixia.xxxxxx"，因为再之前下了CMP判断断点，现在明显不是字串，返回NULL了，所以寄存器中的值肯定是0了，为了后续能在正确的逻辑，直接修改寄存器值：

修改寄存器值很简单，直接在右侧栏右击进行修改即可。修改成非0值即可，一般就修改成1了，走了正确的逻辑了，可惜的是，在日志中看到解密还是失败的。

三、修改指令

但是到这里，我们已经可以确认一件事就是so中的解密函数逻辑有一层防护就是判断当前调用so的应用包名是否为正确的应用包名，如果不是就不走正确的解密逻辑了。所以这里我们**需要修改一下so指令，让这个判断无效。修改指令其实很简单，我们看到他通过判断strstr函数返回的NULL值，也就是对应 CMP R0,#0 指令值，然后后面有一个BEQ跳转指令，这里我们可以这么改，他不是和0判断吗？其实0就是NULL值，我们把他改成和1比较，这样strstr函数返回了NULL值也就是0，和1比较不相等。那就正确的逻辑了**。这个思路大家要搞清楚，其实也没这么复杂，反正就一个目的，不要走后面的BEQ逻辑就对了。和1比较肯定就不会走BEQ逻辑了。好了下面就来开始修改指令了，这个网上有一个小工具可以简单修改，但我发现了一个更好的在线修改地址：**http://armconverter.com/**；网站打开有点慢，等待一下即可，我们为了验证这个网站准确，我们先输入 CMP R0,#0 这条指令，看看对应的十六进制值和so中的值是否对应：

看到这里有很多架构的对应这个指令的值，先不着急我们去IDA中查看这条指令对应的值：

这里为了后面修改指令方便，借助010Editor工具进行操作：

010Editor工具有两个常用的快捷键，**一个是Ctrl+F全局查找十六进制值，一个是Ctrl+G跳转到指定的地址**。这里我们跳转到5BB0地址处：

看到这里的值是：000050E3值，和上面的转化的arm架构值对应的。那么下面就来修改指令，比较简单，直接修改为：**CMP R0,#0 ==> CMP R0,#1**

对应的十六进制值就是010050E3了，直接去010Editor工具中进行修改：

就这样修改成功了，然后保存，再次用IDA打开修改之后的so文件，看看是否修改成功了：

这时候，我们在用F5键查看他的代码：

看到，的确有效果了，这时候strstr返回NULL了，和1进行比较显然不相等就开始走下面的正确解密逻辑了。修改成功了，到这里有的同学好奇，是否可以直接修改后面的BEQ指令为BNE呢？当然

是可以的，这个方式后面继续介绍，因为我写文章的目的就在于能介绍技术都给介绍，多条路始终是好事。

四、逆向分析签名信息

然后我们把修改之后的so文件拷贝到工程中，再次运行，其实这个结果还是不可以的，因为我们在上面的调试及时过了CMP也是不行的，所以这时候就要猜想了还要哪里有问题呢？当然找问题还得去那块请求数据的smali代码处：

开始的分析okhttp请求代码处忽略了这个地方，去查看这个类：

到了这个内部类中发现了很多关键信息，最重要的莫过于这个UA，请求头信息。而这里有很多信息，还包括了签名信息，继续往下看：

好吧，这下已经肯定就是把签名信息放到了请求头中了，这招也是够狠的，一般人还很难发现，再回到Fiddler抓包看看请求头中的具体信息：

果然在头部中有这几个信息，通过分析可以发现，除了sign字段其他的值可以暂时写死都是表示唯一的，后续需要搞定那个sessionid值。这里先不管写死。然后就来关键看看sign签名字段值怎么来的：

依然调用上面的native方法的，这里为了搞清楚参数值，直接启动hook大法打印参数值即可，没必要分析代码了：

需要注意的是，应用进行拆包了，所以为了确保hook成功，先hook系统的Application类的attach方法拿到正确的类加载器，这个方法已经在很多文章中都介绍了，这里不多说了，直接运行看日志：

看到参数再结合上面的代码可以看到，大致是应用版本号、UUID、时间戳、请求url的path值。这里我们可以把前两个值写死，后面两个值获取即可：

这里的URL就是请求视频列表数据的：

然后我们把上面的头部信息设置到okhttp中即可：

然后运行看看效果，抓包看到头部信息已经设置成功了，看看返回的数据：

看到的确有结果了，但是貌似是错误信息，直接去转码这个Unicode值：

看到，提示是签名校验失败，也就是上面的头部信息中的签名值是错误的。说明那个native函数签名有问题，继续用IDA打开so文件进行查看即可：

这里依然有一个sign函数，获取v18值，到下面进行比较逻辑，进入sign函数看看：

五、最终解决方案

果然这里还是进行判断当前应用是否为应用包名，所以我们需要修改指令了，**这一次不修改该CMP比较的值了，而是修改跳转指令，直接把BEQ改成BNE即可，BEQ是等于跳转，BNE是不等于跳转**。这样改了之后就不在乎上面的CMP指令了，修改指令还是很值钱一样：

然后去010Editor中进行查看十六进制值：

然后再去上面的那个转化网站查看：

可以看到BEQ的十六进制值最后两位是0A，那么改成BNE之后是多少：

变化的就是0A变成了1A值，那么简单了，直接修改010Editor中的0A值即可：

其他的地方都不要动，保存即可，然后再用IDA打开修改之后的so文件，确认是否修改成功：

看到了，果然把BEQ修改成了BNE了，查看代码：

这样就可以了，strstr返回的肯定是NULL也就是0，那么非0就是true了，走了下面的解密逻辑了。
好了，把修改之后的so文件拷贝到我们的demo工程中，再次运行：

这样终于有结果了，我们把这json格式化看看：

好了到这里，我们就成功的拿到了应用视频的数据了。这里遇到的问题和上一篇的某音可以说完全不一样，虽然都有签名信息。下面就来总结一下。

六、知识点总结

本文分析了某拍视频获取数据信息的加密协议，可以看到其实遇到的问题不算难，但是我们依然可以了解到很多技术和知识：

- **第一、**签名信息永远都不过时，可以放在请求字段中，也可以尝试放到请求头中。
- **第二、**对于so防止被别人恶意调用，可以判断是否为当前正确的应用包名信息。
- **第三、**在native层可以直接读取系统文件/proc/self/cmdline值获取应用包名。
- **第四、**在native层遇到判断逻辑，修改一般就两种方案：一种是直接修改BEQ指令为BNE，一种是修改CMP指令比较的值，一般是把0改成1即可。

可以看到某拍的so防护比之前的某音差了一点，至少加点签名校验，反调试等基础判断，然而并没有。

严重说明

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

七、总结

本文在上一篇分析完某音和某山小视频协议解密之后，在此分析某拍视频的，之前我说过这次一定要搞定短视频四小龙，那么下一个大家应该猜到是谁了。当然大家也很好奇为什么我非要死磕短视频呢？因为我有一个大大的计划和项目要启动。等下一篇搞定最后一个短视频协议之后，就告诉大家我到底要干嘛？本文虽然难度低了，**但是为了讲解详细，给大家带来更多的知识，截取了很多图片注释，很累很累的。所以大家看完文章之后觉得有收获就点赞分享，让更多人爱上逆向！**

阅读原文