**AirBuddy 2.6.3 破解分析**

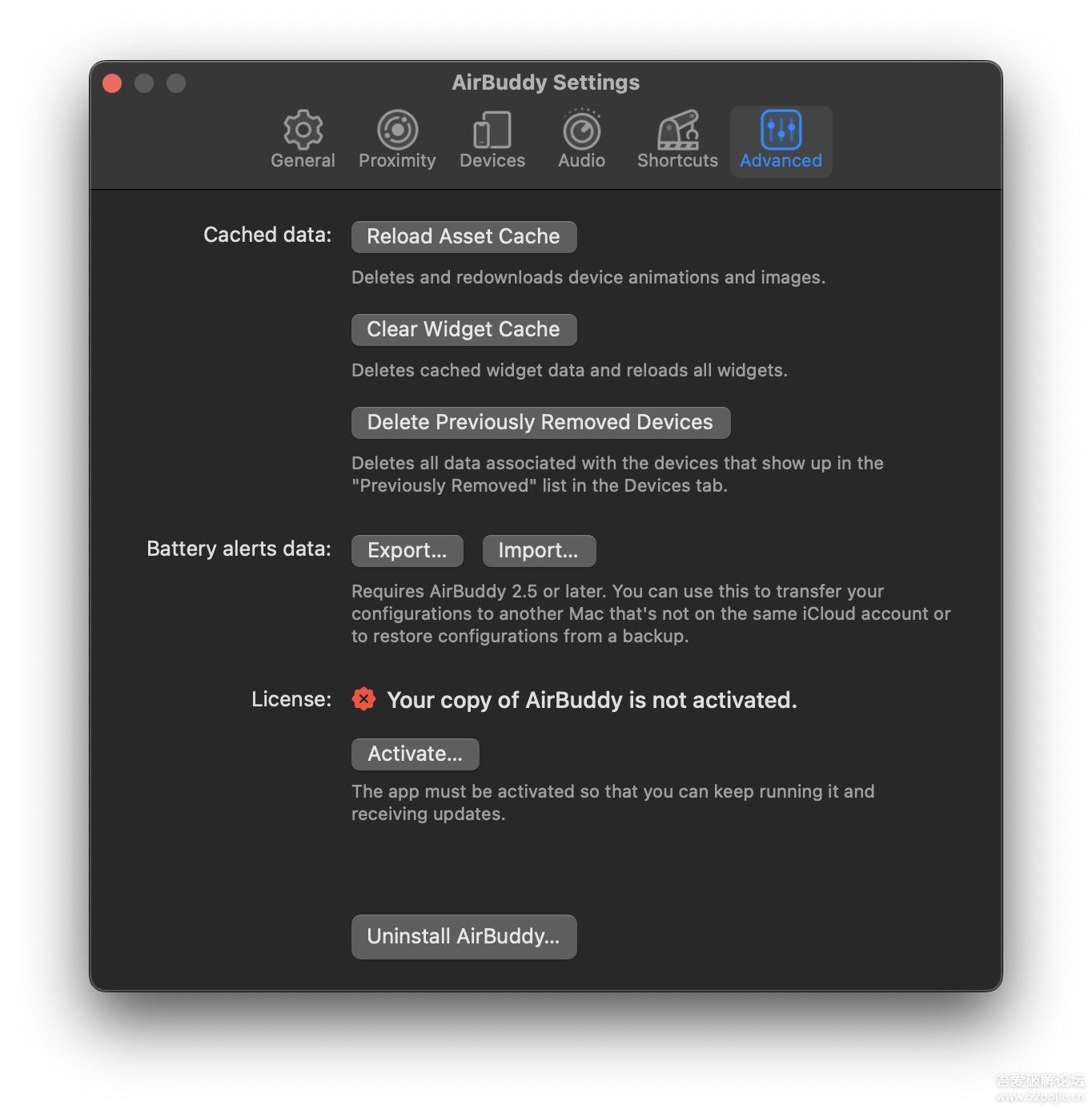
**浅析AirBuddy 2.6.3 (602)**

**0x00 版规限制**

论坛真不让上传商业软件pj成品, 这里只发学习版的二进制文件。  
可以学习但不要传播。  
伸手党能帮你的只能到这里了，别问为什么没有成品。请各位支持正版！  
你问我支不支持正版？你问我资瓷不资瓷那我肯定还是资瓷的。

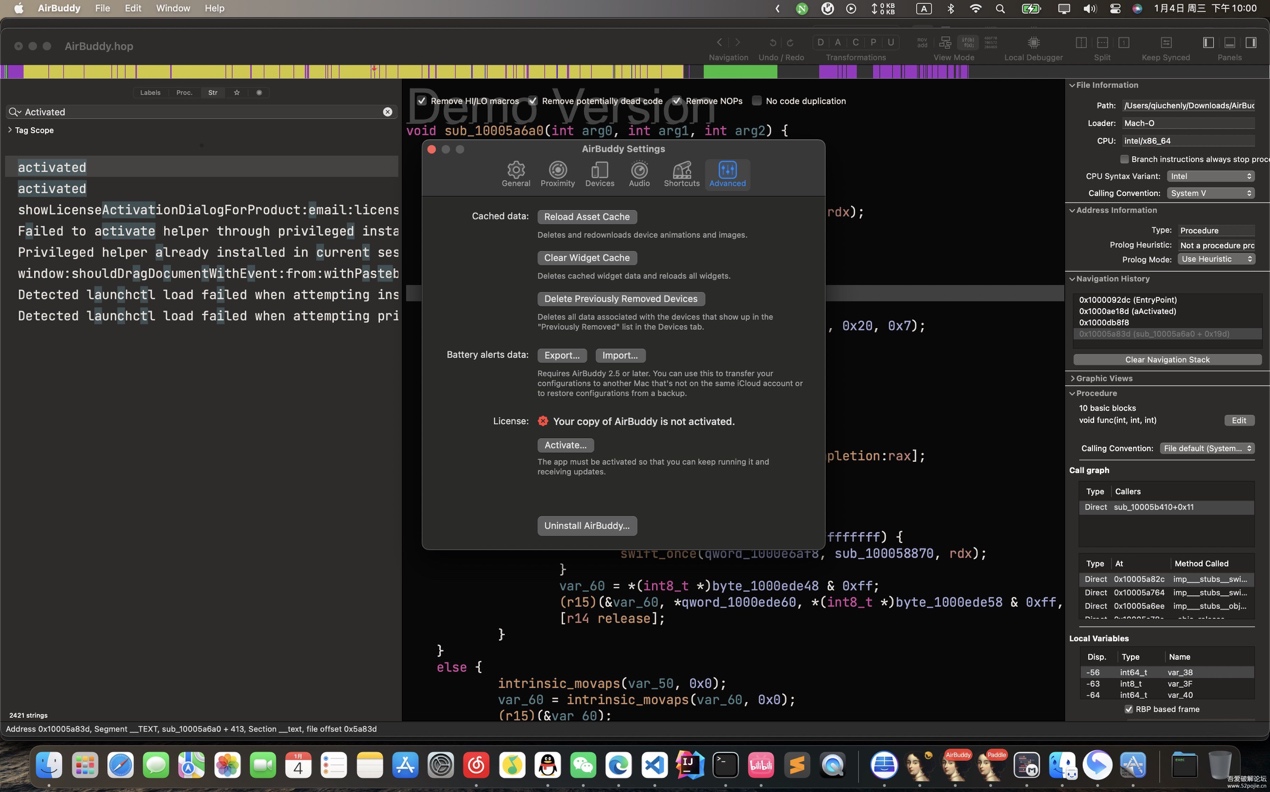
声明: 版禁果，然皆附。故贴此，助正宜。  
 [Paddle.zip](https://www.52pojie.cn/forum.php?mod=attachment&aid=MjU4MzAxN3w3ZmFkMjk1N3wxNjc2NzczMTkwfDE0OTczOTl8MTczMzE0NA%3D%3D) *(522.71 KB, 下载次数: 17)*

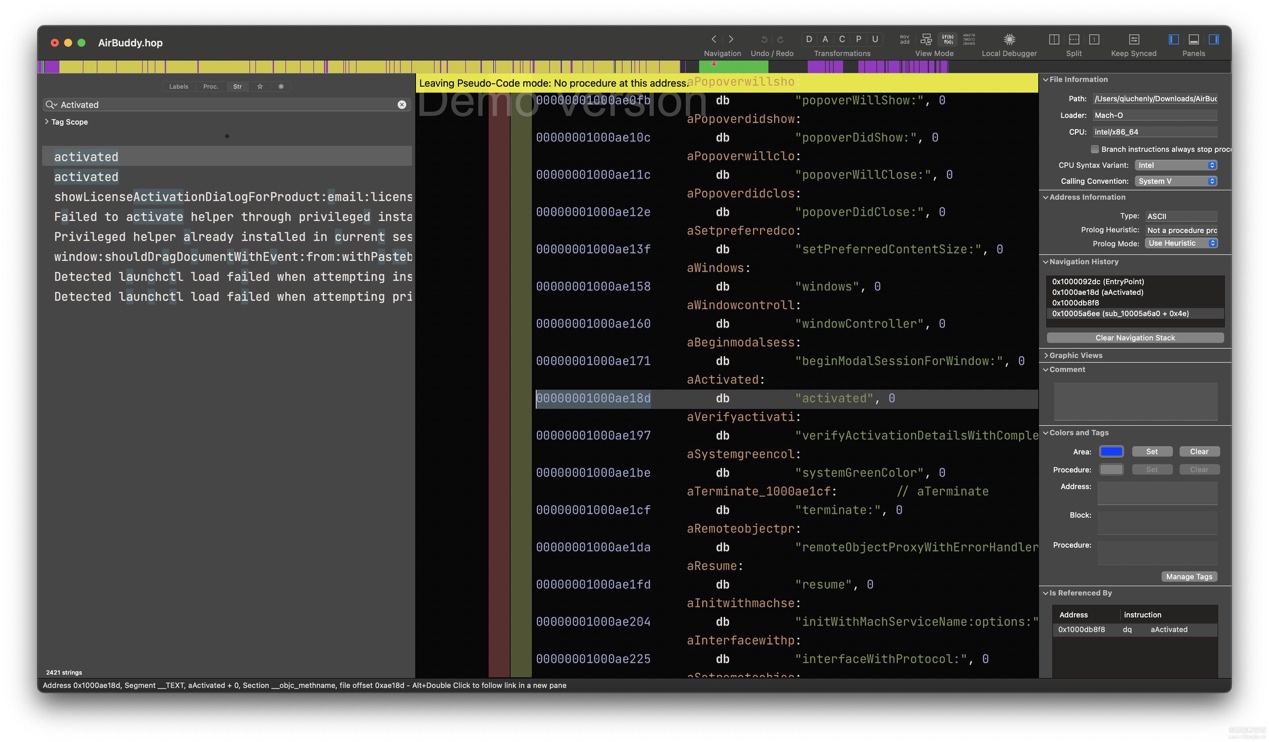
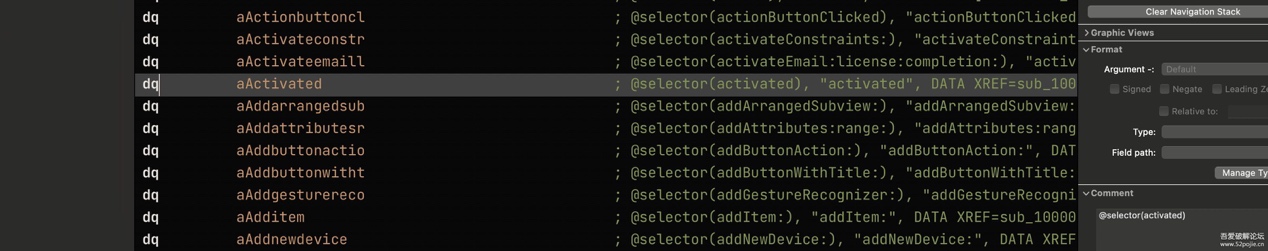
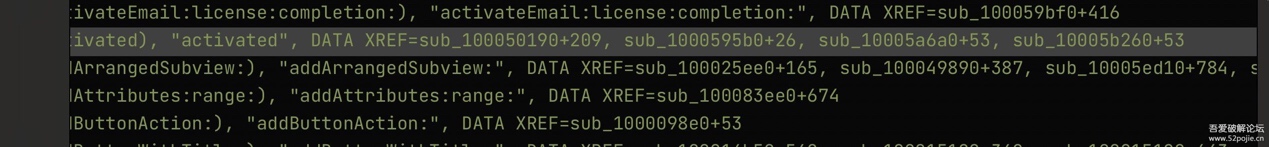
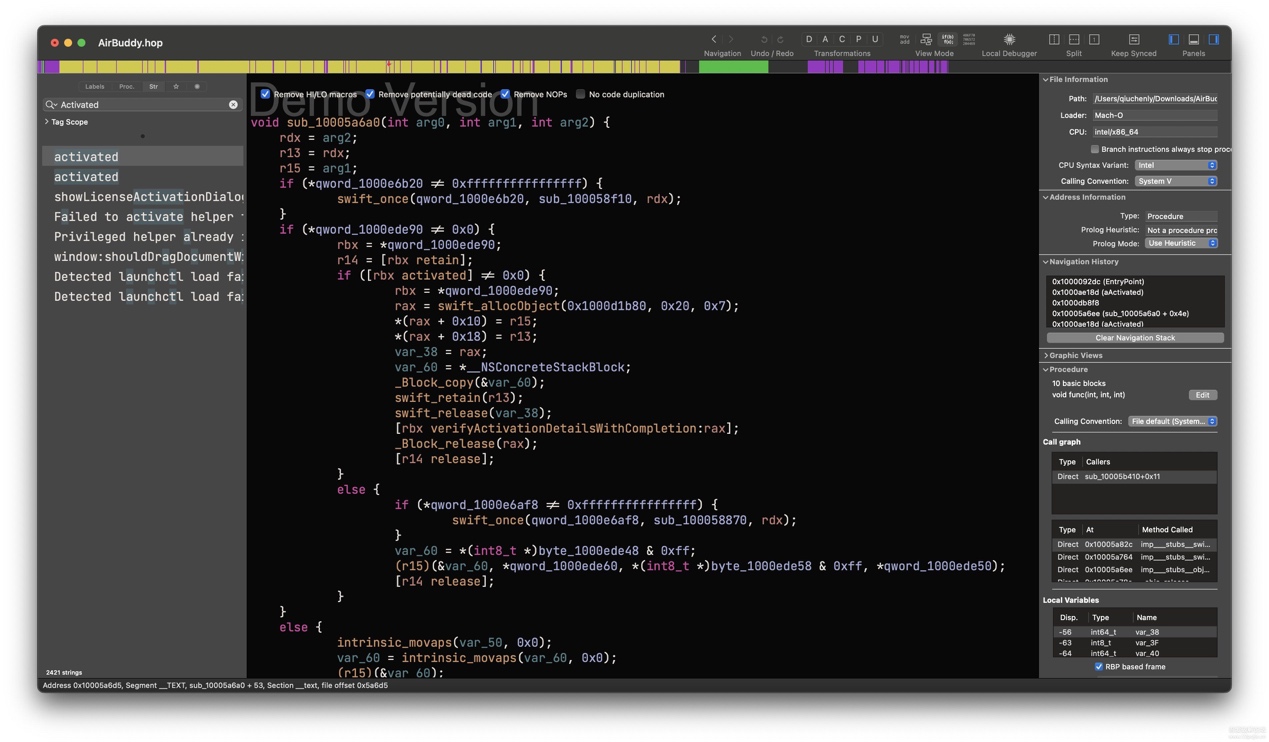
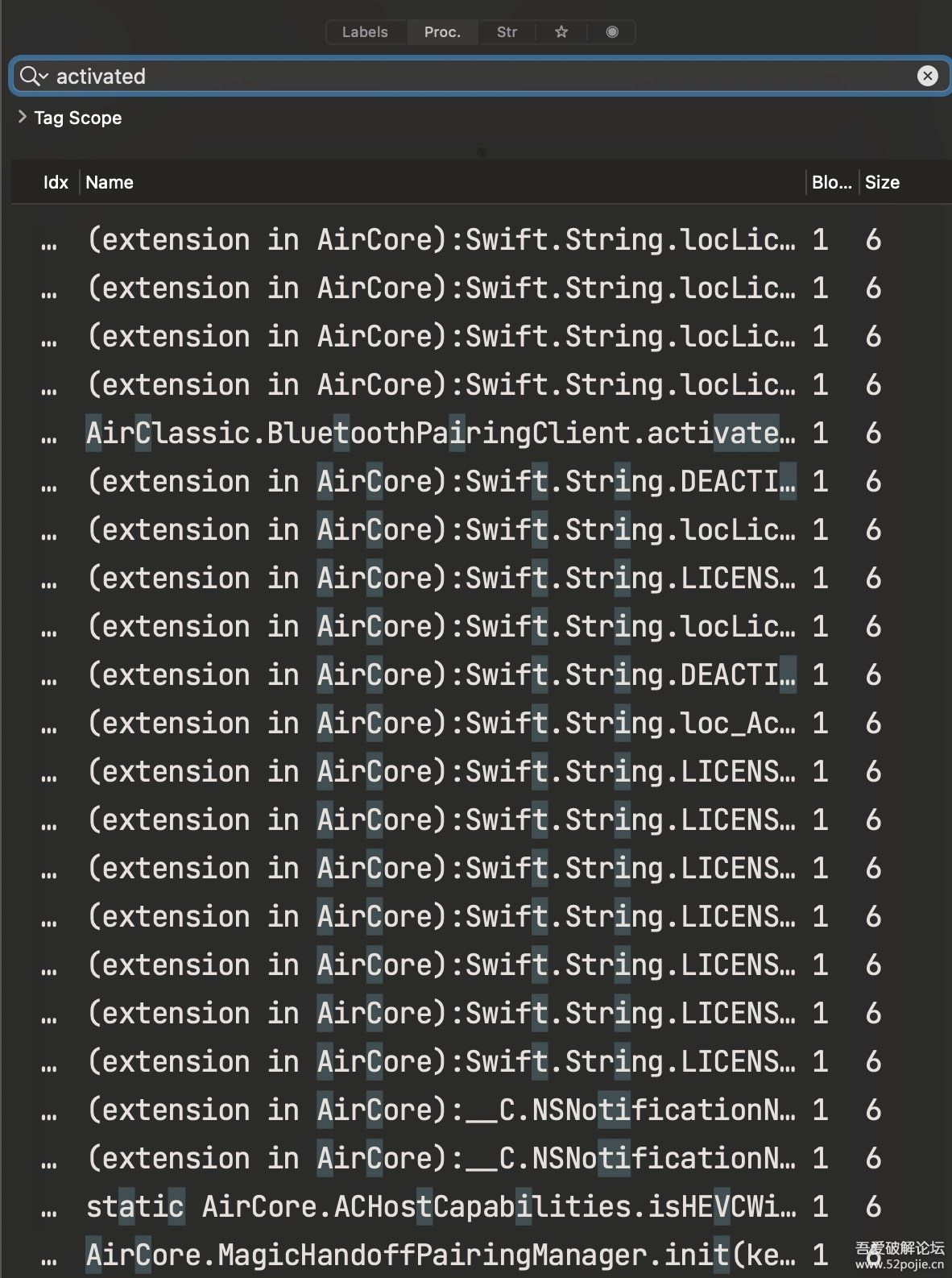
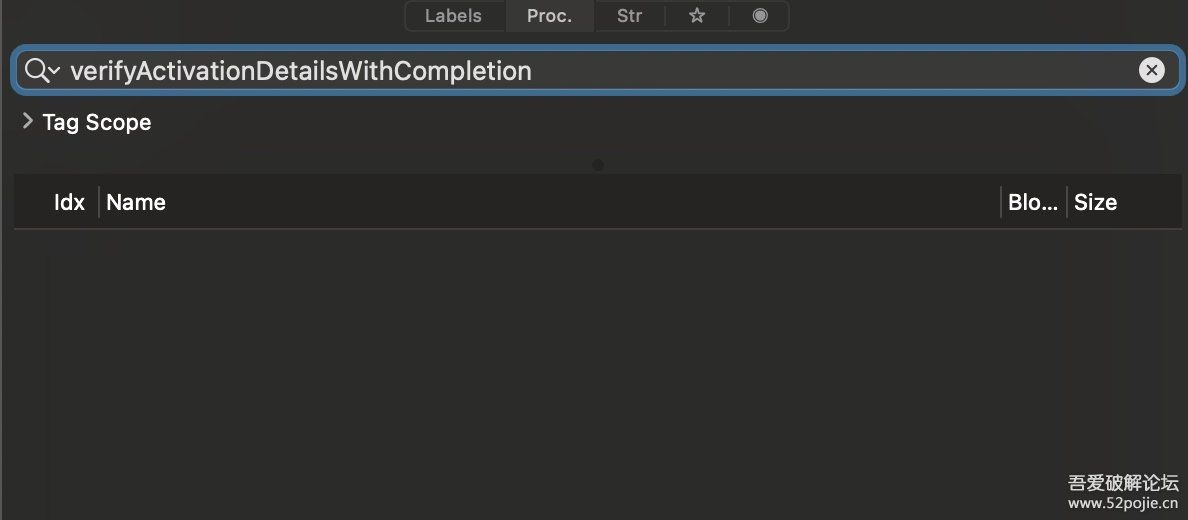
**0x01 灵根孕育大道生 只因黑子逞英雄**



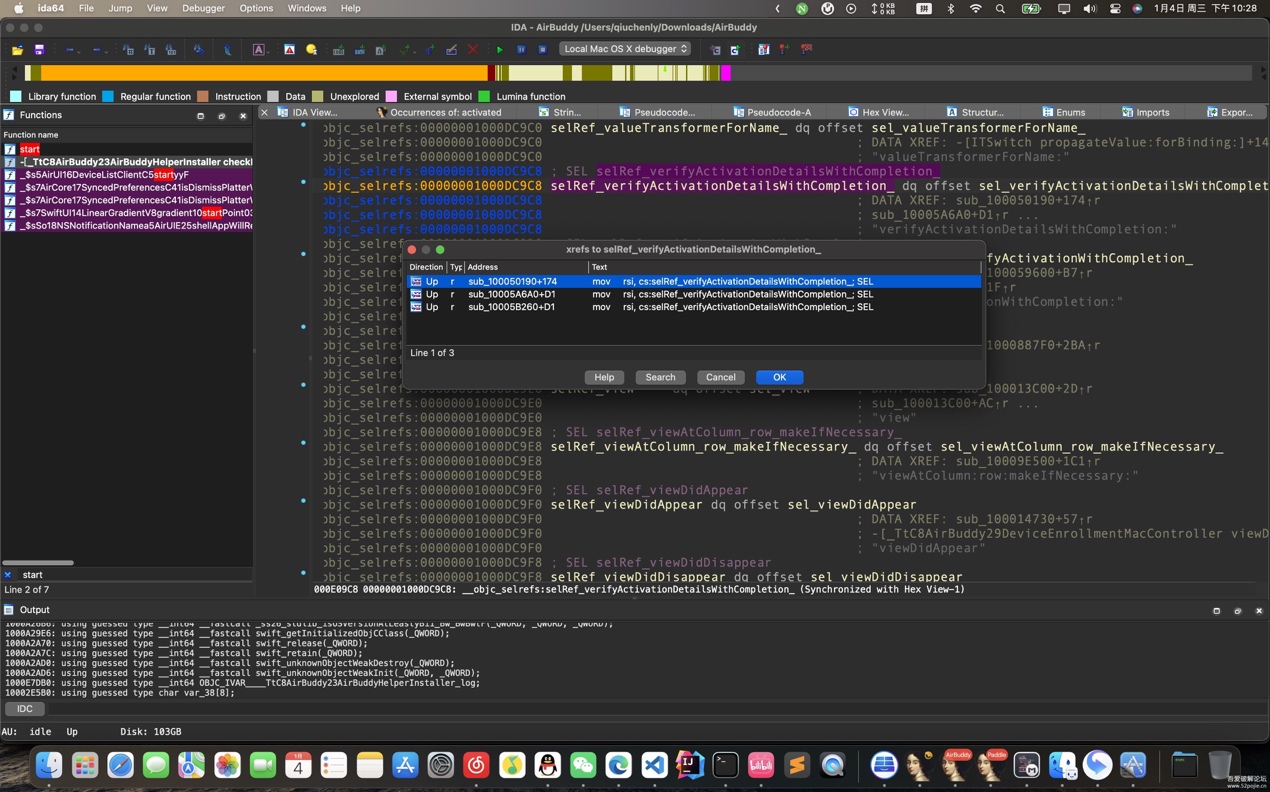
开幕给你一个小小的授权震撼。  
在我的HuaweiBook Pro 16in-ch上运行如图所示。

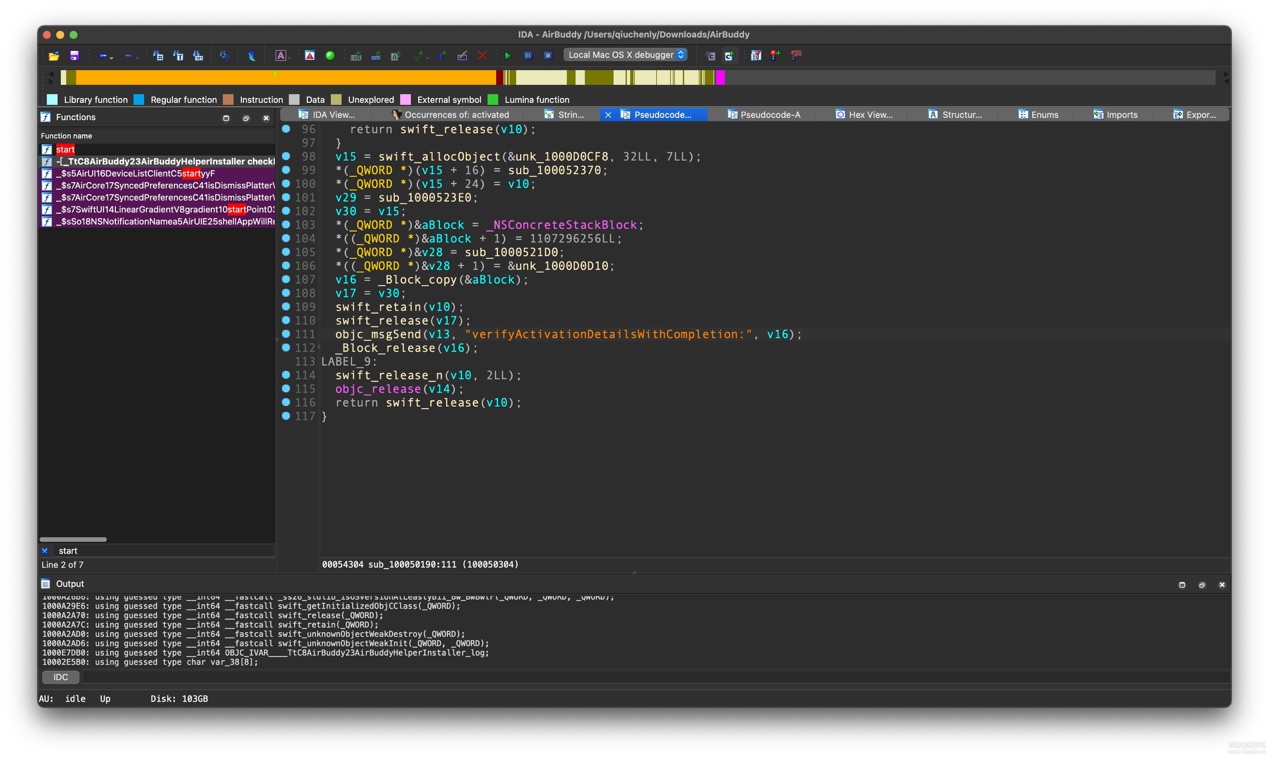
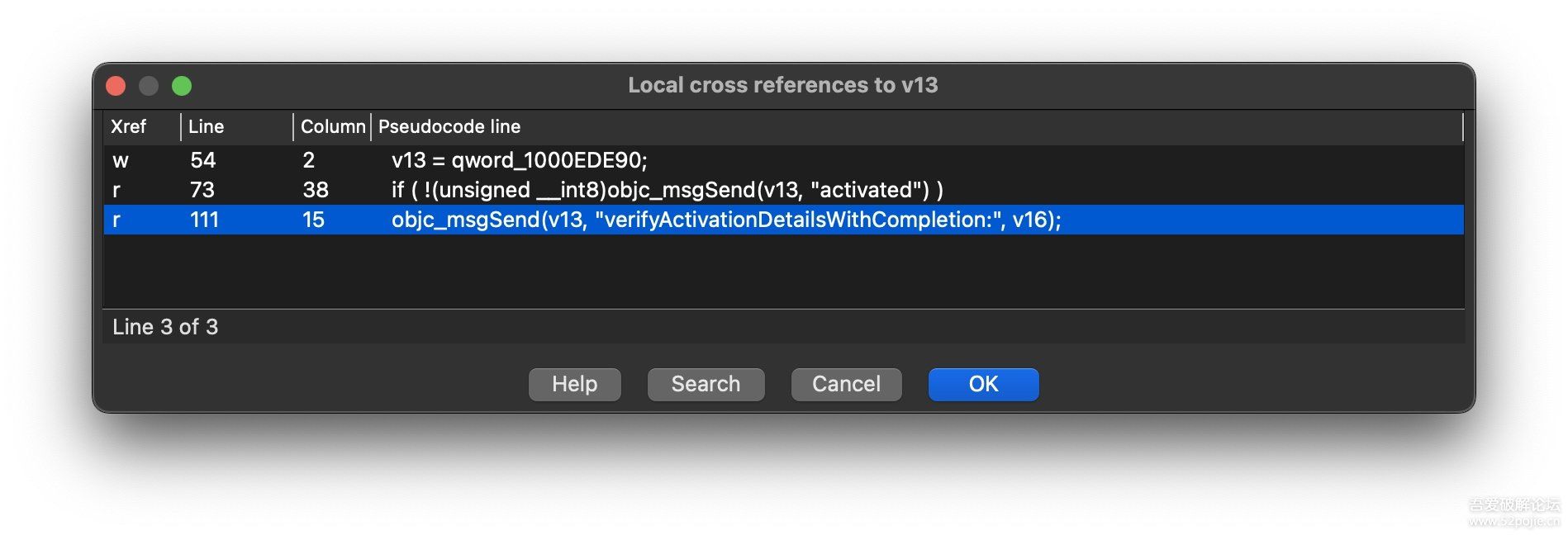
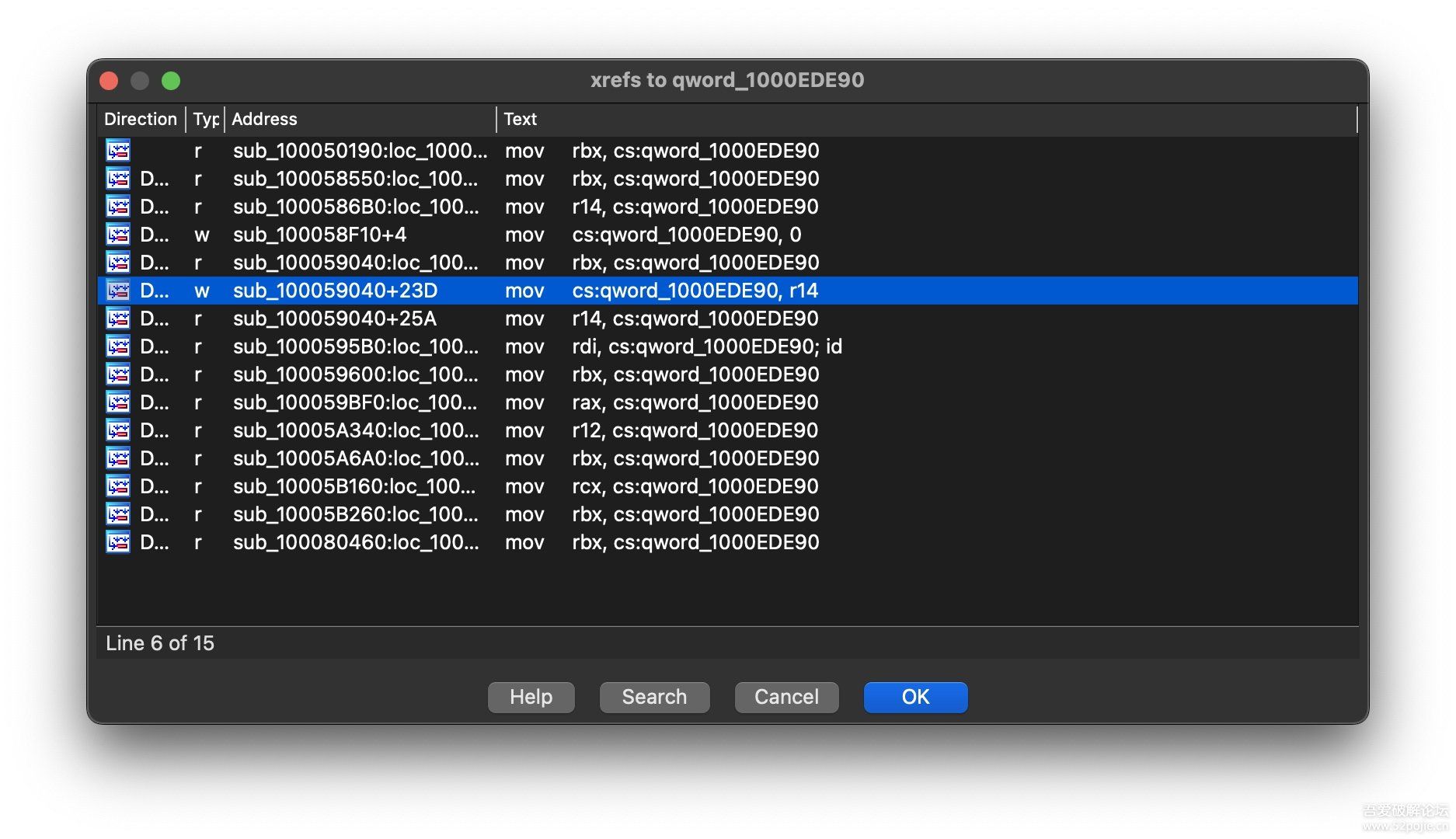
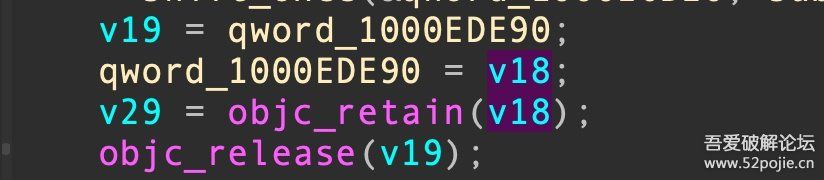
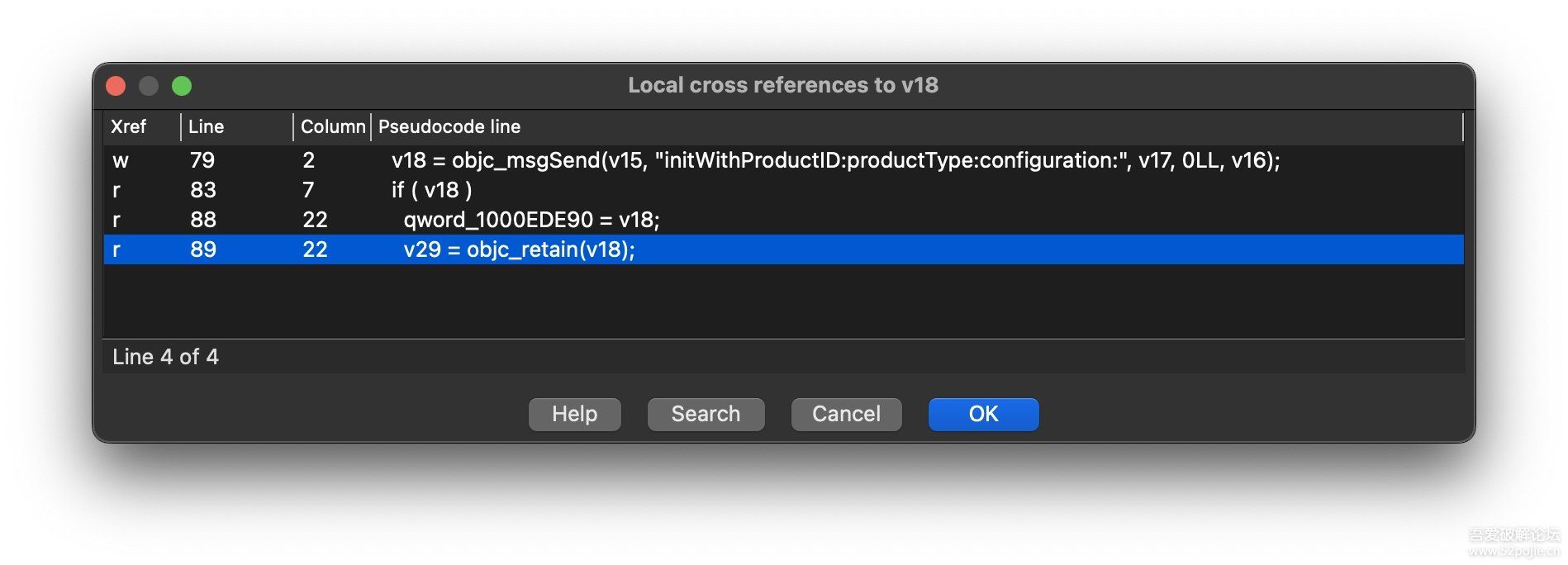
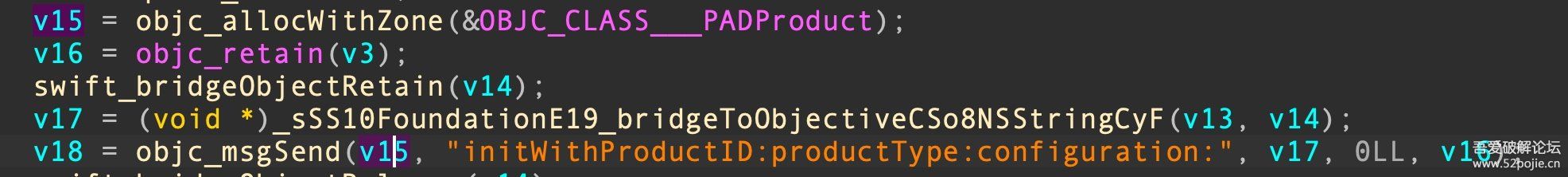
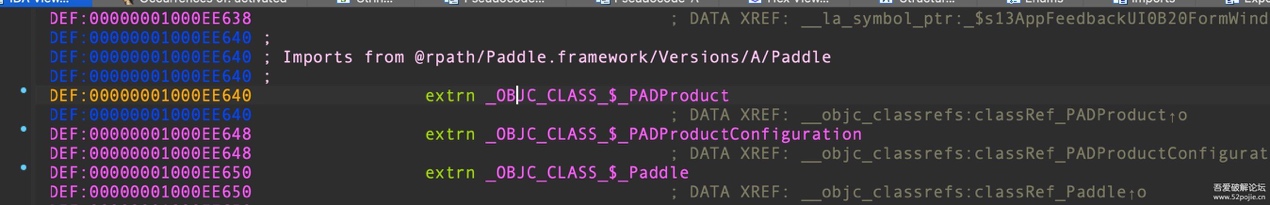
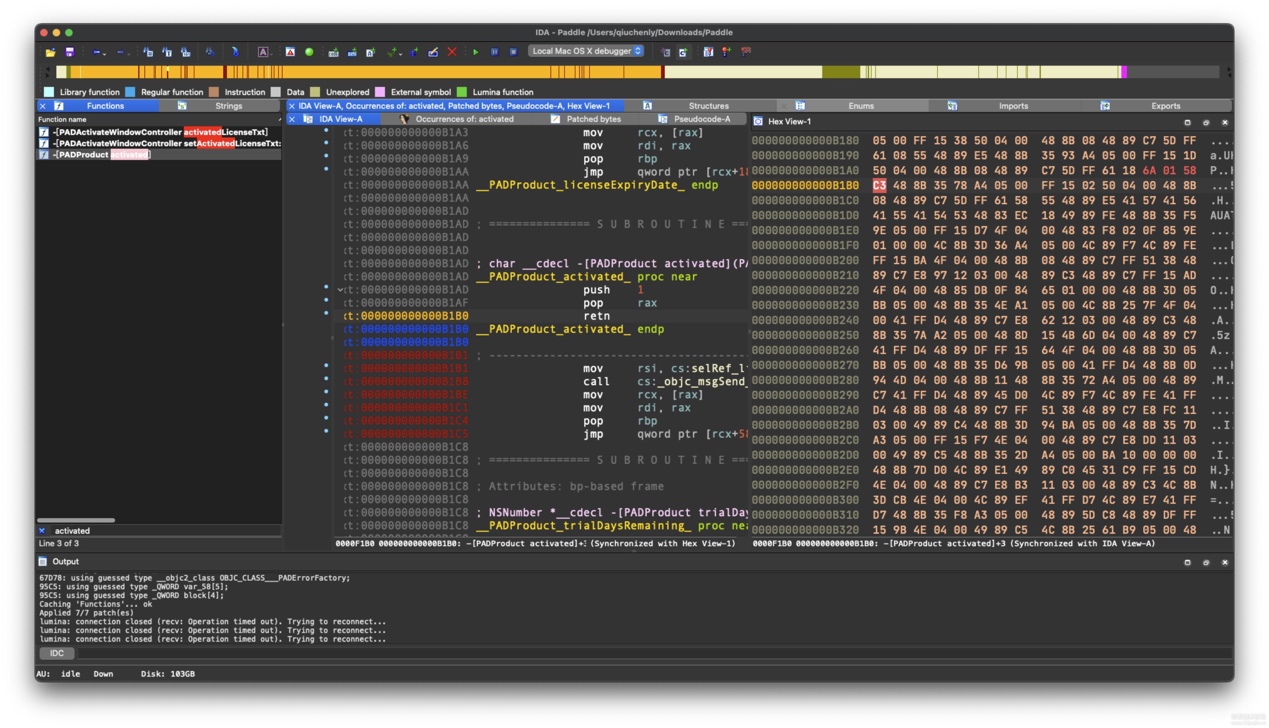
**0x02 正版暗藏关键语 落叶明察二进制**

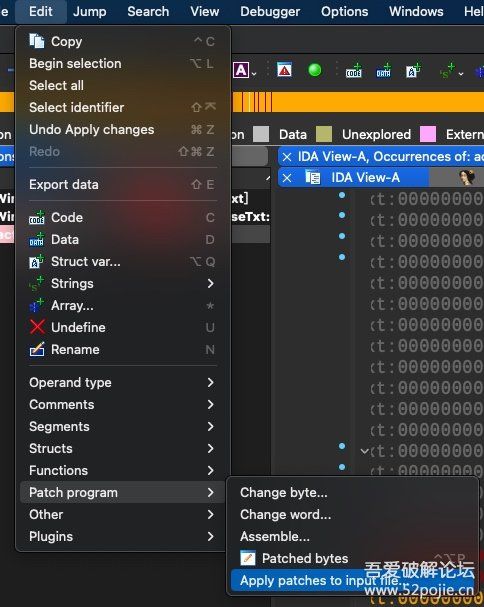
老样子先找破解点。  
然而我根本搜不到字符串内容，所以只能乱搜一气试试看了。  
由于IDA对原生Mach-O支持并不好 所以我这里用Hopper Dissembler看伪代码：  


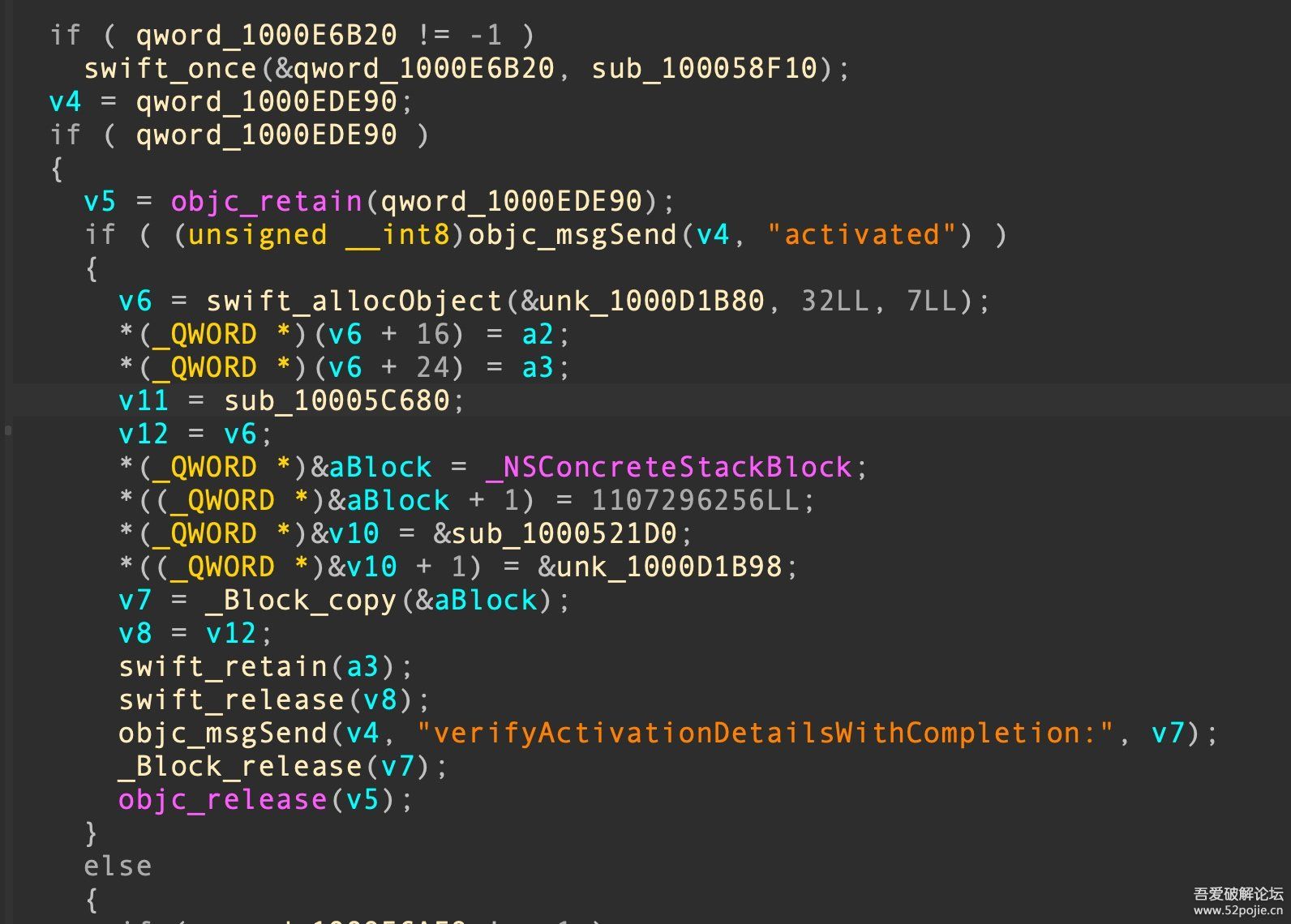
搜索了activated关键字符串看看有没有什么发现：  
可以看到运气很好有两个字符串引用 但是暂时不知道他们干嘛的，选择第一个看下  
   
然后我们继续x查找xref：  
   
我们发现这里引用了这个字符串，这里科普一下selector大概意思就是对象的某个属性名 比如ObjectA中有一个activated这个属性或者函数。  
我们翻一下后面的Data XREF：  
   
发现有四个函数引用了这个属性，那么我们随机抽取一个第三个sub函数来看看有没有什么发现：  
   
点进来查看伪代码，格局瞬间打开。  
我们发现这里很明显有一个 if ([rbx activated] != 0x0)函数的检查，那么我们可以大胆假设小心求证。  
我们假设这个字段返回1表示已激活，那么不等于0x0条件成立，然后下面注意有一个函数：[rbx verifyActivationDetailsWithCompletion:rax];字面意思验证激活信息，这个app的细节还是比较清晰的，所以我们发现这个假设貌似很有道理的样子。  
那么我们就找找怎么才能activated = True，这里为什么不修改跳转呢？因为你修改这里一处 那么以后后面处处都要修改 累死人。你还不知道是不是全部修改完了，所以我们要找到这个函数/属性的sub函数，让他强制返回1即可。  
我们此时双击“activated”你会发现HP毫无反应，这不合理！众所周知，但凡我调用某函数对象，那么必须该函数存在于app中否则我无法调用。那么我们知道macOS也有类似于dll之类的东西，即Framwork，你把它当作dll看也可以。  
我们大胆假设这个字段藏在某个动态链接库中:  
先搜一下这个函数名：  
   
悲剧了，无匹配。  
那么我们该怎么办？  
注意这个函数：verifyActivationDetailsWithCompletion，你会发现他也是搜不到的。这意味着我们可以拿它当突破口，因为这么长一串数据雷同可能性比较小，你要是单纯拿activated去搜能搜出很多无关结果。  
我们搜一下看看哈：  
   
搜不到，这下给我狠狠的蚌埠住了。难道今天就要饮恨于此？  
即然函数中搜不到，字符串只能搜到引用，那么我们就需要换个思路。

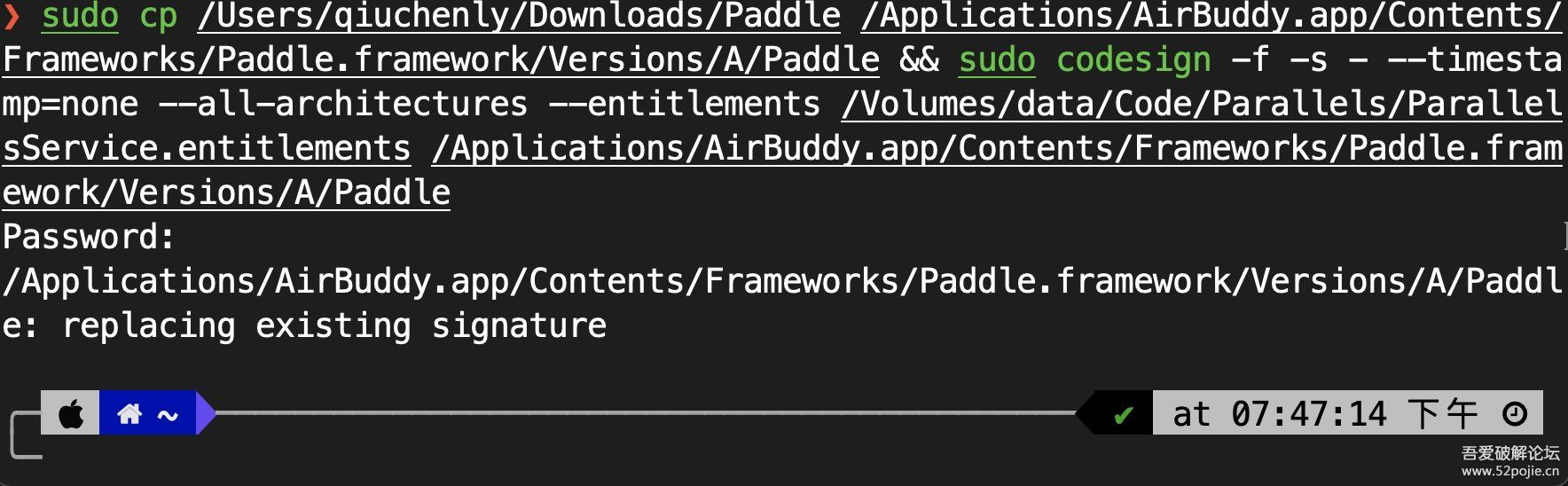
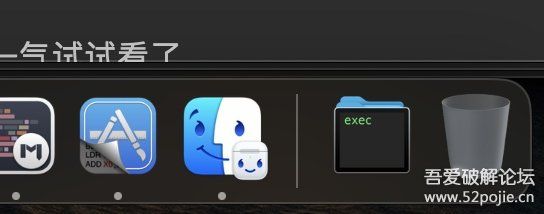
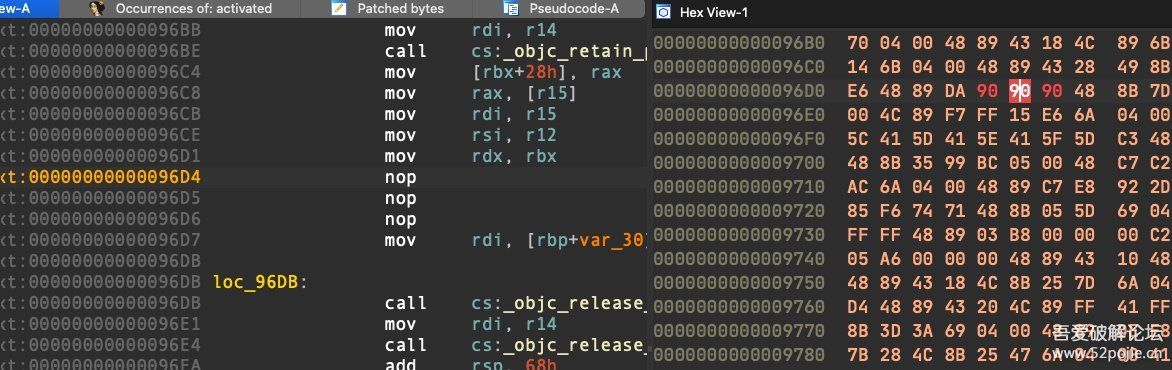
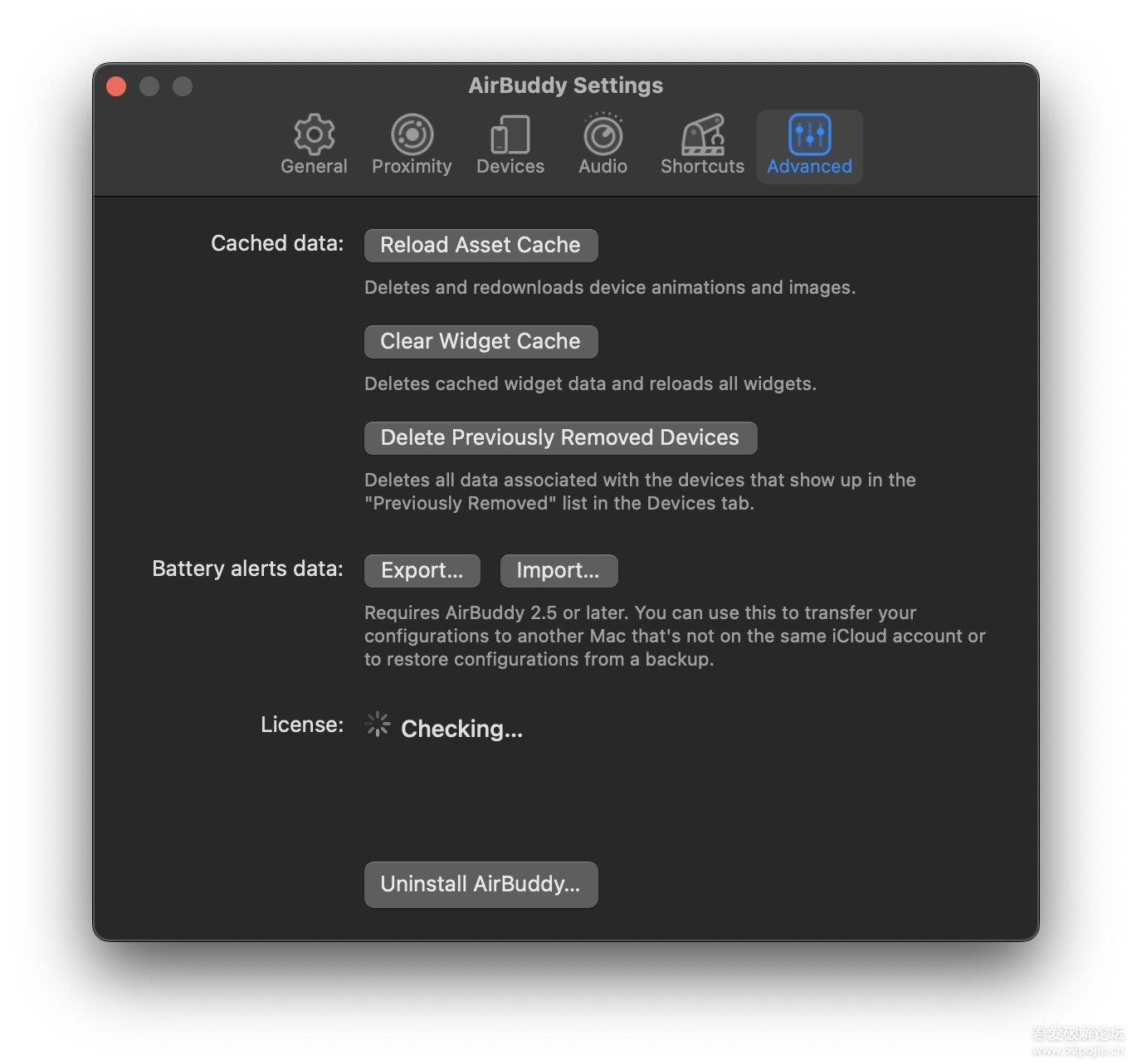
首先整理一下我们目前取得的成果：  
我们知道了两个幽灵函数可能存在于某个动态库中，但我们不知道是哪个。

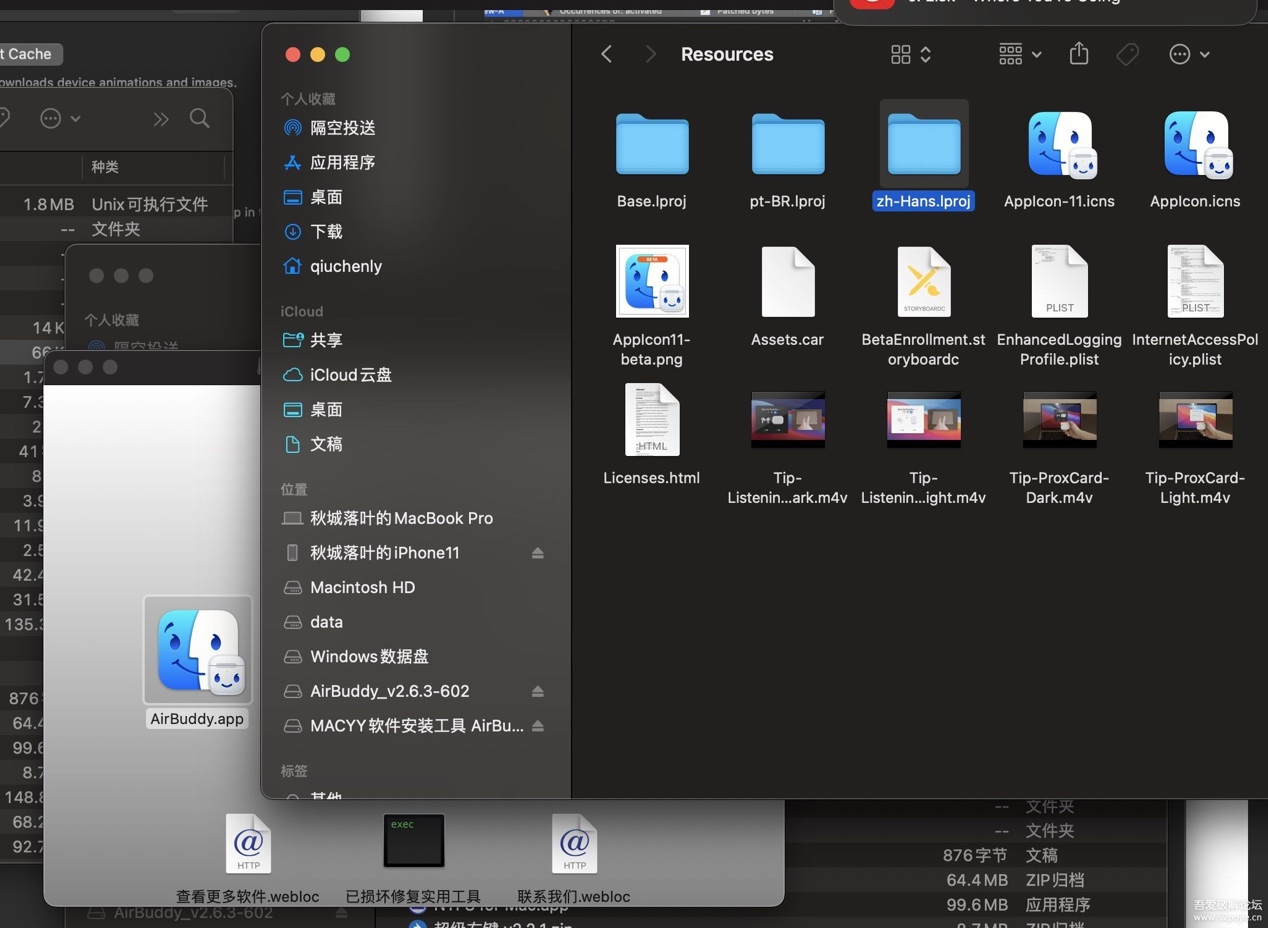
现在我们找个函数来看，即然我搜不到,那么我就溯源：很简单，即然你调用了就有调用方存在，那么我就找是谁调用了，然后再找调用方是何方神圣。  
我们先在IDA中找到他的引用：  


有三个结果，那么我们就随机抽取第一个幸运函数进去查看伪代码：  
   
可以看到msgSend调用方来自v13，那么我们就找一下v13是谁赋值了：  
   
可以看到v13是纯纯的ylg，不仅调用了下面的verify函数还调用了activated，那么就是它了！  
我们跳过去54行看下：  
v13 = qword\_1000EDE90;  
qword\_1000EDE90又从哪里来？  
查找引用发现  
   
来自59040函数的mov赋值,注意看r14寄存器。  
   
跳过来发现是v18寄存器，其实在ASM里面也是r14.  
我们接着找v18哪来的：  
   
从这里来到了v18的赋值地点，这下要给他们一点小小的汇编震撼了：  
   
v18=v15调用了init函数后的返回值，所以我们看到这个PADProduct类就浮现出来了。  
跳过去看下类定义：  
   
OK兄弟们，我们已经狠狠的超到了函数所在位置。  
那么我们找到这个framework解析一番：  
   
成功找到了这个函数。  
我们此时可以说进入了一个全新的阶段。  
现在我们回到主程序，我们之前上面说他是判断这个activated函数返回值的，那么我们先改掉这个函数返回值：6a 01 58 c3战神再次出征：  


狠狠的注入激活基因之后我们先保存一番：  


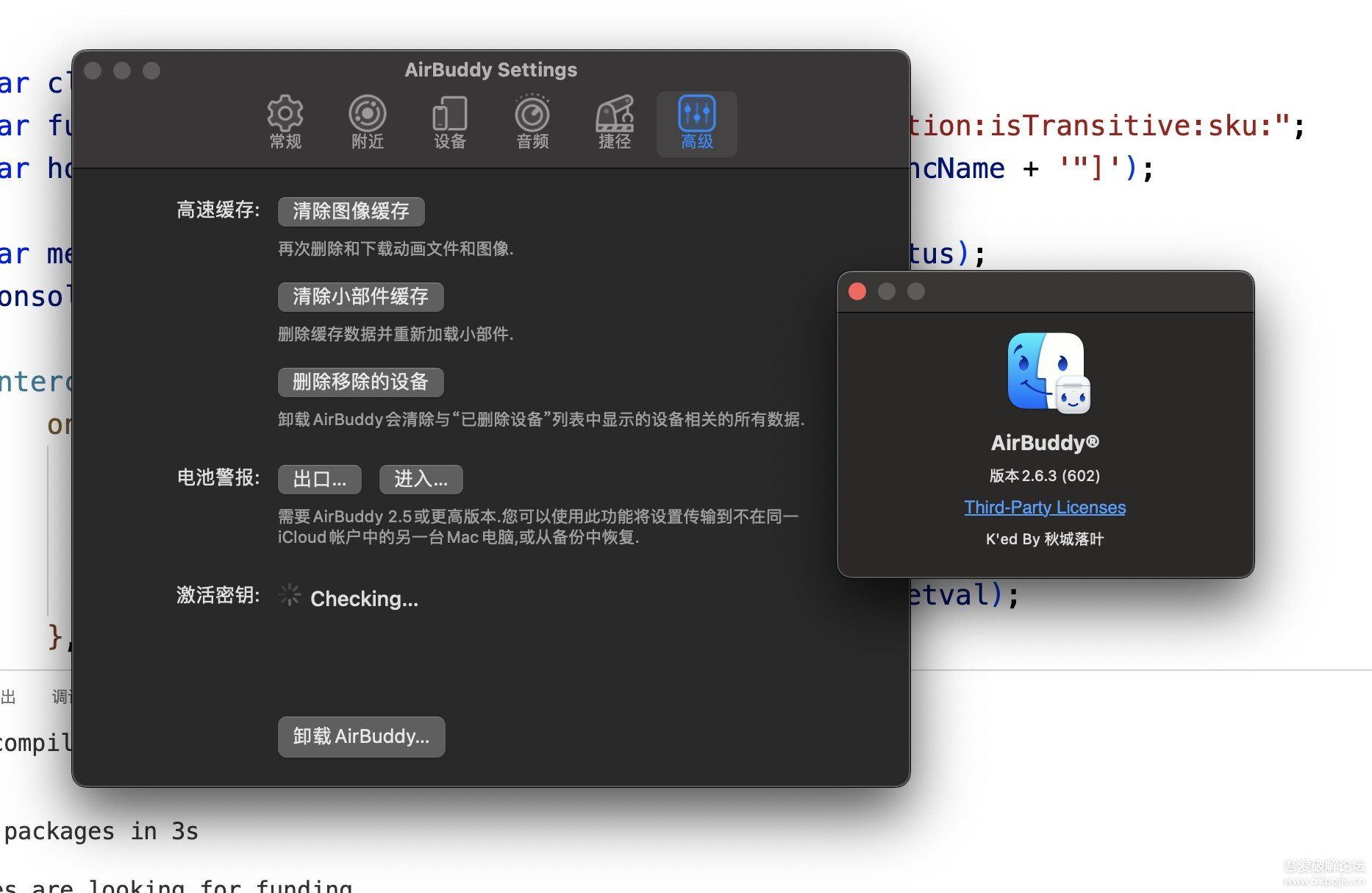
我们回头再看一下判断逻辑：  


主程序中认为，当dll返回1的时候进入verifyActivation检查范围，那么我们返回1做法是对的。  
我们先签名看看效果：  
   
然后我们打开App看看：  
   
小老弟你怎么回事？不显示界面？  
蚌埠住了，还有这种问题。  
那么我们浅浅的分析一下：当激活后他会做一些检查，那么很有可能是下面的检查函数中有什么死循环之类的算法由于我们没有提供正确的授权信息，导致UI不显示或者卡住了，那么我们尝试干掉下面那个验证函数：  
   
给他一点小小的nop震撼。  
保存签名打开app看看。  
   
成功打开，现在License显示Checking中，其实就是我nop掉了检查函数，现在授权是通过的，但是我没有做深入的修改导致它这里不显示激活成功之类的信息，这个Checking也是写死的，由于我nop掉了更新UI的信息函数它根本不动。边上的那个加载动画好像是系统自带的控件。

现在可以说是成功激活！  
下一步就是汉化，这个app居然不做中文汉化，思想意识形态是有问题的。  


从网上旧版本的破解版安装包里翻出汉化包，复制进app中，打开即可破解并汉化完成。

**0x03 是非成败转头空 一朝梦醒心怅然**



其实我这写的挺凌乱的