利用纯地址进行HOOK

结合上一次的Demo我们进行一点改进。在真实的逆向过程中，我们是没有函数符号的，因为符号表脱掉了。
接下来，我们通过纯地址的方式来HOOK函数。

分析MachO获取函数地址

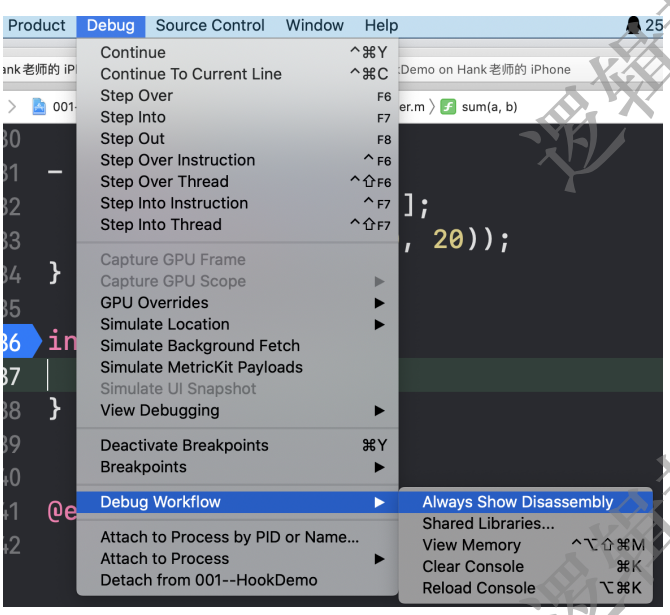
首先我们要用到一款软件。MachOView（我们逆向班的童鞋应该不陌生了）

利用它来分析我们Demo的主程序。接下来如何定位到我们的sum函数地址呢?

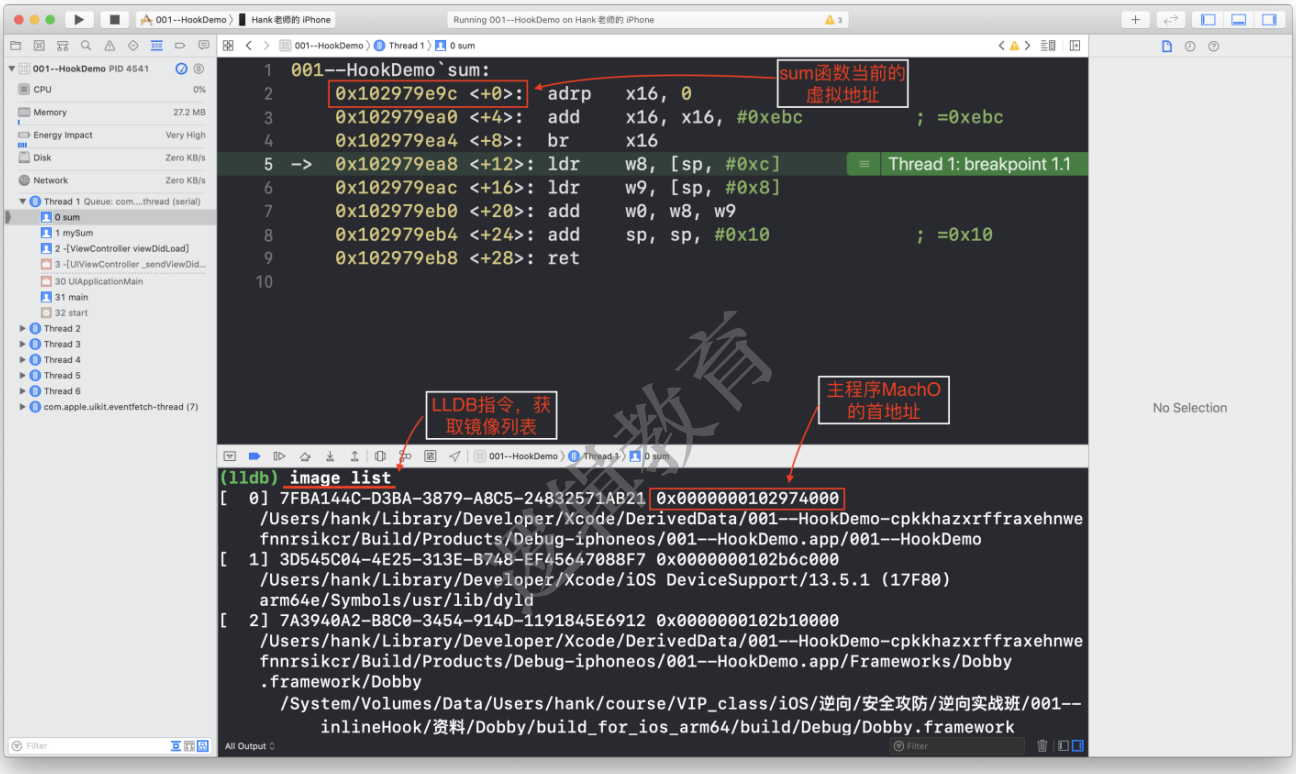
通过LLDB调试获取Sum函数的文件偏移地址。

- 首先在sum函数上下一个断点
- 然后通过汇编显示确定函数地址，在Xcode自带的Debug设置中设置一下搞定

Debug --> Debug Workflow --> Always Show Disassembly（一直显示汇编）



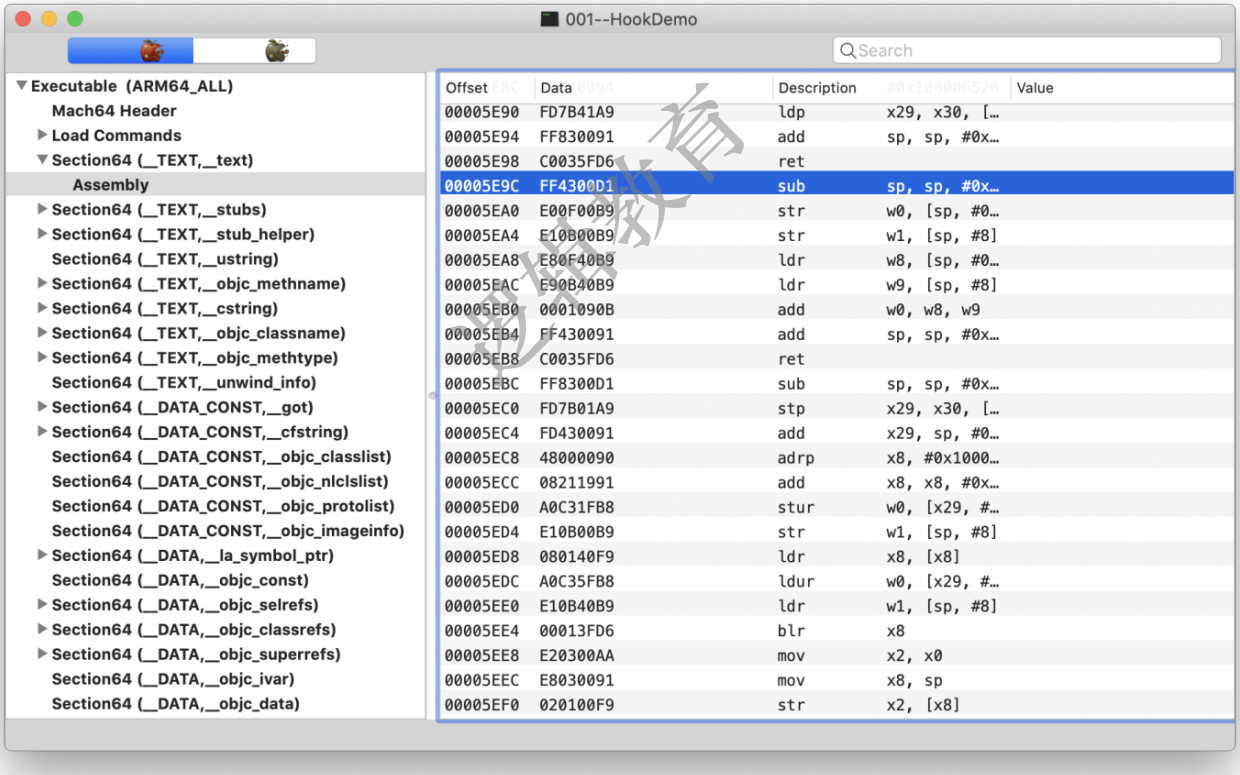
- 接下来通过LLDB指令找出主程序的首地址。得到 **sum函数偏移地址 = sum函数地址 – 主程序首地址**



- 我们可以用Mac自带的计算器算出偏移值，很方便（CMD+3 使用编程器）

在MachOView中查看到当前函数

我们打开MachOView，可以看到这里就是我们sum函数的开始位置。所以我们可以验证了sum函数的文件偏移地址就是0x5E9C。那么我们就用这个地址进行接下来的HOOK了。



HOOK前的准备

准备一个指针

注意：由于要方便我们计算，这里定义函数指针不要定义成为函数的结构。而是uintptr_t类型

```
//定义指针,表示sum函数的偏移地址!
static uintptr_t sumP = 0x100005E04;
```

动态获取ASLR

首先导入头#import <mach-o/dyld.h>
然后使用函数_dyld_get_image_vmaddr_slide获取ASLR，我们在Load中就可以这样写。

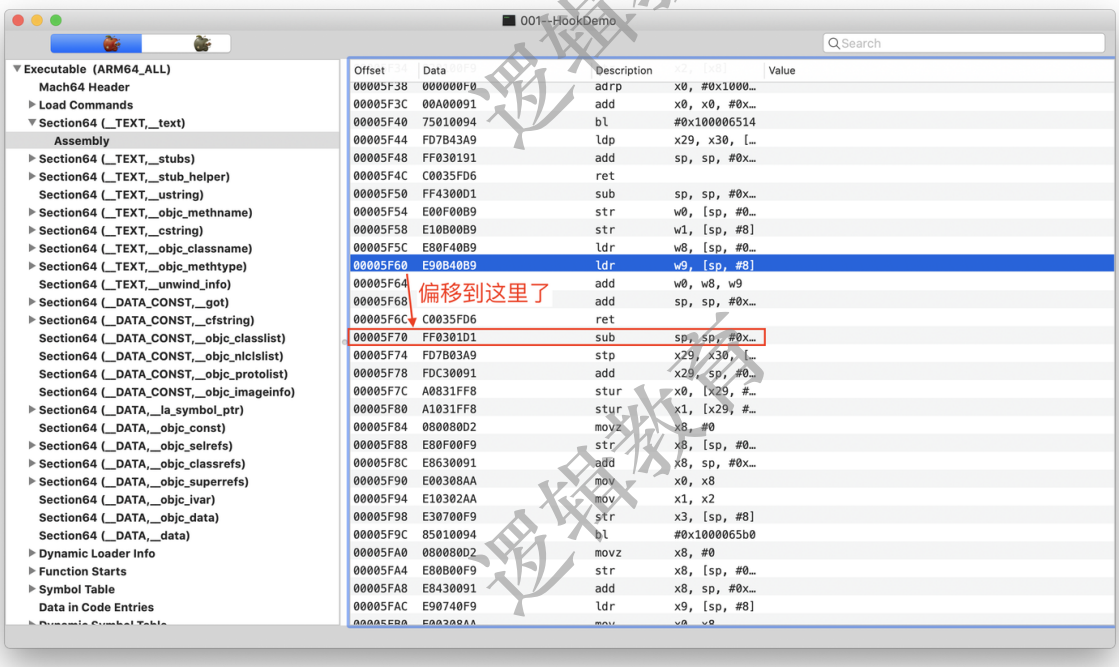
```
+ (void)load {
    //获取ASLR,让sumP变成准确的地址
    //参数0代表imagelist中的主程序（自己）
    uintptr_t aslr = _dyld_get_image_vmaddr_slide(0);
    sumP += aslr;
    //Hook sum
    DobbyHook((void *)sumP, mySum, (void *)&sum_p);
}
```

那么这个时候，我们能否运行成功?

为什么会出错?

原因很简单。我们刚才算出来的sum函数的文件偏移地址，在我们修改了这里的代码之后。变了没？！
当然变了！所以怎么办？只能再次去获取sum函数的偏移。（注意：HOOK别人应用时因为我们是动态库注入，所以不会修改目标应用的MachO，那么偏移值是固定的。没有这么麻烦）

> 注意：
> 这里有一个小技巧！因为修改代码之后，我们的sum函数有可能会没法断点调试（因为程序交换就崩溃），那么没关系，我们不做交换
> 1、将交换的代码注释起来
> 2、获取到最小改动的sum函数偏移地址。
> 3、再讲该执行的代码打开。生成最新的MachO。
> 4、在MachOView中找到那附近（往高地址走）最像函数起始位置的作为sum函数的偏移（一般都是）



这个时候，我们只需要将sum函数的偏移地址修改一下就可以了！

```
//定义指针,表示sum函数的偏移地址！
static uintptr_t sumP = 0x100005F70;
```

运行！通过纯地址也能轻松搞定InlineHook了！这样离我们HOOK三方应用就更加接近了。