
目录

前言	1.1
IDA概览	1.2
IDA快手上手	1.3
IDA通用知识	1.4
版本	1.4.1
下载	1.4.1.1
历史版本	1.4.1.2
界面布局	1.4.2
命名和含义	1.4.3
举例	1.4.3.1
自动分析	1.4.4
搜索	1.4.5
快捷键	1.4.6
IDA功能详解	1.5
查看代码	1.5.1
汇编代码	1.5.1.1
伪代码	1.5.1.2
函数调用	1.5.1.3
结构体类定义	1.5.1.4
字符串	1.5.2
函数列表	1.5.3
导入和导出	1.5.4
打补丁	1.5.5
插件	1.5.6
IDA使用心得	1.6
打开和关闭	1.6.1
ObjC	1.6.2
IDA动态调试	1.7
iOS	1.7.1
搭建环境	1.7.1.1
开始调试	1.7.1.2
调试界面	1.7.1.3
常见问题	1.7.1.4
子教程	1.8
附录	1.9
文档和资料	1.9.1
参考资料	1.9.2

逆向利器：IDA

- 最新版本： `v1.1.4`
- 更新时间： `20250603`

简介

介绍逆向领域中功能强大且好用的利器：IDA。先介绍IDA概览；再介绍IDA的快速上手过程；再介绍IDA中通用的基础知识，包括版本选择、界面相关比如布局等、常见命名和含义、自动分析过程、搜索、快捷键；以及介绍IDA各种功能，包括查看代码，比如汇编代码、F5伪代码、函数调用、结构体的类的定义、字符串、函数列表插件等等；再去记录IDA的使用心得；最后整理一些IDA相关的文档和资料，供参考。

源码+浏览+下载

本书的各种源码、在线浏览地址、多种格式文件下载如下：

HonKit源码

- [crifan/reverse_tool_ida](#): 逆向利器：IDA

如何使用此HonKit源码去生成发布为电子书

详见：[crifan/honkit_template: demo how to use crifan honkit template and demo](#)

在线浏览

- 逆向利器：IDA [book.crifan.org](#)
- 逆向利器：IDA [crifan.github.io](#)

离线下载阅读

- 逆向利器：IDA PDF
- 逆向利器：IDA ePub
- 逆向利器：IDA Mobi

版权和用途说明

此电子书教程的全部内容，如无特别说明，均为本人原创。其中部分内容参考自网络，均已备注了出处。如发现有侵权，请通过邮箱联系我 `admin` 艾特 `crifan.com`，我会尽快删除。谢谢合作。

各种技术类教程，仅作为学习和研究使用。请勿用于任何非法用途。如有非法用途，均与本人无关。

鸣谢

感谢我的老婆陈雪的包容理解和悉心照料，才使得我 `crifan` 有更多精力去专注技术专研和整理归纳出这些电子书和技术教程，特此鸣谢。

其他

作者的其他电子书

本人 `crifan` 还写了其他 `150+` 本电子书教程，感兴趣可移步至：

[crifan/crifan_ebook_readme](#): Crifan的电子书的使用说明

关于作者

关于作者更多介绍，详见：

[关于CrifanLi李茂 – 在路上](#)

crifan.org，使用署名4.0国际(CC BY 4.0)协议发布 all right reserved，powered by Gitbook最后更新：2025-06-03 21:56:53

IDA概览

在逆向领域，有款很功能强大且好用的工具=利器是： IDA

- IDA
 - 版本和名称
 - 概述：
 - IDA 有多个版本，功能最强、用的最多最广泛是： IDA Pro
 - 所以 IDA Pro 常简称为： IDA
 - 详解
 - 后续章节： [版本](#)
 - 概述： 逆向工程的利器
 - 图

IDA PRO, in a nutshell



A disassembler



A debugger

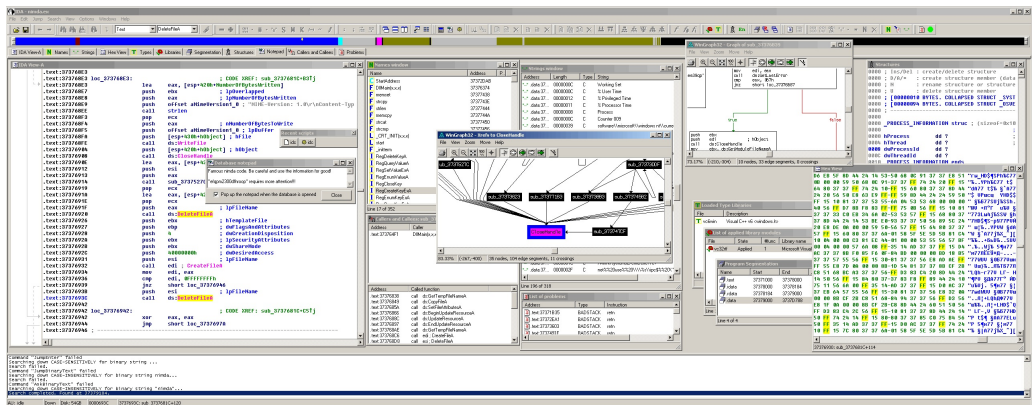


Interactive

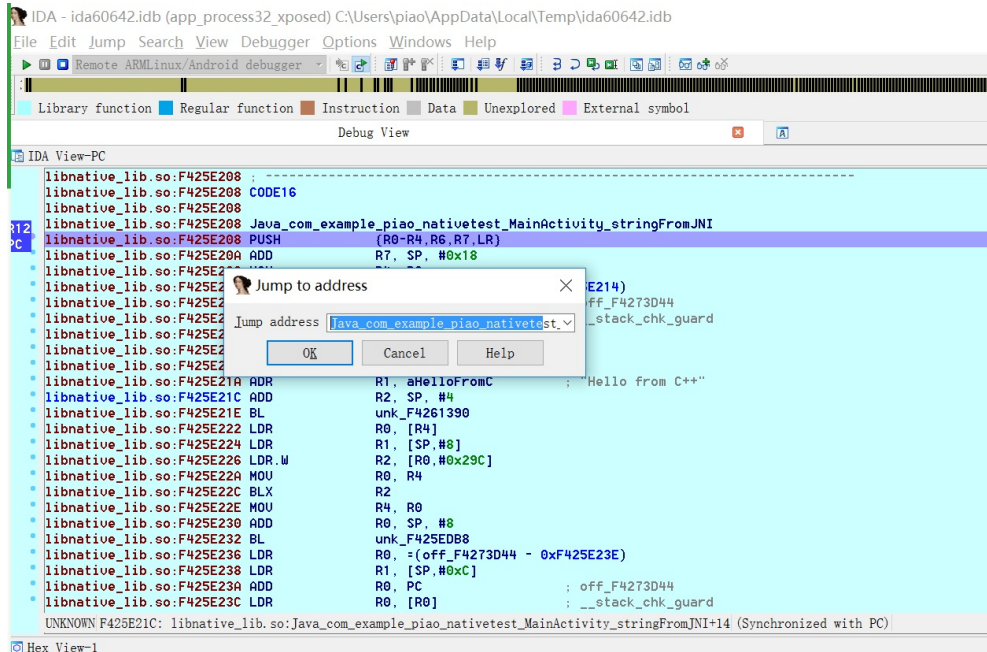
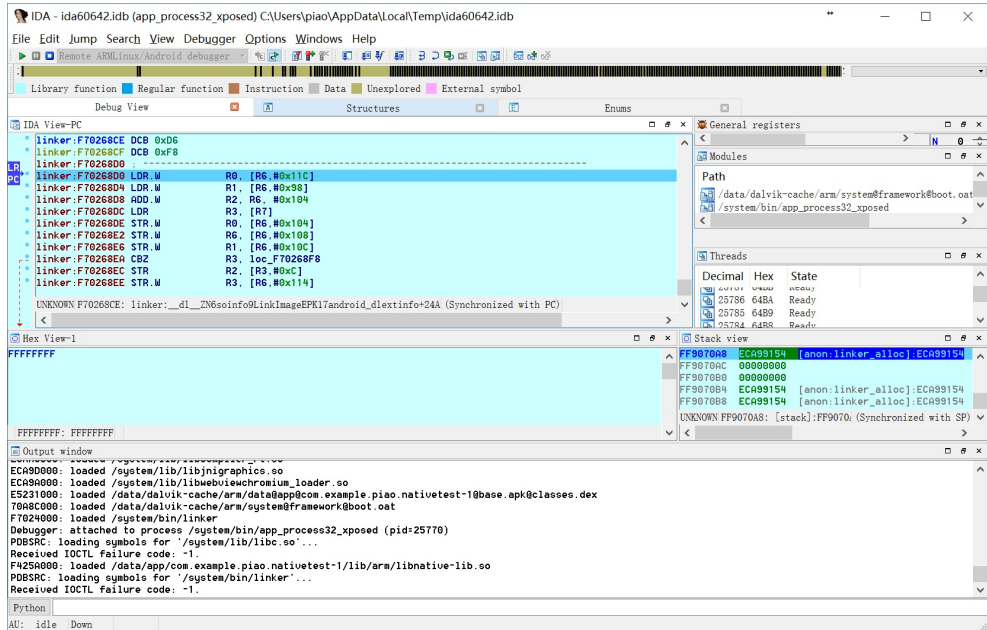


Programmable

- - 文字
 - IDA Pro – the state-of-the-art binary code analysis solution
 - IDA Pro is the flagship product of Hex-Rays , the software provider in reverse engineering.
 - Being an interactive and programmable disassembler and debugger, IDA Pro provides excellent quality performance on different platforms and is compatible with many processors. IDA Pro has become the de-facto standard for the analysis of hostile code, vulnerability research and commercial off-the-shelf validation
 - 常用于
 - iOS逆向
 - 静态分析： 逆向二进制，研究代码逻辑
 - 常用功能： 函数、F5伪代码、字符串、类的结构体定义等等
 - 动态调试： 调试iOS的app
 - Android逆向
 - 支持对Android的静态分析与动态调试
 - 包括： 动态调试so文件
 - 特点
 - 支持多平台运行： Windows 、 Mac 、 Linux (CentOS 、 Ubuntu 等)等
 - 支持多种CPU架构： x86/x64 、 ARM/ARM64 、 MIPS/MIPS64 、 PowerPC/PPC64 、 Motorola 68K/Coldfire 等共68种
 - 支持插件，可以扩展各种功能
 - 可编程=支持API接口调用， 写脚本， 实现各种功能
 - 主页
 - <https://hex-rays.com/>
 - 关于
 - [IDA: About](#)
 - 下载
 - [IDA Support: Download Center](#)
 - 注： IDA Pro 是商业收费软件， 请支持和购买正版
 - 官网截图



■ 调试安卓dex



IDA快手上手

- iOS逆向
 - 常用：（Mac中）用IDA分析代码
 - 下载和安装IDA
 - 找到要分析的iOS的app的二进制文件
 - 拖动二进制到IDA中
 - 等待分析完毕
 - 利用各种功能，研究代码逻辑
 - 常用功能
 - 字符串
 - 搜索感兴趣的关键词
 - 比如：越狱对应的单词：`jailbreak`、`jail`、`jb` 等
 - 函数
 - 搜索已找到的iOS的ObjC的类和函数
 - 然后打开查看代码逻辑
 - 伪代码
 - 当打开函数后，默认是汇编代码，按 `F5` 即可打开 伪代码
 - 近似于自己写的源码，人类可读的那种，就可以分析代码，搞懂函数的基本（甚至全部的）逻辑了
 - 结构体：类的定义
 - 结构体定义的来源
 - 有些是自动分析出来的
 - 如果没有错误，无需调整
 - 有些需要自己额外新增类的定义
 - 效果
 - 如果类的结构体定义是正确的话，那么伪代码中，会自动解析出，类的函数和属性的调用，即可大大增加伪代码的可读性
 - 之后的重点
 - IDA的静态分析：`IDA` 的 `F5`伪代码，查看伪代码的大概逻辑
 - Xcode的动态调试：iOS逆向的 `Xcode+ + MonkeyDev + LLDB` 的动态调试，通过hook等手段，确认调用了哪些函数，参数值如何等等
 - 互相配合：把Xcode中的实时的汇编代码 和 IDA中伪代码，互相对应起来，便于理解代码逻辑
 - 后续的优化代码逻辑：在逐渐搞懂更多代码逻辑后，继续给IDA中的 伪代码 去优化代码，主要是给参数、变量、函数等改名，以及更进一步的，新增或修改类的结构体定义，使得伪代码中自动解析出正确的类的函数和属性的调用等内容。
 - 偶尔用：用IDA调试二进制

crifan.org, 使用署名4.0国际(CC BY 4.0)协议发布 all right reserved, powered by Gitbook最后更新：2024-10-02 09:59:32

IDA通用知识

此处整理IDA中的基础的通用的知识。

- 版本
- 界面
 - 功能布局
 - 显示相关
- 命名和含义
- 自动分析
 - 基本流程
 - 进度
- 搜索
- 快捷键

crifan.org, 使用[署名4.0国际\(CC BY 4.0\)协议](#)发布 all right reserved, powered by Gitbook最后更新: 2023-07-14 22:29:17

版本

IDA软件的不同版本

- 常用结论
 - 此处iOS逆向的话，常用的是 v7.0 的 IDA Pro （中64位的 id64 ）
- 概述
 - IDA Teams : Collaborative reverse-engineering work
 - 团队合作时才需要
 - IDA Pro : The state-of-the-art binary code analysis tool
 - 常简称为 IDA -> 大家常说的IDA，指的是IDA Pro
 - 功能最全，最强大，但价格也最贵
 - IDA Home : Affordable tool for reverse engineering hobbyists
 - 功能相对没 IDA Pro 全，但对于个人（应该）基本够用，价格相对也便宜点（所以宣传是Affordable）
 - IDA Free : Free binary code analysis tool to evaluate IDA's basic functionalities
 - 免费，但功能有限
 - IDA Demo : 仅仅作演示用，了解功能而已
- 详解

o

下载IDA

- [IDA官网](#)
 - [Pricing](#)
 - 免费版
 - [IDA Free](#)
 - Core features at no cost to start your reverse engineering journey
 - [Classroom](#)
 - Teach your class with best-in-class software
 - Explore the free IDA Classroom licenses for academia and education providers. Or use IDA Pro for research and teaching activities at a discount.
 - 收费版本
 - [IDA-pro](#)
 - A powerful disassembler, decompiler and a versatile debugger. In one tool
 - [IDA-home](#)
 - Almost as powerful as Pro, designed for all hobbyists
 - [IDA Pro OEM](#)
 - The license you need to sell software or Saas based on IDA

crifan.org, 使用署名4.0国际(CC BY 4.0)协议发布 all right reserved, powered by Gitbook最后更新: 2024-10-02 10:01:06

IDA的历史版本号

- 心得
 - 刚发现，IDA的版本号，是 Major.Minor.ReleaseDate 的模式
 - 举例
 - IDA 8.1.221006 October 6, 2022
 - 8.1.221006
 - 8.1 : 主次版本号
 - 221006 : 发布日期=October 6, 2022

整理出所有的历史版本：

IDA updates and releases

- IDA 9.x
 - IDA 9.0.240925 September 30, 2024
- IDA 8.x
 - IDA 8.4.240527 (SP2) May 27, 2024
 - IDA 8.4.240320 (SP1) March 20, 2024
 - IDA 8.4.240215 February 15, 2024
 - IDA 8.3.230608 June 8, 2023
 - IDA 8.2.230124 (SP1) January 24, 2023
 - IDA 8.2.221215 December 15, 2022
 - IDA 8.1.221006 October 6, 2022
 - IDA 8.0.220829 (SP1) August 29, 2022
 - IDA 8.0.220729 July 29, 2022
- IDA 7.x
 - IDA 7.7.220118 (SP1) January 18, 2022
 - IDA 7.7.211224 December 24, 2021
 - IDA 7.6.210427 (SP1) April 28, 2021
 - IDA 7.6.210322 March 22, 2021
 - IDA 7.5.201028 (SP3) October 28, 2020
 - IDA 7.5.200728 (SP2) July 28, 2020
 - IDA 7.5.200619 (SP1) June 19, 2020
 - IDA 7.5.200519 May 19, 2020
 - IDA 7.4.191112 (SP1) November 12, 2019
 - IDA 7.4.191011 October 11, 2019
 - IDA 7.3.190614 June 14, 2019
 - IDA 7.2.181105 November 5, 2018
 - IDA 7.1.180227 February 27, 2018
 - IDA 7.0.171130 (SP1) November 30, 2017
 - IDA 7.0.190914 September 14, 2017
- IDA 6.x
 - IDA 6.95.160808 August 08, 2016
 - IDA 6.9.151221 December 21, 2015
 - IDA 6.8.150413 April 13, 2015
 - IDA 6.7.141229 December 29, 2014
 - IDA 6.6.140604 June 04, 2014
 - IDA 6.5.131217 December 17, 2013
 - IDA 6.4.130306 March 6, 2013
 - IDA 6.4 January 10, 2013
 - IDA 6.3 May 31, 2012

- IDA 6.2 October 05, 2011
- IDA 6.1 April 08, 2011
- IDA 6.0 October 01, 2010
- IDA 5.x
 - IDA 5.7 June 25, 2010
 - IDA 5.6 December 30, 2009
 - IDA 5.5 June 12, 2009
 - IDA 5.4 January 29, 2009
 - IDA 5.3 July 14, 2008
 - IDA 5.2 November 20, 2007
 - IDA 5.1 February 21, 2007
 - IDA 5.0 March 23, 2006
- IDA 4.x
 - IDA 4.9(SP) January 27, 2006
 - IDA 4.9 September 25, 2005
 - IDA 4.8 March 15, 2005
 - IDA 4.7 August 2004
 - IDA 4.6 October 27, 2003
 - IDA 4.x August 29, 2022
- IDA 3.x
 - IDA 3.x

crifan.org, 使用署名4.0国际(CC BY 4.0)协议发布 all right reserved, powered by Gitbook最后更新: 2024-10-02 10:01:55

界面布局

此处整理，IDA中关于界面显示和布局方面的内容。

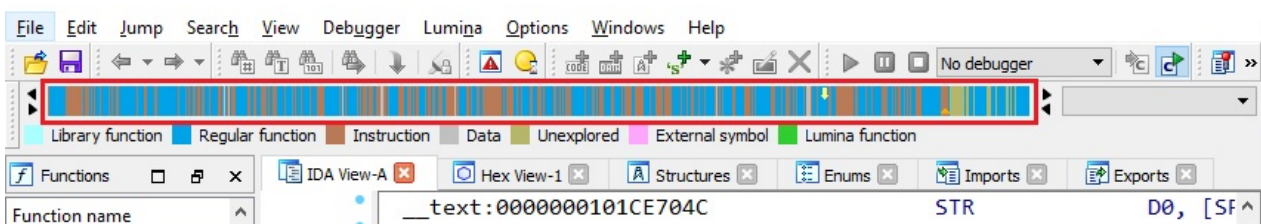
导航条=navigator

关于这个：

Navigation band = navigator = navbar = 导航栏 = 导航条

有专门的介绍

Igor's tip of the week #49: Navigation band – Hex Rays (hex-rays.com)

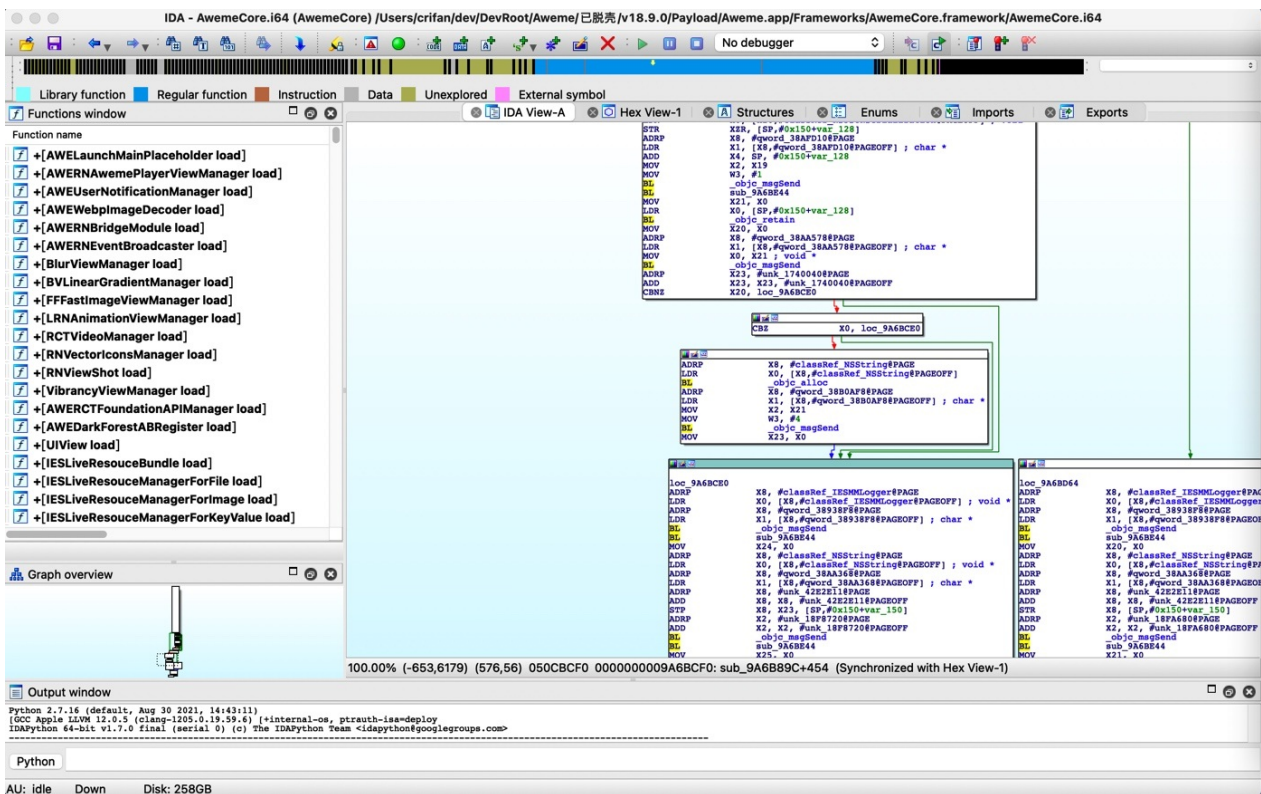


有空可以好好学习看看

切换显示模式

Text View 和 Graph View

IDA View-A：在 Text View 和 Graph View 模式之间切换

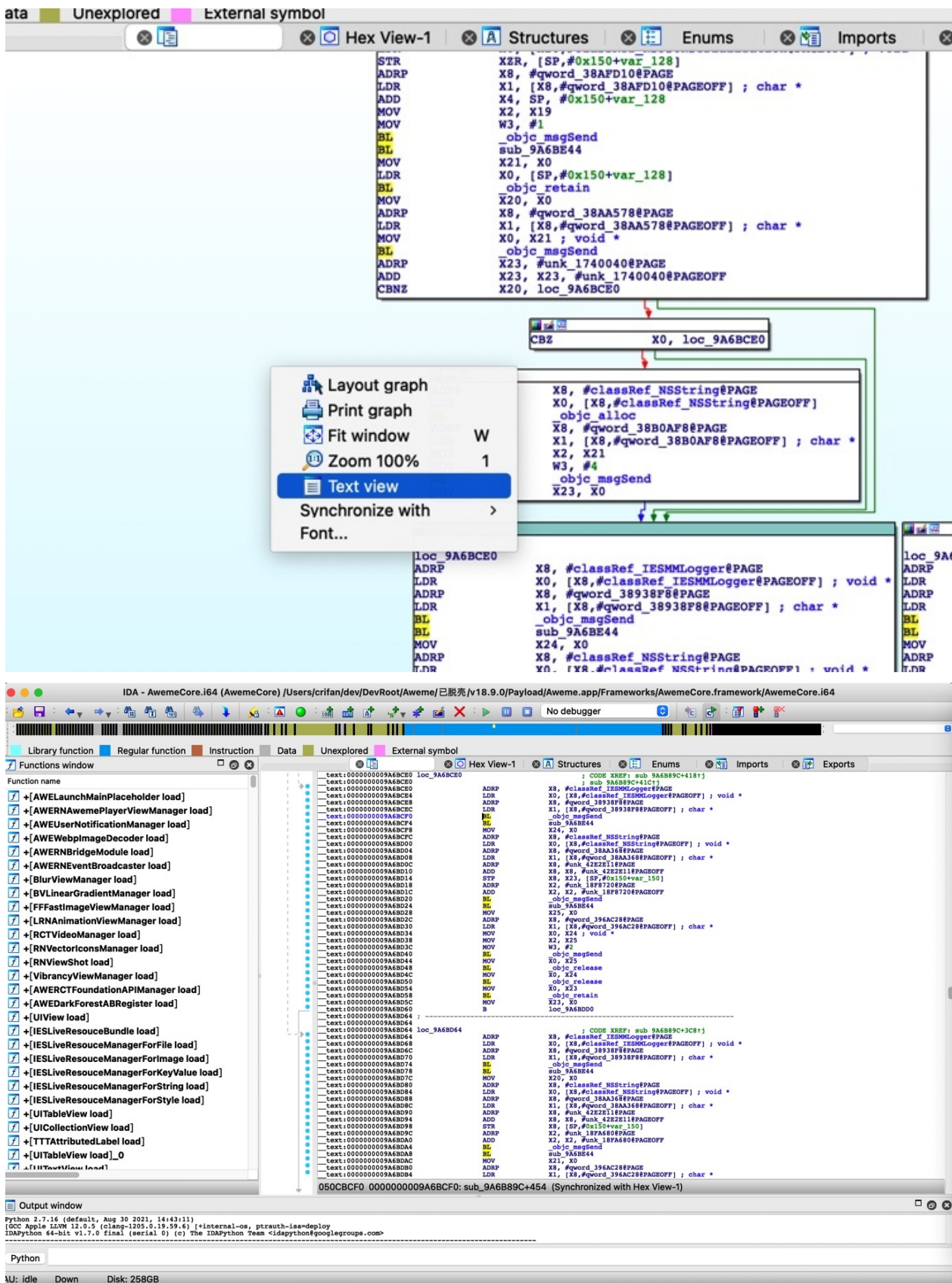


这个叫做：Graph View =图形视图

好处：方便看函数调用的逻辑关系

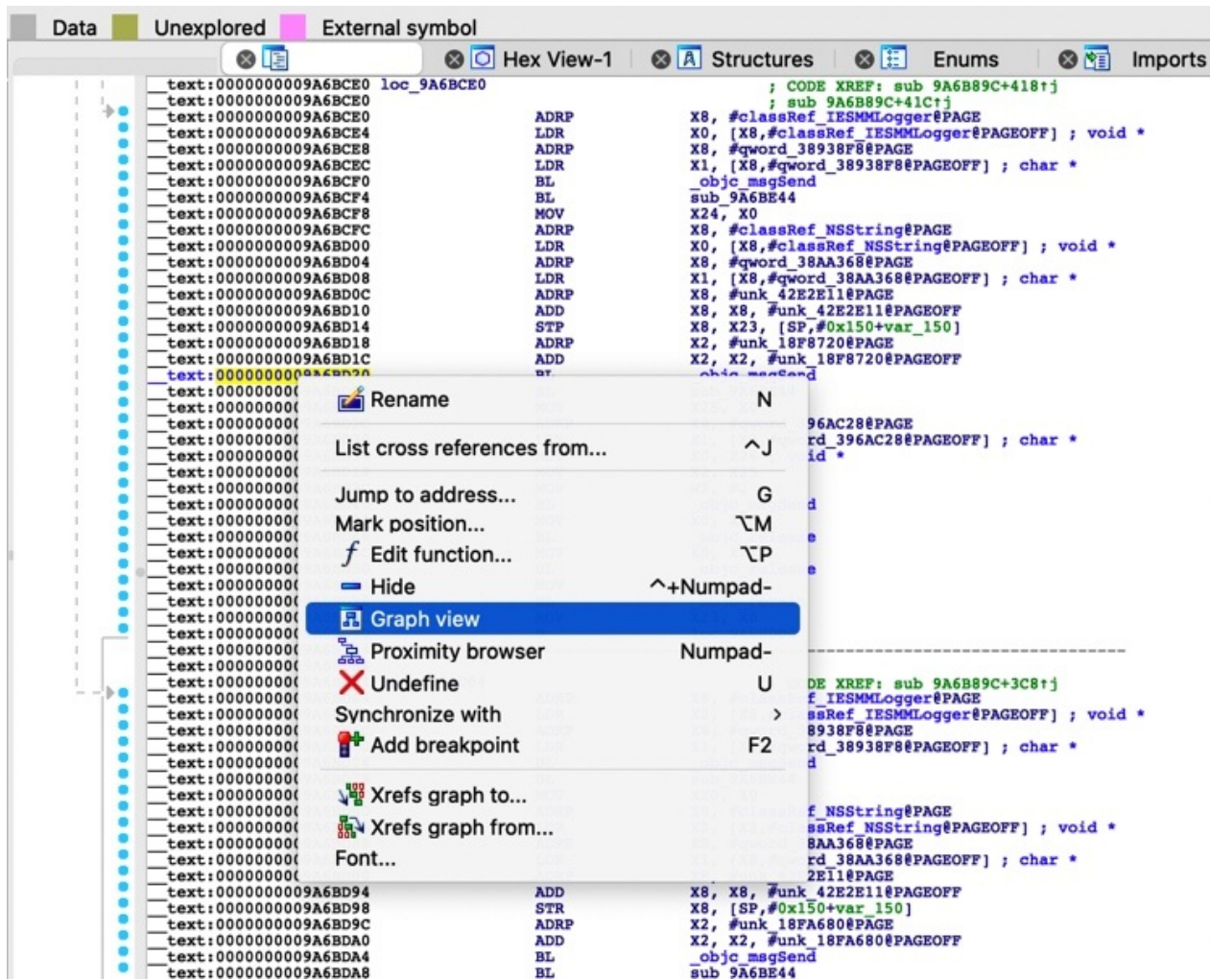
如果想要切换到： 文本模式 ， 汇编模式

则可以： 右键 -> Text View

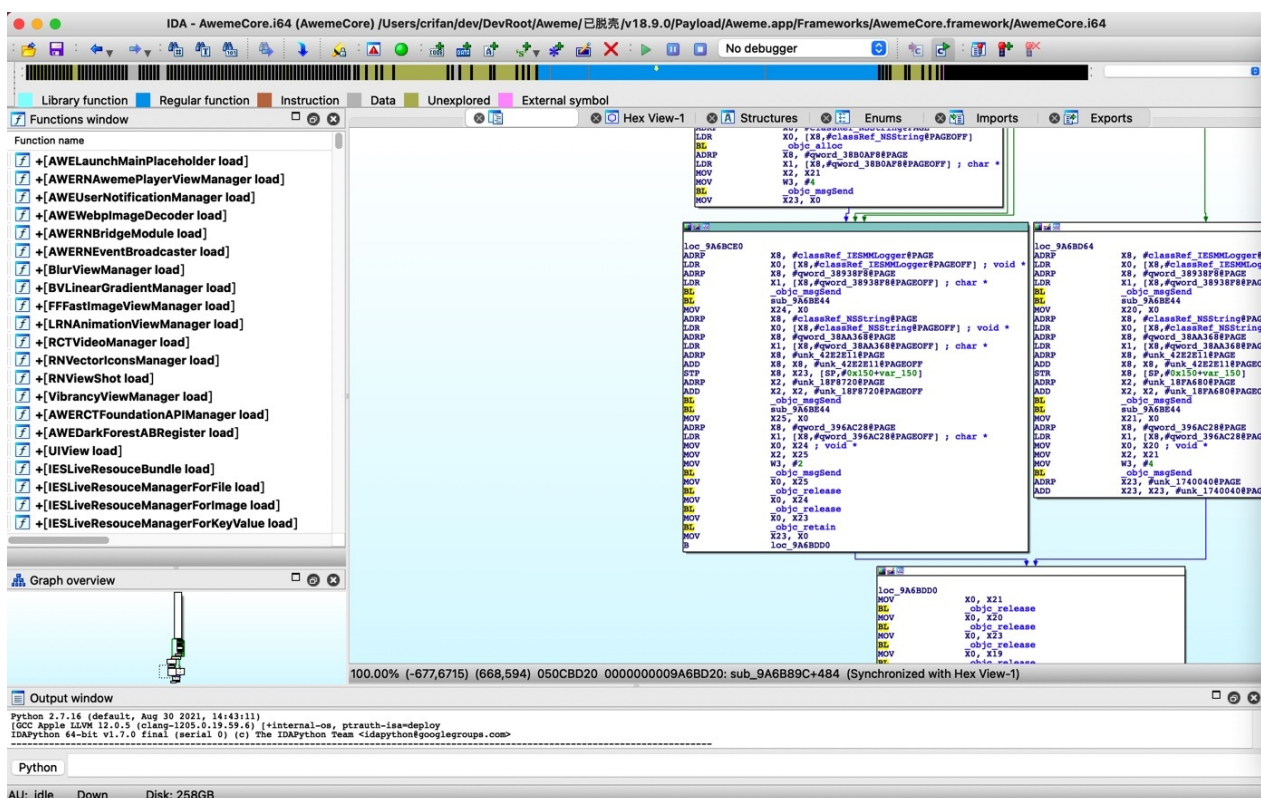


同理，从Text View想要切换到Graph View，也可以：

右键 -> Graph View

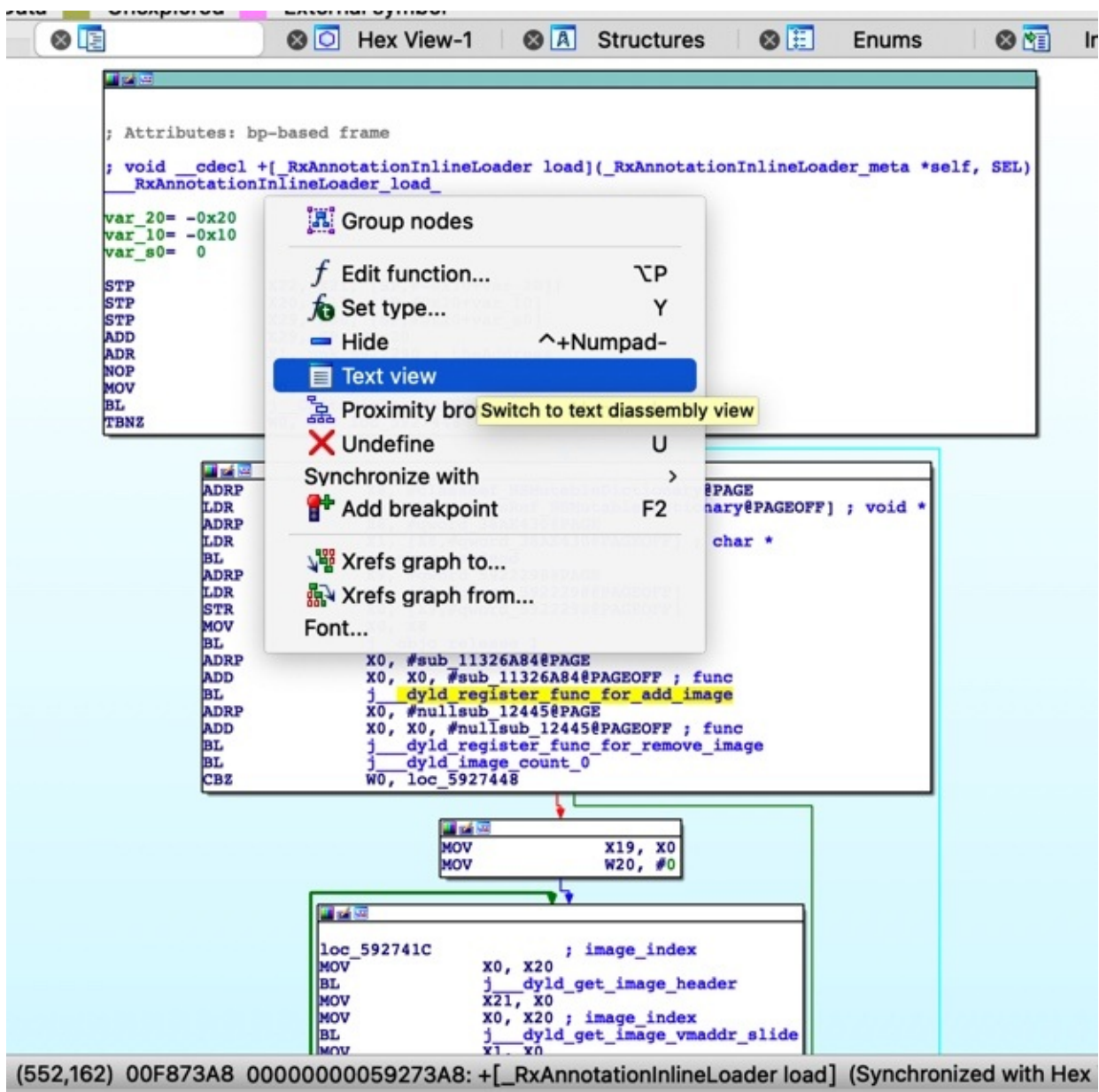


回到了: graph视图

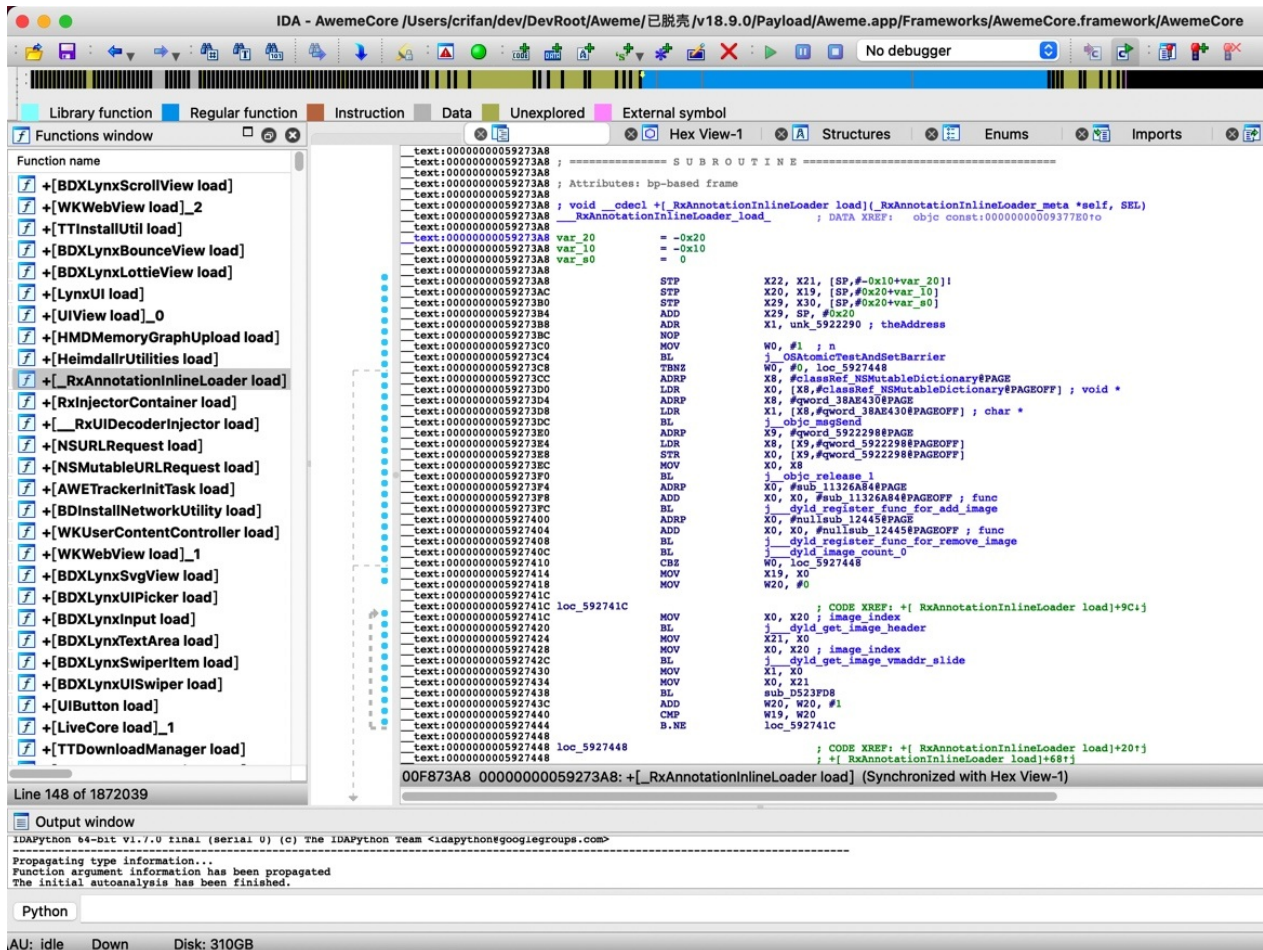


函数调用图 和 Text View

从函数调用图 切换到原先汇编指令形式：

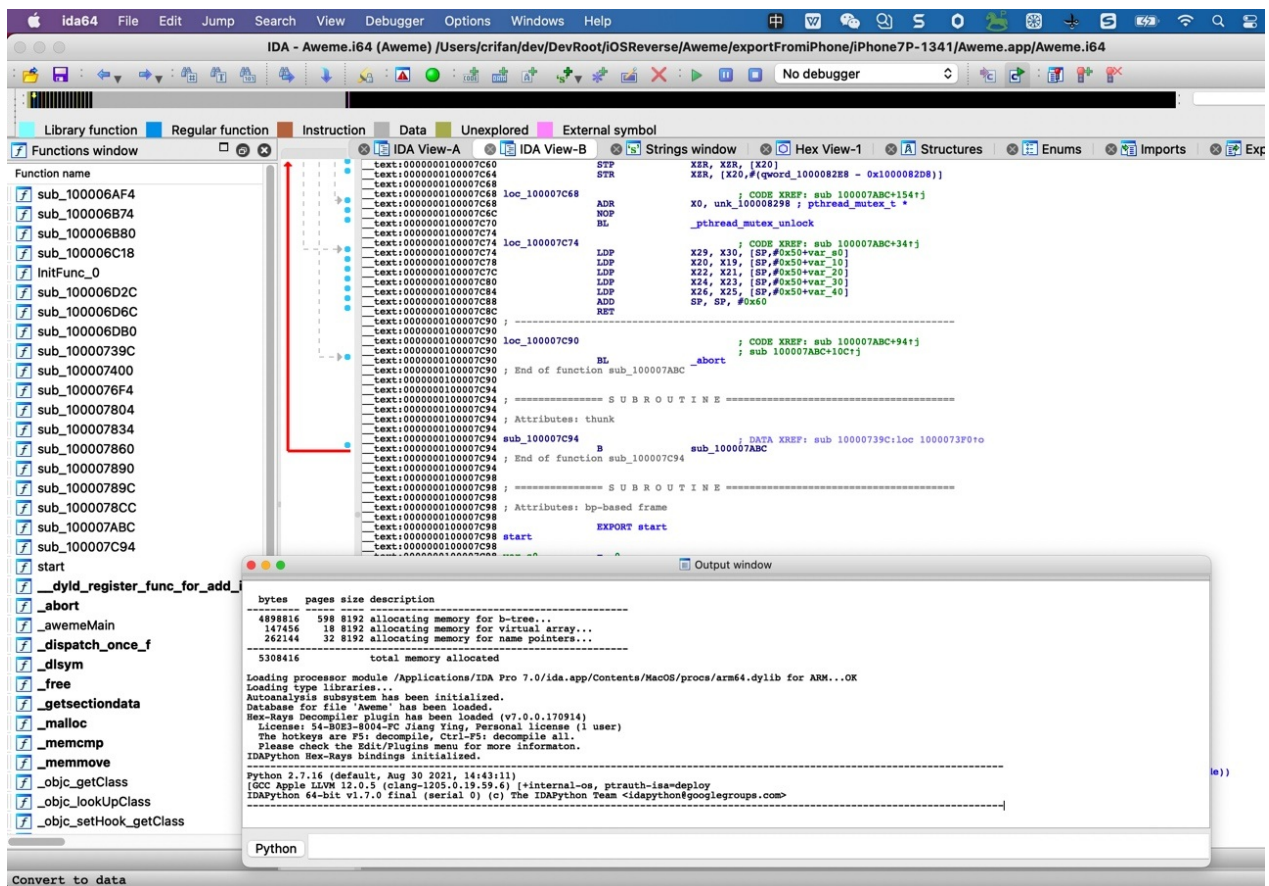


切换到： IDA View



如何把浮动窗口 Output window 固定到底部

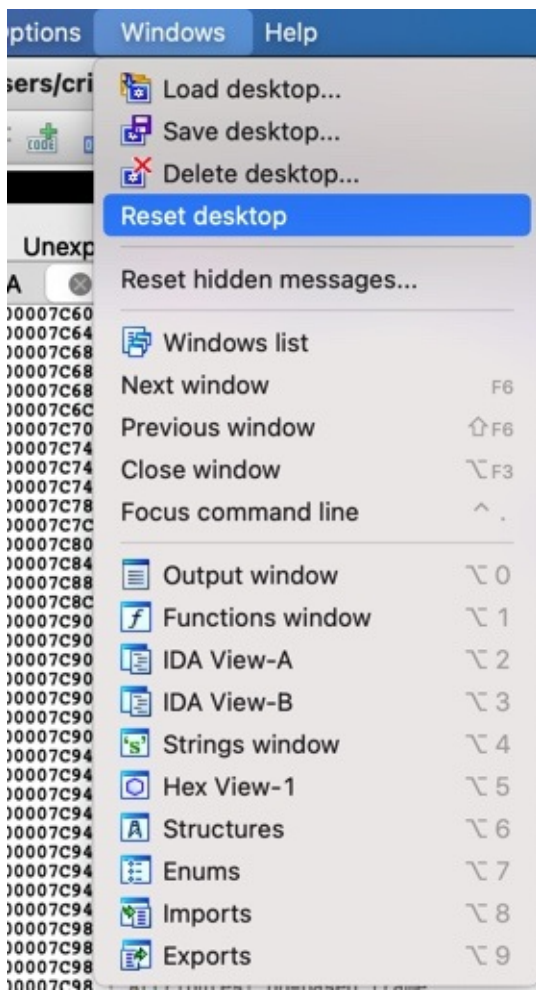
问题: IDA中的 output window , 不知何故, 悬浮在主体窗口上面了:



希望是：能固定到底部或左下角等位置

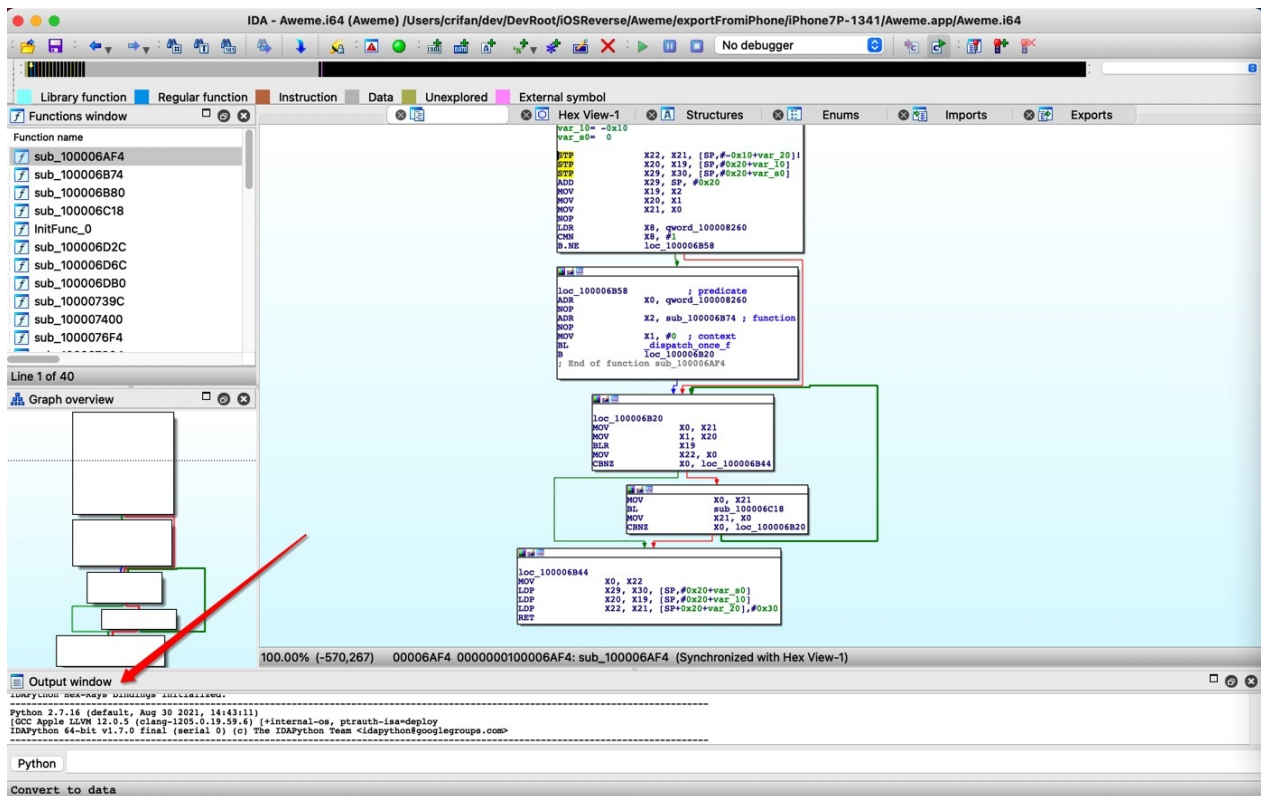
解决办法：

IDA -> Window -> Reset desktop



可以恢复IDA桌面原始布局

-> 从而使得此处的 Output Window，恢复最初的位置=固定在最底部的位置



命名和含义

TODO:

-
- 【已解决】iOS逆向心得：如何从对x8的adrp和ldr计算出对应的qword字符串值

此处整理IDA中，各处看到的，各种名称的命令规则的含义。

背景知识

- 动态链接库 文件类型=后缀
 - Windows : .dll
 - Linux : .so
 - Mac : .dylib

大的类型

- F = Function =函数
 - regular function, which is not a library function.
- L = Library =库
 - library function that can be recognized with different signatures that are part of IDA. If the matching signature is not found, the name is labeled as a regular function.
- I = Imported =导入 (符号=函数)
 - imported name from the shared library. The code from this function/name is not present in the executable and is provided at run time, whereas the library function is embedded into the executable.
- C = Code =代码
 - named code that represent program locations that are not part of any function, which can happen if the name is a part of the symbol table, but the executable never calls this function.
- D = Data =数据
 - named data locations that are usually global variables.
- A = Ascii = (ASCII) 字符串
 - ASCII string data that represents a string terminated with a null byte in the executable.

命名规则

IDA中，对于未命令的内容，会采用默认从缩写命名。其命名规则是：

- IDA常见命名
 - sub = subroutine = 子程序 : 函数
 - locret : 返回指令
 - loc : location= (代码的) 位置
 - off = offset : 某个偏移量，存放某个数据
 - seg = segment : 数据，包含段地址值
 - asc = ascii : 数据，ASCII字符串
 - byte : 数据，字节 (或字节数组)
 - word : 数据，16位数据 (或字数组)
 - dword : 数据，32位数据 (或双字数组)
 - qword : 数据，64位数据 (或4字数组)

- `flt` : 浮点数据, 32位 (或浮点数组)
- `dbl` : 浮点数, 64位 (或双精度数组)
- `tbyte` : 浮点数, 80位 (或扩展精度浮点数)
- `stru = structure` : 结构体(或结构体数组)
- `algn = align` : 对齐指示
- `unk = unknown` : 未处理字节
- 字节相关
 - `db = 1`个字节
 - `b=byte`
 - `dw = 2`个字节
 - `w=word`
 - `dd = 4`个字节
 - `d=double`

crifan.org, 使用署名4.0国际(CC BY 4.0)协议发布 all right reserved, powered by Gitbook最后更新: 2024-10-01 16:12:06

- Unexpl. External symbol

Hex View-1

Occurrences of binary: 01 10 00 D4

Structures

Enu

```
text:00000000B7CBB8C ; End of function sub_B7CBA20
text:00000000B7CBB8C
text:00000000B7CBB90
text:00000000B7CBB90 ; ===== SUBROUTINE =====
text:00000000B7CBB90 ; DATA XREF: sub_B7CBB90+3C+0
text:00000000B7CBB90 sub_B7CBB90 CBZ X0, loc_B7CBB4C
text:00000000B7CBB94 LDP X1, X2, [SP]
text:00000000B7CBB98 LDP X3, X4, [SP, #0x10]
text:00000000B7CBB9C LDP X5, X6, [SP, #0x20]
text:00000000B7CBBA0 LDR X7, [SP, #0x30]
text:00000000B7CBBA4 SUB X0, X0, #0xE9
text:00000000B7CBBAC MOV X16, #0
text:00000000B7CBBB0 SVC 0x80
text:00000000B7CBBB0 ; locret_B7CBC00
text:00000000B7CBBB4 DCB 0x62 ; b
text:00000000B7CBBB5 DCB 0x79 ; y
text:00000000B7CBBB6 DCB 0x20
text:00000000B7CBBB7 DCB 0x6C ; l
text:00000000B7CBBB8 DCB 0x69 ; i
text:00000000B7CBBB9 DCB 0x7A ; z
text:00000000B7CBBBA DCB 0x68 ; h
text:00000000B7CBBBB DCB 0x69 ; i
text:00000000B7CBBBC DCB 0x2E ; .
text:00000000B7CBBBD DCB 0x63 ; c
text:00000000B7CBBBE DCB 0x7A ; z
text:00000000B7CBBBF DCB 0x6C ; l
text:00000000B7CBBC0 DCB 0x20
text:00000000B7CBBC1 DCB 0x30 ; 0
text:00000000B7CBBC2 DCB 0x2E ; .
text:00000000B7CBBC3 DCB 0x31 ; l
text:00000000B7CBB4C loc_B7CBB4C LDP X1, X2, [SP]
text:00000000B7CBB4C LDP X4, X3, [SP, #0x10]
text:00000000B7CBB4C ADR X3, sub_B7CBB90
text:00000000B7CBBD0 MADD X1, X1, X2, XZR
text:00000000B7CBBD4 MOV X2, X4
text:00000000B7CBBD8 ADD X2, X3, X2
text:00000000B7CBBDC ADD X2, X2, X1
text:00000000B7CBBE0 LDR X2, [X2]
text:00000000B7CBBE4 loc_B7CBBE4 CMP X0, X4
text:00000000B7CBBE4 B.EQ loc_B7CBBFC
text:00000000B7CBBEC LDR X1, [X3, X0]
text:00000000B7CBBF0 EOR X2, X2, X1
text:00000000B7CBBF4 ADD X0, X0, #4
text:00000000B7CBBF8 B loc_B7CBBE4
text:00000000B7CBBFC ; =====
text:00000000B7CBBFC loc_B7CBBFC MOV X0, X2 ; CODE XREF: sub_B7CBB90+58+j
text:00000000B7CBBFC
text:00000000B7CBB00 locret_B7CBC00 RET ; CODE XREF: sub_B7CBB90+20+j
text:00000000B7CBB00 ; End of function sub_B7CBB90
text:00000000B7CBB00
text:00000000B7CBB04 DCB 0x20
text:00000000B7CBB05 DCB 0
text:00000000B7CBB06 DCB 0x20
text:00000000B7CBB07 DCB 0xD4
text:00000000B7CBB08 ; ===== SUBROUTINE =====
text:00000000B7CBB08
text:00000000B7CBB08
```

06E2BBB0 00000000B7CBBB0: sub_B7CBB90+20 (Synchronized with Hex View-1)

- 24

- o qword
 - qword_3A97BE0

Hex View-1

Occurrences of binary: 01 10 00 D4

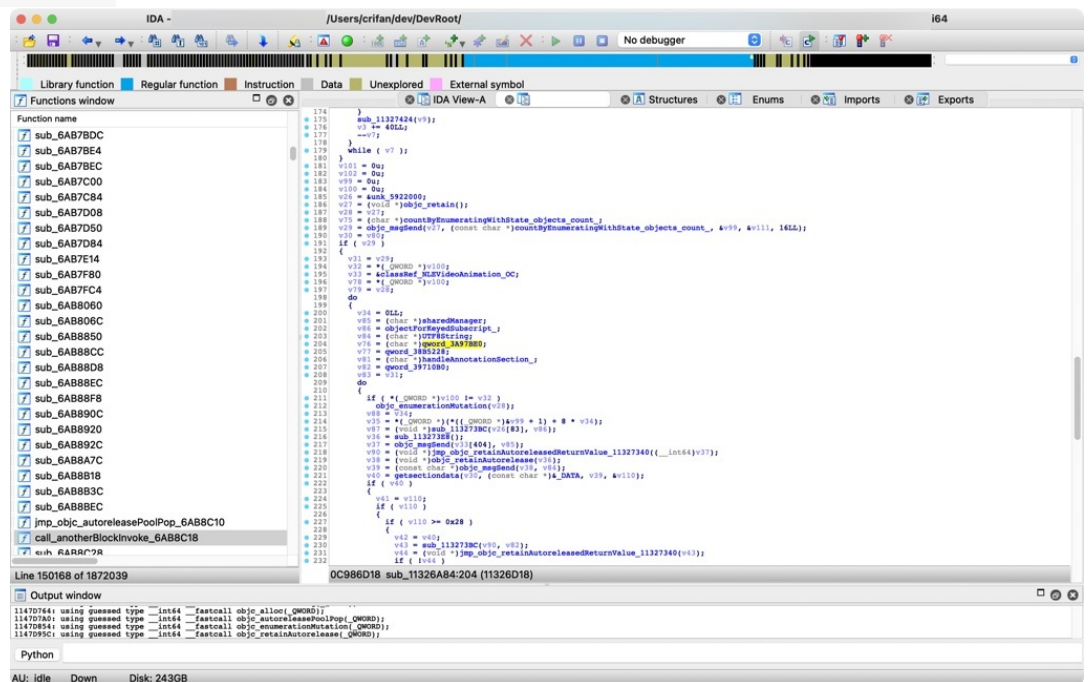
Structures

```

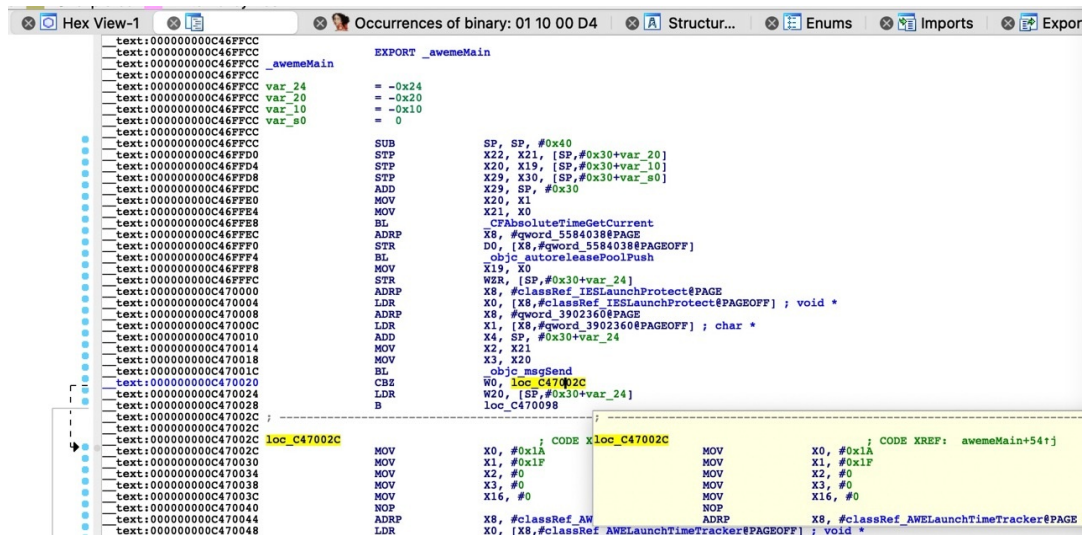
text:000000000C470008      ADRP          X8, #qword_3902360@PAGE
text:000000000C47000C      LDR           X1, [X8,#qword_3902360@PAGEOFF]
text:000000000C470010      ADD          X4, SP, #0xC
text:000000000C470014      MOV          X2, X21
text:000000000C470018      MOV          X3, X20
text:000000000C47001C      BL           _objc_msgSend
text:000000000C470020      CBZ          W0, loc_C47002C
text:000000000C470024      LDR          W20, [SP,#0xC]
text:000000000C470028      B            loc_C470098
; -----
text:000000000C47002C      loc_C47002C      ; CODE XREF: awemeMain+54;j
text:000000000C47002C      MOV          X0, #0x1A
text:000000000C470030      MOV          X1, #0x1F
text:000000000C470034      MOV          X2, #0
text:000000000C470038      MOV          X3, #0
text:000000000C47003C      MOV          X16, #0
text:000000000C470040      SVC          0x80
text:000000000C470044      ADRP         X8, #classRef_AWELaunchTimeTracker@PAGE
text:000000000C470048      LDR          X0, [X8,#classRef_AWELaunchTimeTracker@PAGEOFF]
text:000000000C47004C      ADRP         X8, #qword_38938F8@PAGE
text:000000000C470050      LDR          X1, [X8,#qword_38938F8@PAGEOFF]
text:000000000C470054      BL           _objc_msgSend
text:000000000C470058      MOV          X29, X29
text:000000000C47005C      BL           _objc_retainAutoreleasedReturnValue
text:000000000C470060      MOV          X22, X0
text:000000000C470064      ADRP         X8, #qword_38AA608@PAGE
text:000000000C470068      LDR          X1, [X8,#qword_38AA608@PAGEOFF]
text:000000000C47006C      MOV          W2, #4
text:000000000C470070      BL           _objc_msgSend
text:000000000C470074      MOV          X0, X22
text:000000000C470078      BL           _objc_release
text:000000000C47007C      ADRP         X3, #unk_196E8C0@PAGE
text:000000000C470080      ADD          X3, X3, #unk_196E8C0@PAGEOFF
text:000000000C470084      MOV          X0, X21
text:000000000C470088      MOV          X1, X20
text:000000000C47008C      MOV          X2, #0
text:000000000C470090      BL           UIApplicationMain
text:000000000C470094      MOV          X20, X0
; -----
text:000000000C470098      loc_C470098      ; CODE XREF: awemeMain+5C;j
text:000000000C470098      MOV          X0, X19
text:000000000C47009C      BL           _objc_autoreleasePoolPop
text:000000000C4700A0      MOV          X0, X20
text:000000000C4700A4      LDP          X29, X30, [SP,#0x30]
text:000000000C4700A8      LDP          X20, X19, [SP,#0x20]
text:000000000C4700AC      LDP          X22, X21, [SP,#0x10]
text:000000000C4700B0      ADD          SP, SP, #0x40
text:000000000C4700B4      RET
; End of function _awemeMain

```

- o loc
 - loc_C47002C



- o loc
 - loc_C47002C



详解：

sub函数

关于sub函数的一些细节说明：

- sub_xxx：普通的函数（有代码处理逻辑的）
 - 比如去改名的话，可以改名叫做：
 - sub_BinaryOffset
 - sub_AddressInsideBinary
- nullsub_xxx：空函数（没有任何代码逻辑的）
 - IDA中关于空函数的介绍

Rename empty functions as nullsub_...
This option allows IDA to rename empty functions containing only a "return" instruction as "nullsub_..."
(... is replaced by a serial number: 0,1,2,3...)

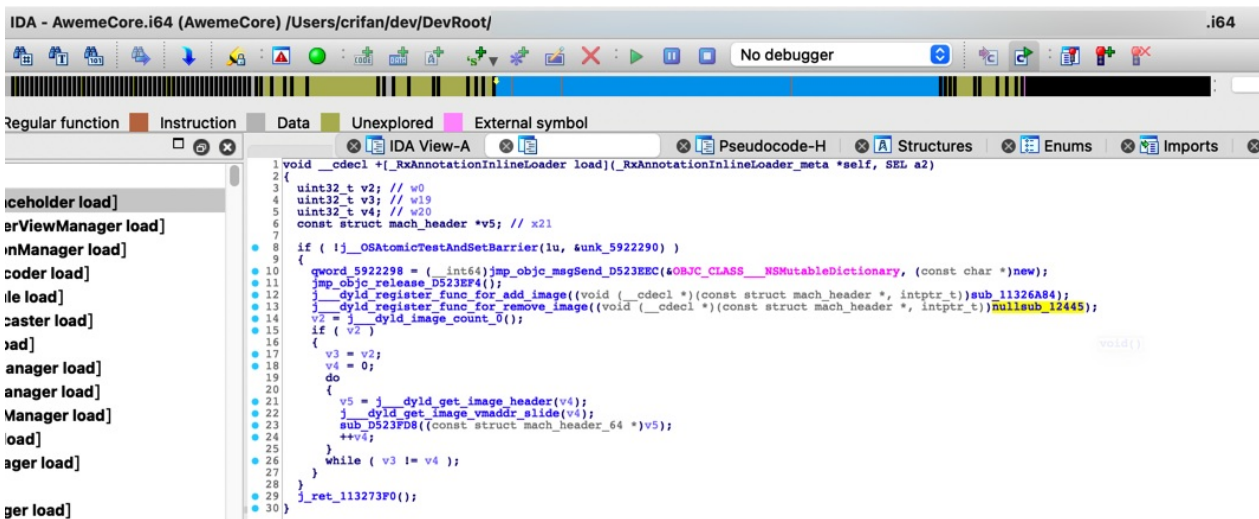
此处给出实例：

举例：

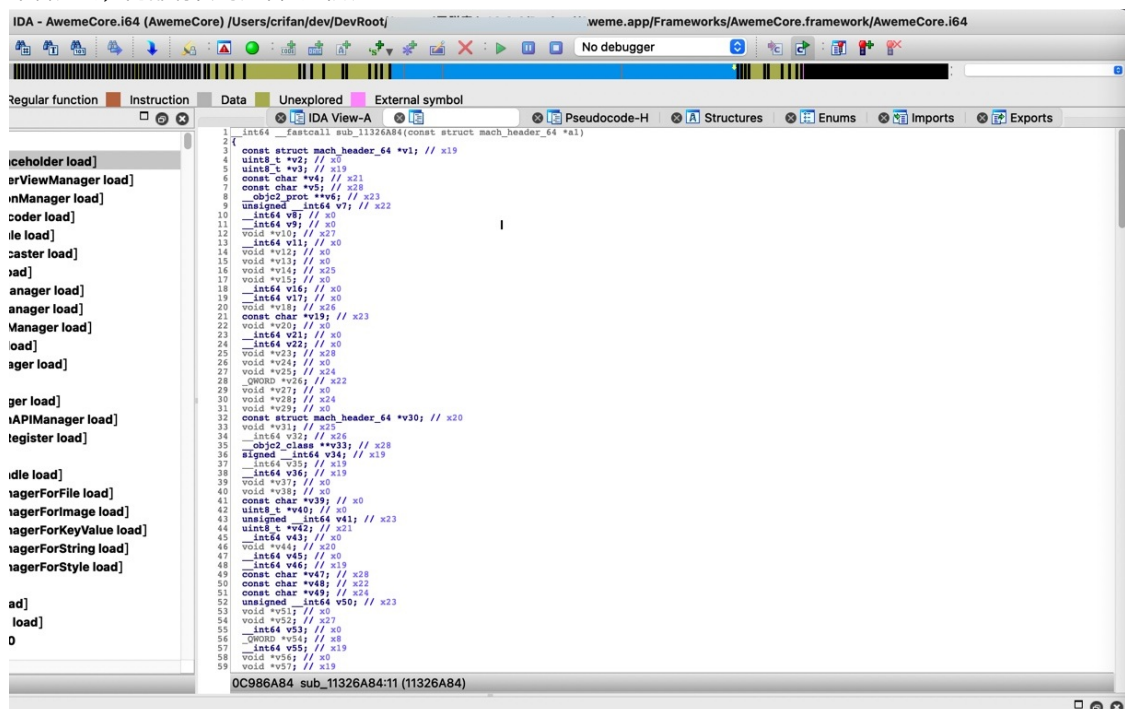
【未解决】研究抖音越狱检测逻辑：_RxAnnotationInlineLoader的load

中的：

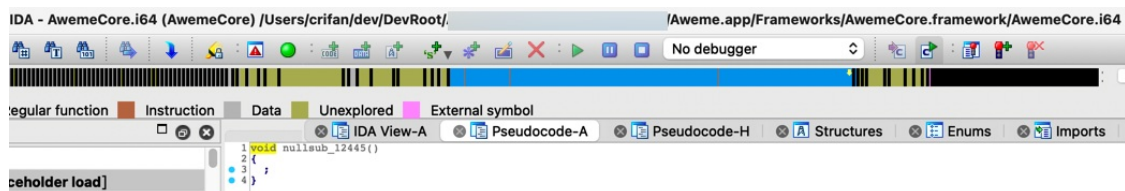
```
void __cdecl +[ _RxAnnotationInlineLoader load ]( _RxAnnotationInlineLoader_meta self, SEL a2 )
{
    ...
    j_dyld_register_func_for_add_image((void (__cdecl *) (const struct mach_header *, intptr_t)) sub_11326A84);
    j_dyld_register_func_for_remove_image((void (__cdecl *) (const struct mach_header *, intptr_t)) nullsub_12445);
}
```



- `_dyld_register_func_for_add_image`传入的函数： `sub_11326A84`
 - 就是个普通的，内部有代码逻辑的函数：



- `_dyld_register_func_for_remove_image`传入的函数： `nullsub_12445`
 - 从名字看，就知道：是个null的 空的 函数
 - 进入看，果然是空的，啥也没有



qword

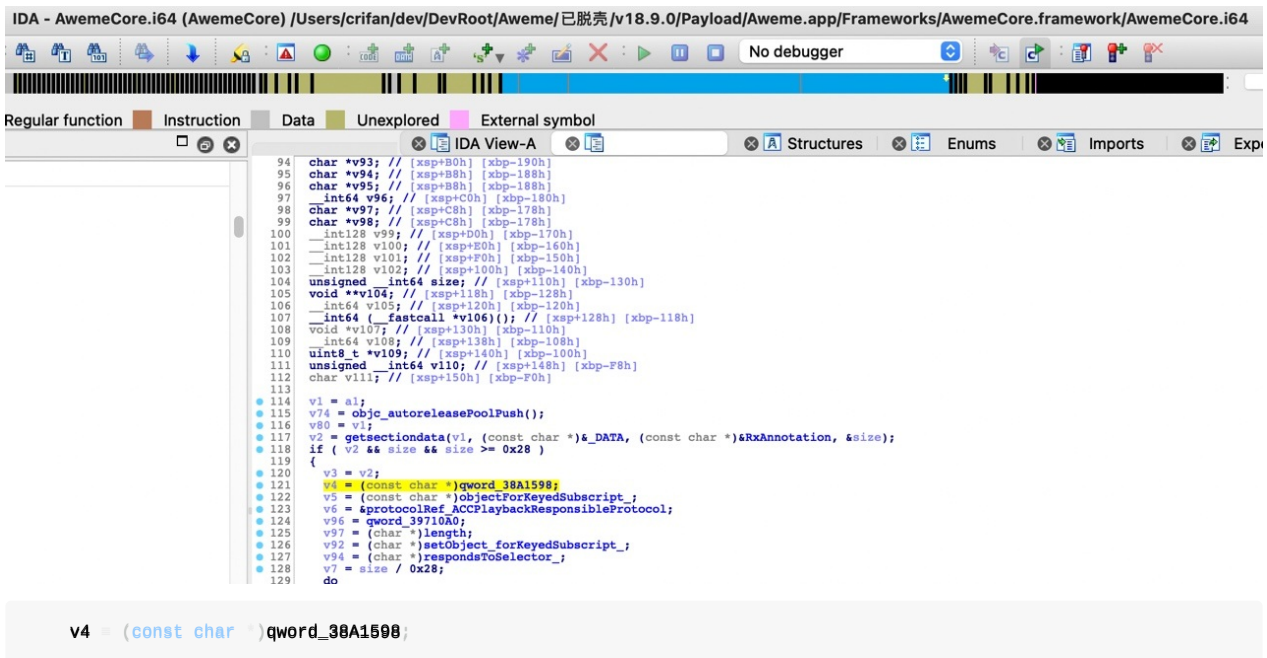
对于qword：

- 常常是：常量字符串
- 偶尔是：其他类型

- 比如字典的指针等等
- 核心逻辑是：
 - `qword_xxx`的`xxx`是二进制内偏移量 + 二进制的ALSR = 实际（字符串的）地址
 - 去查看：实际（字符串的）地址 = （即可查看到）保存了对应的字符串

此处举例说明：

IDA伪代码：



计算 `qword_38A1598` 的实际的值是什么

而通过此处的类型强制转换 (`const char *`) 可以看出是个字符串，所以此处就是去看看：到底字符串的值是什么

先查看当前二进制的ALSR基地址：

```

(lldb) image list o f | grep AwemeCore
[ 0] 0x0000000100a98000 Users crifan Library Developer Xcode DerivedData Aweme fswcidjoxbkibsdwekuzlsfcdqls Build P
roducts Debug iphoneos Aweme app Frameworks AwemeCore.framework AwemeCore

```

再继续计算此处的字符串的值：

```

(lldb) p x 0x0000000100a98000 + 0x38A1598
(long) $7 = 0x0000000104339598
(lldb) x /gx 0x0000000104339598
0x104339598: 0x000000010185c42e
(lldb) po (char *)0x000000010185c42e
"dynamicCast;"

```

unk

unk 本身是IDA伪代码解析后，不知道变量具体的值什么类型，所以无法给出更加精准的变量命令。

此处，通过具体例子来介绍，如何调试和计算真实的值

unk_3F852B3是字符串

比如：

```
__int64 __fastcall sub_11326A84(const struct mach_header_64 a1)
{
    * * *
    v2 = getsectiondata(v1, (const char *) unk_3F852B3, (const char *) unk_47DC9B8, &size);
    if ( v2 && size && size >= 0x28 )
    {
        * * *
        v4 = (const char *)qword_38A1598;
```

其中的：unk_3F852B3

由名字可知，该变量的地址是：3F852B3

IDA中通过地址可以查看到内容=定义是：

```
_D_cstring 000000003F852B3 unk_3F852B3 % 1 ; DATA XREF sub_5B77470 3C10
```

是看不出具体类型和具体的值的。

不过，此处动态调试期间，可以根据地址算出来：

unk_RelativeAddress -> ALSR 基地址 + RelativeAddress = 当前内存地址

先去看：ALSR基地址 = 二进制加载的首地址 = 此处是抖音的 AwemeCore 加载的首地址 = 如前面已计算出是 0x0000000100a98000

计算过程：

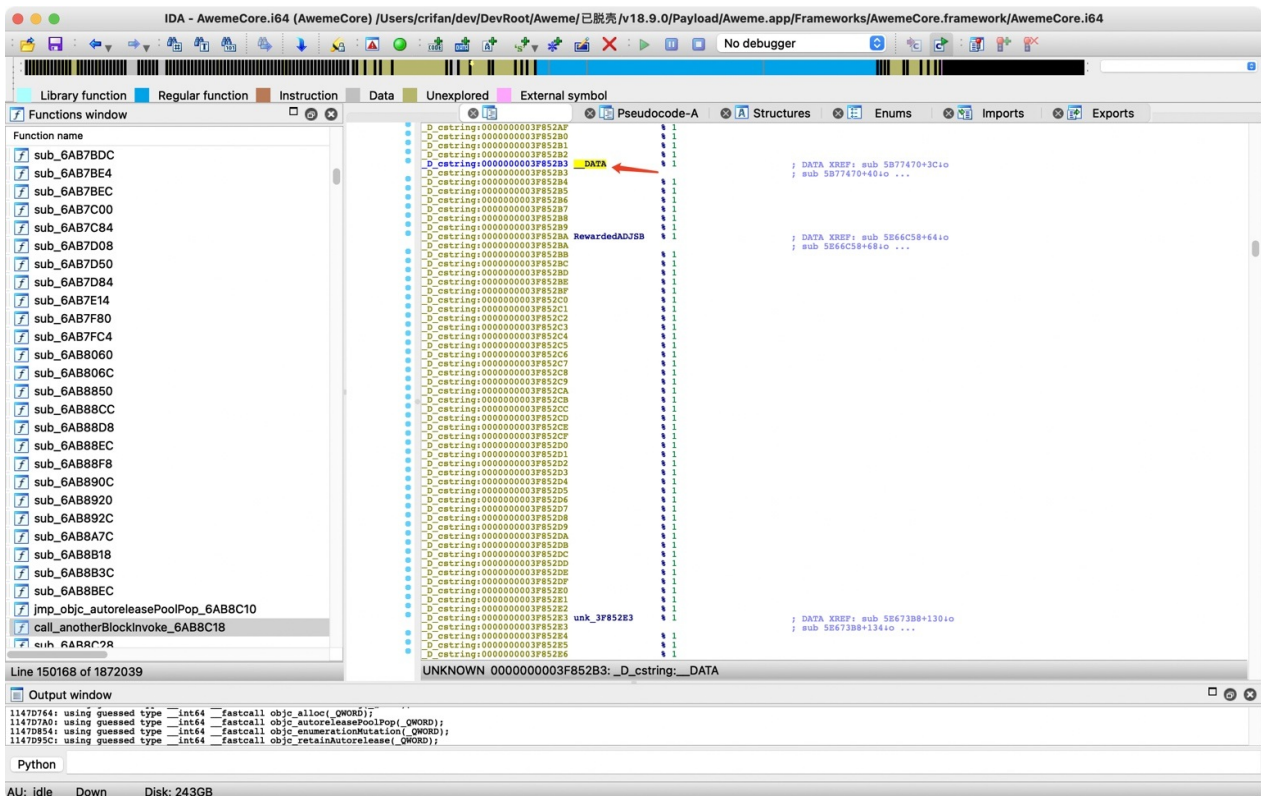
```
(lldb) p/x 0x0000000100a98000 + 0x3F852B3
(long) $0 = 0x0000000104a1d2b3
(lldb) po (char*)0x0000000104a1d2b3
"__DATA"

(lldb) x/lgx 0x0000000104a1d2b3
0x104a1d2b3: 0x5200415441445f5f
(lldb) po (char*)0x5200415441445f5f
""
```

所以才看出此时：

unk_3F852B3 == "__DATA"

所以再去优化IDA中的代码：把 unk_3F852B3 改名为 __DATA



以及，IDA伪代码中，也去刷新即可看到新的变量名：

```
v2 = getsectiondata(v1, (const char *)_DATA, (const char *) RxAnnotation, size);
if ( v2 && size && size >= 0x28 )
```

-> 从而使得代码逻辑稍微更加清晰了一点。

-> 类似的，别的 unk 的地址，也可以计算出调试时的内存地址，从而查看到是什么值

- unk_3F852BA

```
(lldb) p x 0x0000000100a98000 + 0x3F852BA
(long) $3 = 0x0000000104a1d2ba
(lldb) po (char*)0x0000000104a1d2ba
"RewardedADJSB"
```

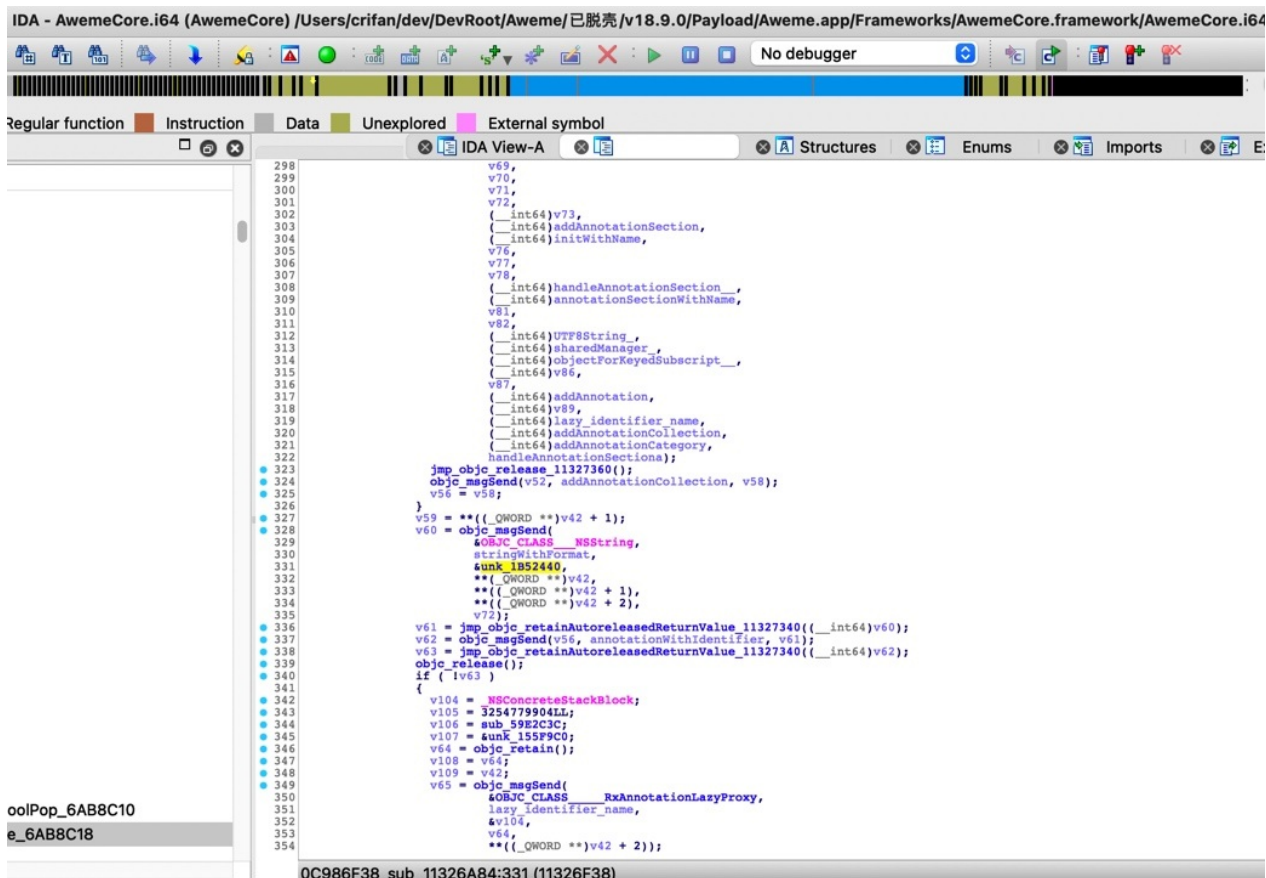
- unk_47DC9B8

```
(lldb) p x 0x0000000100a98000 + 0x47DC9B8
(long) $5 = 0x00000001052749b8
(lldb) po (char*)0x00000001052749b8
"RxAnnotation"
```

unk_1B52440是字符串

再给出一个例子，介绍如何计算unk的值：

IDA伪代码：



```

v60 = objc_msgSend(
    objc_class___NSString,
    stringWithFormat,
    unk_1B52440,
    **((_QWORD **v42 + 1)),
    **((_QWORD **v42 + 1)),
    **((_QWORD **v42 + 2)),
    v72);

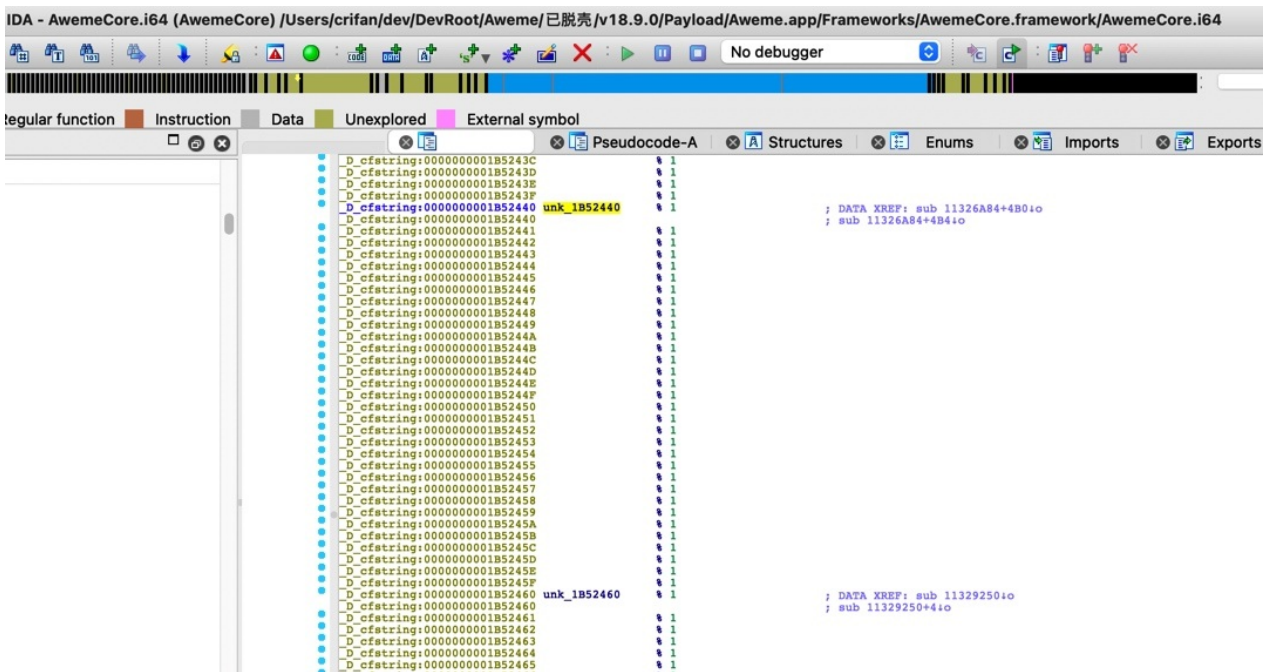
```

IDA中定义是：

```

_D_cfstring 000000001B52440 unk_1B52440 1 ; DATA XREF: sub_11326A84 B0:0

```

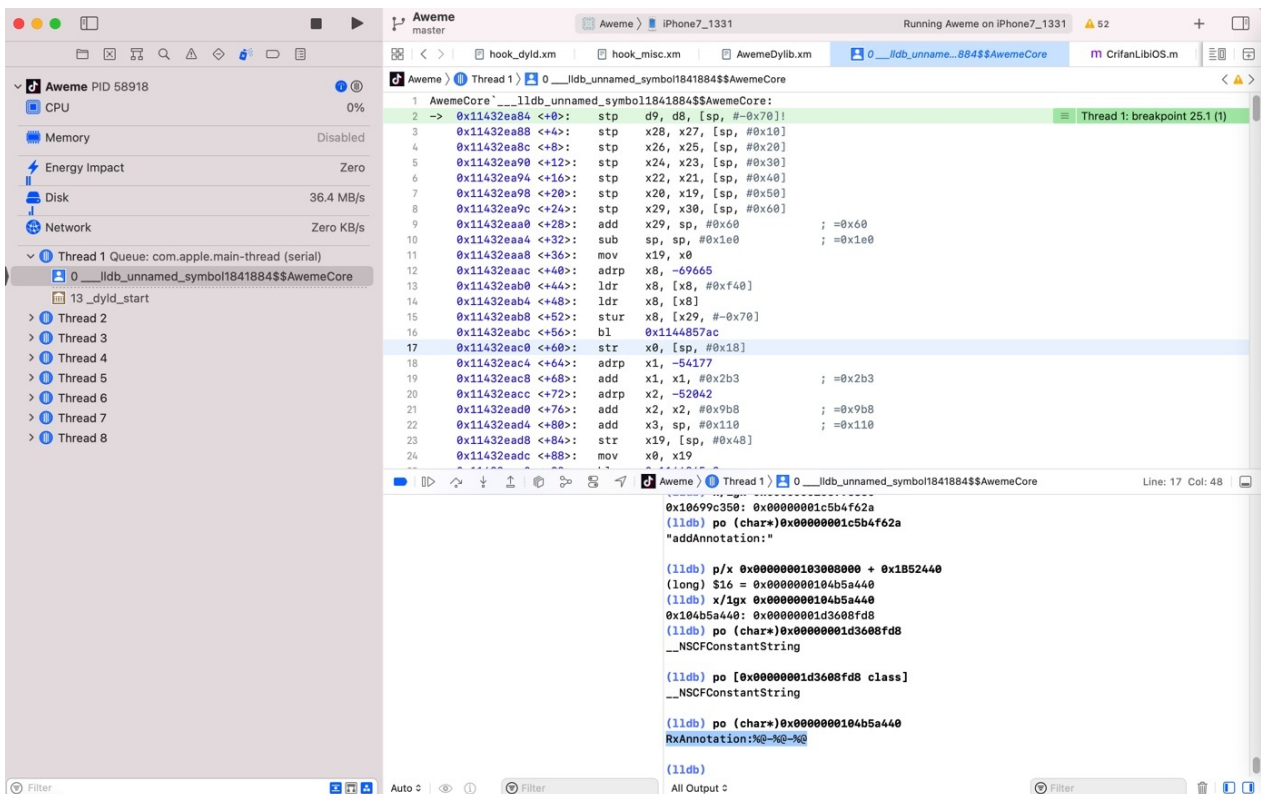



注意到其中的 `_D_cfstring = constant string` ?

去计算值:

```
(lldb) p/x 0x0000000103008000 + 0x1B52440
(long) $16 = 0x0000000104b5a440

(lldb) po (char*)0x0000000104b5a440
RxAnnotation:%%-%%-%%
```



即:

`unk_1B52440 = constant string` 常量字符串: `RxAnnotation:%%-%%-%%`

然后去优化IDA伪代码, 把 `unk_1B52440` 改为更容易读懂的名字, 比如 `RxAnnotation_percentAt_percentAt_percentAt`。

另外类似的例子：

- unk_1942B60

IDA伪代码：

```
jmp_objc_msgSend_D523EEC(HMDNetworkReqModelObj, (const char *)setMethod_, unk_1942B60);
```

IDA中定义：

```
_D_cfstring 000000001942B60 unk_1942B60 % 1 ; DATA XREF: sub_59FA0FC 14!o
_D_cfstring 000000001942B60 ; sub_59FA0FC 18!o ...
```

计算逻辑：

```
(lldb) p x 0x0000000105140000 + 0x1942B60
(long) $22 = 0x0000000106a82b60
(lldb) po (char *)0x0000000106a82b60
POST
(lldb) po [0x0000000106a82b60 class]
__NSCFConstantString
```

-> unk_1942B60 是 __NSCFConstantString 字符串常量: "POST"

crifan.org, 使用署名4.0国际(CC BY 4.0)协议发布 all right reserved, powered by Gitbook最后更新: 2024-10-01 16:13:30

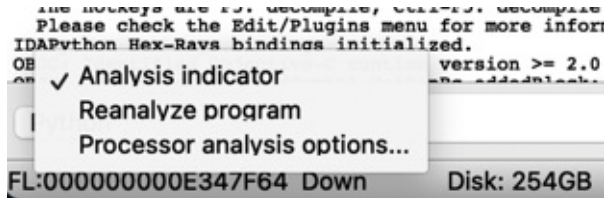
自动分析

典型过程：把二进制文件拖动到IDA后，就开始自动分析`了。

IDA中默认已开启左下角的：

- Analysis indicator = 自动分析指示器

◦ 图



◦ 作用：可以显示当前自动分析的进度

为何要（自动）分析？

我们用IDA去分析二进制中代码的逻辑。

而二进制本身其实只有 0 和 1 二进制数据而已。

而想要分析代码，即查看对应二进制对应的 汇编代码（以及后续的 伪代码），所包含的函数，所包含的字符串等等信息，则需要：

对二进制进行充分的分析，最后才能显示出我们要的上述的各种信息。

而对于二进制加载后的分析过程，IDA叫做：

- 自动分析 = auto analysis

自动分析的过程和阶段

自动分析会先后有多个阶段，其中阶段的名称是2个字母的缩写，可以在左下角看到

具体阶段和含义是：

- FL:<address> execution **F**Low is being traced
- PR:<address> a function (=P**R**ocedure?) is being created
- TL:<address> a function **T**ai**L** is being created
- SP:<address> the **S**tack **P**ointer is being traced
- AC:<address> the **A**ddress is being analyzed
- LL:<number> a signature file is being loaded
- L1:<address> the first pass (**L**evel **1**) of FLIRT
- L2:<address> the second pass (**L**evel **2**) of FLIRT
- L3:<address> the third pass (**L**evel **3**) of FLIRT
- TP:<address> **T**Ppe information is being applied
- FI:<address> the **F**inal pass of autoanalysis
- WF:<address> **W**weak execution **F**low is being traced
- AU: idle **A**utoanalysis is finished

不同阶段举例

- FL=Flow

-
- PR=PRocedure

-
- AC = Address

-
- FI=Final

-
- WF=Weak

-
- WU: idle

◦

自动分析要多长时间

自动分析完毕所需时间，取决于，二进制文件大小，以及其内部逻辑复杂程度

一般来说：

- 很小的二进制文件：耗时很短，马上就结束了
 - 举例
 - 几十KB的抖音Aweme，耗时很短，没几分钟就结束了
- 很大的二进制文件：耗时很久，有的长达数个小时
 - 举例

- 200多MB的抖音AwemeCore, IDA自动解析最终耗时, 大约12个小时

crifan.org, 使用署名4.0国际(CC BY 4.0)协议发布 all right reserved, powered by Gitbook最后更新: 2024-10-03 10:47:50

搜索

TODO:

- **【整理】** IDA使用心得: search搜索查找

IDA中的搜索，可以用于各种地方，包括函数列表，字符串列表，全局搜索，等等。

其中和搜索相关，有些通用的逻辑，此处解释一下。

crifan.org，使用[署名4.0国际\(CC BY 4.0\)协议](#)发布 all right reserved, powered by Gitbook最后更新： 2022-10-23 22:06:14

IDA功能详解

此处整理IDA中各种强大且好用的功能。

- 字符串
- 函数列表
- 查看代码
 - 汇编代码
 - 伪代码
 - F5查看伪代码
 - 导出伪代码
 - 函数调用关系
 - 结构体：设置好类的定义，伪代码自动解析出属性调用
- 导入和导出
- 插件
 - keypatch

crifan.org, 使用署名4.0国际(CC BY 4.0)协议发布 all right reserved, powered by Gitbook最后更新: 2022-10-24 11:44:07

查看代码

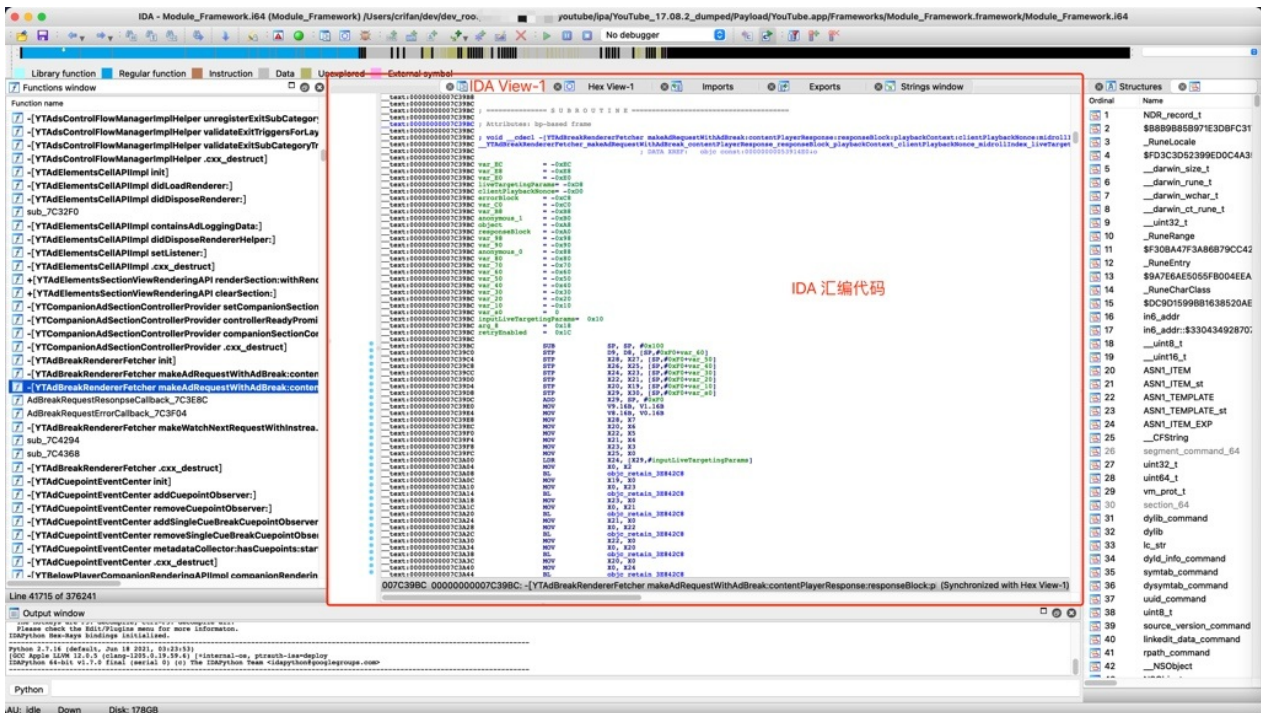
一般主要用IDA来查看和分析代码，代码主要分：

- 汇编代码：从原始二进制的字节码，反汇编得到的汇编代码
- 伪代码：从汇编代码用 F5 反编译得到的，很接近人类写的代码，人类能读懂代码逻辑的代码

crifan.org，使用署名4.0国际(CC BY 4.0)协议发布 all right reserved，powered by Gitbook最后更新：2022-10-23 21:16:18

汇编代码

IDA中的 汇编代码，是从原始二进制的字节码，反汇编得到的汇编代码



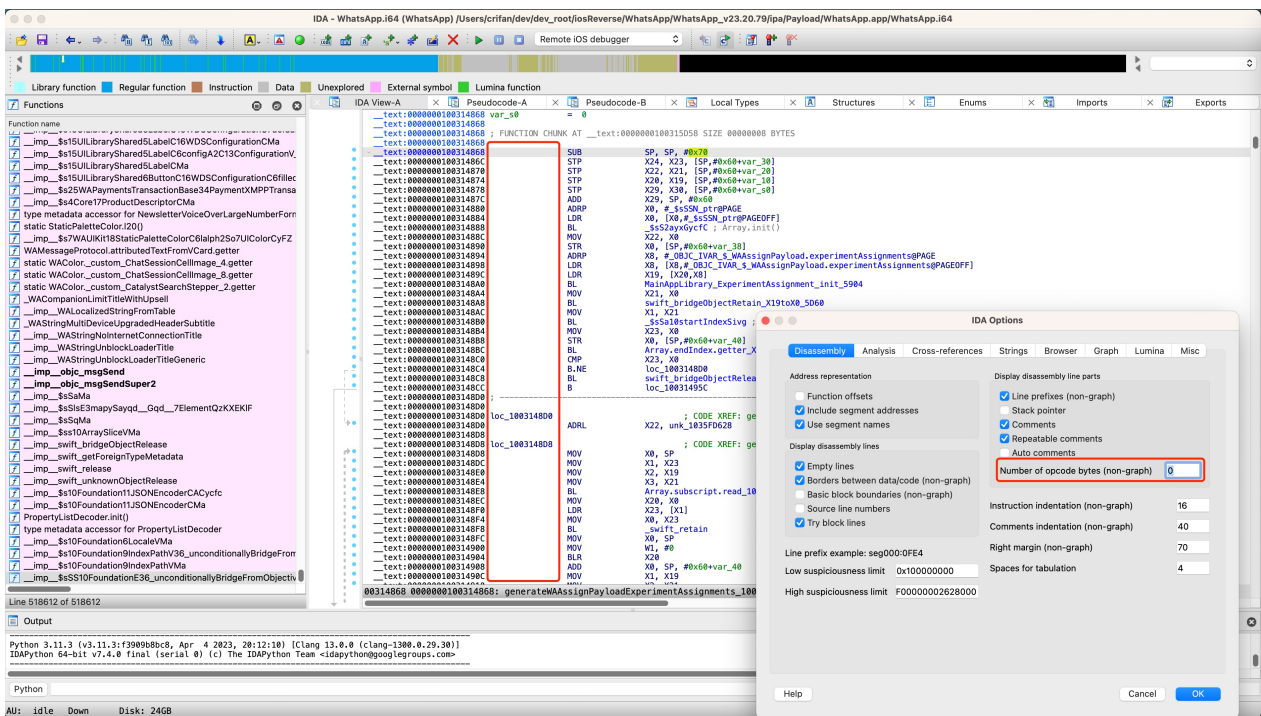
- 一般来说，在逆向尝试搞懂代码逻辑时，不太需要直接查看汇编代码，因为的确很难直接看懂逻辑。
 - 不过，有时候也是可以通过调试，看懂部分汇编代码的逻辑的，比如：
 - [从汇编反推代码逻辑 · iOS逆向：心得集锦](#)

不过有些情况下，会用到汇编代码：

- iOS逆向
 - 静态分析
 - 有些汇编代码中，IDA已帮忙分析和插入了相关的解释信息，值得研究逻辑时去参考
 - 比如，YouTube逆向期间，IDA已帮忙给相关汇编加上了描述，指明了有些代码是vtable的部分
 - 便于分析和对照，寻找对应虚函数的具体实现
 - 动态调试
 - 想要找到调试期间的，Xcode中汇编代码，对应的代码逻辑
 - 往往就需要找到IDA中对应的伪代码是什么
 - 往往就需要先去找到IDA中汇编代码的位置
 - 再去F5（或Tab键）跳转到对应的伪代码的位置

IDA中显示Opcode

IDA中对于显示汇编代码时，默认：没有显示 opcode

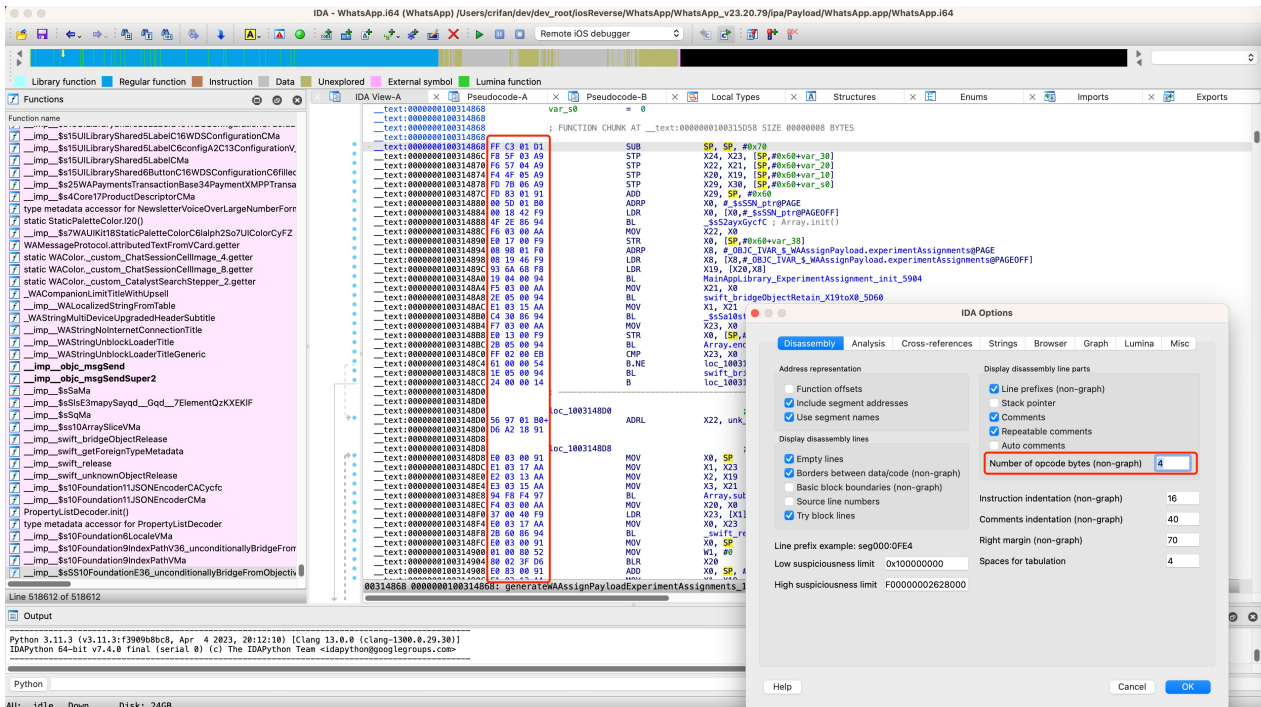


然后去设置中可以开启显示opcode，且可以指定显示几个字节的opcode

比如设置为 4：

- IDA -> Options -> General -> IDA Options -> Disassembly -> Display disassembly line parts
 - Number of opcode bytes(non-graph) : 4

效果是：

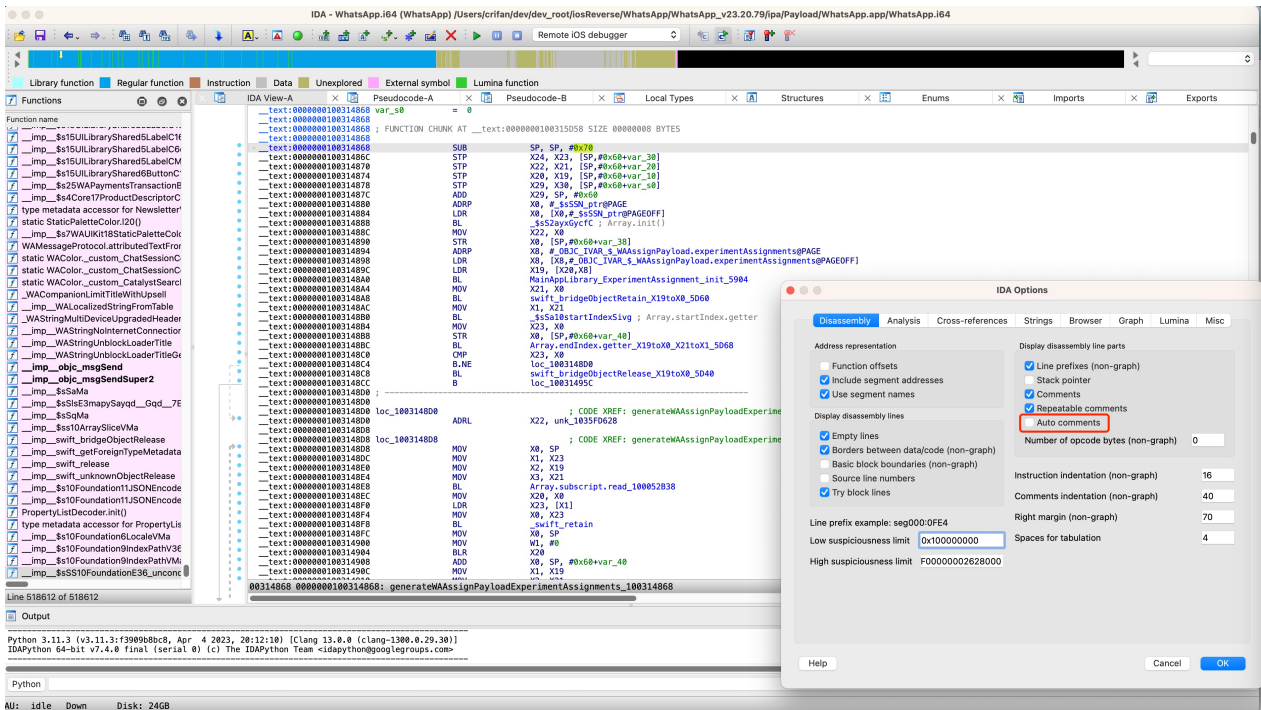


IDA中是否开启：自动注释

对于IDA中汇编代码，可以选择：是否开启 自动注释

默认是：没有开启 自动注释 = Auto comments

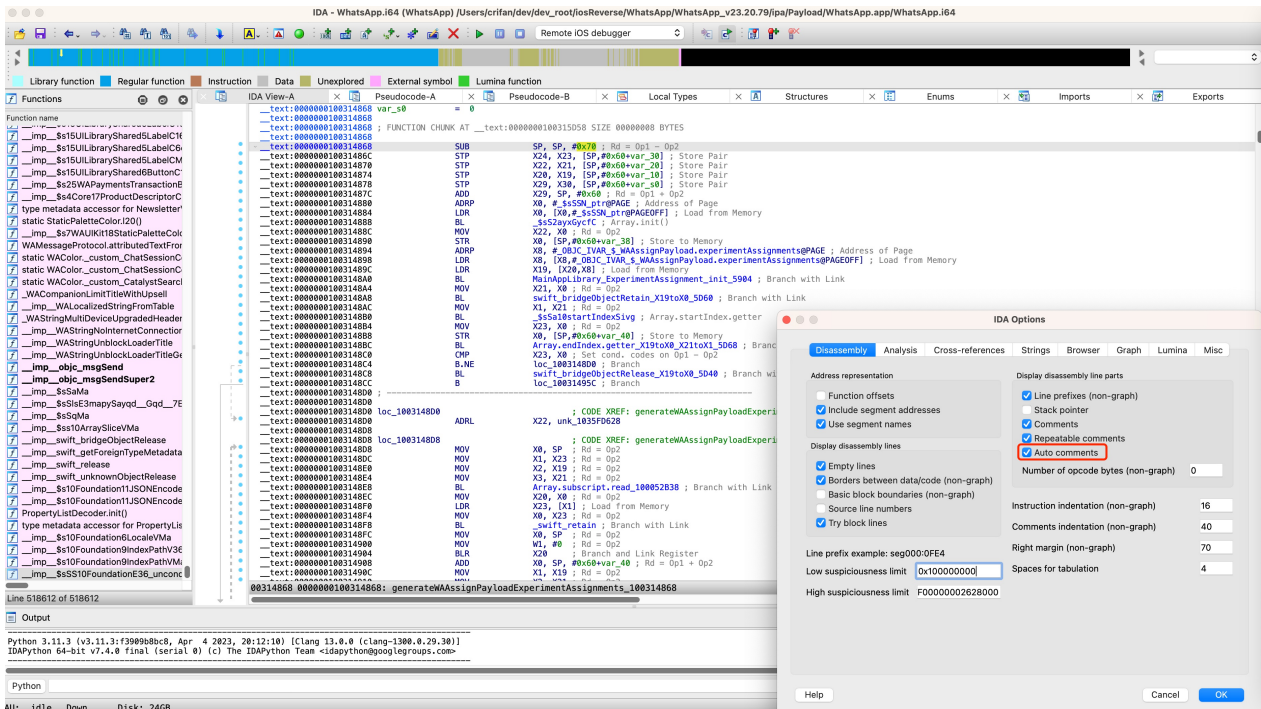
效果是：



可以选择去开启：

- IDA -> Options -> General -> IDA Options -> Disassembly -> Display disassembly line parts
 - 勾选：Auto comments

效果是：



->

自动给各个汇编代码，加上了注释

注释内容，主要是：汇编指令的含义概述

IDA汇编代码如何快速找到匹配的Xcode汇编代码

- 背景
 - iOS逆向调试期间，Xcode去调试汇编代码期间，正在调试某行汇编代码
- 需求
 - 想要找到Xcode正在调试的汇编代码，所对应的IDA中是哪行汇编代码，以及IDA中对应的伪代码是什么，目的是搞懂对应伪代码的大概的代码逻辑，便于尽快辅助的搞懂代码逻辑
- 方法
 - 概述
 - 看Xcode中的汇编和IDA中的汇编的地址最后几位（2位或3位），即可快速对应出代码的位置

详解：

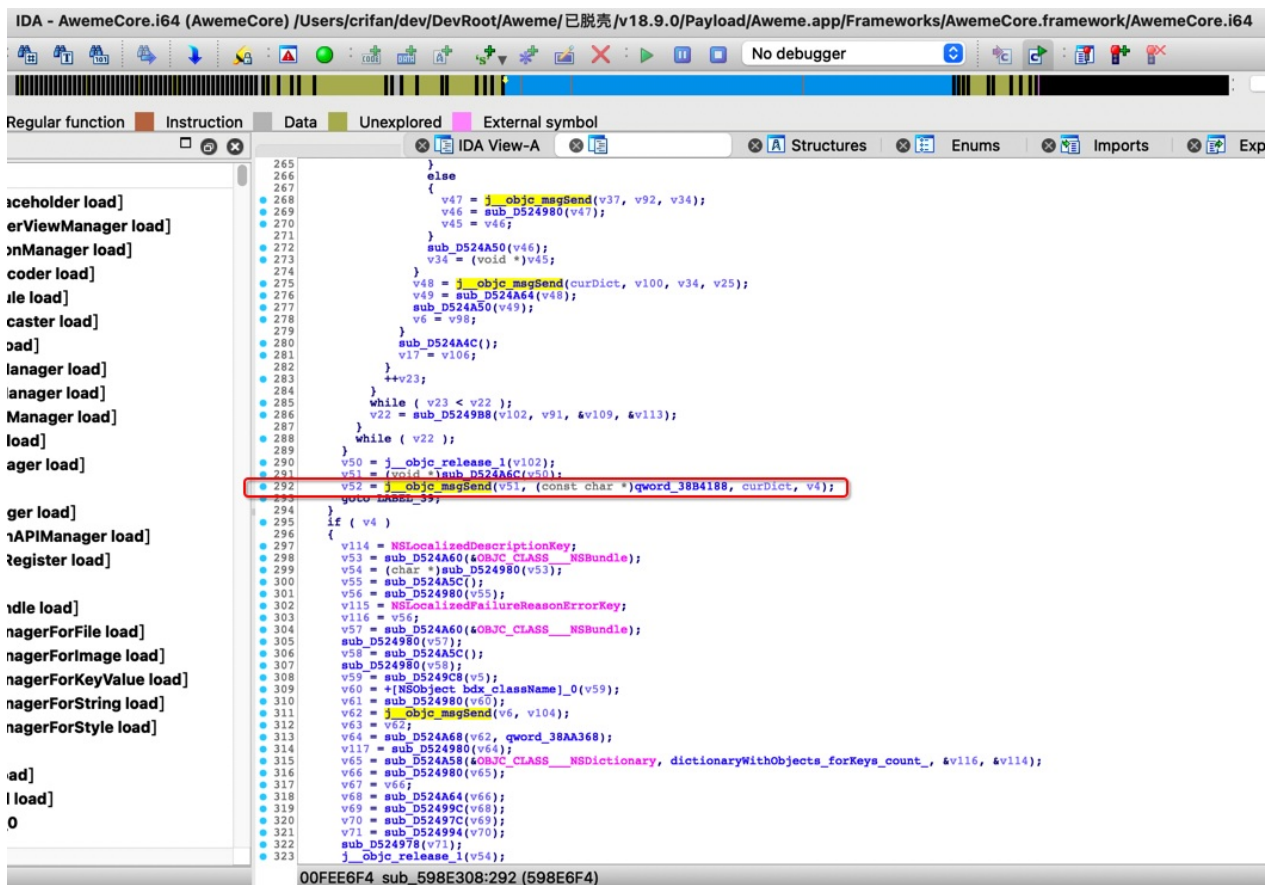
举例

举例1

从某app的IDA伪代码

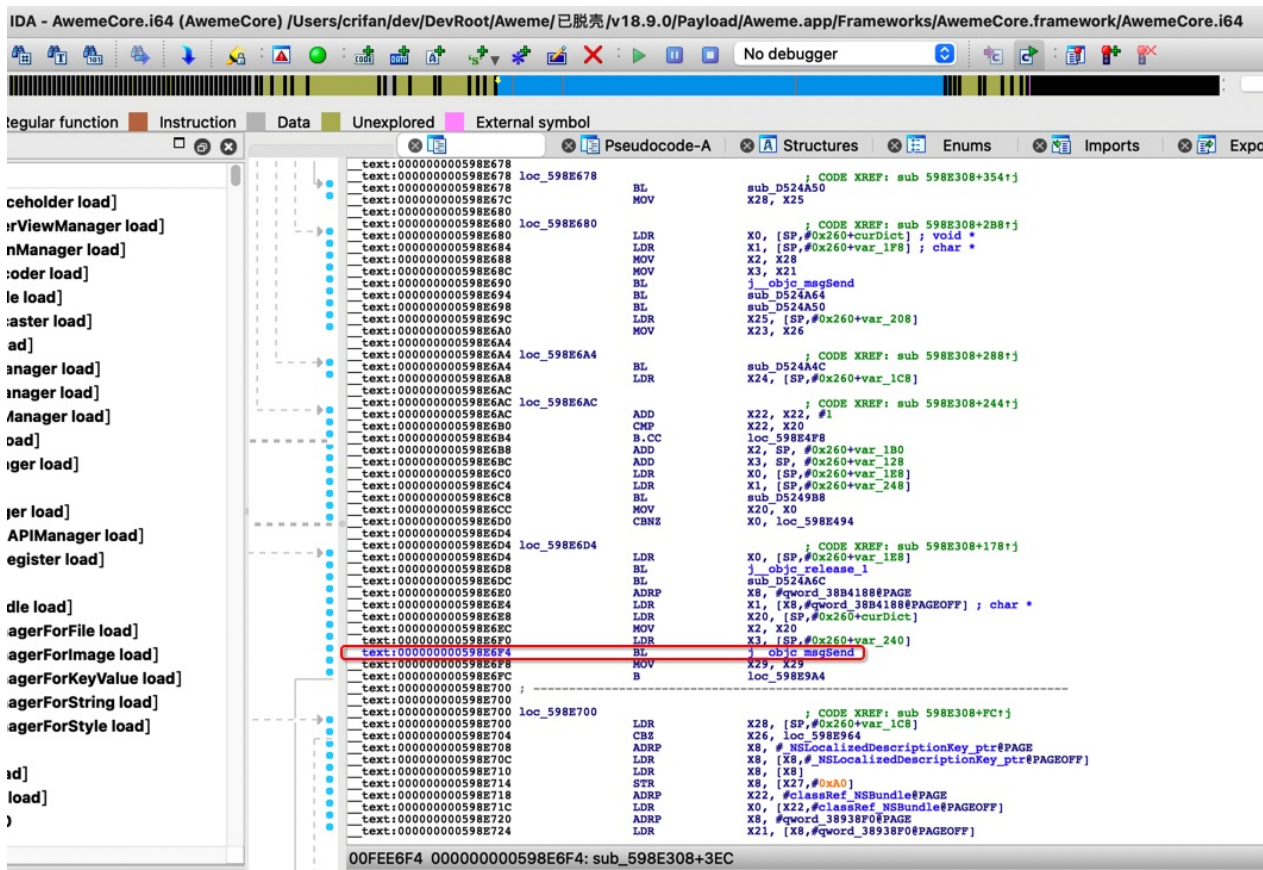
292行

```
v52 = j__objc_msgSend(v51, (const char *)qword_38B4188, curDict, v4);
```



通过Tab键，找到了IDA汇编代码：

```
__text: 000000000598E6F4      BL      j__objc_msgSend
```

此处，不是十分确定，是不是一定对应着，正在调试的，Xcode汇编代码：

```
0x108a0a6f4 <1004>: bl 0x11059fec ; __lldb_unnamed_symbol1189160$$AwemeCore
```

不过其实通过去计算偏移量算出IDA汇编代码的偏移量，和Xcode中显示的+1004，对得上，其实就能判断出来。

不过此处，注意到一个细节，可以快速辅助帮忙确认是否匹配上：

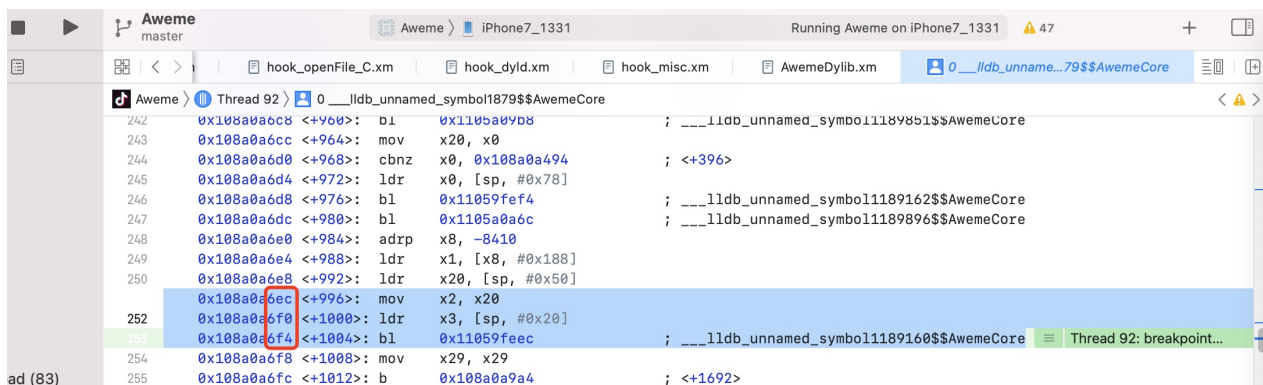
汇编代码的地址的最后2位或最后3位：

IDA汇编代码 和 Xcode汇编代码 的最后2或3位 是一样的

就表示匹配上了：

Xcode汇编代码：

```
0x108a0a6ec <996>: mov x2, x20
0x108a0a6f0 <1000>: ldr x3, [sp, #0x20]
0x108a0a6f4 <1004>: bl 0x11059fec ; __lldb_unnamed_symbol1189160$$AwemeCore
```



-> 前后几行汇编的最后几位是：

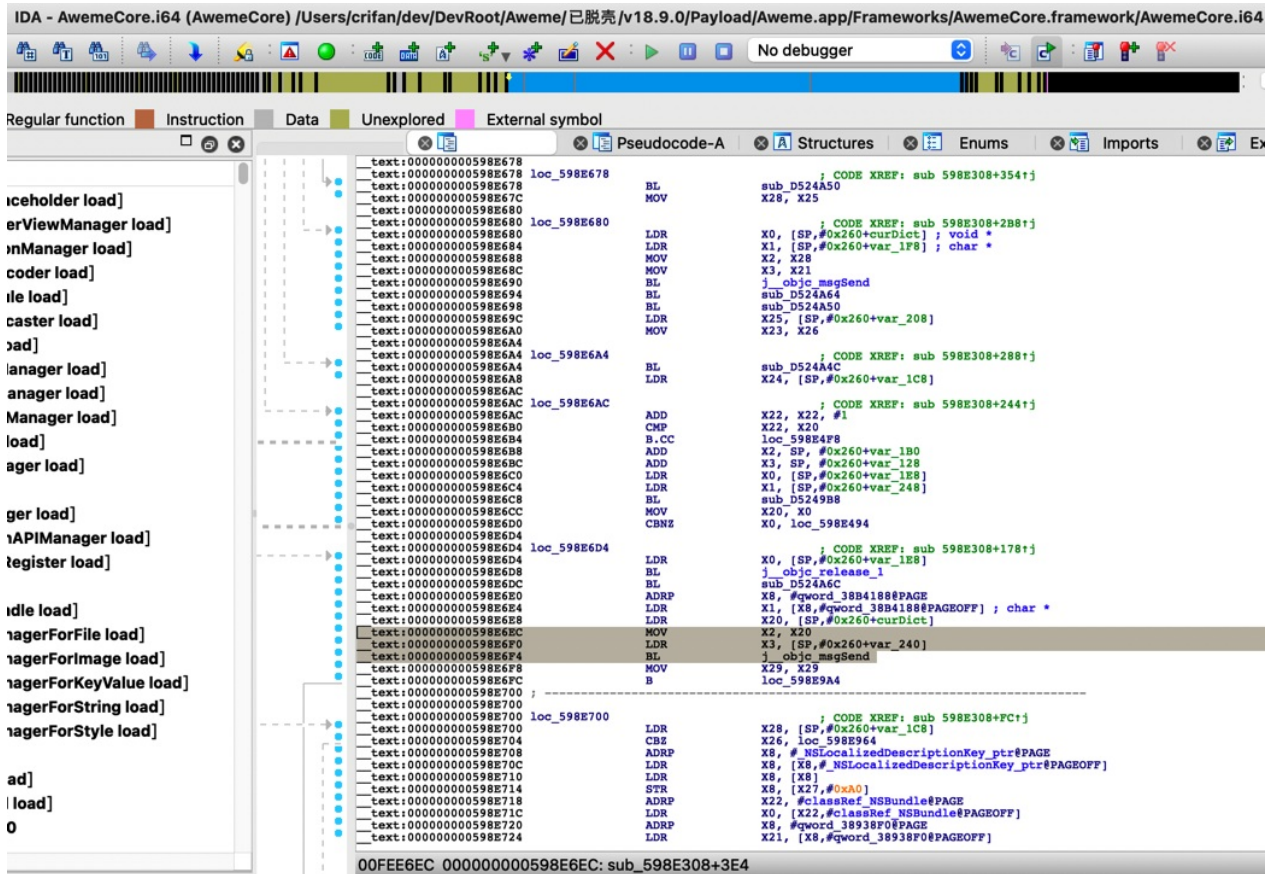
- 6ec
- 6f0
- 6f4

而IDA汇编代码，前后几行：

```

__text:00000000598E6EC      MOV             X2, X20
__text:00000000598E6F0      LDR             X3, [SP,#0x260+var_240]
__text:00000000598E6F4      BL              j__objc_msgSend

```



的最后几位：

- 6EC
- 6F0
- 6F4

分别都是对得上的。

其背后机制是：

- 二进制内的偏移量，相同代码，则肯定是相同的
- 二进制=动态库的基地址=ALSR虽然是变化的
 - 但是ALSR的基地址（一般）肯定是整数倍，至少最后很多位是0

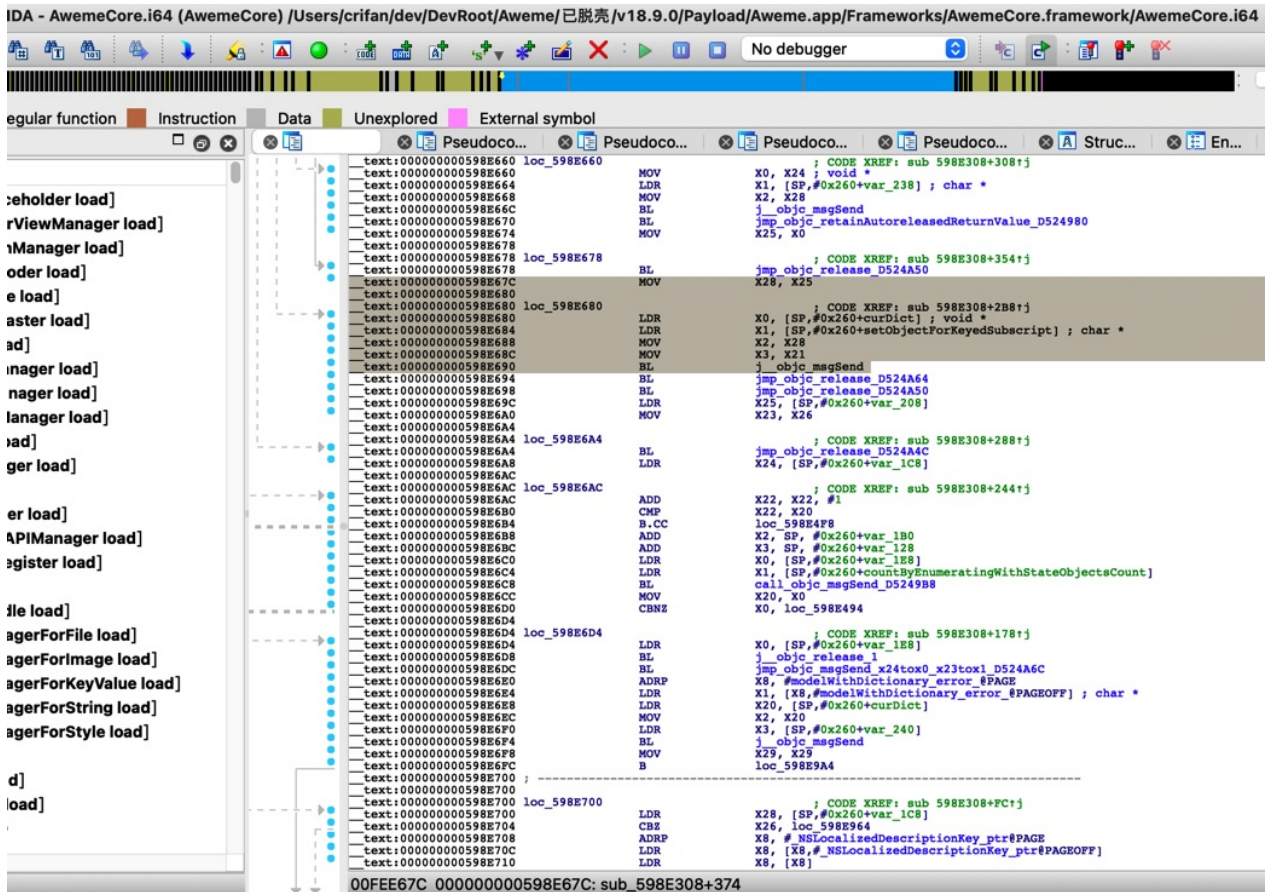
所以最后的地址，最后几位，此处是3位，肯定都是一样的（虽然ALSR起始地址不同）

因此，可以通过最后3位的地址值，对得上，确定就是代码匹配上，对应得上的。

举例2

IDA伪代码：

288行的



然后再去找：

IDA汇编 -> Xcode汇编

即可从找几行连续的，汇编地址的末尾3位

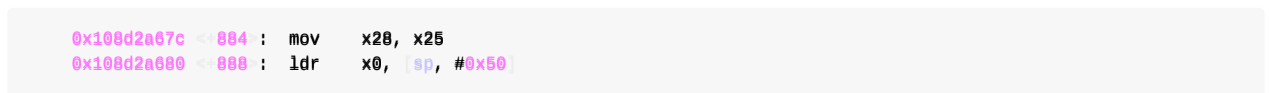
比如：

- 688
- 68C
- 690

即可回头找到Xcode中对应的代码位置：



从而找到 对应代码段：




```
0x108d2a684 < 892 : ldr    x1, [sp, #0x68]
0x108d2a688 < 896 : mov    x2, x28
0x108d2a68c < 900 : mov    x3, x21
0x108d2a690 < 904 : bl     0x1108bfeec ; ____lldb_unnamed_symbol11189160$$AwemeCore
```

效果很好。

就不用麻烦的计算偏移量，即可快速找到：

- IDA中
 - 汇编代码
 - 伪代码
- Xcode中
 - 汇编代码

的相互对应的位置了。

-》便于快速定位代码位置，辅助尽快搞懂代码逻辑。

crifan.org，使用署名4.0国际(CC BY 4.0)协议发布 all right reserved，powered by Gitbook最后更新：2024-10-03 10:26:57

伪代码

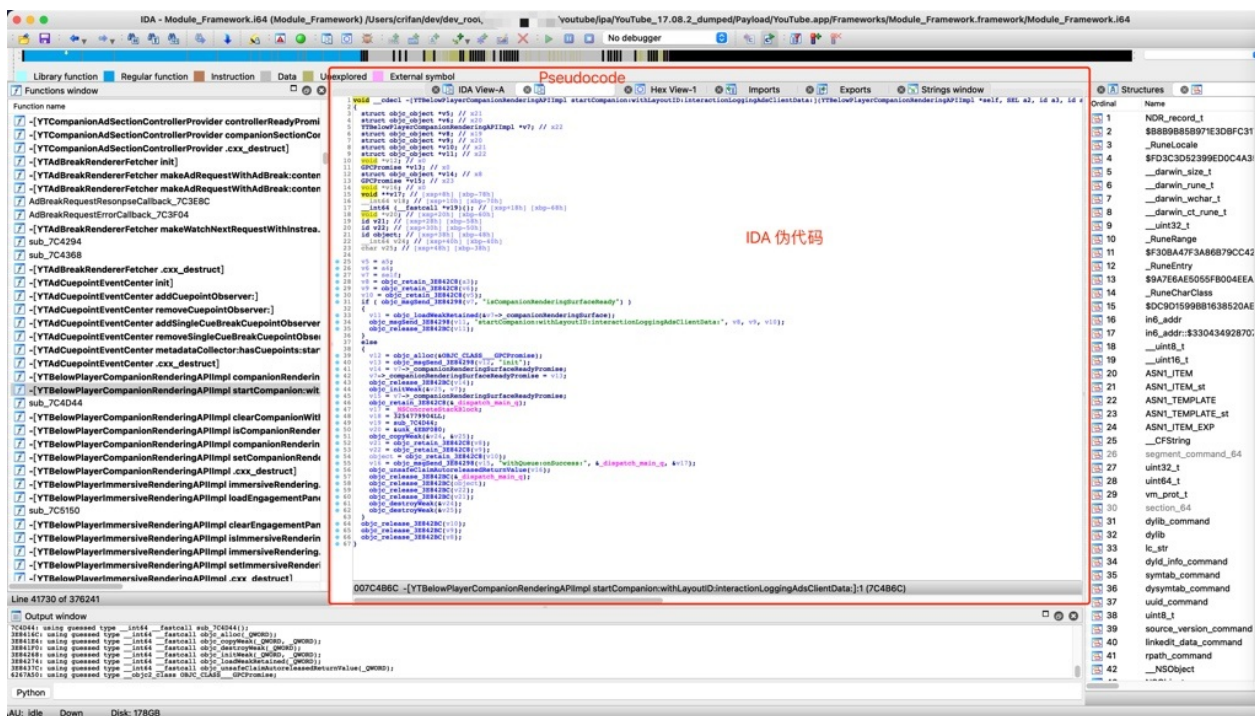
TODO:

- 【未解决】IDA使用心得：伪代码内新增变量
- 【整理】IDA使用心得：刷新当前已打开的伪代码用F5
- 【整理】IDA使用心得：伪代码中的指针+某个数值和0x开头LL结尾的数值不是一个意思
- 【整理】IDA使用心得：给伪代码添加注释
- 【整理】IDA使用心得：伪代码中的可变参数个数的函数去增加或删除参数
- 【整理】iOS逆向心得：IDA使用心得：改变值的显示格式从10进制改为16进制查看Block的flags标志位
- 【整理】iOS逆向心得：IDA Pro使用心得：给函数改名便于快速定位汇编伪代码对应关系
- 【iOS逆向心得】IDA使用心得：伪代码中在修改了别处函数定义后返回导致伪代码中函数调用参数丢失
- 【整理】IDA使用心得：反编译伪代码常见错误
 - [Failures and troubleshooting \(hex-rays.com\)](#)
- 【整理】IDA使用心得：如何理解反汇编后的伪代码的逻辑
- 【记录】IDA使用心得：objc_msgSend跳板函数重命名优化
- 【记录】IDA使用心得：伪代码改名重命名改回默认值
- 【记录】IDA使用心得：给伪代码的变量改名
- 【已解决】IDA使用心得：伪代码中如何找到对应的IDA汇编代码
- 【已解决】IDA中给汇编代码或伪代码改名
- 【整理】IDA使用心得：改名 给变量改类型
- 【整理】IDA使用心得：F5伪代码

代码逆向相关：

- 【已解决】IDA中xsp和xpb是什么意思如何定位地址
- 【整理】IDA使用心得：IDA伪代码和汇编代码 反汇编 逻辑关系 理解

IDA中，支持从 汇编代码，按 F5 快捷键去 反编译 得到的 伪代码 -> 很接近人类写的代码，人类能容易读懂代码逻辑的代码



IDA中最强的功能，应该就属这个 伪代码 了。

IDA反编译出的 伪代码：

- 质量很高：很接近原程序的代码的逻辑
- 且有很多额外好用的功能支持
 - 比如
 - 重命名：rename
 - 变量更改类型：change type
 - 增加减少参数个数
 - 自动解析出类的属性的引用
 - 等等

伪代码中，右键，支持很多功能：

- Rename
- Set type
- Set number representation
- Edit indented comment
- Edit block comment
- Hide/unhide statements
- Split/unsplit expression
- Force call type
- Set call type
- Add/del variadic arguments
- Del function argument
- Add/delete function return type
- Jump to cross reference
- Jump to cross reference globally
- Generate HTML file
- Mark/unmark as decompiled
- Copy to assembly
- Show/hide casts

而根据当前元素类型，（可能）会显示额外菜单=功能=选项：

- 局部变量
 - Reset pointer type
 - Convert to struct *
 - Create new struct type
 - Map to another variable
 - Unmap variable(s)
 - Force new variable
- 联合体union
 - Select union field
- 括号类：圆括号、中括号、花括号
 - Jump to paired paren
- 文本
 - Copy快捷键：Ctrl+C
- C表达式关键字
 - Collapse/uncollapse item

具体细节详见：

[Interactive operation \(hex-rays.com\)](http://hex-rays.com)

导出全部伪代码

IDA中，一般来说，伪代码都是针对单个函数的：反编译再查看单个函数的伪代码。

后来发现，想要导出全部伪代码，也是可以的。

详见：

- 【未解决】用插件导出IDA的YouTube的Module_Framework的全部反汇编的源码伪代码
- 【已解决】IDA中用idat64的Batch Mode尝试反编译导出YouTube的Module_Framework全部代码伪代码

标记为已编译 Mark/unmark as decompiled

This command marks the current function as decompiled. It is a convenient way to track decompiled functions. Feel free to use it any way you want. Marking a function as decompiled will change its background color to the value specified by the MARK_BGCOLOR parameter in the configuration file. The background color will be used in the pseudocode window, in the disassembly listing, and in the function list.

拷贝到汇编 Copy to assembly

This command copies the pseudocode text to the disassembly window. It is available from the popup right-click menu. Please note that only "meaningful" lines are copied. Lines containing curly braces, else/do keywords will be omitted. The copied text is represented as anterior comments in the disassembly. Feel free edit them the way you want. The copied text is static and will not change if the pseudocode text changes.

crifan.org, 使用署名4.0国际(CC BY 4.0)协议发布 all right reserved, powered by Gitbook最后更新： 2022-10-24 12:04:40

函数调用

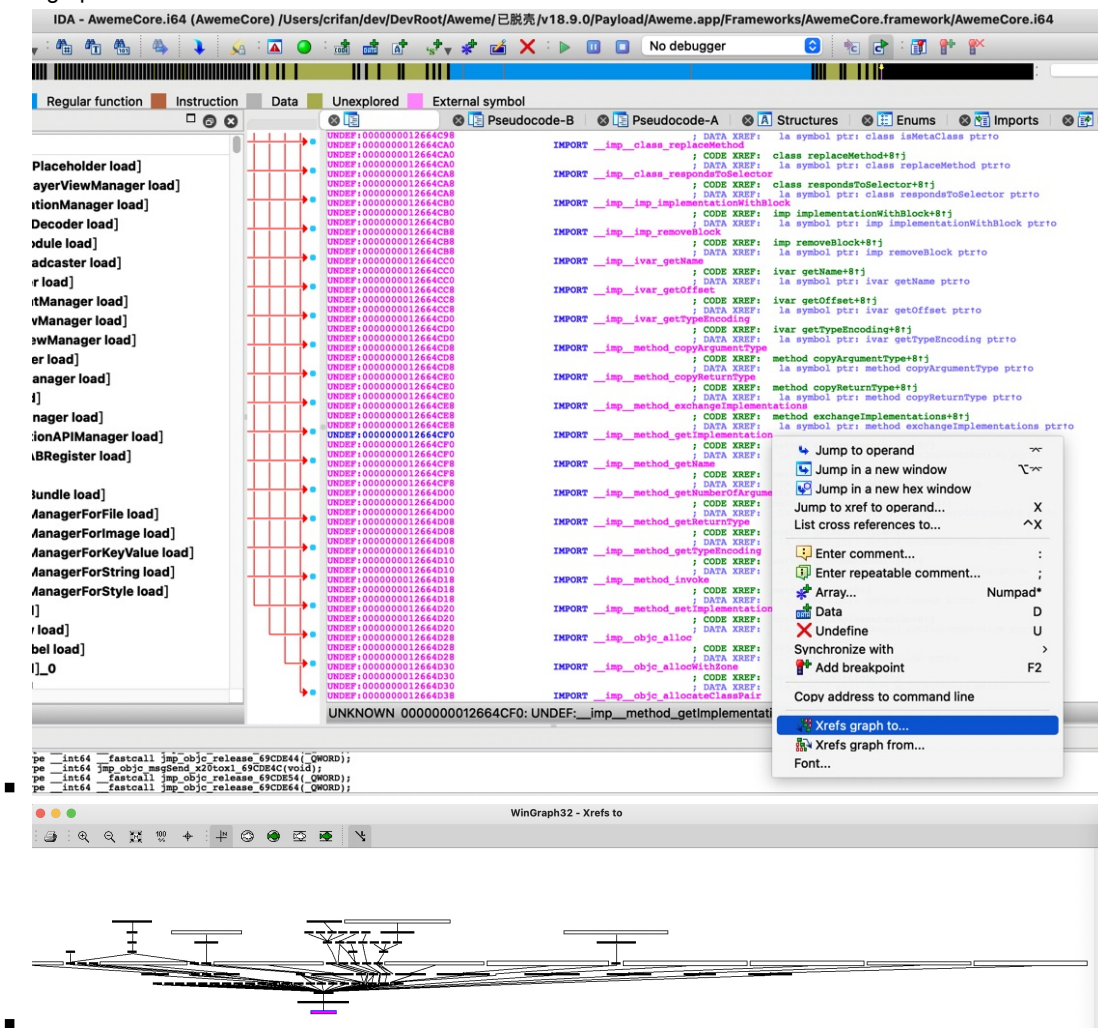
TODO:

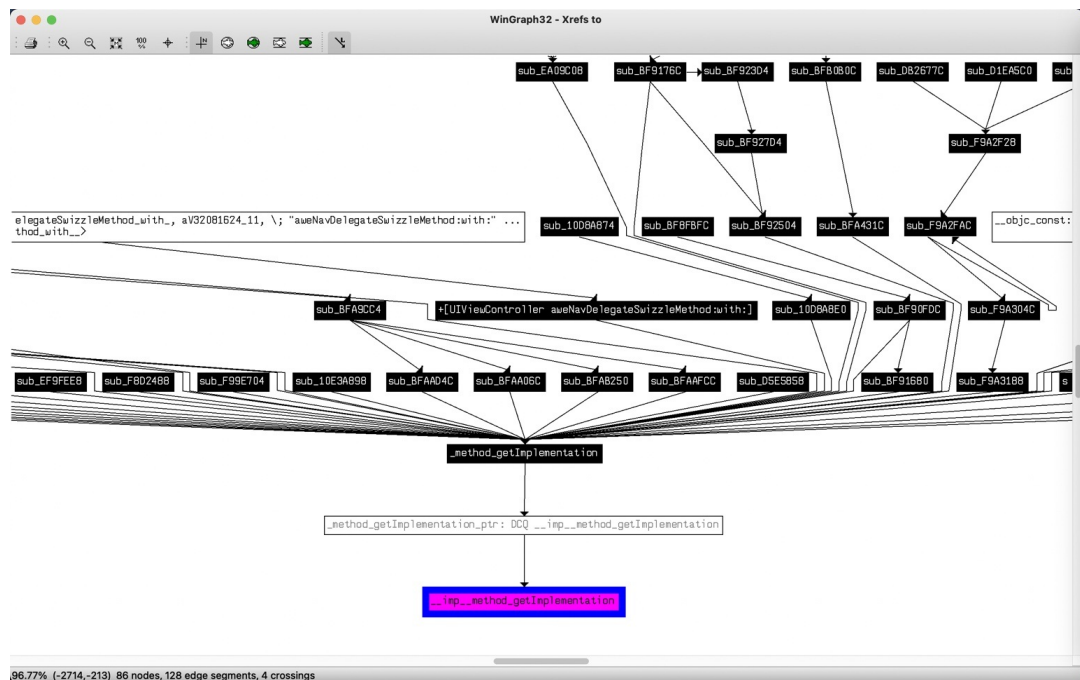
- 【整理】iOS逆向心得：IDA中以列表形式列出函数被调用的地方
 - 【整理】iOS逆向心得之IDA使用心得：查看函数被调用的所有的地方即被调用函数的列表
 - 【整理】iOS逆向心得之IDA使用心得：如何快速找到真正的函数的被调用的列表函数名
- 【整理】IDA 使用心得：交叉引用以列表方式显示
- 【已解决】研究抖音Hook检测：调用到method_getImplementation的地方

IDA中代码分析方面，对于函数的调用的关系，也有很好的支持。

iOS逆向期间，往往涉及到，想要搞懂一个函数，被其他哪些地方调用到了等等，和函数调用关系相关的内容。IDA对此支持的都很好。

- 函数调用
 - 效果举例
 - Xrefs graph to

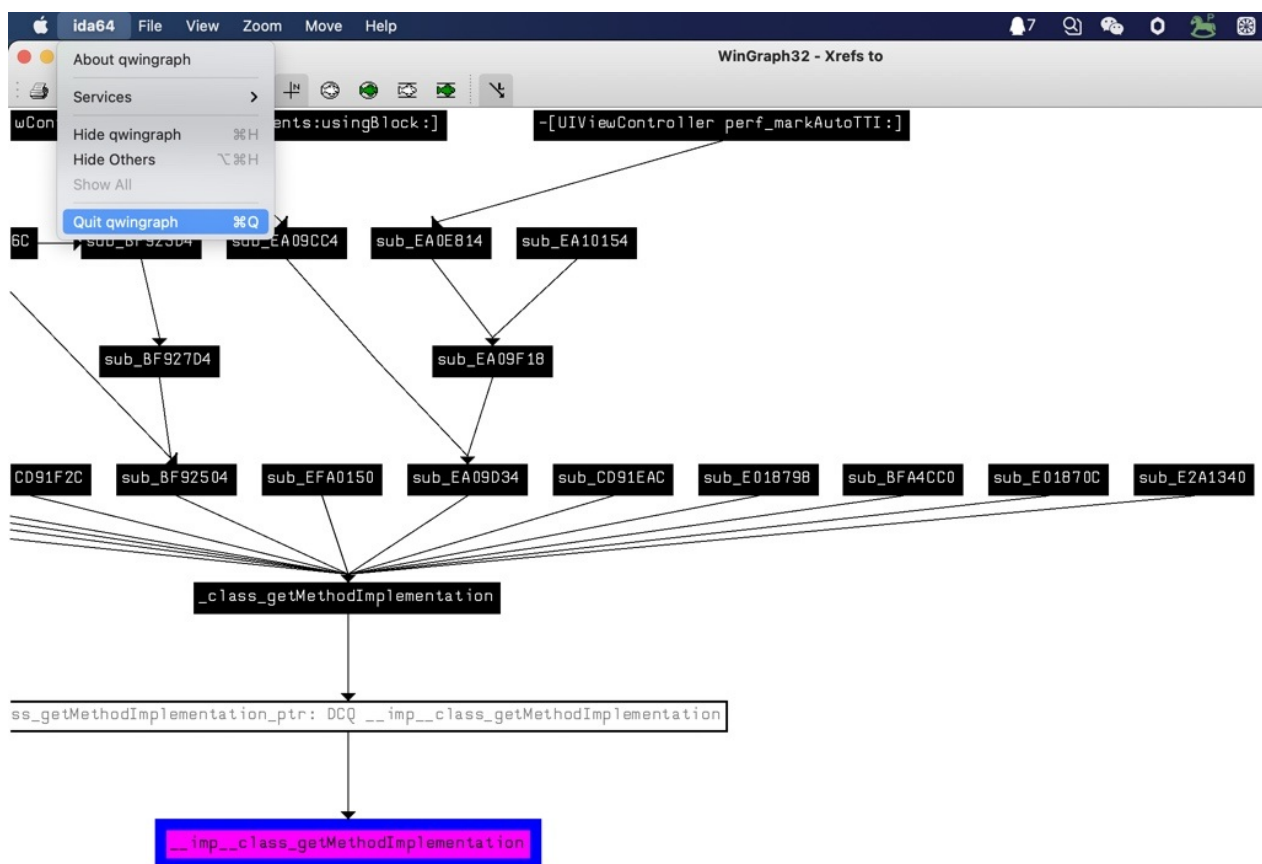




Xrefs to=有哪些地方引用到了此函数

显示界面的底层实现所用的库

IDA中函数调用的graph, 通过 ida64 的quit, 看到的是: qwingraph



而 `qwingraph`，其实是一个插件，底层可视化插件

可以从

[Download center \(hex-rays.com\)](https://hex-rays.com)

找到:

- Qwingraph v1.10
 - Source code the Wingraph we use and modified (GPL)
 - https://hex-rays.com/products/ida/support/freelfiles/qwingraph_src.zip

而 WinGraph32 本身的 关于 的信息是:

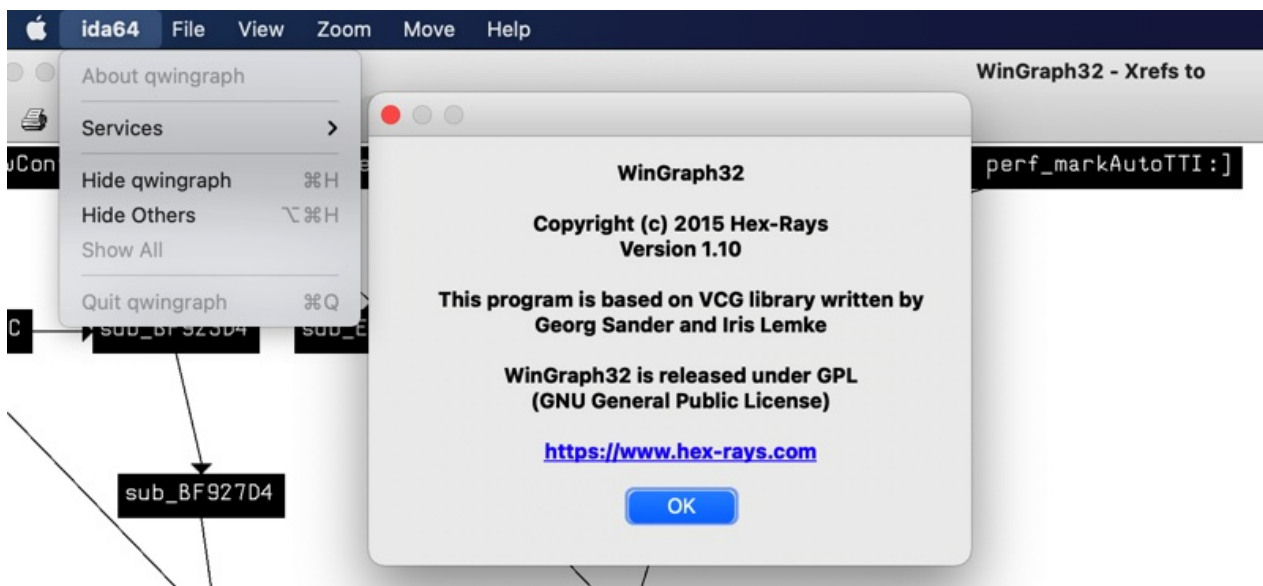
```
WinGraph32

Copyright (c) 2015 Hex-Rays
Version 1.10

This program is based on VCG library written by Georg Sander and Iris Lemke

WinGraph32 is released under GPL (GNU General Public License)

https://www.hex-rays.com
```



crifan.org, 使用署名4.0国际(CC BY 4.0)协议发布 all right reserved, powered by Gitbook最后更新: 2022-11-07 17:52:43

结构体定义

TODO:

- 新增Structure结构体定义 + 且双击后，可以导入到数据库中
 - 【已解决】IDA中如何给Local Types中struct MLServerABRLoader加上嵌入的struct结构体定义
- 【整理】iOS逆向心得：IDA使用心得：修改变量类型Set Ivar Type后IDA可以自动解析结构体的属性和字段
- 【整理】IDA使用心得：类的部分字段无法解析，导致伪代码中类的属性错误，需要手动修复结构体定义
-

IDA中，支持把类的原始定义，通过结构体的形式写出来（甚至自动分析出来对应结构体定义），从而后续的汇编代码和伪代码中，自动解析出类的属性和函数的调用，很是方便。

此处的类的结构体定义，主要涉及到两方面：

- Structures

-
- Local Types

-

且也支持新增自定义的结构体，更改已有类的结构体的字段定义等，很强大好用的功能。

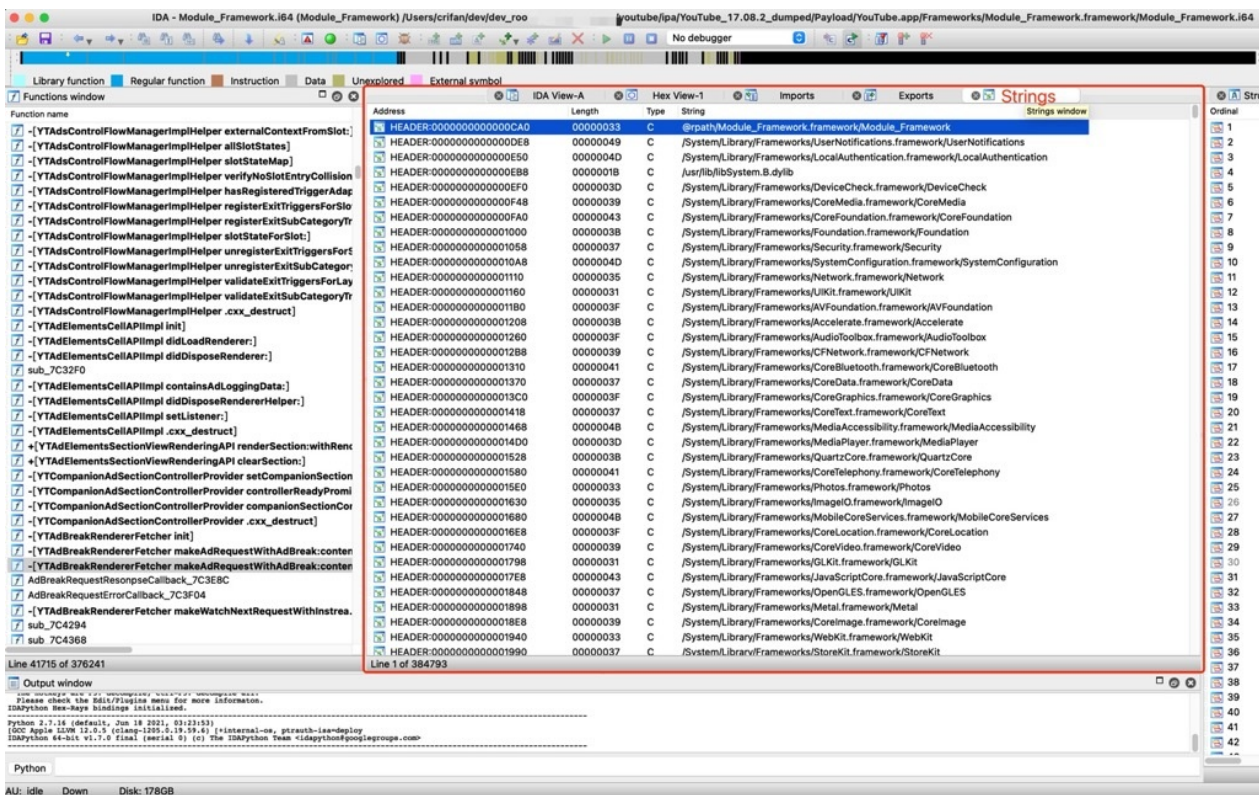
- 创建结构体

-

crifan.org，使用署名4.0国际(CC BY 4.0)协议发布 all right reserved，powered by Gitbook最后更新： 2022-10-24 12:12:29

字符串

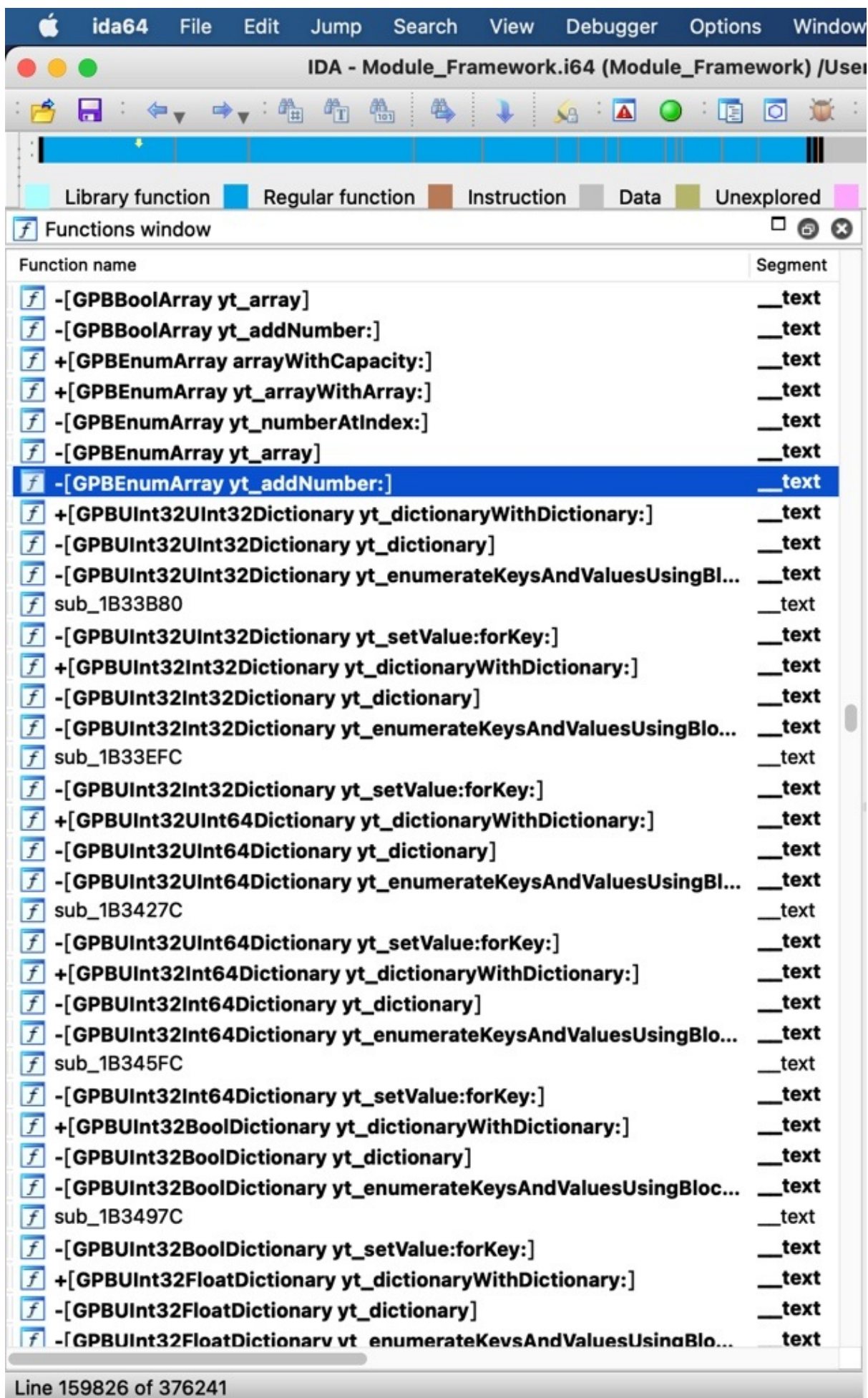
IDA也能自动分析出，二进制中有哪些字符串，放到一个单独视图 strings，供分析和研究。



crifan.org，使用署名4.0国际(CC BY 4.0)协议发布 all right reserved，powered by Gitbook最后更新： 2022-10-24 11:59:05

函数列表

IDA可以分析出，二进制中的所有的函数，并且列出函数的列表，供查找和定位，以及后续代码逻辑的研究。

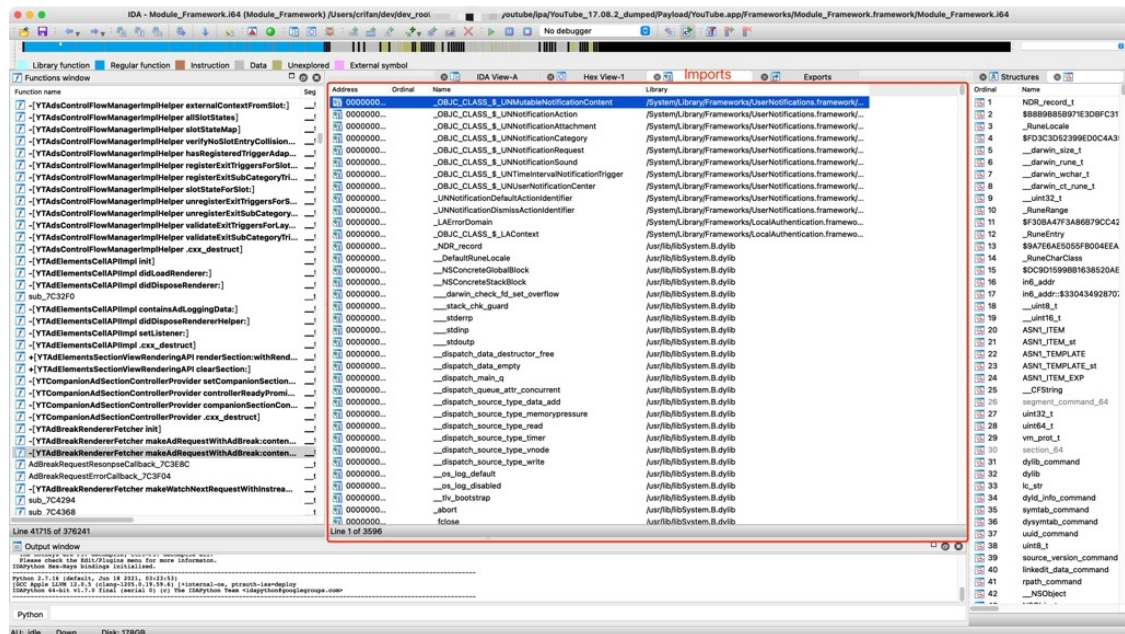


crifan.org, 使用署名4.0国际(CC BY 4.0)协议发布 all right reserved, powered by Gitbook最后更新: 2022-10-24 12:15:30

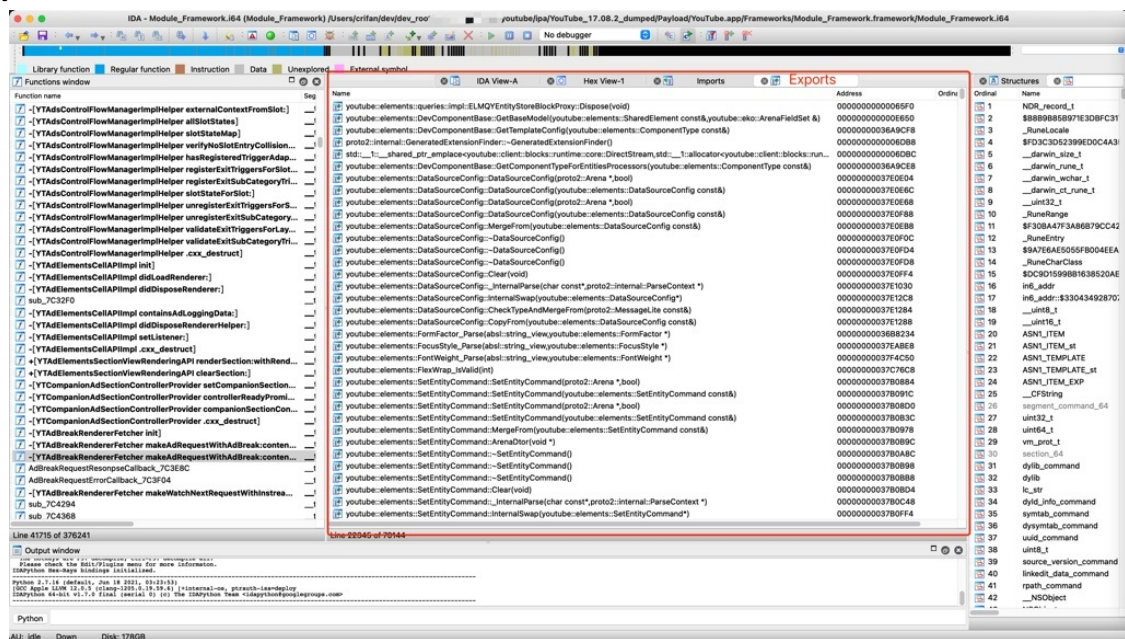
导入和导出

IDA中还有2个独立的窗口是：

- Import=导入
 - 当前二进制，导入了哪些函数
 - = 引用了外部的，别的库的哪些函数
 - 举例



- Export=导出
 - 当前二进制，导出了哪些函数
 - 供别处（比如自己程序的另外的二进制中去）使用
 - 举例



在逆向分析时，可以根据，导出和导出，找到一些相关线索

比如，在iOS逆向的越狱检测和反越狱检测中，就可以去找，当前二进制是否导入了，常用于越狱检测的一些系统函数。

详见：

【整理Book】iOS逆向开发：越狱检测和反越狱检测

crifan.org, 使用署名4.0国际(CC BY 4.0)协议发布 all right reserved, powered by Gitbook最后更新: 2022-10-24 11:55:17

打补丁

IDA还支持打补丁=修改程序的二进制文件

下面详细介绍：

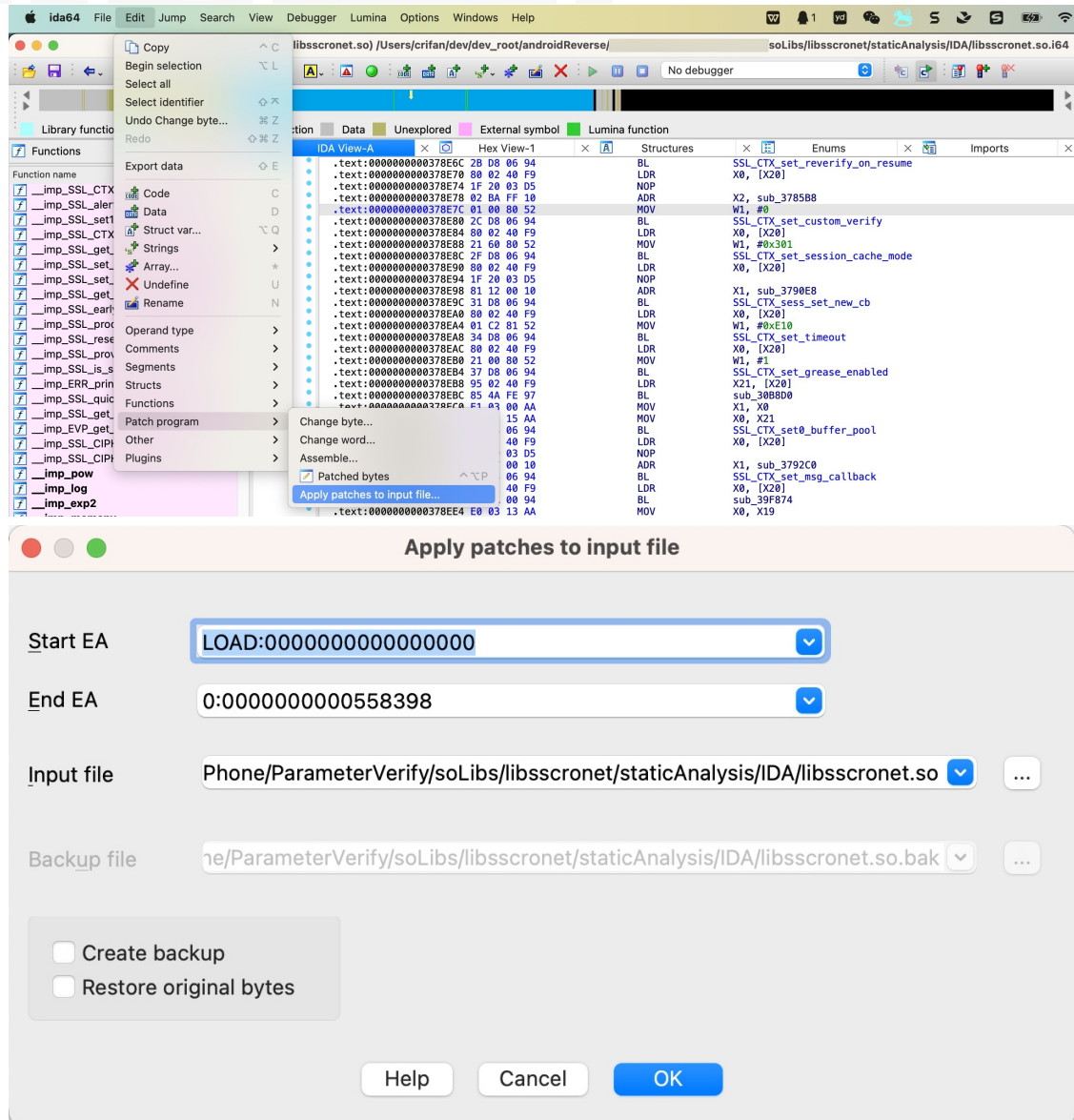
IDA中打patch=打补丁=修改二进制 的步骤：

- 鼠标点击所在要修改的位置
 - 注：只有鼠标事先点击到了，要修改内容，要打补丁的位置，到时候弹框显示出来的内容，才是我们要修改的内容
- IDA -> Edit -> Patch program -> Change byte

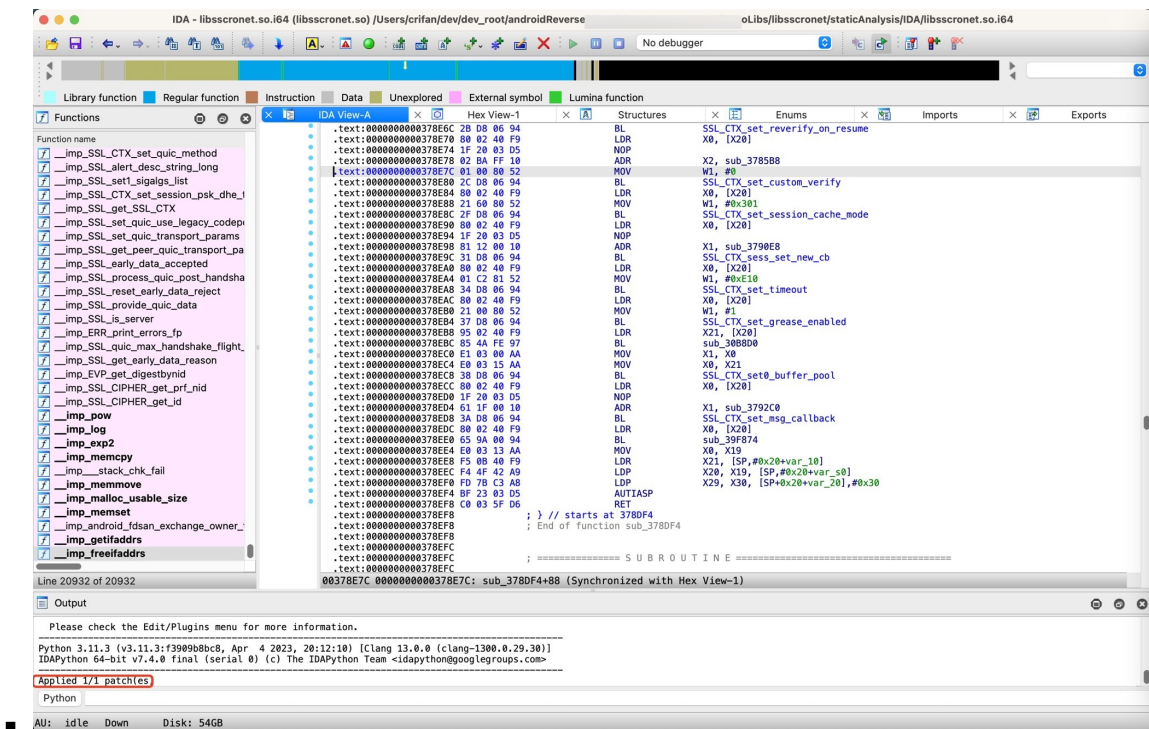
-
- 显示 Patch Bytes 弹框

-
- 然后去修改对应的数据
 - 举例
 - 0x378E7C
 - 从 Origin value :
 - **21 00 80 52** 2C D8 06 94 80 02 40 F9 21 60 80 52
 - 注：此处其实一般只需要修改4个字节，但是默认会显示出16个字节的数据，所以此时后续12个字节的数据，往往保留不变即可
 - 修改 Values 为：
 - **01 00 80 52** 2C D8 06 94 80 02 40 F9 21 60 80 52
 - 即：把第一个字节的 0x21 修改为 0x01，即可
 - 点击OK，真正修改

- 最终记得保存（到原始二进制中）：
 - IDA -> Edit -> Patch program -> Apply patches to input file -> OK



- 注：
 - 如果需要额外备份，则勾选： Create backup



- 注：
 - 保存后：底部会显示log：
 - Applied 1/1 patch(es)

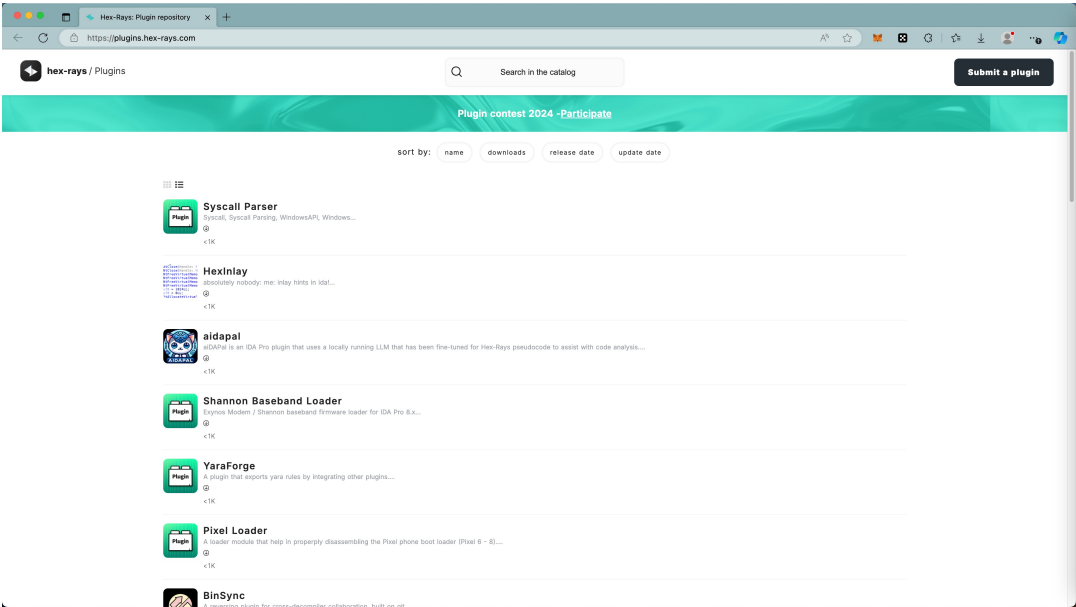
crifan.org, 使用署名4.0国际(CC BY 4.0)协议发布 all right reserved, powered by Gitbook最后更新: 2025-06-03 09:40:05

插件

IDA还支持插件的机制，可以扩展支持更多更强的各种功能。

插件机制

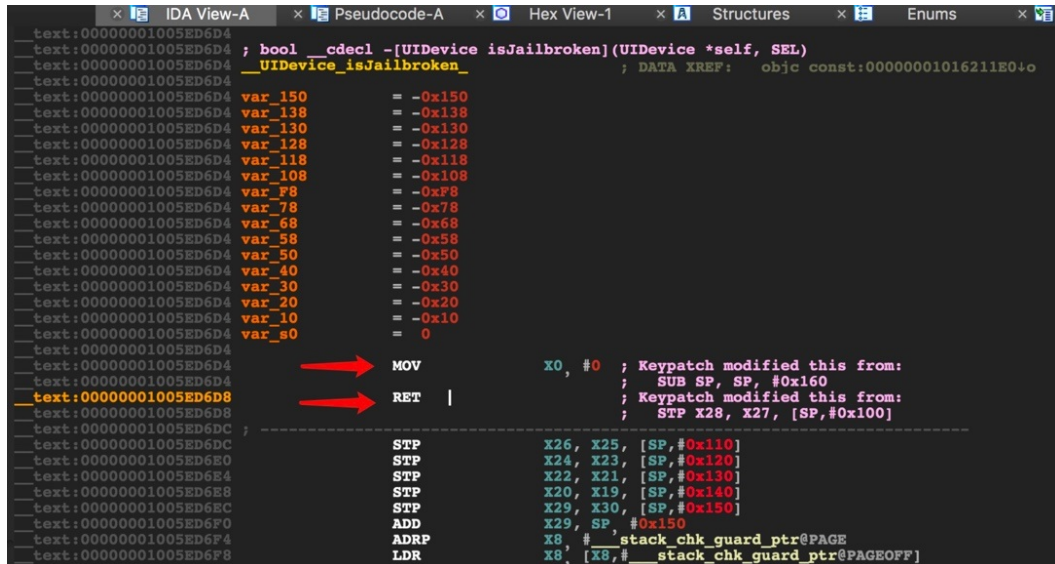
- IDA插件
 - 概述
 - IDA支持插件机制
 - 作用
 - 通过写代码，实现特定功能
 - 举例
 - 写代码，增强对对恶意代码的分析
 - 提取Mach-O中的ObjC的符号表
 - 具体实现
 - 支持多种语言
 - Python
 - 详见
 - [子教程中提到的](#)
 - [IDA插件脚本：IDAPython](#)
 - C++
 - 官网文档
 - [Plugins | Hex-Rays Docs](#)
 - 如何写插件 == [How to write your own plugin? | Hex-Rays Docs](#)
 - 用Python（库：IDAPython）写插件
 - [How to create a plugin in IDAPython? | Hex-Rays Docs](#)
 - 用C++写插件
 - [How to write your own plugin? | Hex-Rays Docs](#)
 - 官网插件库
 - [Hex-Rays: Plugin repository](#)



常用插件

常见的插件有：

- stealth
- findcrypt
- highlighter
- unispector
- keypatch
 - 有机会参考
 - [iOS重打包绕过签名校验防护 | La0s](#)
 - 使用Keypatch插件patch两条汇编 `MOV X0, #0 && RET`



```
text:000000001005ED6D4 ; bool __cdecl -[UIDevice isJailbroken](UIDevice *self, SEL)
text:000000001005ED6D4 __UIDevice_isJailbroken ; DATA XREF: objc const:000000001016211E0+o
text:000000001005ED6D4 var_150 = -0x150
text:000000001005ED6D4 var_138 = -0x138
text:000000001005ED6D4 var_130 = -0x130
text:000000001005ED6D4 var_128 = -0x128
text:000000001005ED6D4 var_118 = -0x118
text:000000001005ED6D4 var_108 = -0x108
text:000000001005ED6D4 var_F8 = -0xF8
text:000000001005ED6D4 var_78 = -0x78
text:000000001005ED6D4 var_68 = -0x68
text:000000001005ED6D4 var_58 = -0x58
text:000000001005ED6D4 var_50 = -0x50
text:000000001005ED6D4 var_40 = -0x40
text:000000001005ED6D4 var_30 = -0x30
text:000000001005ED6D4 var_20 = -0x20
text:000000001005ED6D4 var_10 = -0x10
text:000000001005ED6D4 var_s0 = 0
text:000000001005ED6D4 MOV X0, #0 ; Keypatch modified this from:
text:000000001005ED6D8 RET ; SUB SP, SP, #0x160
text:000000001005ED6D8 ; Keypatch modified this from:
text:000000001005ED6D8 ; STP X28, X27, [SP, #0x100]
text:000000001005ED6DC ; -----
text:000000001005ED6DC STP X26, X25, [SP, #0x110]
text:000000001005ED6E0 STP X24, X23, [SP, #0x120]
text:000000001005ED6E4 STP X22, X21, [SP, #0x130]
text:000000001005ED6E8 STP X20, X19, [SP, #0x140]
text:000000001005ED6EC STP X29, X30, [SP, #0x150]
text:000000001005ED6F0 ADD X29, SP, #0x150
text:000000001005ED6F4 ADRP X8, #'stack_chk_guard_ptr@PAGE
text:000000001005ED6F8 LDR X8, [X8, #'stack_chk_guard_ptr@PAGEOFF]
```

- 去试试：
 - 用keypatch插件，patch打补丁，插入（汇编）指令

其他插件

- 网上别人整理的IDA插件
 - [onethawt/idadplugins-list: A list of IDA Plugins](#)

crifan.org，使用署名4.0国际(CC BY 4.0)协议发布 all right reserved，powered by Gitbook最后更新：2024-10-03 10:43:37

IDA使用心得

TODO:

- 【记录】IDA Pro使用记录
- 【已解决】IDA中View-A中如何方便的选择并拷贝汇编代码
- 【未解决】IDA反编译的伪代码有错误如何修改改动
- 【未解决】IDA调试iPhone中的iOS的app
-
- 【已解决】自己Mac中恢复IDA开发环境
 - 【已解决】Mac Big Sur中运行ida64.app报错：您没有权限来打开应用程序
 - 【已解决】Mac Big Sur中正常运行IDA Pro中的ida.app
- 函数调用图
 - 【记录】研究YouTube广告视频请求相关函数：sub_3B8318C
- 【无需解决】自己Mac打开ida64的文件提示IDA Pro已过期
- 【iOS逆向心得】IDA使用心得：伪代码中在修改了别处函数定义后返回导致伪代码中函数调用参数丢失
- 【整理】iOS逆向心得：IDA Pro使用心得：给函数改名便于快速定位汇编伪代码对应关系
- 【整理】IDA使用心得：通过给函数Set Item Type去修正函数的参数的个数和类型和返回值类型
- 【整理】iOS逆向心得：IDA使用心得：改变值的显示格式从10进制改为16进制查看Block的flags标志位
- 【整理】iOS逆向心得：IDA使用心得：修改变量类型Set Ivar Type后IDA可以自动解析结构体的属性和字段
-
- 【整理】IDA使用心得：根据selector跳转到函数
-
- Local Types 结构体 类定义
 - 【已解决】IDA中如何给Local Types中struct MLServerABRLoader加上嵌入的struct结构体定义
 - 【已解决】IDA中修改类的结构体定义报错：Failed to save the new type Probably some of its symbols are already used elsewhere
 - 【已解决】IDA中如何指定Local Types中HAMDASHSampleBufferSource的bool属性_pendingSeek后偏移量是0x4
 - 【无法解决】IDA尝试修改Structures中类型的属性的offset偏移量时弹框警告：Oops IDA has almost crashed
 - 【已解决】IDA中类HAMDASHSampleBufferSource的属性解析出错：could not convert typeinfo failed to add null invalid type name
 - 【已解决】IDA中Structures结构体定义中修改自定义类的属性的偏移量
 - 【整理】IDA心得：自定义的类的属性偏移量和自动生成的偏移量不匹配
 - 【整理】IDA使用心得：类的部分字段无法解析，导致伪代码中类的属性错误，需要手动修复结构体定义
 - 【未解决】研究YouTube逻辑：补全MLServerABRLoader的字段属性偏移量对应关系
- 【记录】IDA中新增YouTube的CDStruct的相关结构体字段定义
- 【整理】IDA使用心得：跳转到历史列表的函数位置
- 【整理】IDA使用心得：给伪代码添加注释
- 【整理】IDA使用心得：伪代码中的可变参数个数的函数去增加或删除参数
- 【整理】IDA使用心得：伪代码中的指针+某个数值和0x开头LL结尾的数值不是一个意思
- 【已解决】IDA中伪代码含义：std basic_string char_traits allocator append
- 【整理】IDA使用心得：把类结构Structures的窗口放在右边方便对比查看
- 【整理】IDA使用心得：刷新当前已打开的伪代码用F5
- 【未解决】IDA使用心得：伪代码内新增变量
- 【已解决】IDA使用心得：如何修改设置函数结束位置
- 【整理】IDA使用心得：反编译伪代码有bug导致子函数调用丢失
- 【或许解决】IDA伪代码更改变量名等改动代码后切换回来改动丢失

使用IDA逆向分析和调试期间，有很多心得，整理如下，供参考。

界面布局

- 【整理】IDA使用心得：把类结构Structures的窗口放在右边方便对比查看

代码分析

*(_DWORD *) 的含义

```
v7 = *(_DWORD *) (v7 + 8);
```

==

```
v7 = *(v7 + 8)
```

==

汇编的：

```
MOV v7, DWORD PTR [v7 + 0x8]
```

含义解释：

v7 is assigned with the value located at address $v7 + 8 \times \text{sizeof}(\text{DWORD})$. For example, if $v7 = 0xabcd0123$ then $v7 + 8 \times \text{sizeof}(\text{DWORD}) = 0xabcd0143$. Whatever is located at 0xabcd0143 will be assigned to v7.

函数跳转

- Jump跳转
 - 【整理】IDA使用心得：根据selector跳转到函数
 - 【整理】IDA使用心得：跳转到历史列表的函数位置

函数

- 【整理】IDA使用心得：通过给函数Set Item Type去修正函数的参数的个数和类型和返回值类型

类

- 【整理】IDA心得：自定义的类的属性偏移量和自动生成的偏移量不匹配

bug

- 【整理】IDA使用心得：iOS的ObjC伪代码反编译翻译的有错误不够准确

如何处理数组Array

Working with array

https://hex-rays.com/wp-content/uploads/2021/10/igor-tip-of-the-week-S01.pdf

— + ↺ ↻ | 🖨 页面视图 | 🔊 朗读此页内容

09 Oct 2020

<https://hex-rays.com/blog/igor-tip-of-the-week-10-working-with-arrays/>

Arrays are used in IDA to represent a sequence of multiple items of the same type: basic types (byte, word, dword etc.) or complex ones (e.g. structures).

Creating an array

To create an array:

1. Create the first item;
2. Choose "Array..." from the context menu, or press *;
3. Fill in at least the *Array size* field and click OK.

Step 1 is optional; if no data item exists at the current location, a byte array will be created.

Hint: if you select a range before pressing *, *Array size* will be pre-filled with the number of items which fits into the selected range.

Quick menu navigation

Array parameters affect how the array is displayed in the listing and can be set at the time the array is first created or any time later by pressing *.

- **Array size:** total number of elements in the array;
- **Items on a line:** how many items (at most) to print on one line. 0 means to print the maximum number which fits into the disassembly line;
- **Element print width:** how many characters to use for each element. Together with the previous parameter can be used for formatting arrays into nice-looking tables.
For example: 8 items per line, print width -f:

```
db 1, 2, 3, 4, 5, 6, 7, 8
db 9, 10, 11, 12, 13, 14, 15, 16
db 17, 18, 19, 20, 21, 22, 23, 24
db 25, 255, 255, 255, 255, 255, 255, 26
db 27, 28, 29, 30, 31, 32, 33, 34
db 35, 36, 37, 38, 39, 40, 41, 42
```

print width 0:

```
db 1, 2, 3, 4, 5, 6, 7, 8
db 9, 10, 11, 12, 13, 14, 15, 16
db 17, 18, 19, 20, 21, 22, 23, 24
db 25, 255, 255, 255, 255, 255, 255, 26
db 27, 28, 29, 30, 31, 32, 33, 34
db 35, 36, 37, 38, 39, 40, 41, 42
```

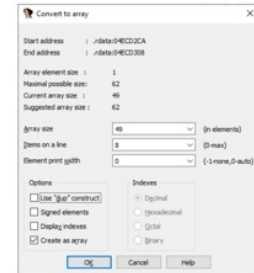
print width 5:

```
db 1, 2, 3, 4, 5, 6, 7, 8
db 9, 10, 11, 12, 13, 14, 15, 16
db 17, 18, 19, 20, 21, 22, 23, 24
db 25, 255, 255, 255, 255, 255, 255, 26
db 27, 28, 29, 30, 31, 32, 33, 34
db 35, 36, 37, 38, 39, 40, 41, 42
```

- Use "dup" construct: for assemblers that support it, repeated items with the same value will be collapsed into a dup expression instead of printing each item separately;

```
dup off: db 0FFh, 0FFh, 0FFh, 0FFh, 0FFh, 0FFh, 0FFh
dup on: db 6 dup(0FFh)
```

- **Signed elements:** integer items will be treated as signed numbers;
- **Display indexes:** for each line, first item's array index will be printed in a comment.
- **Create as array:** if unchecked, IDA will convert the array into separate items.

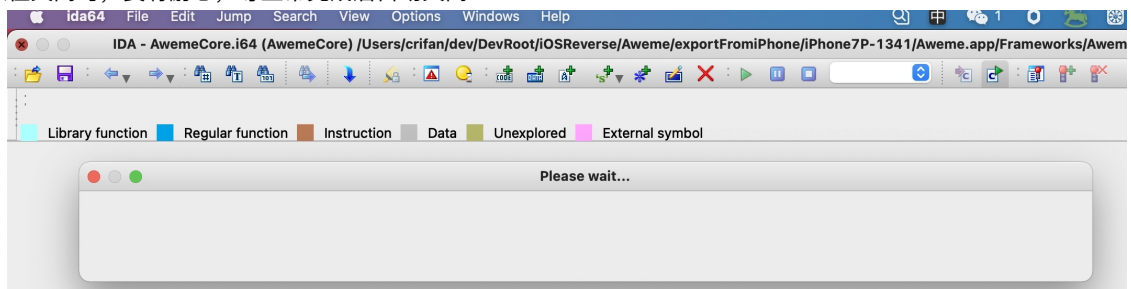


crifan.org, 使用署名4.0国际(CC BY 4.0)协议发布 all right reserved, powered by Gitbook最后更新: 2024-10-03 10:42:21

IDA的打开和关闭

IDA，对于打开比较大（比如100多MB）的二进制的话：

- 打开：
 - 首次打开：需要的解析时间很长
 - 再次打开：也需要点时间（大概几十秒，根据已解析的数据库大小决定）
- 关闭：保存改动，写入数据库，也会耗时较长
 - 所以在关闭时，要有耐心，等正常完成后自动关闭



打开

双击 .i64 文件，可以调用IDA（中的 ida64 ）去打开：



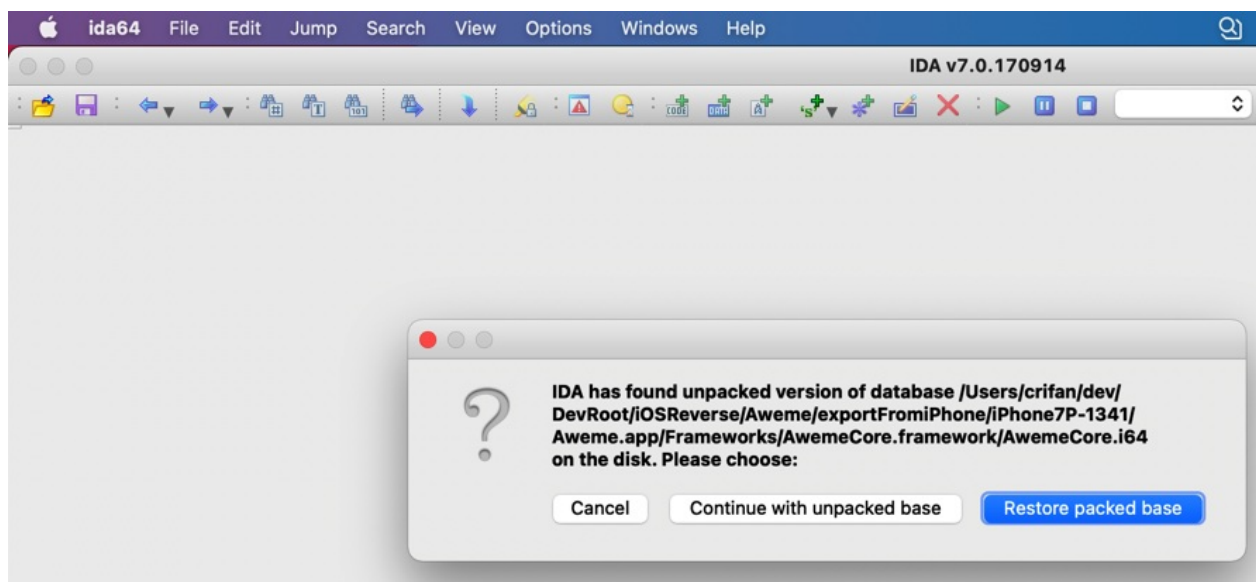
即可正常打开。

如果之前没有正常关闭，则会提示：

IDA has found unpacked version of database on the disk. Please choose:

- Cancel
- Continue with unpacked base
- Restore packed base

一般选：Restore packed base



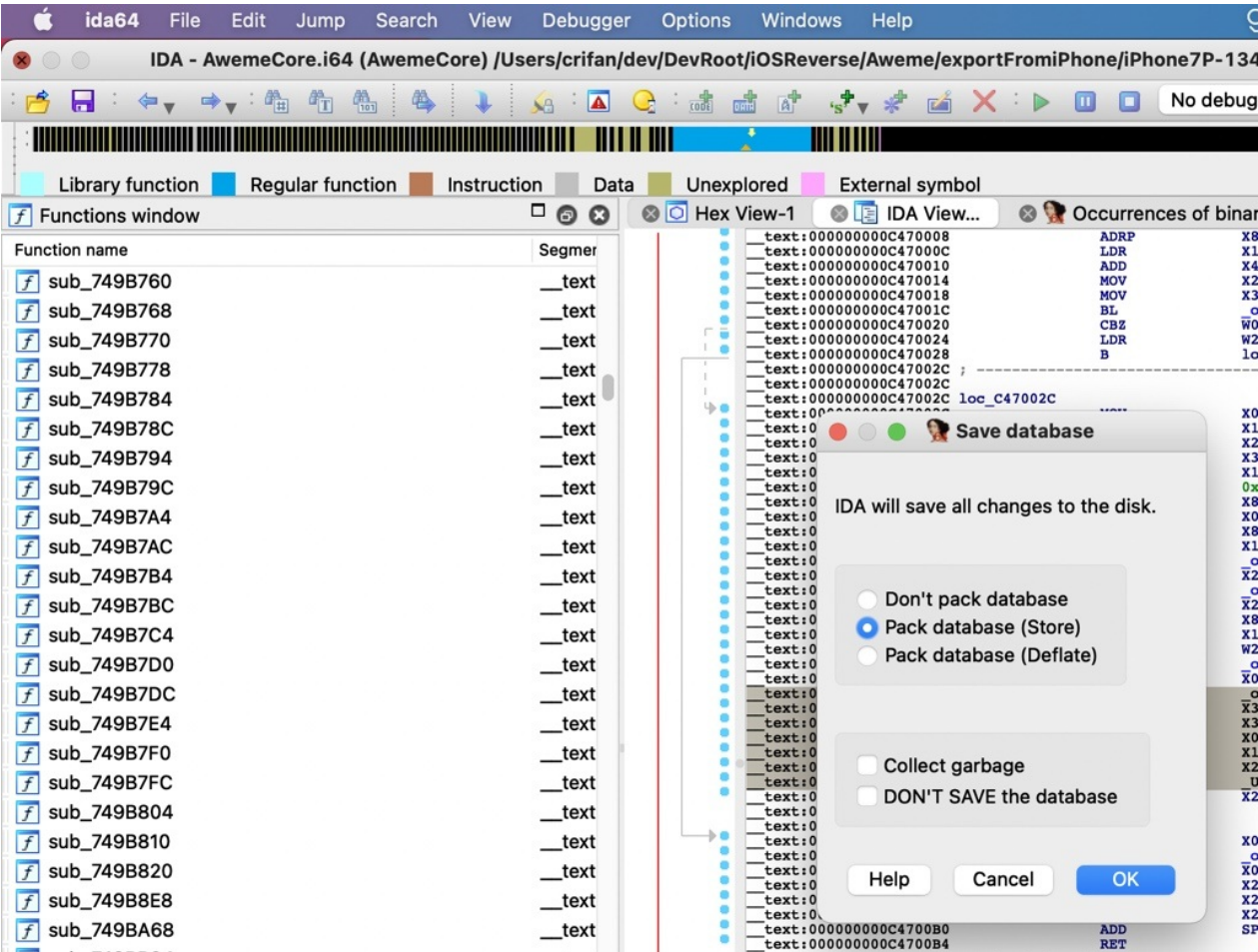
即可。

关闭

去点击关闭时，一般会有提示：

IDA will save all changes to the disk.

- Don't pack database
- Pack database (Store)
- Pack database (Deflate)



一般选默认的： Pack database (Store)

然后保存出的是单个文件： .i64：

AwemeCore.framework				
名称	大小	修改日期	种类	
> _CodeSignature	--	昨天 下午 3:00	文件夹	
AwemeCore	240.7 MB	昨天 下午 3:00	文稿	
AwemeCore.i64	4.3 GB	今天 上午 11:03	IDA Pro (64-bit) Database	
Info.plist	774 字节	昨天 下午 3:00	Property List	

关闭的心得

- 没有特殊情况时，千万不要轻易在关闭时，强制杀掉进程
 - 否则可能会导致：之前的数据库被损坏，再次打开后，之前分析的数据丢失了
 - 比如自己的优化改动，比如给函数变量重命名等

比如之前自己就遇到过：

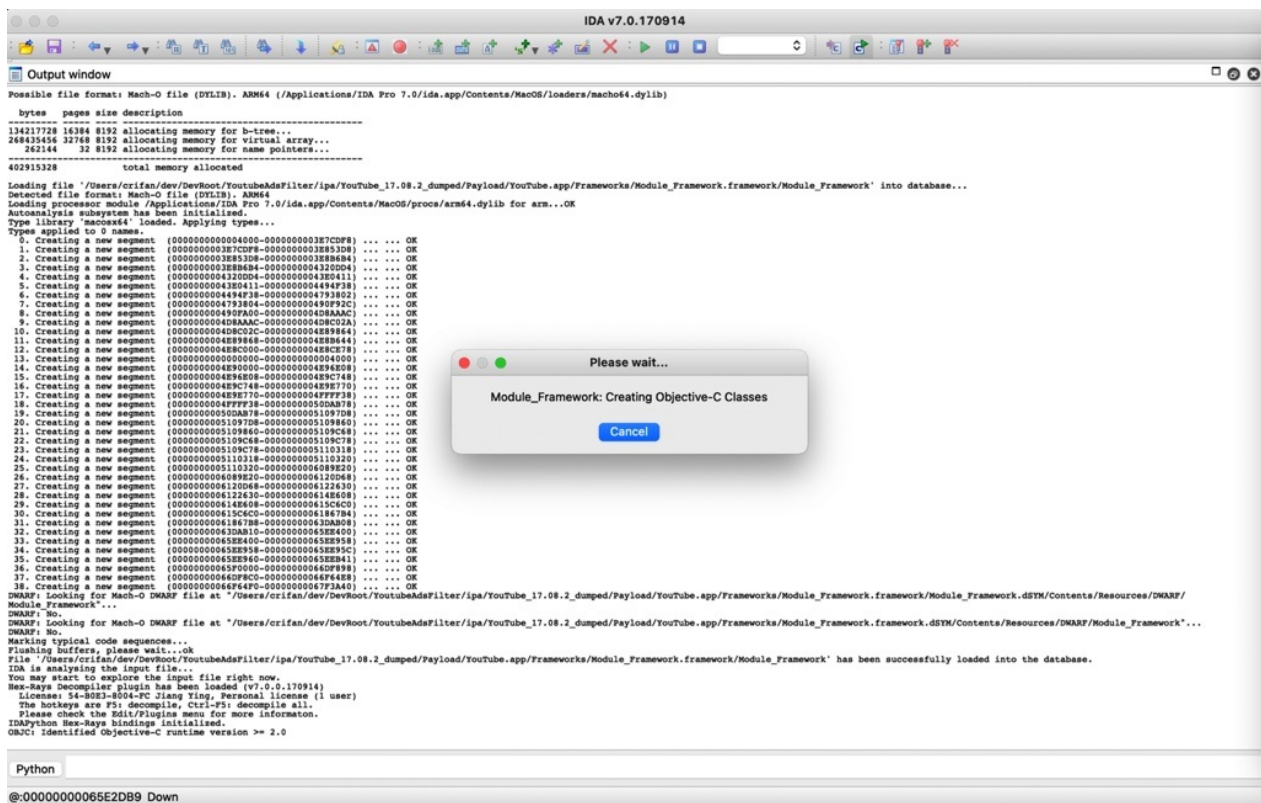
折腾：

【未解决】IDA中用idat64的Batch Mode尝试反编译导出YouTube的Module_Framework全部代码伪代码

期间，对于IDA的text mode的导出过程，觉得是卡死了，强制杀掉了IDA进程

再去打开IDA的GUI mode，即正常通过IDA图标双击打开，结果报错了

只好去，删除残留文件，全新的加载打开



crifan.org, 使用署名4.0国际(CC BY 4.0)协议发布 all right reserved, powered by Gitbook最后更新: 2024-10-03 10:42:40

ObjC

TODO:

- 【已解决】研究抖音设备注册逻辑：___lldb_unnamed_symbol580715\$\$AwemeCore
- 【已解决】iOS逆向心得：如何从对x8的adrp和ldr计算出对应的qword字符串值

背景知识

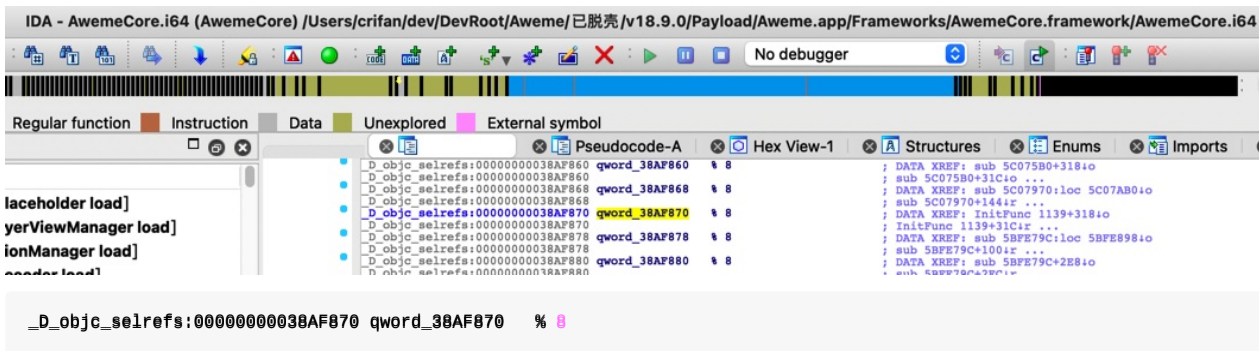
- SectionName 和描述
- __objc_imageinfo 记录 Objective-C 环境信息等，dyld 用它来判断镜像是否是 objc 镜像
- __objc_classlist 记录镜像所定义的类，每个条目都是一个指针，指向到 __objc_data section
- __objc_data 存放真正的类数据，和 __objc_classlist 条目呼应
- __objc_classname 类名列表
- __objc_methodname 方法名列表
- __objc_methodtype 方法类型列表
- __objc_selrefs selector 列表信息，每个条目是指向到 __objc_methname 的指针，记录 selector 的名字
- __objc_classrefs 类引用列表
- __objc_ivar 类的成员变量列表
- __objc_const 存放类的元数据，包括：method list、variable list、property list、class info

->MachOView可以查看二进制的这些段的信息：

RAW	RVA				
		Offset	Data	Description	Value
Section64 (__TEXT,__stub_helper)					
▼ Section64 (__TEXT,__cstring)		068F4D60	0000000102352EFB	Pointer	0x102BACD60->0x102352EFB:"setupSectionNodeData"
C String Literals		068F4D68	0000000102352F10	Pointer	0x102BACD68->0x102352F10:"classifyService"
Section64 (__TEXT,__ustring)		068F4D70	0000000102352F20	Pointer	0x102BACD70->0x102352F20:"insertAddData:success:"
▶ Section64 (__TEXT,__objc_methname)		068F4D78	00000001023174F2	Pointer	0x102BACD78->0x1023174F2:"loadAdsListFromFileWithClass:callBack"
▶ Section64 (__TEXT,__objc_classname)		068F4D80	0000000102352F36	Pointer	0x102BACD80->0x102352F36:"insertLiveListData:isMore:"
▶ Section64 (__TEXT,__objc_methtype)		068F4D88	0000000102317493	Pointer	0x102BACD88->0x102317493:"loadClassifyListFromFileWithClass:"
Section64 (__TEXT,__const)		068F4D90	000000010234F88E	Pointer	0x102BACD90->0x10234F88E:"hasEnterOneLive"
Section64 (__TEXT,__gcc_except_tab)		068F4D98	000000010234666A	Pointer	0x102BACD98->0x10234666A:"showAppstoreToCommentInView:"
Section64 (__TEXT,__unwind_info)		068F4DA0	0000000102317485	Pointer	0x102BACDA0->0x102317485:"fetchAdsInfo:"
▶ Section64 (__TEXT,__eh_frame)		068F4DA8	0000000102352F51	Pointer	0x102BACDA8->0x102352F51:"insertRanksData:"
▶ Section64 (__DATA,__got)		068F4DB0	000000010232170F	Pointer	0x102BACDB0->0x10232170F:"fetchAnchorsRankBlock:"
▶ Section64 (__DATA,__la_symbol_ptr)		068F4DB8	00000001023173C7	Pointer	0x102BACDB8->0x1023173C7:"fetchClassifyListForPullDownWithGe"
▶ Section64 (__DATA,__mod_init_func)		068F4DC0	000000010233F8D3	Pointer	0x102BACDC0->0x10233F8D3:"setIPAddressForLive"
Section64 (__DATA,__const)		068F4DC8	0000000102317454	Pointer	0x102BACDC8->0x102317454:"fetchClassifyListForPullUpWithGend"
▶ Section64 (__DATA,__cstring)		068F4DD0	000000010233E2D8	Pointer	0x102BACDD0->0x10233E2D8:"resetData"
▶ Section64 (__DATA,__objc_classlist)		068F4DD8	0000000102352F62	Pointer	0x102BACDD8->0x102352F62:"buildLiveListData:isMore:"
Section64 (__DATA,__objc_nclclist)		068F4DE0	0000000102326CFB	Pointer	0x102BACDE0->0x102326CFB:"buildDiscoveryArrayWith:"
▶ Section64 (__DATA,__objc_catlist)		068F4DE8	0000000102352FE7	Pointer	0x102BACDE8->0x102352FE7:"textFieldDidChange:"
Section64 (__DATA,__objc_nlcattlist)		068F4DF0	0000000102352FFB	Pointer	0x102BACDF0->0x102352FFB:"setPriceTextField:"
▶ Section64 (__DATA,__objc_protolist)		068F4DF8	000000010235300E	Pointer	0x102BACDF8->0x10235300E:"setMaximumFractionDigits:"
▶ Section64 (__DATA,__objc_imageinfo)		068F4E00	0000000102353028	Pointer	0x102BACE00->0x102353028:"numberFromString:"
Section64 (__DATA,__objc_const)		068F4E08	000000010235303A	Pointer	0x102BACE08->0x10235303A:"inputLimitMaxLength"
▼ Section64 (__DATA,__objc_selrefs)		068F4E10	000000010235304E	Pointer	0x102BACE10->0x10235304E:"u1SubstringToMaxLenth:textField:"
Literal Pointers		068F4E18	000000010235306F	Pointer	0x102BACE18->0x10235306F:"editCallback"
Section64 (__DATA,__objc_protorefs)		068F4E20	0000000102338EF9	Pointer	0x102BACE20->0x102338EF9:"questionPrice"
▶ Section64 (__DATA,__objc_classrefs)		068F4E28	000000010235307C	Pointer	0x102BACE28->0x10235307C:"editStringCallback"
▶ Section64 (__DATA,__objc_superrefs)		068F4E30	0000000102353141	Pointer	0x102BACE30->0x102353141:"initWithRoomInfo:user:"
Section64 (__DATA,__objc_ivar)		068F4E38	0000000102353158	Pointer	0x102BACE38->0x102353158:"haveCacheData"
Section64 (__DATA,__objc_data)		068F4E40	0000000102353E57	Pointer	0x102BACE40->0x102353E57:"updateRoomInfoWithRoomId:"
Section64 (__DATA,__data)		068F4E48	0000000102350EBE	Pointer	0x102BACE48->0x102350EBE:"requestOperalListSuccessBlock:failB"
▶ Function Starts		068F4E50	0000000102350EE6	Pointer	0x102BACE50->0x102350EE6:"uploadLikeNumberLikeNumber:"
▶ Data in Code Entries		068F4E58	0000000102350F02	Pointer	0x102BACE58->0x102350F02:"downloadVideoRoomSuccessBlock:fail"

数据定义含义

_D_objc_selrefs



中的含义是：

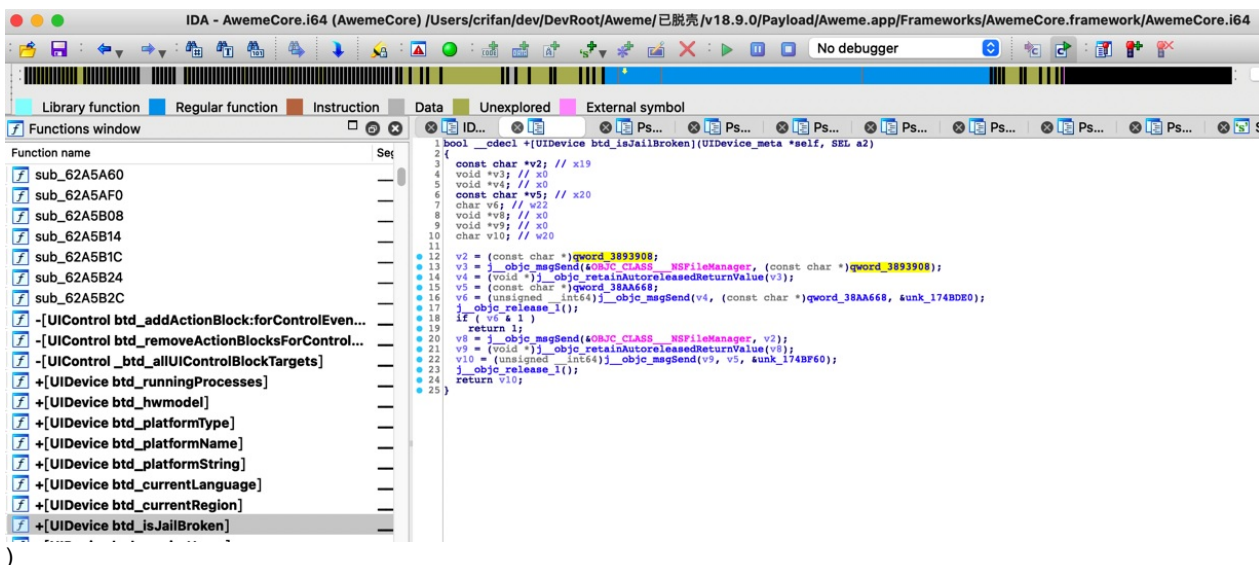
- `_D_objc_selrefs`
 - ObjC的section段 `_objc_selrefs` 中的 `D` = Data = 数据
- `%`
 - align对齐？
- `8`
 - 以 8 字节对齐？

调试出const char* qword是字符串并改名优化定义和伪代码

- 背景

IDA分析抖音AwemeCore时，发现：

伪代码：



```
bool __cdecl __btd_isJailBroken(UIDevice meta self, SEL a2)
{
    const char *v2; // x19
    void *v3; // x0
    void *v4; // x0
    const char *v5; // x20
    char v6; // w22
    void *v7; // x0
    void *v8; // x0
    void *v9; // x0
    char v10; // w20

    v2 = (const char *)qword_3893908;
    v3 = j__objc_msgSend( objc_class__NSFileManager, (const char *)qword_3893908);
    v4 = (void *)j__objc_retainAutoreleasedReturnValue(v3);
    v5 = (const char *)qword_38AA668;
```



```

v6 = (unsigned __int64)j__objc_msgSend(v4, (const char *)qword_38AA668, unk_174BDE0);
j__objc_release_1();
if ( v6 & 1 )
    return 1;
v8 = j__objc_msgSend( OBJC_CLASS__NSFileManager, v2);
v9 = (void *)j__objc_retainAutoreleasedReturnValue(v8);
v10 = (unsigned __int64)j__objc_msgSend(v9, v5, unk_174BF60);
j__objc_release_1();
return v10;
}

```

希望知道 qword_3893908 是什么内容，搞懂后，改名，优化伪代码和定义。

- 过程

经过调试：

```

(lldb) po (BOOL)[(UIDevice*)0x1dfc62a88 btd_isJailBroken]
2022-03-28 14:55:19.836905+0800 Aweme[39040:2680736] hook_openFile_iOS.xm NSFileManager$fileExistsAtPath$: path /Applications/Cydia.app -> isJbPath=True -> isExists=False
2022-03-28 14:55:19.838832+0800 Aweme[39040:2680736] hook_openFile_iOS.xm NSFileManager$fileExistsAtPath$: path /private/var/lib/apt -> isJbPath=True -> isExists=False

```

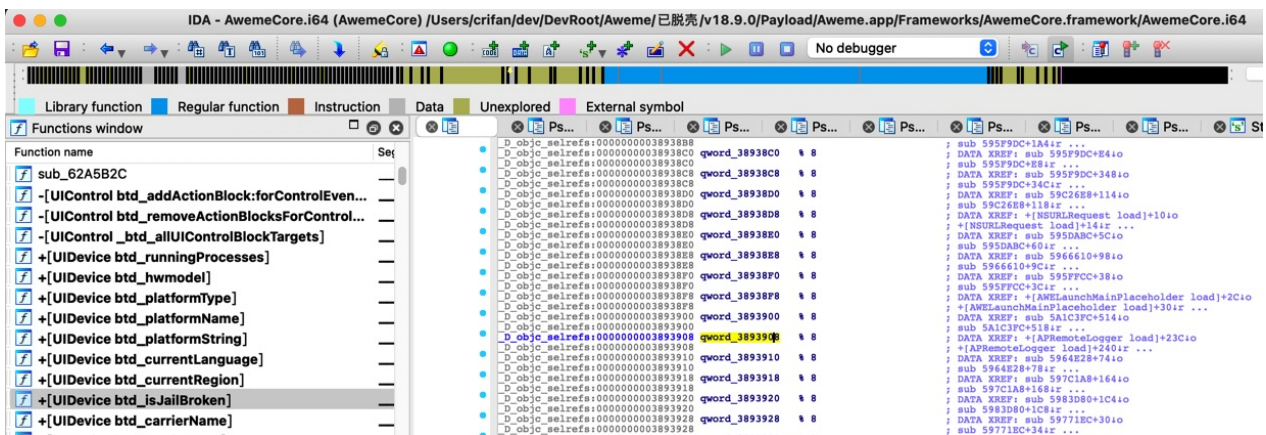
得知 qword_3893908 应该是其中一个越狱路径字符串

但是IDA中却看不出来，只能看出定义是：

```

_D_objc_selrefs:0000000003893908 qword_3893908 8 ; DATA XREF:+[APRemoteLogger load]+23C:0
_D_objc_selrefs:0000000003893908 ;+[APRemoteLogger load]+240:r ...

```



参考资料地址：

- dword
 - 具体的含义，取决于实际情况
 - 指针
 - 指针的指针（地址）
 - 指针的指针的指针
 - 等等
 - 总之就是：
 - offset dword_XXXX = dword_XXXX 的原始地址 = 其指向一个东西 = 具体是啥东西，取决于实际情况
 - 此处抖音 AwemeCore 中，往往是一个字符串
 - 只不过IDA静态分析，看不出具体值
 - qword_xxx
 - 虽然多数情况下是字符串
 - IDA 伪代码中常见写法：(char *)qword_xxx
 - 但是偶尔也有：非字符串的，比如是某个类，比如
 - < 0x281d67bf0>{recursion count = 0, name = nil}

- IDA伪代码写法: `(void *)qword_56AB210`

后续得知，可以通过调试去打印出具体值：

核心逻辑是：

`qword_xxx` 的 `xxx` 是二进制内偏移量 + 二进制的ALSR = 实际（字符串的）地址

去查看：[实际（字符串的）地址] = （即可查看到）保存了对应的字符串

举几个例子：

```
(lldb) image list -o -f | grep AwemeCore
[ 0] 0x0000000104acc000 /Users/crifan/Library/Developer/Xcode/DerivedData/Aweme-fswcidjoxbkibsdwekuzlsfcdqls/Build/P
roducts/Debug-iphonios/Aweme.app/Frameworks/AwemeCore.framework/AwemeCore

qword_3896270

(lldb) p/x 0x0000000104acc000 + 0x3896270
(long) $15 = 0x0000000108362270
(lldb) x/1gx 0x0000000108362270
0x108362270: 0x0000000105859bc6
(lldb) po (char*)0x0000000105859bc6
"onTheFlyParameter"
->
qword_3896270 = "onTheFlyParameter"

qword_38B2148

(lldb) p/x 0x0000000104acc000 + 0x38B2148
(long) $17 = 0x000000010837e148
(lldb) x/1gx 0x000000010837e148
0x10837e148: 0x00000001c5619497
(lldb) po (char*)0x00000001c5619497
"dictionaryWithDictionary:"
- qword_38B2148 = "dictionaryWithDictionary:"

qword_3925208

(lldb) p/x 0x0000000104acc000 + 0x3925208
(long) $19 = 0x00000001083f1208
(lldb) x/1gx 0x00000001083f1208
0x1083f1208: 0x000000010957bb48
(lldb) po (char*)0x000000010957bb48
"configParams"
- qword_3925208 = "configParams"

qword_38A1D98

(lldb) p/x 0x0000000104acc000 + 0x38A1D98
(long) $21 = 0x000000010836dd98
(lldb) x/1gx 0x000000010836dd98
0x10836dd98: 0x0000000105892952
(lldb) po (char*)0x0000000105892952
"ttinstall_stringValueForKey:"
- qword_38A1D98 = "ttinstall_stringValueForKey:"
```

后续还遇到其他类似例子：

- IDA的伪代码中`qword`没有`char*`的强制转换如果是`_D_objc_selrefs`则表示是字符串

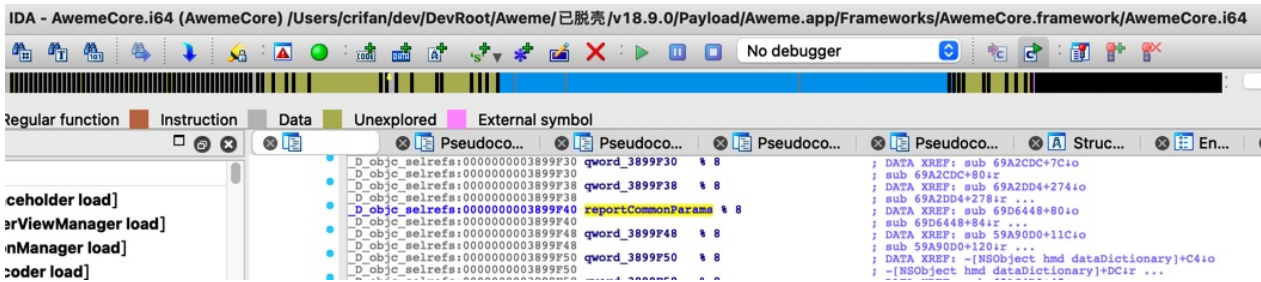
IDA伪代码中：

```
if ( (unsigned int)jmp_objc_msgSend_D523EEC(v0, respondsToSelector_1, qword_3899F40) )
```

虽然 `qword_3899F40` 前面没有 `const char *` 或 `char *` 的强制转换。

但是，看到定义中（修改值后）是：

```
_D_objc_selrefs:000000003899F40 reportCommonParams 8 ; DATA XREF: sub_69D6448+8010
_D_objc_selrefs:000000003899F40 ; sub_69D6448+841r ...
```



即是 objc_selrefs 的类型，即是 objc 的函数的自我引用

-》则可以推断出就是：某个类的某个函数名

所以也就是：字符串类型的值

所以此处去计算和转换成 char * 是合理的

所以算出来是：

- qword_3899F40 -> 0x3899F40

```
(lldb) p/x 0x00000001037b4000 + 0x3899F40
(long) $1 = 0x000000010704df40
(lldb) x/lgx 0x000000010704df40
0x10704df40: 0x00000001045537b4
(lldb) po (char*)0x00000001045537b4
"reportCommonParams"
```

然后再去优化伪代码，改为：

```
if ( (unsigned int)jmp_objc_msgSend_D523EEC(v0, respondsToSelector_1, reportCommonParams) )
```

即可。

crifan.org, 使用署名4.0国际(CC BY 4.0)协议发布 all right reserved, powered by Gitbook最后更新: 2023-07-14 23:04:13

IDA动态调试

IDA支持动态调试程序，包括 `ios` 的app和 `Android` 的app

crifan.org，使用[署名4.0国际\(CC BY 4.0\)协议](#)发布 all right reserved，powered by Gitbook最后更新：2024-10-01 16:22:43

iOS

IDA支持动态调试iOS的app。

此处以 WhatsApp 为例说明具体过程。

crifan.org, 使用署名4.0国际(CC BY 4.0)协议发布 all right reserved, powered by Gitbook最后更新: 2024-10-01 16:23:07

搭建环境

此处IDA中初始化调试iOS的app：WhatsApp的过程是：

获取二进制完整路径

参考：

[二进制文件路径 · iOS逆向：心得集锦](#)

此处去用CocoaTop/爱思助手等，获取到二进制路径：

- iOS的app：WhatsApp
 - 主二进制（在iPhone7中）的完整路径
 - /var/containers/Bundle/Application/2BE964D4-8DF0-4858-A06D-66CA8741ACDC/WhatsApp.app/WhatsApp

先（给某个函数）加断点

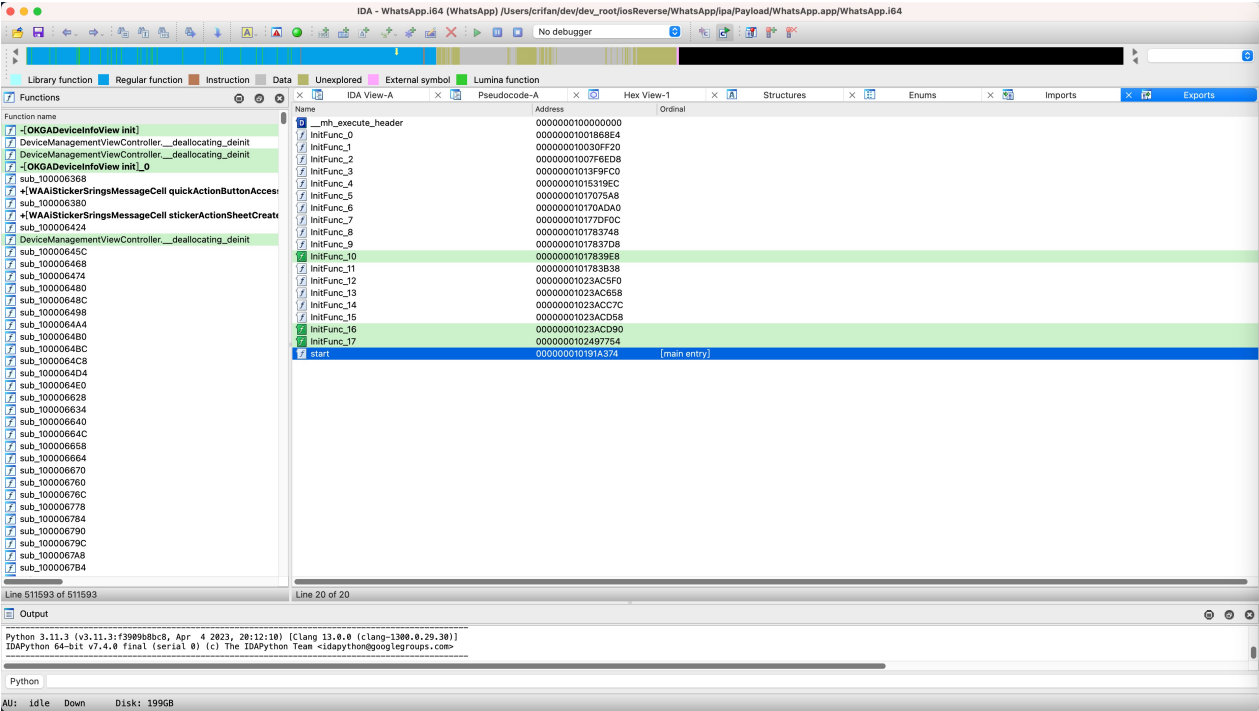
此处选择入口函数：

WhatsApp的入口函数是：start

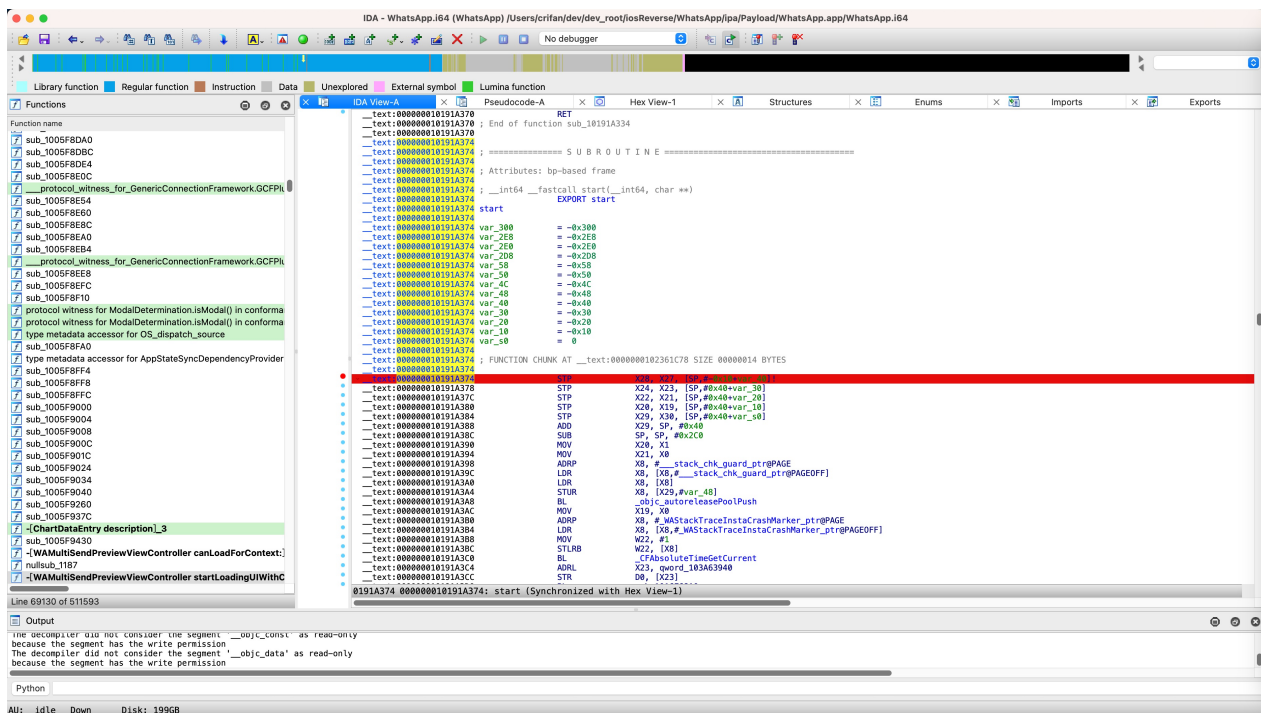
可以在IDA的 Exports 中找到：

```
start      000000010191A374  [main entry]
```

对应的地址：0x10191A374

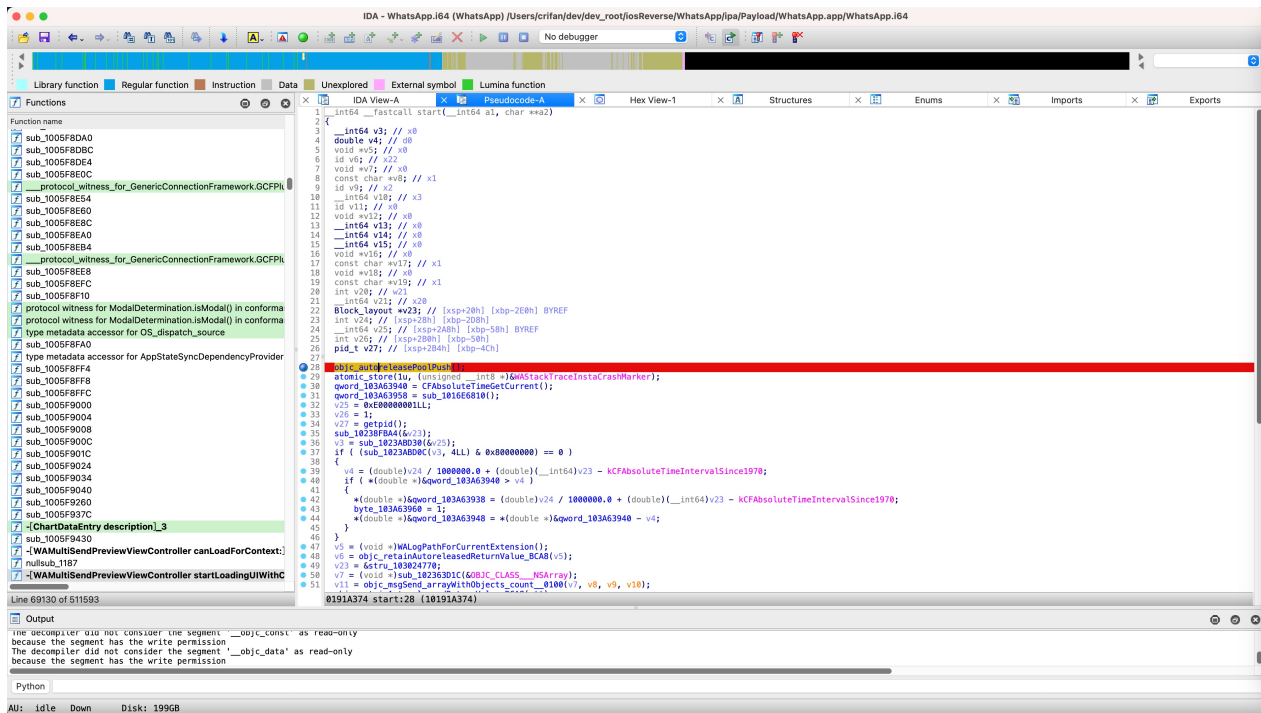


双击进入IDA汇编代码，加上断点：



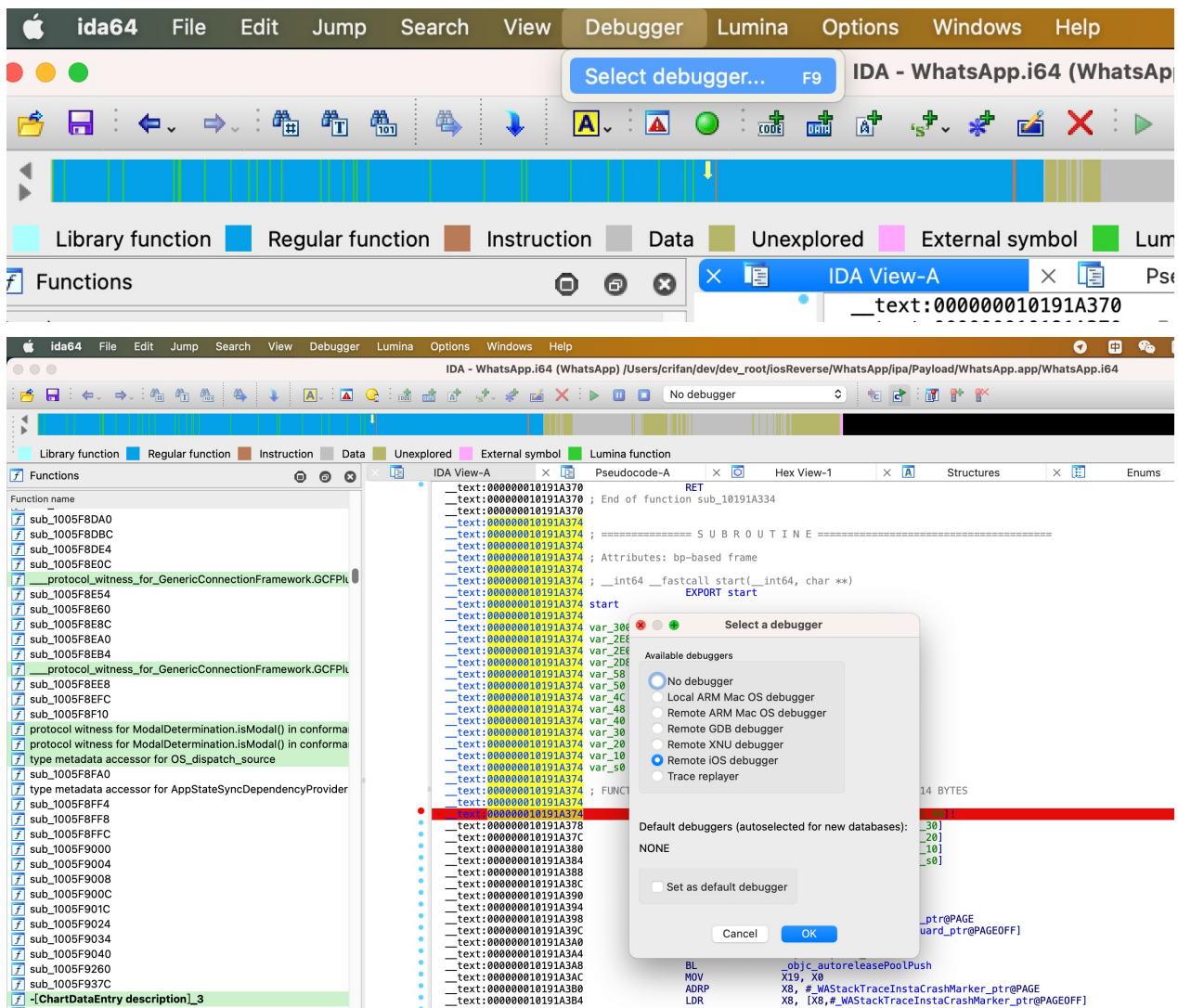
注：

F5伪代码，也是可以加断点的：



设置debugger调试器： Remote iOS Debugger

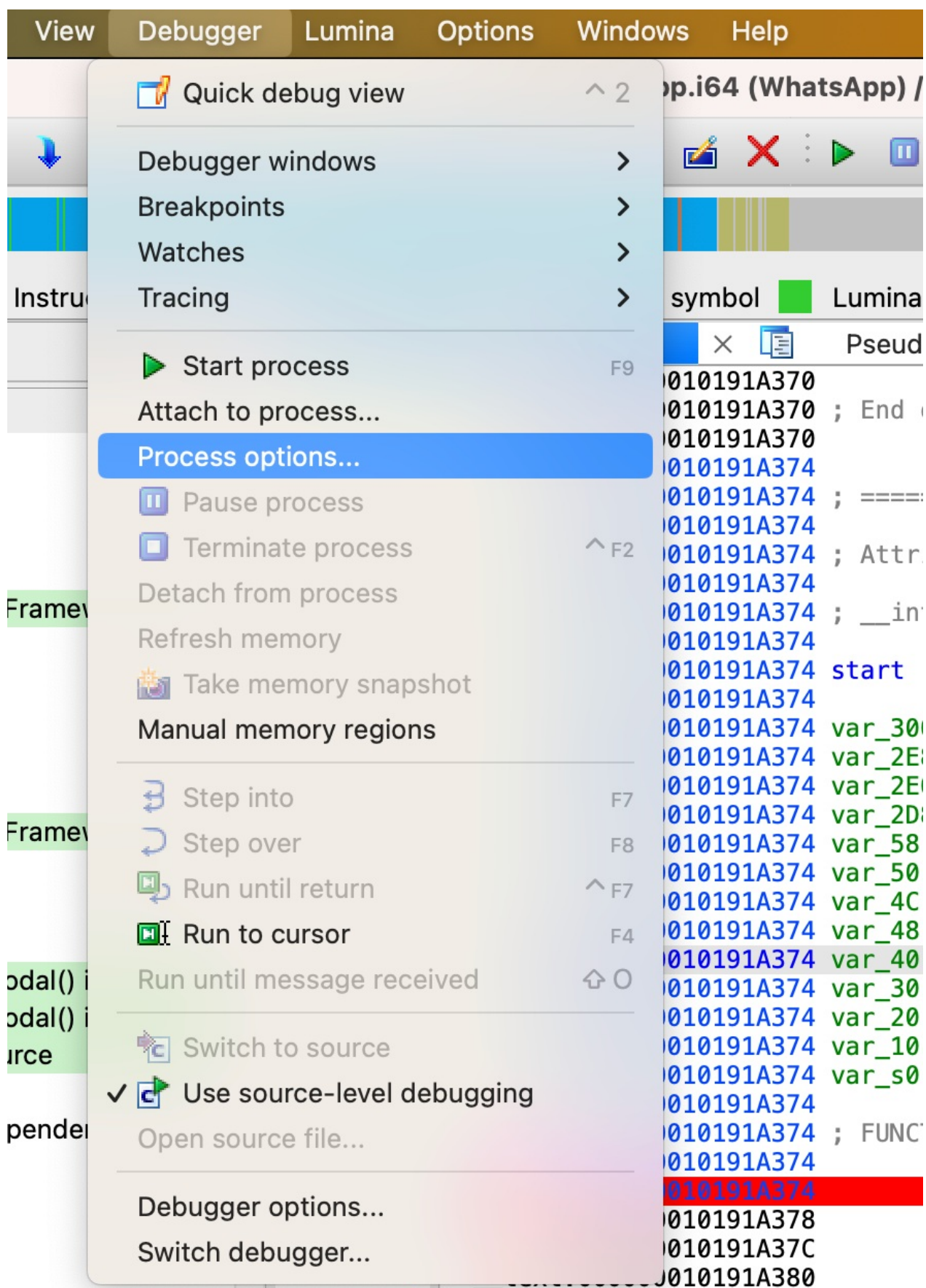
IDA -> Debugger -> Select debugger -> Remote iOS debugger

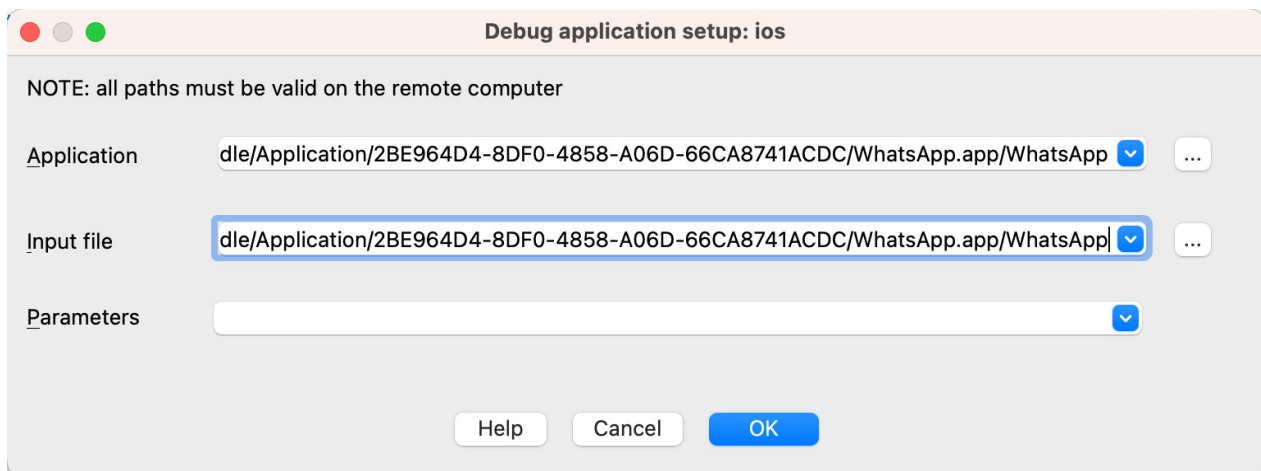


设置进程选项

IDA -> Debugger -> Process options -> 给:

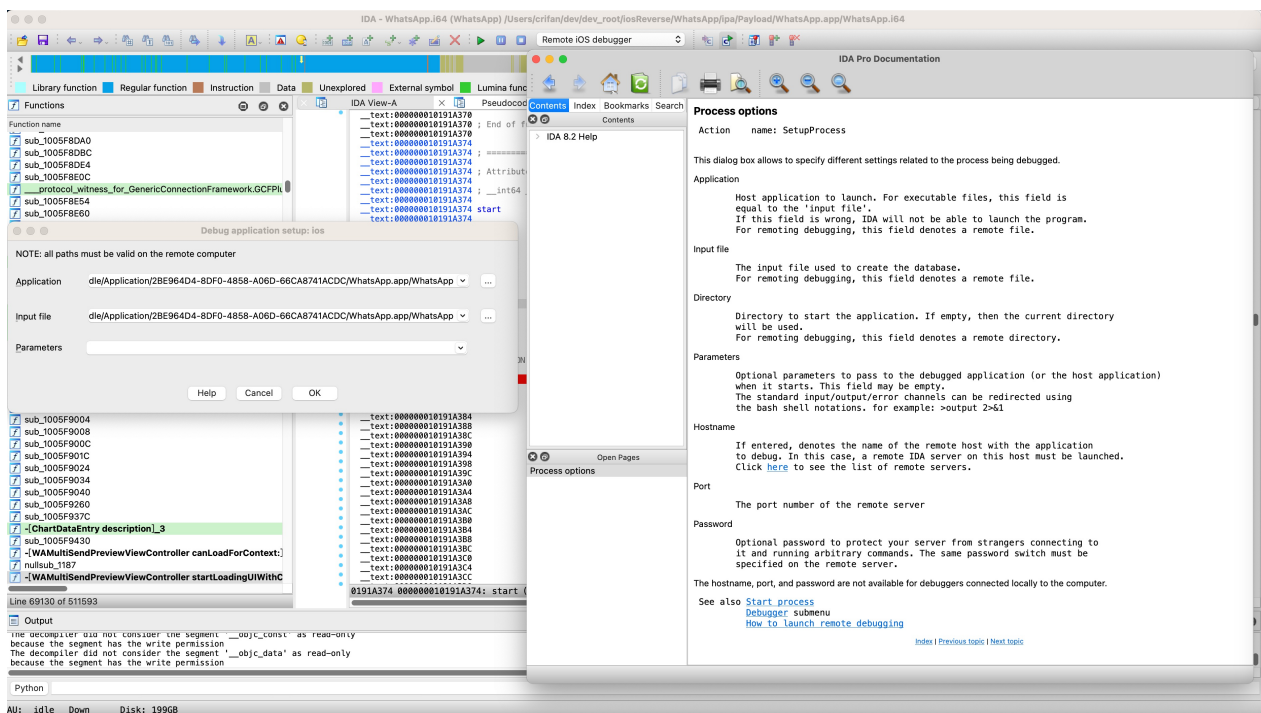
- Application和Input File都设置为: WhatsApp的二进制完整路径
 - /var/containers/Bundle/Application/2BE964D4-8DF0-4858-A06D-66CA8741ACDC/WhatsApp.app/WhatsApp





注：

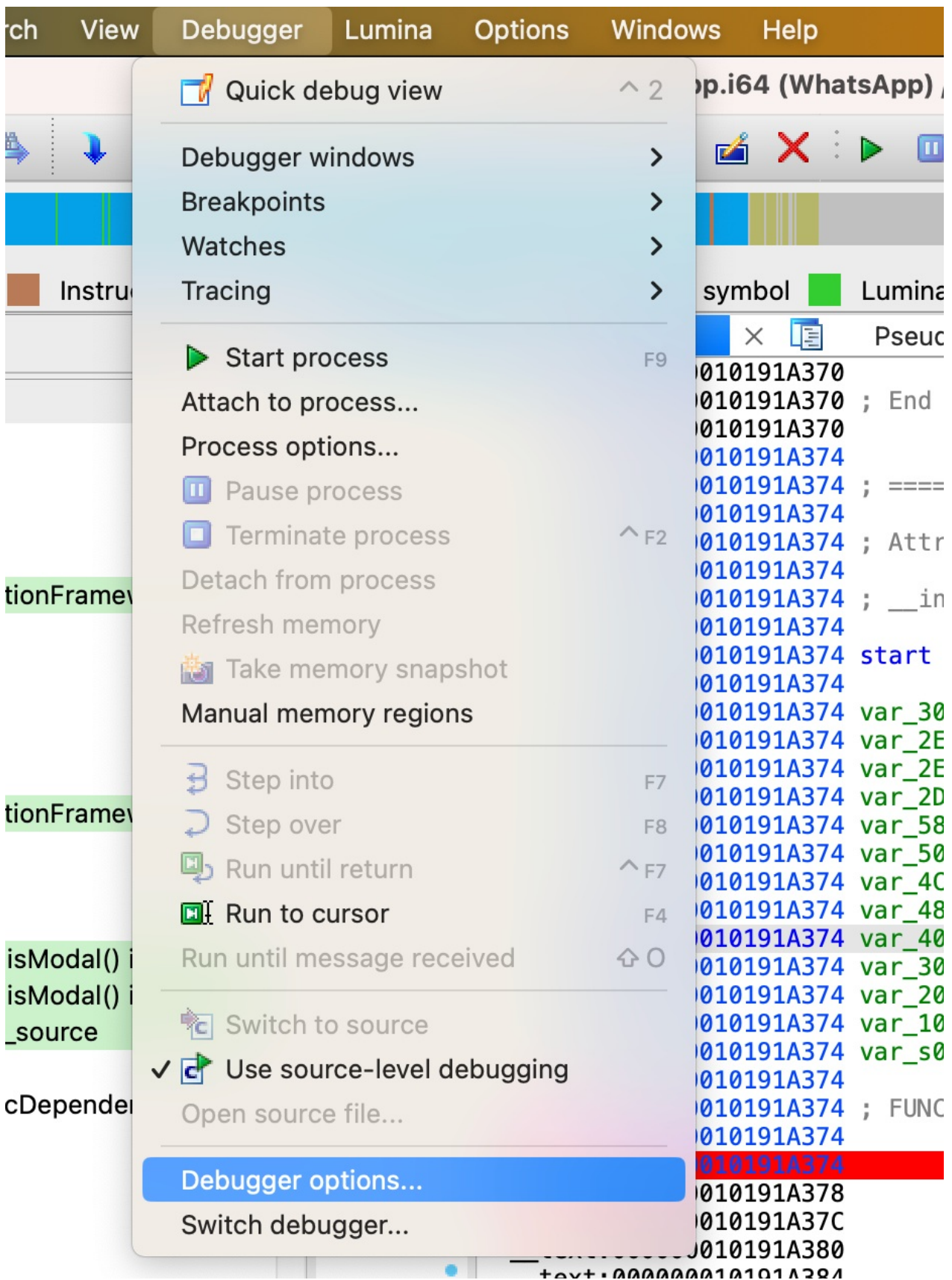
点击Help，可以查看参数含义解释：

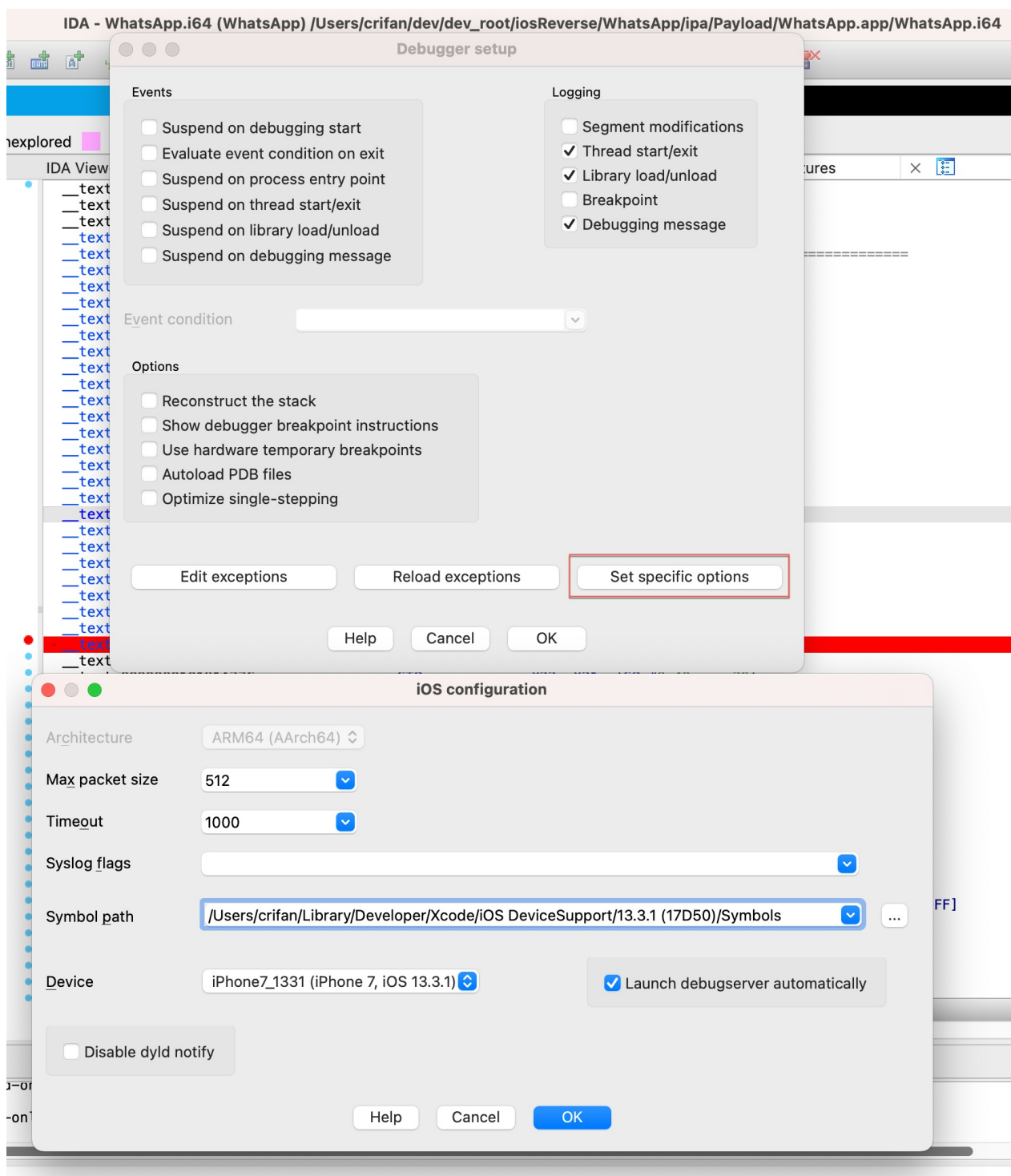


设置调试器选项

IDA -> Debugger -> Debugger options -> Set specific options -> iOS configuration

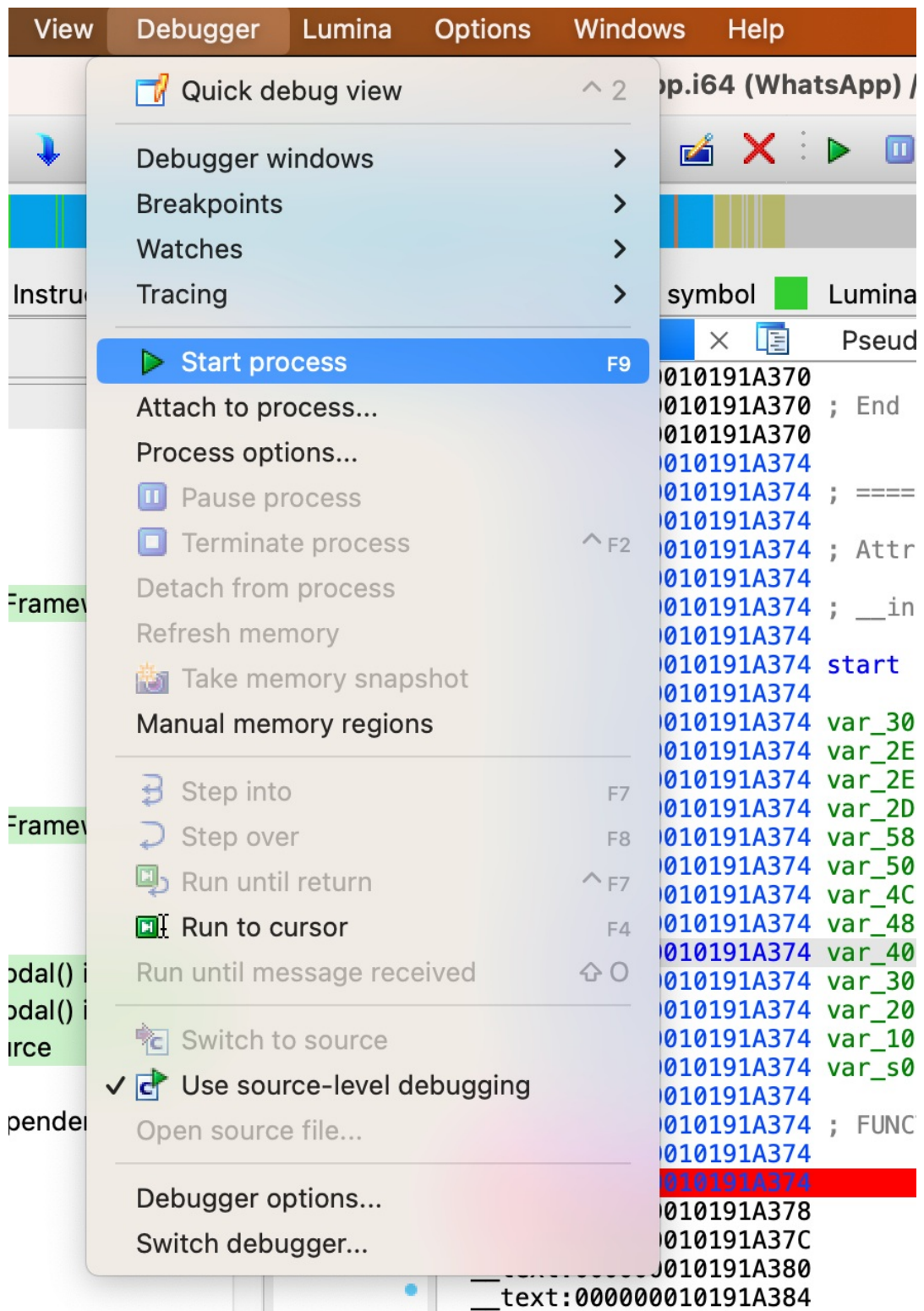
- Max packet size : 512
 - 注：默认值-1，暂不太清楚具体含义
- Timeout : 1000
- Symbol path : /Users/crifan/Library/Developer/Xcode/iOS DeviceSupport/13.3.1 (17D50)/Symbols
 - 可以从 ~/Library/Developer/Xcode/iOS DeviceSupport 找到具体的和你的iPhone设备匹配的上述路径
- Device : iPhone7_1331 (iPhone 7, iOS 13.3.1)
 - 默认已识别和选择当前USB连接到Mac的iPhone设备
 - (默认已) 勾选：Launch debugserver automatically





启动调试

- 有2种方式启动：
 - Spawn模式
 - IDA -> Debugger -> Start process



- Attach模式

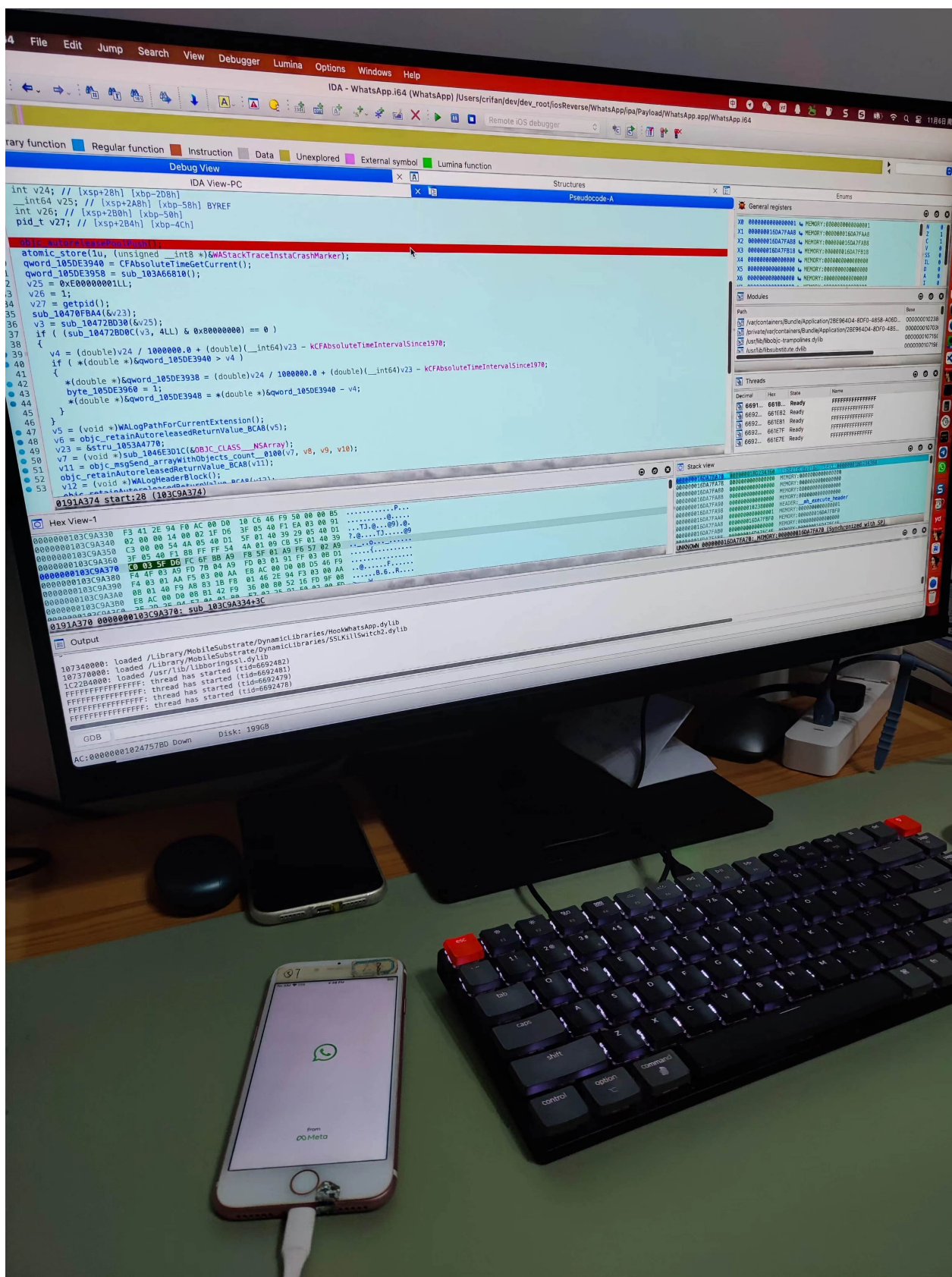
- IDA -> Debugger -> Attach to process

后续即可启动，大概过程是：

- Running
- Importing Symbols
 - 此处，此阶段像卡死了
 - 最终耗时10分钟才解析符号结束
 - 估计是符号表=函数太多，导致的？

然后就进去了IDA调试主界面：

The image shows a workspace for reverse engineering. The primary monitor displays IDA Pro, a disassembler and debugger. The main window shows assembly code for a function, with a red line highlighting a specific instruction. The right-hand pane shows a hex view of the code, and the bottom pane shows a list of threads. A laptop to the left also displays a debugger interface. A smartphone is connected to the monitor via a cable, and a keyboard and mouse are positioned in front of the monitor.



后续即可愉快的调试了。

附录

参考资料

- 网友的
 - [\[iOS Hacker\] IDA 7.0 动态调试 iOS App – exchen's blog](#)
- 官网的
 - 新
 - [Tutorial: Debugging iOS Applications with IDA Pro \(hex-rays.com\)](#)
 - Debugging iOS Applications with IDA Pro
 - Debugging DYLD
 - Debugging the DYLD Shared Cache
 - Debugging System Applications
 - Troubleshooting
 - 旧
 - https://www.hex-rays.com/wp-content/uploads/2019/12/ios_debugger_tutorial.pdf

crifan.org, 使用署名4.0国际(CC BY 4.0)协议发布 all right reserved, powered by Gitbook 最后更新: 2024-10-01 16:44:53

开始调试

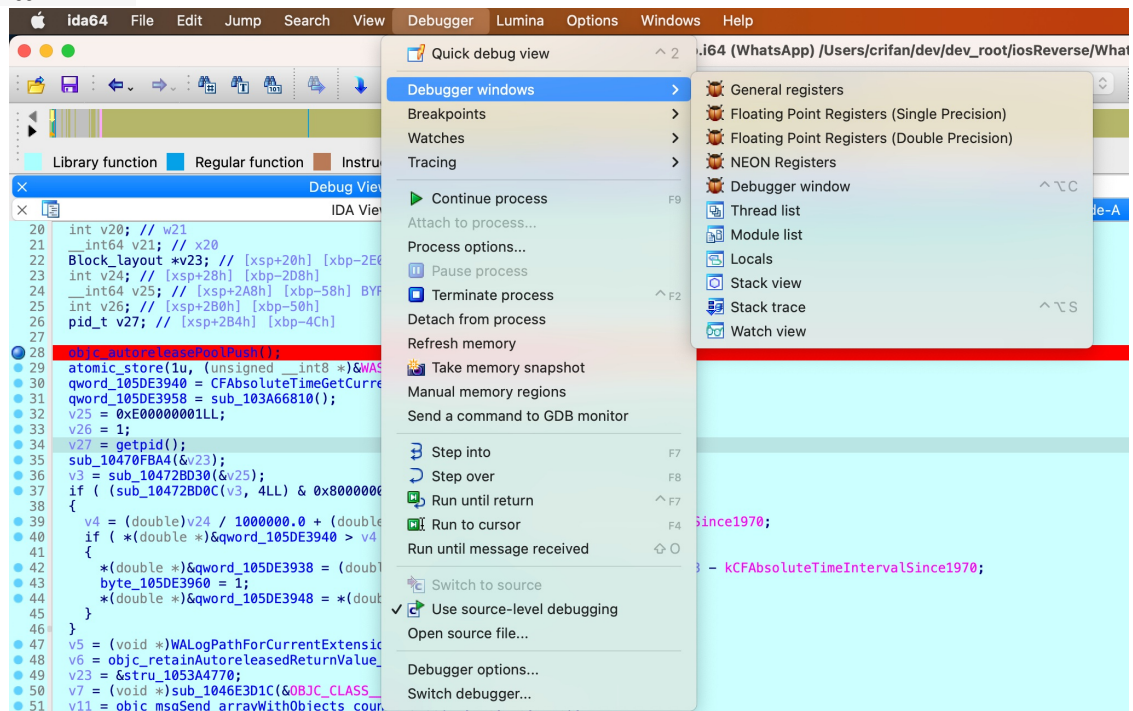
调试命令

- IDA基本调试命令
 - Step Into = F7 = 单步进入 = 单步步入
 - Step Over = F8 = 单步执行 = 单步步过
 - Run until return = Ctrl+F7 = 运行直到返回
 - Run to cursor = F4 = 运行到鼠标处
 - Run until message received = Alt+O

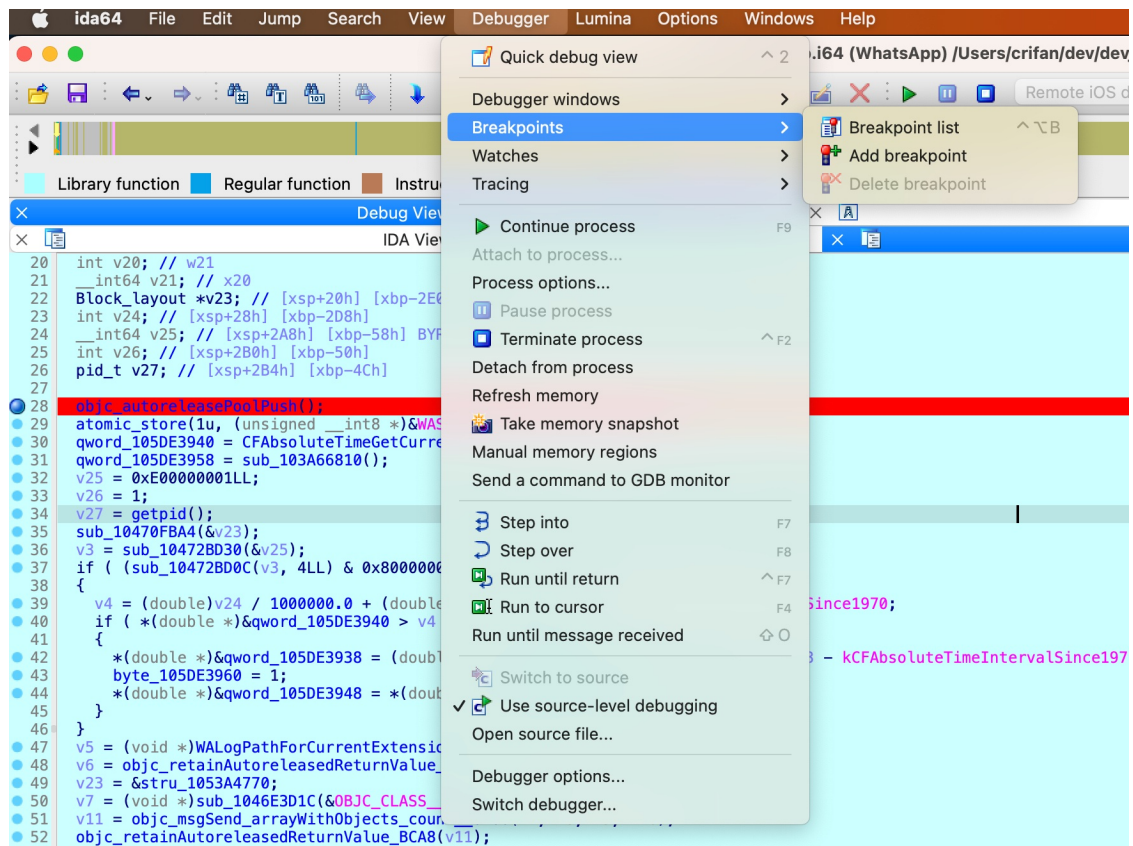
调试菜单

顺带看看相关菜单：

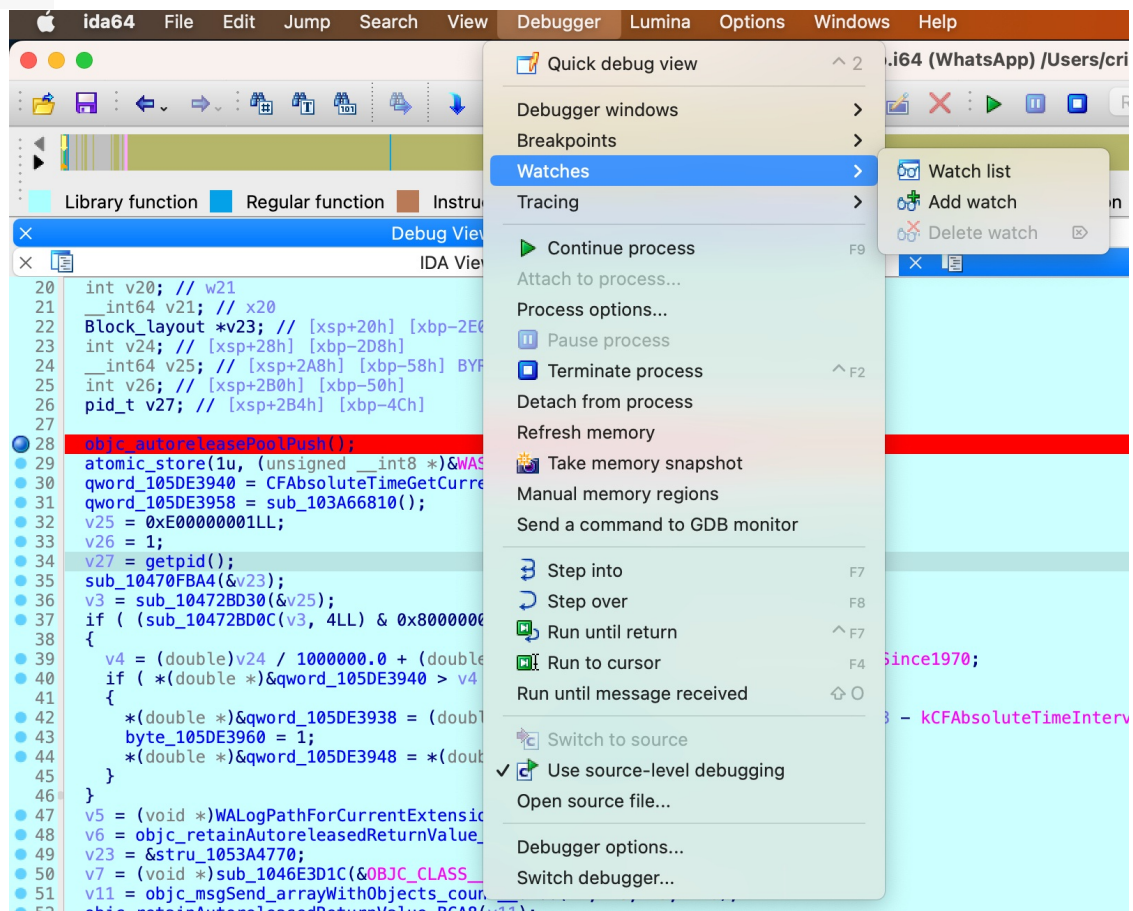
- IDA -> Debugger
 - Debugger window



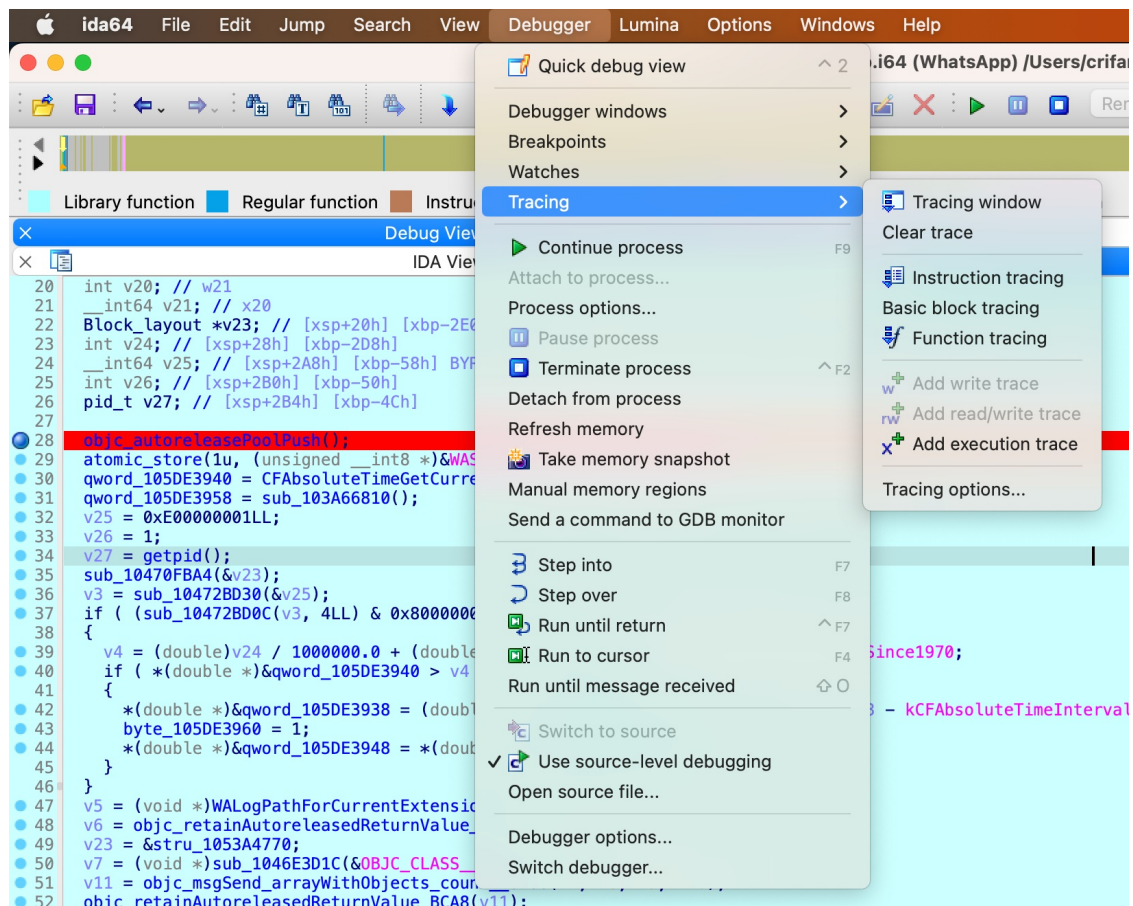
- Breakpoints



o Watches

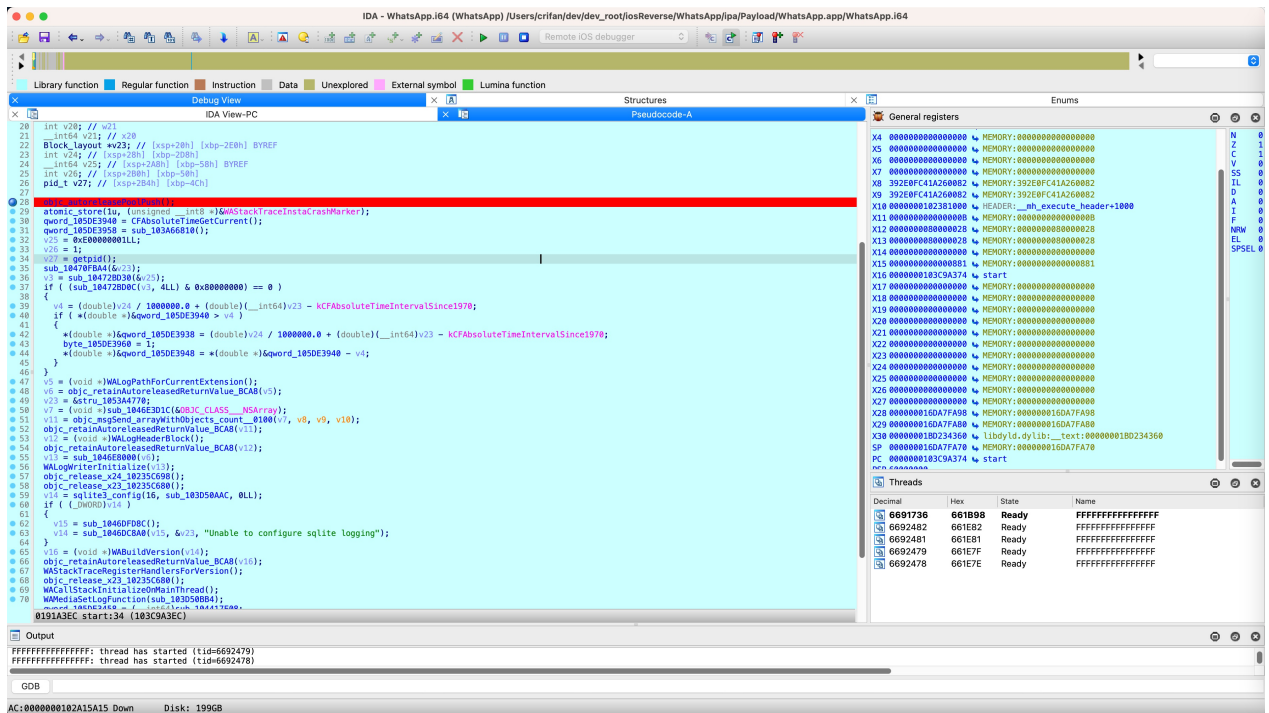


o Tracing



调试过程

之前已进入调试界面：



按了几次F8后，可以看到相关界面数据有变化：

ida64 File Edit Jump Search View Debugger Lumina Options Windows Help

IDA - WhatsApp.i64 (WhatsApp) /Users/crifan/dev/d

Library function Regular function Instruction Data Unexplored External symbol Lumina function

Debug View

IDA View-PC

```

15  __int64 v15; // x0
16  void *v16; // x0
17  const char *v17; // x1
18  void *v18; // x0
19  const char *v19; // x1
20  int v20; // w21
21  __int64 v21; // x20
22  Block_layout *v23; // [xsp+20h] [xbp-2E0h] BYREF
23  int v24; // [xsp+28h] [xbp-2D8h]
24  __int64 v25; // [xsp+2A8h] [xbp-58h] BYREF
25  int v26; // [xsp+2B0h] [xbp-50h]
26  pid_t v27; // [xsp+2B4h] [xbp-4Ch]
27
28  objc_autoreleasePoolPush();
29  atomic_store(1u, (unsigned __int8 *)&WASStackTraceInstaCrashMarker);
30  qword_105DE3940 = CFAbsoluteTimeGetCurrent();
31  qword_105DE3958 = sub_103A66810();
32  v25 = 0xE00000001LL;
33  v26 = 1;
34  v27 = getpid();
35  sub_10470FBA4(&v23);
36  v3 = pid_t v27; // [xsp+2B4h] [xbp-4Ch]
37  if ( (0xA304 & v3) == 0 )
38  {
39      v4 = (double)v24 / 1000000.0 + (double)(__int64)v23 - kCFAbsoluteTimeIntervalSince1970;
40      if ( *(double *)&qword_105DE3940 > v4 )
41      {
42          *(double *)&qword_105DE3938 = (double)v24 / 1000000.0 + (double)(__int64)v23 - kCFAbsoluteTimeIntervalSince1970;
43          byte_105DE3960 = 1;
44          *(double *)&qword_105DE3948 = *(double *)&qword_105DE3940 - v4;
45      }

```

```
iPhone7-1331:~ root# ps -A | grep WhatsApp
41732 ??          0:00.00 /var/containers/Bundle/Application/2BE964D4-8DF0-4858-A06D-66CA8741ACDC/WhatsApp.app/WhatsApp
57080 ttys000      0:00.01 grep WhatsApp
```

->

- 41732 = 0xA304

是对得上的。

不过没有能查看到 局部变量的窗口

感觉只能是：鼠标移动到某个变量，查看单个变量的实时的值。且只能看到：16进制=二进制的值

目前就这样。

crifan.org, 使用署名4.0国际(CC BY 4.0)协议发布 all right reserved, powered by Gitbook最后更新: 2024-10-01 16:58:57

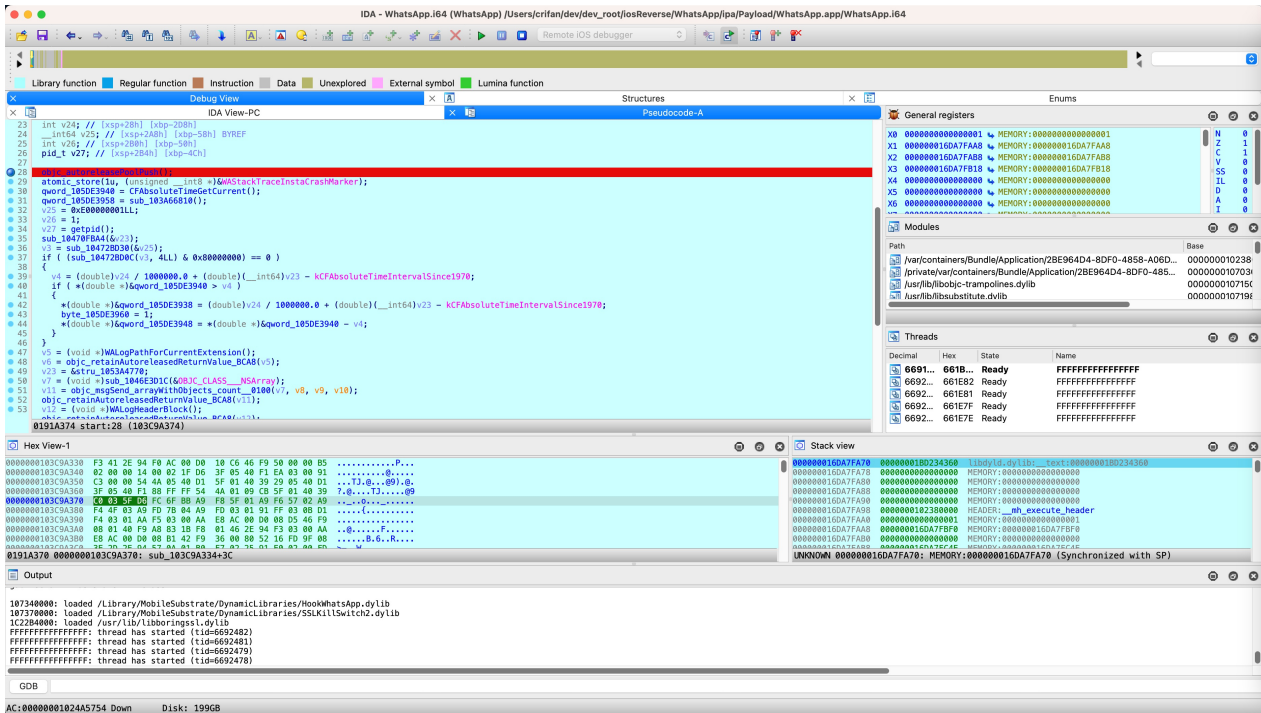
调试界面

之前给：IDA的汇编和F5伪代码，都加了断点：

断点的函数名是：入口函数start

此处进来后，默认进入：

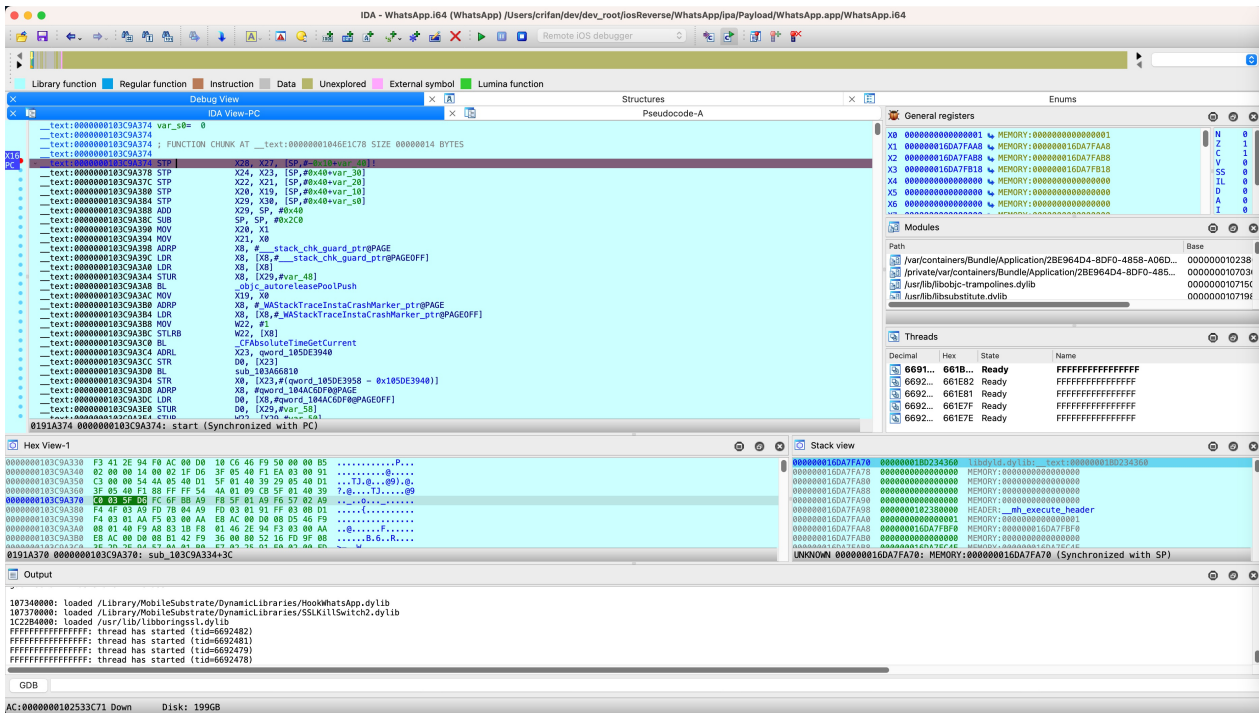
伪代码中的断点：



IDA View-PC

点击 IDA View-PC，切换到：

IDA汇编，可以看到之前给汇编加的断点：



相关log

IDA左下角的最后一部分的log是：

```
insider the segment '__objc_const' as read only
because the segment has the write permission
The decompiler did not consider the segment '__objc_data' as read only
because the segment has the write permission
102380000 process /var/containers/Bundle/Application/2BE964D4-8DF0-4858-A06D-66CA8741ACDC/WhatsApp.app/WhatsApp has
started pid 4294967294
shared cache mapping: 1BCFA0000, 1F8D24000
shared cache mapping: 1FAD24000, 2067E8000
shared cache mapping: 2087E8000, 2112E8000
10703C000 loaded /private/var/containers/Bundle/Application/2BE964D4-8DF0-4858-A06D-66CA8741ACDC/WhatsApp.app/Framework
orks/libswift_Concurrency.dylib
1074BC000 loaded /usr/lib/substitute-inserter.dylib
10792C000 loaded /private/var/containers/Bundle/Application/2BE964D4-8DF0-4858-A06D-66CA8741ACDC/WhatsApp.app/Framework
orks/SharedModules.framework/SharedModules
1BCFDC000 loaded /usr/lib/system/libsystem_trace.dylib
1BCFF3000 loaded /usr/lib/system/libxpc.dylib
1BD025000 loaded /usr/lib/system/libsystem_blocks.dylib
...
1F4CEC000 loaded /System/Library/PrivateFrameworks/AppSSO.framework/AppSSO
1F4D1F000 loaded /System/Library/PrivateFrameworks/CPAnalytics.framework/CPAnalytics
1F5483000 loaded /usr/lib/swift/libswiftCoreML.dylib
1F549F000 loaded /usr/lib/swift/libswiftSpeech.dylib
1074BC000 loaded /usr/lib/substitute-inserter.dylib
gdb objc[41732] Class WADocumentAttachmentStats is implemented in both /private/var/containers/Bundle/Application/2B
E964D4-8DF0-4858-A06D-66CA8741ACDC/WhatsApp.app/Frameworks/SharedModules.framework/SharedModules (0x10928d0c8) and /p
rivate/var/containers/Bund
gdb le Application 2BE964D4-8DF0-4858-A06D-66CA8741ACDC WhatsApp app WhatsApp (0x105979e10). One of the two will be u
sed. Which one is undefined.
objc[41732] Class WChatSearchBarCatalyst is implemented in both /private/var/containers/Bundle/Application/2BE9
gdb 64D4 8DF0 4858 A06D 66CA8741ACDC WhatsApp app Frameworks SharedModules.framework SharedModules (0x1092b2720) and
/private/var/containers/Bundle/Application/2BE964D4-8DF0-4858-A06D-66CA8741ACDC/WhatsApp.app/WhatsApp (0x105979fe0).
One of the two will be use
gdb d. Which one is undefined.
10715C000 loaded /usr/lib/libobjc-trampolines.dylib
107198000 loaded /usr/lib/libsubstitute.dylib
10B800000 loaded /usr/lib/substitute-loader.dylib
1071C0000 loaded /usr/lib/libsubstrate.dylib
1071E4000 loaded /Library/MobileSubstrate/DynamicLibraries/ Choicy.dylib
1071F4000 loaded /Library/MobileSubstrate/DynamicLibraries/AppSyncUnified-FrontBoard.dylib
gdb SubstituteLog Attempted to hook non existant selector "trustStateForApplication:" in class "nil"
```

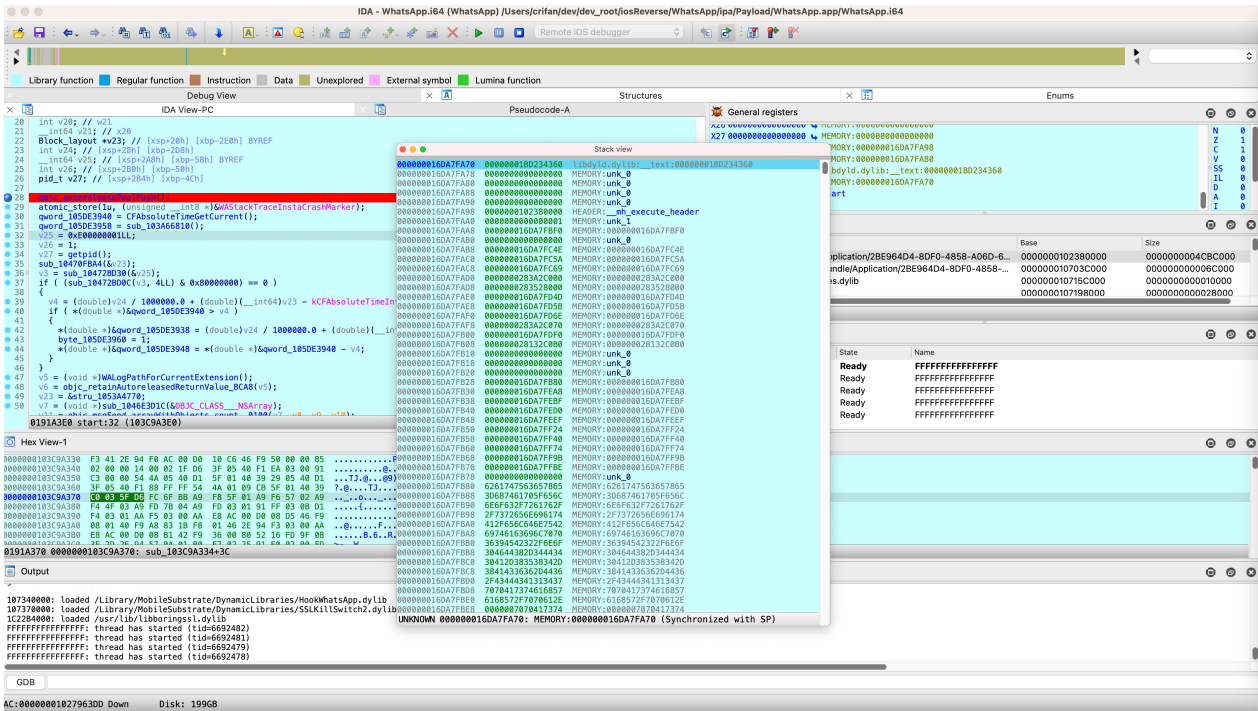
```
SubstituteLog: Attempted to hook non existant selector "trustStateWithTrustRequiredReasons:" in class "nil"
SubstituteLog: Attempted to hook non existant se
gdb lector "trustState" in class "nil"

107340000 loaded /Library/MobileSubstrate/DynamicLibraries/HookWhatsApp.dylib
107370000 loaded /Library/MobileSubstrate/DynamicLibraries/SSLLKillSwitch2.dylib
1C22B4000 loaded /usr/lib/libboringsssl.dylib
FFFFFFFFFFFFFFF: thread has started tid 0692482
FFFFFFFFFFFFFFF: thread has started tid 0692481
FFFFFFFFFFFFFFF: thread has started tid 0692479
FFFFFFFFFFFFFFF: thread has started tid 0692478
```

发现并没有太多的界面，可供查看？

Stack View

点击部分窗口，变成浮动窗口，看看效果：



```
000000016DA7FA70 00000001BD234360 libdyld.dylib:__text:00000001BD234360
000000016DA7FA78 0000000000000000 MEMORY:unk_0
000000016DA7FA80 0000000000000000 MEMORY:unk_0
000000016DA7FA88 0000000000000000 MEMORY:unk_0
000000016DA7FA90 0000000000000000 MEMORY:unk_0
000000016DA7FA98 0000000102380000 HEADER:__mh_execute_header
000000016DA7FAA0 0000000000000001 MEMORY:unk_1
000000016DA7FAA8 000000016DA7FBF0 MEMORY:000000016DA7FBF0
000000016DA7FAB0 0000000000000000 MEMORY:unk_0
000000016DA7FAB8 000000016DA7FC4E MEMORY:000000016DA7FC4E
000000016DA7FAC0 000000016DA7FC5A MEMORY:000000016DA7FC5A
000000016DA7FAC8 000000016DA7FC69 MEMORY:000000016DA7FC69
000000016DA7FAD0 0000000283A2C000 MEMORY:0000000283A2C000
000000016DA7FAD8 0000000283528000 MEMORY:0000000283528000
000000016DA7FAE0 000000016DA7FD4D MEMORY:000000016DA7FD4D
000000016DA7FAE8 000000016DA7FD5B MEMORY:000000016DA7FD5B
000000016DA7FAF0 000000016DA7FD6E MEMORY:000000016DA7FD6E
000000016DA7FAF8 0000000283A2C070 MEMORY:0000000283A2C070
000000016DA7FB00 000000016DA7FDF0 MEMORY:000000016DA7FDF0
000000016DA7FB08 000000028132C0B0 MEMORY:000000028132C0B0
000000016DA7FB10 0000000000000000 MEMORY:unk_0
000000016DA7FB18 0000000000000000 MEMORY:unk_0
000000016DA7FB20 0000000000000000 MEMORY:unk_0
000000016DA7FB28 000000016DA7FB80 MEMORY:000000016DA7FB80
000000016DA7FB30 000000016DA7FEAB MEMORY:000000016DA7FEAB
000000016DA7FB38 000000016DA7FEBF MEMORY:000000016DA7FEBF
```



```
000000016DA7FB40 000000016DA7FED0 MEMORY:000000016DA7FED0
000000016DA7FB48 000000016DA7FEEF MEMORY:000000016DA7FEEF
000000016DA7FB50 000000016DA7FF24 MEMORY:000000016DA7FF24
000000016DA7FB58 000000016DA7FF40 MEMORY:000000016DA7FF40
000000016DA7FB60 000000016DA7FF74 MEMORY:000000016DA7FF74
000000016DA7FB68 000000016DA7FF9B MEMORY:000000016DA7FF9B
000000016DA7FB70 000000016DA7FFBE MEMORY:000000016DA7FFBE
000000016DA7FB78 0000000000000000 MEMORY:unk_0
000000016DA7FB80 6261747563657865 MEMORY:6261747563657865
000000016DA7FB88 3D687461705F656C MEMORY:3D687461705F656C
000000016DA7FB90 6E6F632F7261762F MEMORY:6E6F632F7261762F
000000016DA7FB98 2F7372656E696174 MEMORY:2F7372656E696174
000000016DA7FBA0 412F656C646E7542 MEMORY:412F656C646E7542
000000016DA7FBA8 69746163696C7070 MEMORY:69746163696C7070
000000016DA7FBB0 36394542322F6E6F MEMORY:36394542322F6E6F
000000016DA7FBB8 304644382D344434 MEMORY:304644382D344434
000000016DA7FBC0 30412D383538342D MEMORY:30412D383538342D
000000016DA7FBC8 38414336362D4436 MEMORY:38414336362D4436
```

结果操作一下：都是？了：

```
Stack view
0000000000000000  ??????????????????
0000000000000008  ??????????????????
0000000000000010  ??????????????????
0000000000000018  ??????????????????
0000000000000020  ??????????????????
0000000000000028  ??????????????????
0000000000000030  ??????????????????
0000000000000038  ??????????????????
0000000000000040  ??????????????????
0000000000000048  ??????????????????
0000000000000050  ??????????????????
0000000000000058  ??????????????????
0000000000000060  ??????????????????
0000000000000068  ??????????????????
0000000000000070  ??????????????????
0000000000000078  ??????????????????
0000000000000080  ??????????????????
0000000000000088  ??????????????????
0000000000000090  ??????????????????
0000000000000098  ??????????????????
00000000000000A0  ??????????????????
00000000000000A8  ??????????????????
00000000000000B0  ??????????????????
00000000000000B8  ??????????????????
00000000000000C0  ??????????????????
00000000000000C8  ??????????????????
00000000000000D0  ??????????????????
00000000000000D8  ??????????????????
00000000000000E0  ??????????????????
00000000000000E8  ??????????????????
00000000000000F0  ??????????????????
00000000000000F8  ??????????????????
0000000000000100  ??????????????????
0000000000000108  ??????????????????
0000000000000110  ??????????????????
0000000000000118  ??????????????????
0000000000000120  ??????????????????
0000000000000128  ??????????????????
0000000000000130  ??????????????????
0000000000000138  ??????????????????
0000000000000140  ??????????????????
0000000000000148  ??????????????????
0000000000000150  ??????????????????
0000000000000158  ??????????????????
0000000000000160  ??????????????????
0000000000000168  ??????????????????
0000000000000170  ??????????????????
0000000000000178  ??????????????????
UNKNOWN 0000000000000000: MEMORY:unk_0 (Synchronized with SP)
```

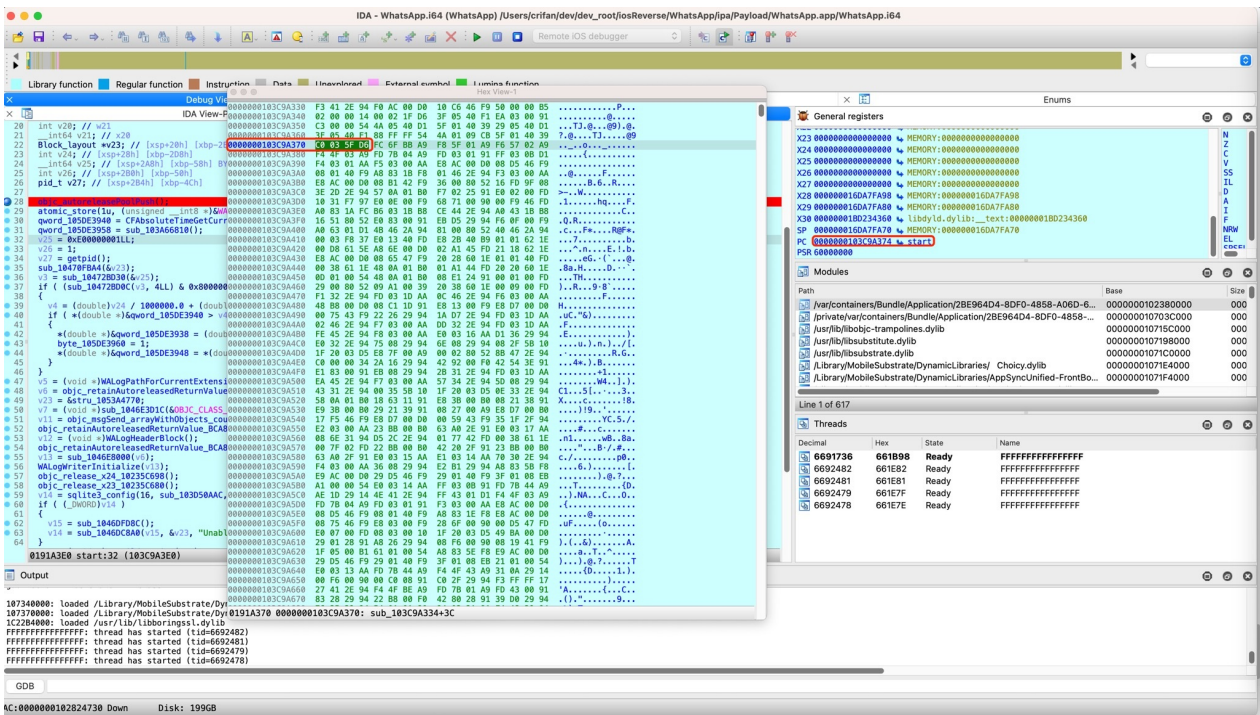
Hex View

去看看：

- Hex View-1



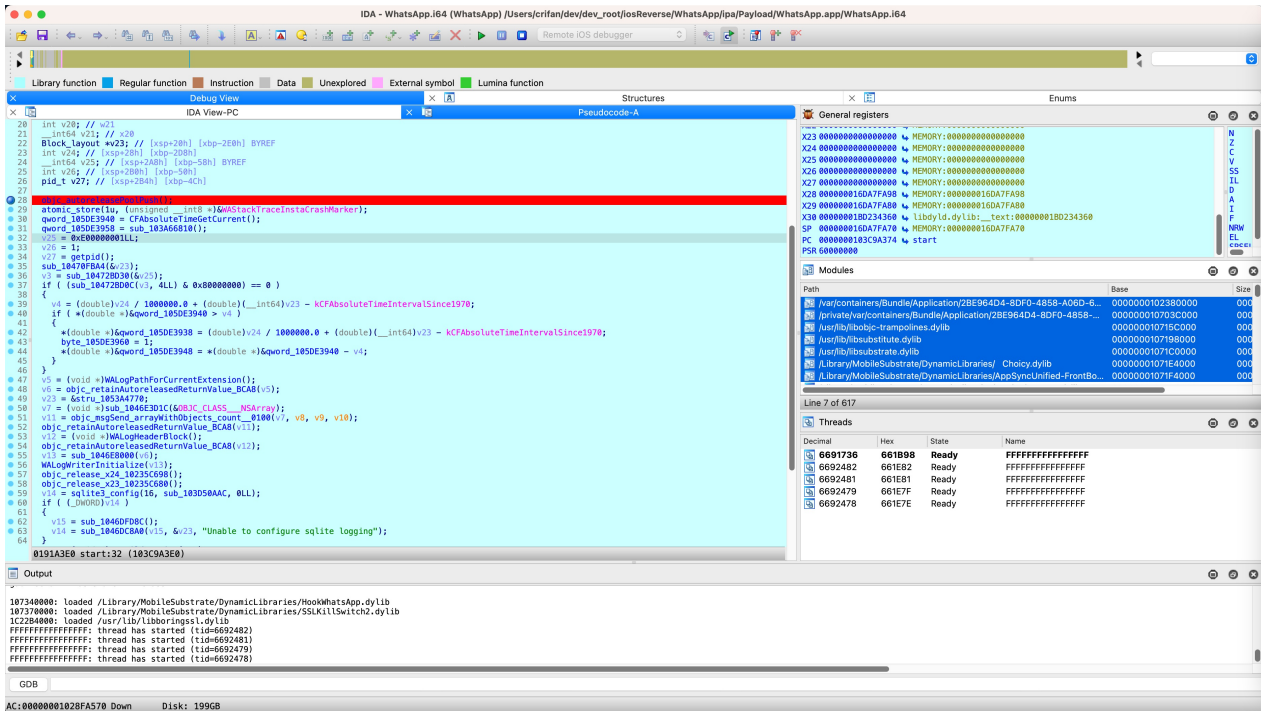
后来发现了：



但是貌似Hex模式也没太多用。关闭。

Modules

去选择Modules：



且可以复制出内容：

```
/var/containers/Bundle/Application/2BE964D4-8DF0-4858-A06D-66CA8741ACDC/WhatsApp.app/WhatsApp 0000000102380000
0000000004CBC000
/private/var/containers/Bundle/Application/2BE964D4-8DF0-4858-A06D-66CA8741ACDC/WhatsApp.app/Frameworks/libswift_Conc
urrency.dylib 000000010703C000 000000000000C000
/usr/lib/libobjc-trampolines.dylib 000000010715C000 0000000000010000
/usr/lib/libsubsystem.dylib 0000000107198000 00000000000028000
/usr/lib/libsubstrate.dylib 00000001071C0000 00000000000014000
/Library/MobileSubstrate/DynamicLibraries/ Choicy.dylib 00000001071E4000 0000000000010000
/Library/MobileSubstrate/DynamicLibraries/AppSyncUnified-FrontBoard.dylib 00000001071F4000 000000000000C000
. . .
```

拖动已变成浮动窗口：

目前的IDA中调试iOS的app的效果是：

IDA - WhatsApp.i64 (WhatsApp) Users/crifan/devroot/ios/Reverse/WhatsApp.app/WhatsApp.App.dylib/WhatsApp.App.dylib

Library function Regular function Instruction Data Unexplored External symbol Lumina function

Debug View IDA View-PC Structures Pseudocode-A General registers Enums

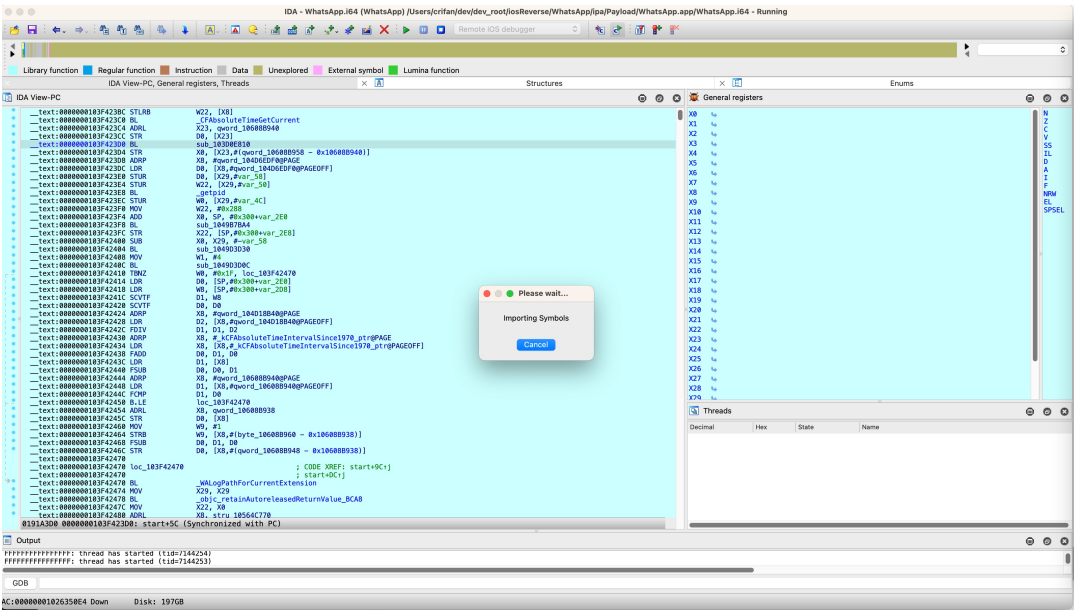
```
20 int v20; // w21
21 int64_t v21; // x20
22 block_layout v23; // [xsp+20h] [xsp-20h] BYREF
23 int v24; // [xsp+28h] [xsp-28h] BYREF
24 int64_t v25; // [xsp+2A0h] [xsp-2A0h] BYREF
25 int v26; // [xsp+2B0h] [xsp-2B0h]
26 pid_t v27; // [xsp+2B4h] [xsp-4Ch]
27
28 // 0191A3E0 start:32 (102C9A3E0)
29 atomic_store1u, [__unsigged_ints + &stackTraceInCrasMarker];
30 word_102DE3948 = &stackTraceInCrasMarker;
31 word_102DE3948 = sub_10A66A10();
32 v20 = 0x00000001LL;
33 v26 = 1;
34 v27 = getpid();
35 sub_1047F804(v23);
36 v27 = sub_104728038(v23);
37 if ( (sub_1047280C(v23, 4LL) & 0x80000000) == 0 )
38 {
39     v24 = (double)v24 / 1000000.0 + (double)(int64_t)v23 - KCFAbsoluteTimeIntervalSince1970;
40     if ( (double)>Kword_102DE3948 = (double)v24 / 1000000.0 + (double)(int64_t)v23 - KCFAbsoluteTimeIntervalSince1970;
41         >Kword_102DE3948 = (double)v24 / 1000000.0 + (double)(int64_t)v23 - KCFAbsoluteTimeIntervalSince1970;
42         >Kword_102DE3948 = (double)v24 / 1000000.0 + (double)(int64_t)v23 - KCFAbsoluteTimeIntervalSince1970;
43         >Kword_102DE3948 = (double)v24 / 1000000.0 + (double)(int64_t)v23 - KCFAbsoluteTimeIntervalSince1970;
44         >Kword_102DE3948 = (double)v24 / 1000000.0 + (double)(int64_t)v23 - KCFAbsoluteTimeIntervalSince1970;
45         >Kword_102DE3948 = (double)v24 / 1000000.0 + (double)(int64_t)v23 - KCFAbsoluteTimeIntervalSince1970;
46         >Kword_102DE3948 = (double)v24 / 1000000.0 + (double)(int64_t)v23 - KCFAbsoluteTimeIntervalSince1970;
47         >Kword_102DE3948 = (double)v24 / 1000000.0 + (double)(int64_t)v23 - KCFAbsoluteTimeIntervalSince1970;
48         >Kword_102DE3948 = (double)v24 / 1000000.0 + (double)(int64_t)v23 - KCFAbsoluteTimeIntervalSince1970;
49         >Kword_102DE3948 = (double)v24 / 1000000.0 + (double)(int64_t)v23 - KCFAbsoluteTimeIntervalSince1970;
50         >Kword_102DE3948 = (double)v24 / 1000000.0 + (double)(int64_t)v23 - KCFAbsoluteTimeIntervalSince1970;
51         >Kword_102DE3948 = (double)v24 / 1000000.0 + (double)(int64_t)v23 - KCFAbsoluteTimeIntervalSince1970;
52         >Kword_102DE3948 = (double)v24 / 1000000.0 + (double)(int64_t)v23 - KCFAbsoluteTimeIntervalSince1970;
53         >Kword_102DE3948 = (double)v24 / 1000000.0 + (double)(int64_t)v23 - KCFAbsoluteTimeIntervalSince1970;
54         >Kword_102DE3948 = (double)v24 / 1000000.0 + (double)(int64_t)v23 - KCFAbsoluteTimeIntervalSince1970;
55         >Kword_102DE3948 = (double)v24 / 1000000.0 + (double)(int64_t)v23 - KCFAbsoluteTimeIntervalSince1970;
56         >Kword_102DE3948 = (double)v24 / 1000000.0 + (double)(int64_t)v23 - KCFAbsoluteTimeIntervalSince1970;
57         >Kword_102DE3948 = (double)v24 / 1000000.0 + (double)(int64_t)v23 - KCFAbsoluteTimeIntervalSince1970;
58         >Kword_102DE3948 = (double)v24 / 1000000.0 + (double)(int64_t)v23 - KCFAbsoluteTimeIntervalSince1970;
59         >Kword_102DE3948 = (double)v24 / 1000000.0 + (double)(int64_t)v23 - KCFAbsoluteTimeIntervalSince1970;
60         >Kword_102DE3948 = (double)v24 / 1000000.0 + (double)(int64_t)v23 - KCFAbsoluteTimeIntervalSince1970;
61         >Kword_102DE3948 = (double)v24 / 1000000.0 + (double)(int64_t)v23 - KCFAbsoluteTimeIntervalSince1970;
62         >Kword_102DE3948 = (double)v24 / 1000000.0 + (double)(int64_t)v23 - KCFAbsoluteTimeIntervalSince1970;
63         >Kword_102DE3948 = (double)v24 / 1000000.0 + (double)(int64_t)v23 - KCFAbsoluteTimeIntervalSince1970;
64         >Kword_102DE3948 = (double)v24 / 1000000.0 + (double)(int64_t)v23 - KCFAbsoluteTimeIntervalSince1970;
65         >Kword_102DE3948 = (double)v24 / 1000000.0 + (double)(int64_t)v23 - KCFAbsoluteTimeIntervalSince1970;
66         >Kword_102DE3948 = (double)v24 / 1000000.0 + (double)(int64_t)v23 - KCFAbsoluteTimeIntervalSince1970;
67         >Kword_102DE3948 = (double)v24 / 1000000.0 + (double)(int64_t)v23 - KCFAbsoluteTimeIntervalSince1970;
68         >Kword_102DE3948 = (double)v24 / 1000000.0 + (double)(int64_t)v23 - KCFAbsoluteTimeIntervalSince1970;
69         >Kword_102DE3948 = (double)v24 / 1000000.0 + (double)(int64_t)v23 - KCFAbsoluteTimeIntervalSince1970;
70         >Kword_102DE3948 = (double)v24 / 1000000.0 + (double)(int64_t)v23 - KCFAbsoluteTimeIntervalSince1970;
71         >Kword_102DE3948 = (double)v24 / 1000000.0 + (double)(int64_t)v23 - KCFAbsoluteTimeIntervalSince1970;
72         >Kword_102DE3948 = (double)v24 / 1000000.0 + (double)(int64_t)v23 - KCFAbsoluteTimeIntervalSince1970;
73         >Kword_102DE3948 = (double)v24 / 1000000.0 + (double)(int64_t)v23 - KCFAbsoluteTimeIntervalSince1970;
74         >Kword_102DE3948 = (double)v24 / 1000000.0 + (double)(int64_t)v23 - KCFAbsoluteTimeIntervalSince1970;
75         >Kword_102DE3948 = (double)v24 / 1000000.0 + (double)(int64_t)v23 - KCFAbsoluteTimeIntervalSince1970;
76         >Kword_102DE3948 = (double)v24 / 1000000.0 + (double)(int64_t)v23 - KCFAbsoluteTimeIntervalSince1970;
77         >Kword_102DE3948 = (double)v24 / 1000000.0 + (double)(int64_t)v23 - KCFAbsoluteTimeIntervalSince1970;
78         >Kword_102DE3948 = (double)v24 / 1000000.0 + (double)(int64_t)v23 - KCFAbsoluteTimeIntervalSince1970;
79         >Kword_102DE3948 = (double)v24 / 1000000.0 + (double)(int64_t)v23 - KCFAbsoluteTimeIntervalSince1970;
80         >Kword_102DE3948 = (double)v24 / 1000000.0 + (double)(int64_t)v23 - KCFAbsoluteTimeIntervalSince1970;
81         >Kword_102DE3948 = (double)v24 / 1000000.0 + (double)(int64_t)v23 - KCFAbsoluteTimeIntervalSince1970;
82         >Kword_102DE3948 = (double)v24 / 1000000.0 + (double)(int64_t)v23 - KCFAbsoluteTimeIntervalSince1970;
83         >Kword_102DE3948 = (double)v24 / 1000000.0 + (double)(int64_t)v23 - KCFAbsoluteTimeIntervalSince1970;
84         >Kword_102DE3948 = (double)v24 / 1000000.0 + (double)(int64_t)v23 - KCFAbsoluteTimeIntervalSince1970;
85         >Kword_102DE3948 = (double)v24 / 1000000.0 + (double)(int64_t)v23 - KCFAbsoluteTimeIntervalSince1970;
86         >Kword_102DE3948 = (double)v24 / 1000000.0 + (double)(int64_t)v23 - KCFAbsoluteTimeIntervalSince1970;
87         >Kword_102DE3948 = (double)v24 / 1000000.0 + (double)(int64_t)v23 - KCFAbsoluteTimeIntervalSince1970;
88         >Kword_102DE3948 = (double)v24 / 1000000.0 + (double)(int64_t)v23 - KCFAbsoluteTimeIntervalSince1970;
89         >Kword_102DE3948 = (double)v24 / 1000000.0 + (double)(int64_t)v23 - KCFAbsoluteTimeIntervalSince1970;
90         >Kword_102DE3948 = (double)v24 / 1000000.0 + (double)(int64_t)v23 - KCFAbsoluteTimeIntervalSince1970;
91         >Kword_102DE3948 = (double)v24 / 1000000.0 + (double)(int64_t)v23 - KCFAbsoluteTimeIntervalSince1970;
92         >Kword_102DE3948 = (double)v24 / 1000000.0 + (double)(int64_t)v23 - KCFAbsoluteTimeIntervalSince1970;
93         >Kword_102DE3948 = (double)v24 / 1000000.0 + (double)(int64_t)v23 - KCFAbsoluteTimeIntervalSince1970;
94         >Kword_102DE3948 = (double)v24 / 1000000.0 + (double)(int64_t)v23 - KCFAbsoluteTimeIntervalSince1970;
95         >Kword_102DE3948 = (double)v24 / 1000000.0 + (double)(int64_t)v23 - KCFAbsoluteTimeIntervalSince1970;
96         >Kword_102DE3948 = (double)v24 / 1000000.0 + (double)(int64_t)v23 - KCFAbsoluteTime
```

常见问题

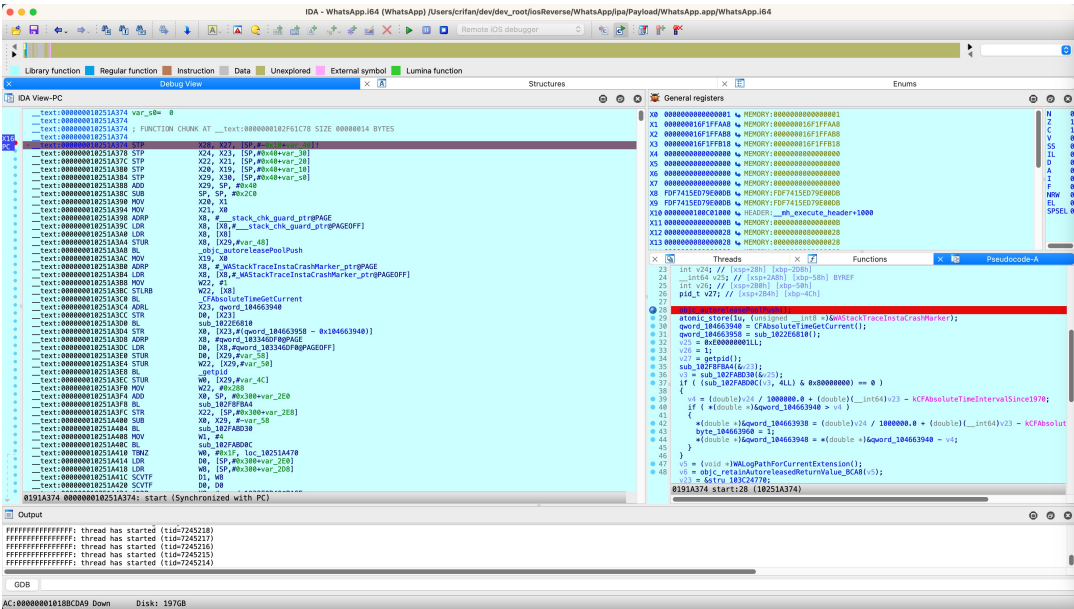
IDA调试iOS

每次调试启动太耗时

- IDA调试iOS的app：WhatsApp
 - 每次调试，都会
 - Importing Symbols导入符号：耗时10分钟左右



- 自动分析：耗时很久



-> 导致每次调试都要等待很长时间：平均要半小时以上

-> 导致IDA调试iOS的app：效率太低，不方便，几乎不可用

无法类似lldb的po打印出对象信息

此处IDA调试iOS的app：WhatsApp

虽然可以鼠标移动到变量上，能查看到16进制=二进制的值

但是希望实现：

Xcode中lldb的po查看到的类的值

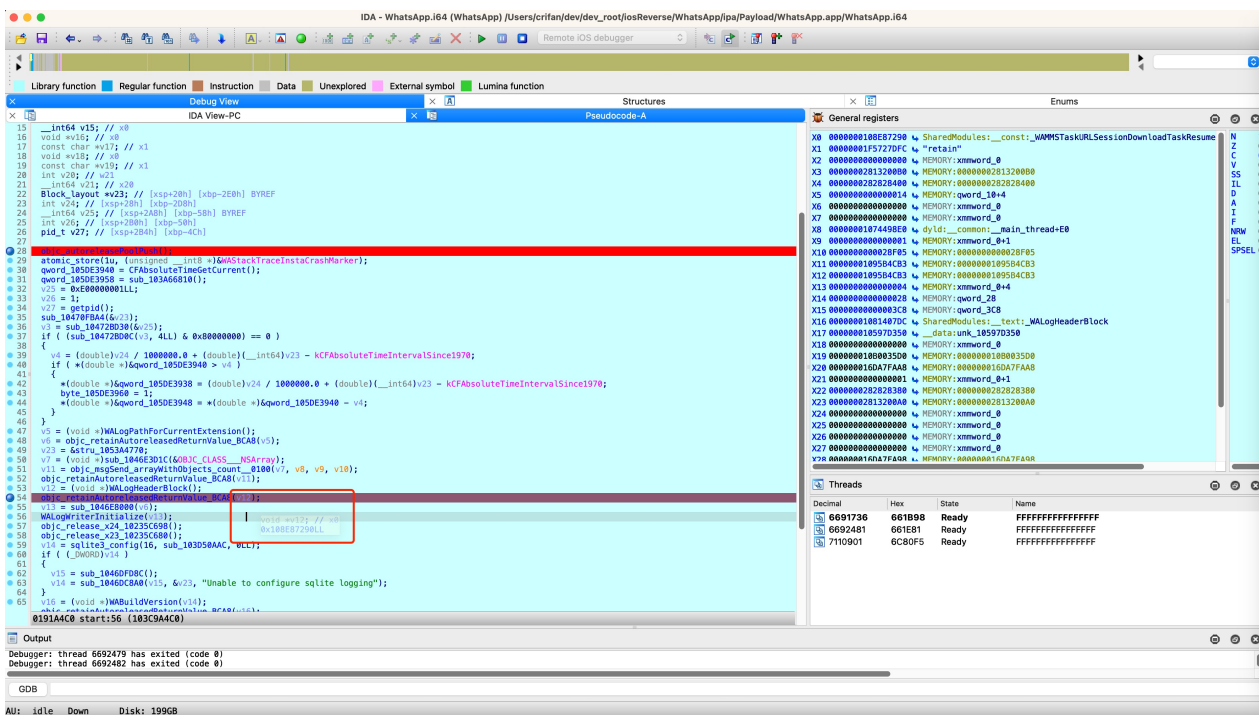
比如：

继续给下面某行：

```
v12 = (void *)WALogHeaderBlock();
objc_retainAutoreleasedReturnValue_BCA8(v12);
```

加了断点。

继续运行后，此时：



按道理：

v12是个Block类型

希望能像Xcode中lldb，去po查看出是Block

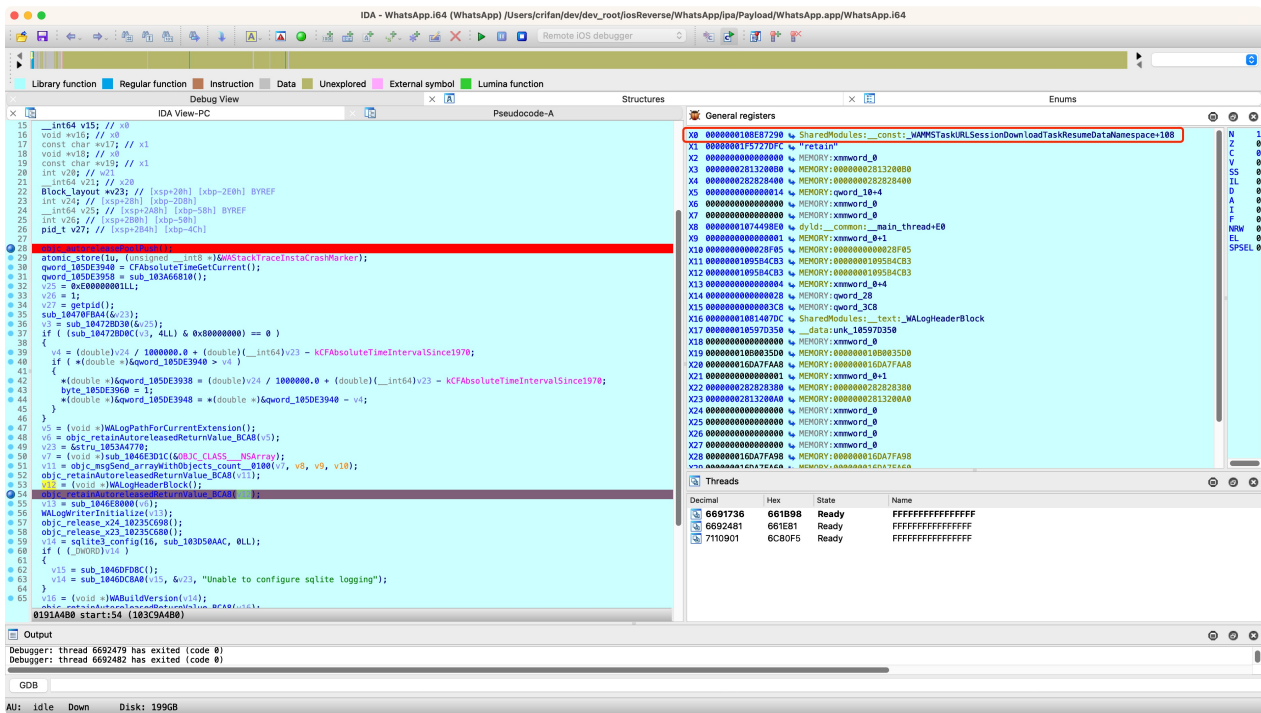
但是此时只能看到16进制的值

且还不方便复制，只能手动写出来：

0x108E87290

不过此处巧了，是x0寄存器的值

所以，寄存器窗口，倒是可以看到对应的值：



且：好像可以解析出是SharedModules中const常量

但是显示的内容，也不是Block

然后经过一番尝试，最终还是：

IDA调试iOS的app中，无法像（Xcode中）lldb去 po 查看出类=对象的信息

TODO：

有机会，可以去参考：

[simulator the lldb command like "po objc_object" inside the IDA debugger. \(github.com\)](https://github.com)

去借鉴别人的脚本，自己实现： po xxx 的效果

子教程

IDA相关子教程：

- [逆向利器：IDA](#)
 - Python插件
 - [IDA插件脚本：IDAPython](#)

crifan.org，使用[署名4.0国际\(CC BY 4.0\)协议](#)发布 all right reserved，powered by Gitbook最后更新：2024-10-01 18:01:49

附录

下面列出相关参考资料。

crifan.org, 使用[署名4.0国际\(CC BY 4.0\)协议](#)发布 all right reserved, powered by Gitbook最后更新: 2022-03-17 20:39:28

文档和资料

此处整理IDA相关的文档、教程、书籍等有价值的参考资料。

IDA官网

IDA的官网：

<https://www.hex-rays.com>

本身就有很多不错的教程和资料。

IDA官网在线文档

- 文档总入口：<https://hex-rays.com/documentation/>
 - Help总入口：<https://hex-rays.com/products/ida/support/idadoc/index.shtml>
 - 反编译器：<https://hex-rays.com/products/decompiler/manual/index.shtml>
 - 其中最常用的一些：
 - 伪代码中右键菜单中的各种功能
 - [Interactive operation](#)
 - 常见问题和错误
 - [Failures and troubleshooting](#)

IDA的tip of week

[tip of week index \(hex-rays.com\)](#)

https://hex-rays.com/wp-content/uploads/2021/10/igor-tip-of-the-week-S01.pdf

页面视图 | 朗读此页内容

Igor's tip of the week

season one

from 07/08/2020 to 13/08/2021



Today, Hex-Rays is excited to launch a special blog series where Igor, one of the experts behind IDA, will provide useful tips and functionalities of IDA that are not always known or less obvious to its users. The first episode of this blog series covers the most useful keyboard shortcuts that will certainly speed up your IDA experience. So, we hope you enjoy this first post and tune in every Friday to read Igor's tip of the week!

Posted the 7th August 2020 by Igor Skochinsky

Usage: basic and advanced usage of IDA features

- #05: Lesser-known keyboard shortcuts in IDA
- #03: Selection in IDA
- #04: More selection!
- #05: Highlight
- #09: Reanalysis
- #13: String literals and custom encodings
- #14: Comments in IDA
- #15: Comments in structures and enums
- #28: Functions list
- #30: Quick views
- #31: Hiding and Collapsing
- #34: Dummy names
- #35: Demangled names
- #36: Working with list views in IDA
- #37: Patching
- #46: Disassembly operand representation
- #47: Hints in IDA

Navigation: moving around the database

- #16: Cross-references
- #17: Cross-references 2
- #20: Going places
- #23: Graph view
- #38: Hex view
- #48: Searching in IDA
- #49: Navigation band
- #50: Execution flow arrows

Types: working with types

- #10: Working with arrays
- #11: Portable operations and constants

Hidden: hidden gems, not widely known but useful functionality

- #06: IDA Release notes
- #19: Function calls
- #21: Calculator and expression evaluation feature in IDA
- #24: Renaming registers
- #39: Export Data
- #41: Binary file loader
- #44: Hex dump loader

Decompiler: related to the Hex-Rays decompiler

- #18: Decompiler and global cross-references
- #27: Fixing the stack pointer
- #40: Decompiler basics
- #42: Renaming and retyping in the decompiler
- #43: Annotating the decompiler output
- #45: Decompiler types

Automation: automating repetitive tasks

- #07: IDA command-line options cheatsheet
- #08: Batch mode under the hood
- #32: Running scripts

Customization: customizing IDA UI to better suit your workflow

- #02: IDA UI actions and where to find them
- #22: IDA desktop layouts
- #25: Disassembly options
- #26: Disassembly options 2
- #29: Color up your IDA
- #33: IDA's user directory (IDAUSR)

有很多有用提示

有空可以看看

IDA官网的下载中心

IDA官网的下载中心:

[Download center \(hex-rays.com\)](https://hex-rays.com)

有很多相关内容, 可供学习和使用:

- SDK and Utilities
 - Some downloads are only available to IDA Pro users and require a password which can be found in the latest download email.
 - IDA SDK 7.7
 - Develop processor modules, loaders and extensions - extended with the source of 30+ modules and 20+ loaders.
 - Please check out the SDK documentation online (or download the zip file for offline use).
 - Flair 7.7
 - Add your own compiler libraries to the FLIRT engine
 - FLIRT = Fast Library Identification and Recognition Technology
 - IDAClang 7.7
 - A type library generator based on libclang. Use this when parsing complex C++ code that tilib cannot handle.
 - Tilib 7.7

- Create your own type libraries
- Loadint 7.7
 - Create your own disassembler comment databases
- idsutils 7.7
 - Create your own IDS files from DLLs
- ios_deploy
 - iOS helper utility to manipulate iOS devices
- PIN tool
 - The source code of our PIN tool. It creates a debugger backend out of Intel's PIN framework
 - See: PIN framework
- TVision 2015 library
 - For the IDA text interface (source code)
- Qwingraph v1.10
 - Source code the Wingraph we use and modified (GPL)
- Sample plugins
 - Stealth
 - Stealth against anti-debugging tricks
 - findcrypt
 - Identifies some frequently used block ciphers
 - highlighter
 - Highlights code that has been single stepped through in a debugging session
 - unispector
 - Extracts unicode strings from an IDA database
 - IDA Pro, Python and Qt
 - Migrating PySide code to PyQt5
 - Using custom viewers from IDAPython
 - Augmenting IDA UI with your own actions
 - Plugin contest pages
 - Our plugins contest pages offer many useful plugins!
 - By year: 2009, 2010, 2011, 2012, 2013, 2014, 2015, 2016, 2017, 2018, 2019
- User contributions
 - Plugins
 - COM Interface Plugin
 - By Dieter Spaar
 - Sobek
 - A simple data-flow analysis plugin by JF Michel
 - PDBPlus
 - By Dean Ashton
 - IDB_2_PAT
 - By J.C Roberts
 - strucrec
 - By Halvar Flake
 - Class Informer plugin
 - To reconstruct C++ classes using the RTTI info
 - By Sirmabus
 - IDC scripts
 - Visual Basic Disassembly IDC script
 - To assist in the disassembly of VB5/VB6 hostile code
 - By Reginald Wong
 - IDC Delphi script
 - Delphi constants and class definitions
 - By Dietrich Teickner
 - h2enum

- converts C/C++ header files to IDA enums - Nice if you have no TIL files
- By Leonid Lisovsky
- IDC PDR script
 - Useful in the analysis of PDR files (port drivers)
 - By Huang Yu
- Loader script
 - For VC++ and Borland C++
 - By Toshiyuki Tega
- Microchipss 16F84 PIC script
 - Defines SFR and bit names for Microchip's 16F84 PIC processor but can be used as a template for other processors
 - By an anonymous contributor
- dumpinfo
 - By JC Roberts
- Pentica-B script
 - Pentica-B import script for the H8-8300
 - By Tom Hayes
- H8 script
 - Improves the initial H8 autoanalysis
 - By Tom Hayes
- Processor modules
 - AMD 29K processor module
 - By Arne Wichmann
 - NEC V830 processor module
 - By Ben Byer
 - Samsung SAM8
 - By Andrew de Quincey. Also available is a plugin that generates files compatible with the SAMA assembler.
- Miscellaneous
 - symload
 - By Dainis Jonitis
 - PE utilities
 - A set of extremely useful PE utilities
 - By Atli Mar Gudmundsson
 - H8 utilities
 - A few utilities that could be useful to H8 developers
 - By Tom Hayes

抽空可以好好找找看看，有哪些值得好好利用的东西。

《The IDA Pro Book - The Unofficial Guide to the World's Most Popular Disassembler.pdf》

- [《The IDA Pro Book - The Unofficial Guide to the World's Most Popular Disassembler.pdf》](#)

-
- 评价：
 - 内容详实，值得抽空好好参考和学习
 - 感觉解释的很到位
 - 先介绍背景知识
 - 再介绍反编译
 - 再介绍工具：IDA

-> 已摘录的部分内容：

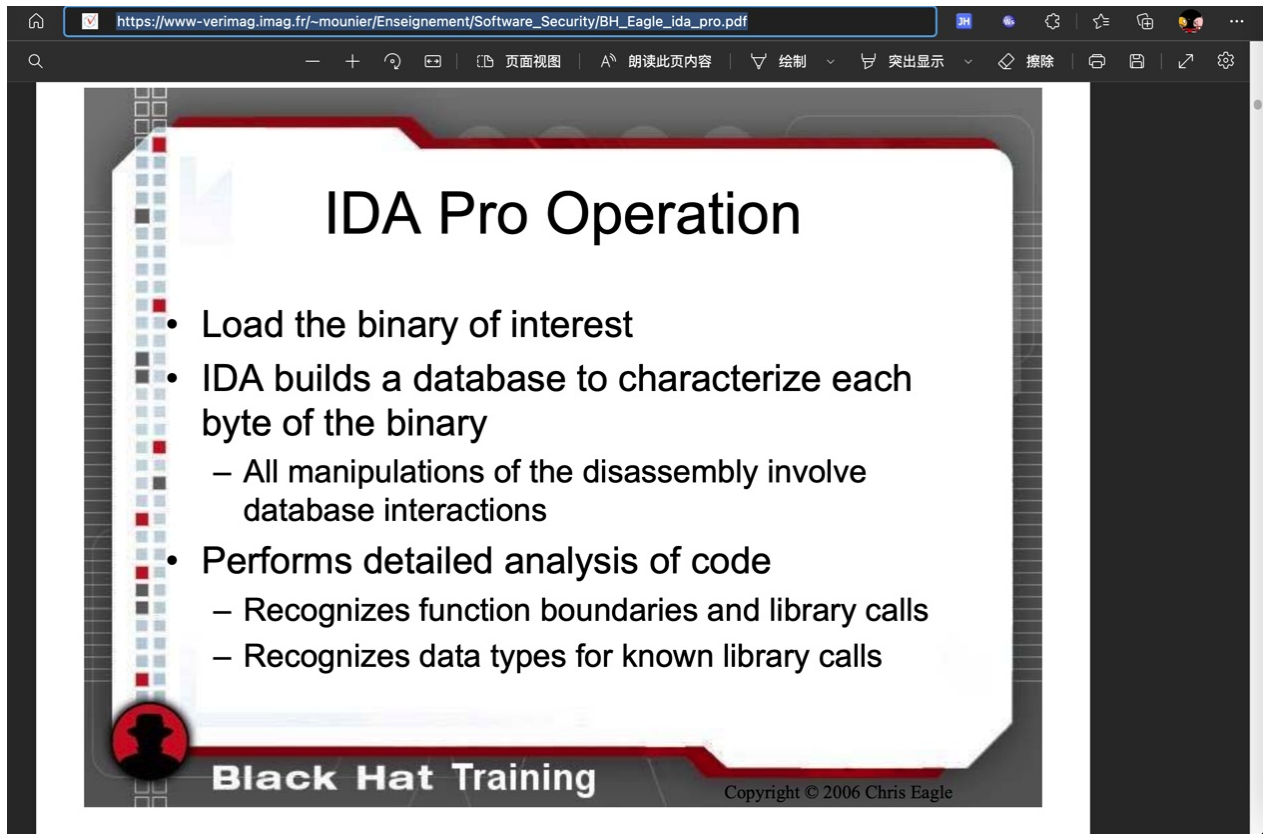
- 编程语言历史
 - First-generation languages
 - These are the lowest form of language, generally consisting of ones and zeros or some shorthand form such as hexadecimal, and readable only by binary ninjas. Things are confusing at this level because it is often difficult to distinguish data from instructions since everything looks pretty much the same. First-generation languages may also be referred to as machine languages, and in some cases byte code, while machine language programs are often referred to as binaries.
 - Second-generation languages
 - Also called assembly languages, second-generation languages are a mere table lookup away from machine language and generally map specific bit patterns, or operation codes (opcodes), to short but memorable character sequences called mnemonics. Occasionally these mnemonics actually help programmers remember the instructions with which they are associated.
 - An assembler is a tool used by programmers to translate their assembly language programs into machine language suitable for execution.
 - Third-generation languages
 - These languages take another step toward the expressive capability of natural languages by introducing keywords and constructs that programmers use as the building blocks for their programs. Third-generation languages are generally platform independent, though programs written using them may be platform dependent as a result of using features unique to a specific operating system. Often-cited examples include FORTRAN, COBOL, C, and Java. Programmers generally use compilers to translate their programs into assembly language or all the way to machine language (or some rough equivalent such as byte code).
 - Fourth-generation languages

- These exist but aren't relevant to this book and will not be discussed
- 反汇编器、反编译器 的由来
 - In a traditional software development model, compilers, assemblers, and linkers are used by themselves or in combination to create executable programs. In order to work our way backwards (or reverse engineer programs), we use tools to undo the assembly and compilation processes. Not surprisingly, such tools are called disassemblers and decompilers, and they do pretty much what their names imply. A disassembler undoes the assembly process, so we should expect assembly language as the output (and therefore machine language as input). Decompilers aim to produce output in a high-level language when given assembly or even machine language as input.
- 为何恢复源代码source code recovery 不太容易?
 - The compilation process is lossy.
 - At the machine language level there are no variable or function names, and variable type information can be determined only by how the data is used rather than explicit type declarations. When you observe 32 bits of data being transferred, you'll need to do some investigative work to determine whether those 32 bits represent an integer, a 32-bit floating point value, or a 32-bit pointer.
 - Compilation is a many-to-many operation.
 - This means that a source program can be translated to assembly language in many different ways, and machine language can be translated back to source in many different ways. As a result, it is quite common that compiling a file and immediately decompiling it may yield a vastly different source file from the one that was input.
 - Decompilers are very language and library dependent.
 - Processing a binary produced by a Delphi compiler with a decompiler designed to generate C code can yield very strange results. Similarly, feeding a compiled Windows binary through a decompiler that has no knowledge of the Windows programming API may not yield anything useful.
 - A nearly perfect disassembly capability is needed in order to accurately decompile a binary.
 - Any errors or omissions in the disassembly phase will almost certainly propagate through to the decompiled code.
- 反汇编的典型用途
 - The purpose of disassembly tools is often to facilitate understanding of programs when source code is unavailable. Common situations in which disassembly is used include these:
 - Analysis of malware
 - Analysis of closed-source software for vulnerabilities
 - Analysis of closed-source software for interoperability
 - Analysis of compiler-generated code to validate compiler performance/correctness
 - Display of program instructions while debugging

BH_Eagle_ida_pro.pdf

无意间看到的别人整理的IDA的内容：

[BH_Eagle_ida_pro.pdf](#)



下载到此处 [BH_Eagle_ida_pro.pdf](https://www.verimag.imag.fr/~mounier/Enseignement/Software_Security/BH_Eagle_ida_pro.pdf) 供下载和学习。

其他待学习和待研究的资料

- [IDA software reverse engineering - Code World \(codetd.com\)](#)
 - IDA的用法，内容很多，值得看看。
- [IDA Pro Tips to Add to Your Bag of Tricks – PT SWARM \(ptsecurity.com\)](#)
 - 这里有很多感觉有用的，值得参考的内容，抽空去研究学习
-

crifan.org, 使用署名4.0国际(CC BY 4.0)协议发布 all right reserved, powered by Gitbook最后更新: 2023-07-14 21:54:41

参考资料

- 【整理】iOS逆向心得：IDA中的unk的含义
- 【整理】IDA使用心得：常见名称及含义
- 【整理】IDA使用心得：多种显示模式
- 【已解决】IDA中浮动窗口Output Window如何固定到底部
- 【未解决】研究抖音越狱检测逻辑：_RxAnnotationInlineLoader的load
- 【未解决】IDA中用idat64的Batch Mode尝试反编译导出YouTube的Module_Framework全部代码伪代码
- 【整理】学习IDA教程：The IDA Pro Book
- 【整理】IDA中一些功能和选项设置
- 【整理】IDA中的自动分析Autoanalysis
- 【记录】用IDA分析加了符号表的抖音AwemeCore二进制
- 【未解决】搞懂IDA中_D_objc_selrefs qword_38AF870 % 8的含义
- 【记录】iOS的二进制中的__objc_selrefs的含义
- 【已解决】IDA中抖音AwemeCore中字符串const char* qword_3893908的原始字符串
- 【已解决】iOS逆向WhatsApp：用IDA去调试WhatsApp
- 【记录】重新启动IDA调试iOS调试环境
- 【记录】IDA调试iOS的WhatsApp：开始调试
- 【记录】IDA调试iOS的WhatsApp：相关调试界面
- 【未解决】IDA调试iOS程序如何实现类似Xcode中lldb的po打印出对象信息
- 【记录】IDA中符号表相关信息
- 【整理】IDA使用心得：OpCode区
- 【整理】IDA使用心得：auto comments
- 【已解决】IDA中查看ARM汇编的伪代码
- 【已解决】IDA使用心得：IDA汇编代码如何快速找到匹配的Xcode汇编代码
- 【未解决】IDA中开启宏指令比如ADRP和ADD变成ADRL
- 【记录】IDA中给libsscronet.so打补丁patch program修改二进制
-
- [Failures and troubleshooting \(hex-rays.com\)](https://hex-rays.com/failures-and-troubleshooting/)
- [Igor's tip of the week #49: Navigation band – Hex Rays \(hex-rays.com\)](https://hex-rays.com/igor-tip-of-the-week-49-navigation-band/)
- [tip of week index \(hex-rays.com\)](https://hex-rays.com/tip-of-week-index/)
- [Interactive operation \(hex-rays.com\)](https://hex-rays.com/interactive-operation/)
- [IDA_Pro_Shortcuts.pdf \(hex-rays.com\)](https://hex-rays.com/ida-pro-shortcuts.pdf)
- [Download center \(hex-rays.com\)](https://hex-rays.com/download-center/)
- [IDA Free](https://hex-rays.com/ida-free/)
- [IDA Evaluation](https://hex-rays.com/ida-evaluation/)
- [BH_Eagle_ida_pro.pdf](https://hex-rays.com/BH_Eagle_ida_pro.pdf)
- [IDA software reverse engineering - Code World \(codetd.com\)](https://codetd.com/ida-software-reverse-engineering/)
- [IDA Pro Tips to Add to Your Bag of Tricks – PT SWARM \(ptsecurity.com\)](https://ptsecurity.com/ida-pro-tips-to-add-to-your-bag-of-tricks/)
- [Hex-Rays interactive operation: Mark/unmark as decompiled](https://hex-rays.com/interactive-operation-mark-unmark-as-decompiled/)
- [Hex-Rays interactive operation: Copy to assembly](https://hex-rays.com/interactive-operation-copy-to-assembly/)
- [ida - What is the meaning of \(_DWORD \) - Reverse Engineering Stack Exchange](https://reverseengineering.stackexchange.com/questions/1011/ida-what-is-the-meaning-of-_DWORD-)
- [iOS重打包绕过签名校验防护 | La0s](https://la0s.com/ios-repack-bypass-signature-validation/)
- [Interactive operation](https://hex-rays.com/interactive-operation/)
- [Failures and troubleshooting](https://hex-rays.com/failures-and-troubleshooting/)
- [Third-party plugins](https://hex-rays.com/third-party-plugins/)
- [onethawt/idaplugins-list: A list of IDA Plugins](https://onethawt.com/ida-plugins-list/)
- [IDA Help: The Interactive Disassembler Help Index](https://hex-rays.com/ida-help-the-interactive-disassembler-help-index/)
- [如何将ida中的悬浮窗口恢复原位_TedLau的博客-CSDN博客_ida窗口还原](https://www.csdn.net/article/10111111)
- [The basics of IDA pro - Infosec Resources \(infosecinstitute.com\)](https://infosecinstitute.com/the-basics-of-ida-pro/)
- [IDA Pro7.0使用技巧总结 - 先知社区 \(aliyun.com\)](https://aliyun.com/ida-pro-7.0/)

- [IDA Help: String literal style dialog \(hex-rays.com\)](#)
- [IDA Help: Processor Type \(hex-rays.com\)](#)
- [IDA-Pro-Hex-Rays.pdf \(software-sources.com\)](#)
- [IDA Help: Analysis options \(hex-rays.com\)](#)
- [Objective-C Runtime 分析 | 张不坏的博客 \(zhangbuhuai.com\)](#)
- [x86 - What's the meaning of dword_XXXX and offset dword_XXXX in IDA? - Reverse Engineering Stack Exchange](#)
- [IDA 8.1 | Hex-Rays Docs](#)
- [IDA 8.2 | Hex-Rays Docs](#)
- [IDA 8.2sp1 | Hex-Rays Docs](#)
- [IDA 8.3 | Hex-Rays Docs](#)
- [IDA 8.4 | Hex-Rays Docs](#)
- [IDA 8.4sp1 | Hex-Rays Docs](#)
- [IDA 8.4sp2 | Hex-Rays Docs](#)
- [IDA 9.0 | Hex-Rays Docs](#)
- [\[iOS Hacker\] IDA 7.0 动态调试 iOS App – exchen's blog](#)
- [Tutorial: Debugging iOS Applications with IDA Pro \(hex-rays.com\)](#)
- [ios_debugger_tutorial](#)
- Mickey Jin on X: "I really enjoy debugging everything inside the IDA Pro, because it is awesome and powerful. But I want to use the lldb command, like "po object" when debugging the objc program. Why don't you support lldb ? @HexRaysSA So I just wrote a simple script: <https://t.co/BzWltwwyLK> <https://t.co/TMDhdrPhqm>" / X (twitter.com)
- [simulator the lldb command like "po objc_object" inside the IDA debugger. \(github.com\)](#)
- [Quotes of this post / X \(twitter.com\)](#)
- [Tutorial: Debugging iOS Applications with IDA Pro \(hex-rays.com\)](#)
-

crifan.org, 使用署名4.0国际(CC BY 4.0)协议发布 all right reserved, powered by Gitbook最后更新: 2025-06-03 09:23:30