

俗话说，工欲善其事，必先利其器，在二进制安全的学习中，使用工具尤为重要，而IDA又是玩二进制的神器，以前在使用IDA的时候，只是用几个比较常用的功能，对于IDA pro权威指南》(第二版)，写下这篇文章，记录自己的学习心得，下面的记录都是在Windows平台下的IDA pro7.0进行的

一些二进制工具

在《IDA pro权威指南》的开篇一两章中，先是介绍了几款常用于二进制研究的工具，我这里简单的记了几个，介绍一波：

C++filt：

可以用于显示出c++中复杂的重载后的函数名称

PE tools：

是一组用于分析Windows系统中正在运行的进程和可执行文件的工具

string：

可以用于直接搜索出elf文件中的所有字符串

参数-a 表示搜索整个文件，参数-t 可以显示出每一个字符串的偏移，参数-e 可以用于搜索更多的字符编码的字符串，如Unicode编码

strip：

可用于elf去符号，去符号后仍然保持正常功能但增加了逆向的难度，出题恶人必备

开发了IDA的天才是Ilfak，他的个人博客有很多IDA的教程

<https://www.hexblog.com/>

IDA目录结构

在IDA的安装根目录下有许多文件夹，各个文件夹存储不同的内容

cfg：包含各种配置文件，基本IDA配置文件ida.cfg,GUI配置文件idagui.cfg，文本模式用户界面配置文件idatui.cfg,

idc：包含IDA内置脚本语言IDC所需要的核心文件

ids：包含一些符号文件

loaders：包含用于识别和解析PE或者ELF

plugins：附加的插件模块

procs：包含处理器模块

常用快捷键

IDA中的快捷键都是和菜单栏的各个功能选项——对应的，基本上你只要能在菜单栏上找到某个功能，也就能看到相应的快捷键，这里记录几个常用的：

a：将数据转换为字符串

f5：一键反汇编

esc：回退键，能够倒回上一部操作的视图（只有在反汇编窗口才是这个作用，如果是在其他窗口按下esc，会关闭该窗口）

shift+f12：可以打开string窗口，一键找出所有的字符串，右击setup，还能对窗口的属性进行设置

ctrl+w：保存ida数据库

ctrl+s：选择某个数据段，直接进行跳转

ctrl+鼠标滚轮：能够调节流程视图的大小

x：对着某个函数、变量按该快捷键，可以查看它的交叉引用

g：直接跳转到某个地址

n：更改变量的名称

y：更改变量的类型

/：在反编译后伪代码的界面中写下注释

\：在反编译后伪代码的界面中隐藏/显示变量和函数的类型描述，有时候变量特别多的时候隐藏掉类型描述看起来会轻松很多

；：在反汇编后的界面中写下注释

ctrl+shift+w：拍摄IDA快照

u：undefine，取消定义函数、代码、数据的定义

常用设置

拍摄快照

由于IDA不提供撤销的功能，如果你不小心按到某个键，导致ida数据库发生了改变，就得重新来过，所以要记得在经常操作的时候，加上快照：file-->take database snapshot

加完快照后，会生成一个新的ida数据库文件，本质上是有点像另存的操作

快捷键：ctrl+shift+w

菜单栏常用设置

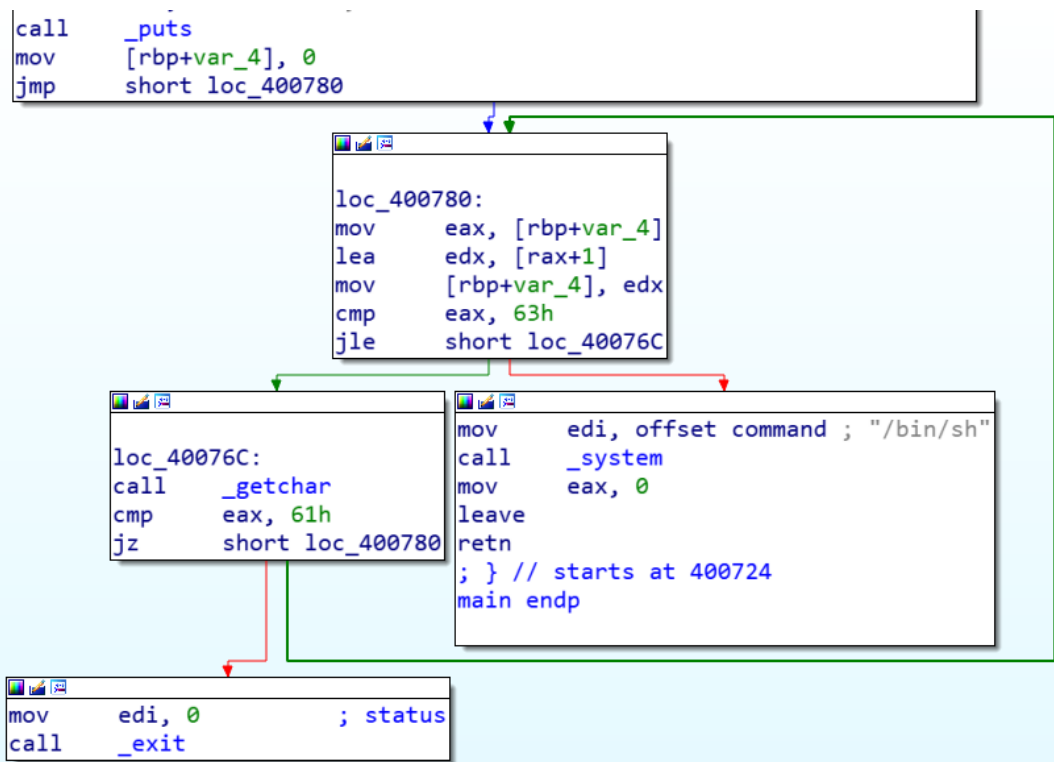
view-->open subviews: 可以恢复你无意中关闭的数据显示窗口

windows-->reset desktop: 可以恢复初始ida布局

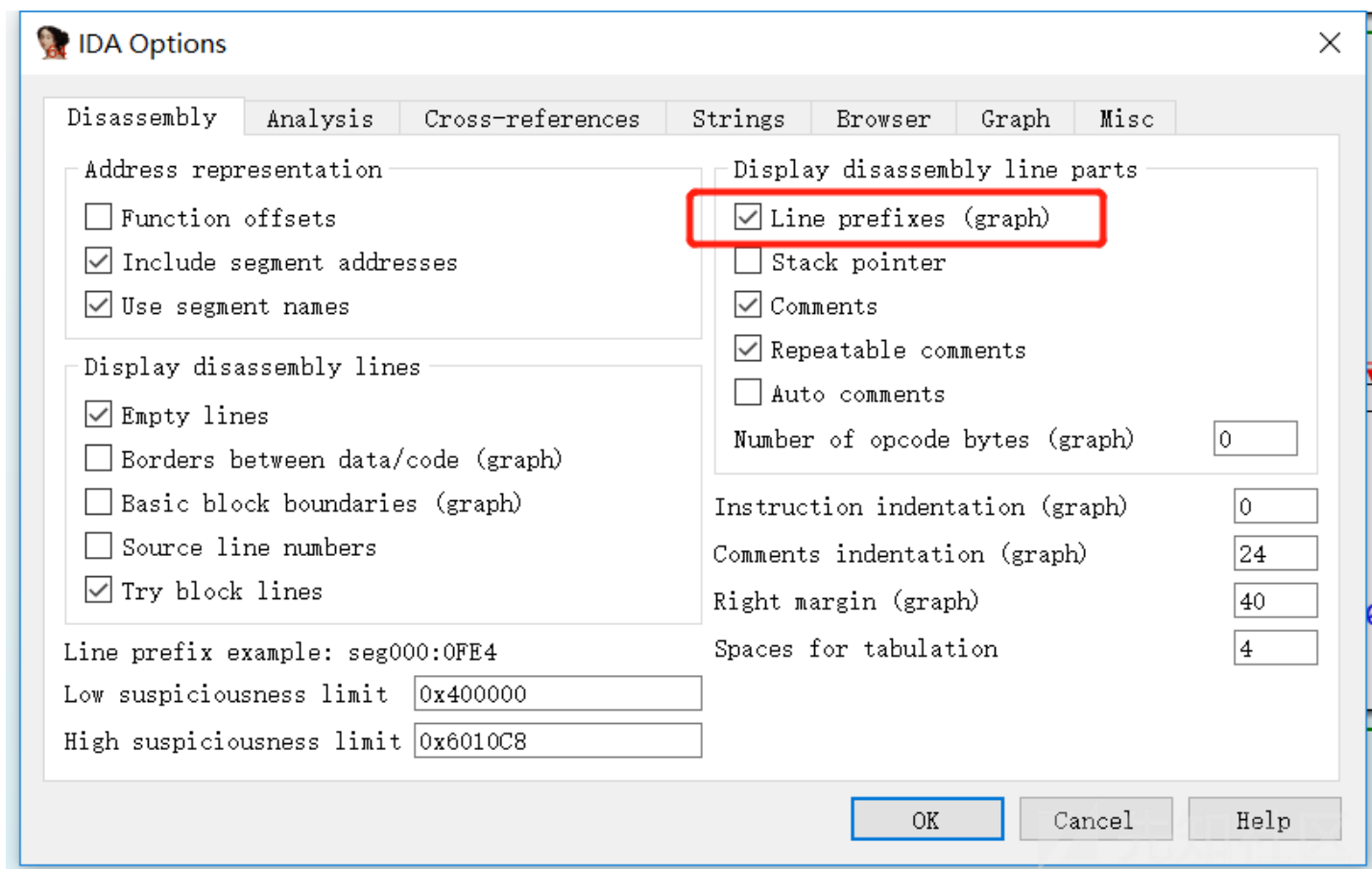
option-->font: 可以改变字体的相关属性

在流程视图中添加地址偏移

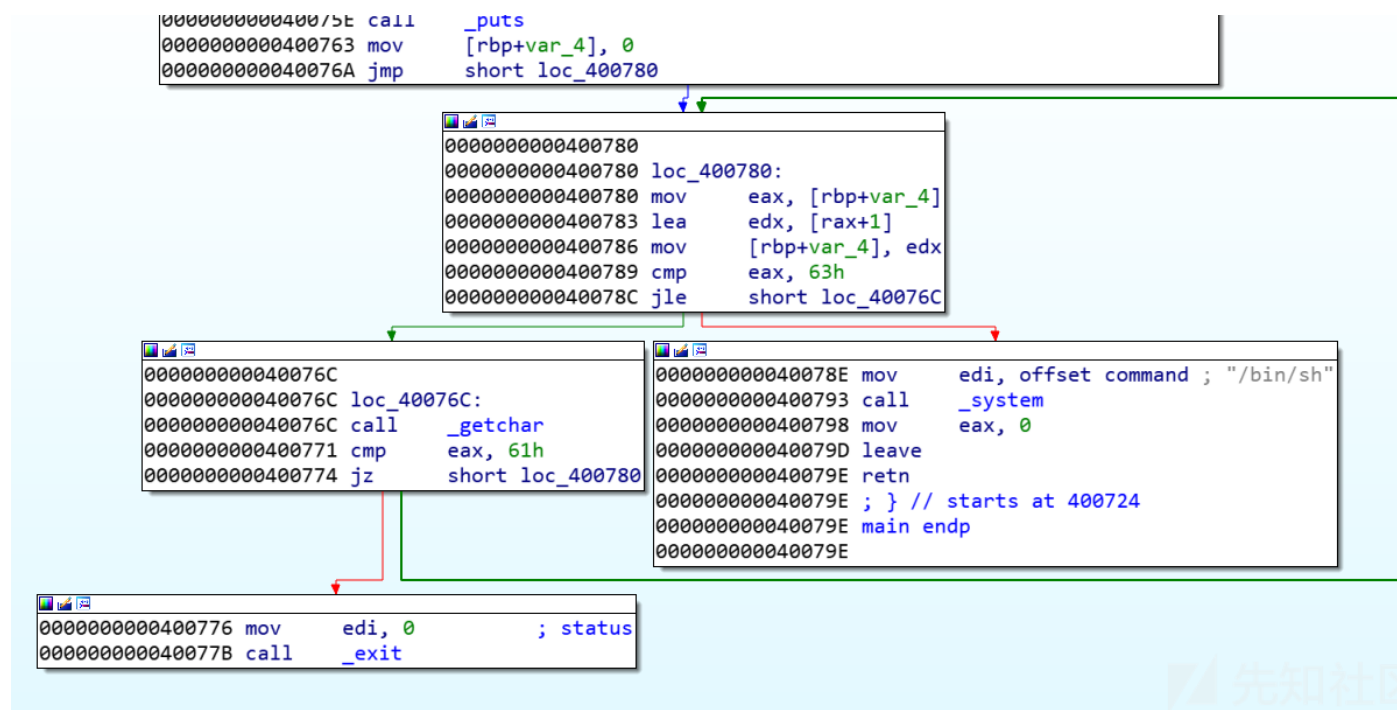
IDA中的流程视图可以说是非常的好用，简单明了地能看出程序的执行流程，尤其是在看if分支代码和循环代码的时候，能够非常直观



但是，我们还可以改得更加好用，在这个视图中添加地址偏移的话，我们取地址就非常方便，不再需要按空格切换视图去找，在菜单栏中设置：option-->general



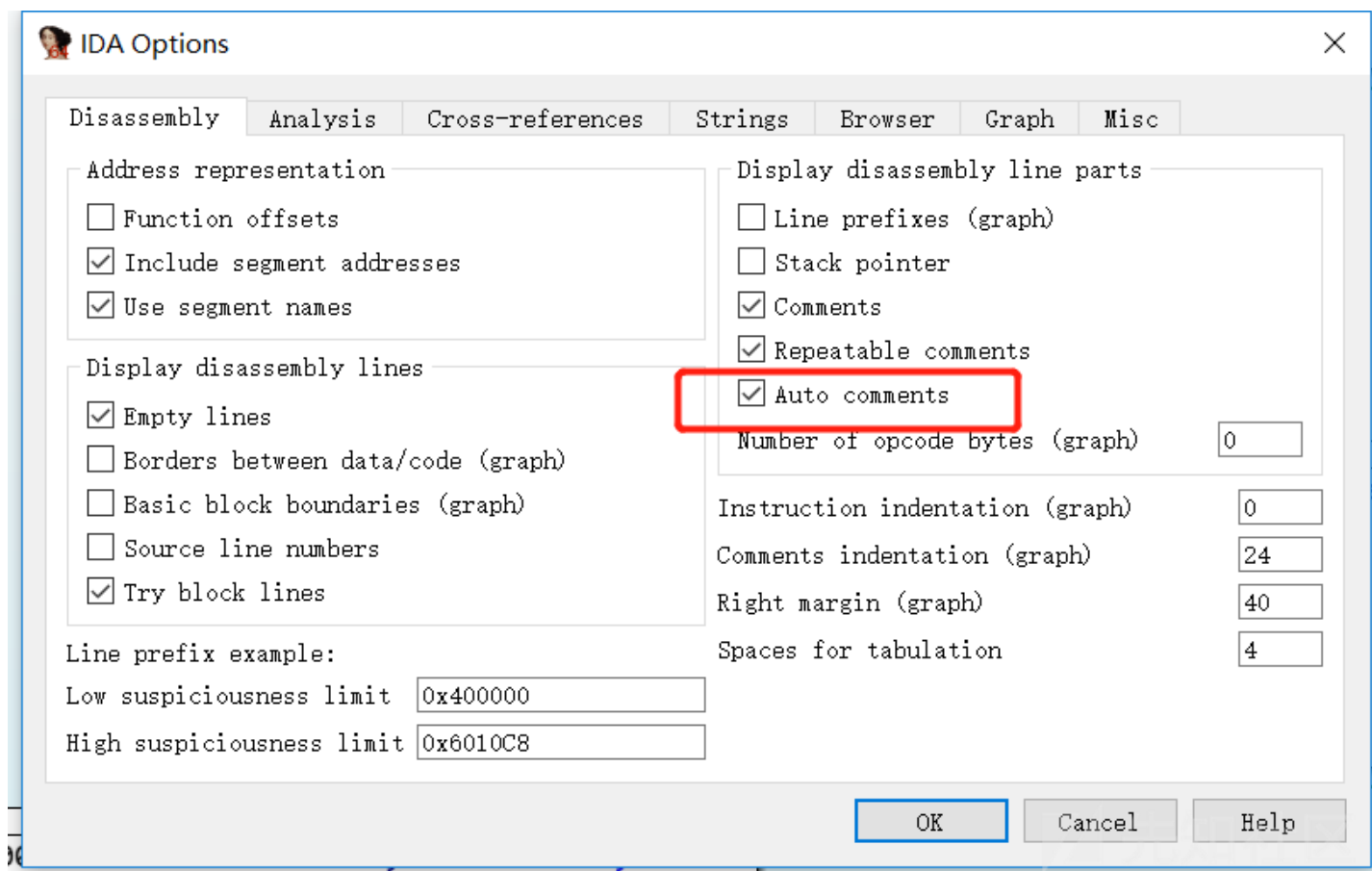
将该选项打钩后就可以看到效果了：



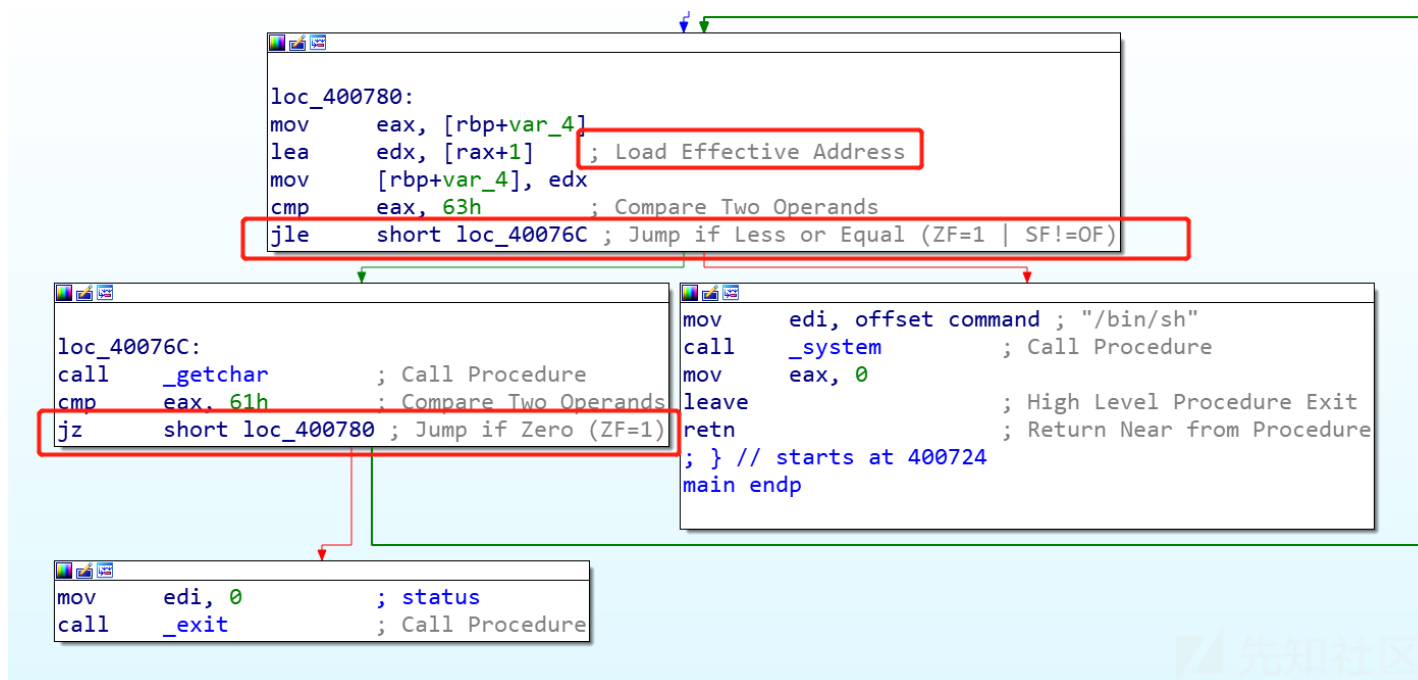
自动添加反汇编注释

这个功能对于萌新来说非常友好，在刚刚初学汇编的时候，难免遇到几个不常用的蛇皮汇编指令，就得自己一个去查，很麻烦，开启了自动注释的功能后，IDA就可以直接告诉你汇编指令的意思

同样是在菜单栏中设置：option-->general



效果如下：



常用操作

创建数组

在操作IDA的时候，经常会遇到需要创建数组的情况，尤其是为了能方便我们看字符串的时候，创建数组显得非常必要，以下我随便找了个数据来创建数组


首先点击选中你想要转换成数组的一块区域：

```

• :0000000000400DAF db 0
• :0000000000400DB0 db 64h ; d
• :0000000000400DB1 db 0FAh
• :0000000000400DB2 db 0FFh
• :0000000000400DB3 db 0FFh
• :0000000000400DB4 db 84h
• :0000000000400DB5 db 0
• :0000000000400DB6 db 0
• :0000000000400DB7 db 0
• :0000000000400DB8 db 5Ah ; Z
• :0000000000400DB9 db 0FBh
• :0000000000400DBA db 0FFh
• :0000000000400DBB db 0FFh
• :0000000000400DBC db 0DCh
• :0000000000400DBD db 0
• :0000000000400DBE db 0
• :0000000000400DBF db 0
• :0000000000400DC0 db 0

```

接着在菜单栏中选择：edit-->array，就会弹出如下的选项框


Convert to array
×

Start address : .eh_frame_hdr:0000000000400DB0

End address : .eh_frame_hdr:0000000000400DC0

Array element size : 1

Maximal possible size: 88

Current array size : 1

Suggested array size : 16

Array size (in elements)

Items on a line (0-max)

Element print width (-1-none, 0-auto)

Options
☒ Use "dup" construct
☐ Signed elements
☐ Display indexes
☒ Create as array

Indexes
☒ Decimal
☐ Hexadecimal
☐ Octal
☐ Binary

下面来解释一下各个参数的意思：

Array element size 这个值表示各数组元素的大小（这里是1个字节），是根据你选中的数据值的大小所决定的

Maximum possible size 这个值是由自动计算得出的，他表示数组中的元素的可能的最大值

Array size 表示数组元素的数量，一般都根据你选定的自动产生默认值

Items on a line 这个表示指定每个反汇编行显示的元素数量，它可以减少显示数组所需的空间

Element print width 这个值用于格式化，当一行显示多个项目时，他控制列宽

Use "dup" construct : 使用重复结构，这个选项可以使得相同的数据值合并起来，用一个重复说明符组合成一项

Signed elements 表示将数据显示为有符号数还是无符号数

Display indexes 显示索引，使得数组索引以常规的形式显示，如果选了这个选项，还会启动右边的Indexes选项栏，用于选择索引的显示格式

Create as array 创建为数组，这个一般默认选上的

创建好了以后，就变成了这样：

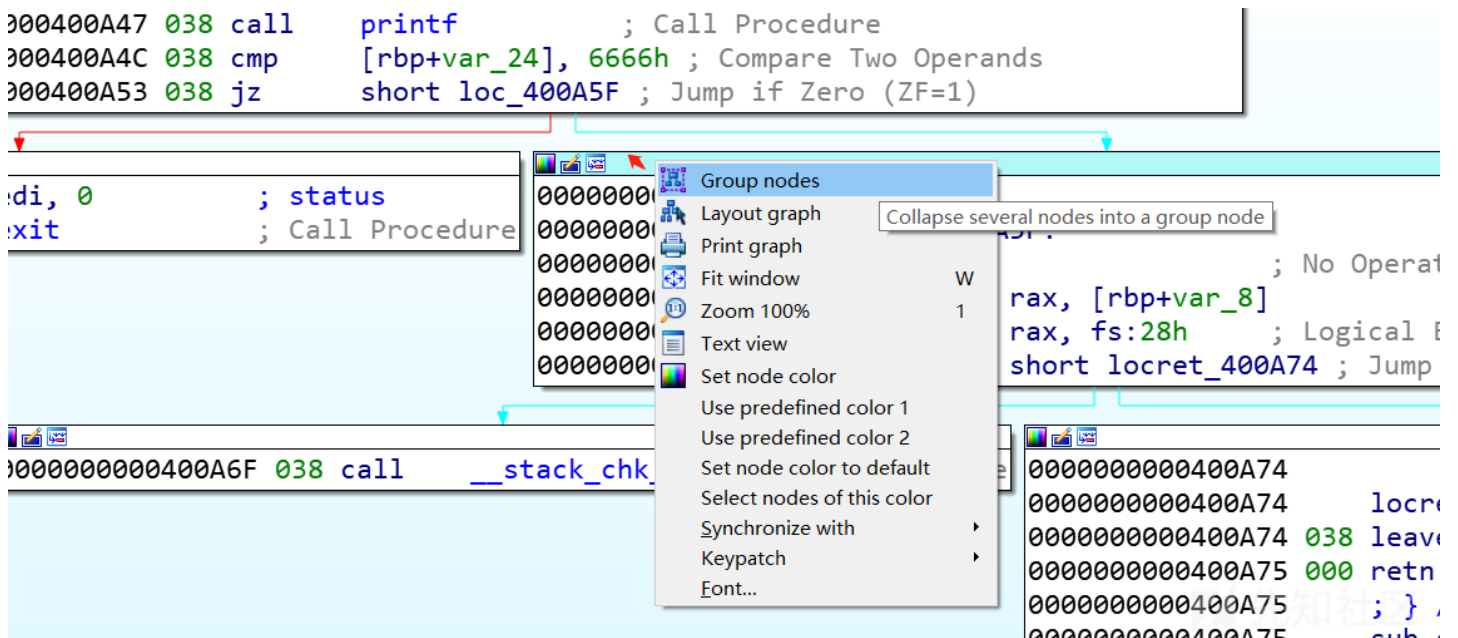
```
000000400DAF          db      0
000000400DB0          db      64h,0FAh,2 dup(0FFh), 84h,3 dup(  0), 5Ah,0FBh,2 dup(0FFh)
000000400DB0          db      0DCh,3 dup(  0)
000000400DC0          db      0BCh
000000400DC1          db      0FBh
)
```

可以看到这些数据已经被当成一个数组折叠到了一起，其中2 dup(0FFh)这样的，表示有两个重复的数据0xff

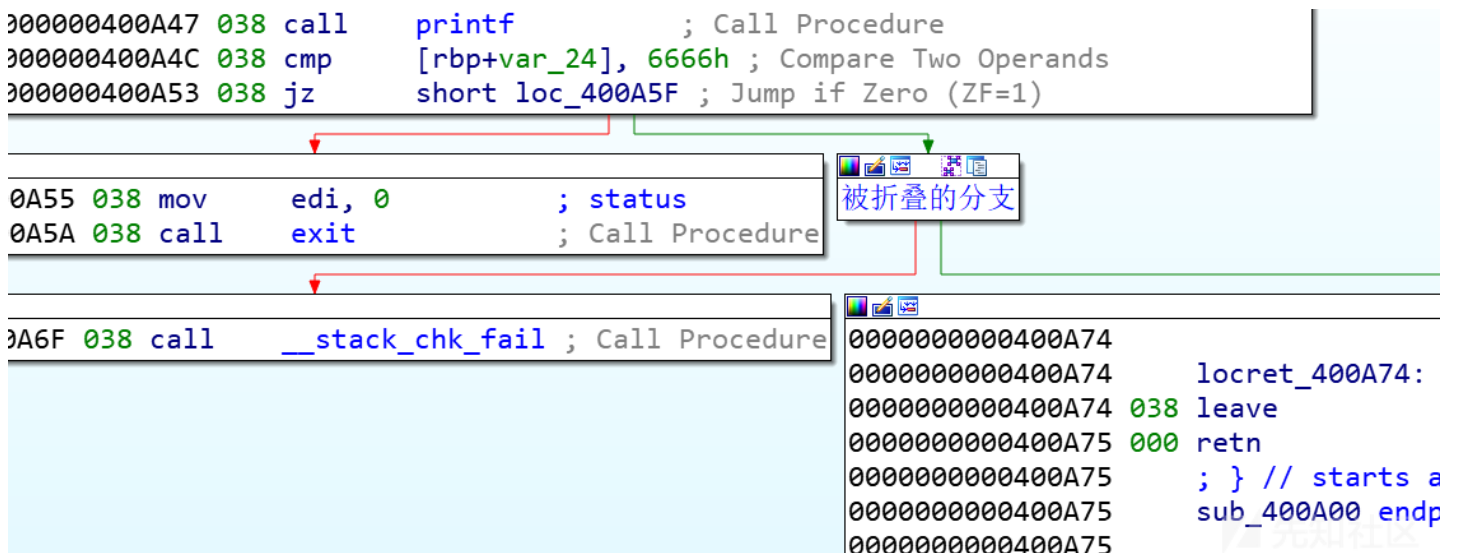
流程图

折叠流程图中的分支

在流程视图中，分支过多的时候，可以在窗口标题处右击选择group nodes，就能把当前块折叠起来



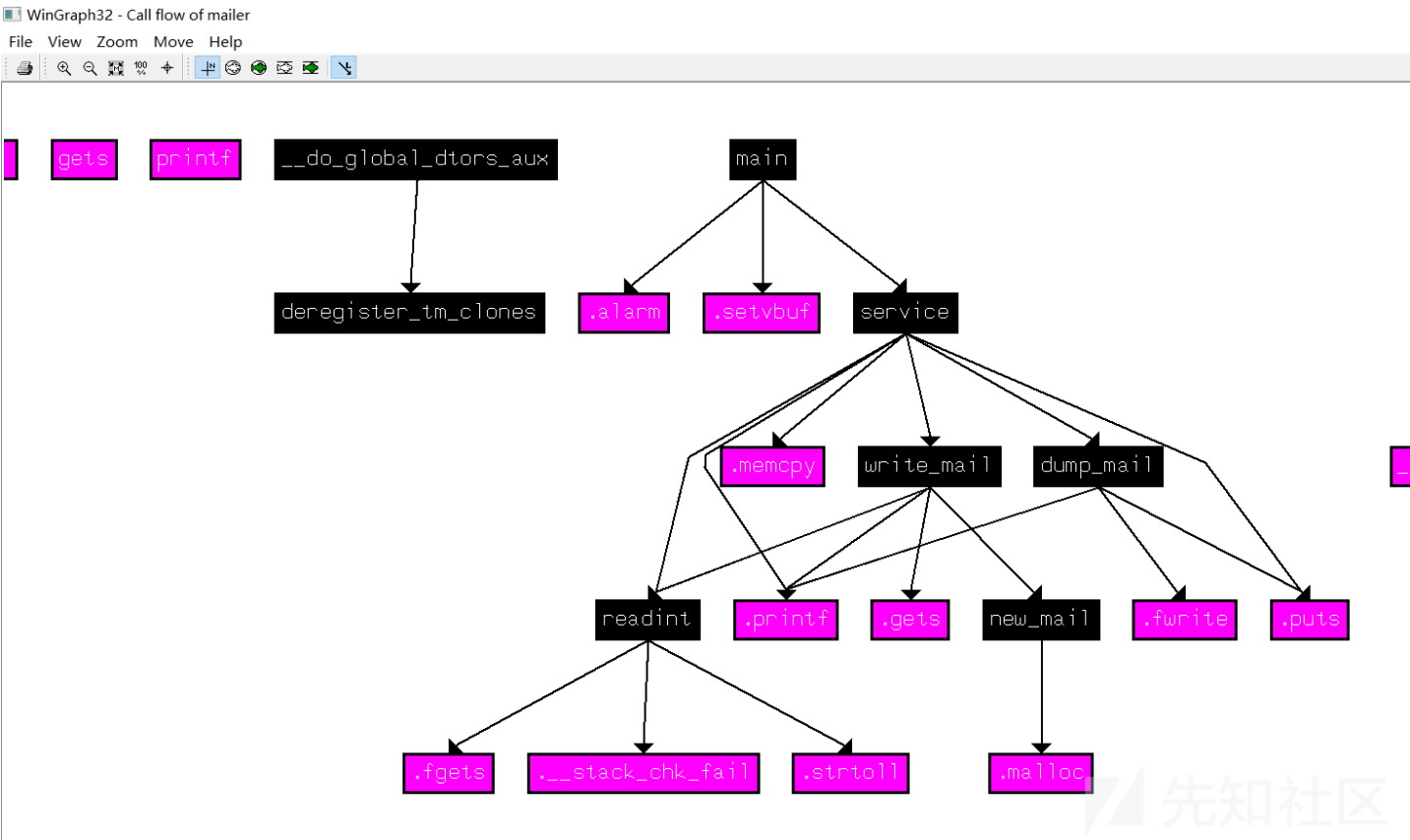
效果如下:



分支块是可以自己命名的，方便自己逆向理解

函数调用图

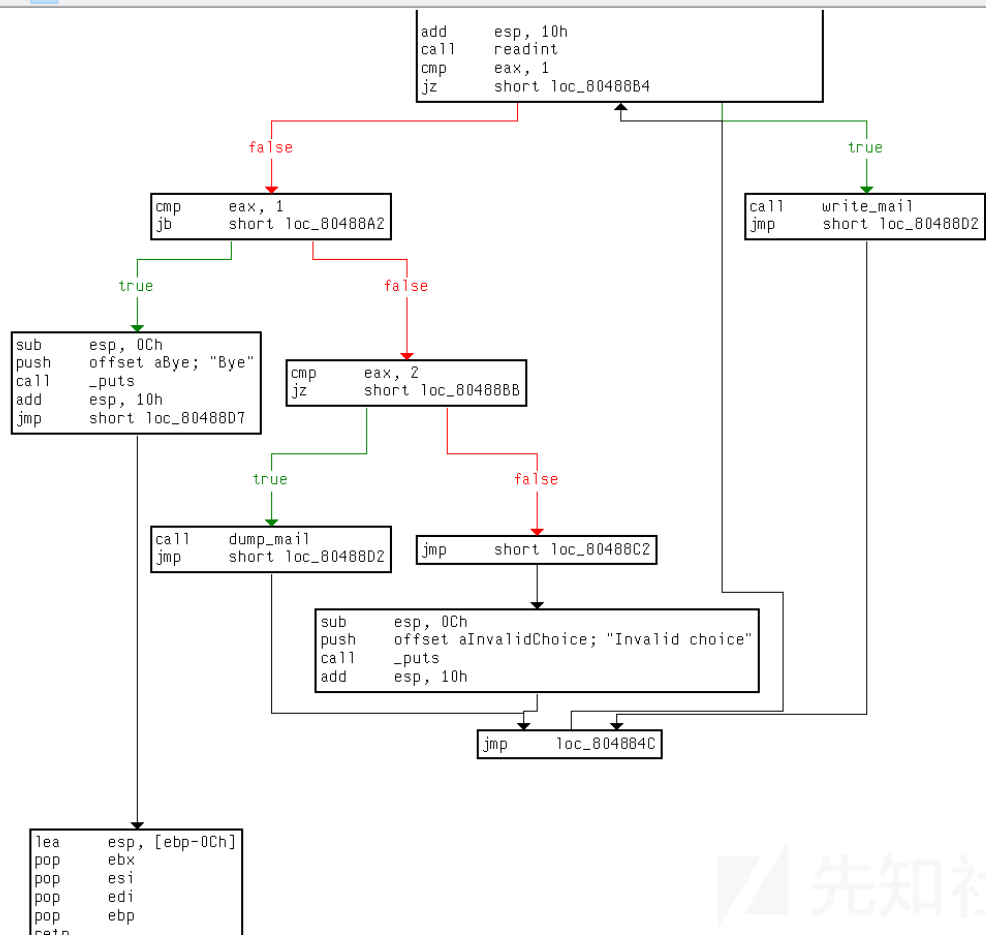
菜单栏中：view-->graphs-->Function calls(快捷键Ctrl+F12)



这个图能很清楚地看到函数之间是如何相互调用的

函数流程图

菜单栏中：view-->graphs-->flowt chart(快捷键F12)

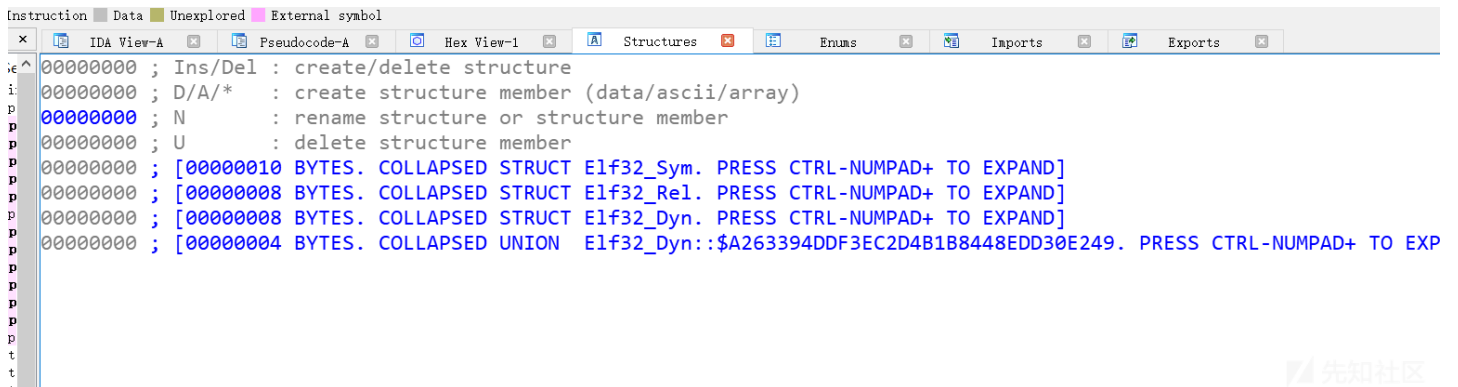


这个其实跟IDA自带的反汇编流程图差不多，他可以导出来作为单独的一张图

创建结构体：

手工创建结构体

创建结构体是在IDA的structures窗口中进行的，这个操作在堆漏洞的pwn题中经常使用



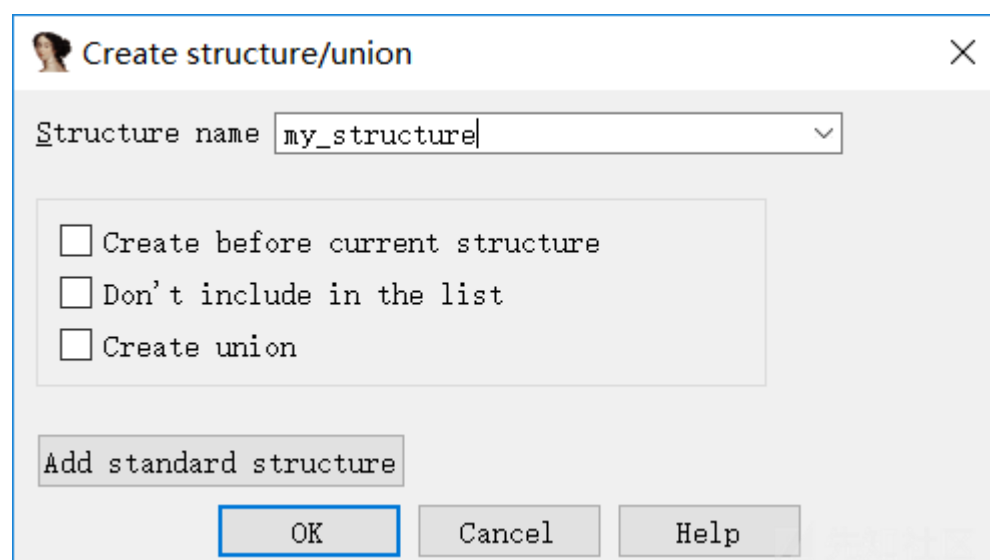
可以看到，这里已经存在了四个结构体，程序本身存在的，可以右击选择hide/unhide,来看具体的结构体的内容


```

IDA View-A  Pseudocode-A  Hex View-1  Structures  Enums  Imports
00000000 ; Ins/Del : create/delete structure
00000000 ; D/A/* : create structure member (data/ascii/array)
00000000 ; N : rename structure or structure member
00000000 ; U : delete structure member
00000000 ; -----
00000000
00000000 Elf32_Sym      struc ; (sizeof=0x10, align=0x4, mappedto_1)
00000000                                ; XREF: LOAD:080481D8/r
00000000                                ; LOAD:080481E8/r ...
00000000 st_name          dd ? ; offset (080482E8)
00000004 st_value     dd ? ; offset (00000000)
00000008 st_size      dd ?
0000000C st_info       db ?
0000000D st_other      db ?
0000000E st_shndx     dw ?
00000010 Elf32_Sym      ends
00000010
00000000 ; [00000008 BYTES. COLLAPSED STRUCT Elf32_Rel. PRESS CTRL-NUMPAD+ TO EXPAND]
00000000 ; [00000008 BYTES. COLLAPSED STRUCT Elf32_Dyn. PRESS CTRL-NUMPAD+ TO EXPAND]
00000000 ; [00000004 BYTES. COLLAPSED UNION Elf32_Dyn::$A263394DDF3EC2D4B1B8448EDD30E249. PRE

```

创建结构体的快捷键是：insert



在弹出的窗口中，可以编辑结构体的名字

这底下有三个复选框，第一个表示显示在当前结构体之前（就会排列在第一位，否则排列在你鼠标选定的位置），第二个表示是否在窗口中显示新的结构体，第三个表示是否

需要注意的是，结构体的大小是它所包含的字段大小的总和，而联合体的大小则等于其中最大字段的大小

在单击ok以后，就定好了一个空的结构体：

```

00000000 ; -----
00000000
00000000 my_structure  struc ; (sizeof=0x0, mappedto_5)
00000000 my_structure  ends
00000000

```

将鼠标放在 ends这一行，单击快捷键D即可添加结构体成员，成员的命名默认是以field_x表示的，x代表了该成员在结构体中的偏移

```

00000000
00000000 my_structure      struc ; (sizeof=0x7, mappedto_5)
00000000 field_0           db ?
00000001 field_1           db ?
00000002 field_2           dd ?
00000006 field_6           db ?
00000007 my_structure      ends
00000007

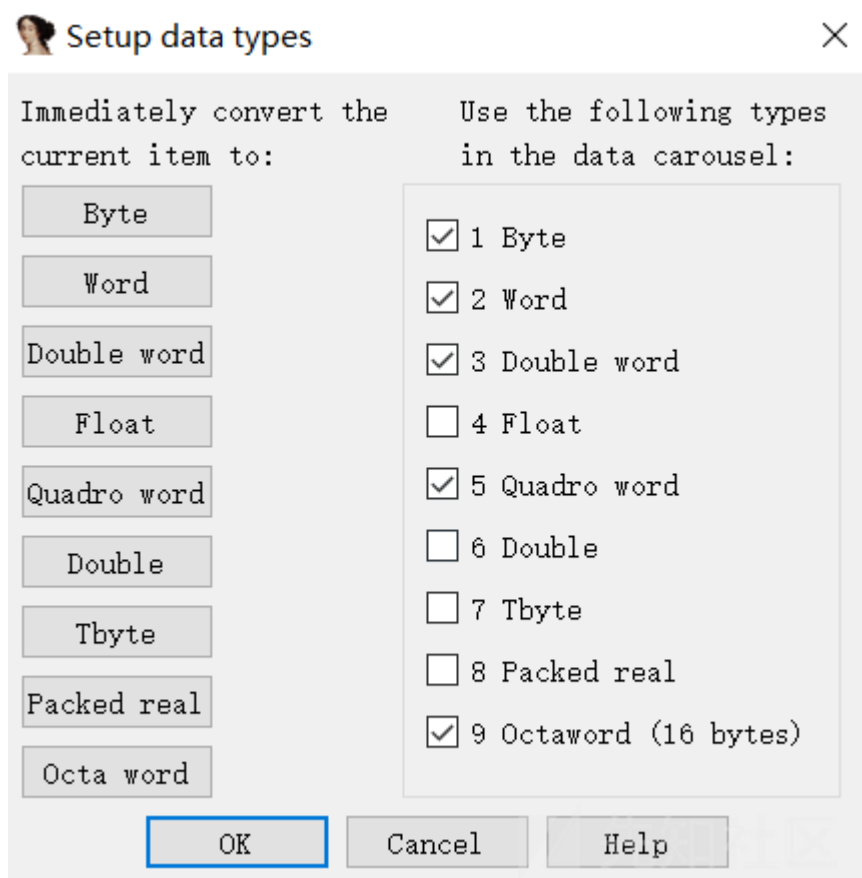
```

先知社区

同时，可以把鼠标放在结构体成员所在的行，按D，就可以切换不同的字节大小

默认情况下可供选择的就只有db，dw，dd（1，2，4字节大小）

如果想添加型的类型，可以在option-->setup data types(快捷键Alt+D)，进行设置



如图，勾选了第五个和第九个的话，就会出现dq和xmmword了（代表了8字节和16字节）

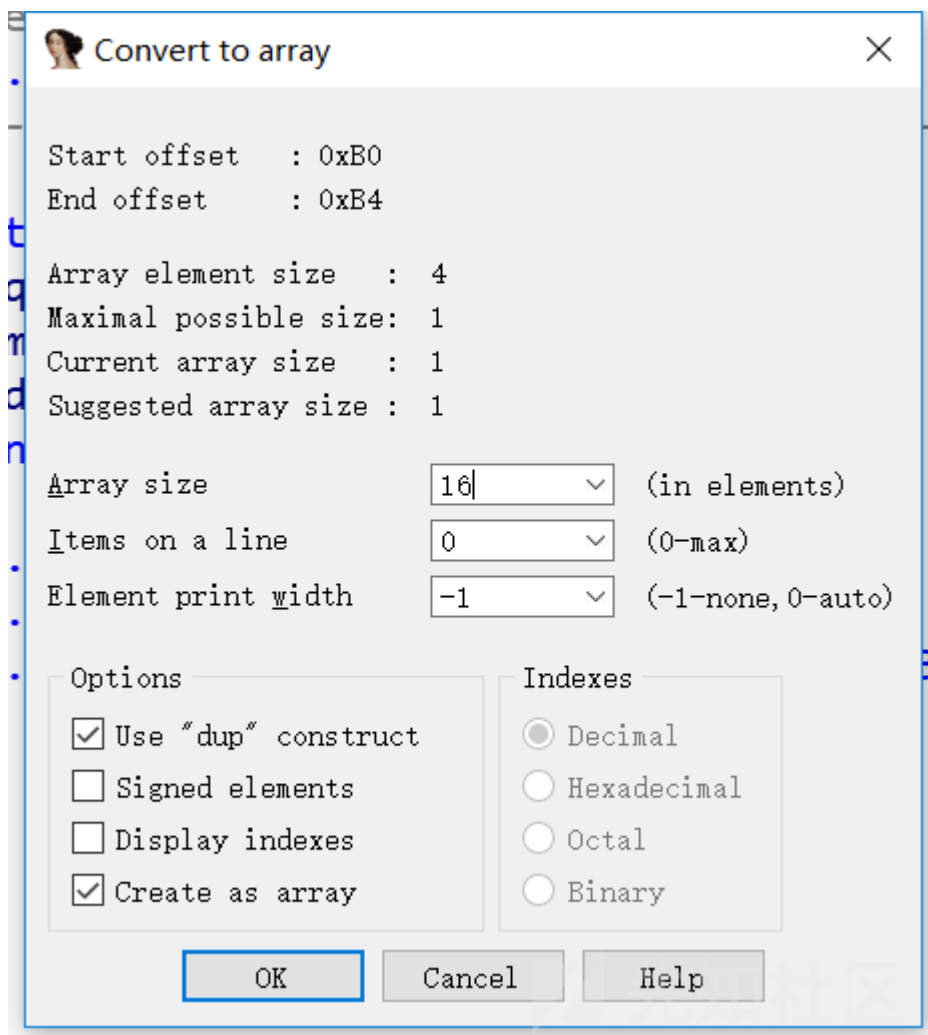
```

0000
0000 my_structure      struc ; (sizeof=0x1F, mappedto_5)
0000 field_18           db ?
0001 field_1           dw ?
0003 field_3           dd ?
0007 field_7           dq ?
000F field_F           xmmword ?
001F my_structure      ends

```

先知社区

如果要添加数组成员则可以对着成员所在的那一行，右击选择array



如图，要创建的是16个元素的4字节数组

如果要删除结构体，那么对着结构体按下delete键即可删除

如果要删除成员，则对着成员按下u (undefine) 但是需要注意的是，这里只是删除了成员的名字，而没有删除它所分配的空间

如图，我们删除了中间的field_10的数组成员：

```

00000000
00000000 my_structure      struc ; (sizeof=0x25, mi
00000000 field_A0          xmmword ?
00000100 field_10          db 20 dup(?)
00000240 field_24          db ?
00000250 my_structure      ends
00000250

```

会变成这样：

```

00000000
00000000 my_structure      struc ; (sizeof=0x25, ma
00000000 field_A0          xmmword ?
00000010                db ? ; undefined
00000011                db ? ; undefined
00000012                db ? ; undefined
00000013                db ? ; undefined
00000014                db ? ; undefined
00000015                db ? ; undefined
00000016                db ? ; undefined
00000017                db ? ; undefined
00000018                db ? ; undefined
00000019                db ? ; undefined
0000001A                db ? ; undefined
0000001B                db ? ; undefined
0000001C                db ? ; undefined
0000001D                db ? ; undefined
0000001E                db ? ; undefined
0000001F                db ? ; undefined
00000020                db ? ; undefined
00000021                db ? ; undefined
00000022                db ? ; undefined
00000023                db ? ; undefined
00000024 field_24          db ?
00000025 my_structure     ends

```

数组所分配的20个字节的空间并没有被删除，这时如果要删除掉这些空间，就需要在原来数组成员所在的第一行中按下Ctrl+S，删除空间（Edit-->shrink struct types）

就可以真正的删除掉成员

给结构体的成员重命名可以用快捷键N

我们在IDA中创建好了结构体以后，就是去应用它了

如图，这是一个典型的堆的题目

```

1 mail *write_mail()
2 {
3     int length; // eax
4     char *v1; // ST1C_4
5     mail *result; // eax
6
7     printf("Content Length: ");
8     length = readint();
9     v1 = (char *)new_mail(length);
10    printf("Title: ");
11    gets(v1 + 4);
12    printf("Content: ");
13    gets(v1 + 0x48);
14    *(DWORD *)v1 = root;
15    result = (mail *)v1;
16    root = (int)v1;
17    return result;
18 }

```



可以看到v1是一个新建的chunk的地址指针，而后的操作都是往chunk不同的偏移位置写入内容，为了方便我们逆向观察，可以将其变成一个结构体，通过v1+4 v1+0x48 这样的偏移，创建好结构体后，将char *v1的类型改成mail

*v1, (快捷键Y可以更改函数、变量的类型和参数) 这个mail是我们创建的结构体的名称，效果如下：

```

1 mail *write_mail()
2 {
3     int length; // eax
4     mail *v1; // ST1C_4
5     mail *result; // eax
6
7     printf("Content Length: ");
8     length = readint();
9     v1 = (mail *)new_mail(length);
10    printf("Title: ");
11    gets((char *)&v1->title);
12    printf("Content: ");
13    gets((char *)&v1->content);
14    v1->root = (char *)root;
15    result = v1;
16    root = (int)v1;
17    return result;
18 }

```

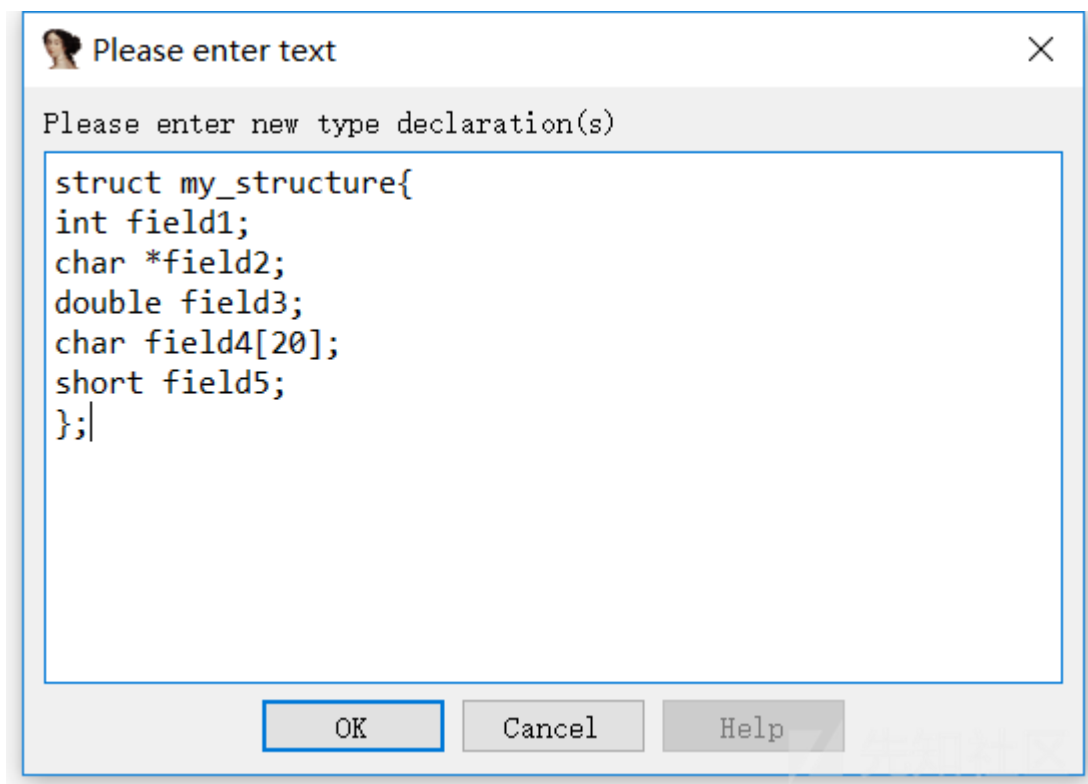


导入C语言声明的结构体

实际上，IDA有提供一个更方便的创建结构体的方法，就是直接写代码导入

在View-->Open Subviews-->Local Types中可以看到本地已有的结构体，在该窗口中右击insert

可以添加新的结构体：

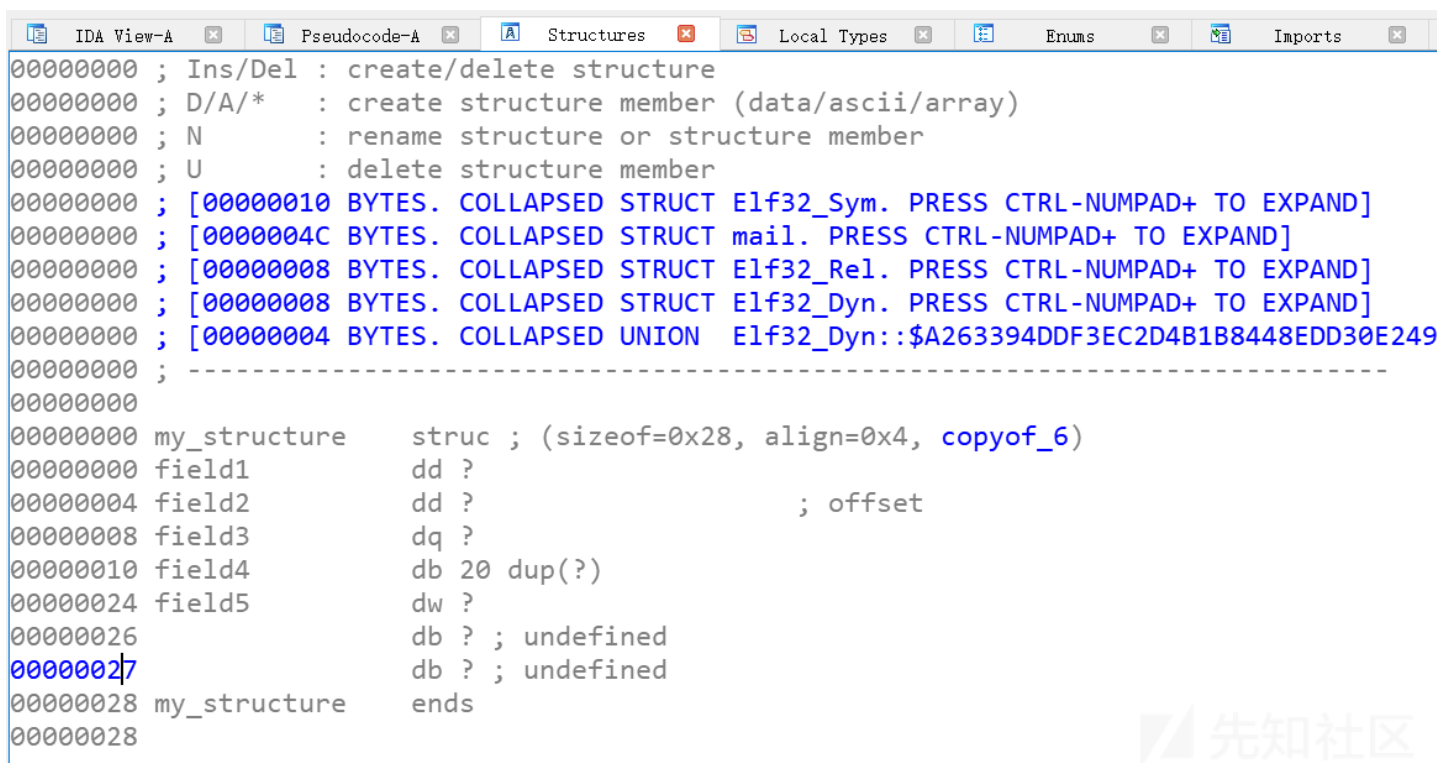


这样就导入了新的结构体：

Ordinal	Name	Size	Sync	Description
1	Elf32_Sym	00000010	Auto	struct __attribute__((aligned(4))) {unsigned __int32 st_name;unsigned __int32 s...
2	Elf32_Rel	00000008	Auto	struct {unsigned __int32 r_offset;unsigned __int32 r_info;}
3	Elf32_Dyn	00000008	Auto	struct {__int32 d_tag;union Elf32_Dyn::\$A263394DDF3EC2D4B1B8448EDD30E249 d_un;}
4	Elf32_Dyn::\$A263394...	00000004	Auto	union {unsigned __int32 d_val;unsigned __int32 d_ptr;}
5	mail	0000004C	Auto	struct {char *root;char *title;_BYTE gap8[60];int length;char *content;}
6	my_structure	00000028		struct {int field1;char *field2;double field3;char field4[20];__int16 field5;}

但同时我们发现structure视图里面，并没有这个结构体，我们需要对着my_structure右击，选择 synchronize to idb

这样structure视图就有了,如图



这里你会发现，多出来两个db的undefined的成员，这是因为ida默认是会把结构体统一4字节对齐的，满足结构体的大小为0x28

IDA动态调试elf：

这里我以在一个在Ubuntu虚拟机中的elf为例子，进行调试

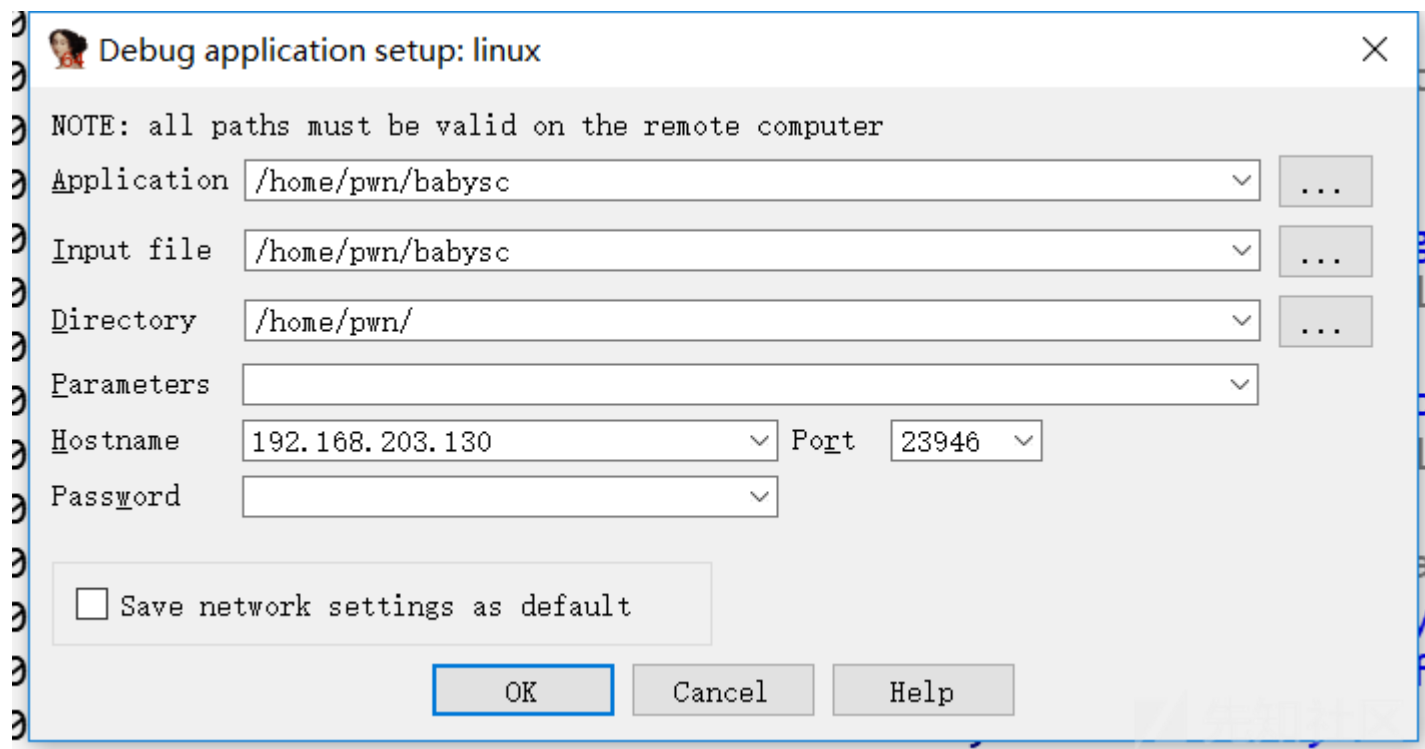
首先把ida目录中的dbgsvr文件夹中的linux_server64拷贝到Ubuntu的elf的文件下，这个elf是64位的所有用的是linux_server64，如果你调试的是32位的程序，你就需要

记得给他们权限，然后在终端运行，这个程序的作用就像是连接ida和虚拟机中elf的桥梁

```
z3.c@ubuntu:~/pwn$ ./linux_server64
IDA Linux 64-bit remote debug server(ST) v1.22. Hex-Rays (c) 2004-2017
Listening on 0.0.0.0:23946...
=====
```

然后再到ida中进行配置：

在菜单栏中选择：debugger-->process options



注意，application和input file 都是填写在虚拟机中的elf的路径，记得要加文件名

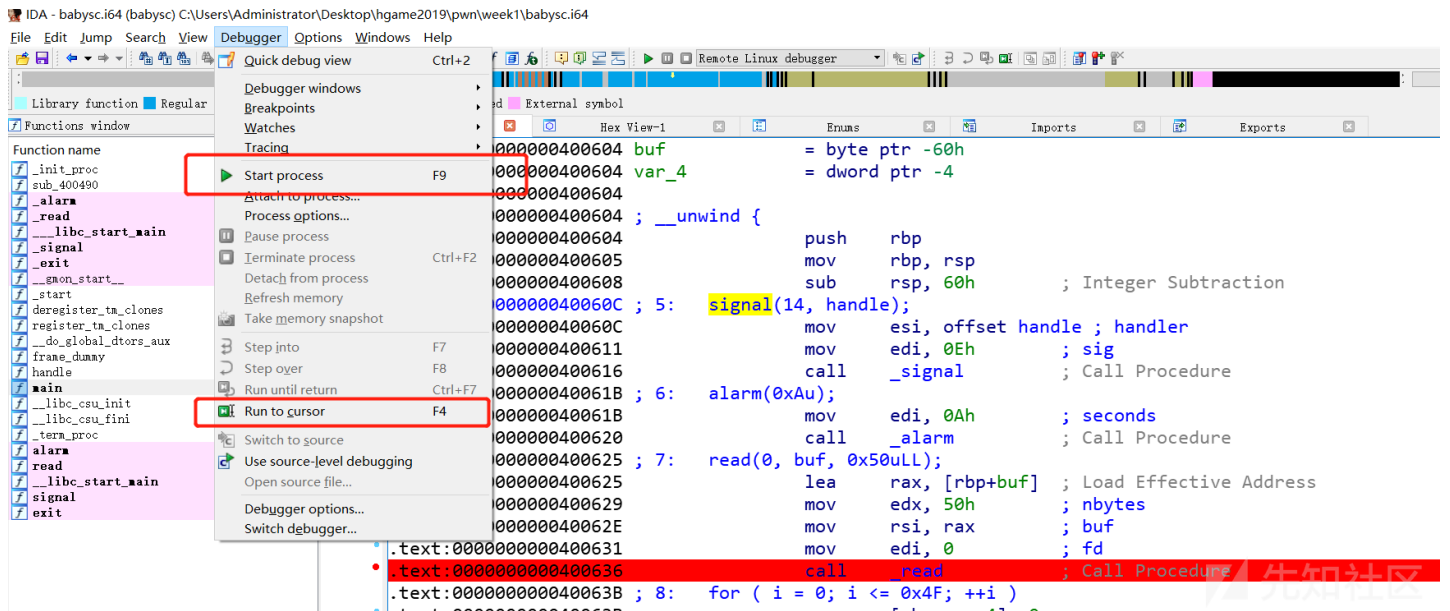
而directory 填写elf所在目录，不用加文件名

hostname是虚拟机的ip地址，port是默认的连接端口

parameter和password一般都不用填

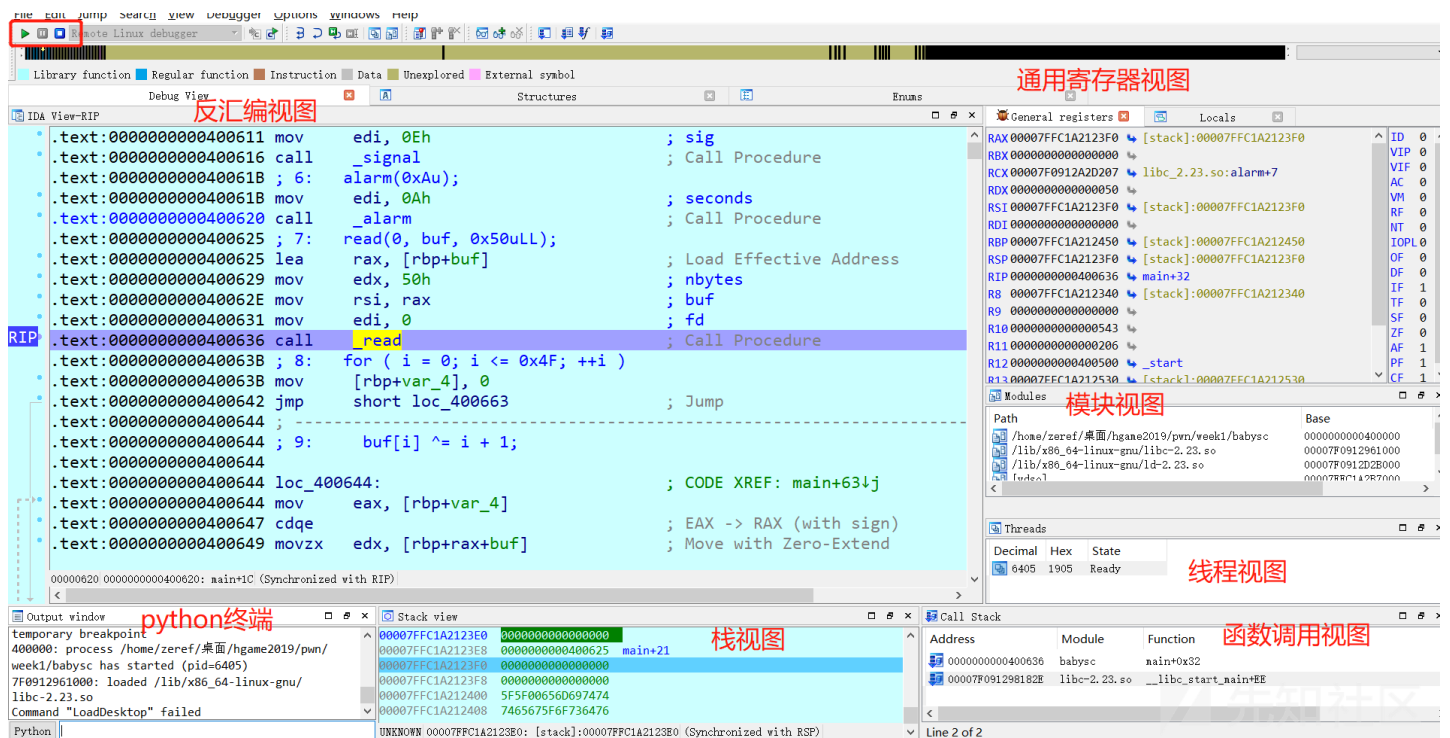
设置好了以后点击ok

接着可以直接在反汇编视图中下断点，只要点击左边的小蓝点即可



这时按下快捷键F9，可以直接开始调试

按下快捷键F4，则直接运行到断点处停下



这个就是基本的各个功能区的介绍，上面是我比较喜欢的常用布局，和ida默认的不太一样，想要自定义添加一些视图的话，可以在debugger-->quick debug view中添加

另外可以在Windows-->save desktop来保持当前的视图布局，以后就可以直接加载使用

下面介绍一些常用的快捷键

- F7 单步入，遇到函数，将进入函数代码内部
- F8 单步过，执行下一条指令，不进入函数代码内部
- F4 运行到光标处（断点处）
- F9 继续运行
- CTRL+F2 终止一个正在运行的调试进程
- CTRL+F7 运行至返回，直到遇到RETN（或断点）时才停止。

知道了这些快捷键后，调试起来就比较容易了，ida调试有个比较方便的地方在于能直接看到函数的真实地址，下断点也非常直观易操作

IDA-python

在IDA的最下面有个不起眼的Output Window的界面，其实是一个终端界面，这里有python终端和IDC终端

```
Output window
Python>print("adasdasd")
adasdasd
Python>print asdsad
Traceback (most recent call last):
  File "<string>", line 1, in <module>
NameError: name 'asdsad' is not defined
Python>print "asdasdasd"
asdasdasd
Python>import pwn
Traceback (most recent call last):
  File "<string>", line 1, in <module>
ImportError: No module named pwn
```

Python

AU: IDC - Native built-in language
Python - IDAPython plugin

这里的python是2.7的版本，虽然老了点，但已经足够我们用了，在IDA的运用中，我们经常需要计算地址，计算偏移，就可以直接在这个终端界面进行操作，非常方便

当然上面说的只是很简单的python用法，真正的IDA-python的用法是这样的：

这里以简单的一道逆向题来做例子

```
1 int __cdecl main(int argc, const char **argv, const char **envp)
2 {
3     char s; // [rsp+0h] [rbp-20h]
4     int v5; // [rsp+18h] [rbp-8h]
5     int i; // [rsp+1Ch] [rbp-4h]
6
7     for ( i = 0; i <= 181; ++i )
8     {
9         envp = (*(judge + i) ^ 0xCu);
10        *(judge + i) ^= 0xCu;
11    }
12    printf("Please input flag:", argv, envp);
13    __isoc99_scanf("%20s", &s);
14    v5 = strlen(&s);
15    if ( v5 == 14 && judge(&s) )
16        puts("Right!");
17    else
18        puts("Wrong!");
19    return 0;
20 }
```

这个程序很简单，一开始来个for循环，把judge函数的内容全部异或0xc，这样就导致了程序一运行就会直接破坏掉judge函数


```

15 char v14; // [rsp+14h] [rbp-14h]
16 char v15; // [rsp+15h] [rbp-13h]
17 int i; // [rsp+24h] [rbp-4h]
18
19 v2 = 102;
20 v3 = 109;
21 v4 = 99;
22 v5 = 100;
23 v6 = 127;
24 v7 = 107;
25 v8 = 55;
26 v9 = 100;
27 v10 = 59;
28 v11 = 86;
29 v12 = 96;
30 v13 = 59;
31 v14 = 110;
32 v15 = 112;
33 for ( i = 0; i <= 13; ++i )
34     *(_BYTE *)(i + a1) ^= i;
35 for ( i = 0; i <= 13; ++i )
36 {
37     if ( *(_BYTE *)(i + a1) != *(&v2 + i) )
38         return 0LL;
39 }
40 return 1LL;
41}

```

00000B00 41400015 (600B00)

这只是一个简单的IDApython的使用例子，实际上这个功能非常强大，能弄出非常骚的操作

打PATCH

打patch，其实就是给程序打补丁，本质上是修改程序的数据，指令等，这在CTF中的AWD赛制中经常用到，发现程序漏洞后马上就要用这个功能给程序打好patch，防止其

这里，我是用一个叫keypatch的插件进行操作的，IDA自带的patch功能不太好用

安装keypatch

这个很简单，教程在[github](#)就有

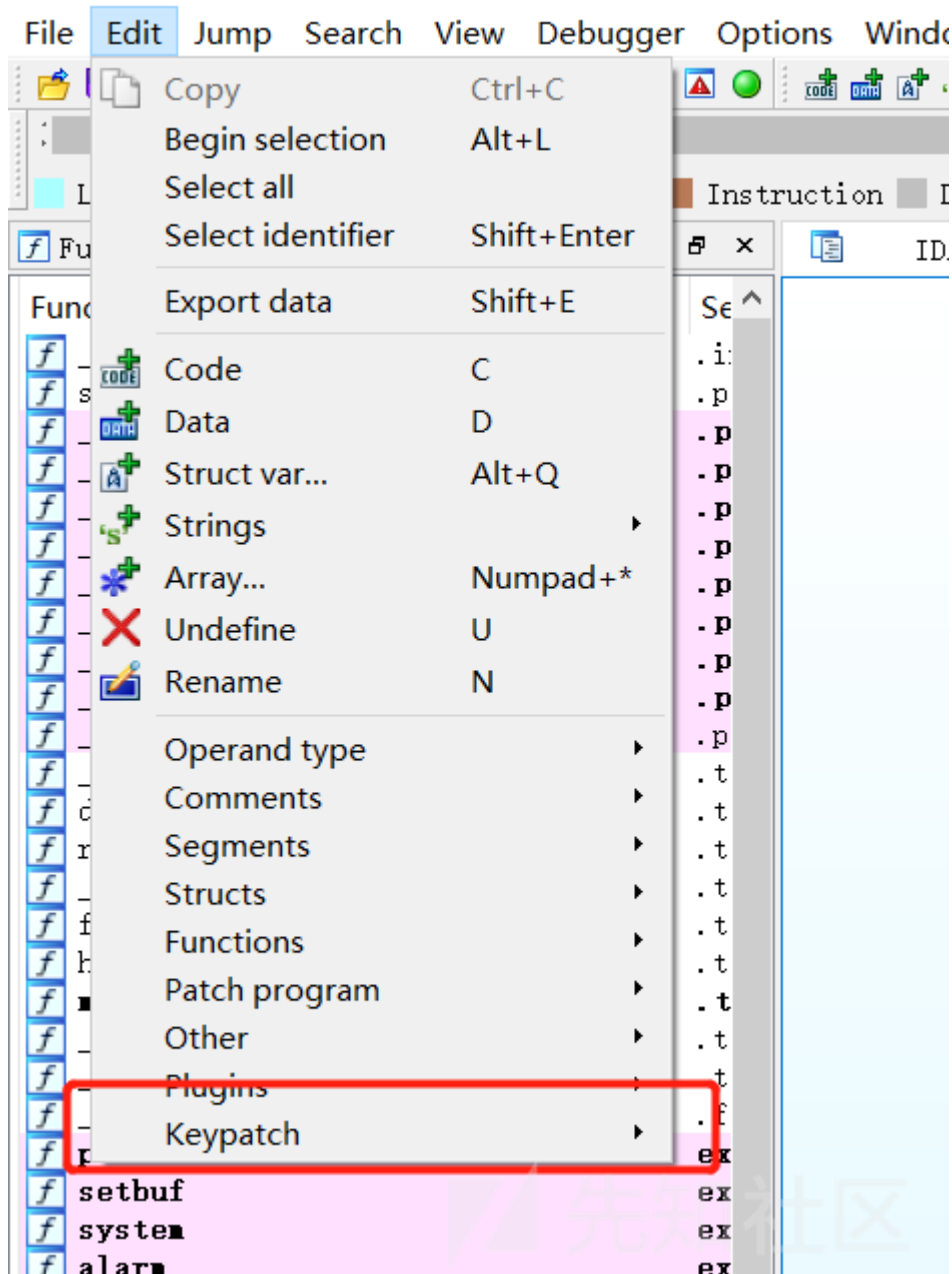
下载Keypatch.py复制到插件目录

IDA 7.0\plugins\Keypatch.py

下载安装keystone python模块，64位系统只需要安装这一个就行

<https://github.com/keystone-engine/keystone/releases/download/0.9.1/keystone-0.9.1-python-win64.msi>

安装好后，你就会发现这里有个keypatch的选项



修改程序指令

如果我们要修改程序本身的指令，怎么做呢

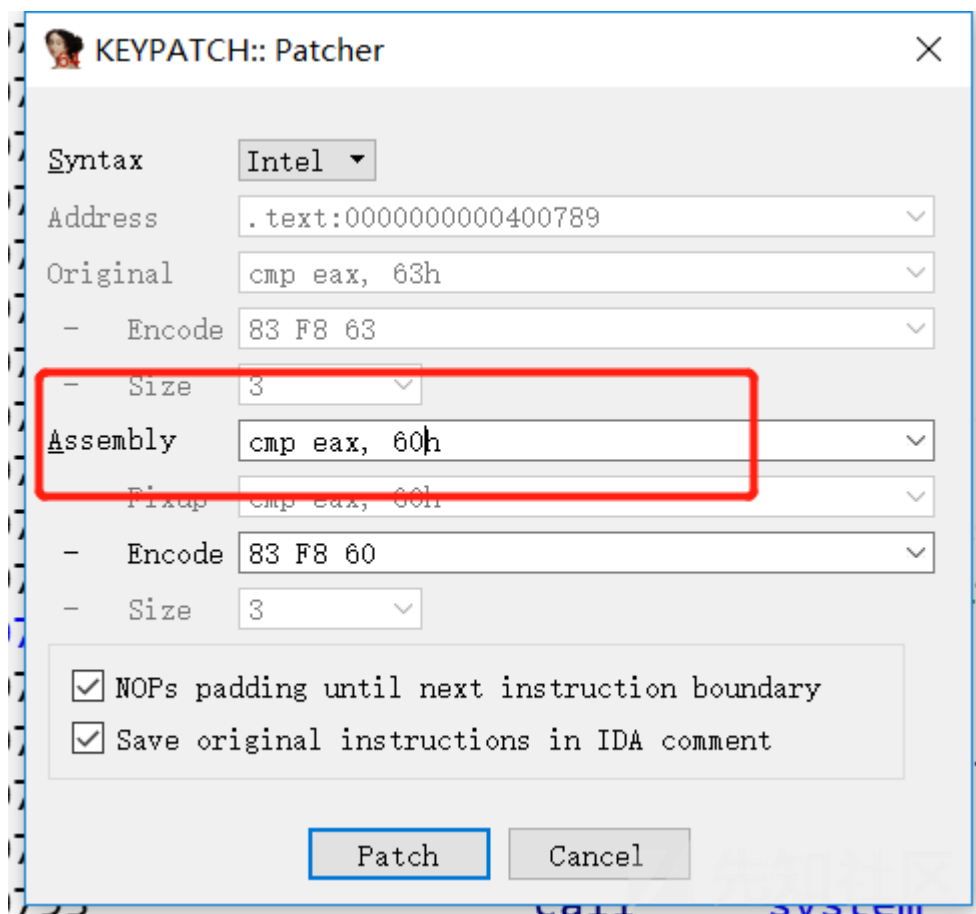
```

00000000400780
00000000400780 loc_400780:                                ; CODE XREF: main+46↑
00000000400780                                ; main+50↑j
00000000400780          mov     eax, [rbp+var_4]
00000000400783 ; 13:      if ( v3 > 99 )
00000000400783          lea     edx, [rax+1]      ; Load Effective Addr
00000000400786          mov     [rbp+var_4], edx
00000000400789          cmp     eax, 63h          ; Compare Two Operand
0000000040078C ; 14:      break;
0000000040078C          jle     short loc_40076C ; Jump if Less or Eq
0000000040078E ; 18:      system("/bin/sh");
0000000040078E          mov     edi, offset command ; "/bin/sh"
00000000400793          call    _system           ; Call Procedure

```

如图，我们要修改63h这个值

将鼠标指向改行，按快捷键Ctrl+Alt+K



直接输入汇编语句即可修改，打好patch后效果如图：

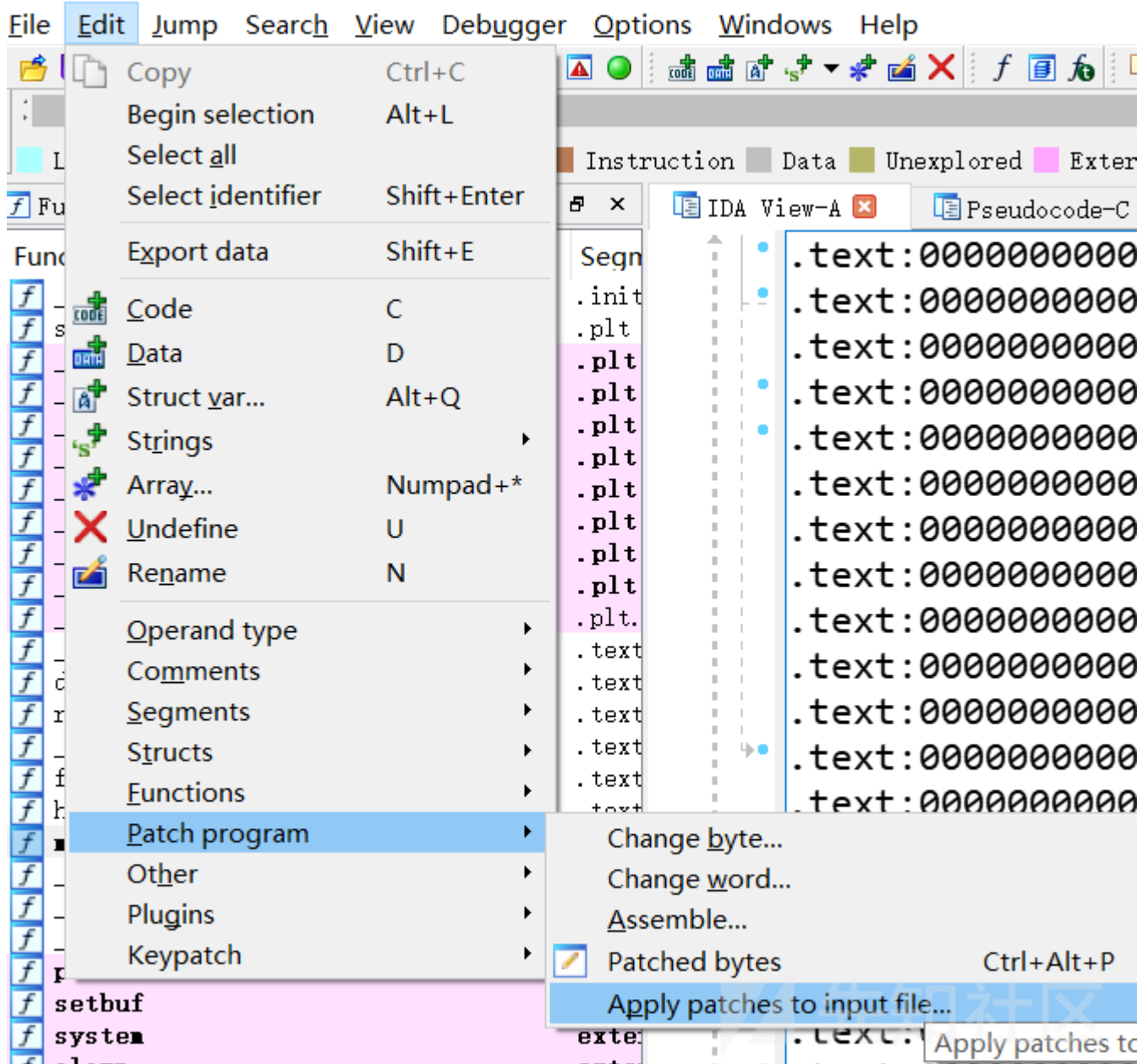
```

'80      ; main+50↑j
'80      mov     eax, [rbp+var_4]
'83 ; 13:      if ( v3 > 99 )
'83          lea     edx, [rax+1]      ; Load Effective Address
'86          mov     [rbp+var_4], edx
'89          cmp     eax, 60h          ; Keypatch modified this from:
'89                                     ; cmp eax, 63h
'8C ; 14:      break;
'8C          ile     short loc 40076C ; Jump if Less or Equal (ZF=1) S

```

这里会生成注释告诉你，这里打过patch，非常人性化

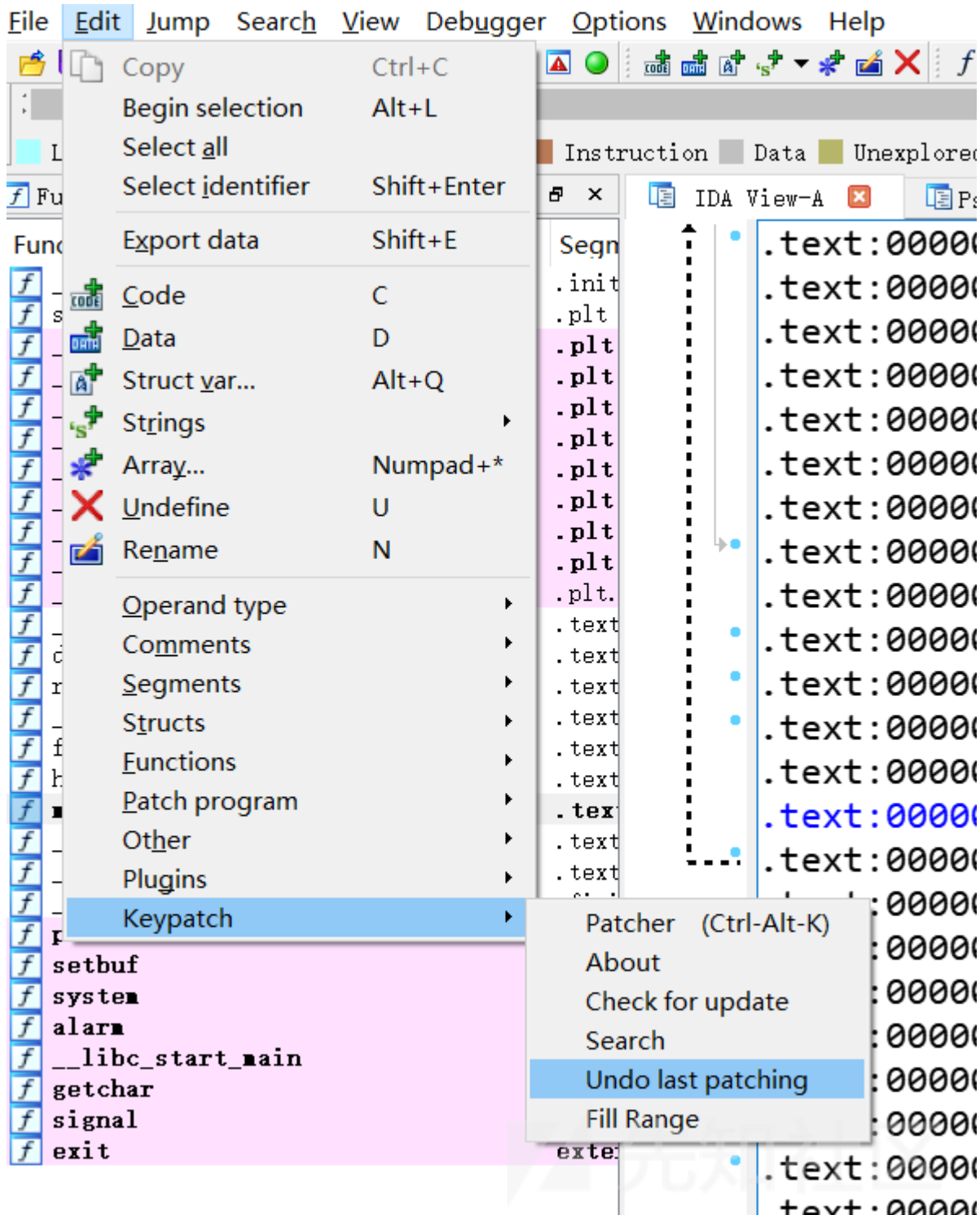
接着还要在菜单栏进行设置才能真正使得patch生效



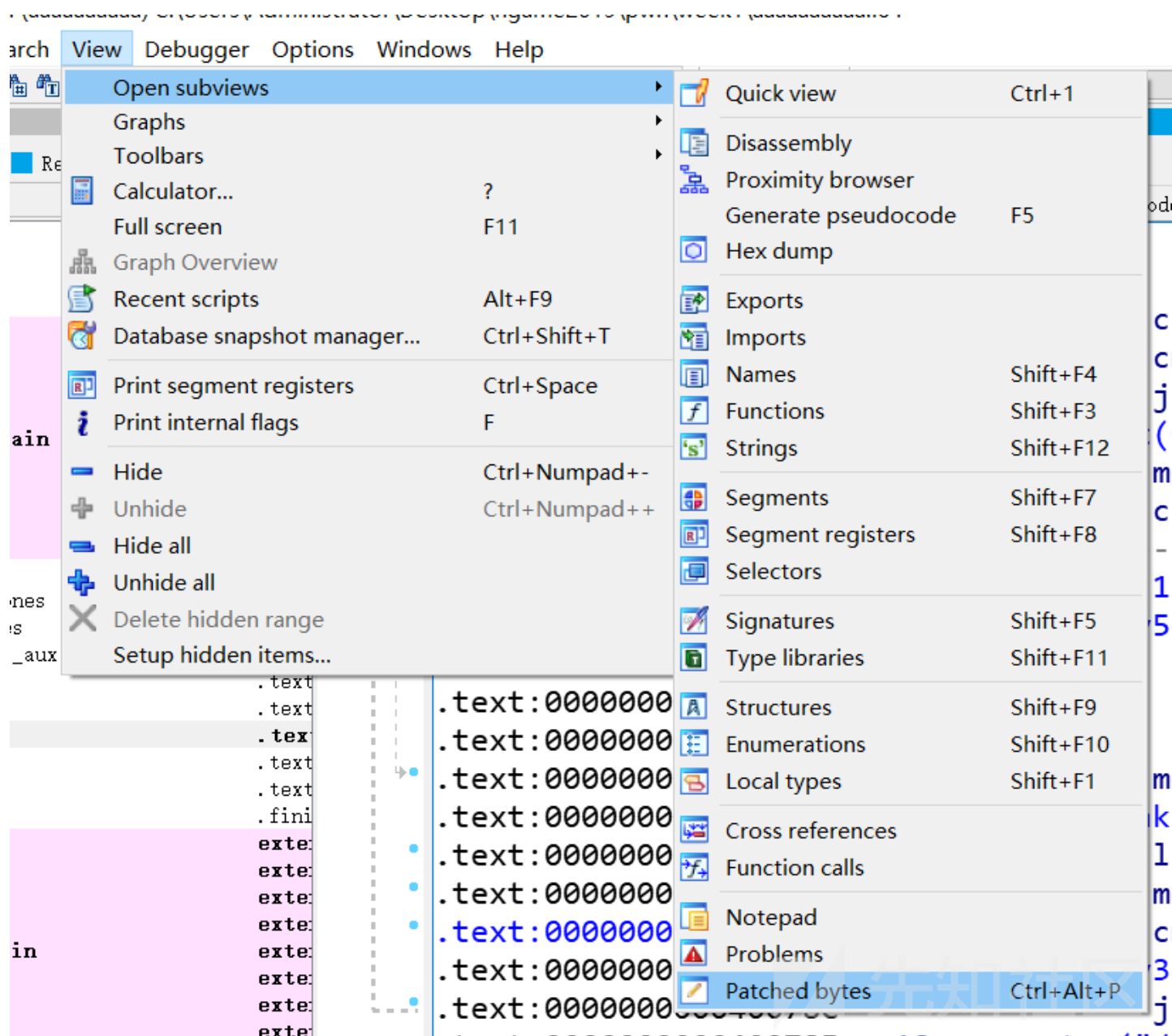
这样一来，原来的程序就已经被修改了

撤销patch

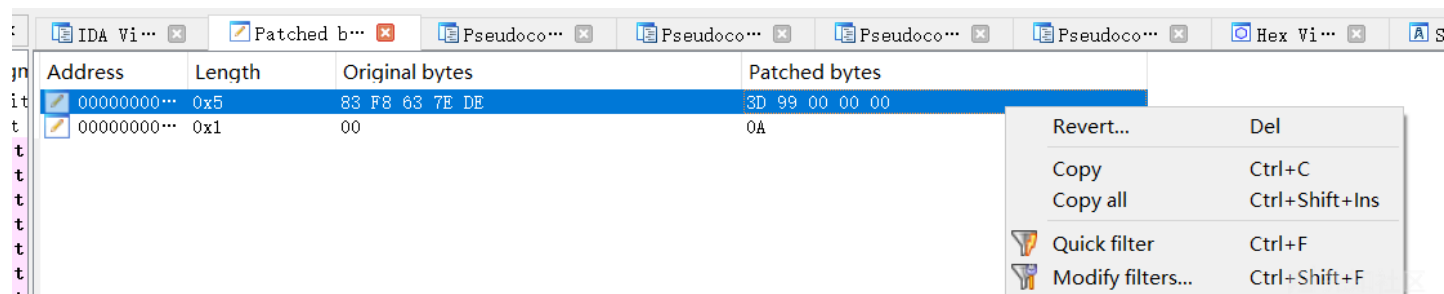
如果不小心打错了patch，就可以在这里进行撤销上一次patch的操作了



但是如果打了很多次patch，不好分清该撤销哪一次的patch，那么可以在菜单栏中打开patched bytes界面

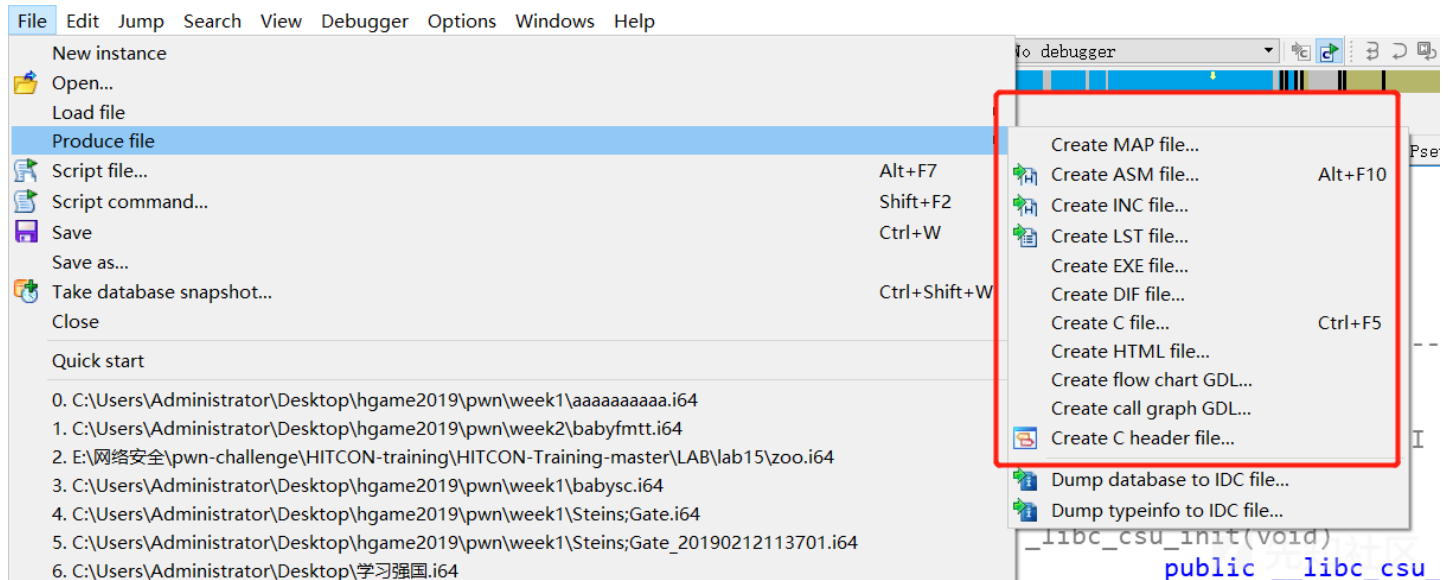


看到所有的patch，要撤销哪一个就右击选择 revert



IDA导出数据文件

在菜单栏中，这里有个选项可以生成各种不同的输出文件



这里简单的介绍前两个文件，后面的大家可以自己去生成测试一下用途，我这里就不详细介绍了

.map文件描述二进制文件的总体结构，包括与构成改二进制文件的节有关的信息，以及每个节中符号的位置。

.asm文件，也就是汇编了，直接能导出ida中反汇编的结果，这个非常实用，有的时候在逆向中经常遇到大量数据加解密的情况，如果在从IDA中一个个慢慢复制可就太没效率了

IDA常见命名意义

IDA经常会自动生成假名字。他们用于表示子函数，程序地址和数据。根据不同的类型和值假名字有不同前缀

sub 指令和子函数起点
locret 返回指令
loc 指令
off 数据，包含偏移量
seg 数据，包含段地址值
asc 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 结构体(或结构体数组)
algn 对齐指示
unk 未处理字节

IDA中有常见的说明符号，如db、dw、dd分别代表了1个字节、2个字节、4个字节

IDA反编译报错

目前来说，我遇到的反编译报错的情况，一般是两种

一是由于程序存在动态加密，导致程序的某些代码段被修改，从而反编译出错，这种情况，就需要去使用IDA-python解密一波，再进行F5反汇编

二是由于某些玄学问题，直接提示了某个地方出错，一般来说，就按照IDA的提示，去进行修改

比如，出现如下报错：



Warning



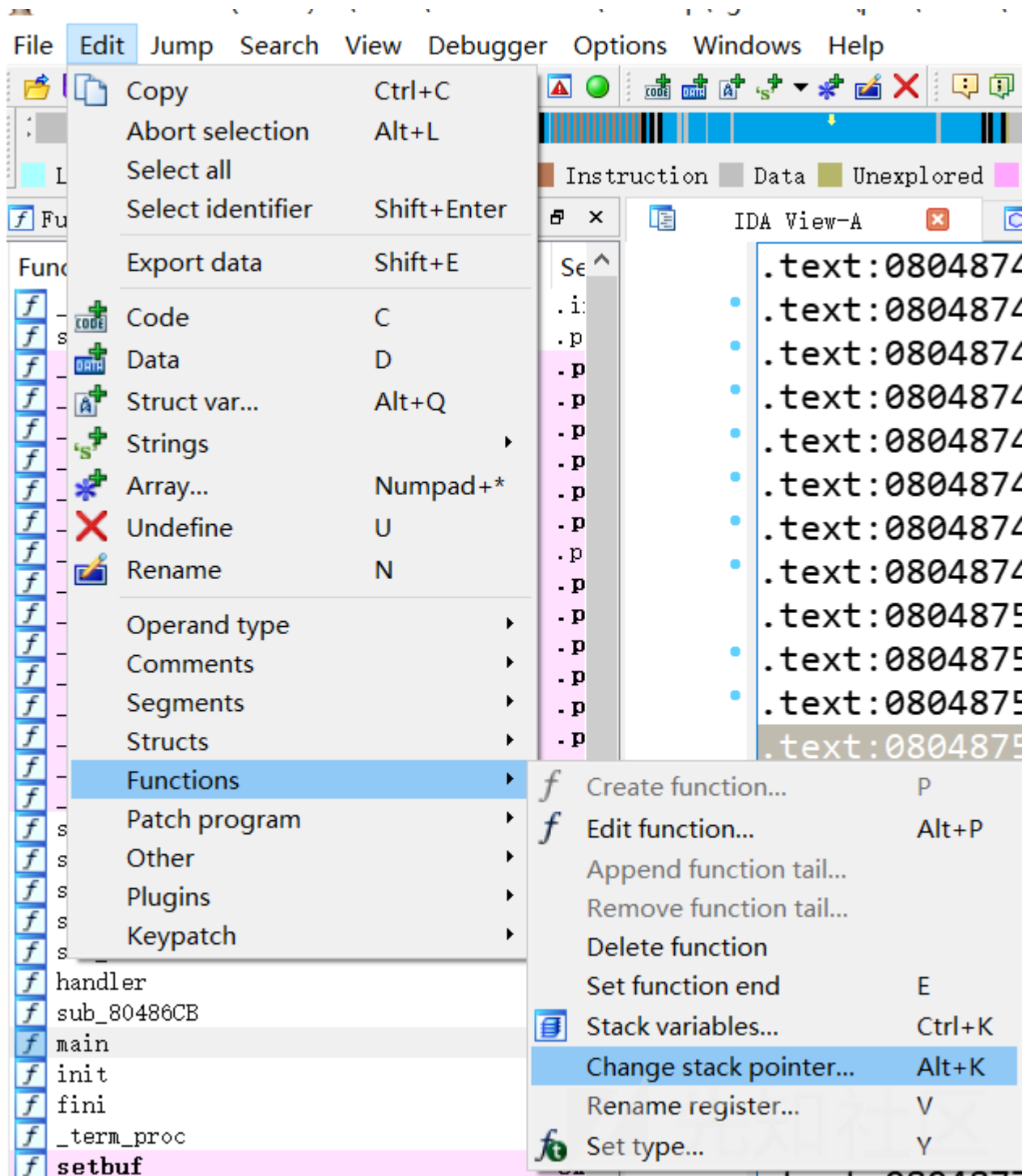
Decompilation failure:

413238: positive sp value has been found

Please refer to the manual to find appropriate actions

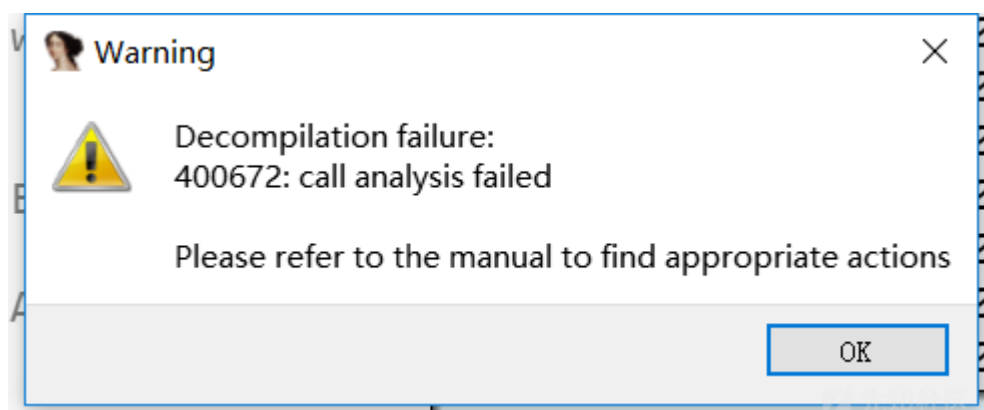
OK

那我们就去找413238这个地址的地方，提示是说sp指针的值没有被找到，说明是这里出错了，那么就去修改sp的值，修改方法如下：



也可以使用快捷键 Alt+K

有的时候，遇到的这种报错



就尝试着把报错的地址的汇编语句改一哈，改成nop，就可以解决问题

目前来说，我遇到报错的情况不多，一般都可以通过以上方法解决

配置IDA

在ida的根目录的cfg文件夹是专门用来存储配置文件的

ida的主配置文件为ida.cfg，另外的还有idagui.cfg，idatui.cfg这两个配置文件对应IDA的GUI配置和文本模式的版本

一、ida.cfg

该文件包含了option-->general中的所有选项的配置，可以通过选项中的描述在配置文件总找到相应的选项

这里举几个例子：

SHOW_AUTOCOMMENTS 表示是否自动生成汇编指令的注释

GRAPH_SHOW_LINEPREFIXES 表示是否在流程控制视图中显示地址

VPAGESIZE

表示内存调整参数，当处理非常大的输入文件时，IDA可能报告内存不足而无法创建新数据库，在这种情况下增大该参数，重新打开输入文件即可解决问题

OPCODE_BYTES 表示要显示的操作码字节数的默认值

INDENTATION 表示指令缩进的距离

NameChars 表示IDA支持的变量命令使用的字符集，默认是数字+字母还有几个特殊符号，如果需要添加就改变该参数

二、idagui.cfg

这个文件主要配置默认的GUI行为，键盘的快捷键等，这个很少需要修改，不做过多介绍。感兴趣的可以自己打开该文件观察，并不难懂，改改快捷键还是很容易的

三、idatui.cfg

这个似乎更加不常用。。。不多说了

需要注意的是，以上三个文件是默认配置，也就是说，每次打开创建新的ida数据库的时候，都会以这三个配置文件的设置进行创建，之前临时在菜单栏的设置就会消失，要

但，凡是都有例外，在option-->font和option-->colors这两个选项是全局选项，修改一次就永久生效的，不用在以上三个配置文件中改

最后

通过这一次系统地去学IDA，发现这个软件真的是非常厉害，我上面也只是简单地记录了平时比较常用的功能和操作，IDA还有很多高级的开发技巧，甚至你还能自定义模块

如果其他大佬还有别的IDA小技巧骚操作，可以留言交流一哈

点击收藏 | 6 关注 | 2

[上一篇：我如何发现这个能让数据库妥协的黑客链](#) [下一篇：安全研究：HTTP Smuggli...](#)

1. 1 条回复



[魔窟之舞](#) 2019-05-06 13:49:55

写的很好！

0 回复Ta

[登录](#) 后跟帖

先知社区

[现在登录](#)

热门节点

[技术文章](#)

[社区小黑板](#)

目录

[RSS](#) [关于社区](#) [友情链接](#) [社区小黑板](#)