# Happy to Reverse

@zjw

# 主要内容

- why ?
- what ?
- how ?

# 一些提示

- why happy?
- 最苦逼的工作
- 0和1的世界

- I am not script kid，you know 0x400000? PE? ELF?stack overflow?dword shoot?off-by-one?shellcode?payload?
- 装x

# 为什么逆向

- 破解（crack）
- 软件、app、iphone、TV .....
- @Geohot　ps3
- pwn2own VS ge

| | Target OS: Windows | |
|---|---|---|
| IDAPRONW | IDA Pro Named License [Windows] | 1129 USD |
| HEXARM64W | ARM64 Decompiler Fixed License [Windows] | 2350 USD |
| HEXARMW | ARM32 Decompiler Fixed License [Windows] | 2350 USD |
| HEXX64W | x64 Decompiler Fixed License [Windows] | 2350 USD |
| HEXX86W | x86 Decompiler Fixed License [Windows] | 2350 USD |
| UPDHEXARM64W | ARM64 Decompiler Fixed License [Windows] Support Renewal | 780 USD |
| UPDHEXARMW | ARM32 Decompiler Fixed Support Renewal [Windows] | 780 USD |
| UPDHEXX64W | x64 Decompiler Support Renewal [Windows] | 780 USD |
| UPDHEXX86W | x86 Decompiler Fixed Support Renewal [Windows] | 780 USD |
| UPDPRONW | IDA Pro Named Support Renewal [Windows] | 379 USD |

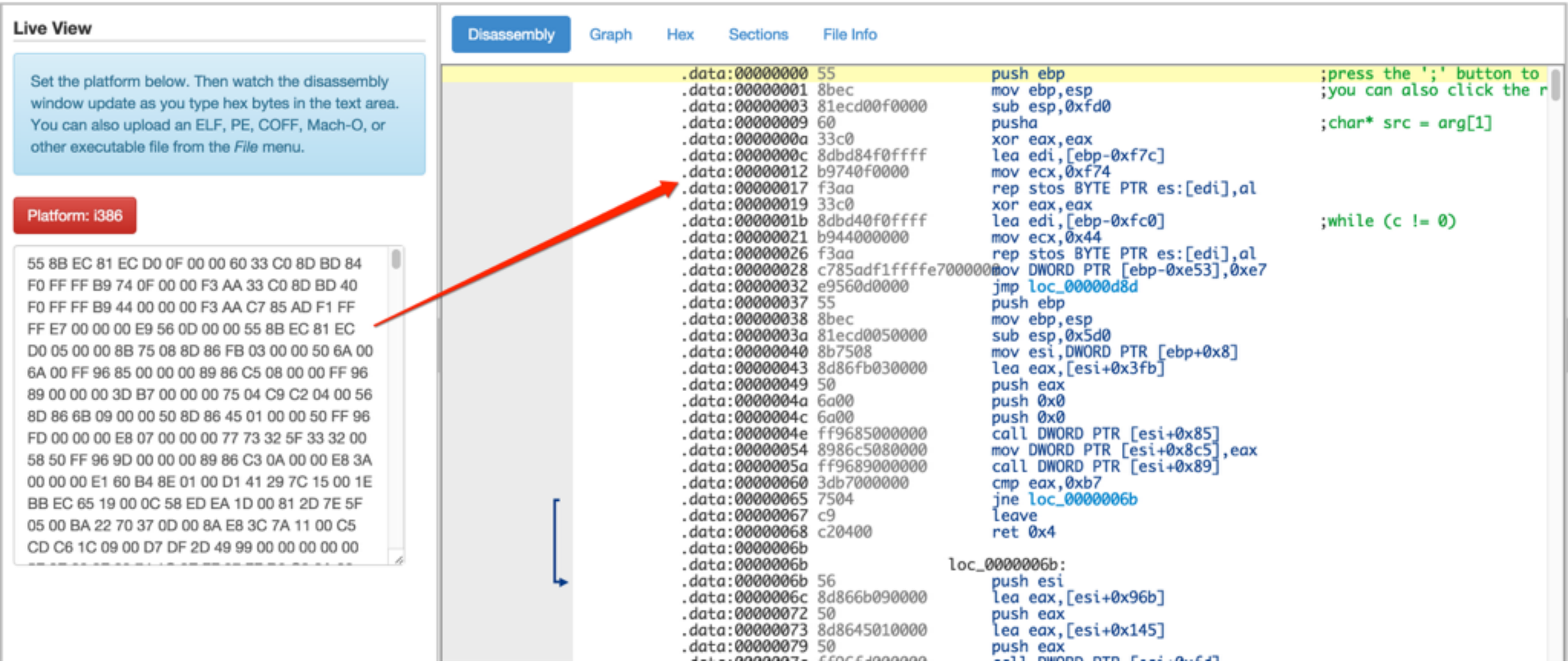
# 为什么逆向

- 对抗恶意软件
  - Stuxnet、Fanny（Equation Group）
  - duqu、flare
  - XcodeGhost
  - ……

# 为什么逆向

- 漏洞发现与利用
  - IE、chrome、firefox
  - flash
  - WormHole
  - ……

# 什么是逆向


Live View
Set the platform below. Then watch the disassembly window update as you type hex bytes in the text area. You can also upload an ELF, PE, COFF, Mach-O, or other executable file from the File menu.
Platform: i386
55 8B EC 81 EC D0 0F 00 00 60 33 C0 8D BD 84
F0 FF FF B9 74 0F 00 00 F3 AA 33 C0 8D BD 40
F0 FF FF B9 44 00 00 00 F3 AA C7 85 AD F1 FF
FF E7 00 00 00 E9 56 0D 00 00 55 8B EC 81 EC
D0 05 00 00 8B 75 08 8D 86 FB 03 00 00 50 6A 00
6A 00 FF 96 85 00 00 00 89 86 C5 08 00 00 FF 96
89 00 00 00 3D B7 00 00 00 75 04 C9 C2 04 00 56
8D 86 6B 09 00 00 50 8D 86 45 01 00 00 50 FF 96
FD 00 00 00 E8 07 00 00 00 77 73 32 5F 33 32 00
58 50 FF 96 9D 00 00 00 89 86 C3 0A 00 00 E8 3A
00 00 00 E1 60 B4 8E 01 00 D1 41 29 7C 15 00 1E
BB EC 65 19 00 0C 58 ED EA 1D 00 81 2D 7E 5F
05 00 BA 22 70 37 0D 00 8A E8 3C 7A 11 00 C5
CD C6 1C 09 00 D7 DF 2D 49 99 00 00 00 00 00
Disassembly
Graph
Hex
Sections
File Info
.data:00000000 55 push ebp ;press the ';' button to
.data:00000001 8bec mov ebp,esp ;you can also click the r
.data:00000003 81ecd00f0000 sub esp,0xfd0
.data:00000009 60 pusha ;char* src = arg[1]
.data:0000000a 33c0 xor eax,eax
.data:0000000c 8dbd84f0ffff lea edi,[ebp-0xf7c]
.data:00000012 b9740f0000 mov ecx,0xf74
.data:00000017 f3aa rep stos BYTE PTR es:[edi],al
.data:00000019 33c0 xor eax,eax
.data:0000001b 8dbd40f0ffff lea edi,[ebp-0xfc0] ;while (c != 0)
.data:00000021 b944000000 mov ecx,0x44
.data:00000026 f3aa rep stos BYTE PTR es:[edi],al
.data:00000028 c785adf1ffffe700000 mov DWORD PTR [ebp-0xe53],0xe7
.data:00000032 e9560d0000 jmp loc_00000d8d
.data:00000037 55 push ebp
.data:00000038 8bec mov ebp,esp
.data:0000003a 81ecd0050000 sub esp,0x5d0
.data:00000040 8b7508 mov esi,DWORD PTR [ebp+0x8]
.data:00000043 8d86fb030000 lea eax,[esi+0x3fb]
.data:00000049 50 push eax
.data:0000004a 6a00 push 0x0
.data:0000004c 6a00 push 0x0
.data:0000004e ff9685000000 call DWORD PTR [esi+0x85]
.data:00000054 8986c5080000 mov DWORD PTR [esi+0x8c5],eax
.data:0000005a ff9689000000 call DWORD PTR [esi+0x89]
.data:00000060 3db7000000 cmp eax,0xb7
.data:00000065 7504 jne loc_0000006b
.data:00000067 c9 leave
.data:00000068 c20400 ret 0x4
.data:0000006b
.data:0000006b loc_0000006b:
.data:0000006b 56 push esi
.data:0000006c 8d866b090000 lea eax,[esi+0x96b]
.data:00000072 50 push eax
.data:00000073 8d8645010000 lea eax,[esi+0x145]
.data:00000079 50 push eax

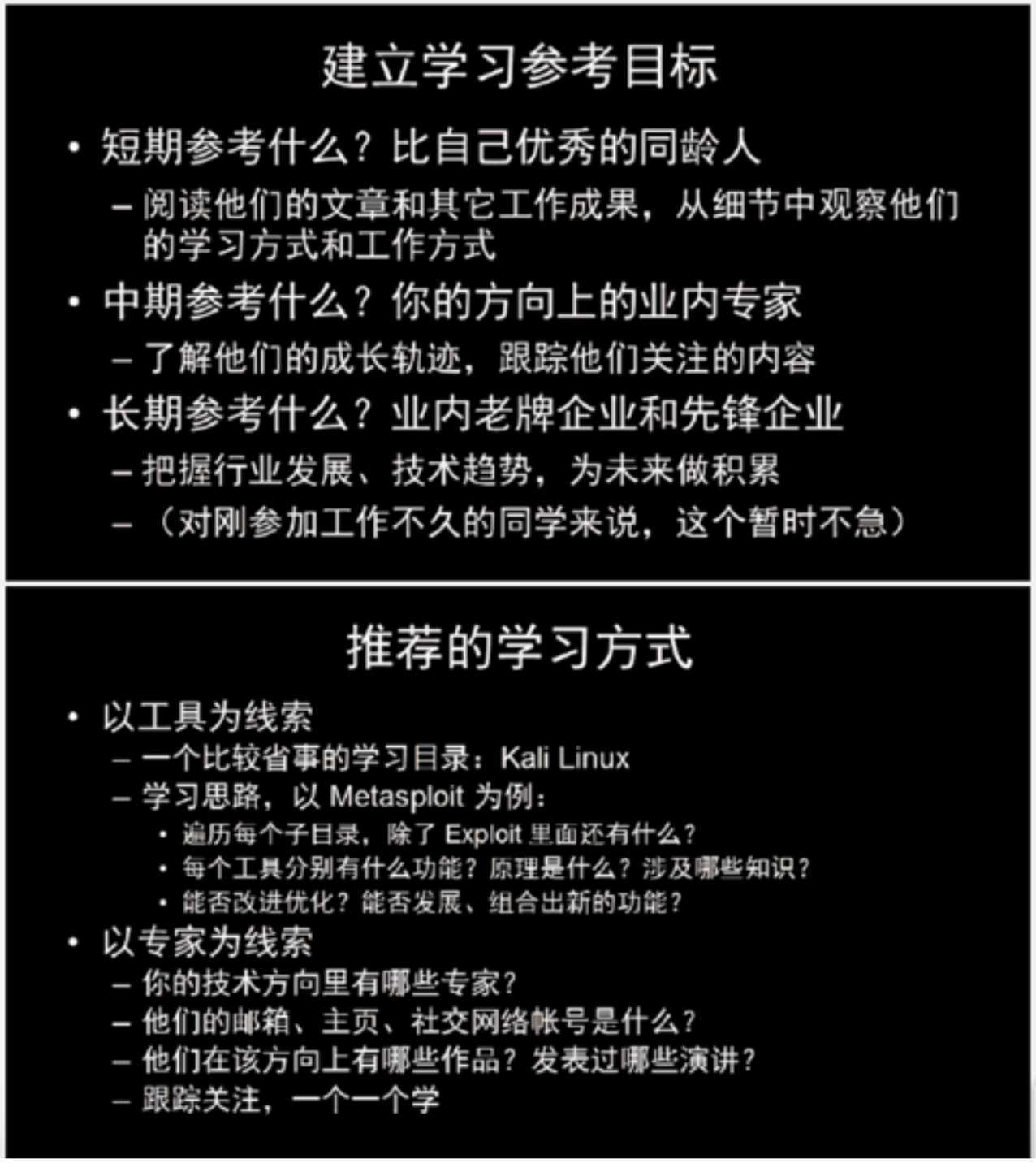
# 怎么学

- 基础知识
  - c语言
  - 汇编语言
- 工具
  - ida、immunity、gdb、od
- 基本编程
  - python是世界上最美的语言
  - c是我心中的女神
  - 福利：http://mp.weixin.qq.com/s?__biz=MjM5NTY1NDI1Mg==&mid=200789388&idx=3&sn=6d7c2f61f69a47c08a0bfdc73e7d8be2


@tombkeeper

写了一个关于技术人员个人成长的 PPT，分享其中三页。

收起 | 查看大图 | 向左旋转 | 向右旋转



# 逆向对象

- windows
  - .exe .dll
    - od、immunity、ida、windbg
- .exe(.net语言)
  - net reflector、ILSpy、de4dot

# 逆向对象

- linux android(java and native code)
  - .elf .so
    - ida
- .apk .jar .dex .odex
- apktool、smali、baksmali、dex2jar、jad、jd-gui、jeb

# 基本词汇

- Instruction：CPU的原指令，例如：将数据在数据区与寄存器之间进行转移操作，对数据进行操作，算术操作。原则上每种CPU会有自己独特的一套指令构架(Instruction Set Architecture(ISA))。

- Machine code: CPU的指令码（机器码），每条指令都会被译成指令码。

- Assembly Language: 汇编语言，助记码和其他一些例如宏那样的特性组成的便于程序员编写的语言。

- CPU register：CPU寄存器，每个CPU都有一些通用寄存器(General Purpose Registers(GPR))。X86有8个，x86-64(amd64)有16个，ARM有16个，最简单去理解寄存器的方法就是，把寄存器想成一个不需要类型的临时变量。想象你在用高级编程语言，并且只有8个32bit的变量。只用这些可以完成非常多的事情。

- from：

    - http://yurichev.com/writings/RE_for_beginners-en.pdf

    - http://drops.wooyun.org/tips/1517

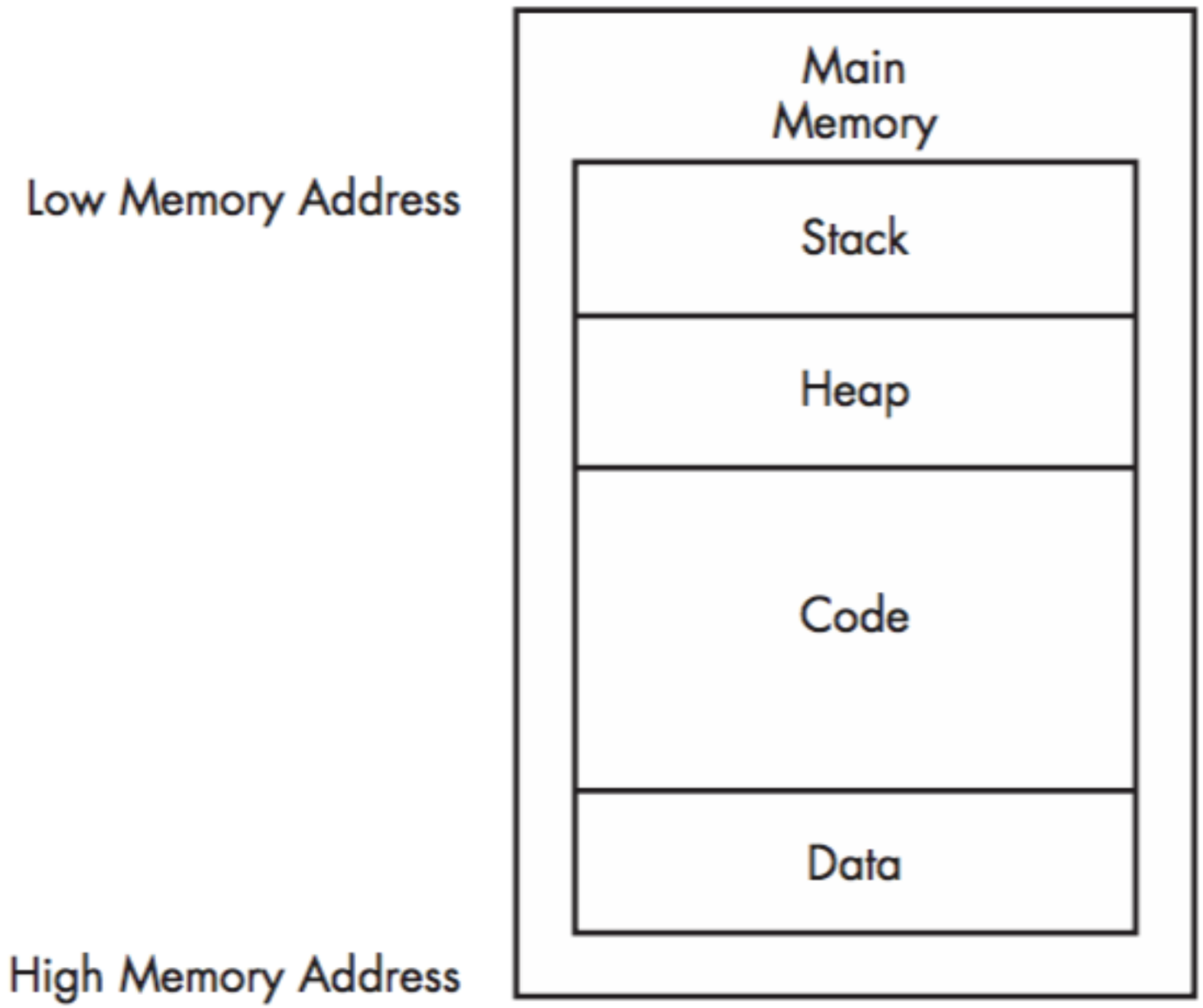
# Hello, world!

```
#include <stdio.h>
int main()
{
  printf("hello, world");
  return 0;
};
```

```
main proc near
var_10 = dword ptr -10h
push ebp
mov ebp, esp
and esp, 0FFFFFFF0h
sub esp, 10h
mov eax, offset aHelloWorld ; "hello, world"
mov [esp+10h+var_10], eax
call _printf
mov eax, 0
leave
retn
main endp
```

```
.text:00000000 main
.text:00000000 10 40 2D E9        STMFD SP!, {R4,LR}
.text:00000004 1E 0E 8F E2        ADR R0, aHelloWorld ; "hello, world"
.text:00000008 15 19 00 EB        BL __2printf
.text:0000000C 00 00 A0 E3        MOV R0, #0
.text:00000010 10 80 BD E8        LDMFD SP!, {R4,PC}
.text:000001EC 68 65 6C 6C+       aHelloWorld DCB "hello, world",0 ; DATA XREF: main+4
```

# 内存地址

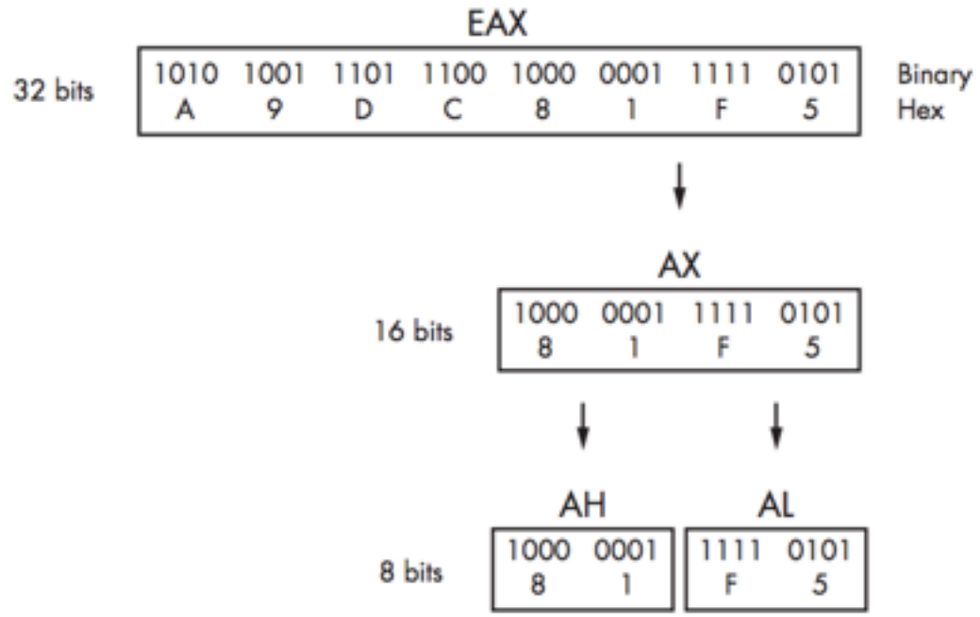

Low Memory Address
Main
Memory
Stack
Heap
Code
Data
High Memory Address

# 寄存器

General registers

EAX (AX, AH, AL) EBX (BX, BH, BL) ECX (CX, CH, CL) EDX (DX, DH, DL) EBP (BP)

ESP (SP) ESI (SI)



# 简单指令

| | |
|---|---|
| mov eax, ebx<br>#eax=ebx | Copies the contents of EBX into the EAX register |
| mov eax, 0x42<br>#eax=0x42 | Copies the value 0x42 into the EAX register |
| mov eax, [0x4037C4]<br>#eax=*(0x4037c4) | Copies the 4 bytes at the memory location 0x4037C4 into the EAX register |
| mov eax, [ebx]<br>#eax=*(ebx) | Copies the 4 bytes at the memory location specified by the EBX register into the EAX register |
| mov eax, [ebx+esi*4]<br>#eax=*(ebx+esi+4) | Copies the 4 bytes at the memory location specified by the result of the equation ebx+esi*4 into the EAX register |

# 算术指令1

| | |
|---|---|
| sub eax, 0x10<br>#eax=eax-0x10 | Subtracts 0x10 from EAX |
| add eax, ebx<br>#eax=eax+ebx | Adds EBX to EAX and stores the result in EAX |
| inc edx<br>#edx=edx+1 | Increments EDX by 1 |
| dec ecx<br>#ecx=ecx-1 | Decrements ECX by 1 |
| mul 0x50<br>#[edx:eax]=eax*0x50 | Multiplies EAX by 0x50 and stores the result in EDX:EAX |
| div 0x75<br>#eax,edx=[edx:eax]/0x75 | Divides EDX:EAX by 0x75 and stores the result in EAX and the remainder in EDX |

# 算术指令2

| **xor eax, eax** | Clears the EAX register |
| --- | --- |
| or eax, 0x7575 | Performs the logical or operation on EAX with 0x7575 |
| mov eax, 0xA<br>shl eax, 2 | EAX = 0x28, because 1010 (0xA in binary) shifted 2 bits left is 101000 (0x28) |
| mov bl, 0xA<br>ror bl, 2 | Rotates the BL register to the right 2 bits; these two instructions result in BL = 10000010, because 1010 rotated 2 bits right is 10000010 |

# 跳转指令

| Instruction | Description |
|---|---|
| `jz loc` | Jump to specified location if ZF = 1. |
| `jnz loc` | Jump to specified location if ZF = 0. |
| `je loc` | Same as `jz`, but commonly used after a `cmp` instruction. Jump will occur if the destination operand equals the source operand. |
| `jne loc` | Same as `jnz`, but commonly used after a `cmp`. Jump will occur if the destination operand is not equal to the source operand. |
| `jg loc` | Performs signed comparison jump after a `cmp` if the destination operand is greater than the source operand. |
| `jge loc` | Performs signed comparison jump after a `cmp` if the destination operand is greater than or equal to the source operand. |
| `ja loc` | Same as `jg`, but an unsigned comparison is performed. |
| `jae loc` | Same as `jge`, but an unsigned comparison is performed. |
| `jl loc` | Performs signed comparison jump after a `cmp` if the destination operand is less than the source operand. |
| `jle loc` | Performs signed comparison jump after a `cmp` if the destination operand is less than or equal to the source operand. |
| `jb loc` | Same as `jl`, but an unsigned comparison is performed. |
| `jbe loc` | Same as `jle`, but an unsigned comparison is performed. |
| `jo loc` | Jump if the previous instruction set the overflow flag (OF = 1). |
| `js loc` | Jump if the sign flag is set (SF = 1). |
| `jecxz loc` | Jump to location if ECX = 0. |

# 重复指令

| | |
|---|---|
| rep | Repeat until ECX = 0 |
| repe, repz | Repeat until ECX = 0 or ZF = 0 |
| repne, repnz | Repeat until ECX = 0 or ZF = 1 |

# 栈

- 函数调用，变量（local）存储
- LIFO(last in,first out)
- ESP => top of stack
- EBP => base pointer,track local var and param
- push、pop、call、leave、enter、ret

# 函数调用

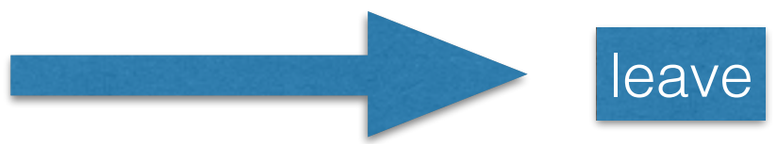
- push压入参数（一般从右向左）
- 调用call，将返回地址（call下一条的地址）压入
- 函数prologue

```
push    ebp

mov     ebp, esp

sub     esp, X
```

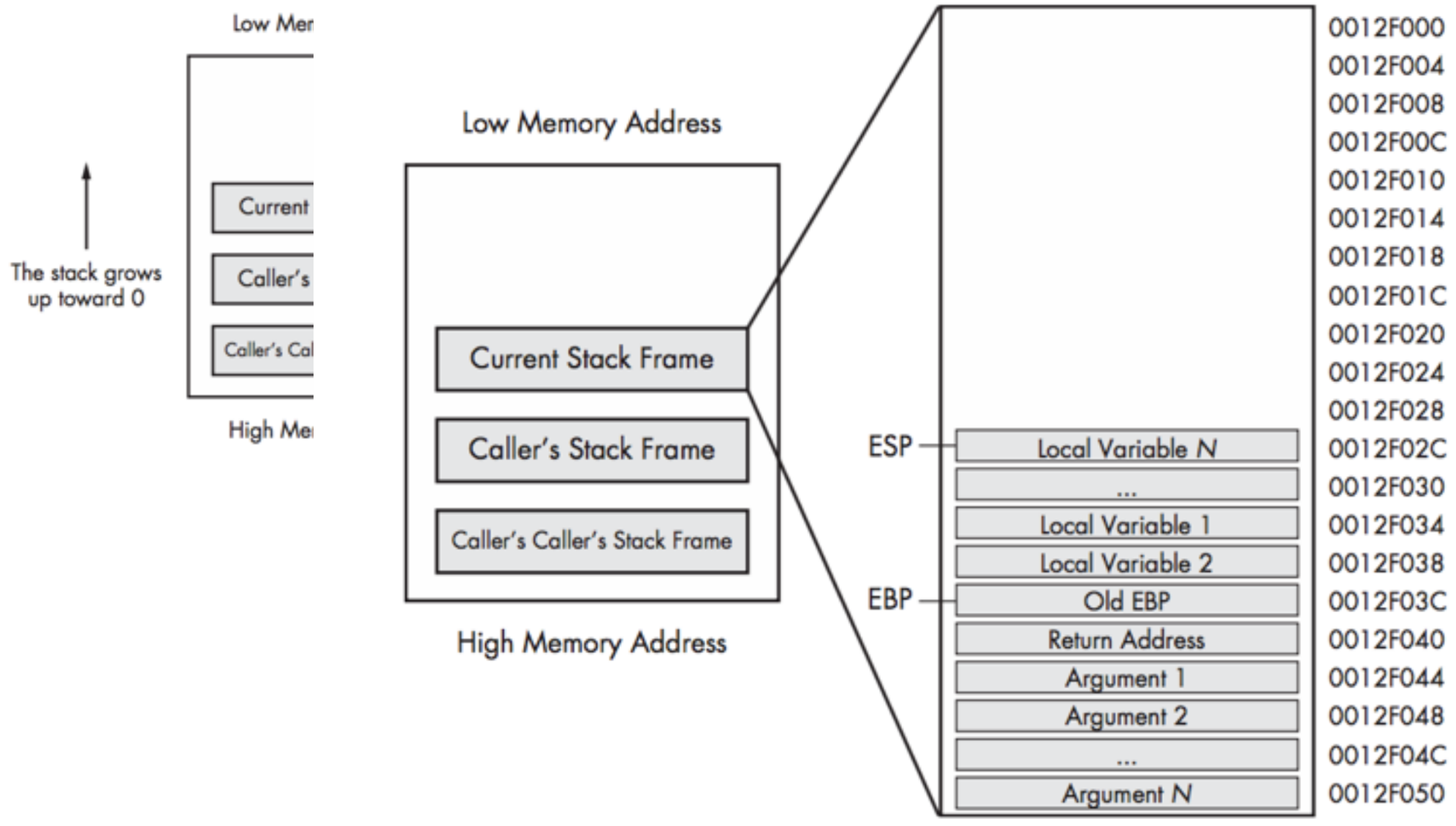
# 函数调用

- 函数体
- 函数epilogue

```
mov     esp, ebp
pop     ebp
ret
```



# 栈的布局


Low Mer
The stack grows
up toward 0
Current
Caller's
Caller's Cal
High Me
Low Memory Address
Current Stack Frame
Caller's Stack Frame
Caller's Caller's Stack Frame
High Memory Address
ESP
EBP
Local Variable N
...
Local Variable 1
Local Variable 2
Old EBP
Return Address
Argument 1
Argument 2
...
Argument N
0012F000
0012F004
0012F008
0012F00C
0012F010
0012F014
0012F018
0012F01C
0012F020
0012F024
0012F028
0012F02C
0012F030
0012F034
0012F038
0012F03C
0012F040
0012F044
0012F048
0012F04C
0012F050

# 举个例子

push arg3

push arg2

push arg1

call function

add esp, 4*3


| ... | ... |
|---|---|
| ESP-0xC | local variable #2, marked in IDA as var_8 |
| ESP-8 | local variable #1, marked in IDA as var_4 |
| ESP-4 | saved value of EBP |
| ESP | return address |
| ESP+4 | argument#1, marked in IDA as arg_0 |
| ESP+8 | argument#2, marked in IDA as arg_4 |
| ESP+0xC | argument#3, marked in IDA as arg_8 |
| ... | ... |

how about ebp?

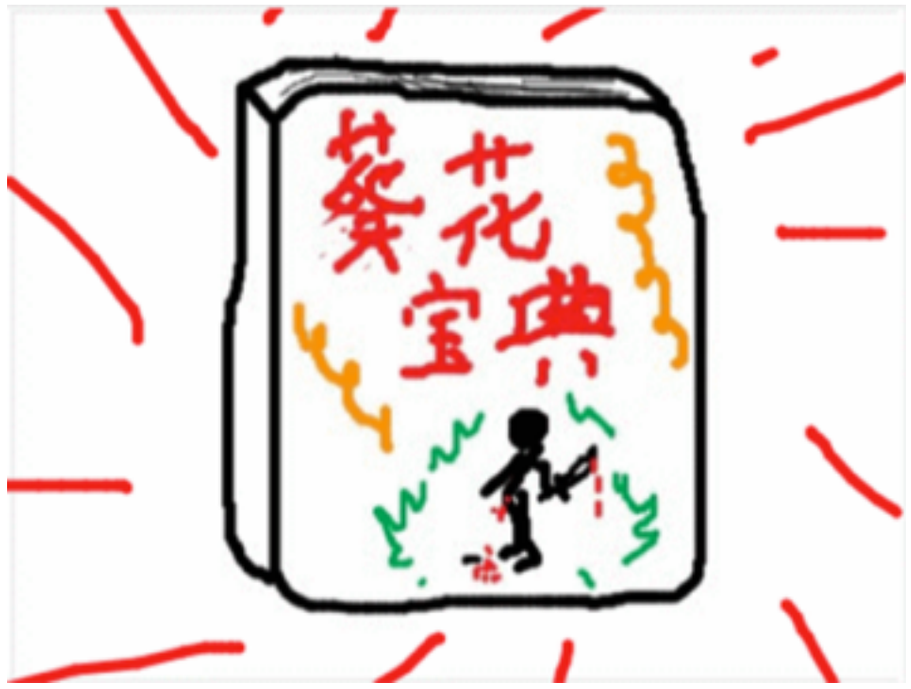
# 找你妹

| | |
|---|---|
| ... | ... |
| ESP-0xC | local variable #2, marked in IDA as var_8 |
| ESP-8 | local variable #1, marked in IDA as var_4 |
| ESP-4 | saved value of EBP |
| ESP | return address |
| ESP+4 | argument#1, marked in IDA as arg_0 |
| ESP+8 | argument#2, marked in IDA as arg_4 |
| ESP+0xC | argument#3, marked in IDA as arg_8 |
| ... | ... |

| | |
|---|---|
| | ... |
| EBP-8 | local variable #2, marked in IDA as var_8 |
| EBP-4 | local variable #1, marked in IDA as var_4 |
| EBP | saved value of EBP |
| EBP+4 | return address |
| EBP+8 | argument#1, marked in IDA as arg_0 |
| EBP+0xC | argument#2, marked in IDA as arg_4 |
| EBP+0x10 | argument#3, marked in IDA as arg_8 |
| ... | ... |

# 小提示

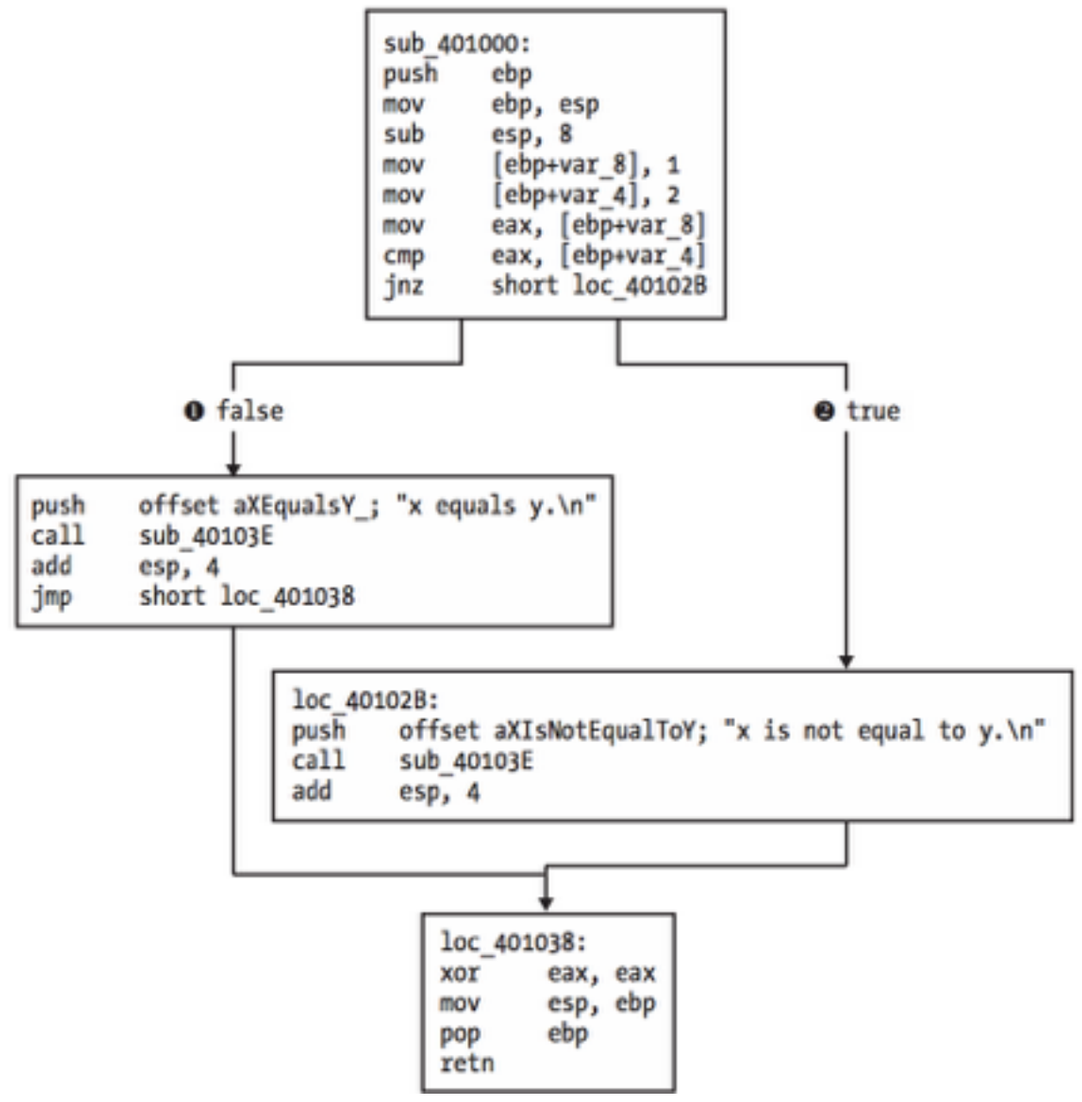
- 如何打好基础，修炼神功?
  - “要练此功，必先自虐”
  - 装个c/c++的编译器




- 针对各种程序流程写个demo，然后反汇编

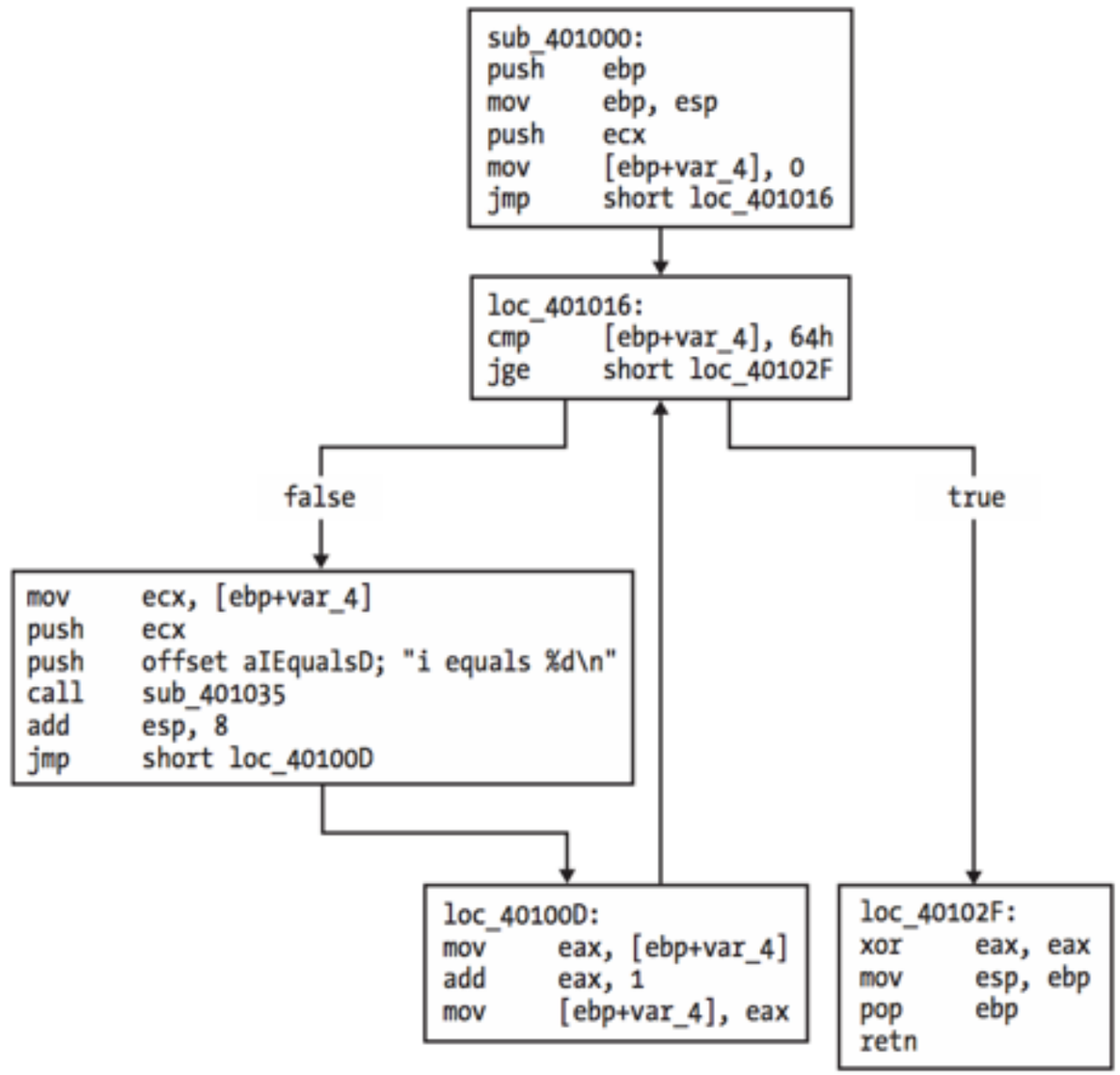
# if

```
int x = 0;
int y = 1;
int z = 2;

if(x == y){
     if(z==0){
          printf("z is zero and x = y.\n");
     }else{
          printf("z is non-zero and x = y.\n");
     }
}else{
     if(z==0){
          printf("z zero and x != y.\n");
     }else{
          printf("z non-zero and x != y.\n");
     }
}
```



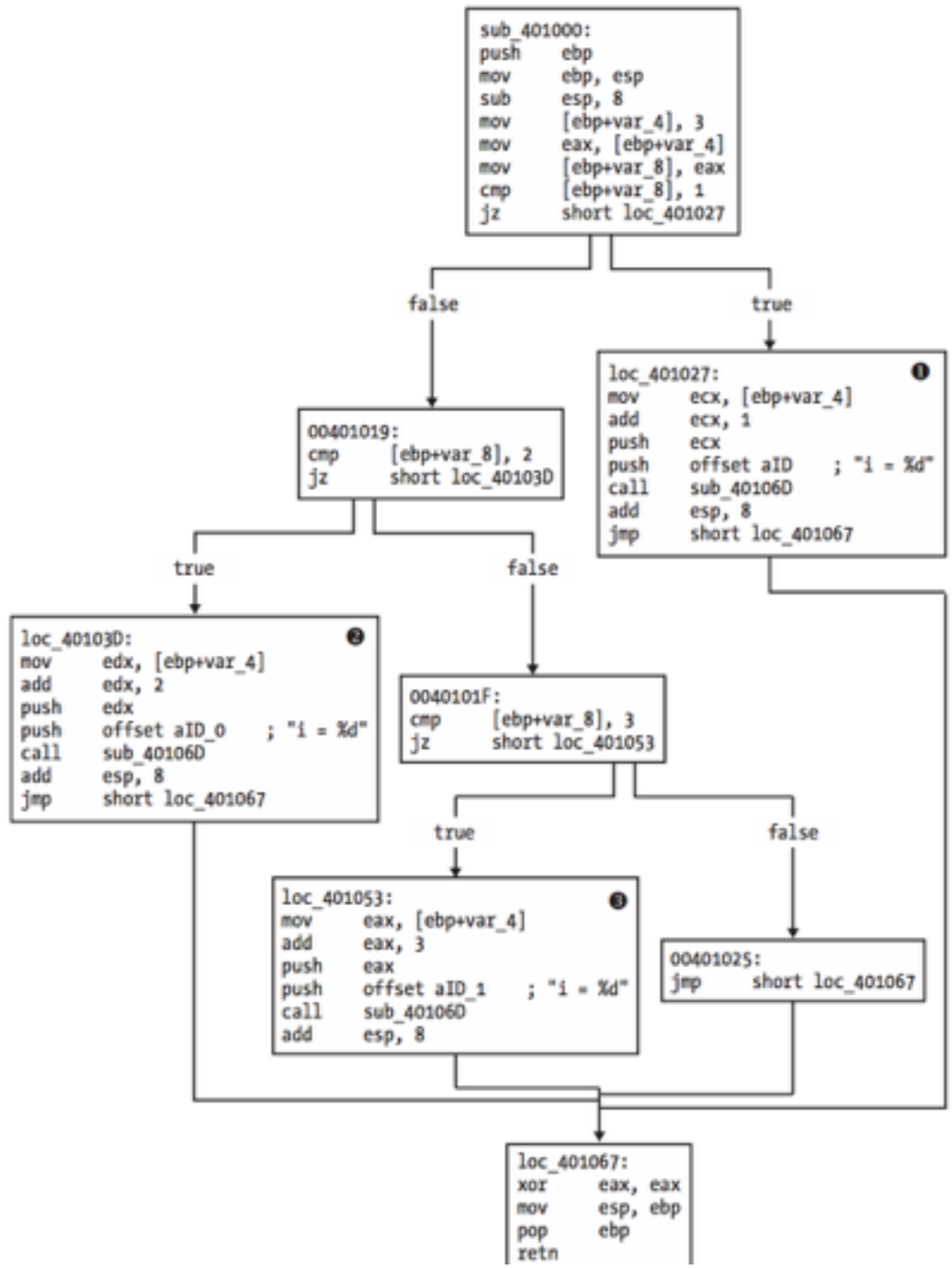
# loops

```
int i;

for(i=0; i<100; i++)
{
   printf("i equals %d\n", i);
}
```



# jump table

```
switch(i)
{
   case 1:
      printf("i = %d", i+1);
      break;
   case 2:
      printf("i = %d", i+2);
      break;
   case 3:
      printf("i = %d", i+3);
      break;
   case 4:
      printf("i = %d", i+3);
      break;
   default:
      break;
}
```



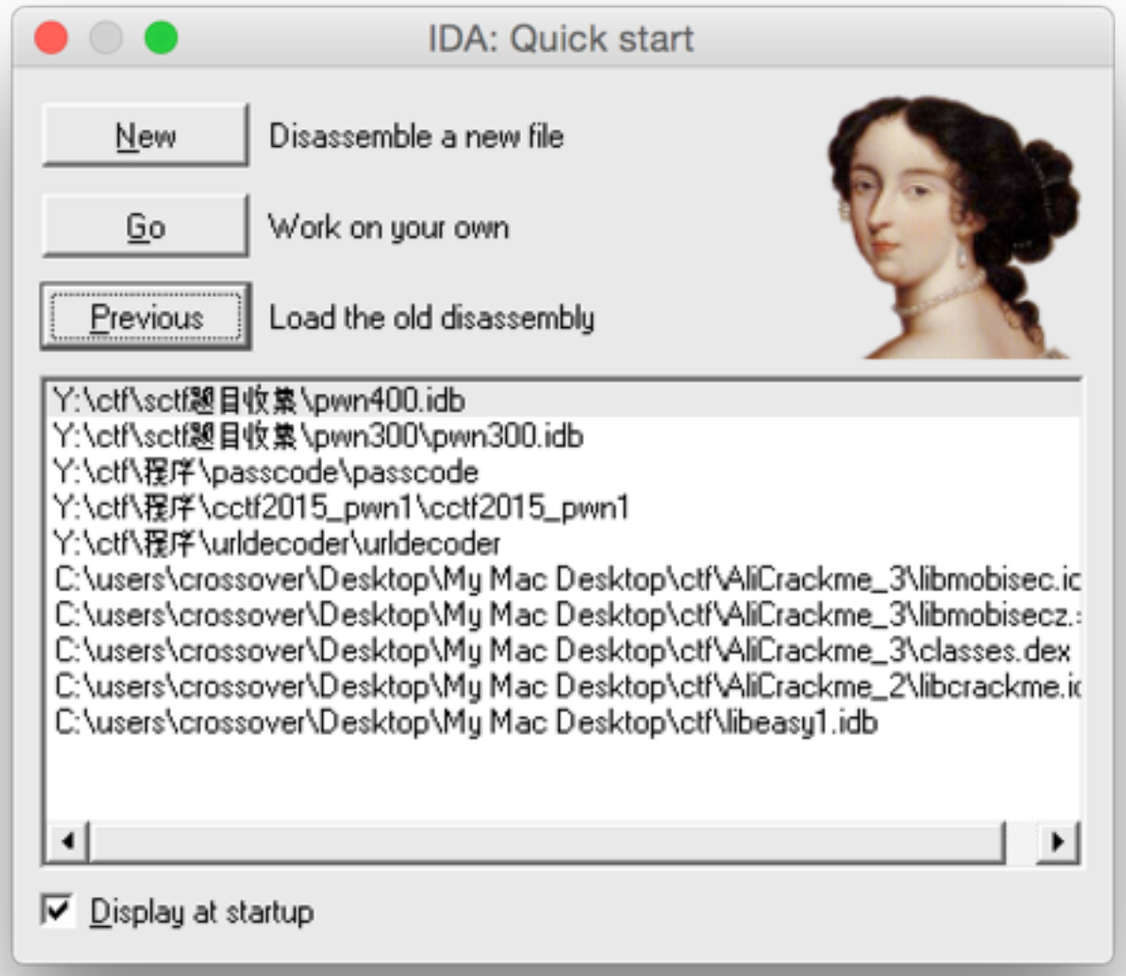
# 神器1.IDA

- 收费 => 贵
- 总部位于比利时列日市
- 天才：Ilfak Guilfanov
- 32位
- 支持50多种处理器
- 最新6.9
- 炫富

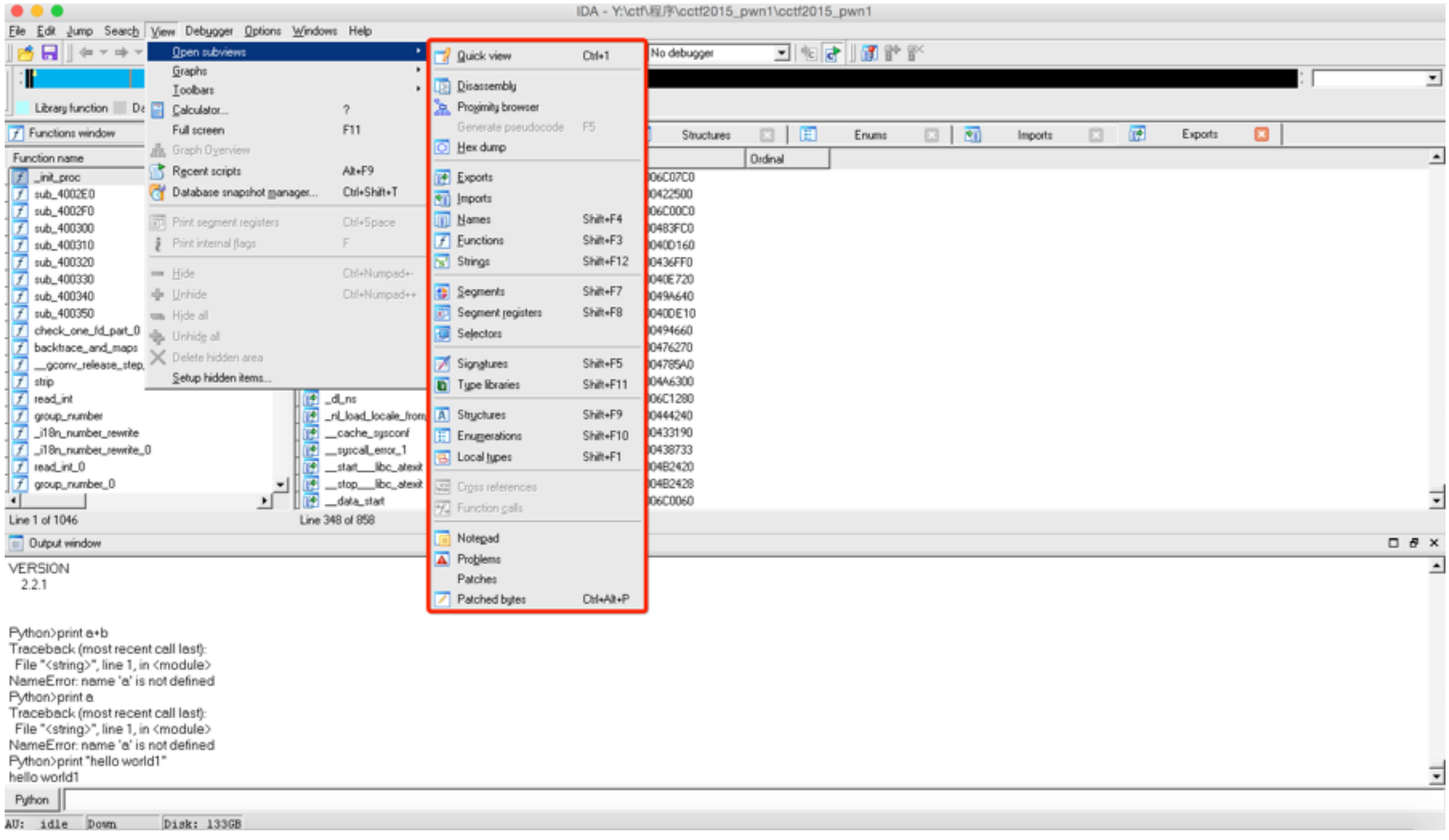
# tips

- 本质是数据库
- 快捷键：空格、x(xrefs)、enter、esc
- 重要的事情说三遍：无“撤销”、无“撤销”、无“撤销”

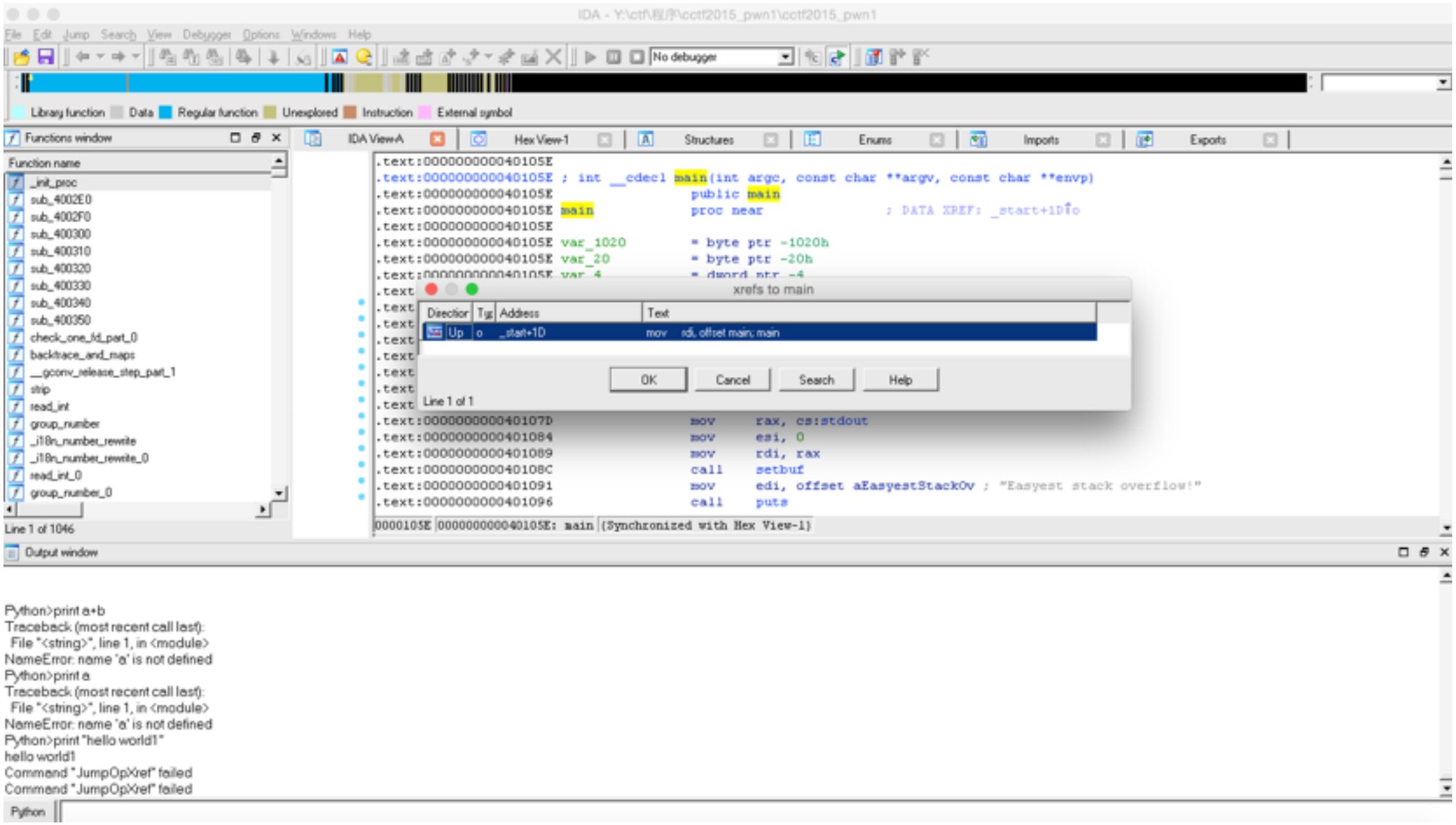
# start


IDA: Quick start
New
Disassemble a new file
Go
Work on your own
Previous
Load the old disassembly
Y:\ctf\sctf题目收集\pwn400.idb
Y:\ctf\sctf题目收集\pwn300\pwn300.idb
Y:\ctf\程序\passcode\passcode
Y:\ctf\程序\cctf2015_pwn1\cctf2015_pwn1
Y:\ctf\程序\urldecoder\urldecoder
C:\users\crossover\Desktop\My Mac Desktop\ctf\AliCrackme_3\libmobisec.ic
C:\users\crossover\Desktop\My Mac Desktop\ctf\AliCrackme_3\libmobisecz.
C:\users\crossover\Desktop\My Mac Desktop\ctf\AliCrackme_3\classes.dex
C:\users\crossover\Desktop\My Mac Desktop\ctf\AliCrackme_2\libcrackme.ic
C:\users\crossover\Desktop\My Mac Desktop\ctf\libeasy1.idb
Display at startup

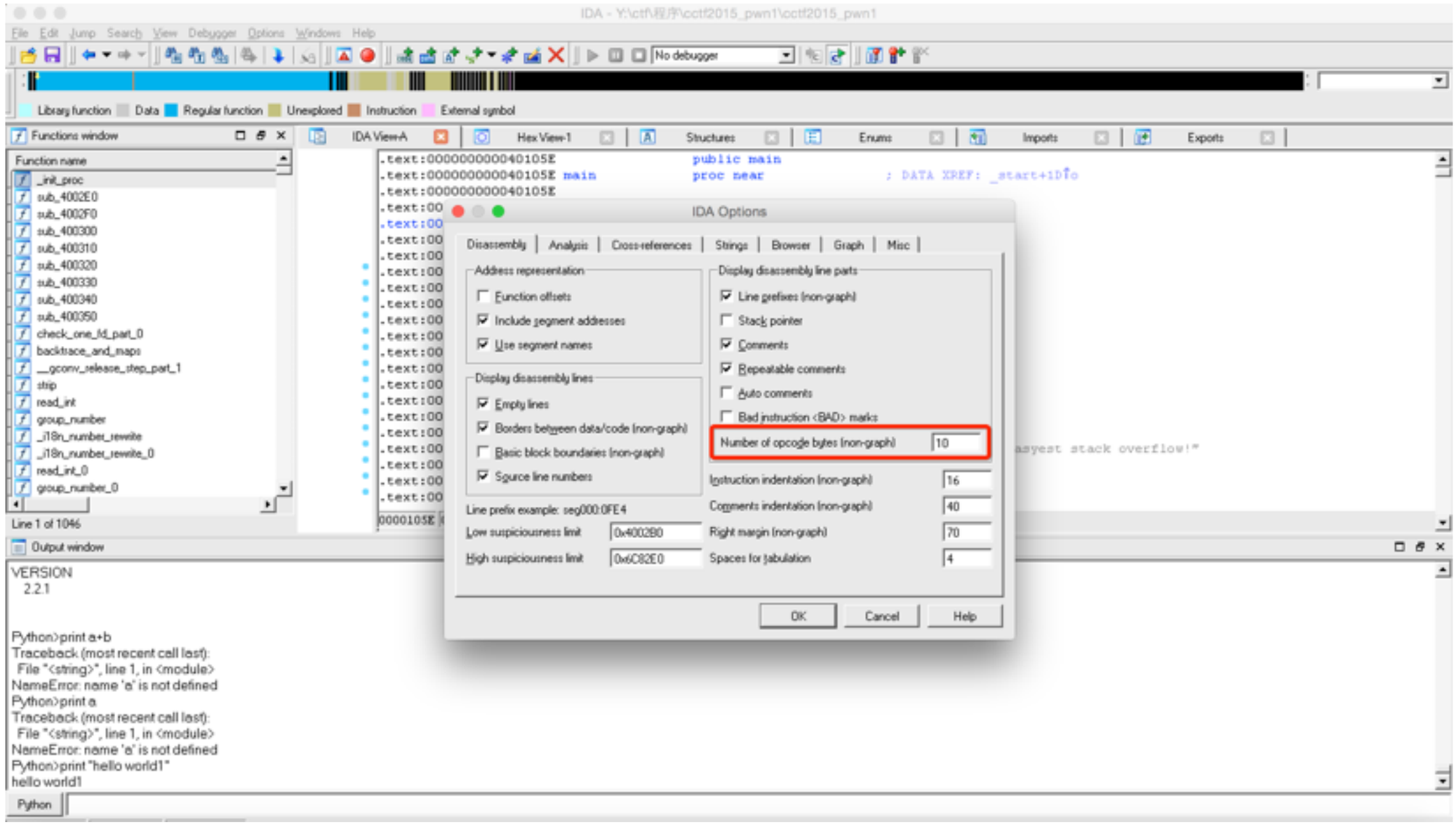
# 主要功能框


IDA - Y:\ctf\程序\cctf2015_pwn1\cctf2015_pwn1
File Edit Jump Search View Debugger Options Windows Help
Open subviews
Graphs
Toolbars
Calculator... ?
Full screen F11
Graph Overview
Recent scripts Alt+F9
Database snapshot manager... Ctrl+Shift+T
Print segment registers Ctrl+Space
Print internal flags F
Hide Ctrl+Numpad-
Unhide Ctrl+Numpad+
Hide all
Unhide all
Delete hidden area
Setup hidden items...
Quick view Ctrl+1
Disassembly
Proximity browser
Generate pseudocode F5
Hex dump
Exports
Imports
Names Shift+F4
Functions Shift+F3
Strings Shift+F12
Segments Shift+F7
Segment registers Shift+F8
Selectors
Signatures Shift+F5
Type libraries Shift+F11
Structures Shift+F9
Enumerations Shift+F10
Local types Shift+F1
Cross references
Function calls
Notepad
Problems
Patches
Patched bytes Ctrl+Alt+P
No debugger
Structures
Enums
Imports
Exports
Ordinal
Functions window
Function name
_init_proc
sub_4002E0
sub_4002F0
sub_400300
sub_400310
sub_400320
sub_400330
sub_400340
sub_400350
check_one_fd_part_0
backtrace_and_maps
__gconv_release_step
strip
read_int
group_number
_i18n_number_rewrite
_i18n_number_rewrite_0
read_int_0
group_number_0
_dl_ns
__cache_sysconf
__syscall_error_1
__data_start
Line 1 of 1046
Line 348 of 858
Output window
VERSION
2.2.1
Python>print a+b
Traceback (most recent call last):
File "<string>", line 1, in <module>
NameError: name 'a' is not defined
Python>print a
Traceback (most recent call last):
File "<string>", line 1, in <module>
NameError: name 'a' is not defined
Python>print "hello world1"
hello world1
Python
AU: idle Down Disk: 133GB

# 交叉引用


IDA - Y:\ctf\程序\cctf2015_pwn1\cctf2015_pwn1
File Edit Jump Search View Debugger Options Windows Help
No debugger
Library function Data Regular function Unexplored Instruction External symbol
Functions window
Function name
_init_proc
sub_4002E0
sub_4002F0
sub_400300
sub_400310
sub_400320
sub_400330
sub_400340
sub_400350
check_one_fd_part_0
backtrace_and_maps
__gconv_release_step_part_1
strip
read_int
group_number
_i18n_number_rewrite
_i18n_number_rewrite_0
read_int_0
group_number_0
Line 1 of 1046
IDA View-A
Hex View-1
Structures
Enums
Imports
Exports
.text:000000000040105E
.text:000000000040105E ; int __cdecl main(int argc, const char **argv, const char **envp)
.text:000000000040105E public main
.text:000000000040105E main proc near ; DATA XREF: _start+1D↑o
.text:000000000040105E
.text:000000000040105E var_1020 = byte ptr -1020h
.text:000000000040105E var_20 = byte ptr -20h
.text:000000000040105E var_4 = dword ptr -4
xrefs to main
Direction Typ Address Text
Up o _start+1D mov rdi, offset main; main
OK Cancel Search Help
Line 1 of 1
.text:000000000040107D mov rax, cs:stdout
.text:0000000000401084 mov esi, 0
.text:0000000000401089 mov rdi, rax
.text:000000000040108C call setbuf
.text:0000000000401091 mov edi, offset aEasyestStackOv ; "Easyest stack overflow!"
.text:0000000000401096 call puts
0000105E 000000000040105E: main (Synchronized with Hex View-1)
Output window
Python>print a+b
Traceback (most recent call last):
File "<string>", line 1, in <module>
NameError: name 'a' is not defined
Python>print a
Traceback (most recent call last):
File "<string>", line 1, in <module>
NameError: name 'a' is not defined
Python>print "hello world1"
hello world1
Command "JumpOpXref" failed
Command "JumpOpXref" failed
Python

# 机器码


IDA - Y:\ctf\程序\occtf2015_pwn1\occtf2015_pwn1
File Edit Jump Search View Debugger Options Windows Help
No debugger
Library function Data Regular function Unexplored Instruction External symbol
Functions window
Function name
_init_proc
sub_4002E0
sub_4002F0
sub_400300
sub_400310
sub_400320
sub_400330
sub_400340
sub_400350
check_one_fd_part_0
backtrace_and_maps
__gconv_release_step_part_1
strip
read_int
group_number
_i18n_number_rewrite
_i18n_number_rewrite_0
read_int_0
group_number_0
Line 1 of 1046
IDA View-A
Hex View-1
Structures
Enums
Imports
Exports
.text:000000000040105E public main
.text:000000000040105E main proc near ; DATA XREF: _start+1D↑o
.text:000000000040105E
IDA Options
Disassembly Analysis Cross-references Strings Browser Graph Misc
Address representation
Function offsets
Include segment addresses
Use segment names
Display disassembly line parts
Line prefixes (non-graph)
Stack pointer
Comments
Repeatable comments
Auto comments
Bad instruction <BAD> marks
Number of opcode bytes (non-graph) 10
Display disassembly lines
Empty lines
Borders between data/code (non-graph)
Basic block boundaries (non-graph)
Source line numbers
Line prefix example: seg000:0FE4
Low suspiciousness limit 0x4002B0
High suspiciousness limit 0x6C82E0
Instruction indentation (non-graph) 16
Comments indentation (non-graph) 40
Right margin (non-graph) 70
Spaces for tabulation 4
OK Cancel Help
Output window
VERSION
2.2.1
Python>print a+b
Traceback (most recent call last):
File "<string>", line 1, in <module>
NameError: name 'a' is not defined
Python>print a
Traceback (most recent call last):
File "<string>", line 1, in <module>
NameError: name 'a' is not defined
Python>print "hello world1"
hello world1
Python

# 反汇编去同步

- 光标1处按u

```
LOAD:0A04B0D1                 call    near ptr loc_A04B0D6+1
LOAD:0A04B0D6
LOAD:0A04B0D6 loc_A04B0D6:                            ; CODE XREF: start+11↓p
LOAD:0A04B0D6                 mov     dword ptr [eax-73h], 0FFEB0A40h
LOAD:0A04B0D6 start           endp
LOAD:0A04B0D6
LOAD:0A04B0DD
LOAD:0A04B0DD loc_A04B0DD:                            ; CODE XREF: LOAD:0A04B14C↓j
LOAD:0A04B0DD                 loopne  loc_A04B06F
LOAD:0A04B0DF                 mov     dword ptr [eax+56h], 5CDAB950h
LOAD:0A04B0E6                 iret
LOAD:0A04B0E6 ;---------------------------------------------------------------
LOAD:0A04B0E7                 db 47h
LOAD:0A04B0E8                 db 31h, 0FFh, 66h
LOAD:0A04B0EB ;---------------------------------------------------------------
LOAD:0A04B0EB
LOAD:0A04B0EB loc_A04B0EB:                            ; CODE XREF: LOAD:0A04B098↑j
LOAD:0A04B0EB                 mov     edi, 0C7810D98h
```

- 在地址loc_A04B0D7处按c

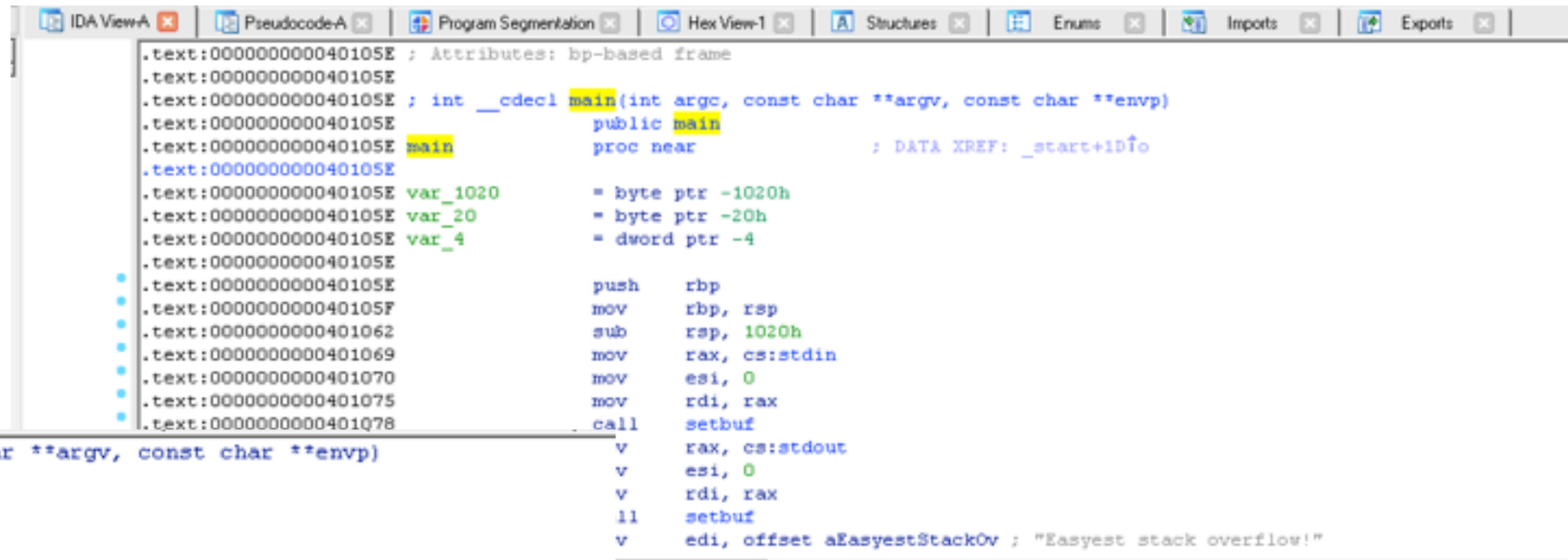
```
LOAD:0A04B0D1                 call    loc_A04B0D7
LOAD:0A04B0D1 ;---------------------------------------------------------------
LOAD:0A04B0D6                 db 0C7h ; ¦
LOAD:0A04B0D7 ;---------------------------------------------------------------
LOAD:0A04B0D7
LOAD:0A04B0D7 loc_A04B0D7:                            ; CODE XREF: start+11↑p
LOAD:0A04B0D7                 pop     eax
LOAD:0A04B0D8                 lea     eax, [eax+0Ah]
LOAD:0A04B0DB
```

# 动态计算

```
LOAD:0A04B3BE          mov     ecx, 7F131760h  ; ecx = 7F131760
LOAD:0A04B3C3          xor     edi, edi        ; edi = 00000000
LOAD:0A04B3C5          mov     di, 1156h       ; edi = 00001156
LOAD:0A04B3C9          add     edi, 133AC000h  ; edi = 133AD156
LOAD:0A04B3CF          xor     ecx, edi        ; ecx = 6C29C636
LOAD:0A04B3D1          sub     ecx, 622545CEh  ; ecx = 0A048068
LOAD:0A04B3D7          mov     edi, ecx        ; edi = 0A048068
LOAD:0A04B3D9          pop     eax
LOAD:0A04B3DA          pop     esi
LOAD:0A04B3DB          pop     ebx
LOAD:0A04B3DC          pop     edx
LOAD:0A04B3DD          pop     ecx
❶ LOAD:0A04B3DE        xchg    edi, [esp]      ; TOS = 0A048068
LOAD:0A04B3E1          retn                    ; return to 0A048068
```

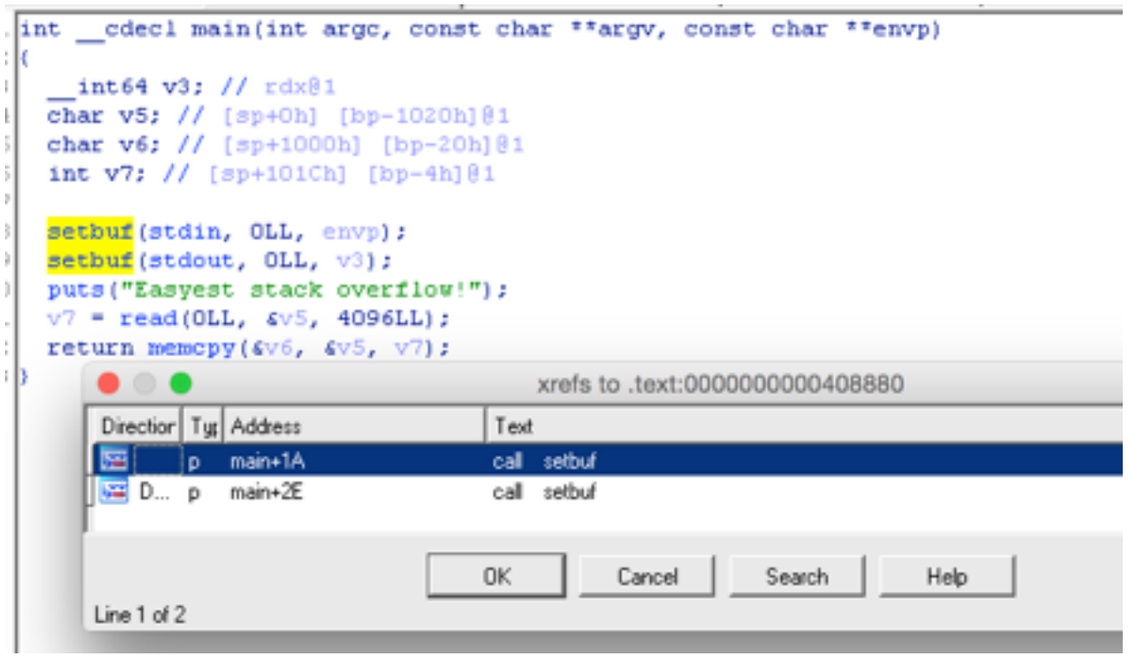
# hex-rays

- 被神化的F5



# CodeXplorer

- 听说，逆向时F5和CodeXplorer更配哦！


- 快捷键j，可以从伪代码快速跳到汇编代码的相应位置

# IDApython

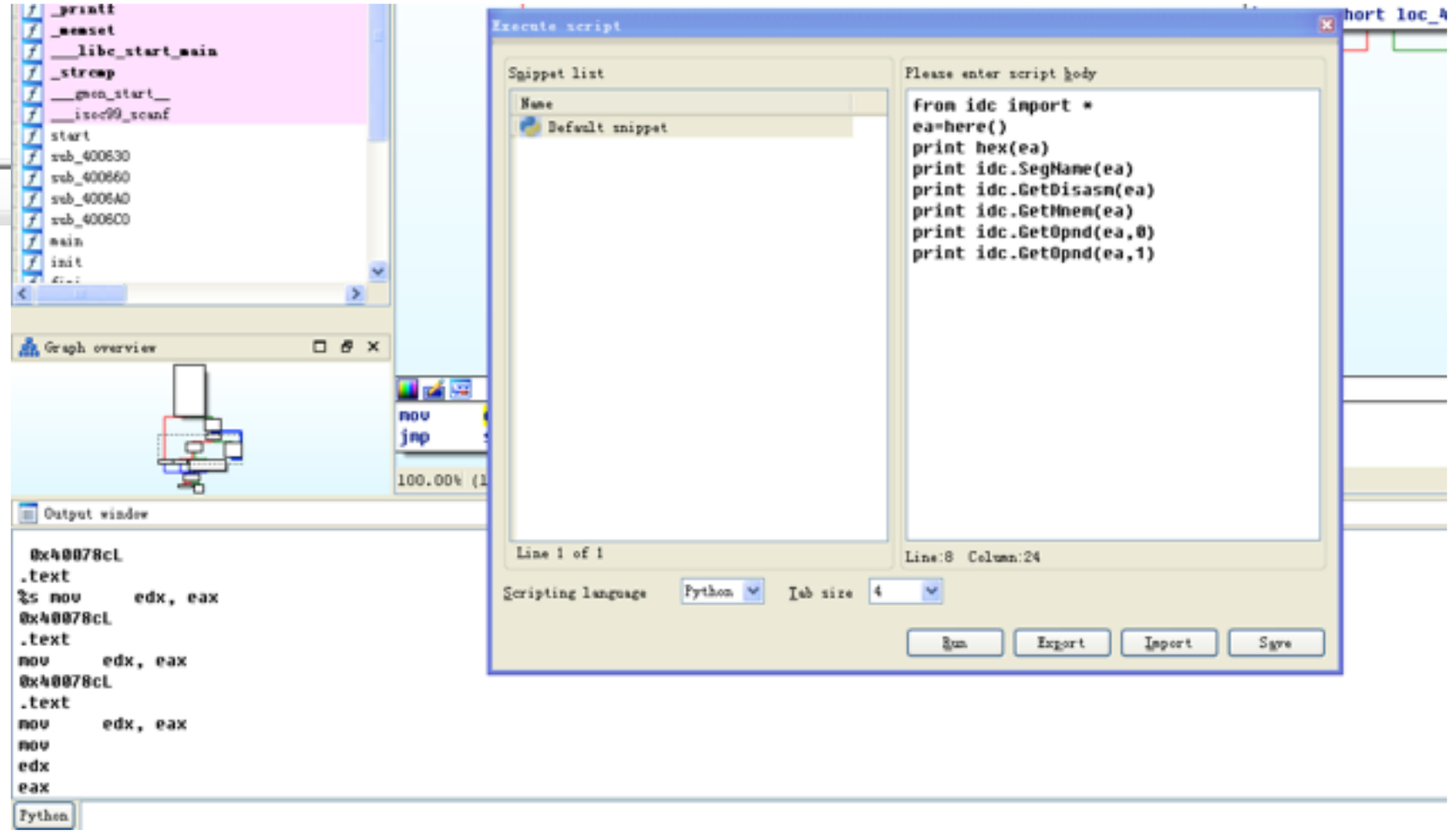

- http://researchcenter.paloaltonetworks.com/2015/12/using-idapython-to-make-your-life-easier-part-1{2，3，4}/

- IDAPython-Book

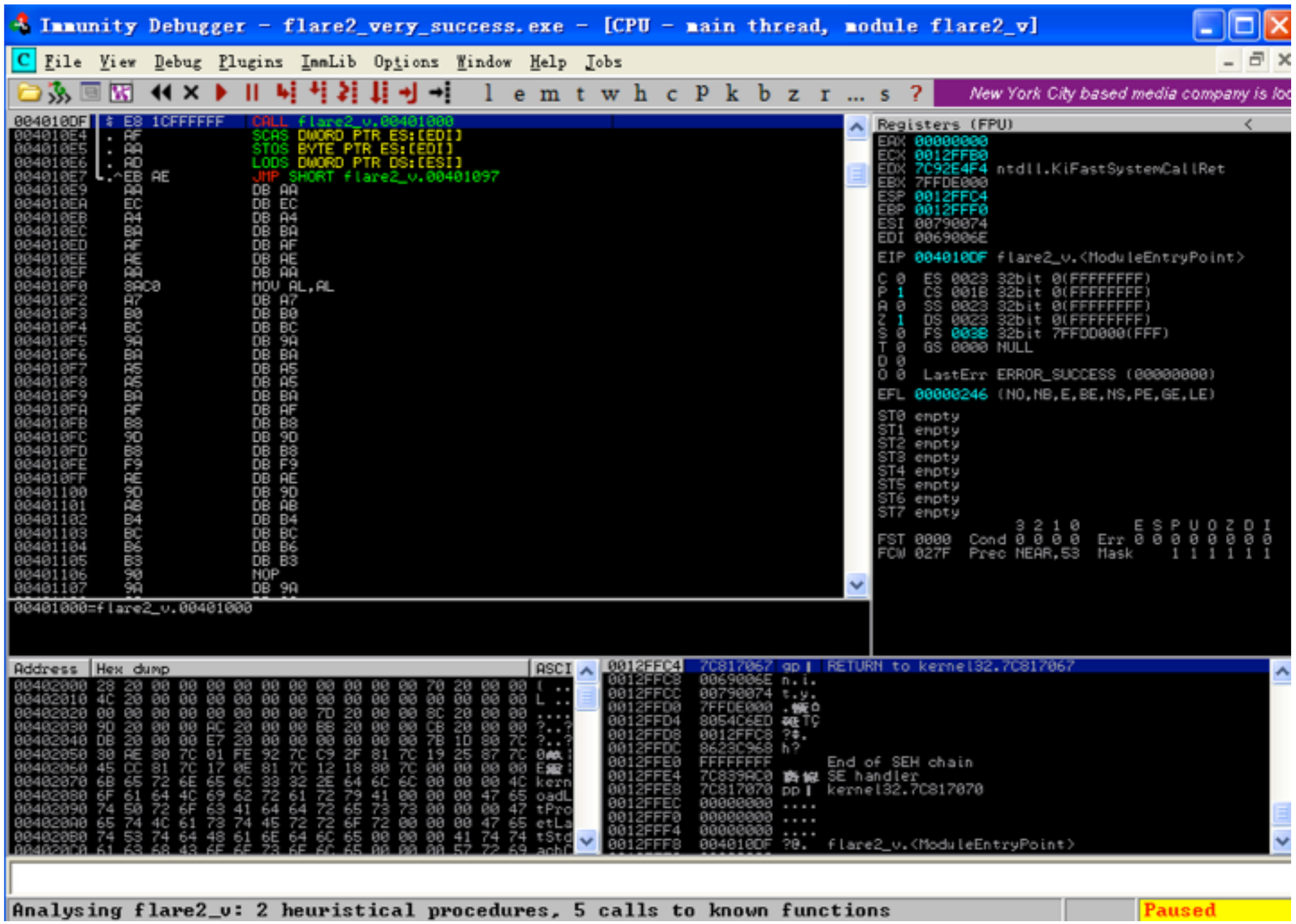
# 两种方式

```
Python>ea=here()
Python>print ea
134514054
Python>print hex(ea)
0x8048586L
80484A0: call analysis failed
Python>ea=here()
Python>print hex(ea)
0x80485b4L
Python>hex(MinEA())
0x80483e0L
Python>hex(MaxEA())
0x804a19cL
```

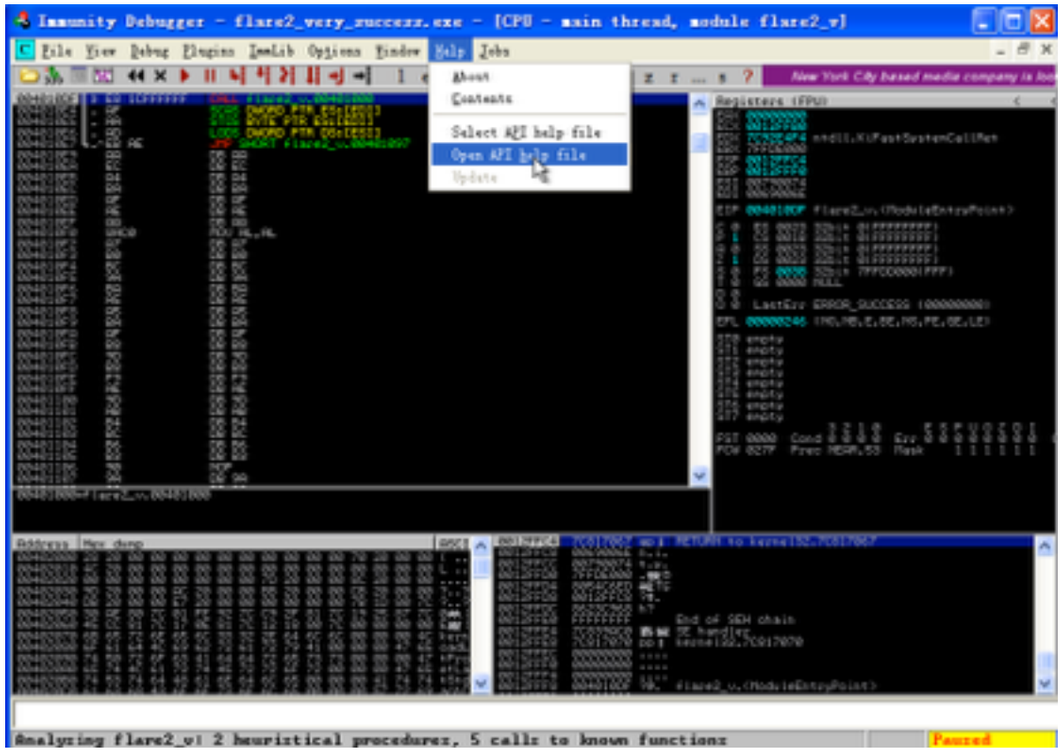
Python

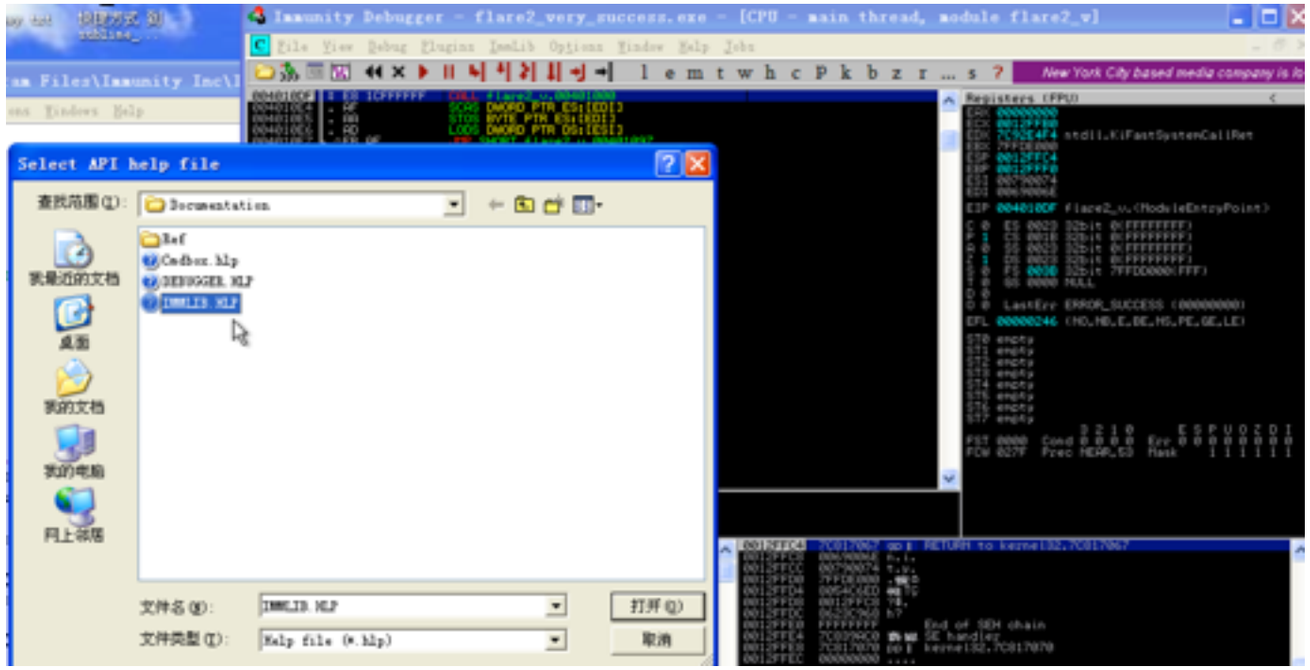


# 神器2.Immunity

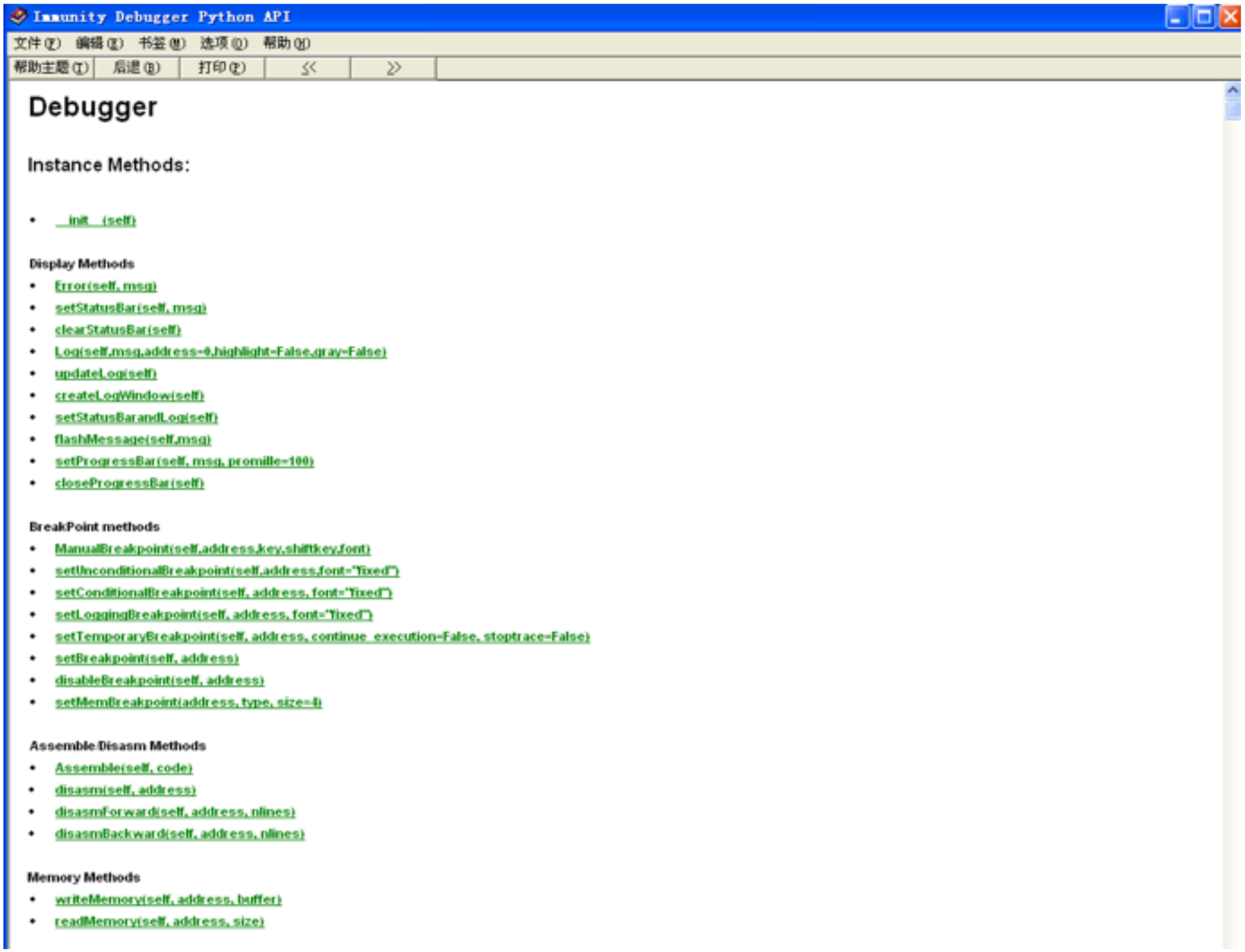


# help


Immunity Debugger - flare2_very_success.exe - [CPU - main thread, module flare2_v]
File View Debug Plugins ImmLib Options Window Help Jobs
About
Contents
Select API help file
Open API help file
Update
Analyzing flare2_v: 2 heuristical procedures, 5 calls to known functions
Paused


Select API help file
查找范围(I):
Documentation
Ref
Cmdhex.hlp
DEBUGGER.HLP
IMMLIB.HLP
文件名(N):
IMMLIB.HLP
文件类型(T):
Help file (*.hlp)
打开(O)
取消

# python： Immlib



# 写法

- 注意红框的中的内容，是必须存在的!

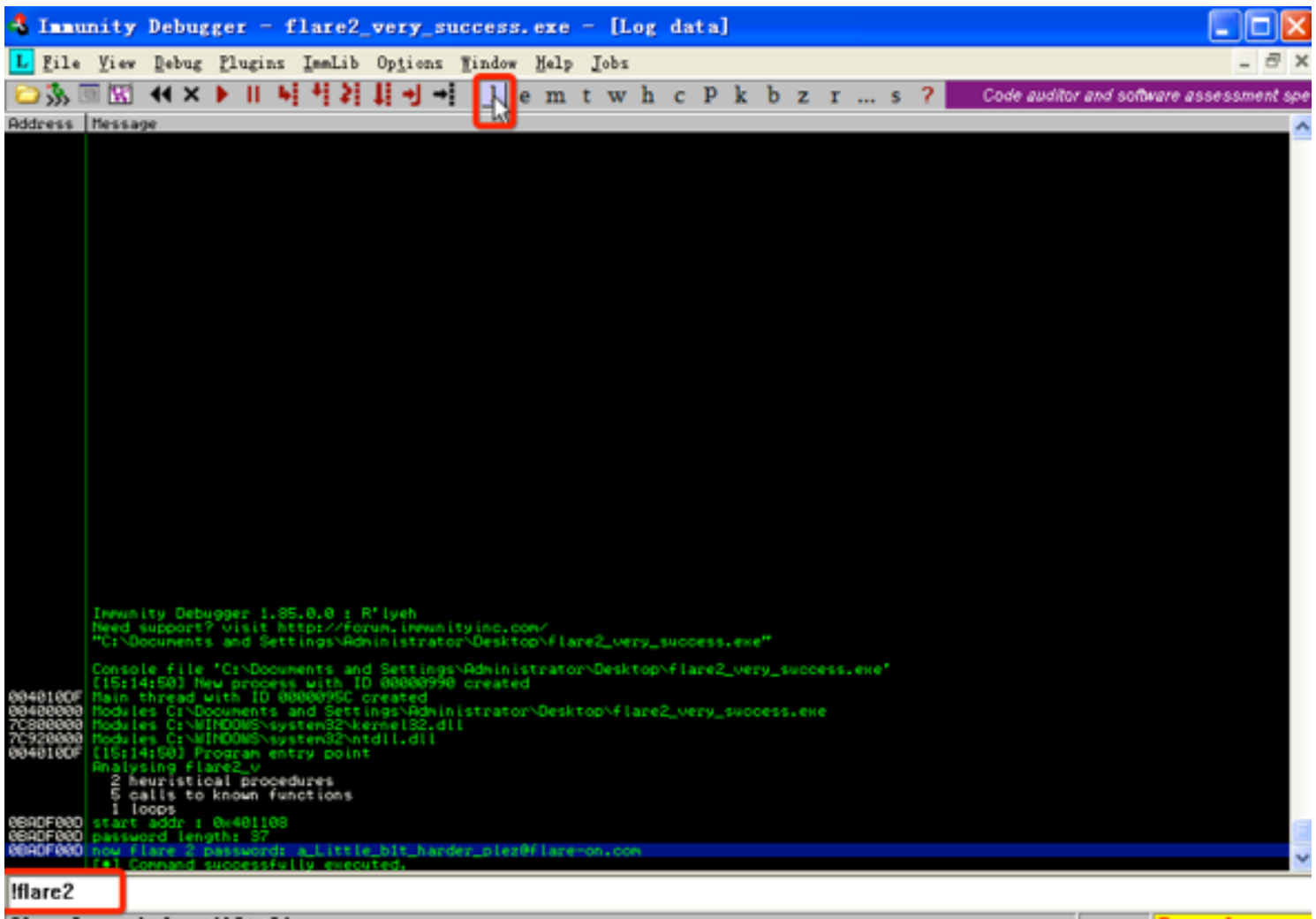
flare2.py - C:\Program Files\Immunity Inc\Immunity Debugger\PyComman...

File Edit Format Run Options Windows Help

```
#!usr/bin/env python

from immlib import *

def main(args):
    imm = Debugger()
    addr = 0x00401108
    imm.log("start addr : %s" % hex(addr))

    len = 0x25
    imm.log("password length: %s" % len)

    movoff = 0
    tmp = 0
    decode_pass = []
    password = ''
    for i in range(0,len):
        key = imm.readMemory(addr-i,0x1)#get key
        tmp += ord(key)
        if i < 2:
            a = 1 << movoff
        else:
            movoff = (tmp-ord(key)) & 0x3
            a = 1 << movoff
        #imm.log("tmp:%d===a:%d" % (tmp,a))
        pass_byte = ((ord(key)-a-1) & 0xff) ^ 0xc7
        #imm.log("%x" % pass_byte)
        decode_pass.append(pass_byte)

        password += chr(decode_pass[i])
    imm.log("now flare 2 password: %s" % password)

    return "[*] Command successfully executed."
```

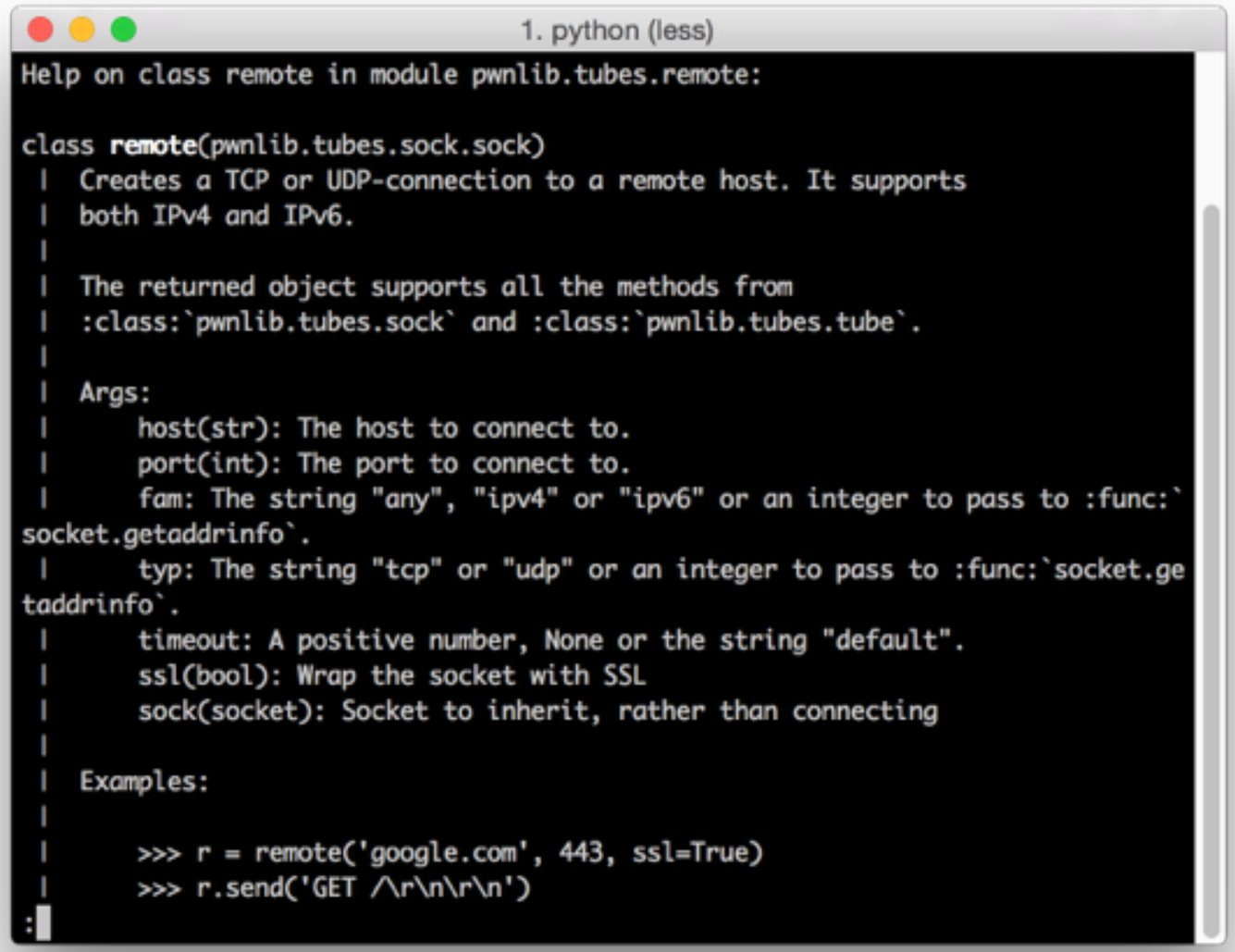
# 使用

- 保存在immunity安装目录下的PyCommands
- 在命令行输入!filename.py



# 神器3.pwntools

- pip install pwntools
- import pwn



# revserse example 1

- 2014 sctf 逆向工程 50分
- xp + ida(64) + idapython

```
➜  sctf题目收集  file re50
re50: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically linked (u
ses shared libs), for GNU/Linux 2.6.24, stripped
```

# revserse example 2

- Second FireEye FLARE On Challenge (2015)第一題
- xp + IDA + IDAPython
- 7-zip
- resource_hacker

# exploit example

- 2014 sctf pwn200
- kali + IDA + peda + pwntools
- echo 0 > randomize_va_space

- gdb ./pwn200
- b write
- conti
- print system
- checksec
- searchmem “/bin/sh”

# stack layout

# first rewrite

```
gdb-peda$ x/20x $ebp-0x1c
0xbffe3e6c:     0x73    0x79    0x63    0x6c    0x6f    0x76    0x65    0x72
0xbffe3e74:     0x00    0xff    0xff    0xff    0x42    0x42    0x42    0x42
0xbffe3e7c:     0xff    0x00    0x00    0x00
gdb-peda$ x/10x $ebp-0xc
0xbffe3e7c:     0xff    0x00    0x00    0x00    0xf4    0xef    0x79    0xb7
0xbffe3e84:     0x00    0x00
gdb-peda$ print $ebp-0xc
$2 = (void *) 0xbffe3e7c
gdb-peda$
```

# second rewrite

```
gdb-peda$ x/200x $3
0xbffe3dec:     0x0a    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xbffe3df4:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xbffe3dfc:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xbffe3e04:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xbffe3e0c:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xbffe3e14:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xbffe3e1c:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xbffe3e24:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xbffe3e2c:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xbffe3e34:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xbffe3e3c:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xbffe3e44:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xbffe3e4c:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xbffe3e54:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xbffe3e5c:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xbffe3e64:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xbffe3e6c:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xbffe3e74:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xbffe3e7c:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xbffe3e84:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xbffe3e8c:     0x10    0xaf    0xe9    0xb7    0xef    0xbe    0xad    0xde
0xbffe3e94:     0xd4    0xaa    0xf9    0xb7    0x0a    0x3f    0xfe    0xbf
0xbffe3e9c:     0x46    0x5e    0x65    0xb7    0x01    0x00    0x00    0x00
0xbffe3ea4:     0x44    0x3f    0xfe    0xbf    0x4c    0x3f    0xfe    0xbf
0xbffe3eac:     0x60    0x08    0x7c    0xb7    0x21    0x68    0x7d    0xb7
```

# malware example

- 真实案例
- xp + Immunity + IDA + Processmon +wireshark
- 命令：

  file => 格式等信息

  strings => 字符串 (-a)
- tips：“眼见不一定为实” => show me the code

# 总结

- 第一次静态分析是忽略关键的API
- 第一次动态分析时忽略的程序的路径检查，导致异常无法设置断点
- VB的API调用较为特殊，需要加强
- 模拟环境较为粗糙，server端应可定制化发包
- IDA很强大，找两本书看看！

# thank you

- http://202.108.211.5/
- Questions?