

CTF逆向工程

r4phael@SUS



Why reverse



SUS

- 恶意软件分析
- 漏洞分析
- 编译器性能检查
- 查看调试信息



How to disassembly



SUS

- 基础反汇编算法
 - 确定代码区域
 - 读取该指令
 - 解析该指令的操作数
 - 输出为一条汇编指令，继续读取
- 线性扫描反汇编
- 基于控制流的反汇编

目录

content

01

IDA Introduction (1)

02

Anti-Reverse

03

Reverse in CTF

04

IDA Introduction (2)

05

Ollydbg

06

Assembly Language



IDA Introduction



SUS

IDA Pro (简称IDA) 是DataRescue公司 (www.datarescue.com) 出品的一款交互式反汇编工具, 它功能强大、操作复杂, 要完全掌握它, 需要很多知识。IDA最主要的特性是交互和多处理器。操作者可以通过对IDA的交互来指导IDA更好地反汇编, IDA并不自动解决程序中的问题, 但它会按用户的指令找到可疑之处, 用户的工作是通知IDA怎样去做。比如人工指定编译器类型, 对变量名、结构定义、数组等定义等。这样的交互能力在反汇编大型软件时显得尤为重要。多处理器特点是指IDA支持常见处理器平台上的软件产品。IDA支持的文件类型非常丰富, 除了常见的PE格式, 还支持Windows,DOS,UNIX,Mac,Java,.NET等平台的文件格式。





IDA Introduction - Directories



SUS

- `cfg`: 包含各种配置文件
- `idc`: 包含IDA内置脚本IDC的核心内容
- `ids`: 包含用于描述可被加载到IDA的二进制文件引用的共享库内容
- `loaders`: 包含在加载过程中用于识别PE或ELF等文件格式的扩展
- `plugins`: 用社区用户开发的第三方扩展功能
- `procs`: 包含提供不同架构机器语言到汇编语言的转换功能
- `sig`: 包含IDA的FLIRT技术生成的指纹
- `til`: 包含一些类型库信息



IDA Introduction - Quick start



SUS

New:

直接打开一个新的文件进行逆向

Go:

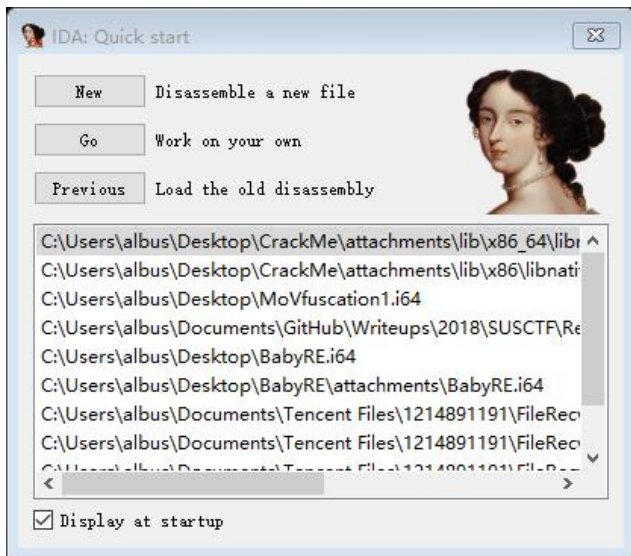
打开一个空白界面

Previous:

打开上一个工程

注意:

IDA分为32位和64位，要根据逆向的目标文件进行选择。



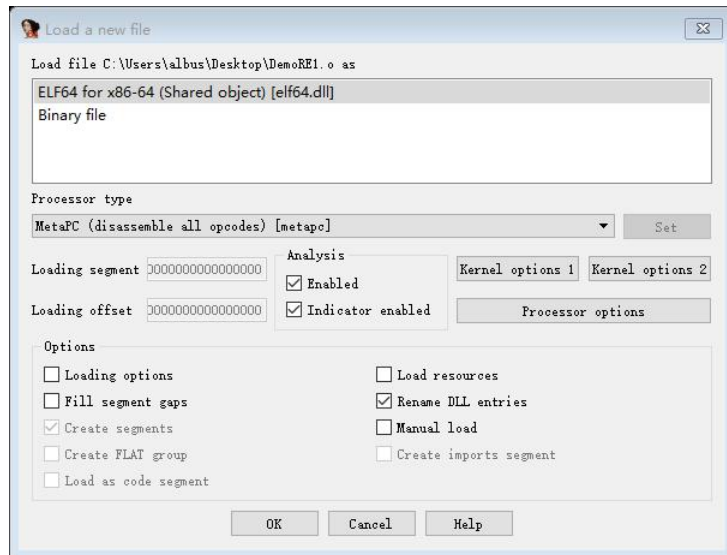


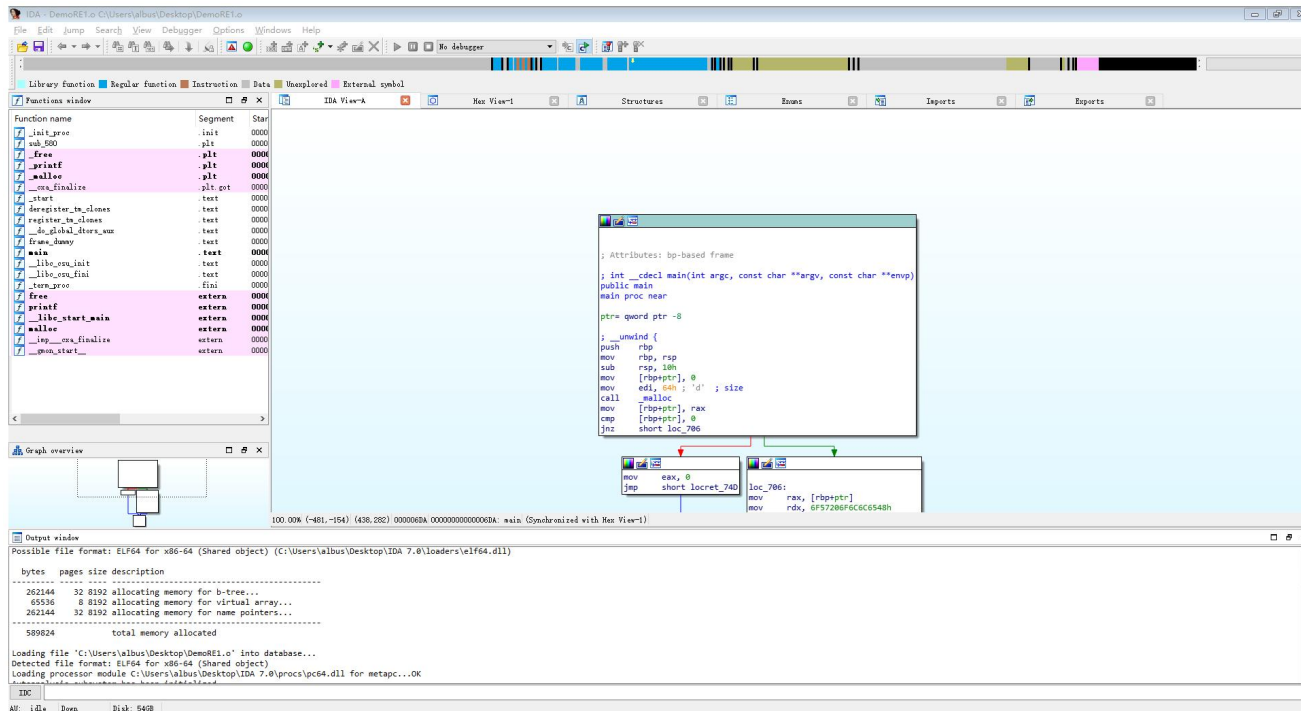
IDA Introduction - File loading



在打开一个新文件时，会看到右面这个窗口：

- 可用文件类型
- 处理器类型
- 内核选项
- 处理器选项







IDA Introduction - IDA View



SUS

```
IDA  IDA View-A  Hex View-1  Structures  Enums  Exports

.text:00000000000006DA ; ===== SUBROUTINE =====
.text:00000000000006DA
.text:00000000000006DA ; Attributes: bp-based frame
.text:00000000000006DA
.text:00000000000006DA ; int __cdecl main(int argc, const char **argv, const char **envp)
.text:00000000000006DA public main
.text:00000000000006DA main proc near ; DATA XREF: _start+1D10
.text:00000000000006DA ptr = qword ptr -8
.text:00000000000006DA
.text:00000000000006DA ; __unwind {
.text:00000000000006DA push rbp
.text:00000000000006DB mov rbp, rsp
.text:00000000000006DE sub rsp, 10h
.text:00000000000006E2 mov [rbp+ptr], 0
.text:00000000000006EA mov edi, 64h ; 'd' ; size
.text:00000000000006EF call _malloc
.text:00000000000006F4 mov [rbp+ptr], rax
.text:00000000000006F8 cmp [rbp+ptr], 0
.text:00000000000006FD jnz short loc_706
.text:00000000000006FF mov eax, 0
.text:0000000000000704 jmp short locret_74D
.text:0000000000000706 ;
.text:0000000000000706
.text:0000000000000706 loc_706: ; CODE XREF: main+231j
.text:0000000000000706 mov rax, [rbp+ptr]
.text:000000000000070A mov rdx, 6F57206F6C6C6548h
.text:0000000000000714 mov [rax], rdx
.text:0000000000000717 mov dword ptr [rax+8], 21646C72h
.text:000000000000071E mov word ptr [rax+0Ch], 0Ah
.text:0000000000000724 mov rax, [rbp+ptr]
.text:0000000000000728 mov rsi, rax
.text:000000000000072B lea rdi, format ; "%s"
.text:0000000000000732 mov eax, 0
.text:0000000000000737 call _printf
.text:000000000000073C mov rax, [rbp+ptr]
.text:0000000000000740 mov rdi, rax ; ptr
.text:0000000000000743 call _free
.text:0000000000000748 mov eax, 0
.text:000000000000074D
.text:000000000000074D locret_74D: ; CODE XREF: main+2A1j
.text:000000000000074D leave
.text:000000000000074E retn
.text:000000000000074E ; } // starts at 6DA
.text:000000000000074E main endp
.text:000000000000074E
```



IDA Introduction - Names

Names window

| Name | Address | Public |
|-------------------|-----------------|--------|
| f start | 000000000400940 | P |
| f nullsub_1 | 000000000418220 | |
| f nullsub_2 | 000000000443F20 | |
| f nullsub_5 | 00000000044FB68 | |
| f nullsub_6 | 000000000475144 | |
| f nullsub_3 | 00000000047F410 | |
| f nullsub_4 | 00000000049D640 | |
| A aDevUrandom | 0000000004A21E4 | |
| A aFatalDoubleFre | 0000000004A21F1 | |
| A a0x4x0x4x | 0000000004A220F | |
| A aErrorIpOutOfBo | 0000000004A221F | |
| A aErrorInvalidRe | 0000000004A2238 | |
| A aMessageVmHalt | 0000000004A2252 | |
| A a5s | 0000000004A2265 | |
| A a5sS | 0000000004A226A | |
| A a5sSS | 0000000004A2272 | |
| A a5sSSS | 0000000004A227E | |

Line 2 of 932



IDA Introduction - Output



SUS

Output window

```
Loading processor module C:\Program Files\IDA 7.0\procs\pc64.dll for metapc...OK
Loading type libraries...
Autoanalysis subsystem has been initialized.
Database for file 'esrever-mv' has been loaded.
Hex-Rays Decompiler plugin has been loaded (v7.0.0.170914)
  License: 55-BAE5-8A04-93 Jiang Ying, Personal license (1 user)
  The hotkeys are F5: decompile, Ctrl-F5: decompile all.
  Please check the Edit/Plugins menu for more informaton.
IDAPython Hex-Rays bindings initialized.
```

```
-----
Python 2.7.13 (v2.7.13:a06454b1afa1, Dec 17 2016, 20:53:40) [MSC v.1500 64 bit (AMD64)]
IDAPython 64-bit v1.7.0 final (serial 0) (c) The IDAPython Team <idapython@googlegroups.com>
-----
```

Python



IDA Introduction - Strings



SUS

| Unexplored External symbol | | | |
|---|--------|------|-----------------------------|
| IDA View-A Pseudocode-A Strings window Hex View-1 Struc | | | |
| Address | Length | Type | String |
| LOAD:0000... 0000001C | | C | /lib64/ld-linux-x86-64.so.2 |
| LOAD:0000... 0000000A | | C | libc.so.6 |
| LOAD:0000... 00000007 | | C | printf |
| LOAD:0000... 00000007 | | C | malloc |
| LOAD:0000... 0000000F | | C | __cxa_finalize |
| LOAD:0000... 00000012 | | C | __libc_start_main |
| LOAD:0000... 00000005 | | C | free |
| LOAD:0000... 0000000C | | C | GLIBC_2.2.5 |
| LOAD:0000... 0000001C | | C | _ITM_deregisterTMCloneTable |
| LOAD:0000... 0000000F | | C | __gmon_start__ |
| LOAD:0000... 0000001A | | C | _ITM_registerTMCloneTable |
| .eh_frame... 00000006 | | C | ;*3\$\n |



IDA Introduction - Hex View



SUS

```
IDA View-A  Strings window  Hex View-1  Structures

0000000000400930  00 00 48 89 C5 E9 B7 FD FF FF 66 0F 1F 44 00 00  ..H.....f..D..
0000000000400940  31 ED 49 89 D1 5E 48 89 E2 48 83 E4 F0 50 54 49  1....H.....TI
0000000000400950  C7 C0 D0 27 40 00 48 C7 C1 40 27 40 00 48 C7 C7  ....@.H..@'@.H..
0000000000400960  F0 05 40 00 E8 87 15 00 00 F4 66 0F 1F 44 00 00  .........D...
0000000000400970  B8 F7 DD 6C 00 55 48 2D F0 DD 6C 00 48 83 F8 0E  ....UH-....H...
0000000000400980  48 89 E5 76 1B B8 00 00 00 00 48 85 C0 74 11 5D  H.....H....]
0000000000400990  BF F0 DD 6C 00 FF E0 66 0F 1F 84 00 00 00 00 00  ....
00000000004009A0  5D C3 0F 1F 40 00 66 2E 0F 1F 84 00 00 00 00 00  ]...@.f.....
00000000004009B0  BE F0 DD 6C 00 55 48 81 EE F0 DD 6C 00 48 C1 FE  ....UH....l.H..
00000000004009C0  03 48 89 E5 48 89 F0 48 C1 E8 3F 48 01 C6 48 D1  .H.....?H....
00000000004009D0  FE 74 15 B8 00 00 00 00 48 85 C0 74 0B 5D BF F0  .t.....H....]..
00000000004009E0  DD 6C 00 FF E0 0F 1F 00 5D C3 66 0F 1F 44 00 00  ....]....D...
00000000004009F0  80 3D 09 D4 2C 00 00 75 22 55 48 89 E5 E8 6E FF  .=.....u"UH....
0000000000400A00  FF FF B8 50 F4 49 00 48 85 C0 74 07 BF E0 FA 4B  ...P.....
0000000000400A10  00 FF D0 5D C6 05 E5 D3 2C 00 01 F3 C3 0F 1F 00  ....
0000000000400A20  55 B8 20 F2 49 00 48 85 C0 48 89 E5 74 0F BE 20  U.....
0000000000400A30  DE 6C 00 BF E0 FA 4B 00 E8 E3 E7 09 00 BF F8 BE  ....
0000000000400A40  6C 00 48 83 3F 00 75 08 5D E9 62 FF FF FF 66 90  1.H?.u.]....f.
0000000000400A50  B8 00 00 00 00 48 85 C0 74 EE FF D0 E8 EA 66 90  ....H.....
0000000000400A60  48 83 EC 18 64 48 8B 04 25 28 00 00 00 48 89 44  H....H..%(...H.D
0000000000400A70  24 08 31 C0 0F B7 87 08 00 01 00 66 85 C0 74 50  $.1.....f...P
0000000000400A80  66 83 F8 01 74 1A 31 C0 48 8B 4C 24 08 64 48 33  f...t.1...L$.dH3
0000000000400A90  0C 25 28 00 00 00 75 5F 48 83 C4 18 C3 0F 1F 00  .%(...u_H.....
0000000000400AA0  0F B7 87 0A 00 01 00 48 8D 74 24 07 BA 01 00 00  ....H.t$....
0000000000400AB0  00 BF 01 00 00 00 88 44 24 07 E8 41 F8 03 00 48  ....D$......H
0000000000400AC0  83 F8 01 0F 95 C0 0F B6 C0 F7 D8 E8 B8 0F 1F 00  ....
0000000000400AD0  48 8D 74 24 07 BA 01 00 00 00 31 FF E8 BF F7 03  H.t$.....1....
0000000000400AE0  00 48 89 C2 B8 FF FF FF FF 48 83 FA 01 75 99 66  .H..'.H..u.f
0000000000400AF0  0F BE 44 24 07 EB 91 E8 B4 33 04 00 0F 1F 40 00  ..D$.....3....@.
0000000000400B00  48 81 EC D8 00 00 00 84 C0 48 89 54 24 30 48 89  H.....T$0H.
0000000000400B10  4C 24 38 4C 89 44 24 40 4C 89 4C 24 48 74 37 0F  L$8L.D$@L.L$Ht7.


00000976 0000000000400976: sub_400970+6 (Synchronized with IDA View-A)
```



IDA Introduction - Imports & Exports



| Imports | | | |
|---------|---------|------|---------|
| Address | Ordinal | Name | Library |
| | | | |

| Exports | | |
|---|------------------|--------------|
| Name | Address | Ordinal |
|  start | 0000000000400940 | [main entry] |
| | | |

Line 1 of 1



IDA Introduction - Functions



SUS

| Functions window | | |
|------------------|--|---------|
| Function name | | Segment |
| sub_4002C8 | | .init |
| sub_4002F0 | | .plt |
| sub_400300 | | .plt |
| sub_400310 | | .plt |
| sub_400320 | | .plt |
| sub_400330 | | .plt |
| sub_400340 | | .plt |
| sub_400350 | | .plt |
| sub_400360 | | .plt |
| sub_400370 | | .plt |
| sub_400380 | | .plt |
| sub_400390 | | .text |
| sub_4004D3 | | .text |
| sub_4004ED | | .text |
| sub_400551 | | .text |
| sub_40059B | | .text |
| sub_4005C0 | | .text |
| sub_4005F0 | | .text |
| sub_4006A0 | | .text |
| start | | .text |
| sub_400970 | | .text |
| sub_4009B0 | | .text |
| sub_4009F0 | | .text |
| sub_400A20 | | .text |
| sub_400A60 | | .text |
| sub_400B00 | | .text |
| sub_400BE0 | | .text |



IDA Introduction - Structures & Enums



The screenshot shows two overlapping windows from the IDA Pro interface. The 'Structures' window is in the background, displaying a list of collapsed structures. The 'Enums' window is in the foreground, showing a list of enumeration types and their associated symbolic constants.

Structures

```
00000000 ; [00000080 BYTES. COLLAPSED STRUCT __sigset_t. PRESS CTRL-NUMPAD+ TO EXPAND]
00000000 ; [00000010 BYTES. COLLAPSED STRUCT iovec. PRESS CTRL-NUMPAD+ TO EXPAND]
00000000 ; [00000018 BYTES. COLLAPSED STRUCT gcc_va_list. PRESS CTRL-NUMPAD+ TO EXPAND]
```

6. gcc_va_list:00000000

Enums

```
; Ins/Del/Ctrl-E: create/delete/edit enumeration types
; N/Ctrl-N      : create/edit a symbolic constant
; U            : delete a symbolic constant
; ; or :       : set a comment for the current item
;
; For bitfields the line prefixes display the bitmask
```



IDA Introduction - Pseudocode



SUS

```
IDA View-A  Pseudocode-A  Strings window  Hex View-1  St
1 int __cdecl main(int argc, const char **argv, const char **envp)
2 {
3     _DWORD *ptr; // [rsp+8h] [rbp-8h]
4
5     ptr = malloc(0x64uLL);
6     if ( ptr )
7     {
8         *(_QWORD *)ptr = 8022916924116329800LL;
9         ptr[2] = 560229490;
10        *((_WORD *)ptr + 6) = 10;
11        printf("%s", ptr);
12        free(ptr);
13    }
14    return 0;
15 }
```

目录

content

01

IDA Introduction (1)

02

Anti-Reverse

03

Reverse in CTF

04

IDA Introduction (2)

05

Ollydbg

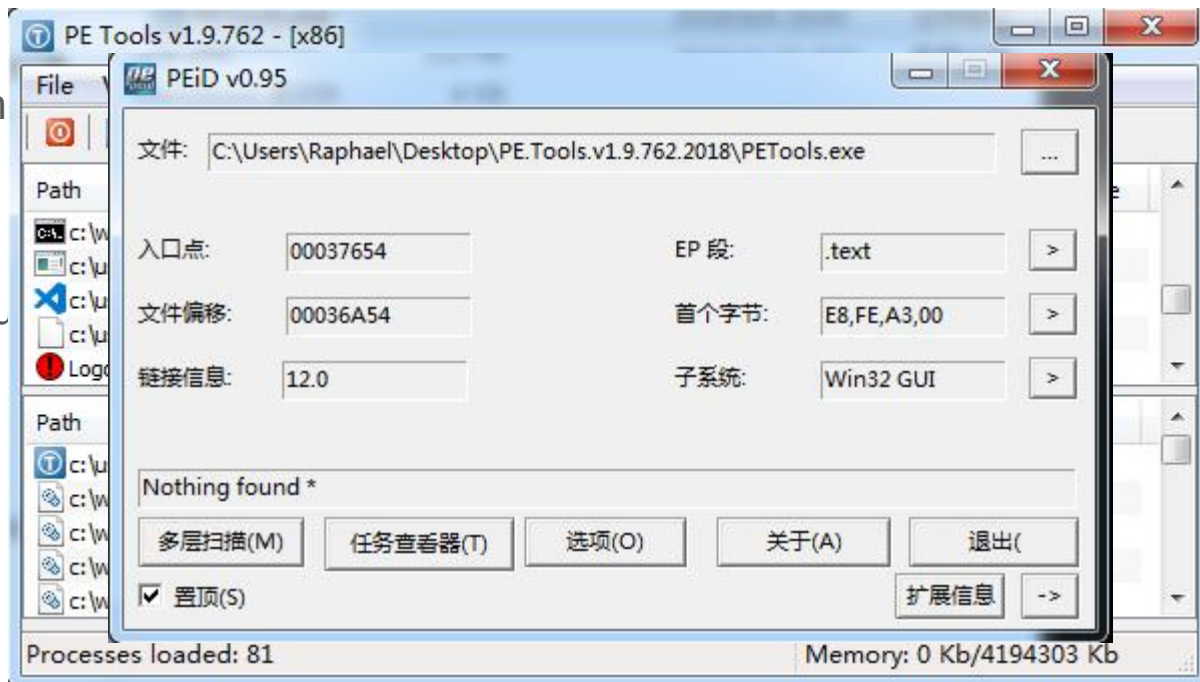
06

Assembly Language



Binary file info

- Win
-
-
- Linu
-



外壳保护技术，用户执行的实际上是这个外壳的程序，而这个外壳程序负责把用户原来的程序在内存中解开压缩，并把控制权交还给解开后的真正的程序，由于一切工作都是在内存中运行，用户根本不知道也不需要知道其运行过程，并且对执行速度没有什么影响。如果在外壳程序中加入对软件锁或钥匙盘的验证部分，它就是我们所说的外壳保护了。

壳一般分为两种：压缩壳(Packer) 和加密壳(Protector)

压缩壳是对源程序代码进行压缩后，重新封装并生成可执行文件的一种壳。杀毒软件会针对这些壳专门开发与之对应的解压缩引擎，对加壳后的软件解压缩后进行杀毒，因而通常情况下使用压缩壳难以达到免杀目的

加密壳相比压缩壳的功能要复杂的多，加密壳侧重在软件保护上，压缩重新体积不是其主要任务，通常加密壳会将程序代码混淆加密，已达到防破解的目的，一些功能强大的加密壳可以给程序添加一些额外功能，包括限制软件使用时间，给软件添加注册功能等。

壳的加载流程：

- 保存入口参数
- 获取所需API
- 解密原程序
- 跳转回原程序入口点



Unpack



SUS

要对抗外壳保护技术，首先要确定壳的种类，针对性的去除

侦壳：

PEiD、PE Tools

常见压缩壳：

UPX、ASPack、PECompact

常见加密壳：

ASProtect、VMProtect



Unpack - ESP定律



SUS

1. 程序刚载入开始 pushad/pushfd
2. 将全部寄存器压栈后就设对 ESP 寄存器设硬件断点
3. 运行程序, 触发断点
4. 删除硬件断点开始分析
5. dump脱壳后的程序
6. 重建导入表 (IAT)



Obfuscator



SUS

代码混淆意味着代码保护。混淆修改后的代码更难以被理解。例如它通常被用于 DRM（数字版权保护）软件中，用于隐藏算法和加密密钥等机密信息来保护多媒体内容。

当任何人都可以获取访问你的代码或二进制程序，但你不想让别人了解程序的工作原理的时候，你就需要代码混淆。但混淆的安全性基于代码模糊程度，破解它也只是时间问题。因此混淆工具的安全性取决于攻击者打破它所必须花费的时间。



Obfuscator - O-LLVM

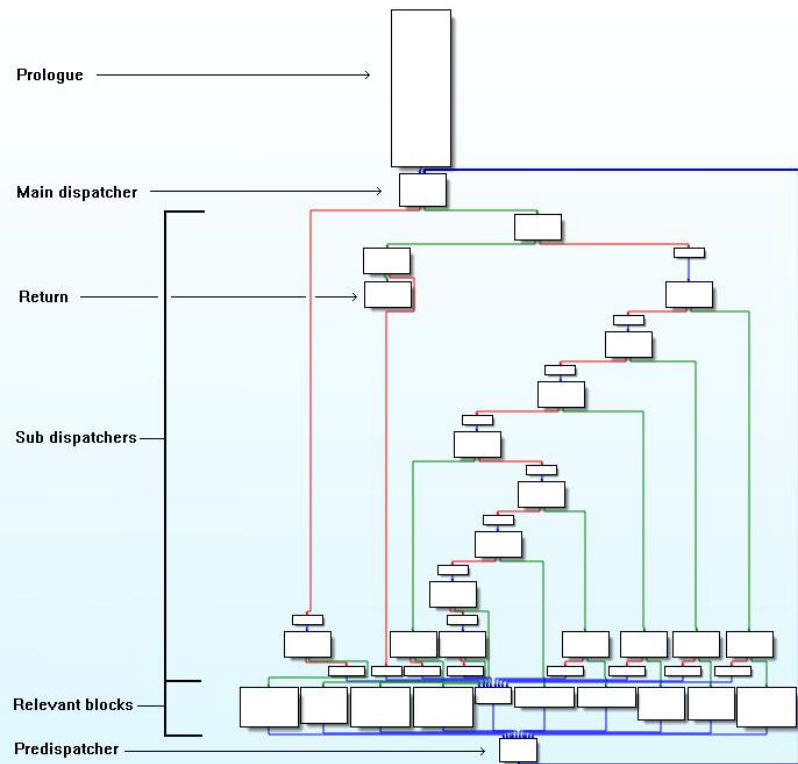


SUS

O-LLVM

项目地址: o-llvm.org

O-LLVM是针对函数控制流进行混淆的混淆器，最终将函数控制流扁平化以达到干扰逆向人员阅读的目的。目前对于这个混淆器没有成熟的自动化反混淆工具，一般是通过符号执行或动态调试以及大量的时间来破解。





Obfuscator - Movfuscator



SUS

MoVfuscator

项目地址:

<https://github.com/xoreaxeaxeax/movfuscator>

MoV混淆器的本质是一个编译器，将原始代码全部用mov指令编译，达到指令级别的混淆。目前没有任何能够还原代码的工具，仅能还原跳转指令。对于这种题目出题人一般会留下可供解题的弱点，如根据执行指令数或输入输出判断flag是否正确，从而通过爆破得出答案。

```
mov     ecx,DWORD PTR [edx+ecx*4]
mov     WORD PTR  dx:0x01fc562,dc
mov     DWORD PTR  dx:0x01fc4ce,ecx
mov     eax,0x0
mov     al,byte 0x01fc4d9
mov     al,BYTE PTR [eax+0x000535d9]
mov     dx:0x01fc560,eax
mov     eax,dc:0x01fc560
mov     ecx,DWORD PTR [eax*4+0x01fc564]
mov     DWORD PTR  dx:0x01fc5e4,ecx
mov     ecx,DWORD PTR [eax*4+0x01fc594]
mov     DWORD PTR  dx:0x01fc5ac,ecx
mov     eax,dc:0x01fc5e4
mov     max,DWORD PTR [eax]
mov     dx:0x01fc580,eax
mov     eax,dc:0x01fc580
mov     dx:0x01fc4d0,eax
mov     eax,dc:0x01fc524
mov     dx:0x01fc4ce,eax
mov     eax,0x0
mov     ecx,0x0
mov     DWORD PTR  dx:0x01fc4d0,0x1
mov     ax,dc:0x01fc4d0
mov     cx,WORD PTR  dx:0x01fc4c4
mov     cx,WORD PTR  [ecx*2+0x01167530]
mov     ecx,DWORD PTR [eax*4+0x00057490]
mov     ecx,DWORD PTR [edx+ecx*4]
mov     ecx,DWORD PTR [edx*4+0x00057490]
mov     ecx,DWORD PTR  dx:0x01fc4d0
mov     ecx,DWORD PTR [edx+ecx*4]
mov     WORD PTR  dx:0x01fc580,dc
mov     DWORD PTR  dx:0x01fc4ce,ecx
mov     ax,dc:0x01fc4c2
mov     cx,WORD PTR  dx:0x01fc4c6
mov     DWORD PTR  [max],ecx
mov     max,dc:0x01fc5ac
mov     max,DWORD PTR [max]
mov     dx:0x01fc57c,eax
mov     eax,0x0
mov     al,dc:0x01fc57e
mov     al,BYTE PTR [eax+0x00053a94]
mov     dx:0x01fc57e,al
mov     max,dc:0x01fc5ac
mov     ecx,DWORD PTR  dx:0x01fc57c
mov     DWORD PTR  [max],ecx
mov     ecx,0x0
mov     dl,BYTE PTR  dx:0x01fc551
mov     max,DWORD PTR [edx*4+0x00055660]
mov     dx:0x01fc4d0,eax
mov     eax,0x0
mov     ecx,0x0
mov     al,dc:0x01fc55c
mov     dl,BYTE PTR  dx:0x01fc4d0
mov     max,DWORD PTR [eax*4+0x01fc30]
mov     max,DWORD PTR [eax+edx*4+0x01fc00]
mov     dx:0x01fc55c,al
mov     BYTE PTR  dx:0x01fc4d0,ah
mov     max,0x0
mov     ecx,0x0
mov     al,dc:0x01fc55d
mov     dl,BYTE PTR  dx:0x01fc4d0
mov     max,DWORD PTR [eax*4+0x01fc30]
mov     max,DWORD PTR [eax+edx*4+0x01fc00]
mov     dx:0x01fc55d,al
mov     BYTE PTR  dx:0x01fc4d0,ah
mov     max,0x0
mov     ecx,0x0
mov     al,dc:0x01fc55e
```



Junk code



SUS

花指令是对抗反汇编的有效手段之一，正常代码添加了花指令之后，可以破坏静态反汇编的过程，使反汇编的结果出现错误。错误的反汇编结果会造成破解者的分析工作大量增加，进而使之不能理解程序的结构和算法，也就很难破解程序，从而达到保护代码的目的。

花指令分为可执行花指令和不可执行花指令，可执行花指令是指这段指令可以正常运行,不改变寄存器的值，反汇编器可以正常反汇编。不可执行花指令一般是一些垃圾数据，特征是这些代码位于代码不可达到的分支中。



Anti-Debug



SUS

<https://bbs.pediy.com/thread-70470.htm>

目录

content

01

IDA Introduction (1)

02

Anti-Reverse

03

Reverse in CTF

04

IDA Introduction (2)

05

Ollydbg

06

Assembly Language



Find checkpoint



SUS

1. 正面逆向（沿控制流分析）
2. 从信息输入/输出处分析
3. 利用字符串寻找



Common algorithm



SUS

1. 没算法
2. 简单异或（连续异或）
3. 编码（Base系列）
4. 解方程
5. 加密算法（RC4、TEA等）
6. 摘要算法（MD5、SHA1）
7. 脑洞算法（走迷宫……）

```
0 *****\x00\x00\x00\x00\x00\x00
1 #####\x00\x00\x00\x00\x00\x00
2 *****\x00\x00\x00\x00\x00\x00
3 #####\x00\x00\x00\x00\x00\x00
4 *****\x00\x00\x00\x00\x00\x00
5 #####\x00\x00\x00\x00\x00\x00
6 *****\x00\x00\x00\x00\x00\x00
7 #####\x00\x00\x00\x00\x00\x00
8 *****\x00\x00\x00\x00\x00\x00
9 #####\x00\x00\x00\x00\x00\x00
10 *****\x00\x00\x00\x00\x00\x00
11 #####\x00\x00\x00\x00\x00\x00
12 *****\x00\x00\x00\x00\x00\x00
13 #####\x00\x00\x00\x00\x00\x00
14 *****\x00\x00\x00\x00\x00\x00
15 #####\x00\x00\x00\x00\x00\x00
16 *****\x00\x00\x00\x00\x00\x00
17 #####\x00\x00\x00\x00\x00\x00
18 *****\x00\x00\x00\x00\x00\x00
19 #####\x00\x00\x00\x00\x00\x00
20 *****\x00\x00\x00\x00\x00\x00
21 #####\x00\x00\x00\x00\x00\x00
22 *****\x00\x00\x00\x00\x00\x00
23 #####\x00\x00\x00\x00\x00\x00
24 *****\x00\x00\x00\x00\x00\x00
25 #####\x00\x00\x00\x00\x00\x00
26 *****\x00\x00\x00\x00\x00\x00
27 #####\x00\x00\x00\x00\x00\x00
28 *****\x00\x00\x00\x00\x00\x00
29 #####\x00\x00\x00\x00\x00\x00
30 *****\x00\x00\x00\x00\x00\x00
31 #####\x00\x00\x00\x00\x00\x00
32 *****\x00
```




Tips



SUS

1. 尽量多写代码
2. 用户代码集中
3. 大胆猜测
4. 区分代码
5. 耐心！



CTF in the future



SUS

1. 代码量增大
2. 程序结构复杂
3. 运用一些现代语言特性（面向对象、Template）
4. 一些奇怪的语言（不只是C/C++）
5. 编译器开启优化，加壳等等
6. 可能还会有固件和驱动程序的逆向

目录

content

01

IDA Introduction (1)

02

Anti-Reverse

03

Reverse in CTF

04

IDA Introduction (2)

05

Ollydbg

06

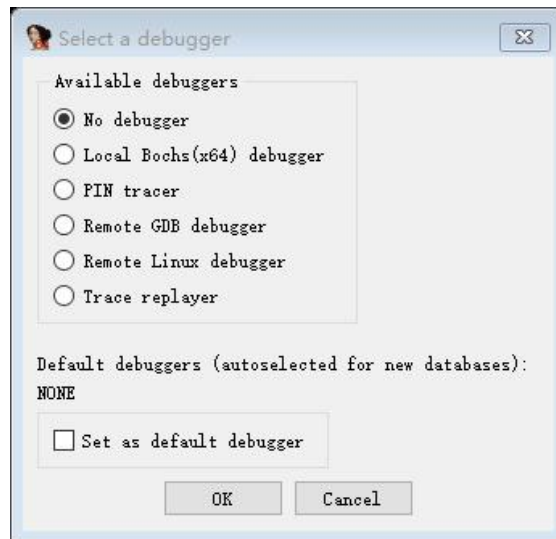
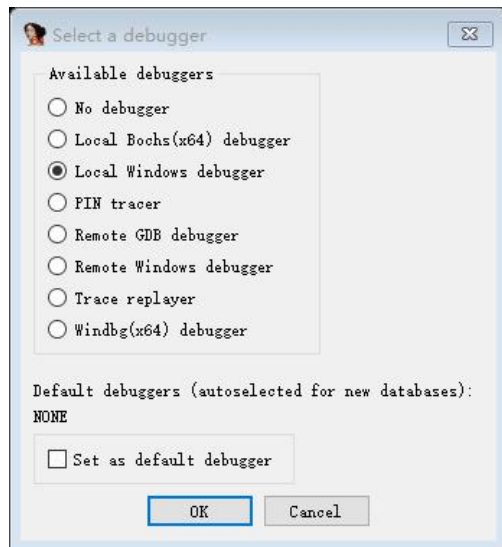
Assembly Language

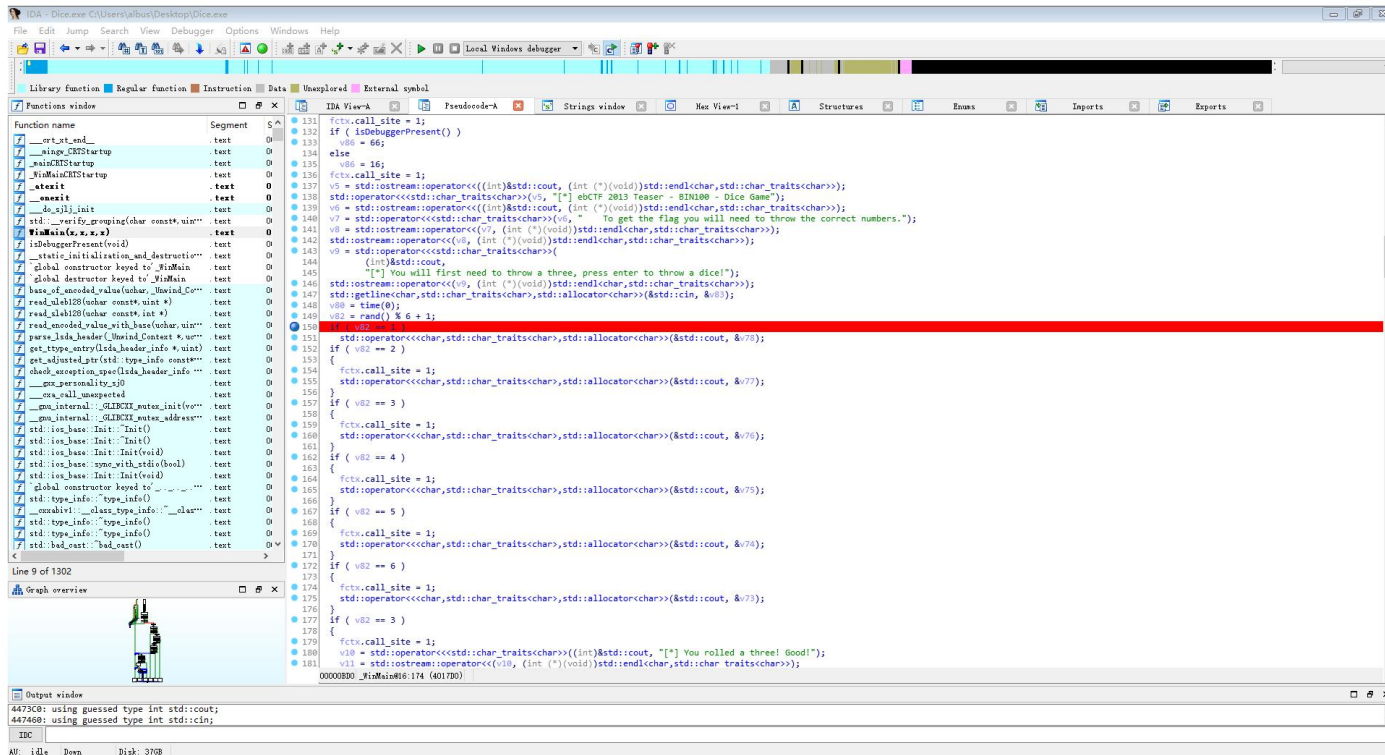


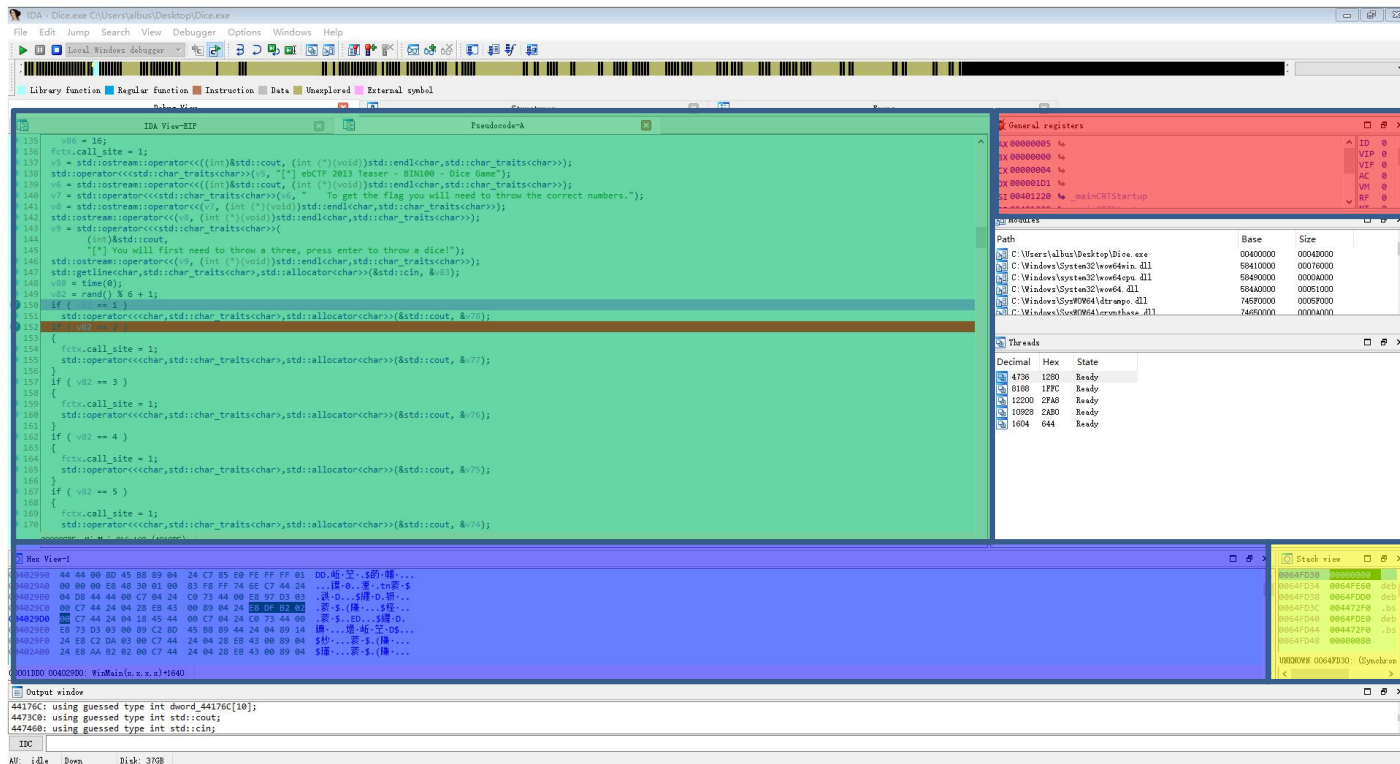
IDA Introduction - Debugger



通过Debugger -> switch debugger菜单打开调试器选择窗口





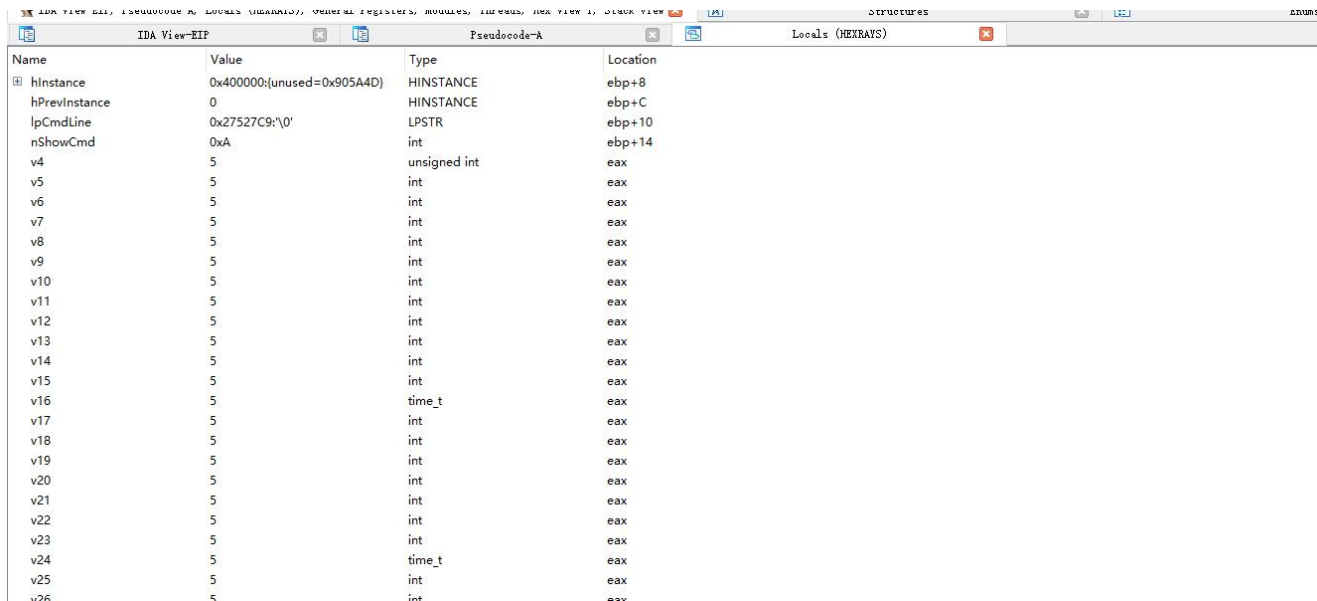




IDA Introduction - Debugger



通过Debugger -> Debugger Windows -> Locals菜单打开变量监视窗口



| Name | Value | Type | Location |
|---------------|----------------------------|--------------|----------|
| hInstance | 0x400000;(unused=0x905A4D) | HINSTANCE | ebp+8 |
| hPrevInstance | 0 | HINSTANCE | ebp+C |
| lpCmdLine | 0x27527C9:"\0" | LPSTR | ebp+10 |
| nShowCmd | 0xA | int | ebp+14 |
| v4 | 5 | unsigned int | eax |
| v5 | 5 | int | eax |
| v6 | 5 | int | eax |
| v7 | 5 | int | eax |
| v8 | 5 | int | eax |
| v9 | 5 | int | eax |
| v10 | 5 | int | eax |
| v11 | 5 | int | eax |
| v12 | 5 | int | eax |
| v13 | 5 | int | eax |
| v14 | 5 | int | eax |
| v15 | 5 | int | eax |
| v16 | 5 | time_t | eax |
| v17 | 5 | int | eax |
| v18 | 5 | int | eax |
| v19 | 5 | int | eax |
| v20 | 5 | int | eax |
| v21 | 5 | int | eax |
| v22 | 5 | int | eax |
| v23 | 5 | int | eax |
| v24 | 5 | time_t | eax |
| v25 | 5 | int | eax |
| v26 | 5 | int | eax |

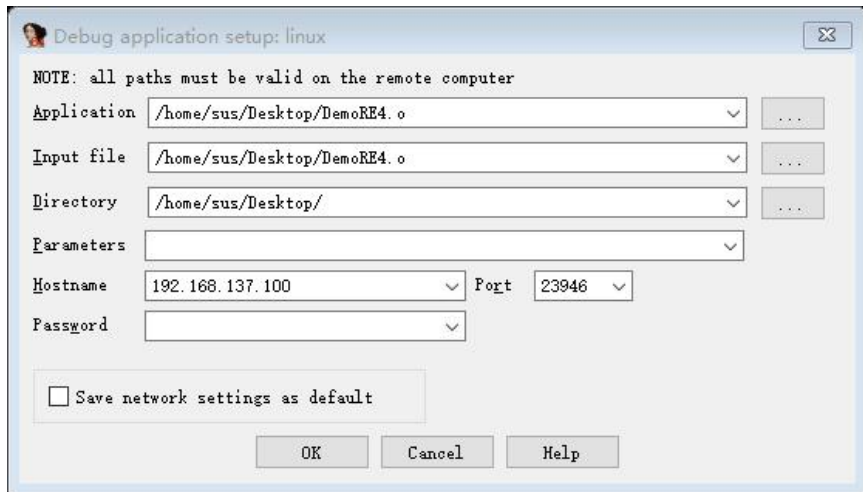


IDA Introduction - Debugger



dbgsrv (Debug Server) 文件夹中是ida提供的远程调试服务器

打开 Debugger -> Process options



Application: 应用程序文件名

Input file: 输入文件名

Directory: 输入文件所在文件夹

Hostname: 远程主机ip

Port: 端口 (默认23946)



IDA Introduction - Patch



SUS

Patch (打补丁) 是指修改原程序指令，使其按照我们希望的方式运行，可以是破解原程序流程，也可以是修补原程序漏洞。

在Hex View，将光标定位至目标位置，按F2进入编辑模式，再按F2退出

在IDA View，将光标定位至目标位置，Edit -> Patch Program -> Assemble，可以修改汇编指令，但要注意指令长度不能超过原指令长度，如果比原指令短，则需要用nop补齐

0x90 nop



IDA Introduction - Handle error



SUS

1. Positive sp value

正常操作：

- 1) 用Option->General->Disassembly, 将选项Stack pointer打勾；
- 2) 仔细观察每条call sub_xxxxxx前后的堆栈指针是否平衡；
- 3) 有时还要看被调用的sub_xxxxxx内部的堆栈情况，主要是看入栈的参数与ret xx是否匹配；
- 4) 注意观察jmp指令前后的堆栈是否有变化；

非正常操作：

Alt+K, 调整sp值，改为一个较大的负值



IDA Introduction - Handle error



SUS

2. Call analysis failed

一般是由于函数调用参数分析出错导致

对于间接调用可以设置调用地址，对于直接调用可以检查调用目标的类型

3. Stack frame is too big (十分罕见)

IDA在分析栈帧时出现错误，可能是由于花指令或是混淆器导致，需要手动检查，并去除花指令。

目录

content

01

IDA Introduction (1)

02

Anti-Reverse

03

Reverse in CTF

04

IDA Introduction (2)

05

Ollydbg

06

Assembly Language



Ollydbg - Views



SUS

- 窗口
 - L -> Log
 - E -> Executables
 - M -> Memory
 - T -> Threads
 - W -> Windows
 - H -> Handles
 - C -> CPU
 - P -> Patches
 - K -> call stacks
 - B -> Breakpoints
 - R -> Reference
 - ... -> run trace

目录

content

01

IDA Introduction (1)

02

Anti-Reverse

03

Reverse in CTF

04

IDA Introduction (2)

05

Ollydbg

06

Assembly Language



Assembly



SUS

- 指令集
 - x86/x64
 - arm
 - mips
- CPU运行方式
 - 执行当前指令->取出下一条指令->执行下一条指令
- 单位
 - BIT (位)
 - BYTE (字节)
 - WORD (字)
 - DWORD (双字 DOUBLE WORD)
 - QWORD (四字 QUAD-WORD)



Assembly - Registers



SUS

- 寄存器是CPU中的存储结构，用于暂存指令、数据、地址
- 通用寄存器
- 指令指针寄存器（EIP）



| 8-byte register | Bytes 5-8 | Bytes 7-8 | Byte 8 |
|-----------------|-----------|-----------|--------|
| %rax | %eax | %ax | %al |
| %rcx | %ecx | %cx | %cl |
| %rdx | %edx | %dx | %dl |
| %rbx | %ebx | %bx | %bl |
| %rsi | %esi | %si | %sil |
| %rdi | %edi | %di | %dil |
| %rsp | %esp | %sp | %spl |
| %rbp | %ebp | %bp | %bpl |
| %r8 | %r8d | %r8w | %r8b |
| %r9 | %r9d | %r9w | %r9b |
| %r10 | %r10d | %r10w | %r10b |
| %r11 | %r11d | %r11w | %r11b |
| %r12 | %r12d | %r12w | %r12b |
| %r13 | %r13d | %r13w | %r13b |
| %r14 | %r14d | %r14w | %r14b |
| %r15 | %r15d | %r15w | %r15b |



Assembly - Registers



SUS

- 标志寄存器代表程序的运行状态。在32位CPU中有32个不同的标志寄存器，我们只关心其中的3个：ZF、OF、CF。在逆向工程中，你了解了标志寄存器就能知道程序在这一步是否会跳转，标志寄存器就是一个标志，只能是0或者1，它们决定了是否要执行某个指令。
- Z-Flag(零标志)
- The O-Flag(溢出标志)
- The C-Flag(进位标志)



Assembly - Segment



SUS

- 内存中的一个段储存了指令(CS)、数据(DS)、堆栈(SS)或者其他段(ES)。每个段都有一个偏移量，在32位应用程序下，这些偏移量由 00000000 到 FFFFFFFF。
- 段偏移地址 + 数据偏移地址 = 相对基址的真实偏移



Assembly - Stack



SUS

- 栈是内存里可以存放稍后会用到的东西的地方。
- 栈可以看做一个FILO的队列
- push命令就是向栈中压入数据，pop命令就是从栈中取出最后放入的数据并且把它存进具体的寄存器中。



Assembly - Instructions



SUS

- 数据传送指令: `mov push pop lea`
- 算术运算指令: `add sub mul div dec inc`
- 位运算指令: `not neg or xor and`
- 条件、跳转指令: `cmp nop call int jmp rep ret`

Thanks for watching