

專注數位化靶場的攻防專家

逆向工程



- 哈希函數——MD5、Sha-1等
- 對稱加密演算法——DES、3DES、AES
- 非對稱加密演算法——RSA



現代密碼學

目錄

CONTENTS

01 逆向工程基礎

02 工具介紹

03 常見手法分析

04 例題講解

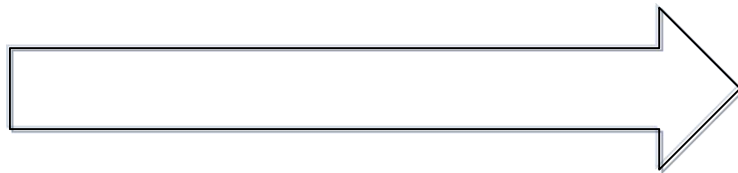


逆向工程基礎

軟體逆向工程（Software Reverse Engineering）又稱軟體反向工程，是指從可運行的程式系統出發，運用解密、反彙編、系統分析、程式理解等多種電腦技術，對軟體的結構、流程、演算法、代碼等進行逆向拆解和分析，推導出軟體產品的源代碼、設計原理、結構、演算法、處理過程、運行方法及相關文檔等。通常，人們把對軟體進行反向分析的整個過程統稱為軟體逆向工程，把在這個過程中所採用的技術都統稱為軟體逆向工程技術。逆向分析者的工作看起來更有趣的原因是，繞過種種保護和限制，找到代碼中的“寶藏”——錯誤、漏洞或者被加密演算法掩蓋的數據結構

可運行的程
序系統

解密、反彙編、系統分析、程式理解



源代碼 設計原理結構
演算法
處理過程運行方法相
關文檔
...

總得想個辦法守住自己的飯碗



Win10激活工具 Win10激活码 Win10密钥key Win10镜像之家



3天前 Win10激活栏目收集了最新Win10激活工具、Win10激活密钥key、随着Win10版本的不断更新,Win10之家为大家提供了最热门的Win10激活工具,可以永久免费激活Win10专业版系统。
www.win10com.com/win10jihuo/ 百度快照

其他人还在搜

- 激活饮料 苹果激活软件 激活饮料为什么没有了 oppo手机怎么查询激活
- win10在线激活 系统激活工具win7免费 windows10激活教程 怎么激活windows10

gopojie.com 百度快照

破解- 异次元软件世界



2021年5月26日 免破解无需会员下载提速/客户端加速设置 Win, Mac, Linux 2020-07-4 如今国内网盘其实越来越少,像 115 网盘、奶牛快传、城通网盘、OneDrive 等虽然各有特点,但如果说到市场占有率和...

www.iplaysoft.com/tag/破解/ 百度快照

共享之家-原破解帝国 最专业的破解版软件下载站。



06-01[其它软件] 印章制作大师V11.0绿色破解版 06-10[金蝶] 金蝶12.1专业版正版注册机.无限 06-16[其它软件] ReadyFor4GB 05-18[其它软件] 浪潮软件注册机 06-09[用友] 用友T+13.0数据字典 06-16[...

...om/ 百度快照

[原创工具] [免登录]百度网盘满速下载工具 - 52破解 (复制链接)

发表于 2018-5-30 13:58 | 只看该作者

本帖最后由 菩提叶 于 2018-9-18 21:57 编辑

【功能介绍】

1、百度网盘不限速下载 (正常情况下都可以满宽带)



全 安卓破解软件下载 手游神器...

下载,盒子集合破解版软件,内购破解游戏,无限金币破解版



假設你是一個伺服器的管理員

某天發現你的伺服器上多了個程式，所有應用文件都被加密了

桌面上的檔提示你中病毒了

我們如何修復我們的系統，還原加密的檔



Windows



- ◆ PE (Portable Executable) 可移植的可執行檔
- ◆ 尾碼: exe、dll、sys

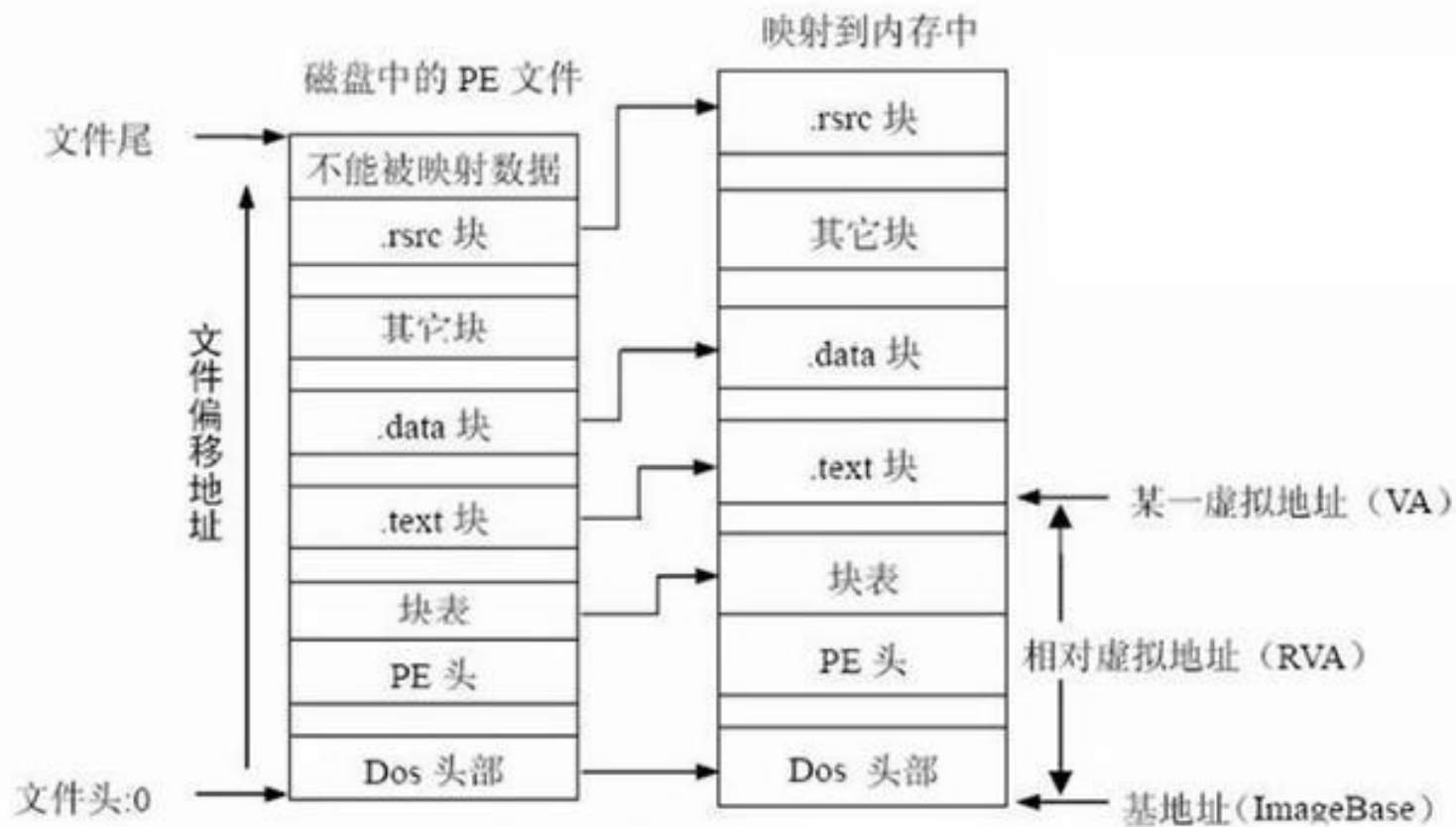
```

WinRAR_3.93_SC.exe
Edit As: Hex Run Script Run Template
0 1 2 3 4 5 6 7 8 9 A B C D E F 0123456789ABCDEF
0000h: 4D 5A 50 00 02 00 00 00 04 00 0F 00 FF FF 00 00 MZP.....vÿ..
0010h: B8 00 00 00 00 00 00 00 40 00 1A 00 00 00 00 00 .....@.....
0020h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0030h: 00 00 00 00 00 00 00 00 00 00 00 00 00 02 00 00 .....
0040h: BA 10 00 0E 1F B4 09 CD 21 B8 01 4C CD 21 90 90 °....'í!..Lí!..
0050h: 54 68 69 73 20 70 72 6F 67 72 61 6D 20 6D 75 73 This program mus
0060h: 74 20 62 65 20 72 75 6E 20 75 6E 64 65 72 20 57 t be run under W
0070h: 69 6E 33 32 0D 0A 24 37 00 00 00 00 00 00 00 00 in32..$7.....
0080h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0090h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00A0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00B0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00C0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00D0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00E0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00F0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0100h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0110h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0120h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0130h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0140h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0150h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0160h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0170h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0180h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0190h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
01A0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
01B0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
01C0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
01D0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
01E0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
01F0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0200h: 50 45 00 00 4C 01 03 00 08 BE B4 3F 00 00 00 00 PE.L...%?....
0210h: 00 00 00 00 E0 00 0F 01 0B 01 05 00 00 E0 00 00 .....
0220h: 00 20 00 00 00 60 01 00 C0 48 02 00 00 70 01 00 .....
0230h: 00 50 02 00 00 00 40 00 00 10 00 00 00 02 00 00 .....
0240h: 04 00 00 00 00 00 00 00 04 00 00 00 00 00 00 00 .....
0250h: 00 70 02 00 00 10 00 00 00 00 00 00 02 00 00 00 .....

```

PE檔結構

- .rsrc
 - 模組的資源資訊
- .data
 - 變數資訊
- .text
 - 可執行代碼段



現在換個角度，你是破解者

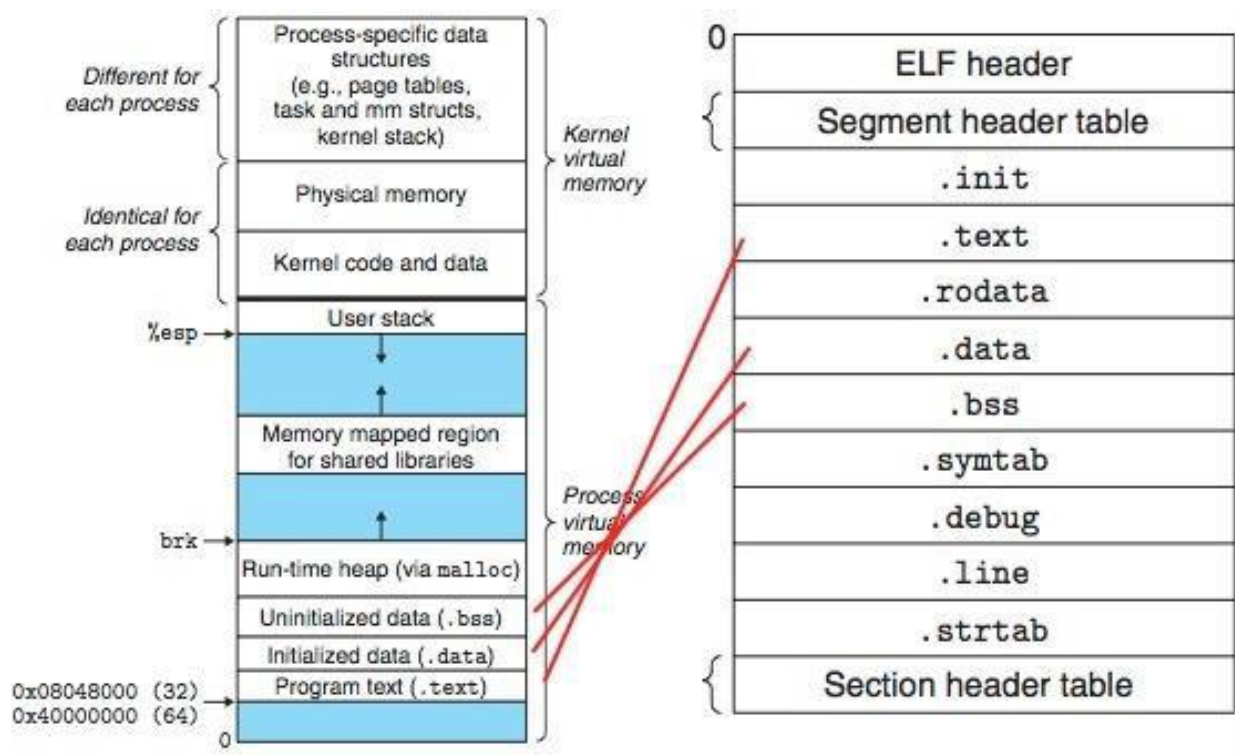
Linux

- ELF (Executable and Linkable Format) 可執行和可鏈接檔
- 尾碼: 0, so, elf



ELF檔結構

```
wifi_dnld.elf
Edit As: Hex Run Script Run Template
0 1 2 3 4 5 6 7 8 9 A B C D E F 0123456789ABCDEF
0000h: 7F 45 4C 46 01 02 01 00 00 00 00 00 00 00 00 00 .ELF.....
0010h: 00 02 18 AD 00 00 00 01 80 00 00 00 00 00 00 34 ...-....€.....4
0020h: 00 02 B0 D0 00 00 00 00 00 34 00 20 00 05 00 28 ..°Ð.....4. ... (
0030h: 00 15 00 12 00 00 00 01 00 00 04 00 80 00 00 00 .....€...
0040h: 80 00 00 00 00 02 90 2C 00 02 90 30 00 00 00 07 €. ...., ...0....
0050h: 00 00 04 00 00 00 00 00 00 00 00 00 00 00 00 04
```

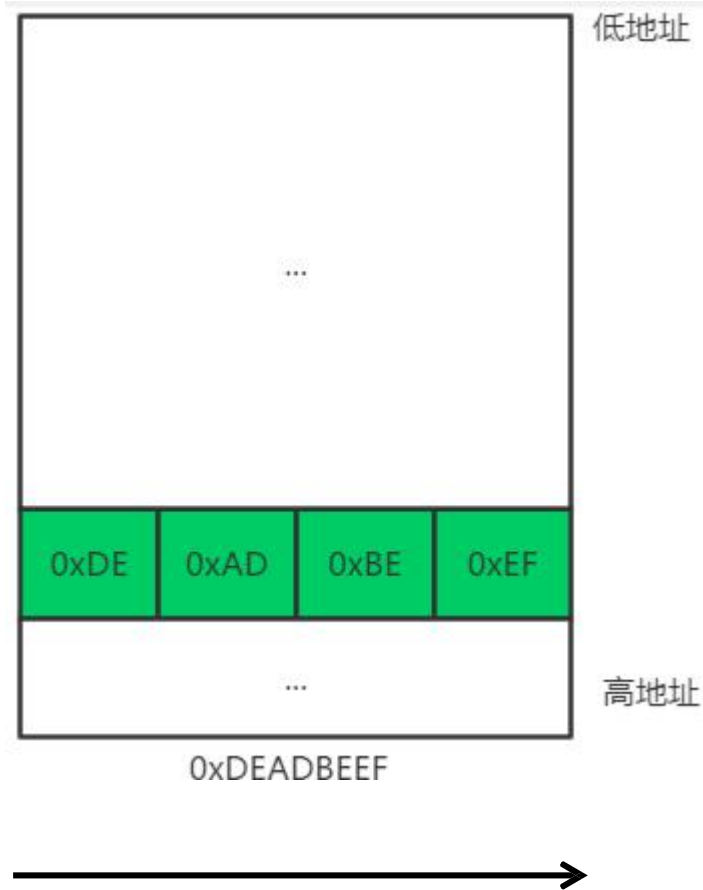


- **.text**
 - 已編譯程序的機器代碼
- **.symtab**
 - 符號表它存放在程式中被定義和引用的函數和全局變數的資訊
- **.strtab**
 - 一個字串表，其內容包括.symtab和.debug節中的符號表
- **.bss**
 - 存程式中未初始化的全局變數和靜態變數的一塊記憶體區域。特點是可讀寫的



Big-Endian (大端)

- Arch
 - MIPS
 - ARM
- 高位元組保存在內存的低地址中，低位元組保存在內存的高地址中



彙編:

```
mov eax,[A]
mov ebx,[B]
add eax,ebx
```



```
seg000:00000000
seg000:00000000
seg000:00000004
seg000:00000008
seg000:00000009
seg000:0000000A
seg000:0000000B
seg000:0000000C
seg000:0000000D
seg000:0000000E
seg000:0000000F
seg000:00000010
seg000:00000011
seg000:00000012
seg000:00000013
seg000:00000014
seg000:00000014 seg000
```

```
assume cs:seg000
assume es:nothing, ss:nothing, ds:nothing, fs:nothing, gs:nothing
dd 0DEADBEEFh
dd 19260817h
db 0A1h
db 0
db 0
db 0
db 8Bh
db 1Dh
db 4
db 0
db 0
db 1
db 0D8h
ends
```

組合語言中的每一條指令，本質上來說都只是【助記符】

每一條【助記符】都有其對應的機器碼



數據傳輸指令

- **MOV**

- MOV eax, [ESP]

- **PUSH**

- **POP**

- **LEA**

- LEA eax, [0xABCD]

算術運算指令

- **ADD**

- **INC**

- **SUB**

- **DEC**

- **MUL/IMUL**

- **DIV/IDIV**

邏輯運算指令

- **AND**

- **OR**

- **XOR**

- **NOT**

- **SHL**

- **SHR**



程式轉移指令

- 無條件轉移

- JMP

- CALL

- RET/RETN

- 有條件轉移

- J[Condition]

- JG/JNLE 大於轉移.

- JGE/JNL 大於或等於轉移.

- JL/JNGE 小於轉移.

- JLE/JNG 小於或等於轉移

返回指令

- RET

- POP eip

- LEAVE

- MOV esp, ebp

- POP ebp



X86架構

- 4個數據寄存器(EAX、EBX、ECX和EDX)
- 2個變址和指針寄存器(ESI和EDI) 2個指針寄存器(ESP和EBP)
- 6個段寄存器(ES、CS、SS、DS、FS和GS)
- 1個指令指針寄存器(EIP) 1個標誌寄存器(EFlags)



X86_64架構

- 12個數據寄存器(RAX、RBX、RCX、RDX、R8-R15)
- 2個變址和指針寄存器(RSI和RDI) 2個指針寄存器(RSP和 RBP)
- 6個段寄存器(ES、CS、SS、DS、FS和GS)
- 1個指令指針寄存器(RIP) 1個標誌寄存器(RFlags)



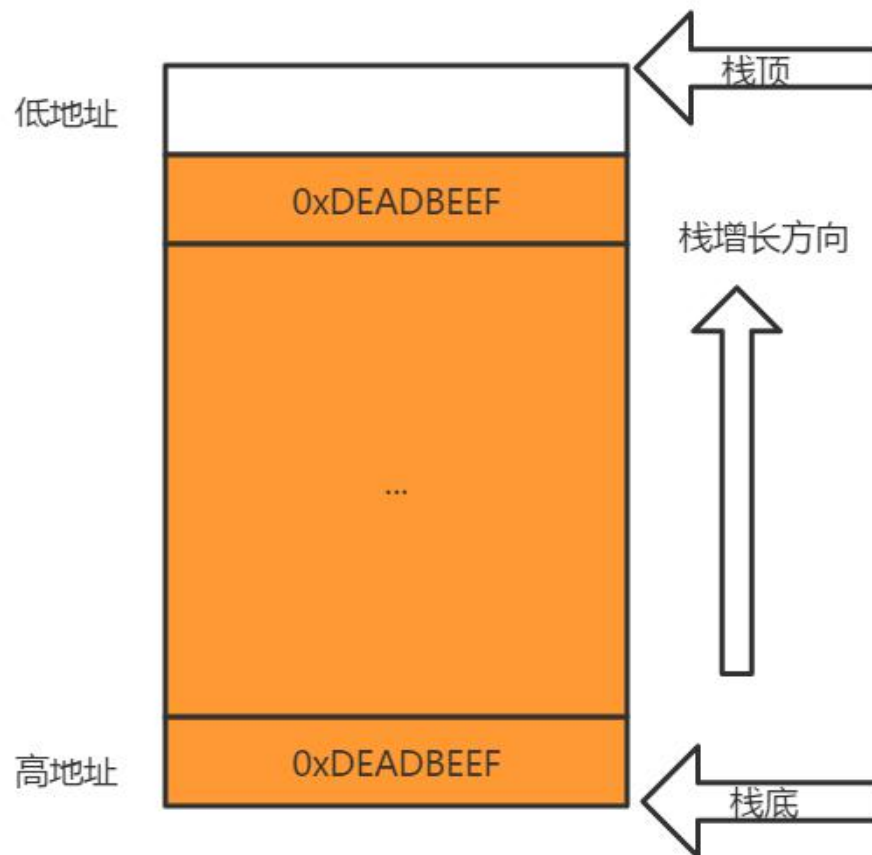
彙編書寫格式

➤ Intel

1. `push ebp`
2. `mov ebp,esp`
3. `sub esp,0x10`
4. `mov DWORD PTR [ebp-0x4],0x0`
5. `mov DWORD PTR [ebp-0x4],0x3`
6. `mov eax,0x0`
7. `leave`
8. `ret`

棧 (stack)

- 數據結構
 - (LIFO, Last In First Out, 後進先出)
- 存儲內容
 - 變數, 函數調用資訊
- 棧增長方向
 - 高地址 -> 低地址



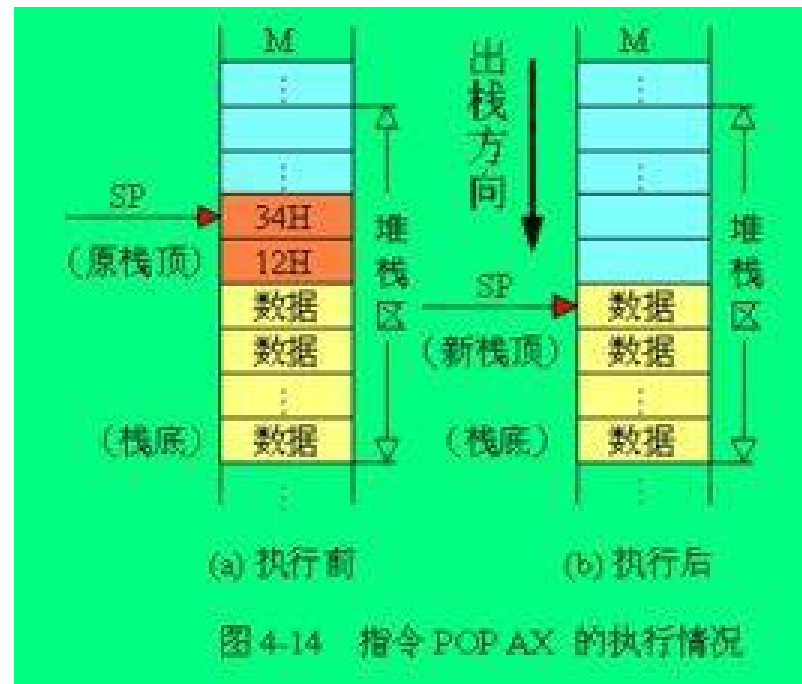


棧操作相關寄存器

- ESP(RSP, x86_64)
 - Extended Stack Pointer
 - 當前棧頂指針
- EBP(RBP, x86_64)
 - Extended Base Pointer
 - 系統棧最上層的棧的底部

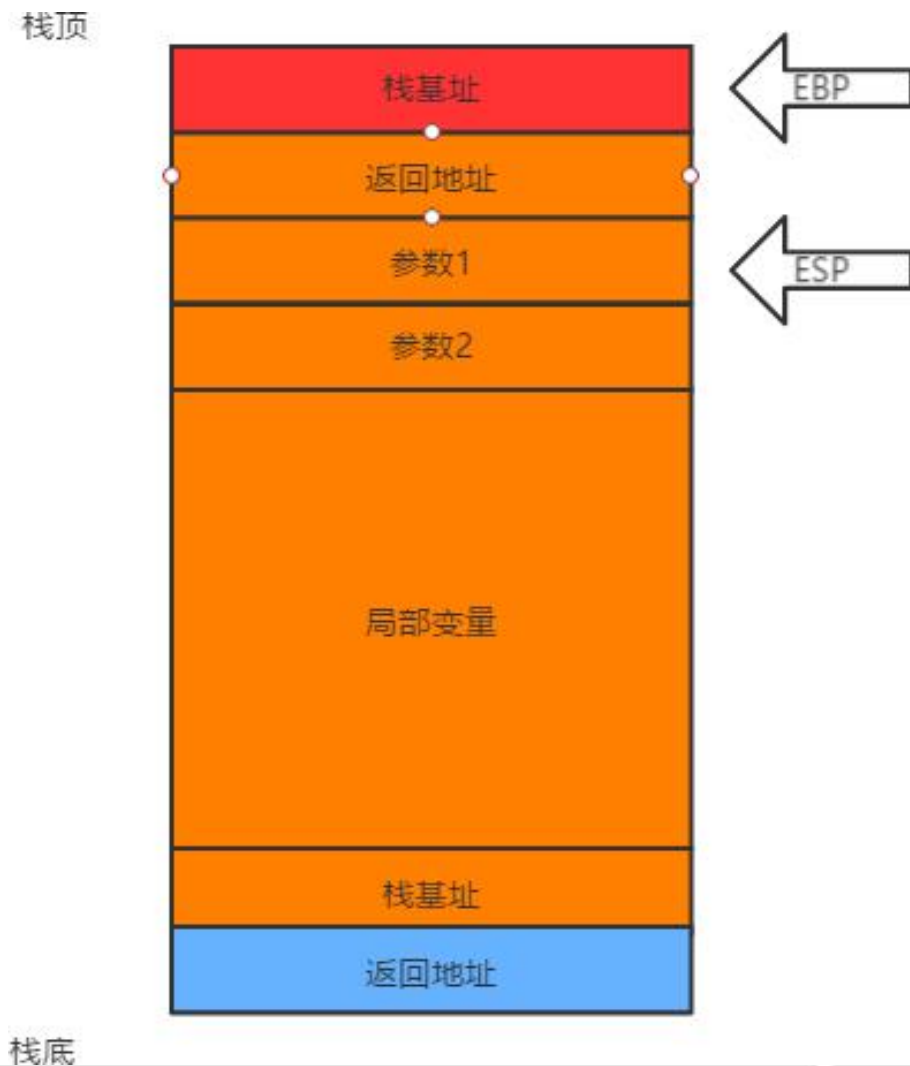
常用棧操作

- PUSH eax
 - SUB esp, 4
 - MOV [esp], eax
- POP eax
 - MOV eax, [esp]
 - ADD esp, 4

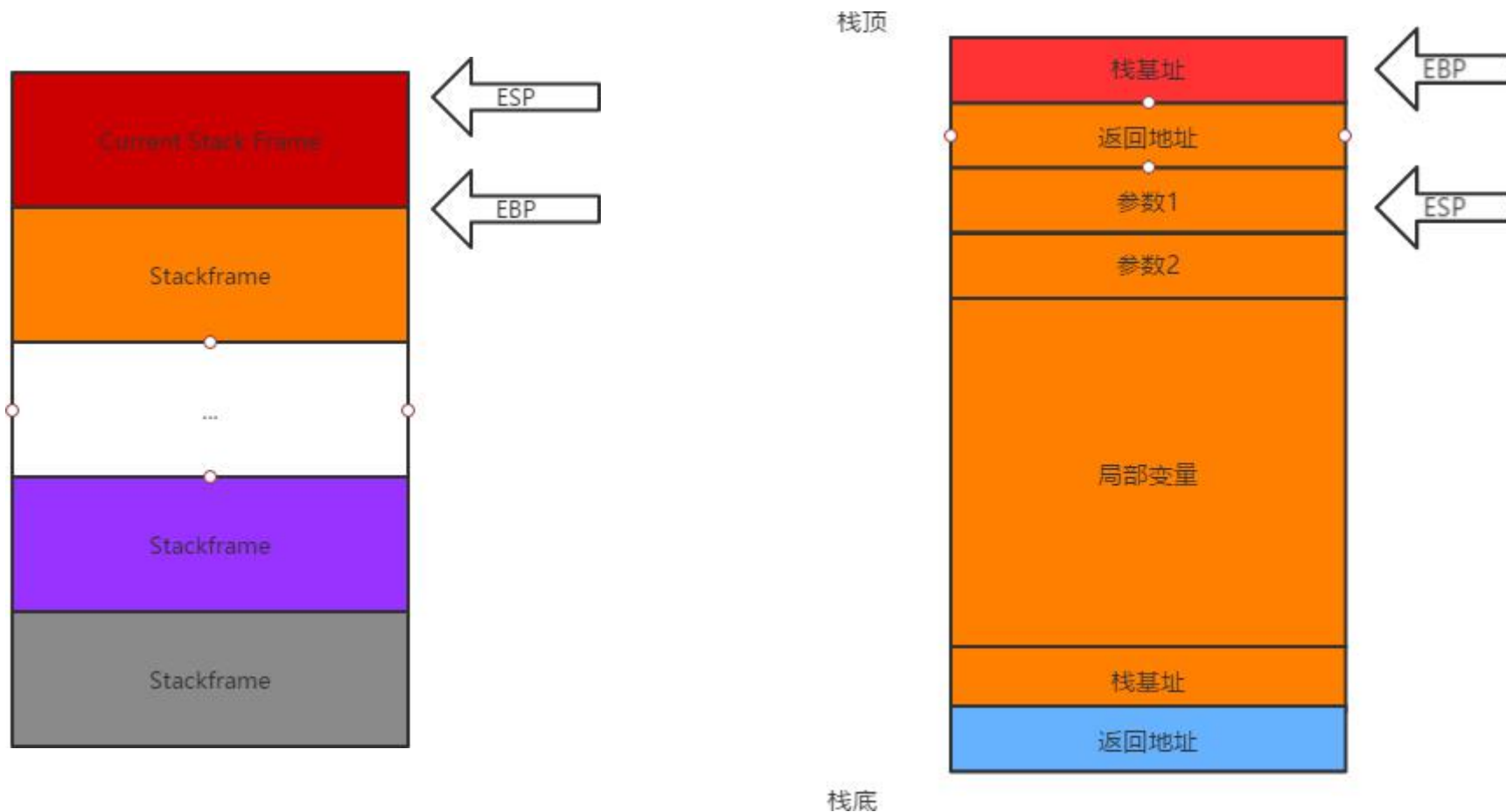


棧幀 (stack frame)

- 棧幀也叫過程活動記錄，是編譯器用來實現過程/函數調用的一種數據結構。



棧幀





堆特點

- 1.堆是程式運行時動態分配的記憶體。
- 2.堆在使用時需要程式員使用專用的函數進行申請，如C語言中的 `malloc` 等函數、C++中的 `new` 函數等。
- 3.一般用堆指針來使用申請的記憶體
- 4.使用完畢後要通過堆釋放函數進行回收，如 `free,delete` 等

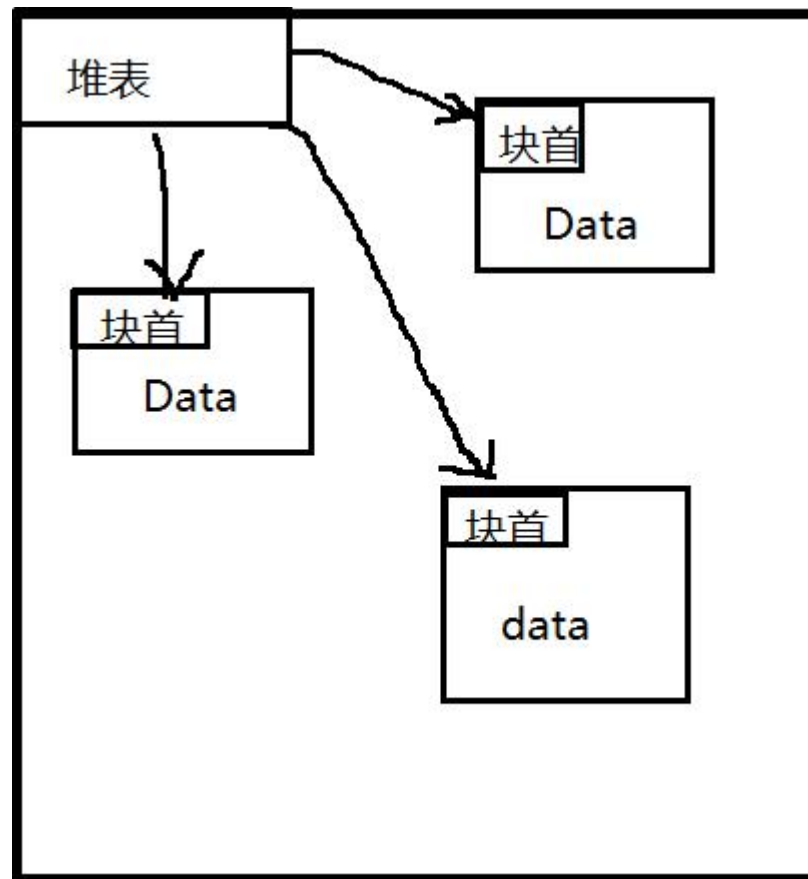


堆數據結構

- 堆塊
 - 堆區內存按不同大小組織成塊，以堆塊為單位進行標識
 - 包括兩個部分：塊首和塊身。
 - 塊首：堆塊頭部的幾個位元組，用來標識這個塊首自身的信息
 - 塊身：在塊首後面的部分，分配給用戶使用的數據區

堆數據結構

- 堆表：位於堆區的起始位置，用於檢索堆區中所有堆塊主要資訊。
- 堆塊分配
- 堆塊釋放
- 堆塊合併
 - 系統發現物理地址相連的堆塊都未使用，則合併這兩個堆塊為一個堆塊





現在換個角度，你是破解者

堆與棧對比

• Heap

- 運行時動態分配
- 程式員申請
- 程式員回收
- 地址空間內 “雜亂的”

□ Stack

- 預先分配
- 系統自動分配
- 系統自動回收
- 地址空間內 “整齊的”



函數傳參規則

- x86
 - 所有參數都在棧上
 - 函數返回值在eax寄存器中
- x86_64
 - 函數參數依次存在rdi、rsi、rdx、rcx、r8、r9中
 - 若參數超過6個的話，之後的參數存在棧上
 - 函數返回值在rax寄存器中



常見函數調用方式

- `_stdcall`
 - Windows API 默認的函數調用規則
 - 函數參數由右向左入棧
 - 函數調用結束後由被調用函數清除棧內數據
- `_cdecl`
 - C/C++ 默認的函數調用規則
 - 函數參數由右向左入棧
 - 函數調用結束後由調用者函數清除棧內數據
- `_fastcall`
 - 適用於對性能要求較高的場合



逆向工具



- 逆向工程

- 以反彙編閱讀源碼的方式去推斷其數據結構、體系結構和程式設計資訊

- 方法：

- 1. 通過資訊交換所得的觀察。

- 2. 反彙編

- 3. 反編譯



什麼是逆向工程?

動態調試工具——OllyDebug

The screenshot displays the OllyDbg interface with the following components:

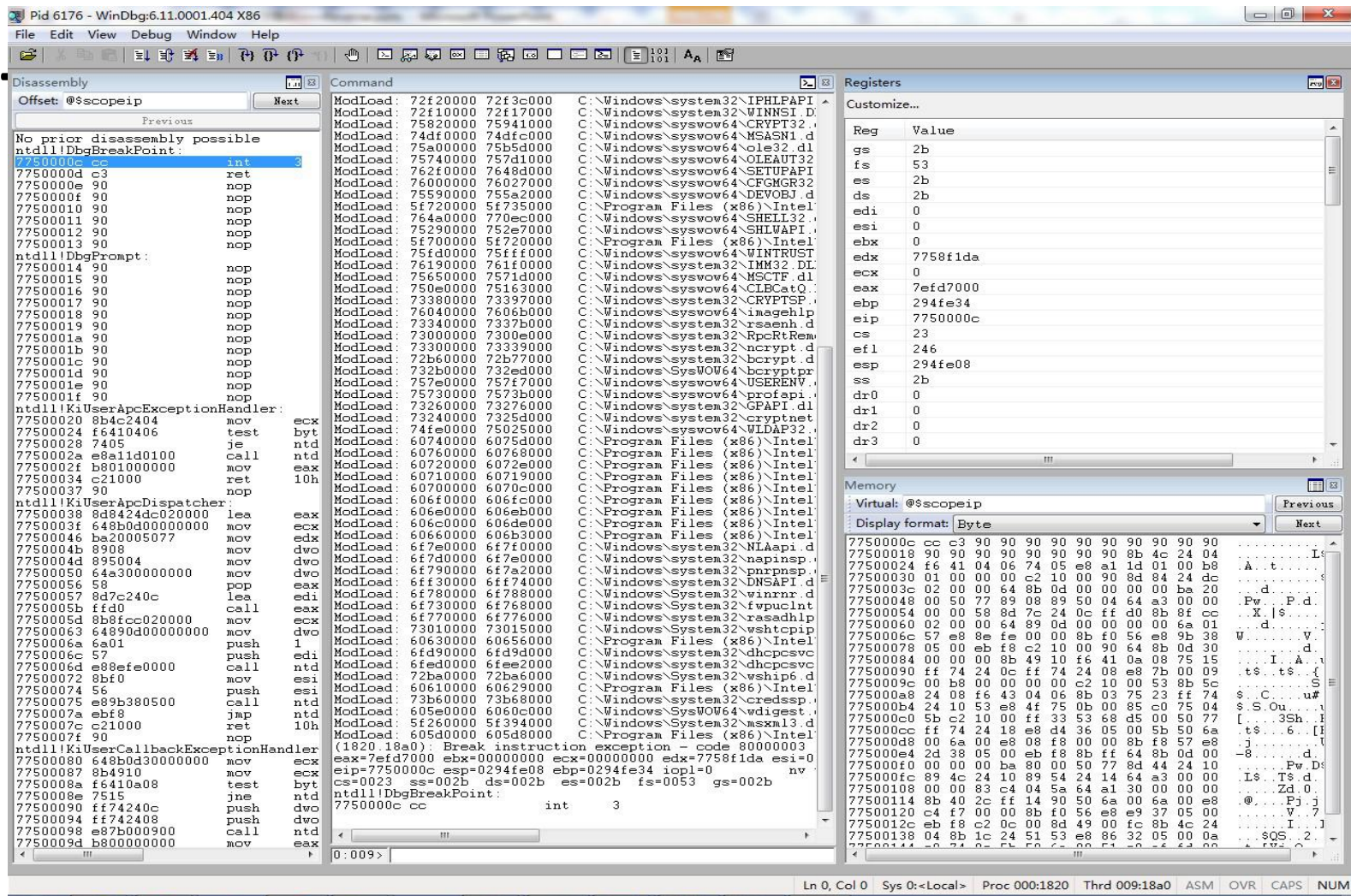
- Code Area (代碼區):** Located in the upper-left pane, showing assembly instructions such as `jmp short 00401012`, `mov eax, dword ptr ds:[0x4B011B]`, and `call < jmp.&KERNEL32.GetModuleHandleA >`.
- Registers Area (寄存器區):** Located in the upper-right pane, displaying the state of CPU registers including EAX, ECX, EDX, and EIP.
- Stack Area (棧區):** Located in the lower-right pane, showing the current stack frame with return addresses like `RETURN to kerne132.7518336A`.
- Disassembly Pane:** The central pane shows the assembly code with comments and hex values.
- Hex View Pane:** The lower-left pane shows the raw hex data of the code.



什麼是逆向工程?

動態調試工具一 Windbg

- 可進行內核調試





什麼是逆向工程?

靜態調試工具——IDA

- 32/64都適用
- 偽代碼查看
- 可以遠程動態調試

```
IDA View-A | Pseudocode-A | Hex View-1 | Structures
1 void __thiscall sub_18001F10(void *this, int a2, int a3)
2 {
3     void *v3; // edi@1
4     int v4; // ebp@1
5     unsigned int v5; // eax@4
6     int v6; // eax@5
7     int v7; // ebx@7
8     int v8; // eax@8
9     int v9; // edi@8
10    int v10; // ecx@10
11    unsigned int v11; // [sp+28h] [bp+4h]@7
12
13    v3 = this;
14    v4 = 0;
15    if ( a3 )
16    {
17        if ( *(_DWORD *)a3 )
18        {
19            sub_180018A0(a3);
20            qfree(*(_DWORD *)a3);
21            *(_DWORD *)a3 = 0;
22            *(_DWORD *)(a3 + 8) = 0;
23        }
24        v5 = *((_DWORD *)v3 + 7);
25        if ( a2 >= v5 )
26            v6 = v5 - 1;
27        else
28            v6 = a2;
29        v11 = 0;
30        v7 = *(_DWORD *)v3 + 6 + 272 * v6;
31        if ( *(_DWORD *)(v7 + 4) > 0u )
32        {
33            do
```

00001310 sub_18001F10:1



什麼是逆向工程?

動態調試工具——gdb

- Linux
- 添加pwndbg插件
後有代碼著色

```
gdb-peda$ start
[-----registers-----]
EAX: 0xbffff7f4 --> 0xbffff916 ("/root/a.out")
EBX: 0xb7fcbff4 --> 0x155d7c
ECX: 0xd5eaa03
EDX: 0x1
ESI: 0x0
EDI: 0x0
EBP: 0xbffff748 --> 0xbffff7c8 --> 0x0
ESP: 0xbffff748 --> 0xbffff7c8 --> 0x0
EIP: 0x80483e7 (<main+3>:      and     esp,0xffffffff)
EFLAGS: 0x200246 (carry PARITY adjust ZERO sign trap INTERRUPT direction overflow)
[-----code-----]
0x80483e3 <frame_dummy+35>: nop
0x80483e4 <main>:      push   ebp
0x80483e5 <main+1>:    mov    ebp,esp
=> 0x80483e7 <main+3>:  and   esp,0xffffffff
0x80483ea <main+6>:    sub   esp,0x110
0x80483f0 <main+12>:   mov   eax,DWORD PTR [ebp+0xc]
0x80483f3 <main+15>:   add   eax,0x4
0x80483f6 <main+18>:   mov   eax,DWORD PTR [eax]
[-----stack-----]
0000| 0xbffff748 --> 0xbffff7c8 --> 0x0
0004| 0xbffff74c --> 0xb7e8cbd6 (<_libc_start_main+230>:      mov    DWORD PTR [e
0008| 0xbffff750 --> 0x1
0012| 0xbffff754 --> 0xbffff7f4 --> 0xbffff916 ("/root/a.out")
0016| 0xbffff758 --> 0xbffff7fc --> 0xbffff922 ("SHELL=/bin/bash")
0020| 0xbffff75c --> 0xb7fe1858 --> 0xb7e76000 --> 0x464c457f
0024| 0xbffff760 --> 0xbffff7b0 --> 0x0
0028| 0xbffff764 --> 0xffffffff
[-----]
Legend: code, data, rodata, value

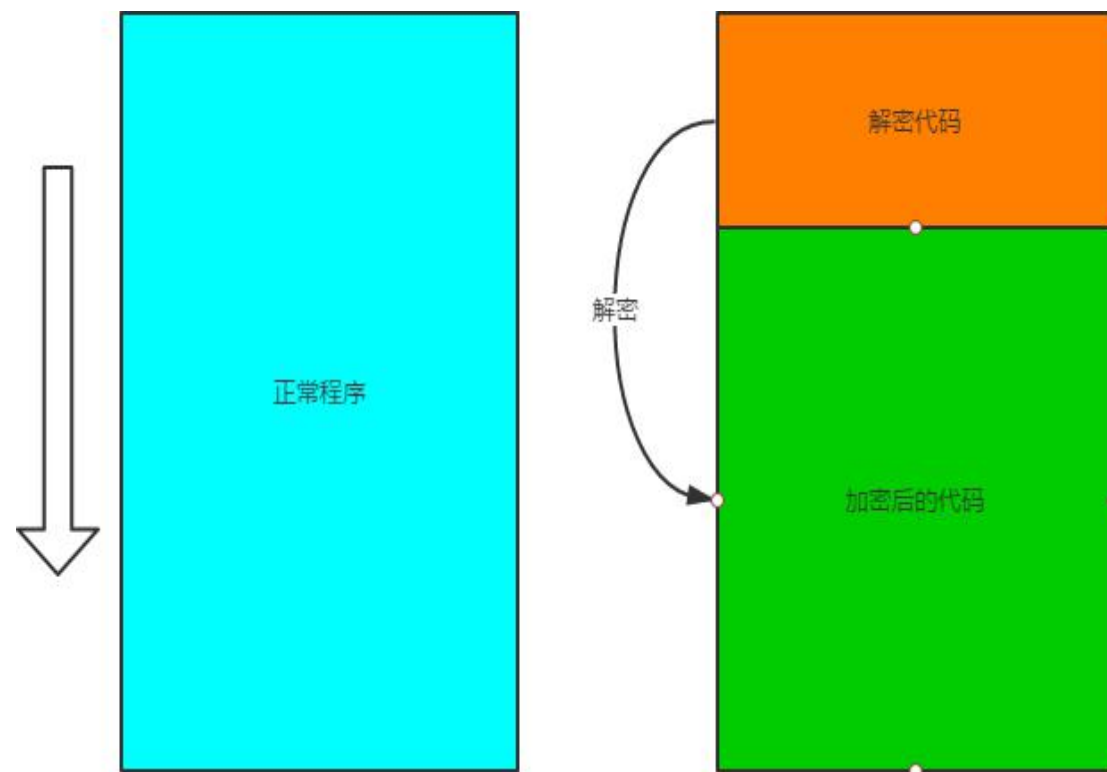
Temporary breakpoint 1, 0x080483e7 in main ()
gdb-peda$
```



逆向知識

關於“殼”

- 壓縮殼：壓縮資源，減小體積
 - ASPack、UPX、PECompact
- 加密殼：提供保護，提供功能
 - ASProtect、Armadillo、EXECryptor
- 虛擬機殼：提供增強保護
 - VMProtect、Themida





殼的工作流程

程式從被加密/壓縮狀態到可運行狀態的過程，

可以類比為金蟬脫殼

在蟬脫出殼之前，我們無從得知殼中生物的真面貌，

我們只能看到醜陋的蟬外殼

蟬脫出殼之後，

原先的殼（解壓代碼）被丟棄，

其中的生物顯露出其中的真面目（原本的功能性代碼）

The screenshot shows a debugger window with the following content:

```

Function name
FindPESectionExec
GetPEImageBase
IsNonwritableInCurr
mingw_enum_imp
mingwthr_run_key
w64_mingwthr_ad
w64_mingwthr_rei
mingw_TLScallback
chkstk ms
C_specific_handler
_set_app_type
_getmainargs
_get_invalid_paramet
_set_invalid_paramet
malloc
strlen
memcpy
_cexit
_amsg_exit
_initterm
exit
puts
scanf
Line 46 of 98
Graph overview

```

```

1 int __cdecl main(int argc, const char **argv, const char **envp)
2 {
3     char Str[44]; // [rsp+20h] [rbp-30h]
4     int i; // [rsp+4Ch] [rbp-4h]
5
6     _main();
7     puts("password:");
8     scanf("%32s", Str);
9     if ( strlen(Str) != 20 )
10    {
11        puts("fail!");
12        system("pause");
13        exit(0);
14    }
15    for ( i = 0; i <= 19; ++i )
16    {
17        if ( Str[i] != flag[i] )
18        {
19            puts("fail!");
20            system("pause");
21            exit(0);
22        }
23    }
24    puts("success!");
25    system("pause");
26    return 0;
27 }

```

```

; use64

thing, ds:UPX0, fs:nothing, gs:nothing

5CE719F0D17A7h, 2012700007D3500h

; DATA XREF: start+4↓
3B4828EC8348h, 0D231FDED99930A34h
32BF0C20720797h, 5A4D38FBB2FB7F81h
774C085008B055Fh, 0E82502B9DD6FEE76h
CDDD9361C060Bh, 32FE090289481815h
32D108940129B08h, 38830A102CBA6FEDh
FFB76E5601B9C3h, 3C4863489066A7EBh
5472EFB76DF685h, 45320BF9951848B7h
3B0C860F0EF0B8h, 0C2950FC985F000F8h
C68C065F05790Dh, 44367478837083D8h
3E6B4384F38DFD6h, 5EA6058D4CC59BEFh
11C0E7805D976F6h, 0B7DB3B2F20244489h
FC33876FFF8477Ah, 5356575554415541h
dq 0B67A2D7498EC8148h, 54AC0DB999337636h, 46EF6DFDBD4D363Dh
dq 4EBC995AB48F3D7h, 0FDBF30250426658Ch, 7006317B1D2E8F7Fh
dq 7082258B4CFF3108h, 0EDFEC6394811EB0Ch, 0E8B92C27840FDDEDh

```



壓縮殼

縮減 PE 檔的大小，隱藏了 PE 檔內部代碼和資源，便於網路傳輸和保存。

通常壓縮殼有兩類用途

- 單純用於壓縮普通 PE 檔的壓縮殼
- 對原始檔案進行較大變形，嚴重破壞 PE 檔頭，經常用於壓縮惡意程式

常見的壓縮殼有：**Upx**、ASpack、PECompat



加密殼

加密殼或稱保護殼，應用有多種防止代碼逆向分析的技術，最主要的功能是保護 PE 免受代碼逆向分析。加密殼保護的 PE 程式通常比原文件大得多。目前加密殼大量用於對安全性要求高，對破解敏感的應用程式，同時也有惡意程式用於避免（降低）殺毒軟體的檢測查殺。

常見的加密殼有：ASProtector、Armadillo、EXECryptor、Themida、VMProtect

- ESP定律
 - 試用範圍：壓縮殼，部分加密殼
- 原理
 - 如果我們要返回父程式，則當我們在堆疊中進行堆疊的操作的時候，一定要保證在RET這條指令之前，ESP指向的是我們壓入棧中的地址。這也就是著名的“堆疊平衡”原理
 - Pushad/Popad等



壓縮殼

壓縮程式體積，順帶保護代碼

UPX是一個開源的，免費的可執行程式壓縮殼。

作者團隊仍然在維護這個專案。目前保持一年一次到兩次更新。

如何下載：百度一下UPX，搜索結果第一個便是UPX的官網。

如何使用：根據自己的操作系統選用對應的UPX軟體。任意一個平臺上的UPX都可以對所有平臺（Windows, linux, dos, arm等）的可執行程序進行加殼/脫殼。

加殼指令：`./upx -9 ./input_file`

脫殼指令：`./upx -d ./packed_file`

The screenshot shows the UPX website homepage. At the top, the UPX logo is displayed in a stylized font. Below the logo, the text reads "the Ultimate Packer for eXecutables". There are two buttons: "View source on GitHub" and "Download latest release". The main content area has a "Welcome" section with a green heading. Below it, a paragraph describes UPX as a free, portable, extendable, high-performance executable packer. A link to the Wikipedia entry is provided. The "Blog Posts" section follows with a green heading and a list of recent releases and a moving announcement.

UPX

the Ultimate Packer for eXecutables

[View source on GitHub](#) [Download latest release](#)

Welcome

UPX is a free, portable, extendable, high-performance **executable packer** for several executable formats.

Please also see the [Wikipedia entry](#) for some more background info.

Blog Posts

- 23 Jan 2020 » [UPX 3.96 released](#)
- 26 Aug 2018 » [UPX 3.95 released](#)
- 12 May 2017 » [UPX 3.94 released](#)
- 29 Jan 2017 » [UPX 3.93 released](#)
- 11 Dec 2016 » [UPX 3.92 released](#)
- 01 Sep 2016 » [Moving to GitHub](#)



壓縮程式體積，順帶保護代碼

```
gstalker@ubuntu:~/Desktop$ ./upx -9 ./babyXOR.exe
Ultimate Packer for eXecutables
Copyright (C) 1996 - 2020
UPX 3.96      Markus Oberhumer, Laszlo Molnar & John Reiser   Jan 23rd 2020

      File size      Ratio      Format      Name
-----
131367 ->  67367  51.28%  win64/pe  babyXOR.exe

Packed 1 file.
```

UPX是一個壓縮殼

UPX的主要功能是壓縮可執行程式，我們嘗試用UPX加密一個程式時會有上面這個圖中的輸出

指令：`./upx -9 ./executablefile`

-9是盡可能壓縮的意思



加密殼或稱保護殼，應用有多種防止代碼逆向分析的技術，最主要的功能是保護 PE 免受代碼逆向分析。加密殼保護的 PE 程式通常比原文件大得多。目前加密殼大量用於對安全性要求高，對破解敏感的應用程式，同時也有惡意程式用於避免（降低）殺毒軟體的檢測查殺。

常見的加密殼有：ASProtector、Armadillo、EXECryptor、Themida、VMProtect



無法脫殼情況

- 帶殼調試
- 關鍵API下中斷點分析
 - Windows 上的MFC程式
 - 彈窗
 - MessageBox(A/W)
 - 讀寫檔
 - ReadFile/WriteFile/CreateFile
 - 反調試退出
 - ExitProcess

花指令

- 作用
 - 干擾逆向分析人員分析程式
- 類別
 - 可執行的
 - 不執行的
- 解決方法
 - 使用OD插件去除花指令

反調試技術

- 查找通用調試器
- 檢測專用調試器
- 檢測中斷點
- 檢測跟蹤
- 檢測補丁



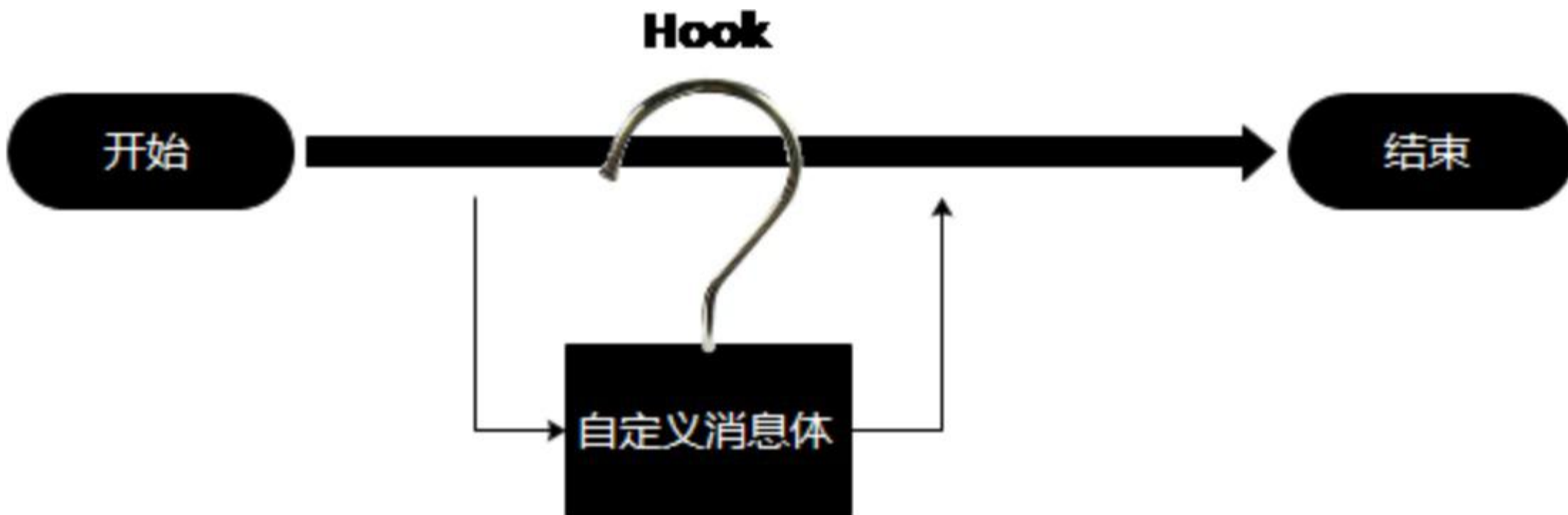
常用反調試函數

- IsDebuggerPresent
- TLS_callback
- ptrace

- 解決方法：
 - 找到函數調用者

 - 刪除反調試函數

- 使用技術手段在運行時動態的將額外代碼依附現進程，從而實現替換現有處理邏輯或插入額外功能的目的。





什麼是虛擬機保護

自定義的指令集

代碼模仿cpu: 解釋器

1. 破解其指令結構和指令含義。
2. 還原指令代碼
3. 寫出逆向代碼

```
5 while ( 1 )
6 {
7     result = e_ip;
8     if ( e_ip >= a2 )
9         return result;
10    switch ( codes[e_ip] )
11    {
12    case 1:
13        stack_mem[e_sp] = e_ax;           // push eax,inc si
14        ++e_ip;
15        ++e_sp;
16        ++e_si;
17        break;
18    case 2:
19        e_ax = codes[e_ip + 1] + usr_input[e_si]; // eax = usr_input[esi] + op[1]
20        e_ip += 2;
21        break;
22    case 3:
23        e_ax = usr_input[e_si] - LOBYTE(codes[e_ip + 1]); // eax = usr_input[esi] - op[1]
24        e_ip += 2;
25        break;
26    case 4:
27        e_ax = codes[e_ip + 1] ^ usr_input[e_si]; // eax = usr_input[esi] ^ op[1]
28        e_ip += 2;
29        break;
30    case 5:
31        e_ax = codes[e_ip + 1] * usr_input[e_si]; // eax = usr_input[esi] * op[1]
32        e_ip += 2;
33        break;
34    case 6:
35        ++e_ip;
36        break;
37    }
```

相關技巧

- 快速定位key比對函數
 - 使用OD字串搜索插件
- 使用F5插件
 - 快速理解加密/解密函數
- 使用脫殼機
 - 儘量使用脫殼機節省時間

C#, Java, Python 逆向

- C#
 - ILSPY
 - .Net Reflector & reflexil
- Java
 - JD-GUI
- Python
 - Uncompyle6
- 方法
 - 搜索源碼

安卓逆向

- 基本套路
- .apk->.dex->jar->源碼
- rar解壓得到dex檔
- dex2jar將.dex->.jar
- 最後使用jd-gui查看源碼