逆向工程

現代密碼學

- 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

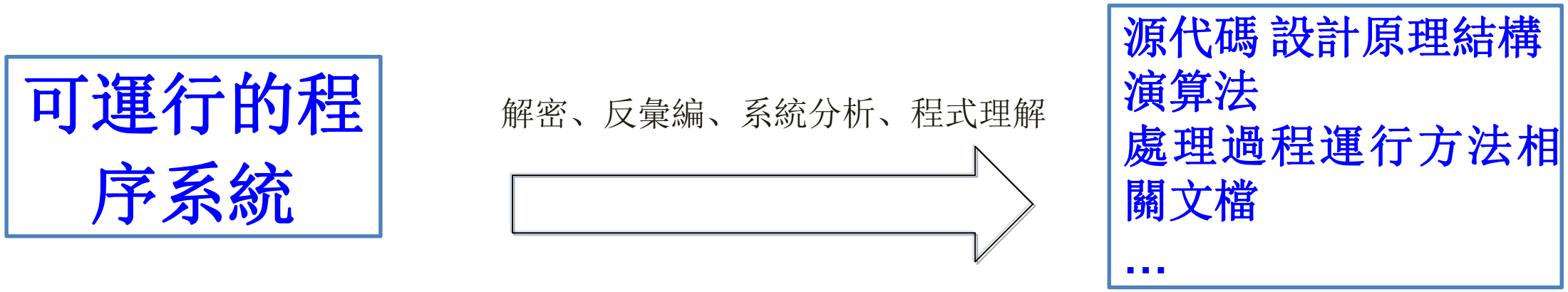
現代密碼學

目 錄
CONTENTS

逆向工程基礎

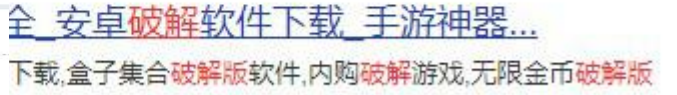
引入

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



保護、加密

總得想個辦法守住自己的飯碗



破解、識別

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

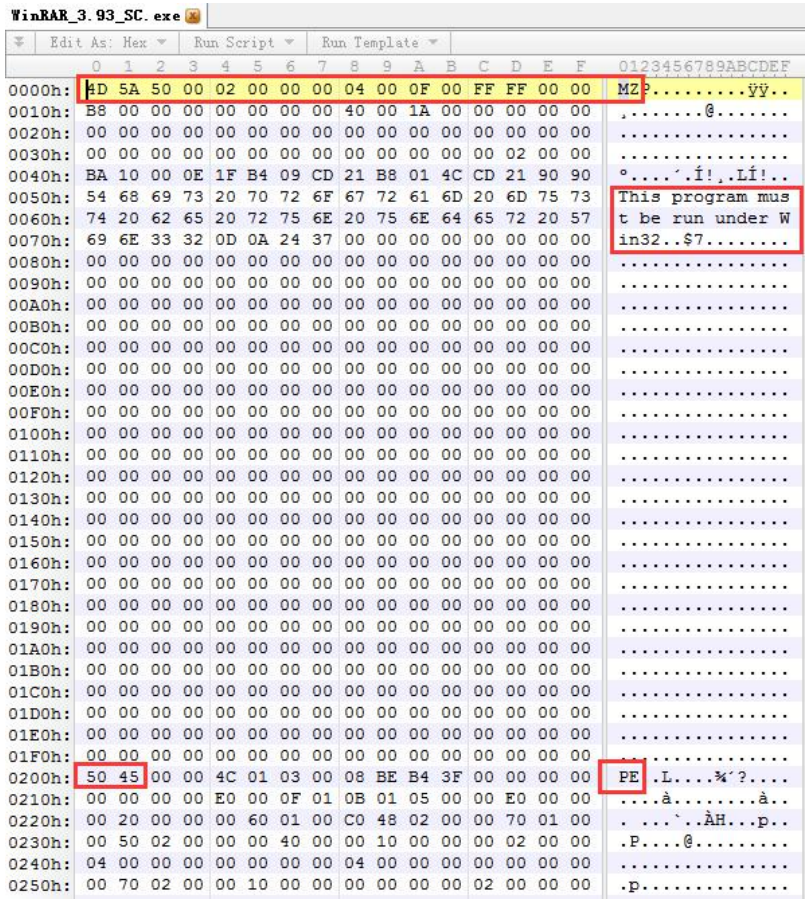
桌面上的檔提示你中病毒了

我們如何修復我們的系統，還原加密的檔

破解、識別

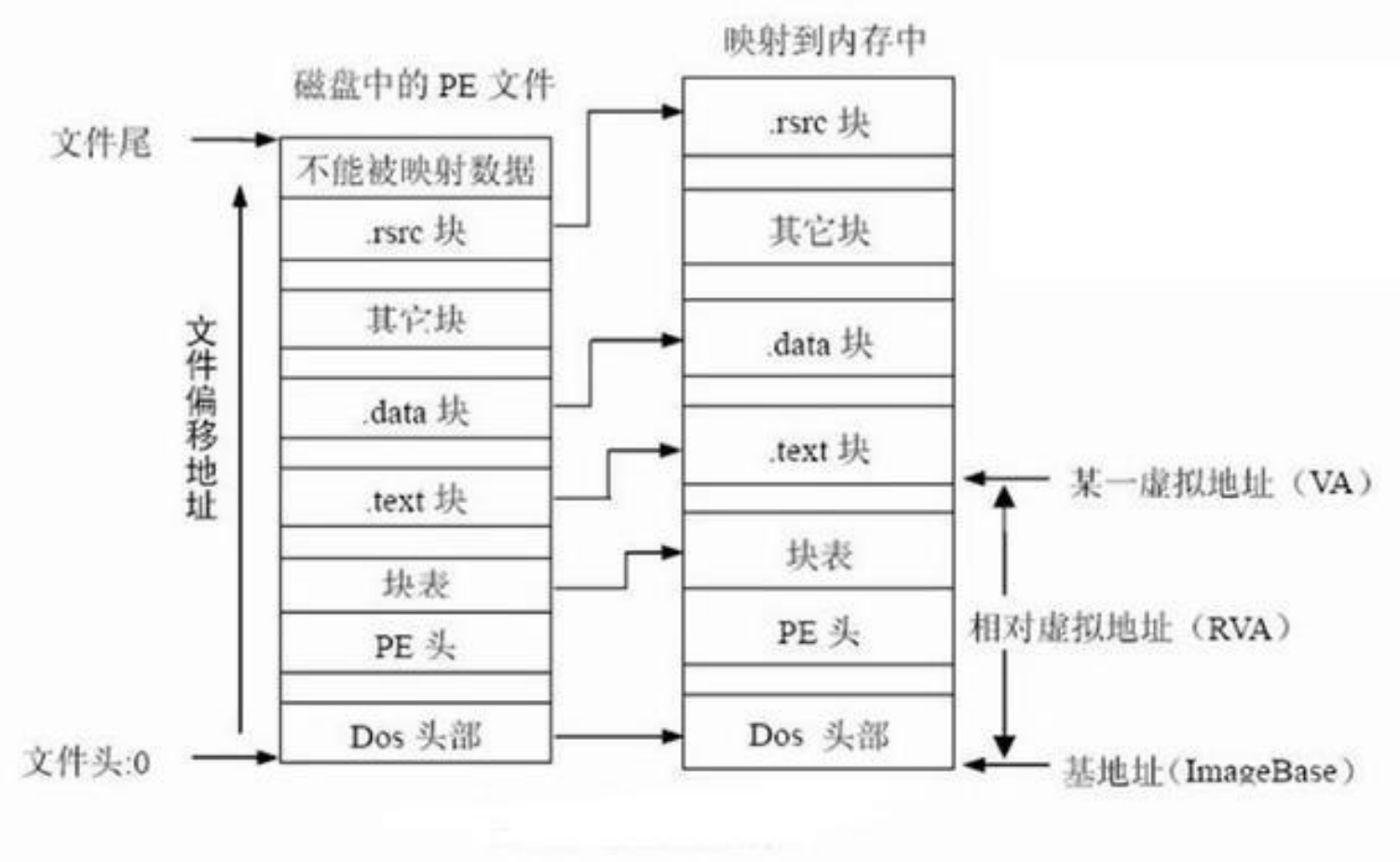
Windows

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



PE檔結構

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



破解、識別

現在換個角度，你是破解者

Linux

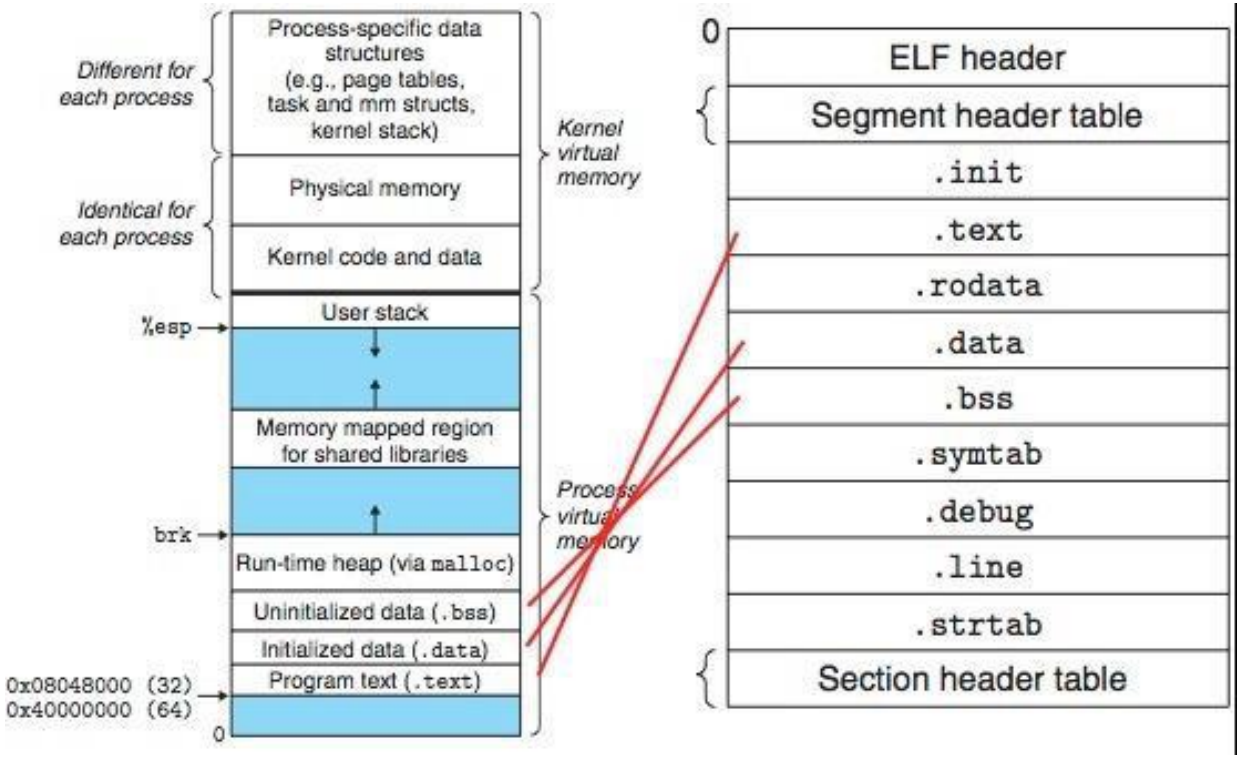
- ELF（Executable and Linkable Format）可執行和可鏈接檔
- 尾碼： 0，so，elf

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....

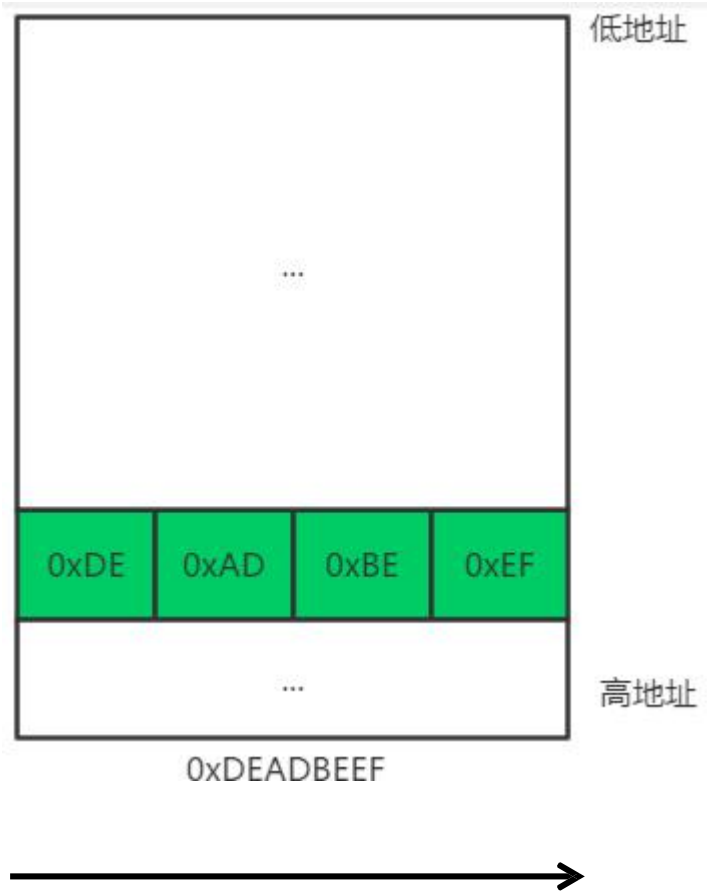


- .text
  - 已編譯程序的機器代碼
- .symtab
  - 符號表它存放在程式中被定義和引用的函數和全局變數的資訊
- .strtab
  - 一個字串表，其內容包括.symtab和.debug節中的符號表
- .bss
  - 存放程式中未初始化的全局變數和靜態變數的一塊記憶體區域。特點是可讀
    寫的

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



破解、識別

彙編：

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

```
seg000:00000000                 assume es:nothing, ss:nothing, ds:nothing, fs:nothing, gs:nothi
seg000:00000000                 dd 0DEADBEEFh
seg000:00000004                 dd 19260817h
seg000:00000008                 db 0A1h
seg000:00000009                 db    0
seg000:0000000A                 db    0
seg000:0000000B                 db    0
seg000:0000000C                 db    0
seg000:0000000D                 db  8Bh
seg000:0000000E                 db  1Dh
seg000:0000000F                 db    4
seg000:00000010                 db    0
seg000:00000011                 db    0
seg000:00000012                 db    0
seg000:00000013                 db    1
seg000:00000014                 db 0D8h
seg000:00000014 seg000          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

破解、識別

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**

破解、識別

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)

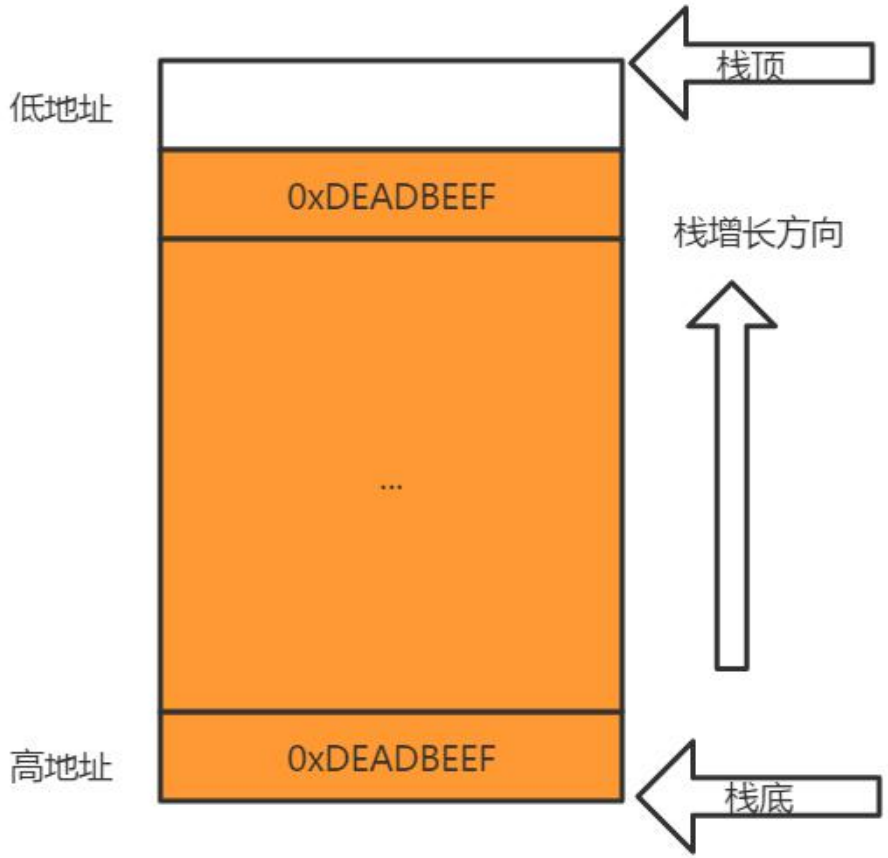
彙編書寫格式

- 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
```

栈（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
- 栈增長方向
  - 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

常用栈操作

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

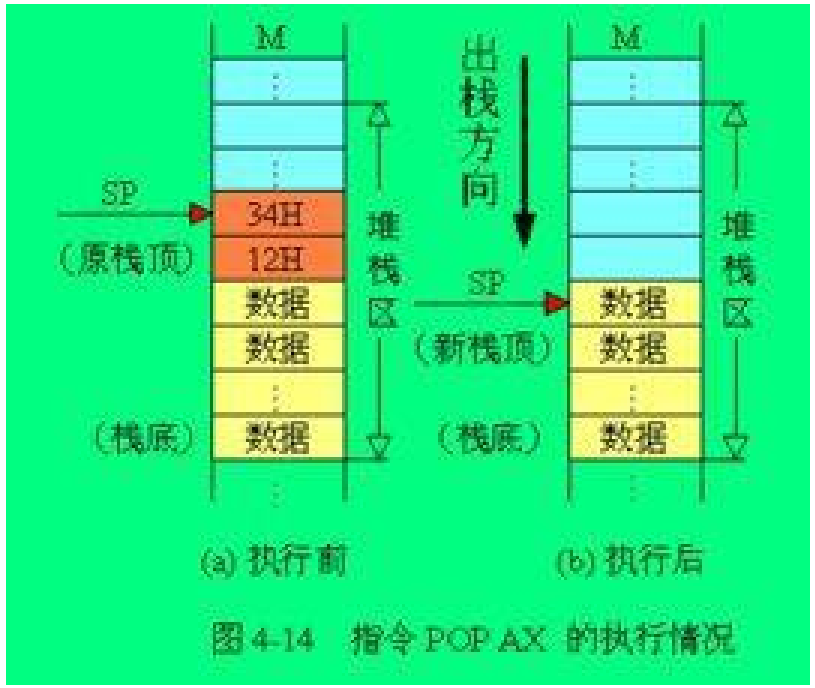
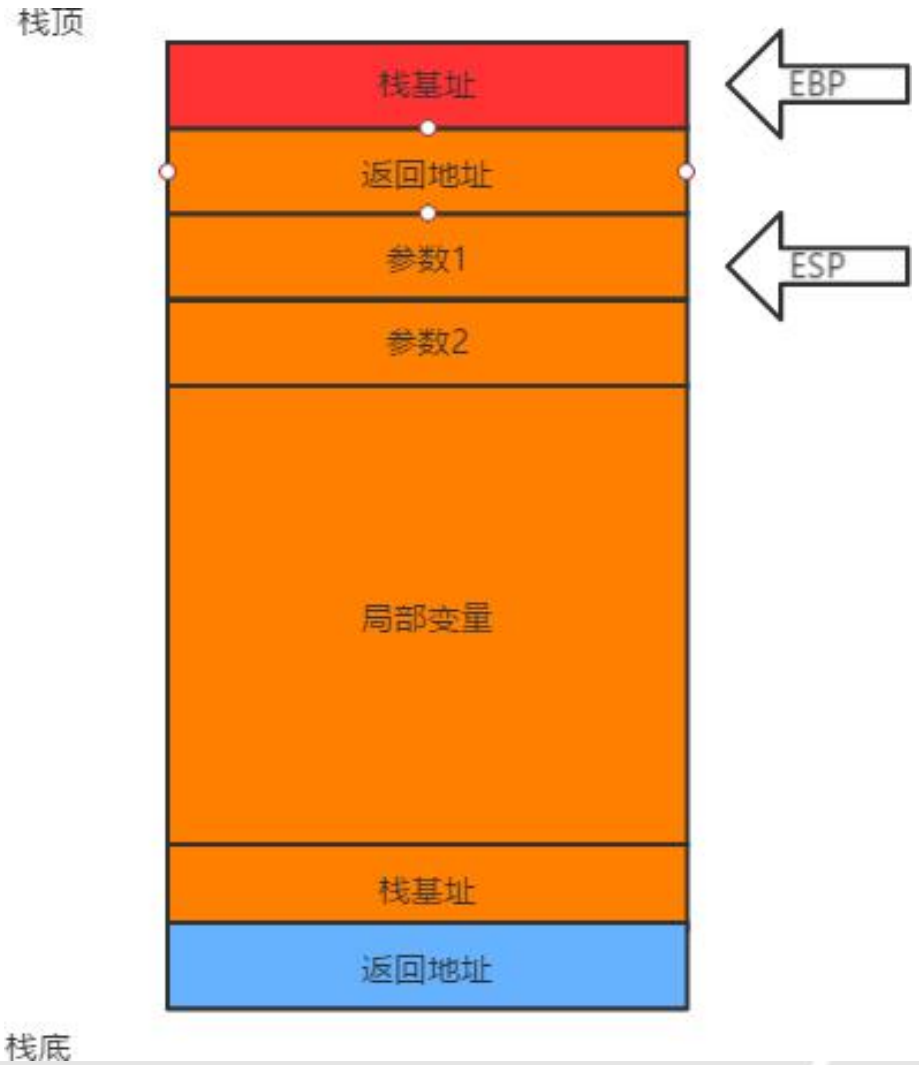


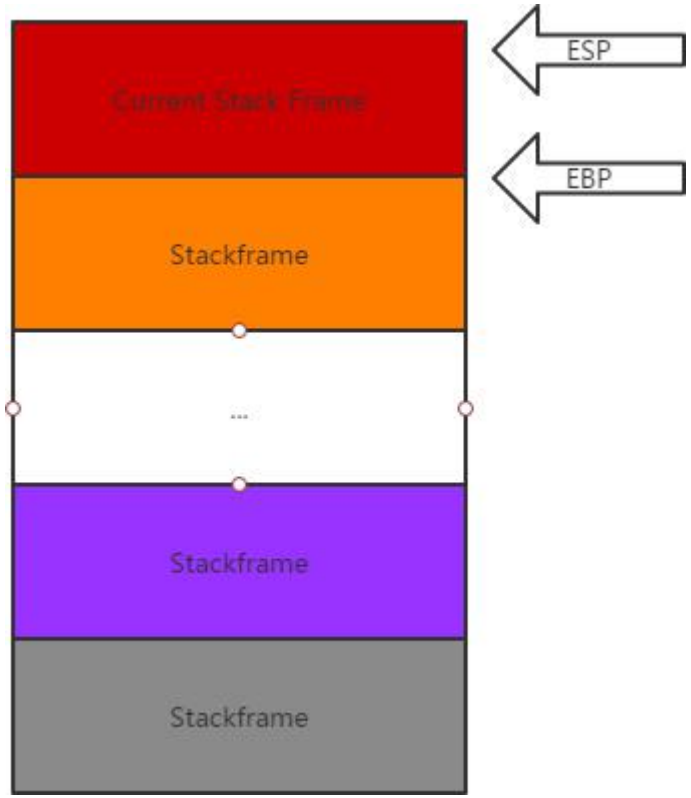
图 4-14 指令 POP AX 的执行情况

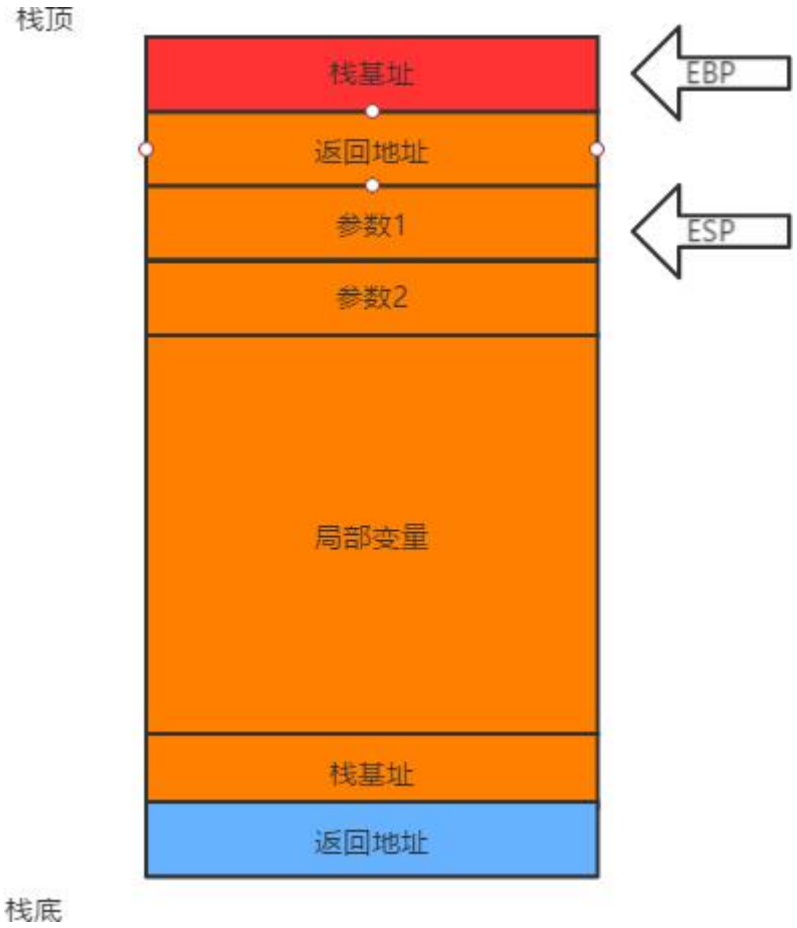
棧幀（stack frame）

- 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。



栈帧


ESP
EBP
Current Stack Frame
Stackframe
...
Stackframe
Stackframe


栈顶
栈基址
返回地址
参数1
参数2
局部变量
栈基址
返回地址
栈底
EBP
ESP

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

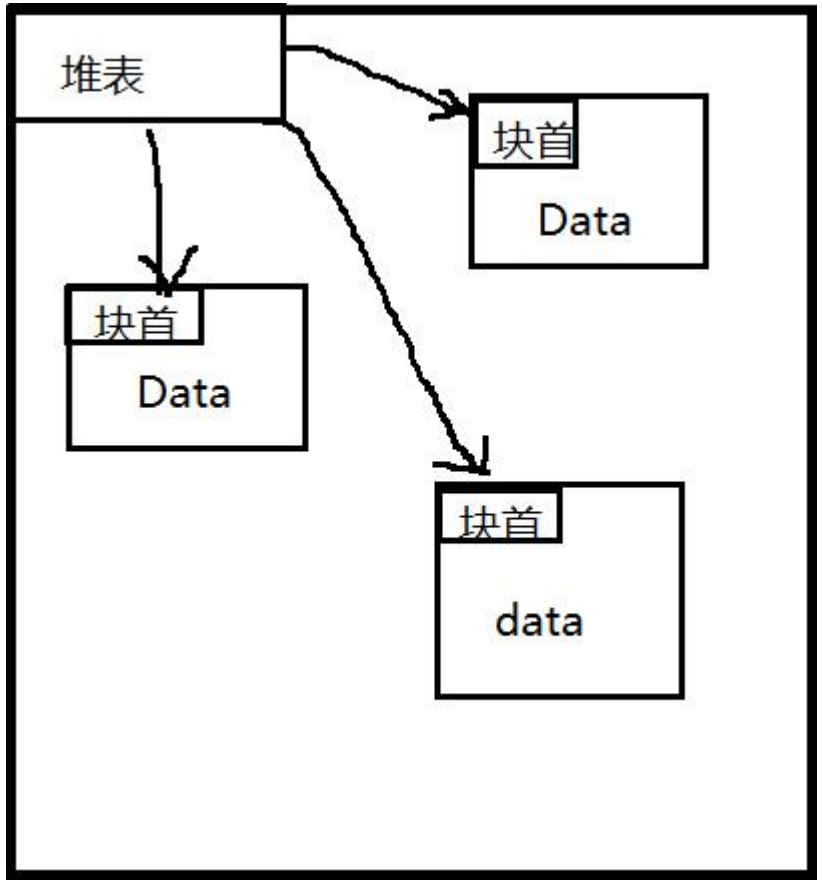
- 堆塊
  - 堆區內存按不同大小組織成塊，以堆塊為單位進行標識
  - 包括兩個部分：塊首和塊身。
  - 塊首：堆塊頭部的幾個位元組，用來標識這個塊首自身的信息
  - 塊身：在塊首後面的部分，分配給用戶使用的數據區

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
  - 所有參數都在栈上
  - 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，之後的參數存在栈上
  - 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

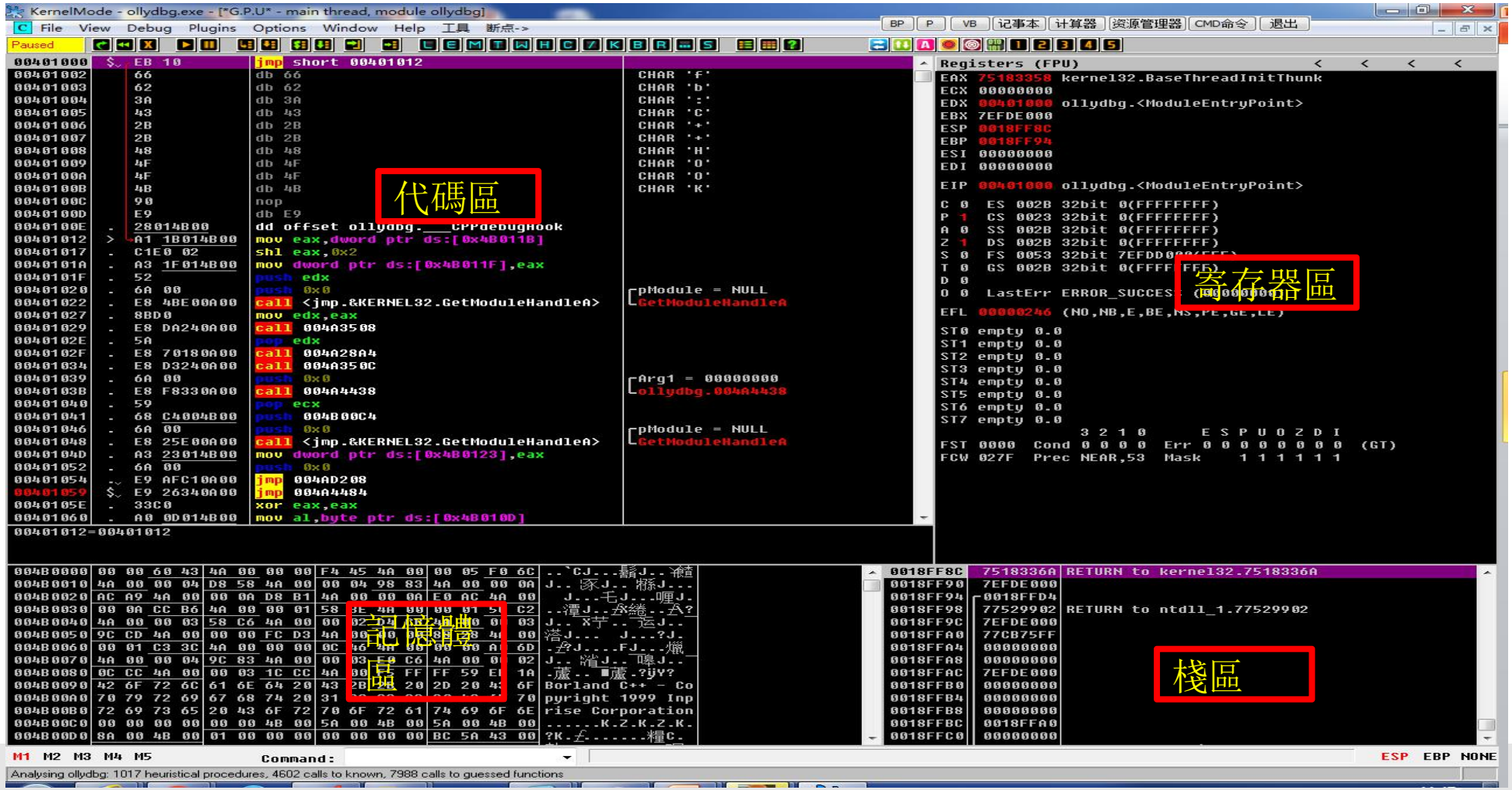
逆向工具

逆向技巧

- 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?

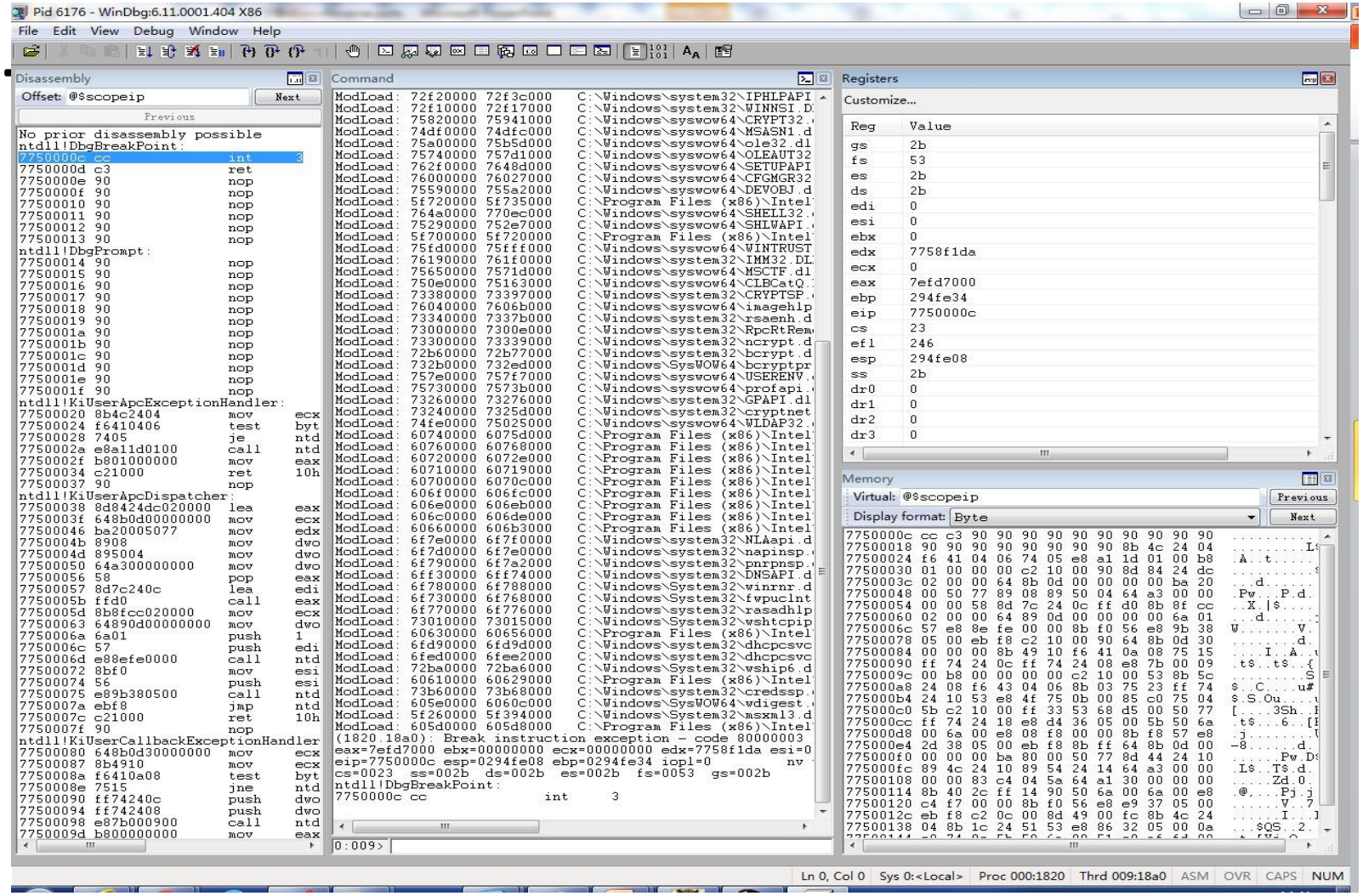
動態調試工具——OllyDebug



什麼是逆向工程?

動態調試工具—Windbg

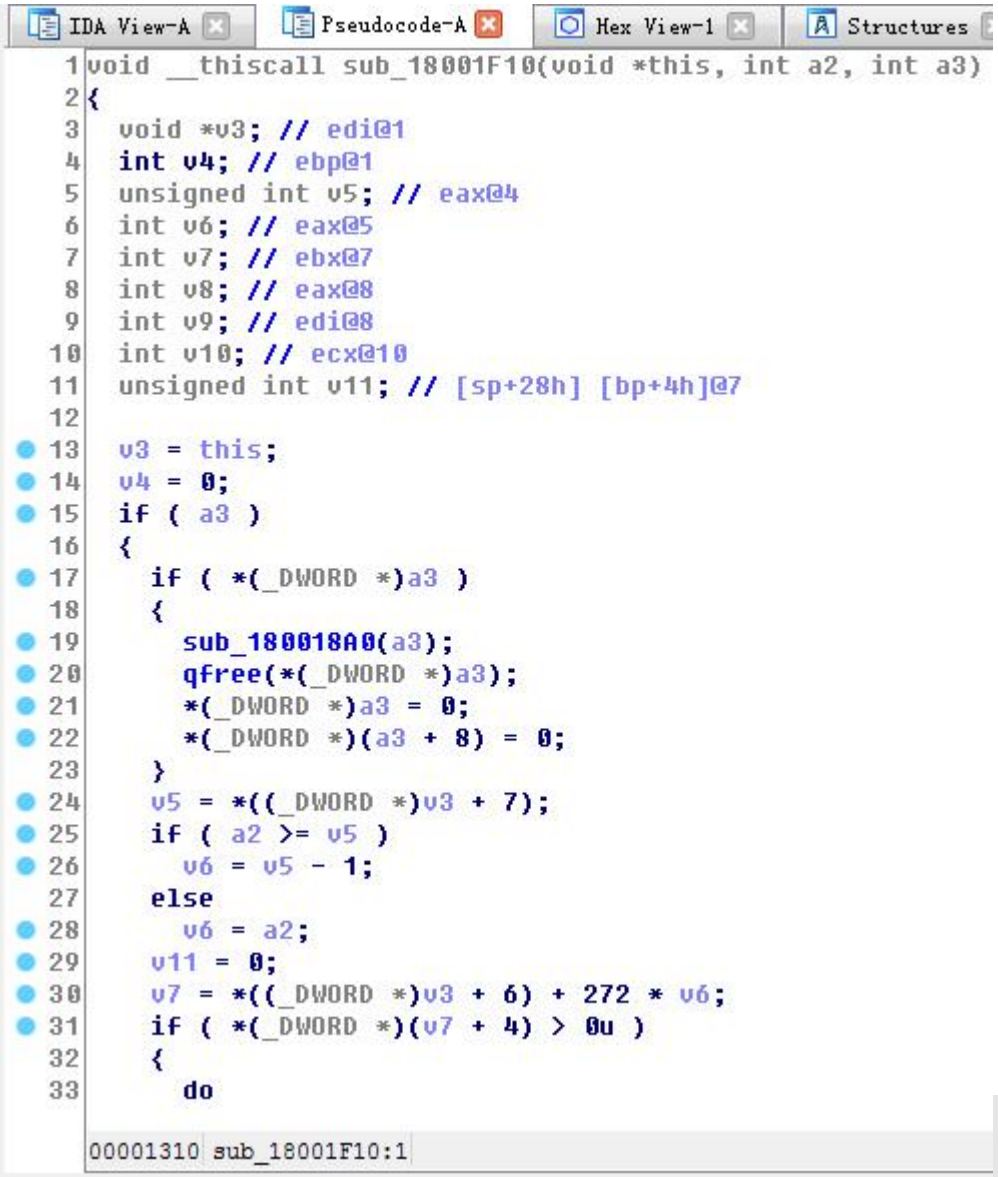
- 可進行內核調試



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



動態調試工具——gdb

- Linux
- 添加pwndbg插件後有代碼著色

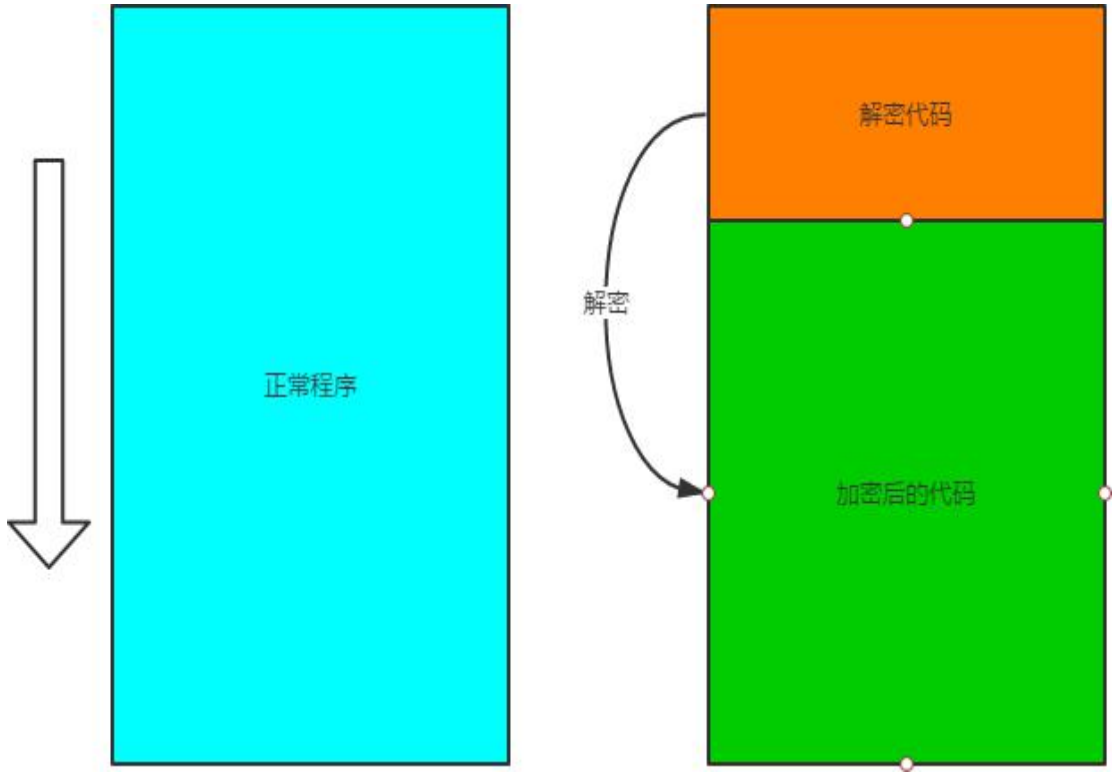
```
gdb-peda$ start
[----------------------------------registers-----------------------------------]
EAX: 0xbffff7f4 --> 0xbffff916 ("/root/a.out")
EBX: 0xb7fcbff4 --> 0x155d7c
ECX: 0xd5eeaa03
EDX: 0x1
ESI: 0x0
EDI: 0x0
EBP: 0xbffff748 --> 0xbffff7c8 --> 0x0
ESP: 0xbffff748 --> 0xbffff7c8 --> 0x0
EIP: 0x80483e7 (<main+3>:       and    esp,0xfffffff0)
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,0xfffffff0
   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

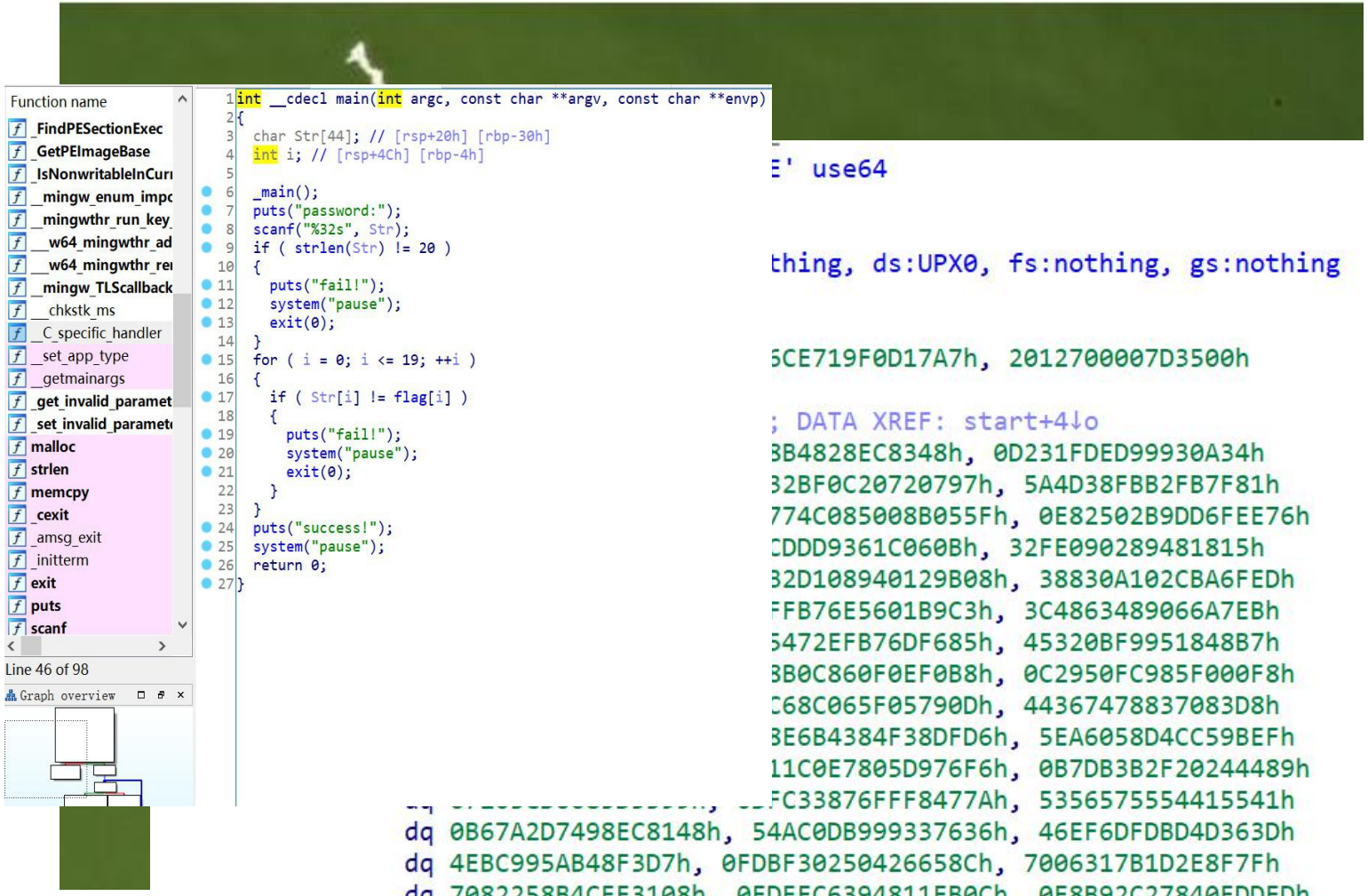
在蟬脫出殼之前，我們無從得知殼中生物的真面貌，

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）



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

脫殼

- 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。

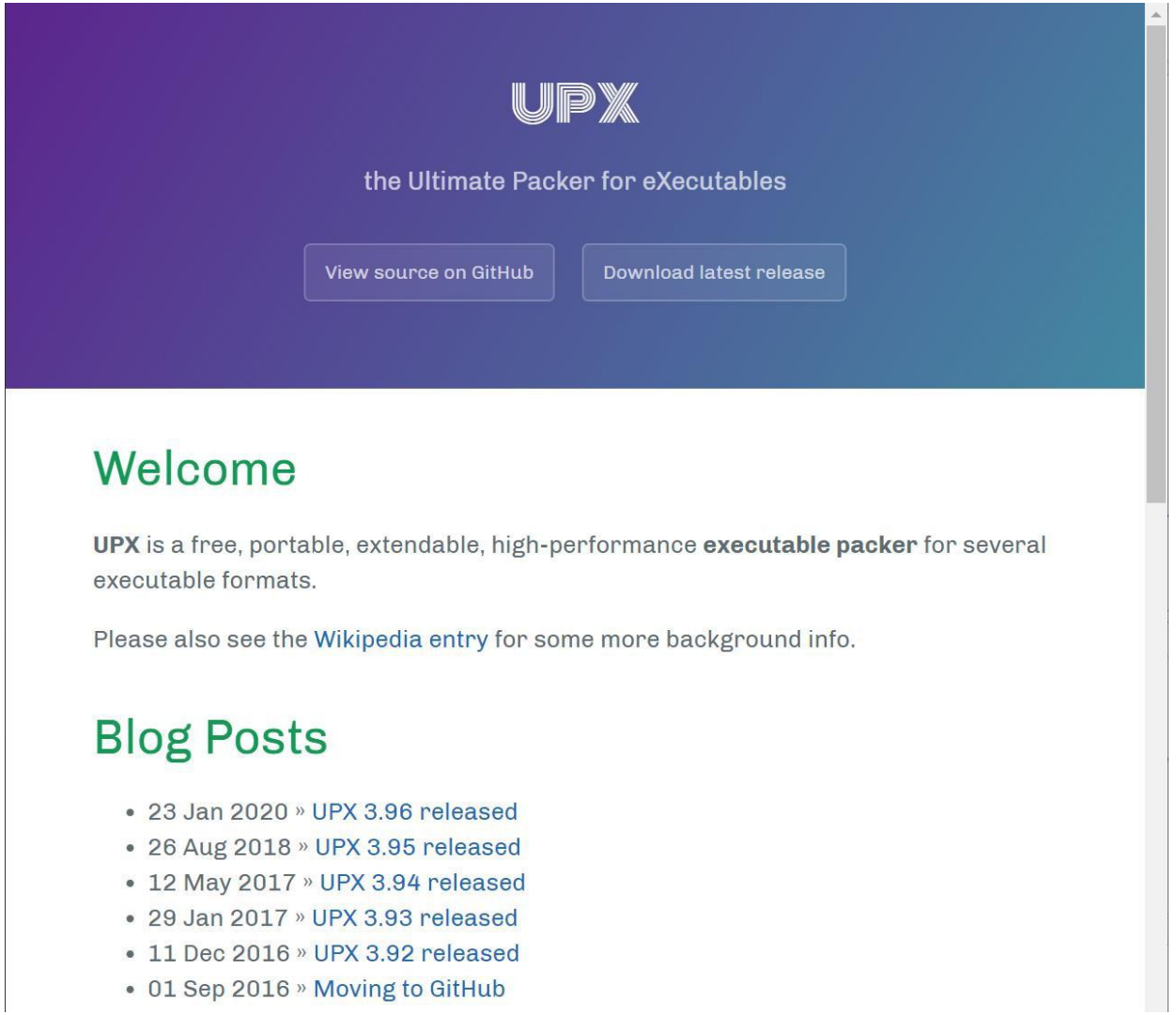
作者團隊仍然在維護這個專案。目前保持一年一次到兩次更新。

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



壓縮殼

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

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

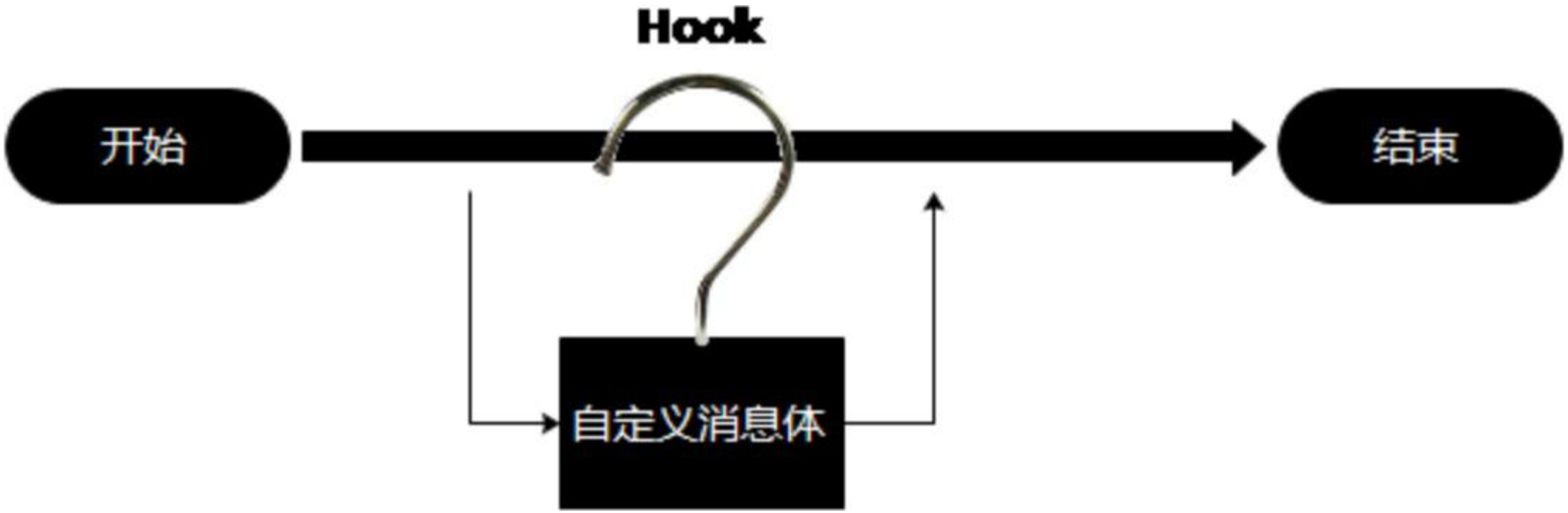
常用反調試函數

- IsDebuggerPresent
- TLS_callback
- ptrace

- 解決方法:
  - 找到函數調用者
  - 刪除反調試函數

hook

- 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

```
while ( 1 )
{
  result = e_ip;
  if ( e_ip >= a2 )
    return result;
  switch ( codes[e_ip] )
  {
    case 1:
      stack_mem[e_sp] = e_ax;                 // push eax,inc si
      ++e_ip;
      ++e_sp;
      ++e_si;
      break;
    case 2:
      e_ax = codes[e_ip + 1] + usr_input[e_si];// eax = usr_input[esi] + op[1]
      e_ip += 2;
      break;
    case 3:
      e_ax = usr_input[e_si] - LOBYTE(codes[e_ip + 1]);// eax = usr_input[esi] - op
      e_ip += 2;
      break;
    case 4:
      e_ax = codes[e_ip + 1] ^ usr_input[e_si];// e_ax = usr_input[esi] ^ op[1]
      e_ip += 2;
      break;
    case 5:
      e_ax = codes[e_ip + 1] * usr_input[e_si];// e_ax = usr_input[esi] * op[1]
      e_ip += 2;
      break;
    case 6:
      ++e_ip;
      break;
```

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