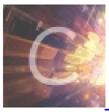
EasyRE WriteUp

原创

Chauncy. 于 2020-05-16 17:45:18 发布 918 收藏
分类专栏： 程序逆向之美 文章标签： 安全


程序逆向之美 专栏收录该内容
11 篇文章 0 订阅
订阅专栏

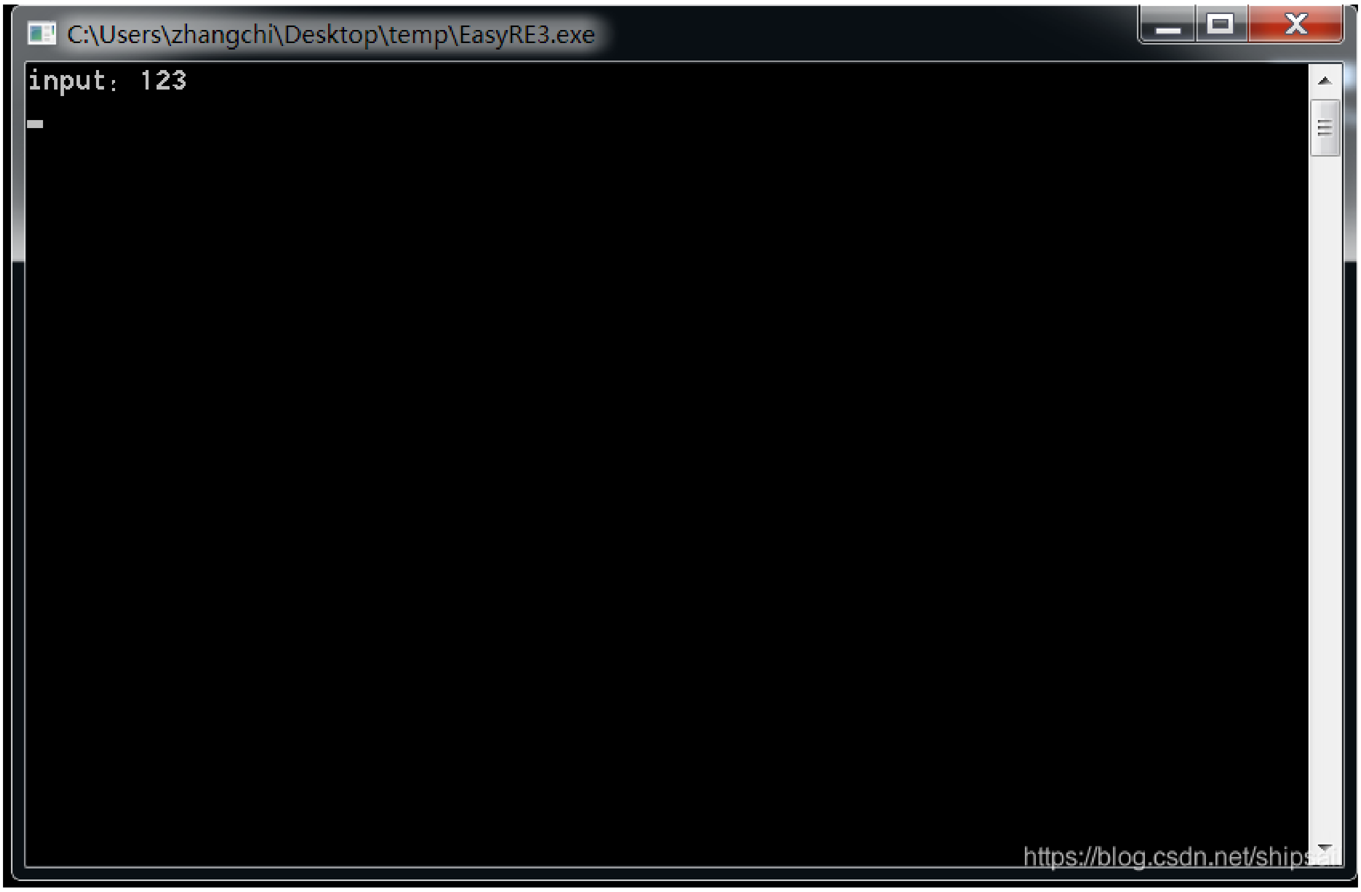
目录

题源：XCTF-Reverse进阶-004

0x0 新知识

XOR 常用于置0
XOR 运算可逆

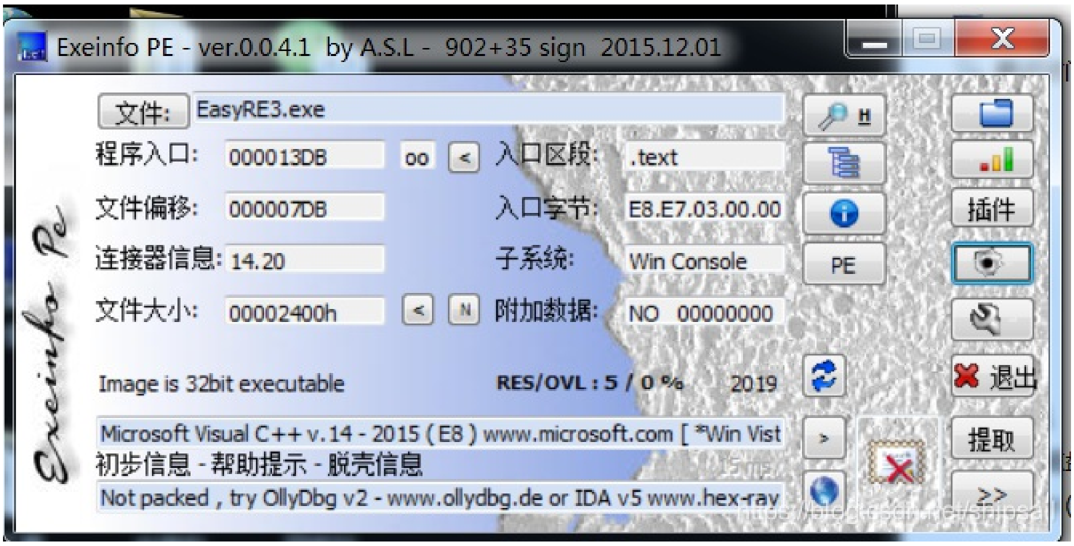
0x1 运行



```
input: 123
```

运行提示输入，回车后即退出

0x2 查壳



文件: EasyRE3.exe

程序入口: 000013DB oo < 入口区段: .text

文件偏移: 000007DB 入口字节: E8.E7.03.00.00

连接器信息: 14.20 子系统: Win Console

文件大小: 00002400h < N 附加数据: NO 00000000

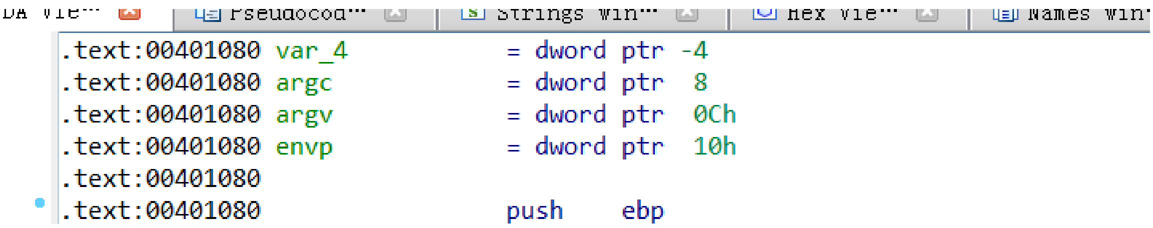
Image is 32bit executable RES/OVL : 5 / 0 % 2019

Microsoft Visual C++ v.14 - 2015 (E8) www.microsoft.com [*Win Vist

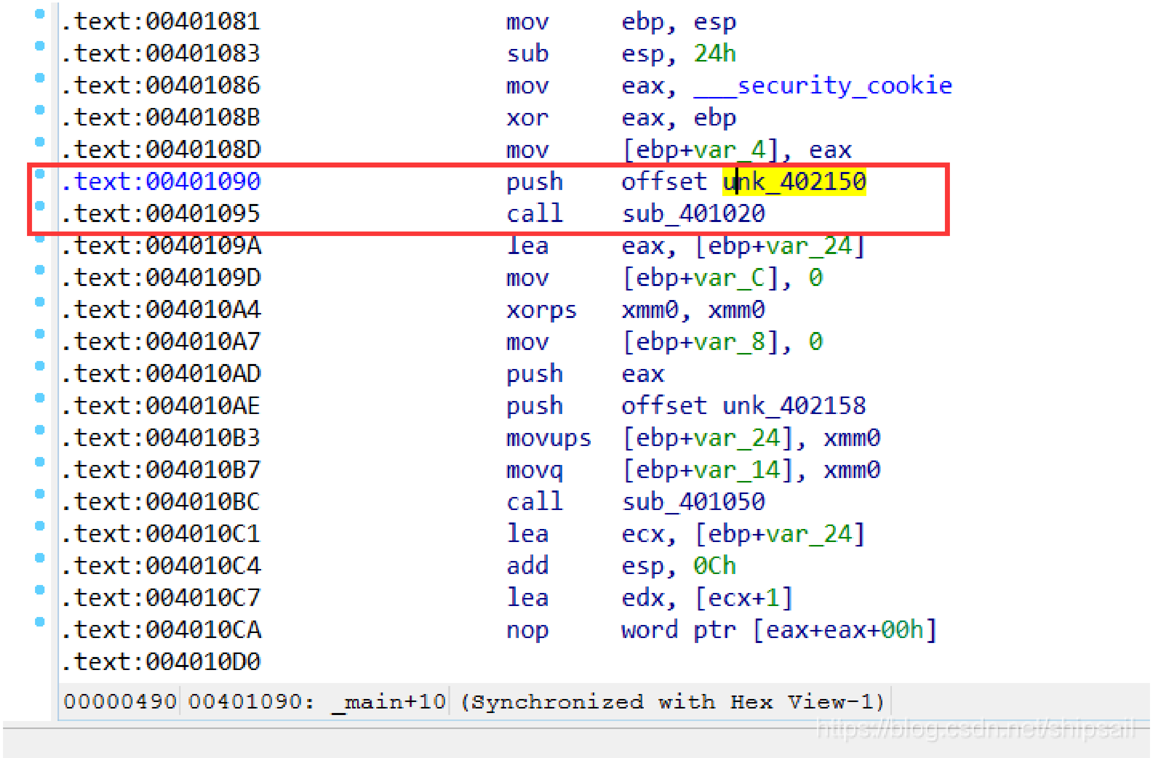
初步信息 - 帮助提示 - 脱壳信息

Not packed , try OllyDbg v2 - www.ollydbg.de or IDA v5 www.hex-ray

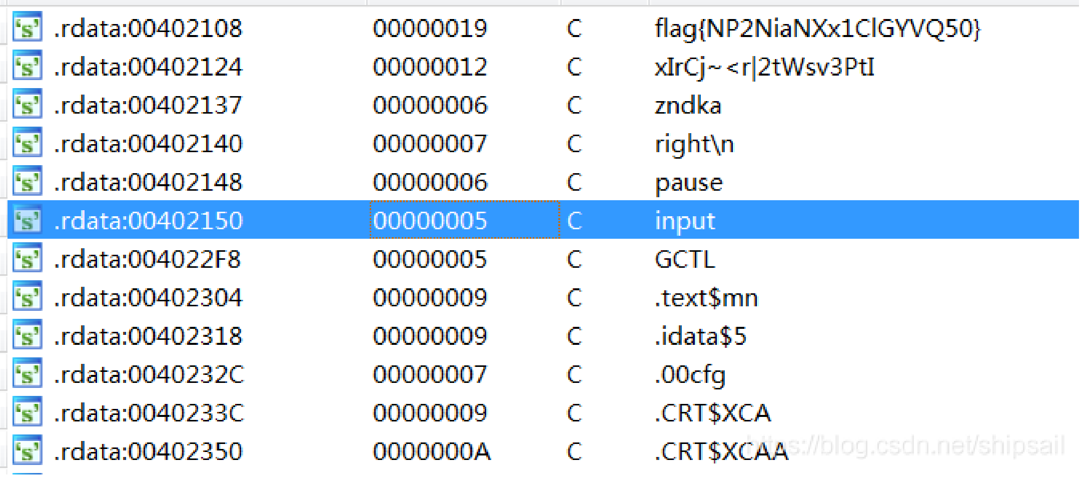
0x3 载入IDA 32bit



```
.text:00401080 var_4           = dword ptr -4
.text:00401080 argc            = dword ptr  8
.text:00401080 argv            = dword ptr  0Ch
.text:00401080 envp            = dword ptr  10h
.text:00401080
.text:00401080                 push    ebp
```

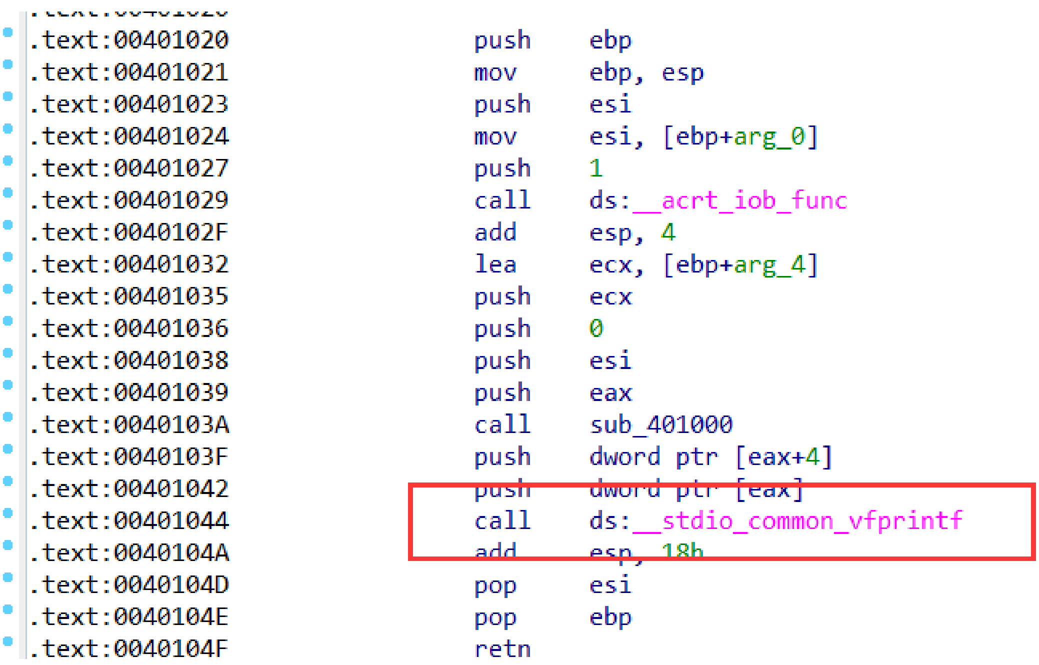


```
.text:00401081                 mov     ebp, esp
.text:00401083                 sub     esp, 24h
.text:00401086                 mov     eax, ___security_cookie
.text:0040108B                 xor     eax, ebp
.text:0040108D                 mov     [ebp+var_4], eax
.text:00401090                 push    offset unk_402150
.text:00401095                 call    sub_401020
.text:0040109A                 lea     eax, [ebp+var_24]
.text:0040109D                 mov     [ebp+var_C], 0
.text:004010A4                 xorps   xmm0, xmm0
.text:004010A7                 mov     [ebp+var_8], 0
.text:004010AD                 push    eax
.text:004010AE                 push    offset unk_402158
.text:004010B3                 movups  [ebp+var_24], xmm0
.text:004010B7                 movq    [ebp+var_14], xmm0
.text:004010BC                 call    sub_401050
.text:004010C1                 lea     ecx, [ebp+var_24]
.text:004010C4                 add     esp, 0Ch
.text:004010C7                 lea     edx, [ecx+1]
.text:004010CA                 nop     word ptr [eax+eax+00h]
.text:004010D0
00000490 00401090: _main+10 (Synchronized with Hex View-1)
```

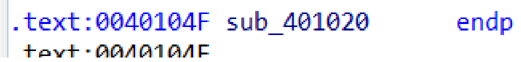


.rdata:00402108	00000019	C	flag{NP2NiaNXx1ClGYVQ50}
.rdata:00402124	00000012	C	xIrCj~<r|2tWsv3PtI
.rdata:00402137	00000006	C	zndka
.rdata:00402140	00000007	C	right\n
.rdata:00402148	00000006	C	pause
.rdata:00402150	00000005	C	input
.rdata:004022F8	00000005	C	GCTL
.rdata:00402304	00000009	C	.text$mn
.rdata:00402318	00000009	C	.idata$5
.rdata:0040232C	00000007	C	.00cfg
.rdata:0040233C	00000009	C	.CRT$XCA
.rdata:00402350	0000000A	C	.CRT$XCAA

可以知道 sub_401020这个call是printf()



```
.text:00401020                 push    ebp
.text:00401021                 mov     ebp, esp
.text:00401023                 push    esi
.text:00401024                 mov     esi, [ebp+arg_0]
.text:00401027                 push    1
.text:00401029                 call    ds:__acrt_iob_func
.text:0040102F                 add     esp, 4
.text:00401032                 lea     ecx, [ebp+arg_4]
.text:00401035                 push    ecx
.text:00401036                 push    0
.text:00401038                 push    esi
.text:00401039                 push    eax
.text:0040103A                 call    sub_401000
.text:0040103F                 push    dword ptr [eax+4]
.text:00401042                 push    dword ptr [eax]
.text:00401044                 call    ds:__stdio_common_vfprintf
.text:0040104A                 add     esp, 18h
.text:0040104D                 pop     esi
.text:0040104E                 pop     ebp
.text:0040104F                 retn
```

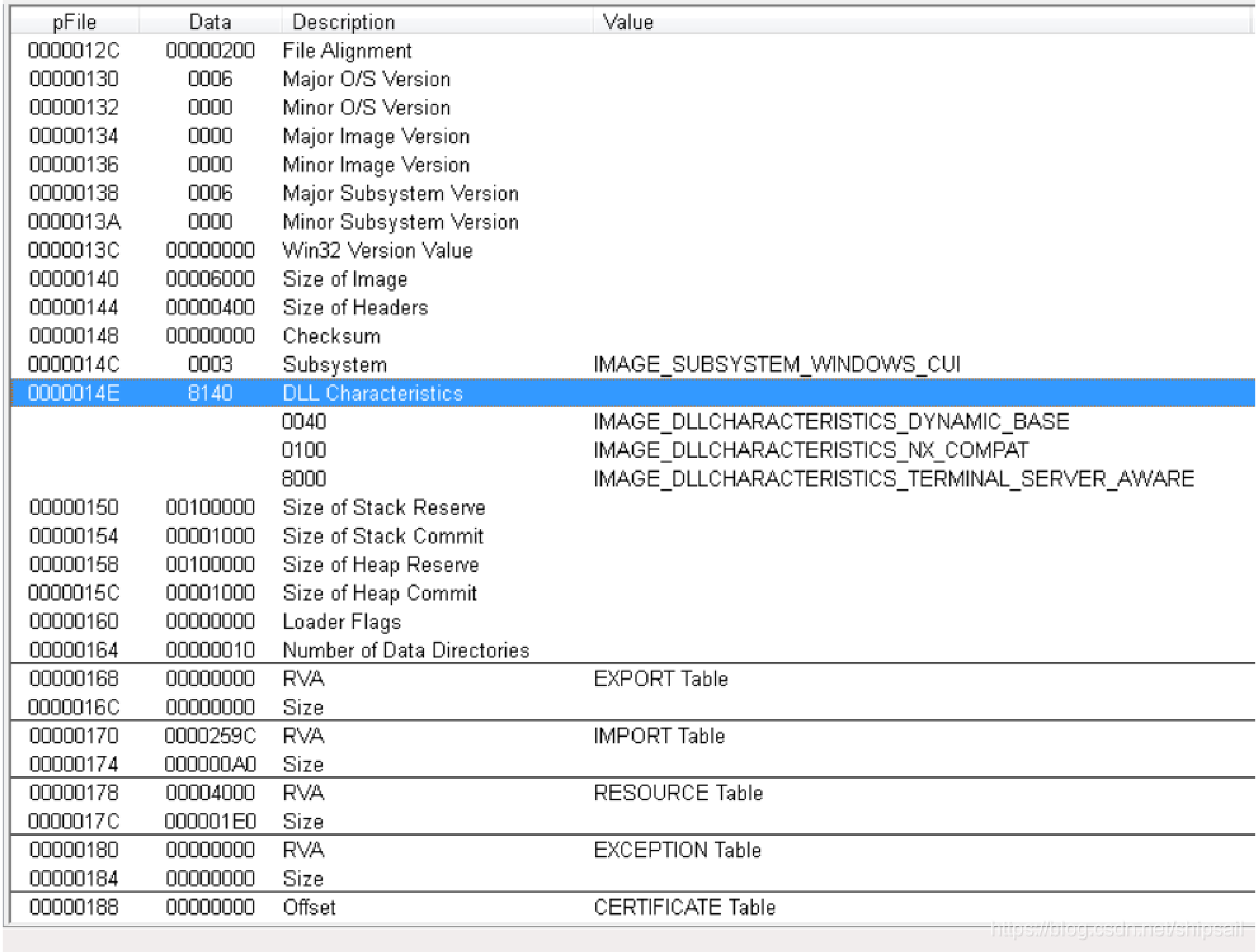


```
.text:0040104F sub_401020      endp
```

因为我还是个小白，并不能像大神们一样看静态汇编代码就完全理解程序逻辑，所以我使用IDA 与 OD动静结合的方式来进行分析。

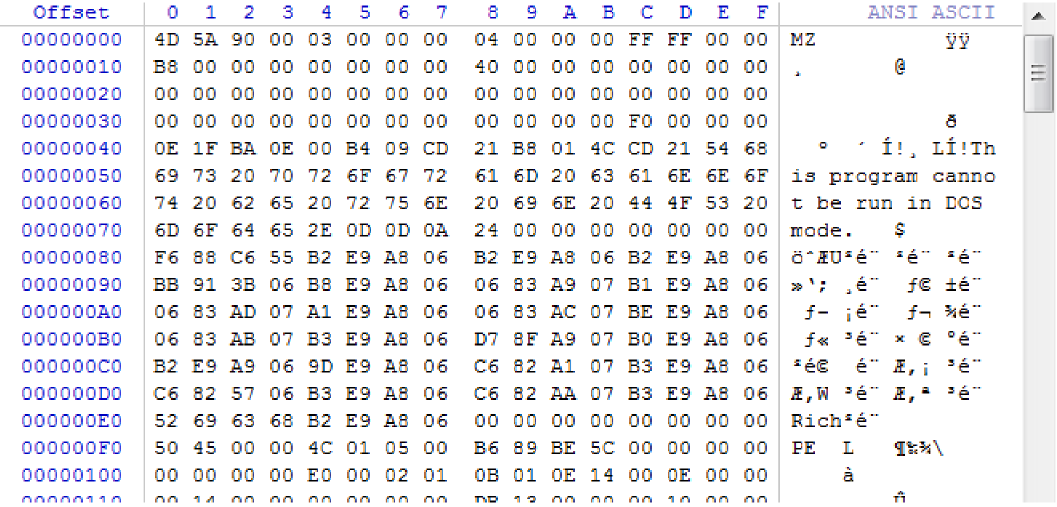
0x4 关闭ASLR，地址对齐

我是在WIN7下进行测试，所以可以手动关闭ASLR使得程序运行不进行随机加载。

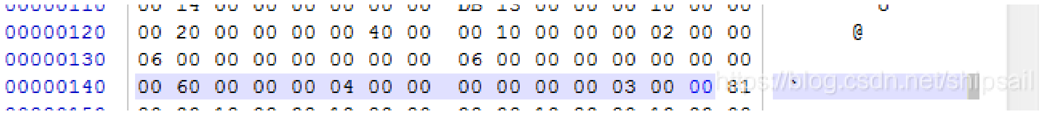


pFile	Data	Description	Value
0000012C	00000200	File Alignment	
00000130	0006	Major O/S Version	
00000132	0000	Minor O/S Version	
00000134	0000	Major Image Version	
00000136	0000	Minor Image Version	
00000138	0006	Major Subsystem Version	
0000013A	0000	Minor Subsystem Version	
0000013C	00000000	Win32 Version Value	
00000140	00006000	Size of Image	
00000144	00000400	Size of Headers	
00000148	00000000	Checksum	
0000014C	0003	Subsystem	IMAGE_SUBSYSTEM_WINDOWS_CUI
0000014E	8140	DLL Characteristics	
		0040	IMAGE_DLLCHARACTERISTICS_DYNAMIC_BASE
		0100	IMAGE_DLLCHARACTERISTICS_NX_COMPAT
		8000	IMAGE_DLLCHARACTERISTICS_TERMINAL_SERVER_AWARE
00000150	00100000	Size of Stack Reserve	
00000154	00001000	Size of Stack Commit	
00000158	00100000	Size of Heap Reserve	
0000015C	00001000	Size of Heap Commit	
00000160	00000000	Loader Flags	
00000164	00000010	Number of Data Directories	
00000168	00000000	RVA	EXPORT Table
0000016C	00000000	Size	
00000170	0000259C	RVA	IMPORT Table
00000174	000000A0	Size	
00000178	00004000	RVA	RESOURCE Table
0000017C	000001E0	Size	
00000180	00000000	RVA	EXCEPTION Table
00000184	00000000	Size	
00000188	00000000	Offset	CERTIFICATE Table

使用VIEW辅助查看文件偏移，使用WinHex修改数据。在Win下数据为小端序存储，所以在十六进制文件中将看到 `40 81`

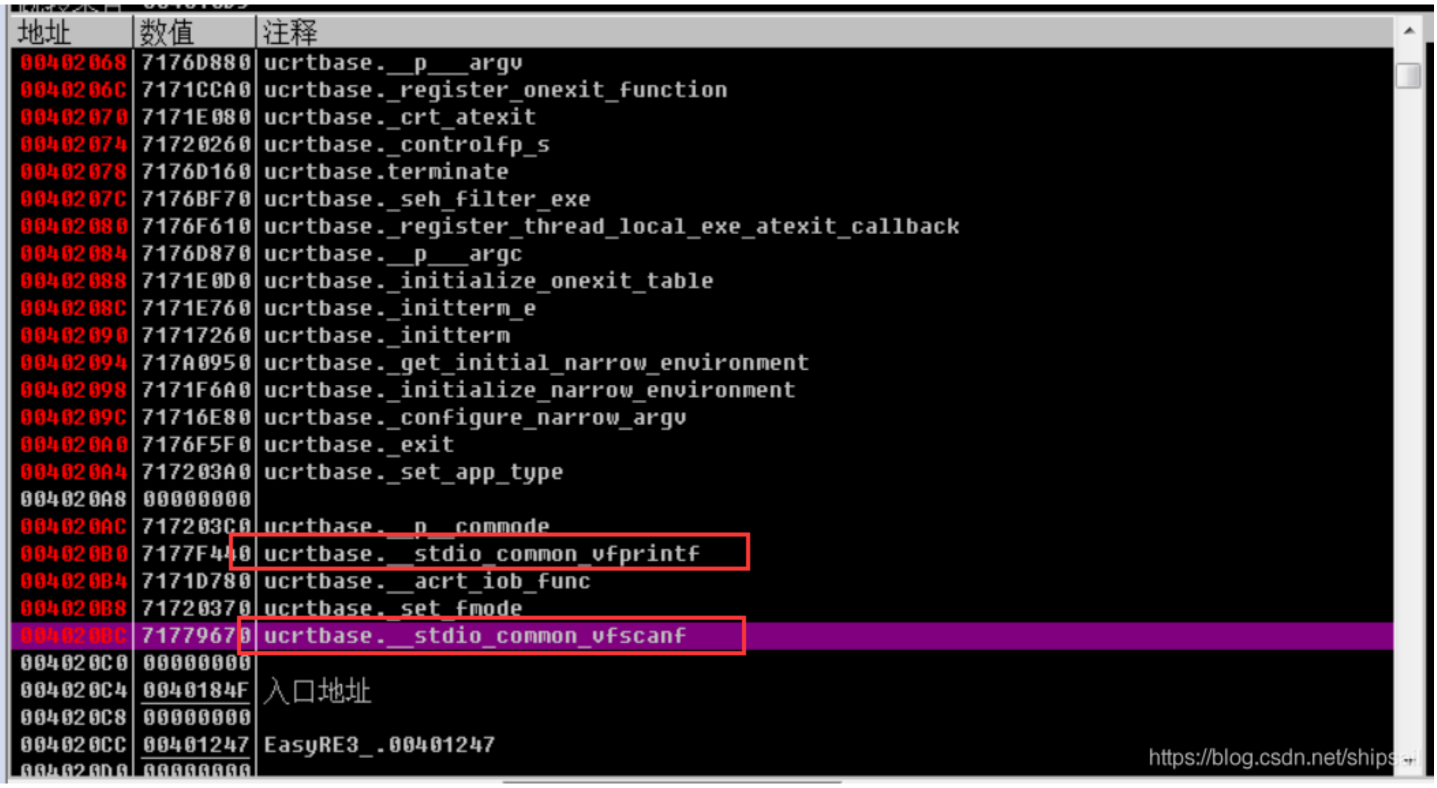


```
  Offset      0  1  2  3  4  5  6  7   8  9  A  B  C  D  E  F        ANSI ASCII
00000000   4D 5A 90 00 03 00 00 00  04 00 00 00 FF FF 00 00   MZ          ÿÿ
00000010   B8 00 00 00 00 00 00 00  40 00 00 00 00 00 00 00   ¸       @
00000020   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00000030   00 00 00 00 00 00 00 00  00 00 00 00 F0 00 00 00               ð
00000040   0E 1F BA 0E 00 B4 09 CD  21 B8 01 4C CD 21 54 68     º  ´ Í!¸ LÍ!Th
00000050   69 73 20 70 72 6F 67 72  61 6D 20 63 61 6E 6E 6F   is program canno
00000060   74 20 62 65 20 72 75 6E  20 69 6E 20 44 4F 53 20   t be run in DOS
00000070   6D 6F 64 65 2E 0D 0D 0A  24 00 00 00 00 00 00 00   mode.   $
00000080   F6 88 C6 55 B2 E9 A8 06  B2 E9 A8 06 B2 E9 A8 06   ö^ÆU²é¨ ²é¨ ²é¨
00000090   BB 91 3B 06 B8 E9 A8 06  06 83 A9 07 B1 E9 A8 06   »'; ¸é¨  ƒ© ±é¨
000000A0   06 83 AD 07 A1 E9 A8 06  06 83 AC 07 BE E9 A8 06    ƒ- ¡é¨  ƒ¬ ¾é¨
000000B0   06 83 AB 07 B3 E9 A8 06  D7 8F A9 07 B0 E9 A8 06    ƒ« ³é¨ × © °é¨
000000C0   B2 E9 A9 06 9D E9 A8 06  C6 82 A1 07 B3 E9 A8 06   ²é©  é¨ Æ,¡ ³é¨
000000D0   C6 82 57 06 B3 E9 A8 06  C6 82 AA 07 B3 E9 A8 06   Æ,W ³é¨ Æ,ª ³é¨
000000E0   52 69 63 68 B2 E9 A8 06  00 00 00 00 00 00 00 00   Rich²é¨
000000F0   50 45 00 00 4C 01 05 00  B6 89 BE 5C 00 00 00 00   PE  L   ¶‰¾\
00000100   00 00 00 00 E0 00 02 01  0B 01 0E 14 00 0E 00 00       à
```



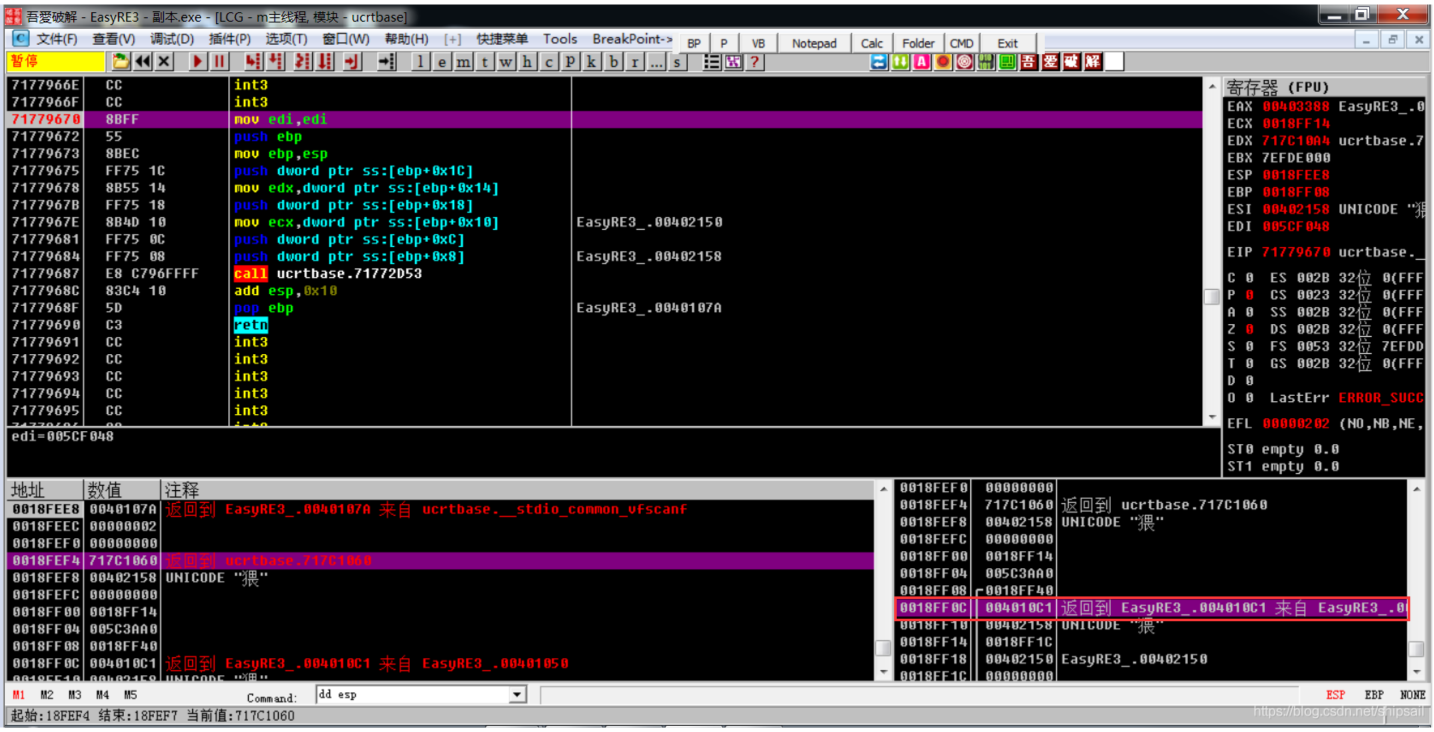
```
00000120  00 20 00 00 00 00 40 00  00 10 00 00 00 02 00 00         @
00000130  06 00 00 00 00 00 00 00  06 00 00 00 00 00 00 00
00000140  00 60 00 00 00 04 00 00  00 00 00 00 03 00 00 81
```

关闭ASLR的目的在于将OD中的地址与IDA中的地址进行对齐方便查看，如果不会修改ASLR还可以通过下API断点进行定位。



```
地址      数值      注释
00402068  7176D880  ucrtbase.__p___argv
0040206C  7171CCA0  ucrtbase._register_onexit_function
00402070  7171E080  ucrtbase._crt_atexit
00402074  71720260  ucrtbase._controlfp_s
00402078  7176D160  ucrtbase.terminate
0040207C  7176BF70  ucrtbase._seh_filter_exe
00402080  7176F610  ucrtbase._register_thread_local_exe_atexit_callback
00402084  7176D870  ucrtbase.__p___argc
00402088  7171E0D0  ucrtbase._initialize_onexit_table
0040208C  7171E760  ucrtbase._initterm_e
00402090  71717260  ucrtbase._initterm
00402094  717A0950  ucrtbase._get_initial_narrow_environment
00402098  7171F6A0  ucrtbase._initialize_narrow_environment
0040209C  71716E80  ucrtbase._configure_narrow_argv
004020A0  7176F5F0  ucrtbase._exit
004020A4  717203A0  ucrtbase._set_app_type
004020A8  00000000
004020AC  717203C0  ucrtbase.__p__commode
004020B0  7177F440  ucrtbase.__stdio_common_vfprintf
004020B4  7171D780  ucrtbase.__acrt_iob_func
004020B8  71720370  ucrtbase._set_fmode
004020BC  71779670  ucrtbase.__stdio_common_vfscanf
004020C0  00000000
004020C4  0040184F  入口地址
004020C8  00000000
004020CC  00401247  EasyRE3_.00401247
```

在IAT中可以看到导入的所有API，我发现了这两个在IDA中出现过的函数，我们要定位程序开始运行的代码段，只要在 scanf 的函数头下断点并通过堆栈回到上一层即可。



吾爱破解 - EasyRE3 - 副本.exe - [LCG - m主线程, 模块 - ucrtbase]
文件(F) 查看(V) 调试(D) 插件(P) 选项(T) 窗口(W) 帮助(H) [+] 快捷菜单 Tools BreakPoint-> BP P VB Notepad Calc Folder CMD Exit
暂停
l e m t w h c P k b r ... s

```
7177966E   CC           int3
7177966F   CC           int3
71779670   8BFF         mov edi,edi
71779672   55           push ebp
71779673   8BEC         mov ebp,esp
71779675   FF75 1C      push dword ptr ss:[ebp+0x1C]
71779678   8B55 14      mov edx,dword ptr ss:[ebp+0x14]
7177967B   FF75 18      push dword ptr ss:[ebp+0x18]
7177967E   8B4D 10      mov ecx,dword ptr ss:[ebp+0x10]        EasyRE3_.00402150
71779681   FF75 0C      push dword ptr ss:[ebp+0xC]
71779684   FF75 08      push dword ptr ss:[ebp+0x8]            EasyRE3_.00402158
71779687   E8 C796FFFF  call ucrtbase.71772D53
7177968C   83C4 10      add esp,0x10
7177968F   5D           pop ebp                                EasyRE3_.0040107A
71779690   C3           retn
71779691   CC           int3
71779692   CC           int3
71779693   CC           int3
71779694   CC           int3
71779695   CC           int3
edi=005CF048
```

```
寄存器 (FPU)
EAX 00403388 EasyRE3_.0
ECX 0018FF14
EDX 717C10A4 ucrtbase.7
EBX 7EFDE000
ESP 0018FEE8
EBP 0018FF08
ESI 00402158 UNICODE "猥
EDI 005CF048

EIP 71779670 ucrtbase._

C 0  ES 002B 32位 0(FFF
P 0  CS 0023 32位 0(FFF
A 0  SS 002B 32位 0(FFF
Z 0  DS 002B 32位 0(FFF
S 0  FS 0053 32位 7EFDD
T 0  GS 002B 32位 0(FFF
D 0
O 0  LastErr ERROR_SUCC

EFL 00000202 (NO,NB,NE,

ST0 empty 0.0
ST1 empty 0.0
```

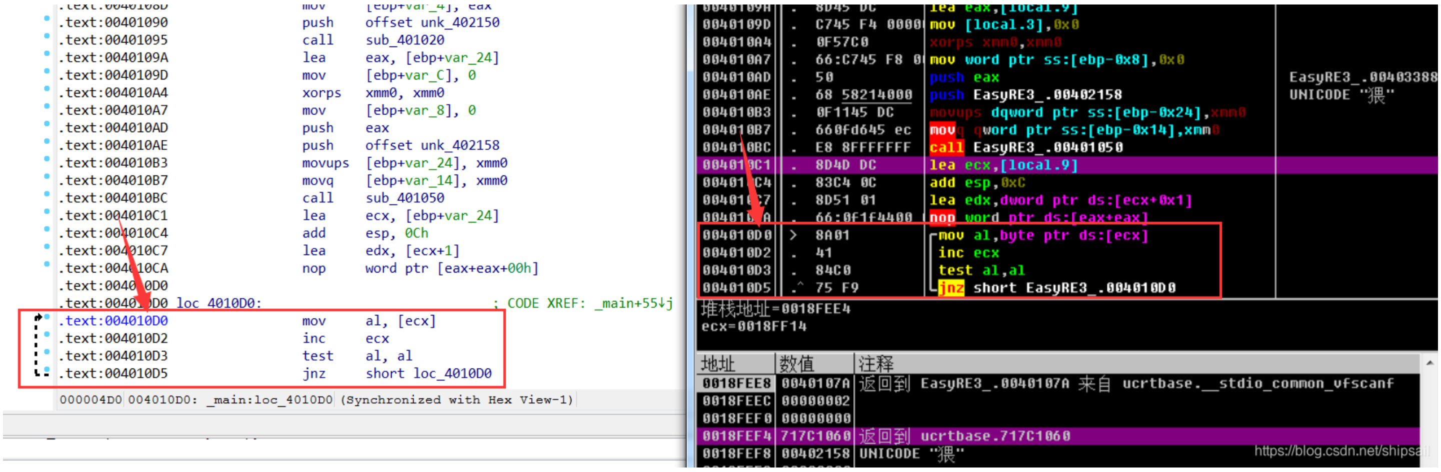
```
地址      数值      注释
0018FEE8  0040107A  返回到 EasyRE3_.0040107A 来自 ucrtbase.__stdio_common_vfscanf
0018FEEC  00000002
0018FEF0  00000000
0018FEF4  717C1060  返回到 ucrtbase.717C1060
0018FEF8  00402158  UNICODE "猥"
0018FEFC  00000000
0018FF00  0018FF14
0018FF04  005C3AA0
0018FF08  0018FF40
0018FF0C  004010C1  返回到 EasyRE3_.004010C1 来自 EasyRE3_.00401050
```

```
0018FEF0  00000000
0018FEF4  717C1060  返回到 ucrtbase.717C1060
0018FEF8  00402158  UNICODE "猥"
0018FEFC  00000000
0018FF00  0018FF14
0018FF04  005C3AA0
0018FF08  0018FF40
0018FF0C  004010C1  返回到 EasyRE3_.004010C1 来自 EasyRE3_.0
0018FF10  00402158  UNICODE "猥"
0018FF14  0018FF1C
0018FF18  00402150  EasyRE3_.00402150
0018FF1C  00000000
```

起始:18FEF4 结束:18FEF7 当前值:717C1060

No debugger
tion Instruction Data Unexplored External symbol Lumina function
IDA··· Sta··· Pse··· Str··· Hex··· Nam··· Str···
文件(F) 查看(V) 调试(D) 插件(P) 选项(T) 窗口(W) 帮助(H) [+] 快捷菜单 Tools BreakPoint->
暂停
l e m t w h c P k b r ... s

```
0040108D  .  8945 FC      mov [local.1],eax          EasyRE3_.00403388
00401090  .  68 50214000  push EasyRE3_.00402150
00401095  .  E8 86FFFFFF  call EasyRE3_.00401020
```



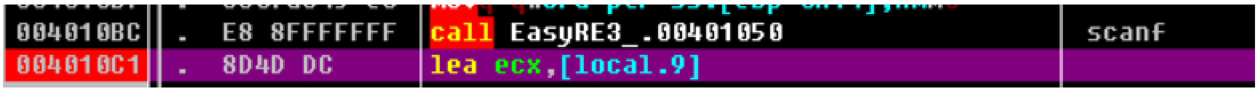
```
.text:0040108D                 mov     [ebp+var_4], eax
.text:00401090                 push    offset unk_402150
.text:00401095                 call    sub_401020
.text:0040109A                 lea     eax, [ebp+var_24]
.text:0040109D                 mov     [ebp+var_C], 0
.text:004010A4                 xorps   xmm0, xmm0
.text:004010A7                 mov     [ebp+var_8], 0
.text:004010AD                 push    eax
.text:004010AE                 push    offset unk_402158
.text:004010B3                 movups  [ebp+var_24], xmm0
.text:004010B7                 movq    [ebp+var_14], xmm0
.text:004010BC                 call    sub_401050
.text:004010C1                 lea     ecx, [ebp+var_24]
.text:004010C4                 add     esp, 0Ch
.text:004010C7                 lea     edx, [ecx+1]
.text:004010CA                 nop     word ptr [eax+eax+00h]
.text:004010D0
.text:004010D0 loc_4010D0:                             ; CODE XREF: _main+55↓j
.text:004010D0                 mov     al, [ecx]
.text:004010D2                 inc     ecx
.text:004010D3                 test    al, al
.text:004010D5                 jnz     short loc_4010D0
000004D0 004010D0: _main:loc_4010D0 (Synchronized with Hex View-1)
```

```
0040109A  .  8D45 DC        lea eax,[local.9]
0040109D  .  C745 F4 0000(  mov [local.3],0x0
004010A4  .  0F57C0         xorps xmm0,xmm0
004010A7  .  66:C745 F8 0(  mov word ptr ss:[ebp-0x8],0x0
004010AD  .  50             push eax                                       EasyRE3_.00403388
004010AE  .  68 58214000    push EasyRE3_.00402158                         UNICODE "猥"
004010B3  .  0F1145 DC      movups dqword ptr ss:[ebp-0x24],xmm0
004010B7  .  660fd645 ec    movq qword ptr ss:[ebp-0x14],xmm0
004010BC  .  E8 8FFFFFFF    call EasyRE3_.00401050
004010C1  .  8D4D DC        lea ecx,[local.9]
004010C4  .  83C4 0C        add esp,0xC
004010C7  .  8D51 01        lea edx,dword ptr ds:[ecx+0x1]
004010CA  .  66:0f1f4400 (  nop word ptr ds:[eax+eax]
004010D0  >  8A01           ┌mov al,byte ptr ds:[ecx]
004010D2  .  41             │inc ecx
004010D3  .  84C0           │test al,al
004010D5  .^ 75 F9          └jnz short EasyRE3_.004010D0
堆栈地址=0018FEE4
ecx=0018FF14
```

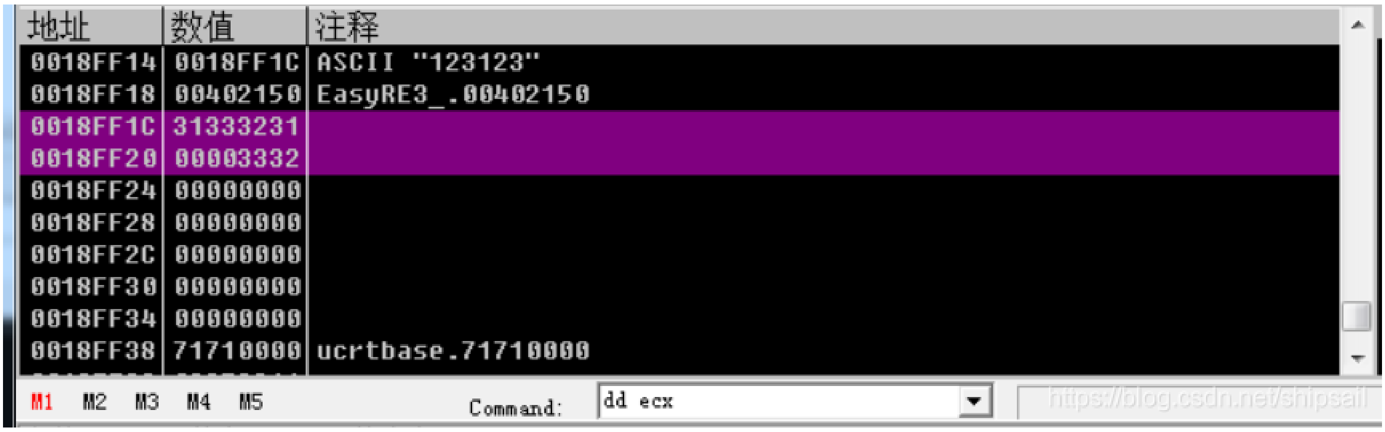
```
地址      数值      注释
0018FEE8  0040107A  返回到 EasyRE3_.0040107A 来自 ucrtbase.__stdio_common_vfscanf
0018FEEC  00000002
0018FEF0  00000000
0018FEF4  717C1060  返回到 ucrtbase.717C1060
0018FEF8  00402158  UNICODE "猥"
```

https://blog.csdn.net/shipsail

0x5 分析



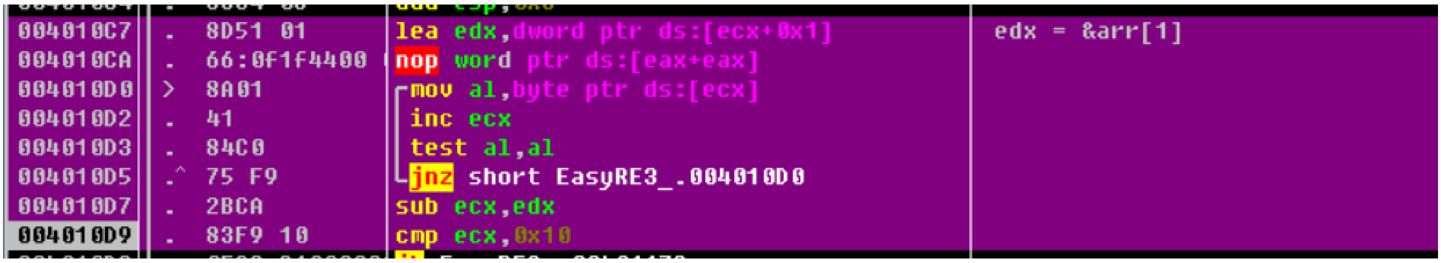
```
004010BC  .  E8 8FFFFFFF   call EasyRE3_.00401050          scanf
004010C1  .  8D4D DC       lea ecx,[local.9]
```



地址	数值	注释
0018FF14	0018FF1C	ASCII "123123"
0018FF18	00402150	EasyRE3_.00402150
0018FF1C	31333231	
0018FF20	00003332	
0018FF24	00000000	
0018FF28	00000000	
0018FF2C	00000000	
0018FF30	00000000	
0018FF34	00000000	
0018FF38	71710000	ucrtbase.71710000

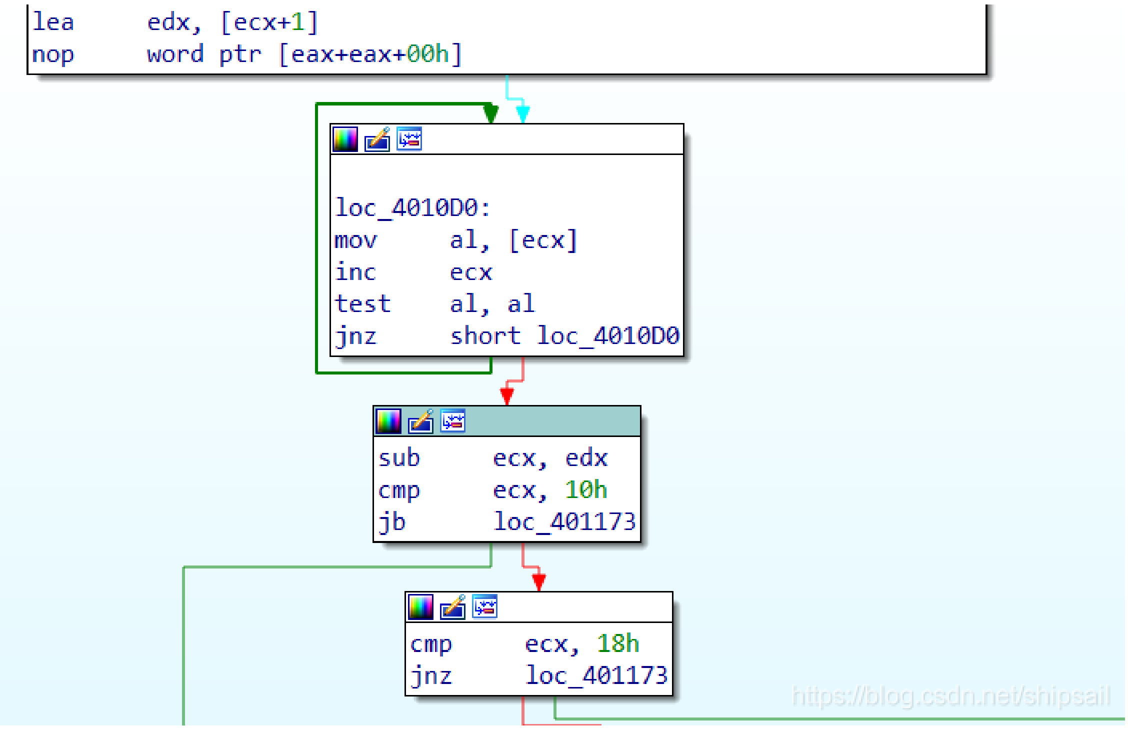
Command: dd ecx

ecx被赋值为数组的首地址（char *）



```
004010C7  .  8D51 01        lea edx,dword ptr ds:[ecx+0x1]     edx = &arr[1]
004010CA  .  66:0F1F4400    nop word ptr ds:[eax+eax]
004010D0  >  8A01          ┌mov al,byte ptr ds:[ecx]
004010D2  .  41            │inc ecx
004010D3  .  84C0          │test al,al
004010D5  .^ 75 F9         └jnz short EasyRE3_.004010D0
004010D7  .  2BCA           sub ecx,edx
004010D9  .  83F9 10        cmp ecx,0x10
```

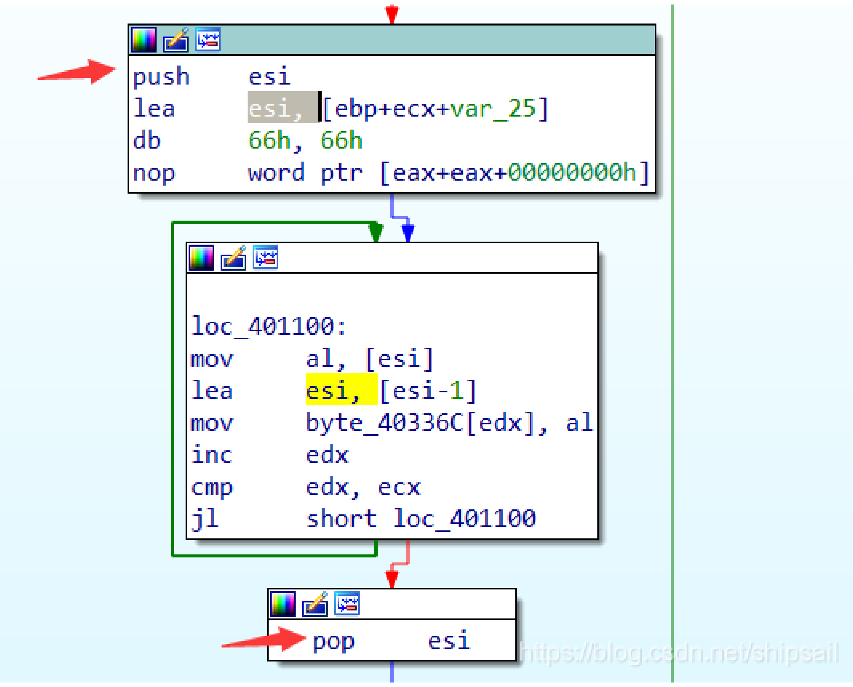
经典的字符串数组遍历（说是经典，其实我是调试后才能知道这是取字符串长度，嘿嘿嘿），再来看看IDA中的Graph。



```
lea     edx, [ecx+1]
nop     word ptr [eax+eax+00h]

loc_4010D0:
mov     al, [ecx]
inc     ecx
test    al, al
jnz     short loc_4010D0

sub     ecx, edx
cmp     ecx, 10h
jb      loc_401173

cmp     ecx, 18h
jnz     loc_401173
```

```
edx = ecx + 0x1
do{
 al = ecx
 ecx++
}while( al&al != 0)
sub ecx,edx
cmp ecx,0x10
```

最终字符串长度存放在ecx中，长度必须为`0x18位`



```
push    esi
lea     esi, [ebp+ecx+var_25]
db      66h, 66h
nop     word ptr [eax+eax+00000000h]
```

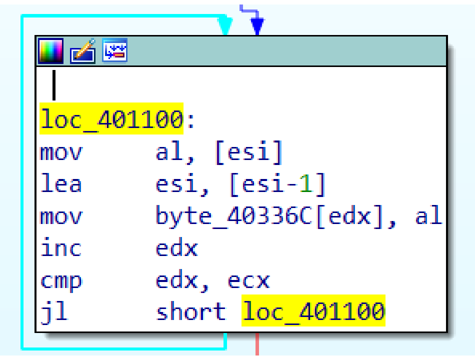
```

loc_401100:
mov     al, [esi]
lea     esi, [esi-1]
mov     byte_40336C[edx], al
inc     edx
cmp     edx, ecx
jl      short loc_401100
```

```
pop     esi
```

这里看到`push esi`和`pop esi`临时保存esi的状态，可以知道esi是一个临时变量 。`xor edx,edx`自身与自身进行异或得到结果0x0，`mov esi , ebp + ecx - 0x25`可知esi存放了数组的最后一个字符的地址。通过以下计算得到：

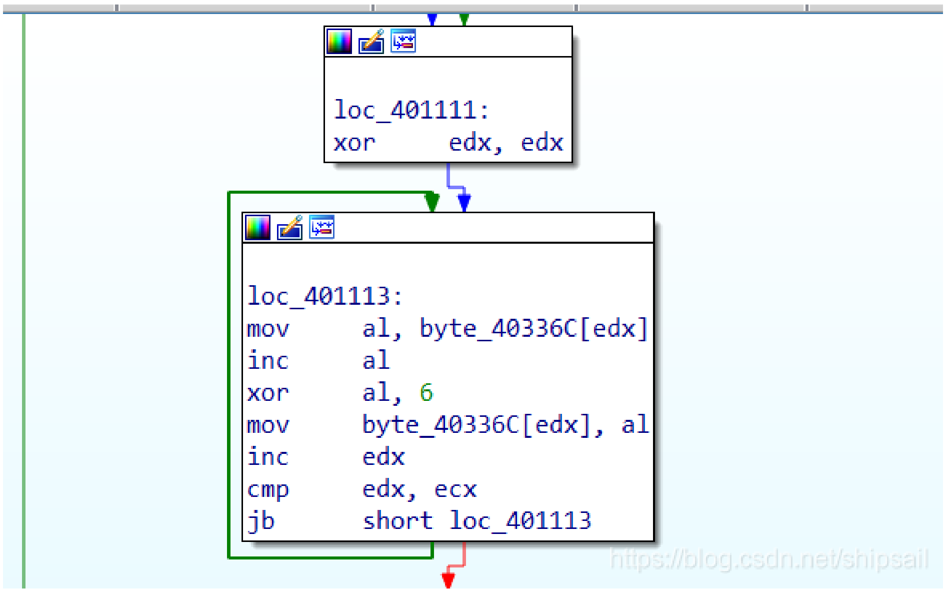
`arr[0] = ebp - 0x24`

`arr[0x18-1] = ebp - 0x24 + ecx - 0x1`



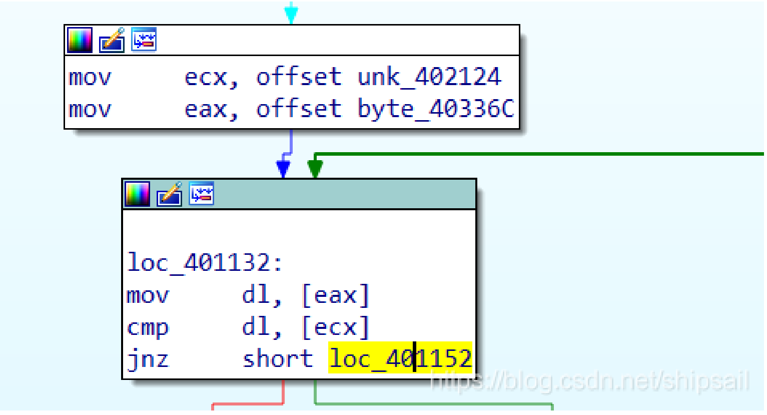
```

loc_401100:
mov     al, [esi]
lea     esi, [esi-1]
mov     byte_40336C[edx], al
inc     edx
cmp     edx, ecx
jl      short loc_401100
```

通过这个循环可以看出esi递减，edx递增，将字符串进行逆序操作。



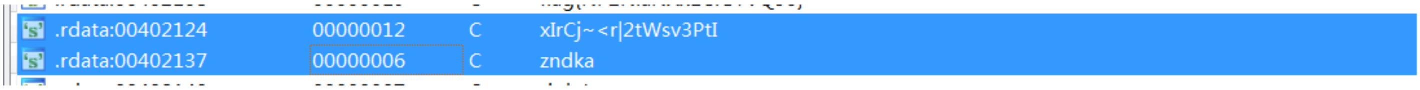
```
loc_401111:
xor     edx, edx

loc_401113:
mov     al, byte_40336C[edx]
inc     al
xor     al, 6
mov     byte_40336C[edx], al
inc     edx
cmp     edx, ecx
jb      short loc_401113
```

紧跟着的循环再次将 edx 寄存器置0，用于循环计次
将每一个元素自增1后与0x6做异或运算后存回



```
mov     ecx, offset unk_402124
mov     eax, offset byte_40336C

loc_401132:
mov     dl, [eax]
cmp     dl, [ecx]
jnz     short loc_401152
```

比较两个字符串

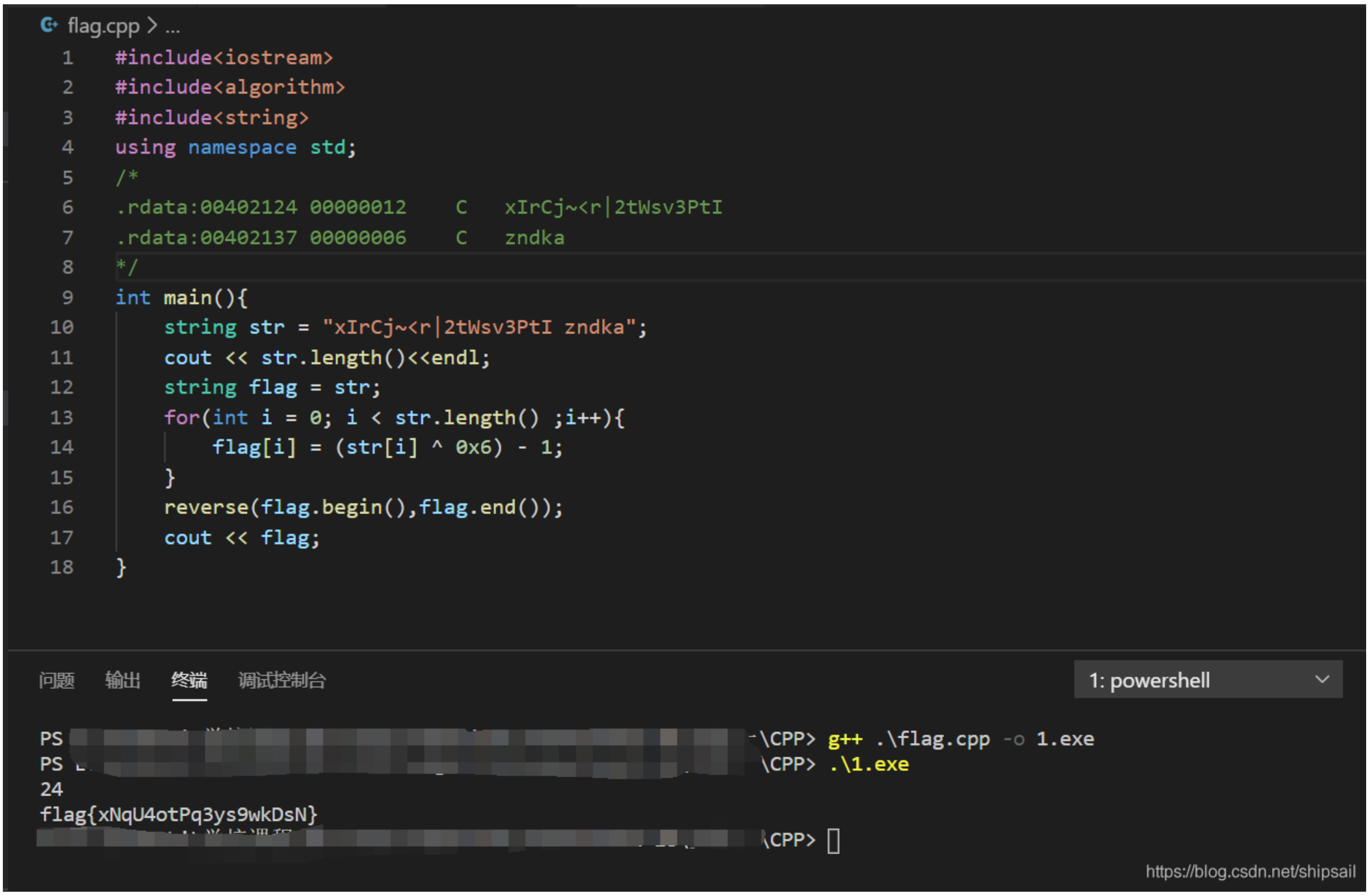


```
.rdata:00402124    00000012    C    xIrCj~<r|2tWsv3PtI
.rdata:00402137    00000006    C    zndka
```

0x6 总结

1. 输入
2. 长度限制为0x18 即24个字符
3. 逆序数组
4. 每个元素进行 +1 ^ 6 操作
5. 对比字符串是否相等

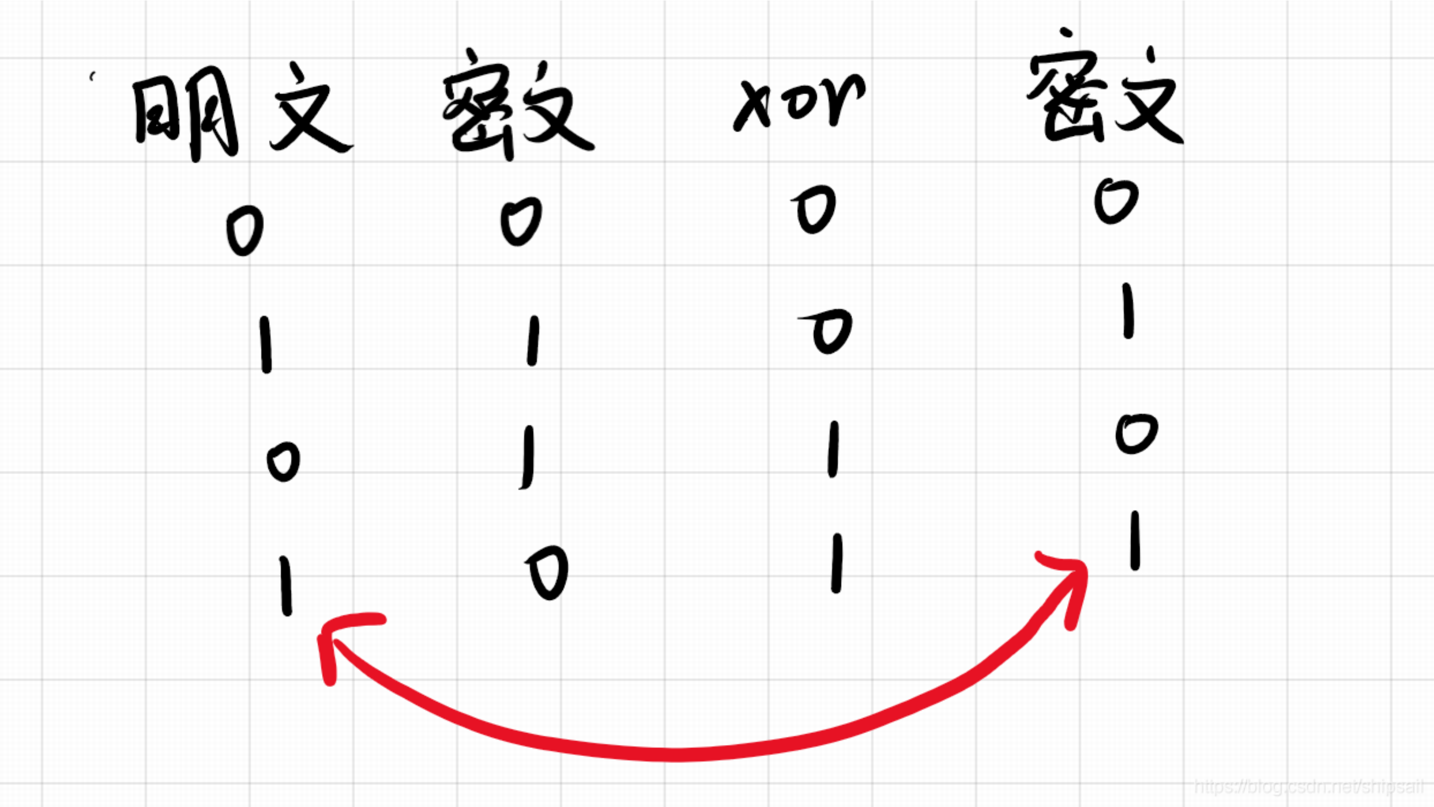
0x7 脚本编写



```cpp
#include<iostream>
#include<algorithm>
#include<string>
using namespace std;
/*
.rdata:00402124 00000012    C   xIrCj~<r|2tWsv3PtI
.rdata:00402137 00000006    C   zndka
*/
int main(){
    string str = "xIrCj~<r|2tWsv3PtI zndka";
    cout << str.length()<<endl;
    string flag = str;
    for(int i = 0; i < str.length() ;i++){
        flag[i] = (str[i] ^ 0x6) - 1;
    }
    reverse(flag.begin(),flag.end());
    cout << flag;
}
```

```
PS [redacted]\CPP> g++ .\flag.cpp -o 1.exe
PS [redacted]\CPP> .\1.exe
24
flag{xNqU4otPq3ys9wkDsN}
[redacted]\CPP> 
```

异或运算是可逆的，下面是争对1bit的运算结果



明文	密文	xor	密文
0	0	0	0
1	1	0	1
0	1	1	0
1	0	1	1

```
#include<iostream>
#include<algorithm>
#include<string>
using namespace std;
/*
.rdata:00402124 00000012 C xIrCj~<r|2tWsv3PtI
.rdata:00402137 00000006 C zndka
*/
int main(){
    string str = "xIrCj~<r|2tWsv3PtI zndka";
    cout << str.length()<<endl;
    string flag = str;
    for(int i = 0; i < str.length() ;i++){
        flag[i] = (str[i] ^ 0x6) - 1;
    }
    reverse(flag.begin(),flag.end());
    cout << flag;
}
```

恭喜您答对了

0x8 尝试IDA一键反编译

```c
int __cdecl main(int argc, const char **argv, const char **envp)
{
  unsigned int v3; // kr00_4
  signed int v4; // edx
  char *v5; // esi
  char v6; // al
  unsigned int v7; // edx
  int v8; // eax
  __int128 v10; // [esp+2h] [ebp-24h]
  __int64 v11; // [esp+12h] [ebp-14h]
  int v12; // [esp+1Ah] [ebp-Ch]
  __int16 v13; // [esp+1Eh] [ebp-8h]

  sub_401020(&unk_402150, v10);
  v12 = 0;
  v13 = 0;
  v10 = 0i64;
  v11 = 0i64;
  sub_401050((const char *)&unk_402158, (unsigned int)&v10);
  v3 = strlen((const char *)&v10);
  if ( v3 >= 0x10 && v3 == 24 )
  {
    v4 = 0;
    v5 = (char *)&v11 + 7;
    do
    {
      v6 = *v5--;
      byte_40336C[v4++] = v6;
    }
    while ( v4 < 24 );
    v7 = 0;
    do
    {
      byte_40336C[v7] = (byte_40336C[v7] + 1) ^ 6;
      ++v7;
    }
    while ( v7 < 0x18 );
    v8 = strcmp(byte_40336C, (const char *)&unk_402124);
    if ( v8 )
      v8 = -(v8 < 0) | 1;
    if ( !v8 )
    {
      sub_401020("right\n", v10);
      system("pause");
    }
  }
  return 0;
}
```

可以发现，获取长度、逆序等，如果一开始直接看IDA的反编译结果效率会更高，虽然代码看着有点奇怪...慢慢适应吧！我也是今天刚刚安装好IDA！！！继续冲压。