PE文件结构解析

20181656李涵威

1. **请解析USER32.DLL前5个导出函数的信息，要求列举AddressOfNams、AddressOfOrdinals、AddressOfFunctions的详细数据。**

首先导入USER32.DLL到winhex和lordPE

找到USER32.DLL导出表位置

```
00081D80  00 00 00 00 74 99 24 58  00 00 00 00 C8 56 08 00    t™$X    ÈV
00081D90  DC 05 00 00 EB 03 00 00  3E 03 00 00 A8 33 08 00  Ü   ë   >   ¨3
00081DA0  54 43 08 00 4C 50 08 00  10 B2 02 00 30 AE 02 00  TC  LP  ²  0®
```

可以得到关于导出表的一些信息：

名称	RVA
nName	0x000856C8
nBase	0x000005DC
NumberOfFunctions	0x000003E8
NumberOfNames	0x0000033E
AddressOfFunctions	0x000833A8
AddressOfNames	0x00084354
AddressOfNamesOrdinals	0x0008504C

1. AddressOfNamesOrdinals

选择先检查AddressOfNamesOrdinals，是因为可能会存在前几个函数并不具备名字的情况。

首先使用LordPE计算它的RAW，RAW=0x00083A4C

```
00083A40  2E 8E 08 00 38 8E 08 00  43 8E 08 00 02 00 03 00  .Ž  8Ž  CŽ
00083A50  04 00 05 00 06 00 07 00  08 00 09 00 0A 00 0B 00
00083A60  0C 00 0D 00 0E 00 0F 00  10 00 11 00 12 00 13 00
00083A70  14 00 15 00 16 00 17 00  18 00 19 00 1A 00 1B 00
00083A80  1C 00 1D 00 1E 00 1F 00  20 00 21 00 22 00 23 00              ! " #
00083A90  24 00 25 00 26 00 27 00  28 00 29 00 2A 00 2B 00  $ % & ' ( ) * +
```

可以看见它的序号是从2开始，而AddressOfNamesOrdinals和AddressOfNames一一对应，**说明前两个函数并没有名字。**同时函数序号=nBase+NamesOrdinals。

2. AddressOfFunctions

首先用LordPE计算它在文件中位置，得出RAW=0x00081DA8

```
00081DA0  54 43 08 00 4C 50 08 00  10 B2 02 00 30 AE 02 00  TC  LP  ²  0®
00081DB0  10 83 00 00 F0 AC 02 00  A0 35 02 00 EC 7C 01 00   ƒ  ð¬   5  ì|
00081DC0  70 F3 07 00 C4 21 04 00  F0 7C 00 00 F8 BF 01 00  pó  Ä!  ð|  ø¿
00081DD0  8C A8 07 00 D0 D8 07 00  8C 36 01 00 70 1B 04 00  Œ¨  ÐØ  Œ6  p
00081DE0  90 D1 00 00 B4 CC 00 00  10 6E 01 00 10 AD 02 00   Ñ  ´Ì   n   -
```

可以看见第一个函数的RVA等于0x0002B210，后续同理

3. AddressOfNames

还是用LordPE计算它在文件中位置，RAW=0x00082D54

```
00082D50  2C 06 04 00 D3 56 08 00  EA 56 08 00 05 57 08 00  ,   ÓV  êV   W
00082D60  16 57 08 00 29 57 08 00  34 57 08 00 4E 57 08 00   W  )W  4W  NW
00082D70  67 57 08 00 75 57 08 00  7E 57 08 00 8A 57 08 00  gW  uW  ~W  ŠW
00082D80  96 57 08 00 AB 57 08 00  BD 57 08 00 D1 57 08 00  -W  «W  ½W  ÑW
00082D90  DC 57 08 00 E7 57 08 00  F8 57 08 00 0F 58 08 00  ÜW  çW  øW   X
00082DA0  27 58 08 00 41 58 08 00  5B 58 08 00 73 58 08 00  'X  AX  [X  sX
```

可以得到指向这些名称的地址(RVA)，采用第一个进行验证

第一个RVA=0x000856D3，RAW=0x000840D3

```
000840C0  47 03 48 03 49 03 4A 03  55 53 45 52 33 32 2E 64  G H I J USER32.d
000840D0  6C 6C 00 41 63 74 69 76  61 74 65 4B 65 79 62 6F  ll ActivateKeybo
000840E0  61 72 64 4C 61 79 6F 75  74 00 41 64 64 43 6C 69  ardLayout AddCli
000840F0  70 62 6F 61 72 64 46 6F  72 6D 61 74 4C 69 73 74  pboardFormatList
00084100  65 6E 65 72 00 41 64 6A  75 73 74 57 69 6E 64 6F  ener AdjustWindo
00084110  77 52 65 63 74 00 41 64  6A 75 73 74 57 69 6E 64  wRect AdjustWind
00084120  6F 77 52 65 63 74 45 78  00 41 6C 69 67 6E 52 65  owRectEx AlignRe
00084130  63 74 73 00 41 6C 6C 6F  77 46 6F 72 65 67 72 6F  cts AllowForegro
```

可以看见右边信息已经找到了函数名称。

4. 综上

根据上面分析可以得到前五个函数的信息为

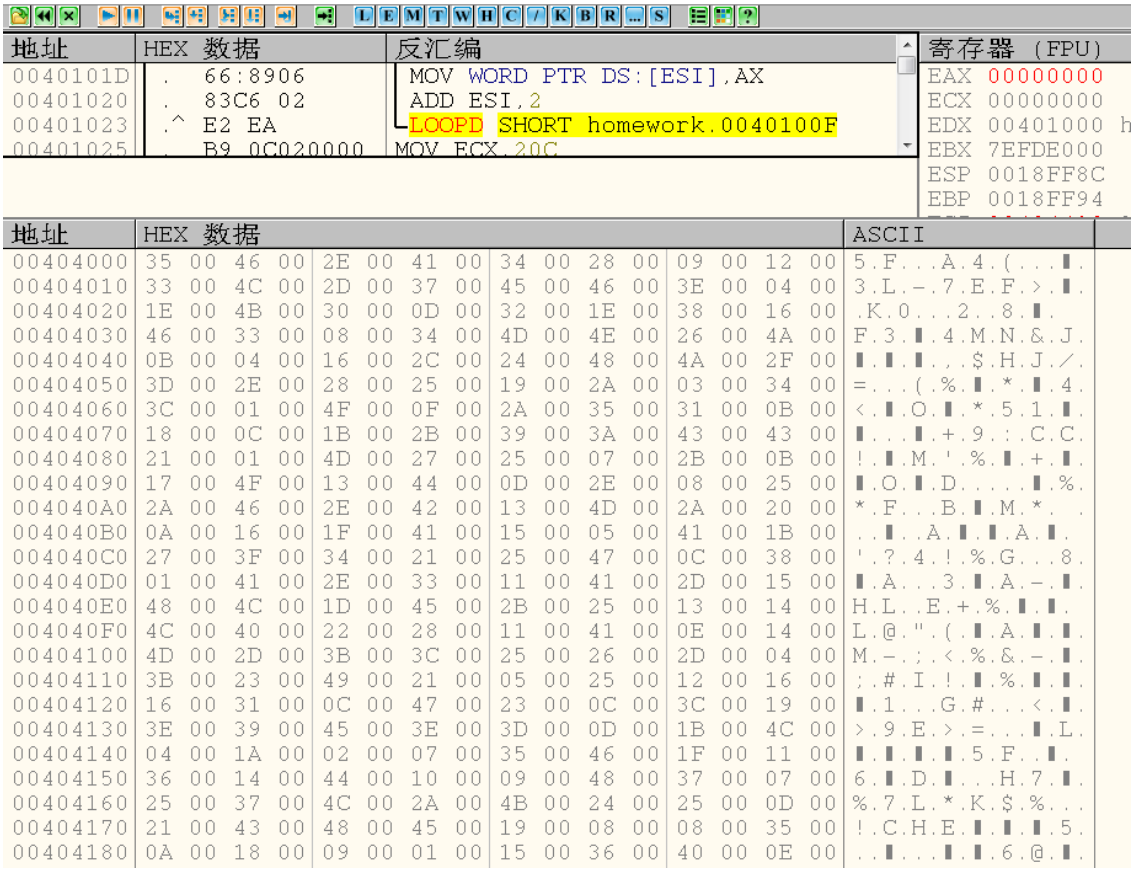
函数编号	函数RVA	函数名称
0x05DC	0X0002B210	无
0x05DD	0x0002AE30	无
0x05DE	0x00008310	ActivateKeyboardLayout
0x05DF	0x0002ACF0	AddClipboardFormatListener
0x05E0	0x000235A0	AdjustWindowRect

---

2. **编写程序并生成exe文件，要求定义1048个字节长度的word数组，在程序中对该数组赋随机数，然后查找该数组的最小值，并调用MessageBox函数和ExitProcess函数。**

```
include C:\irvine\Irvine32.inc
includelib C:\irvine\Kernel32.lib
includelib C:\irvine\User32.lib
includelib C:\irvine\Irvine32.lib
.stack 4096
.data
array word 524 DUP(?)
str1 byte "hello",0
str2 byte "dialog",0
.code
main PROC
    mov ecx,lengthof array
    mov esi,offset array
    call Randomize;随机数种子初始化
L1:
    mov eax,0
    mov ax,50h;设置随机数范围0-50
    call RandomRange;使得ax获得一个随机数
    mov [esi],ax
    add  esi,type word
    loop L1
```

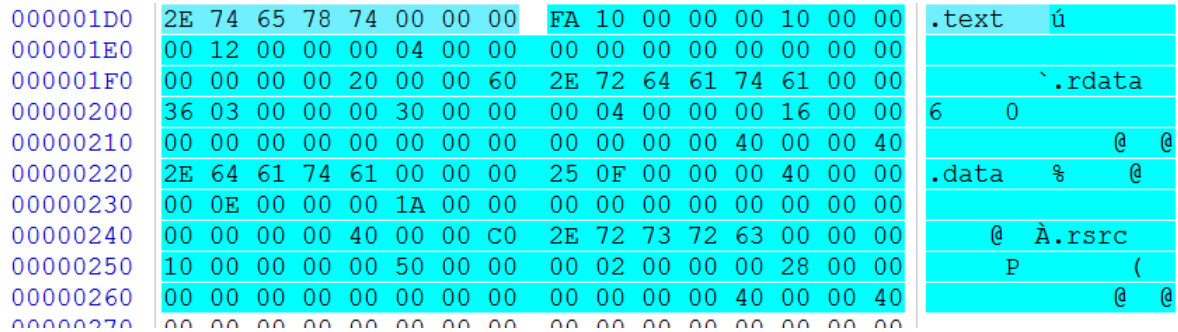
```
22
23      mov ecx,lengthof array
24      mov esi,offset array
25      mov ax,50h
26  L2:;循环遍历得到最小值
27      cmp [esi],ax
28      JB change;遇见更小的跳入change把值给ax
29      add esi,type word
30      loop L2
31
32      INVOKE MessageBox,NULL,offset str1,offset str2,MB_OK;调用两次是担心结束
    时在L2处结束但是代码继续向下往change执行,导致ECX值出错。
33      INVOKE ExitProcess,0
34  change:
35      mov ax,[esi]
36      add esi,type word
37      loop L2
38
39      INVOKE MessageBox,NULL,offset str1,offset str2,MB_OK
40      INVOKE ExitProcess,0
41  main endp
42  end main
```



可以看见成功赋值，同时eax的后16位置ax得到最小值0。函数满足需求。

3. **请解析题2生成exe文件的节表，加载前、后导入函数的详细信息。**

- 首先是关于节表信息



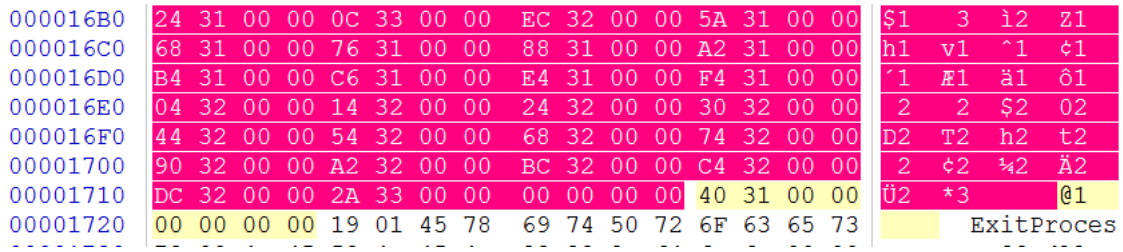
可以得到：

节表名称	内存中偏移量	内存中地址	文件中偏移量	文件中大小
.text	0x00001000	0x000010FA	0x00000400	0x00001200
.rdata	0x00003000	0x00000336	0x00001600	0x00000400
.data	0x00004000	0x00000F25	0x00001A00	0x00000E00
.rsrc	0x00005000	0x00000010	0x00002800	0x00000200

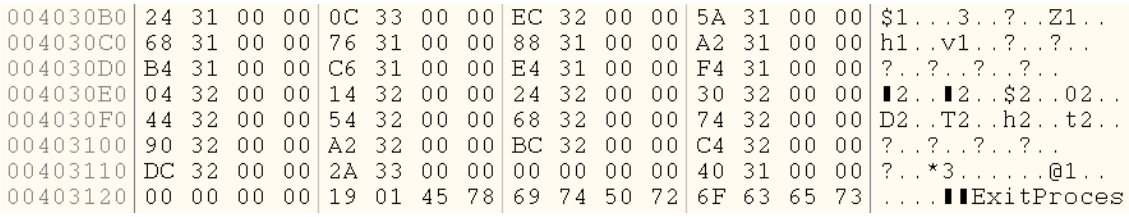
- 关于加载前、后导入函数的详细信息

  主要的关注点在于`OriginalFirstThunk`和`FirstThunk`，**在winhex中分析为导入前的信息，在OllyDbg中分析为导入后的信息。**

  1. 首先是关于`OriginalFirstThunk`
  - 在winhex中



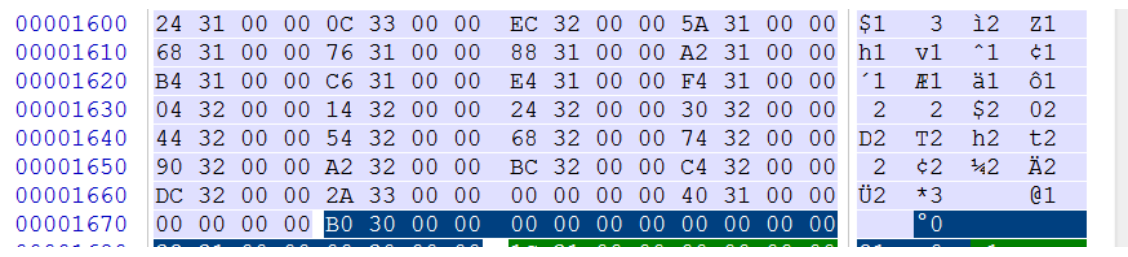
  - 在OD中



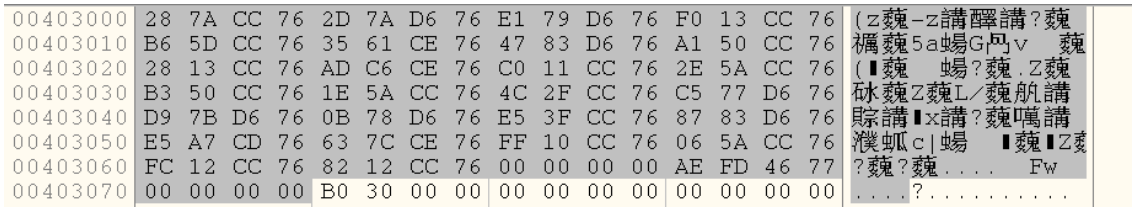
  可以看见他们没有任何区别，因为无论加载前、后，它都指向函数编号和名称

  2. 其次是关于`FirstThunk`
  - 在winhex中

    可以发现导入前`FirstThunk`和`OriginalFirstThunk`一样。



  - 在OD中



可以看见这时候就出现不同了，因为在加载后，调用的函数调入内存，FirstThunk则不再跟OriginalFirstThunk指向相同的地方，而是指向了函数执行地址。可以以MessageBox函数为例子，如下图所示40306C地址处的DWORD值是它的函数地址，对比上图的得到验证，其函数地址为0x7746FDAE