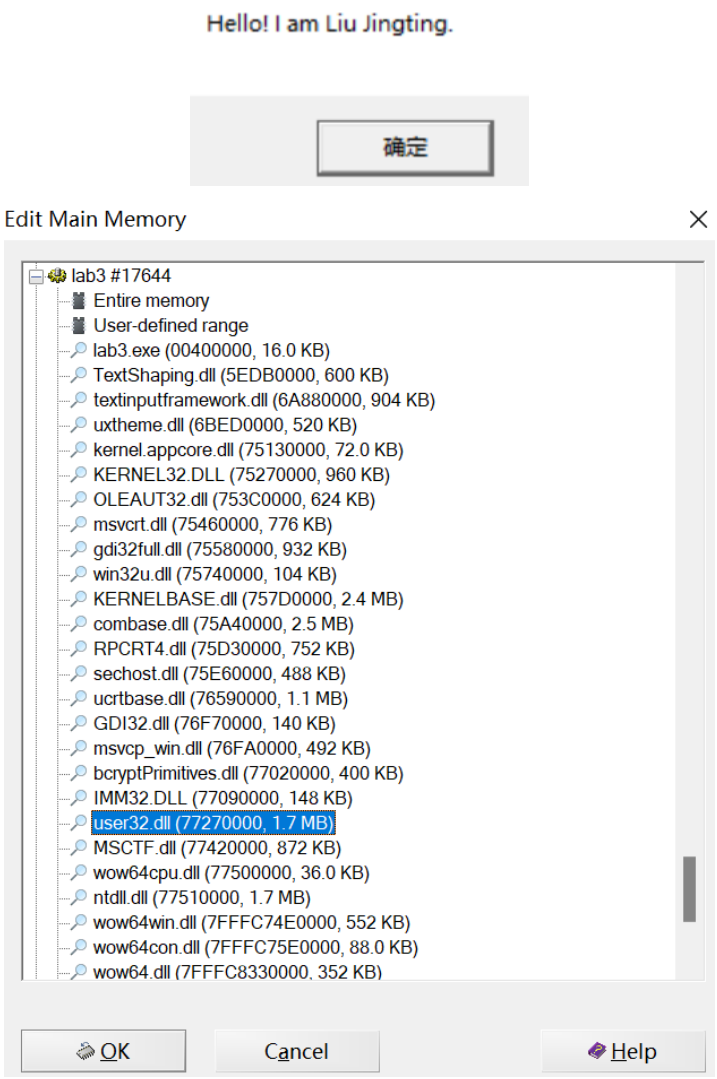
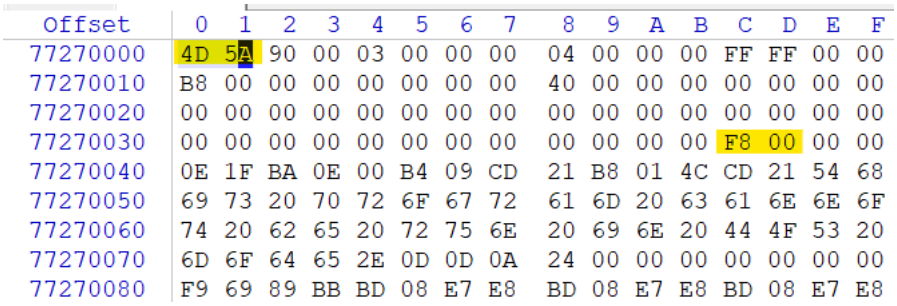
PE 文件结构解析作业

一、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。

1. 首先运行 lab3.exe 文件，然后在 WinHex 中进入 USER32.DLL 进程中去

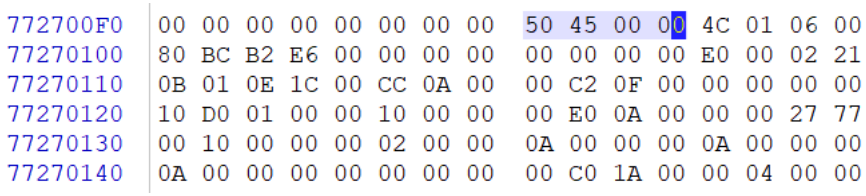


2. 首先通过 5A4D 定位到 MZ 头的位置，找到偏移 3C 的 PE 头偏移地址 00F8



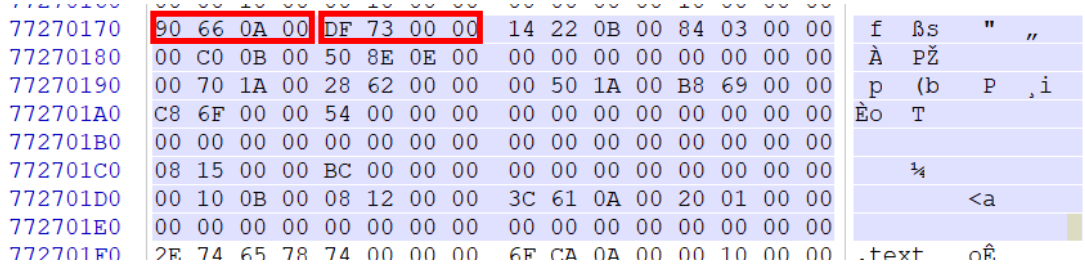
Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
77270000	4D	5A	90	00	03	00	00	00	04	00	00	00	FF	FF	00	00
77270010	B8	00	00	00	00	00	00	00	40	00	00	00	00	00	00	00
77270020	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
77270030	00	00	00	00	00	00	00	00	00	00	00	00	F8	00	00	00
77270040	0E	1F	BA	0E	00	B4	09	CD	21	B8	01	4C	CD	21	54	68
77270050	69	73	20	70	72	6F	67	72	61	6D	20	63	61	6E	6E	6F
77270060	74	20	62	65	20	72	75	6E	20	69	6E	20	44	4F	53	20
77270070	6D	6F	64	65	2E	0D	0D	0A	24	00	00	00	00	00	00	00
77270080	F9	69	89	BB	BD	08	E7	E8	BD	08	E7	E8	BD	08	E7	E8

3. 定位到 00F8 的位置，从这里开始是 PE 头，这里可以找到 PE 头标识符 4550。



772700F0	00	00	00	00	00	00	00	00	50	45	00	00	4C	01	06	00
77270100	80	BC	B2	E6	00	00	00	00	00	00	00	00	E0	00	02	21
77270110	0B	01	0E	1C	00	CC	0A	00	00	C2	0F	00	00	00	00	00
77270120	10	D0	01	00	00	10	00	00	00	E0	0A	00	00	00	27	77
77270130	00	10	00	00	00	02	00	00	0A	00	00	00	0A	00	00	00
77270140	0A	00	00	00	00	00	00	00	00	C0	1A	00	00	04	00	00

4. 数据目录表 DataDirectory 是从 PE 头偏移 78h 字节开始的，所以找到 170h 的位置，如下图所示，标紫色的部分就是数据目录表。



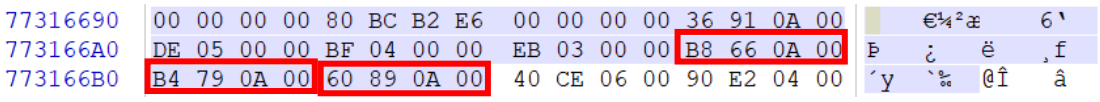
77270170	90	66	0A	00	DF	73	00	00	14	22	0B	00	84	03	00	00
77270180	00	C0	0B	00	50	8E	0E	00	00	00	00	00	00	00	00	00
77270190	00	70	1A	00	28	62	00	00	00	50	1A	00	B8	69	00	00
772701A0	C8	6F	00	00	54	00	00	00	00	00	00	00	00	00	00	00
772701B0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
772701C0	08	15	00	00	BC	00	00	00	00	00	00	00	00	00	00	00
772701D0	00	10	0B	00	08	12	00	00	3C	61	0A	00	20	01	00	00
772701E0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
772701F0	2E	74	65	78	74	00	00	00	6F	CA	0A	00	00	10	00	00

从数据目录表里可以找到导出表的 RVA 和导出表的大小（红色方框标注处）：

- **导出表 RVA：000A6690h**
- **导出表大小：000073DFh**

5. 接下来需要找到导出表，因为这里已经是在进程里面了，所以直接用 77270000+A6690=77316690 的位置，从这里开始 24h 为导出表。

如下图所示：



根据导出表信息可以得到：

- AddressOfNames ： A79B4h（RVA）

  →773179B4h（进程中的地址）

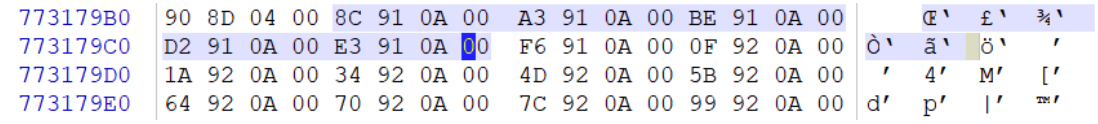
- AddressOfNameOrdinals ： A8960h

- AddressOfFunctions ： A66B8h（RVA）

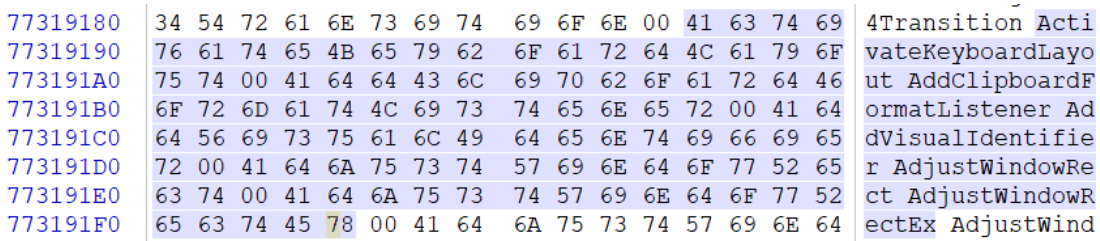
  →773166B8h（进程中的地址）

6. 根据 AddressOfNames 和 AddressOfFunctions 的地址定位到这两个表。如下图所示：

➢ AddressOfNames（每项 32 位）



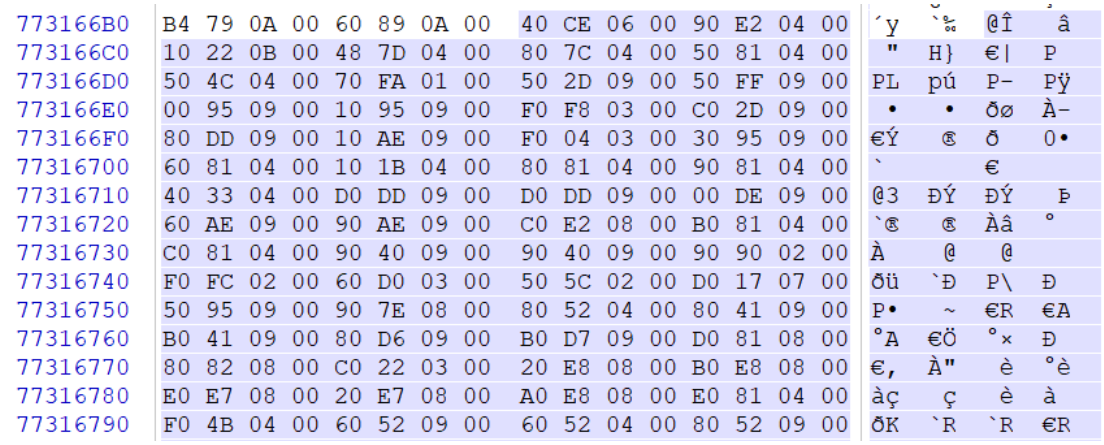
对应去查找可以找到如下函数名：



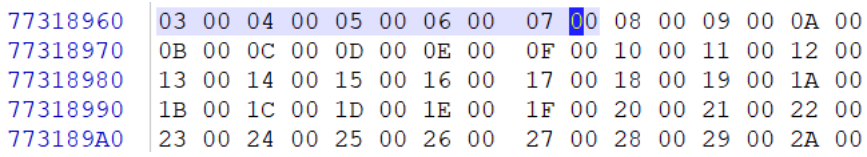
所以前五个导出函数信息如下：

函数序号	AddressOfNames（RVA）	函数名在进程中的地址	函数名
1	000A918C	7731918C	ActivateKeyboardLayout
2	000A91A3	773191A3	AddClipboardFormatListener
3	000A91BE	773191BE	AddVisualIdentifier
4	000A91D2	773191D2	AdjustWindowRect
5	000A91E3	773191E3	AdjustWindowRectEx

- AddressOfFunctions（每项 32 位）



- AddressOfNameOrdinals （每项 16 位）



二、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。

（一）代码

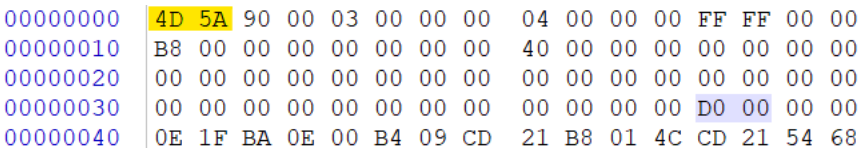
```
.386
.model flat,stdcall
option casemap:none
.stack 4096

include irvine32.inc
includelib irvine32.lib
includelib user32.lib
includelib kernel32.lib
includelib masm32.lib
ExitProcess PROTO, dwExitCode:DWORD

.data
n=512
arr word n DUP(?)
msg db 'Hello! I am Liu Jingting.', 0
min word 0FFFh

.code
main proc
    invoke MessageBox,NULL,offset msg, NULL, MB_OK
    mov esi,offset arr
    mov ecx,lengthof arr
    mov bx,min

L1:

    ;mov eax,10000h
    push 1000h
    call RandomRange
    mov [esi],ax

    cmp bx,ax
    jb L2
    mov bx,ax
```

```
L2:
    add esi,TYPE WORD
    sub ecx,1
    cmp cx,0
    ja L1

    mov min,bx
    invoke ExitProcess,NULL
    ret

main endp
end main
```

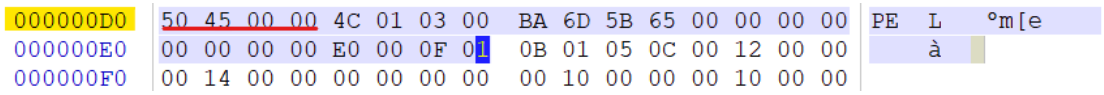
（二）运行结果

三、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。

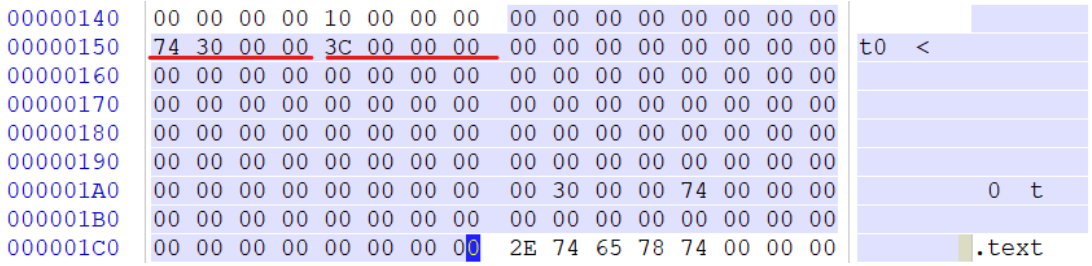
1. 使用 WinHex64 打开 homework2.exe 文件，如下图所示，5A4D 定位到 MZ 文件头的位置，距 MZ 文件头偏移 3Ch 位置的数值指向 PE 文件头的位置，这里是 D0h。



2. 定位到 00D0 的位置，从这里开始是 PE 头，这里可以找到 PE 头标识符 4550，后边跟着的 14h 个字节是 file_header。



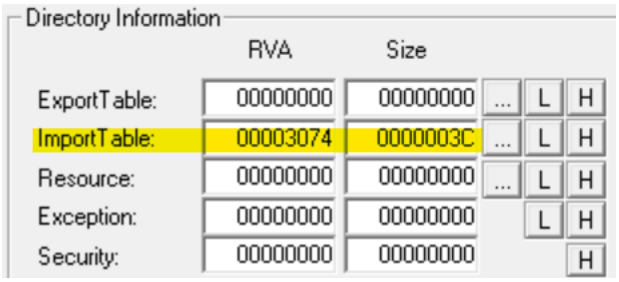
3. 数据目录表 DataDirectory 是从 PE 头偏移 78h 字节开始的，所以找到 140h 的位置，如下图所示，标紫色的部分就是数据目录表。



根据上图可以得到导出表的相关信息：

- **导出表 RVA：3074h**
- **导出表大小：3Ch**

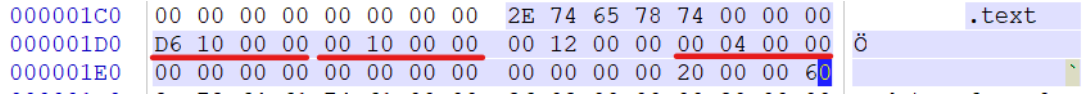
在 LordPE 中验证一下没有问题，如下图所示：



4.分析节表

因为现在 PE 文件是在文件状态下，还没有进入进程中，所以需要先判断导入表在哪个节表中，在计算它在文件中的偏移地址。

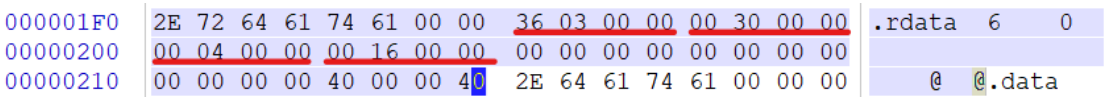
首先，我们查看 text 节的信息：



可以得到：

- VirtualSize:10D6h
- RVA：1000h
- 在文件中的偏移： 400h
- 在文件中对齐后的尺寸：1200h

然后，我们查看 rdata 节：



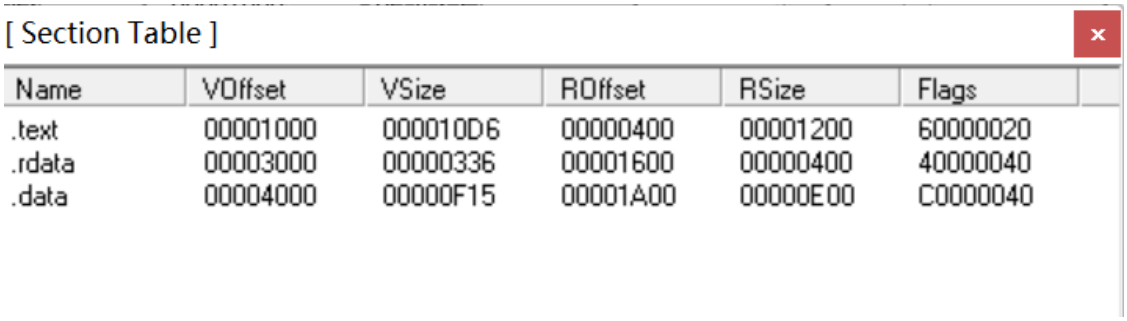
- VirtualSize:336h
- RVA：3000h
- 在文件中的偏移： 400h
- 在文件中对齐后的尺寸：1600h

最后，我们查看 data 节：

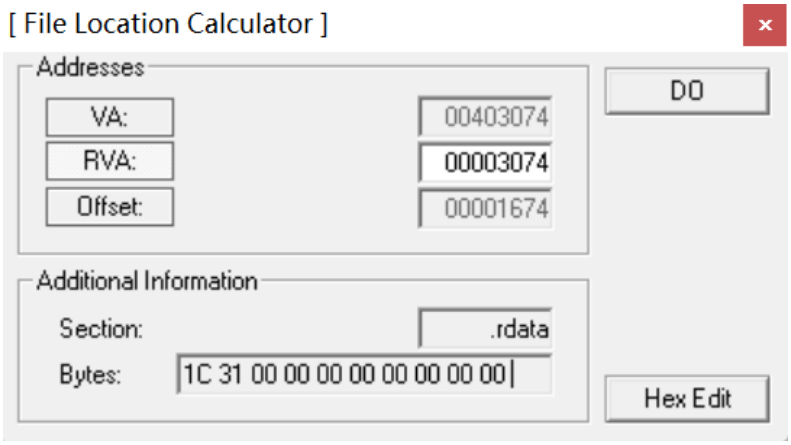
```
00000210  00 00 00 00 40 00 00 40  2E 64 61 74 61 00 00 00    @  @.data
00000220  15 0F 00 00 00 40 00 00  00 0E 00 00 00 1A 00 00       @
00000230  00 00 00 00 00 00 00 00  00 00 00 00 40 00 00 C0                @  À
```

- VirtualSize:F15h
- RVA：4000h
- 在文件中的偏移： E00h
- 在文件中对齐后的尺寸：1A00h

使用 LordPE 可以得到所有节表的信息，与上述分析结果对比验证，分析结果正确，LordPE 结果如下图所示：



5. 由上述分析可知，导入表是在 rdata 节中，所以计算导入表在文件中的偏移位置为：3074h-3000h+1600h=1674h。在 LordPE 中计算验证一下，如下图：



（1）USER32.DLL

【加载前】

```
1. ImageImportDescriptor:
 OriginalFirstThunk:   0x0000311C
 TimeDateStamp:        0x00000000   (GMT: Thu Jan 01 00:00:00 1970)
 ForwarderChain:       0x00000000
 Name:                 0x00003132   ("USER32.dll")
 FirstThunk:           0x0000306C

 Ordinal/Hint API name
 ------------ ------------------------------------
 0x020E       "MessageBoxA"
```

```
00001670  00 00 00 00 1C 31 00 00  00 00 00 00 00 00 00 00      1
00001680  32 31 00 00 6C 30 00 00  B0 30 00 00 00 00 00 00  21  10  °0
```

这时 OriginalFirstThunk 是 311Ch（171Ch），FirstThunk 是 306Ch（166Ch）。

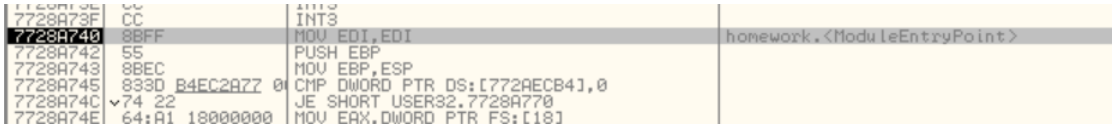
```
00001660  DE 32 00 00 FE 32 00 00  00 00 00 00 24 31 00 00  Þ2  þ2      $1
00001710  DE 32 00 00 FE 32 00 00  00 00 00 00 24 31 00 00  Þ2  þ2      $1
```

如上图所示，可以看到加载前 OriginalFirstThunk 和 FirstThunk 的值是一样的。

【加载后】

```
00403060  30 45 A4 76 20 45 A4 76  00 00 00 00 40 A7 28 77
00403070  00 00 00 00 1C 31 00 00  00 00 00 00 00 00 00 00
00403080  32 31 00 00 6C 30 00 00  B0 30 00 00 00 00 00 00
00403090  00 00 00 00 28 33 00 00  00 30 00 00 00 00 00 00
004030A0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
004030B0  16 32 00 00 1C 33 00 00  3E 31 00 00 4C 31 00 00
004030C0  5A 31 00 00 68 31 00 00  7A 31 00 00 94 31 00 00
004030D0  A6 31 00 00 B8 31 00 00  D6 31 00 00 E6 31 00 00
004030E0  F6 31 00 00 06 32 00 00  22 32 00 00 36 32 00 00
004030F0  46 32 00 00 5A 32 00 00  66 32 00 00 82 32 00 00
00403100  94 32 00 00 AE 32 00 00  B6 32 00 00 CE 32 00 00
00403110  DE 32 00 00 FE 32 00 00  00 00 00 00 24 31 00 00
00403120  00 00 00 00 0E 02 4D 65  73 73 61 67 65 42 6F 78
```

可以看出这时候 FirstThunk 中的值已经变成 7728A740h，指向函数入口地址，在 OllyDbg 里逐步运行程序发现 MessageBoxA 执行的首地址确实是 7728A740h。如下图所示：



回到 WinHex，在 403124h 处找到函数名，如下图所示：

```
00403120  00 00 00 00 0E 02 4D 65  73 73 61 67 65 42 6F 78      MessageBox
00403130  41 00 55 53 45 52 33 32  2E 64 6C 6C 00 00 19 01  A USER32.dll
```

（2）KERNEL32.DLL

【加载前】

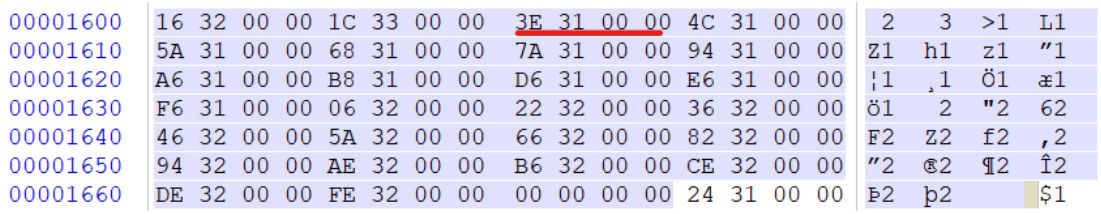
```
2. ImageImportDescriptor:
 OriginalFirstThunk:   0x000030B0
 TimeDateStamp:        0x00000000   (GMT: Thu Jan 01 00:00:00 1970)
 ForwarderChain:       0x00000000
 Name:                 0x00003328   ("KERNEL32.dll")
 FirstThunk:           0x00003000

 Ordinal/Hint API name
 ------------ ----------------------------------------
 0x0348        "LocalFree"
 0x0525        "WriteFile"
 0x0119        "ExitProcess"
 0x0052        "CloseHandle"
 0x0088        "CreateFileA"
 0x015D        "FormatMessageA"
 0x0156        "FlushConsoleInputBuffer"
 0x0186        "GetCommandLineA"
0x01AC        "GetConsoleMode"
0x01B2        "GetConsoleScreenBufferInfo"
0x0202        "GetLastError"
0x0203        "GetLocalTime"
0x0264        "GetStdHandle"
0x0277        "GetSystemTime"
0x038B        "PeekConsoleInputA"
0x03B4        "ReadConsoleA"
0x03B5        "ReadConsoleInputA"
0x03C0        "ReadFile"
0x0431        "SetConsoleCursorPosition"
0x043D        "SetConsoleMode"
0x0446        "SetConsoleTextAttribute"
0x04B2        "Sleep"
0x04BD        "SystemTimeToFileTime"
0x051A        "WriteConsoleA"
 0x0521        "WriteConsoleOutputCharacterA"
 0x0520        "WriteConsoleOutputAttribute"
```

从 WinHex 中可以看到：

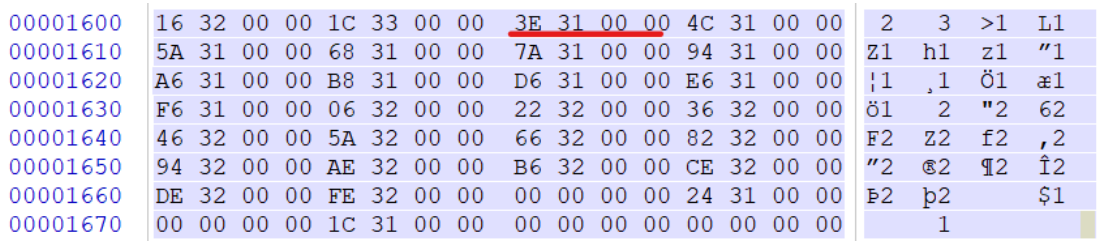
这时 OriginalFirstThunk 是 30B0h（16B0h），FirstThunk 是 3000h（1600h）。

```
00001680  32 31 00 00 6C 30 00 00  B0 30 00 00 00 00 00 00  21  10  °0
00001690  00 00 00 00 28 33 00 00  00 30 00 00 00 00 00 00      (3   0
```

因为 ExitProcess 是 KERNEL.DLL 中第三个函数，所以 ExitProcess 函数的函数名地址在文件偏移 1608h 处，如下图所示：

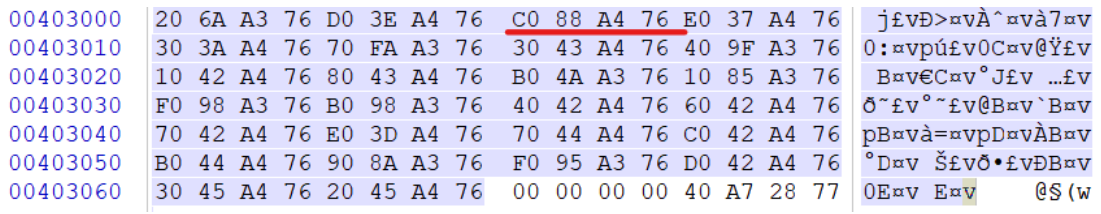


再看一下这时 FirstThunk 的值：

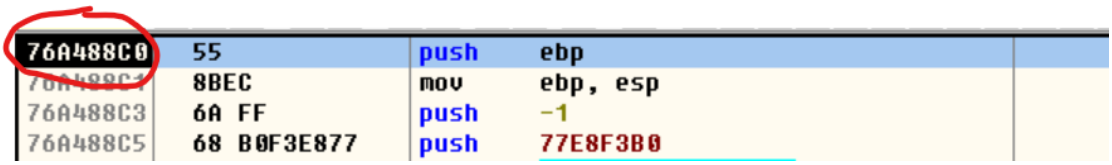


如上图所示，可以看到加载前 OriginalFirstThunk 和 FirstThunk 的值是一样的。

【加载后】



可以看出这时候 FirstThunk 中的值已经变成 76A488C0h，指向函数入口地址，在 OllyDbg 里逐步运行程序发现 ExitProcess 函数执行的首地址确实是 76A488C0h。如下图所示：



回到 WinHex，在 40313Eh 处找到函数名，如下图所示：