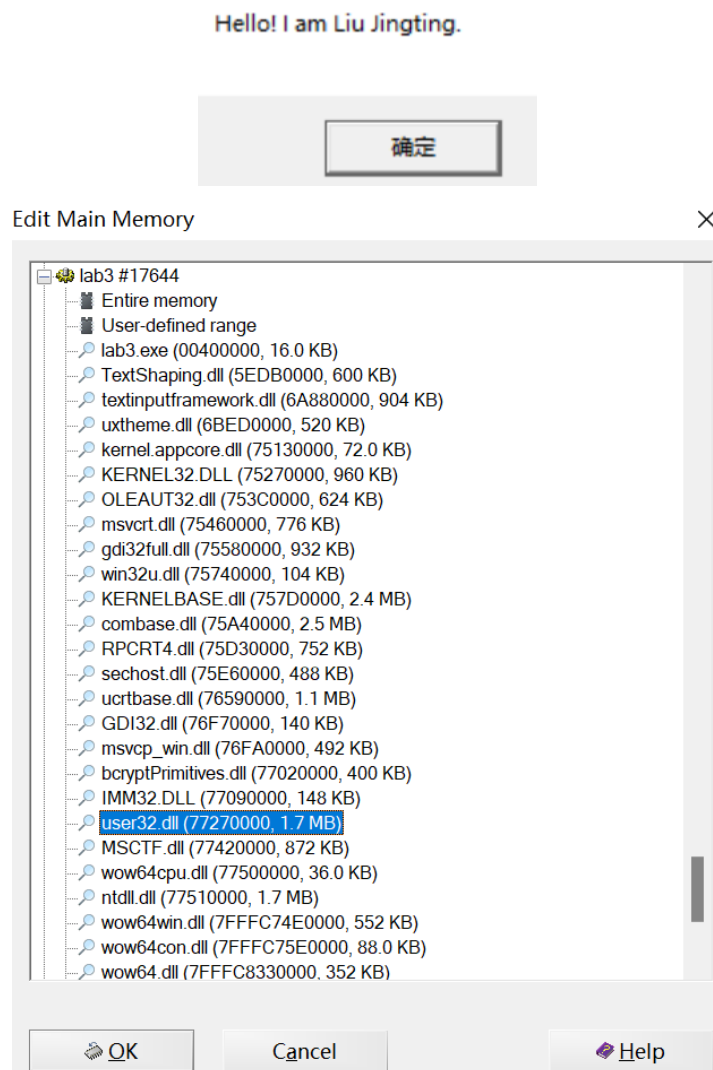


PE 文件结构解析作业

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



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 | f | ßs | " | " |
| 77270180 | 00 | C0 | 0B | 00 | 50 | 8E | 0E | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | À | PŽ | | |
| 77270190 | 00 | 70 | 1A | 00 | 28 | 62 | 00 | 00 | 00 | 50 | 1A | 00 | B8 | 69 | 00 | 00 | p | (b | P | ,i |
| 772701A0 | C8 | 6F | 00 | 00 | 54 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | Èo | T | | |
| 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 | | ¼ | | |
| 772701D0 | 00 | 10 | 0B | 00 | 08 | 12 | 00 | 00 | 3C | 61 | 0A | 00 | 20 | 01 | 00 | 00 | | <a | | |
| 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 | | .text | oË | |

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：

| | | | |
|----------|-------------------------|-------------------------|------------|
| 77316690 | 00 00 00 00 80 BC B2 E6 | 00 00 00 00 36 91 0A 00 | €¼²æ 6\ |
| 773166A0 | DE 05 00 00 BF 04 00 00 | EB 03 00 00 B8 66 0A 00 | È ÿ ë ,f |
| 773166B0 | B4 79 0A 00 60 89 0A 00 | 40 CE 06 00 90 E2 04 00 | 'y `‰ @î â |

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)

| | | | |
|----------|-------------------------|-------------------------|-------------|
| 773179B0 | 90 8D 04 00 8C 91 0A 00 | A3 91 0A 00 BE 91 0A 00 | Œ\ £\ ¼\ |
| 773179C0 | D2 91 0A 00 E3 91 0A 00 | F6 91 0A 00 0F 92 0A 00 | ò\ ã\ ö\ ' |
| 773179D0 | 1A 92 0A 00 34 92 0A 00 | 4D 92 0A 00 5B 92 0A 00 | ' 4' M' [' |
| 773179E0 | 64 92 0A 00 70 92 0A 00 | 7C 92 0A 00 99 92 0A 00 | d' p' ' ¯' |

对应去查找可以找到如下函数名：

| | | | |
|----------|-------------------------|-------------------------|------------------|
| 77319180 | 34 54 72 61 6E 73 69 74 | 69 6F 6E 00 41 63 74 69 | 4Transition Acti |
| 77319190 | 76 61 74 65 4B 65 79 62 | 6F 61 72 64 4C 61 79 6F | vateKeyboardLayo |
| 773191A0 | 75 74 00 41 64 64 43 6C | 69 70 62 6F 61 72 64 46 | ut AddClipboardF |
| 773191B0 | 6F 72 6D 61 74 4C 69 73 | 74 65 6E 65 72 00 41 64 | ormatListener Ad |
| 773191C0 | 64 56 69 73 75 61 6C 49 | 64 65 6E 74 69 66 69 65 | dVisualIdentifie |
| 773191D0 | 72 00 41 64 6A 75 73 74 | 57 69 6E 64 6F 77 52 65 | r AdjustWindowRe |
| 773191E0 | 63 74 00 41 64 6A 75 73 | 74 57 69 6E 64 6F 77 52 | ct AdjustWindowR |
| 773191F0 | 65 63 74 45 78 00 41 64 | 6A 75 73 74 57 69 6E 64 | ectEx AdjustWind |

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)

| | | |
|----------|---|-------------|
| 773166B0 | B4 79 0A 00 60 89 0A 00 40 CE 06 00 90 E2 04 00 | ´y ˘% @î á |
| 773166C0 | 10 22 0B 00 48 7D 04 00 80 7C 04 00 50 81 04 00 | " H} € P |
| 773166D0 | 50 4C 04 00 70 FA 01 00 50 2D 09 00 50 FF 09 00 | PL pú P- Pý |
| 773166E0 | 00 95 09 00 10 95 09 00 F0 F8 03 00 C0 2D 09 00 | • • øø À- |
| 773166F0 | 80 DD 09 00 10 AE 09 00 F0 04 03 00 30 95 09 00 | €Ý € ð 0• |
| 77316700 | 60 81 04 00 10 1B 04 00 80 81 04 00 90 81 04 00 | ` € |
| 77316710 | 40 33 04 00 D0 DD 09 00 D0 DD 09 00 00 DE 09 00 | @3 ĐÝ ĐÝ Đ |
| 77316720 | 60 AE 09 00 90 AE 09 00 C0 E2 08 00 B0 81 04 00 | `€ € Àâ ° |
| 77316730 | C0 81 04 00 90 40 09 00 90 40 09 00 90 90 02 00 | À @ @ |
| 77316740 | F0 FC 02 00 60 D0 03 00 50 5C 02 00 D0 17 07 00 | őü `Đ P\ Đ |
| 77316750 | 50 95 09 00 90 7E 08 00 80 52 04 00 80 41 09 00 | P• ~ €R €A |
| 77316760 | B0 41 09 00 80 D6 09 00 B0 D7 09 00 D0 81 08 00 | °A €Ö °× Đ |
| 77316770 | 80 82 08 00 C0 22 03 00 20 E8 08 00 B0 E8 08 00 | €, À" è °è |
| 77316780 | E0 E7 08 00 20 E7 08 00 A0 E8 08 00 E0 81 04 00 | àç ç è à |
| 77316790 | F0 4B 04 00 60 52 09 00 60 52 04 00 80 52 09 00 | ők `R `R €R |

➤ AddressOfNameOrdinals (每项 16 位)

| | |
|----------|---|
| 77318960 | 03 00 04 00 05 00 06 00 07 00 08 00 09 00 0A 00 |
| 77318970 | 0B 00 0C 00 0D 00 0E 00 0F 00 10 00 11 00 12 00 |
| 77318980 | 13 00 14 00 15 00 16 00 17 00 18 00 19 00 1A 00 |
| 77318990 | 1B 00 1C 00 1D 00 1E 00 1F 00 20 00 21 00 22 00 |
| 773189A0 | 23 00 24 00 25 00 26 00 27 00 28 00 29 00 2A 00 |

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 0FFFFh

.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

Hello! I am Liu Jingting.



确定

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。

| | | |
|----------|-------------------------|-------------------------|
| 00000000 | 4D 5A 90 00 03 00 00 00 | 04 00 00 00 FF FF 00 00 |
| 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 D0 00 00 00 |
| 00000040 | 0E 1F BA 0E 00 B4 09 CD | 21 B8 01 4C CD 21 54 68 |

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。

| | | | |
|----------|-------------------------|-------------------------|-----------|
| 000000D0 | 50 45 00 00 4C 01 03 00 | BA 6D 5B 65 00 00 00 00 | PE L °m[e |
| 000000E0 | 00 00 00 00 E0 00 0F 01 | 0B 01 05 0C 00 12 00 00 | à |
| 000000F0 | 00 14 00 00 00 00 00 00 | 00 10 00 00 00 10 00 00 | |

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。

| | | | |
|----------|-------------------------|-------------------------|-------|
| 00000140 | 00 00 00 00 10 00 00 00 | 00 00 00 00 00 00 00 00 | |
| 00000150 | 74 30 00 00 3C 00 00 00 | 00 00 00 00 00 00 00 00 | t0 < |
| 00000160 | 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00 | |
| 00000170 | 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00 | |
| 00000180 | 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00 | |
| 00000190 | 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00 | |
| 000001A0 | 00 00 00 00 00 00 00 00 | 00 30 00 00 74 00 00 00 | 0 t |
| 000001B0 | 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00 | |
| 000001C0 | 00 00 00 00 00 00 00 00 | 2E 74 65 78 74 00 00 00 | .text |

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：

| Directory Information | | | | |
|-----------------------|----------|----------|-----|-----|
| | RVA | Size | | |
| ExportTable: | 00000000 | 00000000 | ... | L H |
| ImportTable: | 00003074 | 0000003C | ... | L H |
| Resource: | 00000000 | 00000000 | ... | L H |
| Exception: | 00000000 | 00000000 | | L H |
| Security: | 00000000 | 00000000 | | H |

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：

| | | | |
|----------|-------------------------|-------------------------|-------|
| 000001C0 | 00 00 00 00 00 00 00 00 | 2E 74 65 78 74 00 00 00 | .text |
| 000001D0 | D6 10 00 00 00 10 00 00 | 00 12 00 00 00 04 00 00 | ö |
| 000001E0 | 00 00 00 00 00 00 00 00 | 00 00 00 00 20 00 00 60 | |

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：

| | | | |
|----------|-------------------------|-------------------------|------------|
| 000001F0 | 2E 72 64 61 74 61 00 00 | 36 03 00 00 00 30 00 00 | .rdata 6 0 |
| 00000200 | 00 04 00 00 00 16 00 00 | 00 00 00 00 00 00 00 00 | |
| 00000210 | 00 00 00 00 40 00 00 40 | 2E 64 61 74 61 00 00 00 | @ @.data |

- 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 | @ A |

- 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：

| [Section Table] | | | | | |
|-------------------|----------|----------|----------|----------|----------|
| Name | VOffset | VSize | ROffset | RSize | Flags |
| .text | 00001000 | 000010D6 | 00000400 | 00001200 | 60000020 |
| .rdata | 00003000 | 00000336 | 00001600 | 00000400 | 40000040 |
| .data | 00004000 | 00000F15 | 00001A00 | 00000E00 | C0000040 |

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：

[File Location Calculator]

Addresses

VA:

00403074

RVA:

00003074

Offset:

00001674

Additional Information

Section:

.rdata

Bytes:

1C 31 00 00 00 00 00 00 00 00

DO

Hex Edit

(1) USER32.DLL

【加载前】

```
1. ImageImportDescriptor:
OriginalFirstThunk: 0x0000311C
TimeStamp: 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 | E2 p2 $1
00001710 | DE 32 00 00 FE 32 00 00 | 00 00 00 00 24 31 00 00 | E2 p2 $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:

| | | | |
|----------|-----------------|-------------------------------|-----------------------------|
| 7728A73E | CC | INT3 | |
| 7728A73F | 8BFF | MOV EDI,EDI | homework.<ModuleEntryPoint> |
| 7728A742 | 55 | PUSH EBP | |
| 7728A743 | 8BEC | MOV EBP,ESP | |
| 7728A745 | 833D B4EC2A77 0 | CMP DWORD PTR DS:[772AECB4],0 | |
| 7728A74C | 74 22 | JE SHORT USER32.7728A770 | |
| 7728A74E | 64:A1 18000000 | MOV EAX,DWORD PTR FS:[18] | |

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 0C 00 00 19 01 | A USER32.dll |

(2) KERNEL32.DLL

【加载前】

```

2. ImageImportDescriptor:
OriginalFirstThunk: 0x000030B0
TimeStamp: 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：

| | | |
|----------|---|-------------|
| 00001600 | 16 32 00 00 1C 33 00 00 3E 31 00 00 4C 31 00 00 | 2 3 >1 L1 |
| 00001610 | 5A 31 00 00 68 31 00 00 7A 31 00 00 94 31 00 00 | Z1 h1 z1 "1 |
| 00001620 | A6 31 00 00 B8 31 00 00 D6 31 00 00 E6 31 00 00 | 1 ,1 Ō1 æ1 |
| 00001630 | F6 31 00 00 06 32 00 00 22 32 00 00 36 32 00 00 | ö1 2 "2 62 |
| 00001640 | 46 32 00 00 5A 32 00 00 66 32 00 00 82 32 00 00 | F2 z2 f2 ,2 |
| 00001650 | 94 32 00 00 AE 32 00 00 B6 32 00 00 CE 32 00 00 | "2 2 2 î2 |
| 00001660 | DE 32 00 00 FE 32 00 00 00 00 00 00 24 31 00 00 | Ê2 þ2 \$1 |

再看一下这时 FirstThunk 的值：

| | | |
|----------|---|-------------|
| 00001600 | 16 32 00 00 1C 33 00 00 3E 31 00 00 4C 31 00 00 | 2 3 >1 L1 |
| 00001610 | 5A 31 00 00 68 31 00 00 7A 31 00 00 94 31 00 00 | Z1 h1 z1 "1 |
| 00001620 | A6 31 00 00 B8 31 00 00 D6 31 00 00 E6 31 00 00 | 1 ,1 Ō1 æ1 |
| 00001630 | F6 31 00 00 06 32 00 00 22 32 00 00 36 32 00 00 | ö1 2 "2 62 |
| 00001640 | 46 32 00 00 5A 32 00 00 66 32 00 00 82 32 00 00 | F2 z2 f2 ,2 |
| 00001650 | 94 32 00 00 AE 32 00 00 B6 32 00 00 CE 32 00 00 | "2 2 2 î2 |
| 00001660 | DE 32 00 00 FE 32 00 00 00 00 00 00 24 31 00 00 | Ê2 þ2 \$1 |
| 00001670 | 00 00 00 00 1C 31 00 00 00 00 00 00 00 00 00 00 | 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】

| | | |
|----------|---|-------------------|
| 00403000 | 20 6A A3 76 D0 3E A4 76 C0 88 A4 76 E0 37 A4 76 | jfvD>nvÀ^nvà7nv |
| 00403010 | 30 3A A4 76 70 FA A3 76 30 43 A4 76 40 9F A3 76 | 0:avpúfv0Cav@Yfv |
| 00403020 | 10 42 A4 76 80 43 A4 76 B0 4A A3 76 10 85 A3 76 | Bav€Cav°Jfv ...fv |
| 00403030 | F0 98 A3 76 B0 98 A3 76 40 42 A4 76 60 42 A4 76 | ô~fv°~fv@Bav`Bav |
| 00403040 | 70 42 A4 76 E0 3D A4 76 70 44 A4 76 C0 42 A4 76 | pBavà=avpDnvÀBav |
| 00403050 | B0 44 A4 76 90 8A A3 76 F0 95 A3 76 D0 42 A4 76 | °Dnv Šfvô•fvDBnv |
| 00403060 | 30 45 A4 76 20 45 A4 76 00 00 00 00 40 A7 28 77 | 0Eav Eav @S(w |

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：

| | | | |
|----------|-------------|------|----------|
| 76A488C0 | 55 | push | ebp |
| 76A488C1 | 8BEC | mov | ebp, esp |
| 76A488C3 | 6A FF | push | -1 |
| 76A488C5 | 68 B0F3E877 | push | 77E8F3B0 |

回到 WinHex，在 40313Eh 处找到函数名，如下图所示：

| | | |
|----------|---|------------------|
| 00403130 | 41 00 55 53 45 52 33 32 2E 64 6C 6C 00 00 19 01 | A USER32.dll |
| 00403140 | 45 78 69 74 50 72 6F 63 65 73 73 00 52 00 43 6C | ExitProcess R C1 |