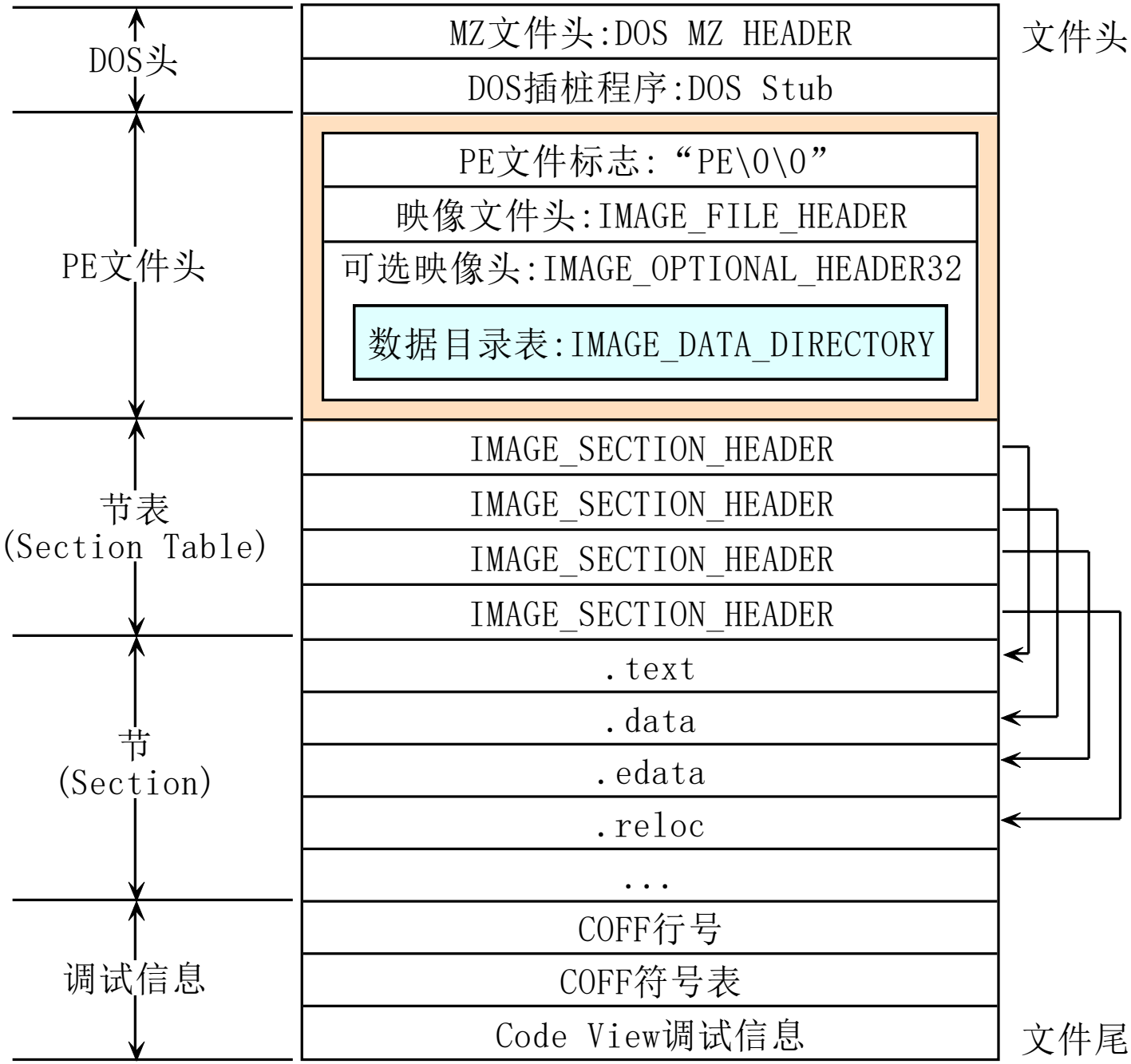
实验二

病毒尾区段寄生实验

病毒寄生的问题

- 如果将病毒直接粘贴的在PE文件后面，用x64dbg加载后，在内存中没有看到粘结上的部分。
- 可见PE文件加载时，直接粘贴的代码并未获得PE加载器的认可。
- 合理的猜想是，PE文件加载器应该根据某些字段来将文件内容加载到内存
- 所以，一定有什么字段描述了有效的代码部分有多长，哪些代码应该加载，而我们只是粘贴了代码，却没有修改相关字段，所以没有被PE文件加载器加载到内存，寄生失败

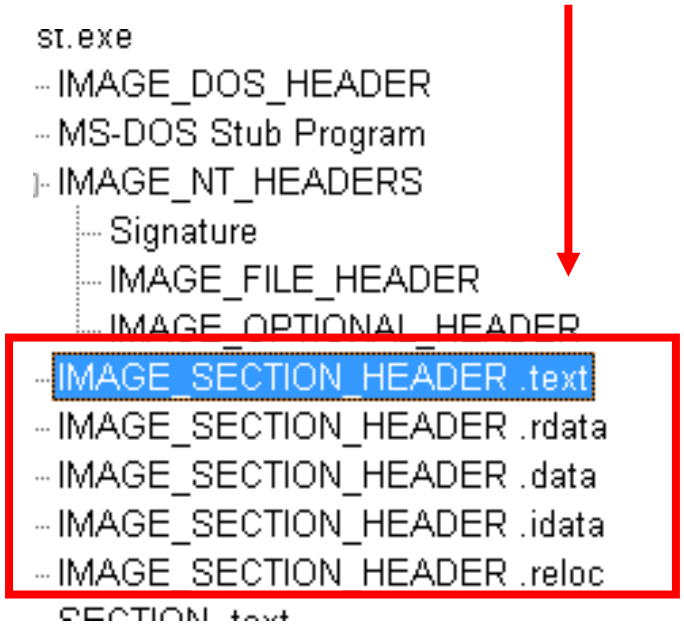

DOS头
PE文件头
节表
(Section Table)
节
(Section)
调试信息
MZ文件头:DOS MZ HEADER
DOS插桩程序:DOS Stub
PE文件标志:“PE\0\0”
映像文件头:IMAGE_FILE_HEADER
可选映像头:IMAGE_OPTIONAL_HEADER32
数据目录表:IMAGE_DATA_DIRECTORY
IMAGE_SECTION_HEADER
IMAGE_SECTION_HEADER
IMAGE_SECTION_HEADER
IMAGE_SECTION_HEADER
.text
.data
.edata
.reloc
...
COFF行号
COFF符号表
Code View调试信息
文件头
文件尾

PE描述文件大小的字段

```
typedef struct _IMAGE_SECTION_HEADER {
    BYTE    Name[IMAGE_SIZEOF_SHORT_NAME];
    union {
            DWORD   PhysicalAddress;
            DWORD   VirtualSize;
    } Misc;
    DWORD   VirtualAddress;
    DWORD   SizeOfRawData;
    DWORD   PointerToRawData;
    DWORD   PointerToRelocations;
    DWORD   PointerToLinenumbers;
    WORD    NumberOfRelocations;
    WORD    NumberOfLinenumbers;
    DWORD   Characteristics;
} IMAGE_SECTION_HEADER, *PIMAGE_SECTION_HEADER;
```

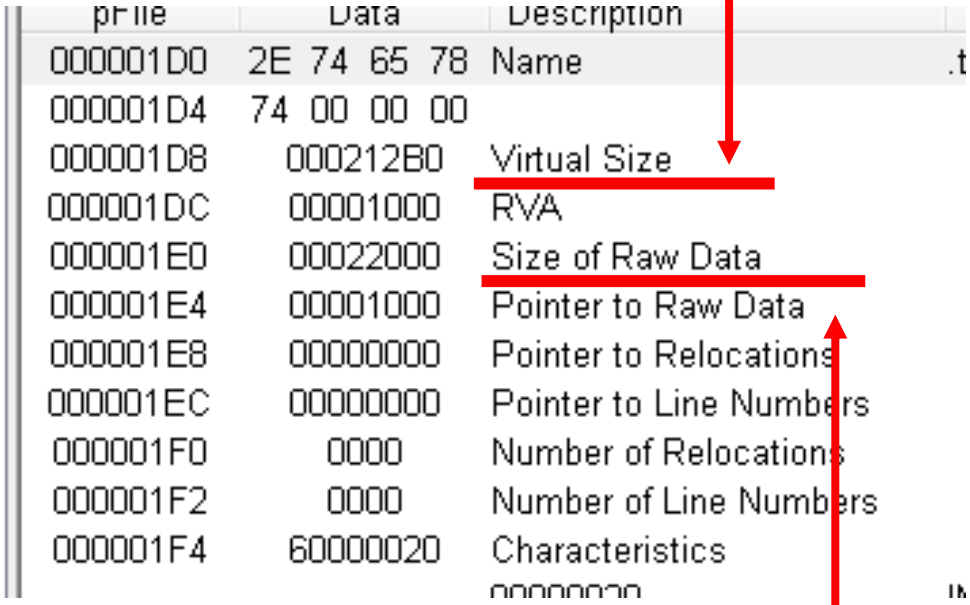
- 1）区段表中每个区段头都对应一个节，其中描述了对应节的文件大小（SizeOfRawData）和加载到内存的大小（VirtualSize）
- 两者可能不同，文件大小小于内存大小时，将在内存中补0

每一个节都有一个对应的结构ImageSectionHeader



VirtualSize表明本段加载到内存后的大小

即加载到内存的实际字节数（未对齐）

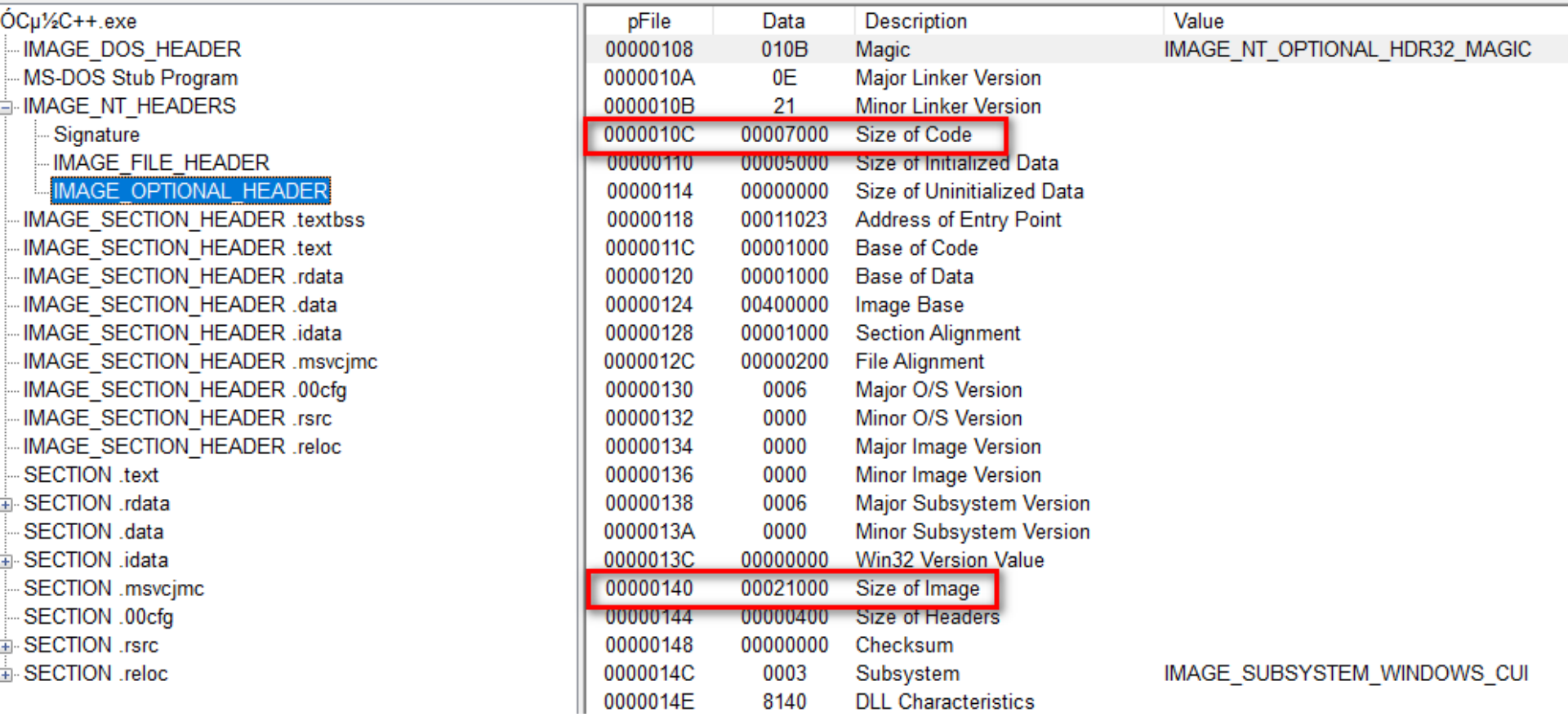


SizeOfRawData字段表明本节在文件中的大小（对齐后）

因此，必须是FileAlignment（在可选映像头中）的整数倍

PE描述文件大小的字段

- 在PE可选头中，SizeOfImage给出了整个程序包括所有头部加载到内存后的大小。其值应该是SectionAlignment的整数倍

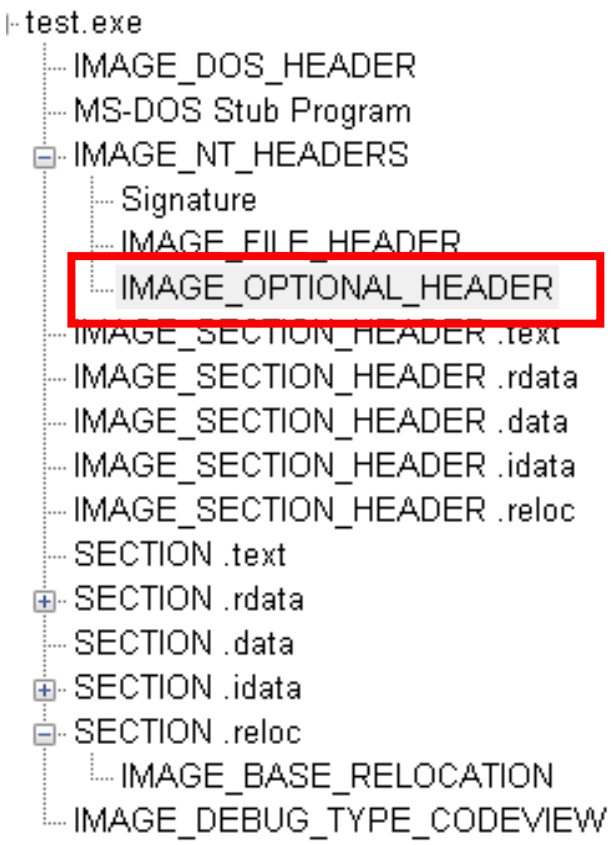


回到可选映像头ImageOptionalHeader中观察：

SizeOfImage给出了整个文件在内存中对齐后的大小

SectionAlignment是内存对齐的粒度

FileAlignment是文件对齐的粒度



pFile	Data	Description
000000F0	010B	Magic
000000F2	06	Major Linker Version
000000F3	00	Minor Linker Version
000000F4	00022000	Size of Code
000000F8	0000A000	Size of Initialized Data
000000FC	00000000	Size of Uninitialized Data
00000100	00001690	Address of Entry Point
00000104	00001000	Base of Code
00000108	00001000	Base of Data
0000010C	00300000	Image Base
00000110	00001000	Section Alignment
00000114	00001000	File Alignment
00000118	0004	Major O/S Version
0000011A	0000	Minor O/S Version
0000011C	0000	Major Image Version
0000011E	0000	Minor Image Version
00000120	0004	Major Subsystem Version
00000122	0000	Minor Subsystem Version
00000124	00000000	Win32 Version Value
00000128	0002D000	Size of Image

小结

- 1）区段表中每个区段头都对应一个节，其中描述了对应节的文件大小（SizeOfRawData）和加载到内存的大小（VirtualSize）
- 2）在PE可选头中，SizeOfImage给出了整个程序包括所有头部加载到内存后的大小。其值应该是SectionAlignment的整数倍
- **PE文件总大小和每个节的大小都有参数**

病毒寄生在PE文件尾区段

本实验在PE文件最后一个区段末尾添加指令，并检验其是否被加载到内存。

本实验的意义：为病毒寄生摸索关键技术点。

实验步骤：

- 1. 如果该区段具有足够空洞，我们就在文件中将指令加到该节的空洞中，然后修改节表中的VirtualSize字段（未对齐的内存大小）为修改后的大小。而对齐后的文件大小SizeOfRawData保持不变。
- 2. 如果该节空洞不够，我们还需要在末尾增加一个新节，这就修改节的VirtualSize, SizeOfRawData字段，以及整个文件的SizeOfImage字段

实验

寄生在PE文件最后节的空洞

寄生后文件长度不变

1. 观察最后一个节是否具有空洞

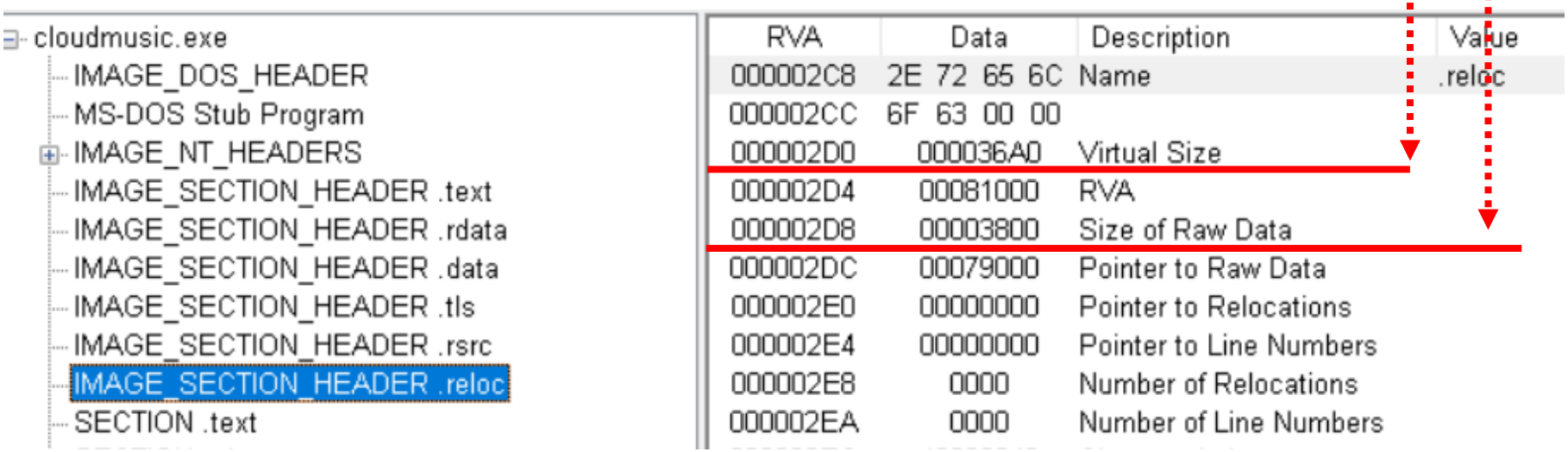
打开一个PE文件，判断其最后一个节是否具有空洞文件大小

最后一个节.reloc

对齐大小SizeOfRawData = 3800

实际大小VirtualSize = 36A0

VirtualSize < SizeOfRawData，说明reloc节具有空洞

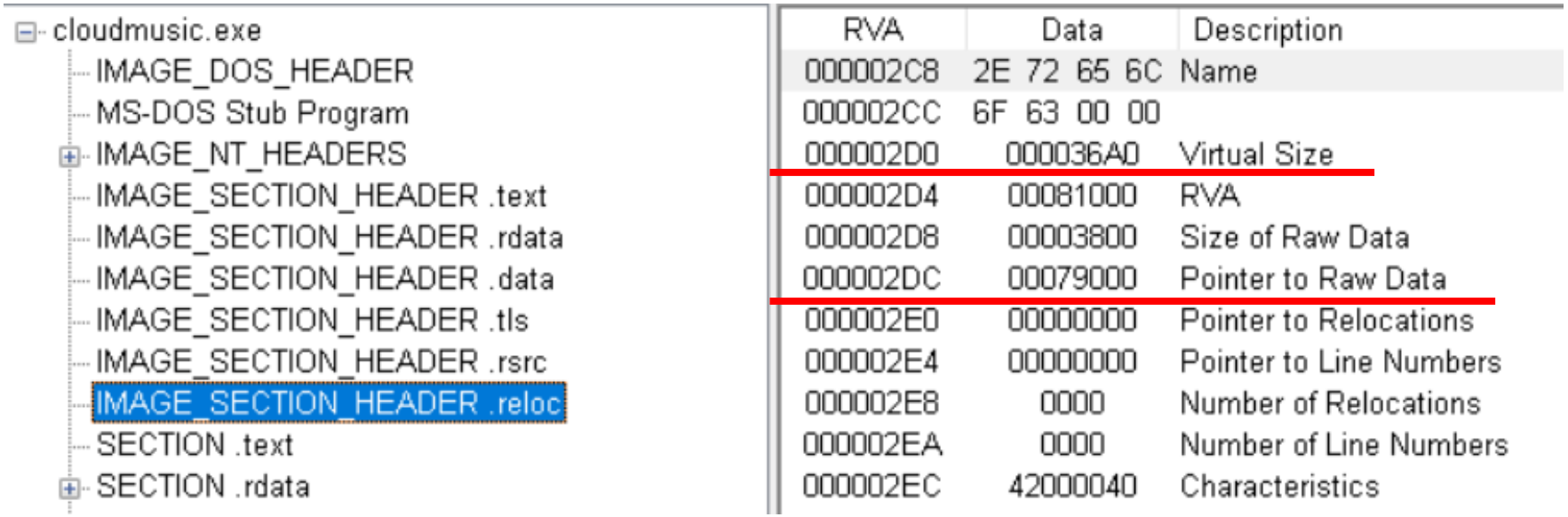


2. 在文件中定位寄生位置

reloc节的空洞大小是3800-36A0=160，现在我们准备在空洞插入4字节的机器码 eb 02 90 90，4字节，先来定位文件中的寄生位置：

寄生位置的RVA是36A0，将其转化为文件位置为：

79000 (PointerToRawData) + 36A0 (VirtualSize) = 7C6A0



3. 在文件中填入寄生代码

用UE打开文件，Ctrl+G定位7C6A0，填入机器码

cloudmusic.exe

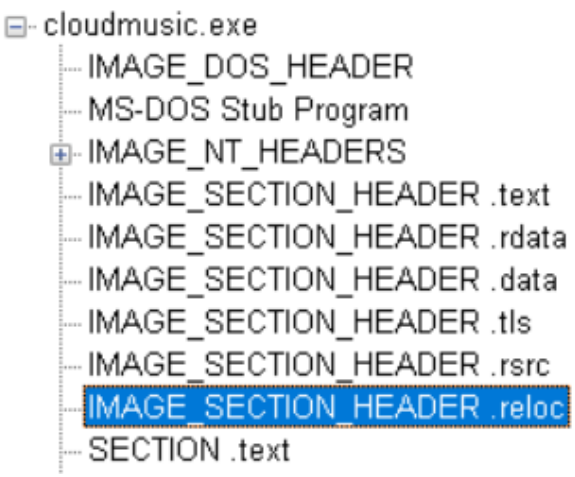
```
0007c690h: 20 3B 44 3B 48 3B 4C 3B 50 3B 54 3B 58 3B 00 00
0007c6a0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0007c6b0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0007c6c0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0007c6d0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

cloudmusic.exe

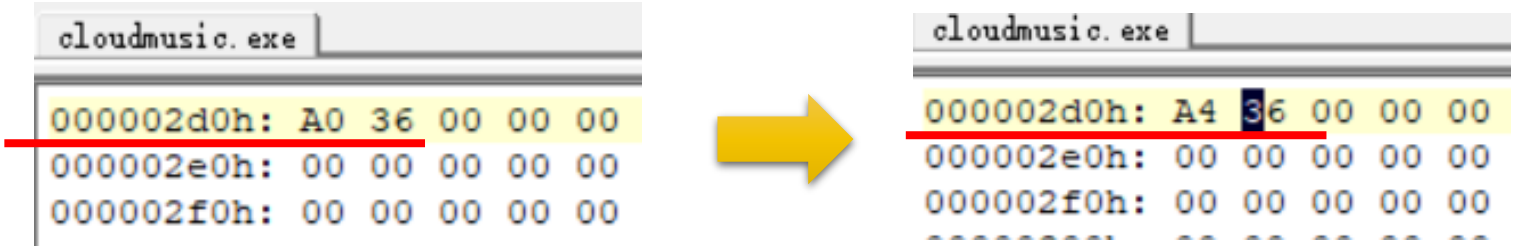
```
0007c690h: 20 3B 44 3B 48 3B 4C 3B 50 3B 54 3B 58 3B 00 00
0007c6a0h: eb 02 90 90 00 00 00 00 00 00 00 00 00 00 00 00
0007c6b0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0007c6c0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0007c6d0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

4. 修改PE文件的相关字段

- 有哪些字段需要修改呢？首先，最后一个节的实际大小（VirtualSize）肯定需要修改，要由原值36A0修改为36A4
- 用pFile查看VirtualSize字段在文件偏移2D0处，用UE进行修改



pFile	Data	Description
000002C8	2E 72 65 6C	Name
000002CC	6F 63 00 00	
000002D0	000036A0	Virtual Size
000002D4	00081000	RVA
000002D8	00003800	Size of Raw Data
000002DC	00079000	Pointer to Raw Data
000002E0	00000000	Pointer to Relocations
000002E4	00000000	Pointer to Line Numbers
000002E8	0000	Number of Relocations
000002EA	0000	Number of Line Numbers

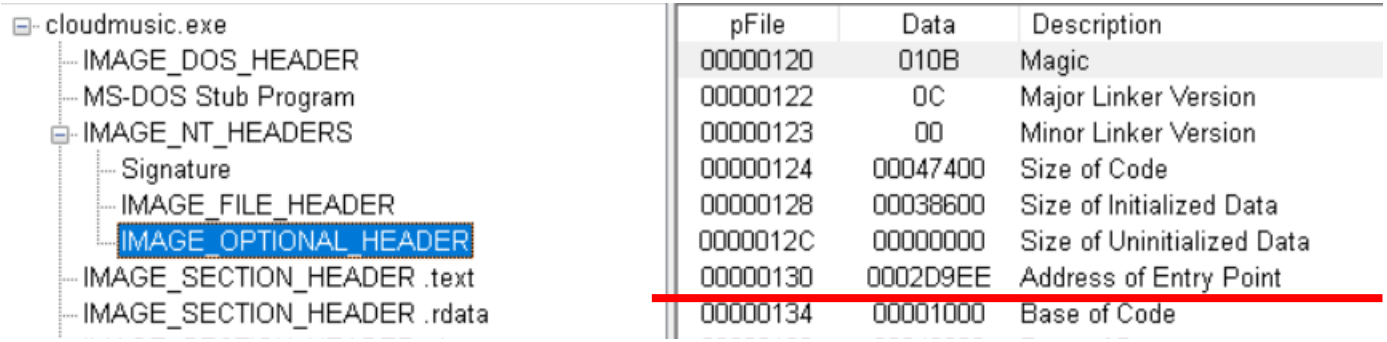


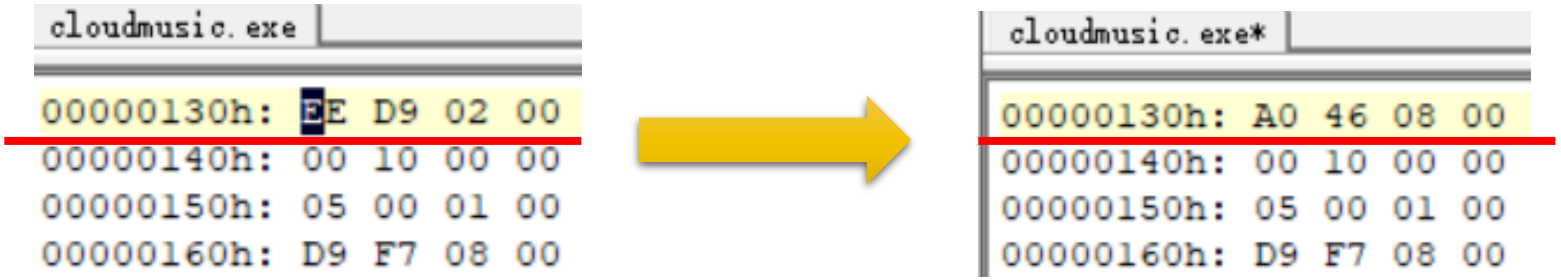
由于没有增加新节，所以我们先不用修改SizeOfRawData以及SizeOfImage字段

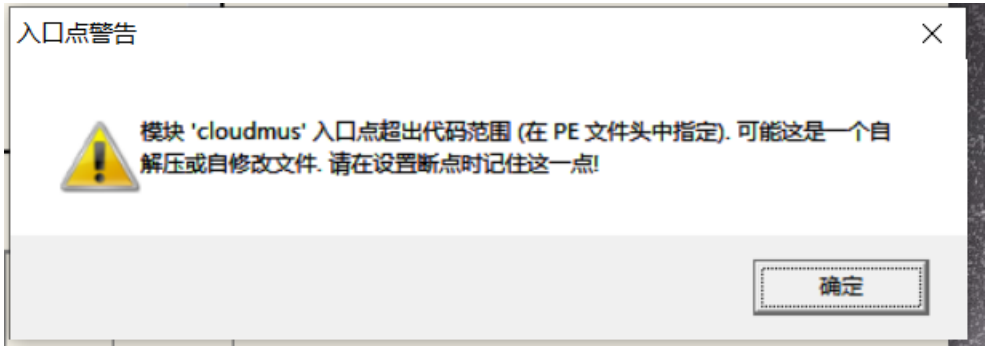
5. 修改入口点RVA
验证寄生代码被加载到内存

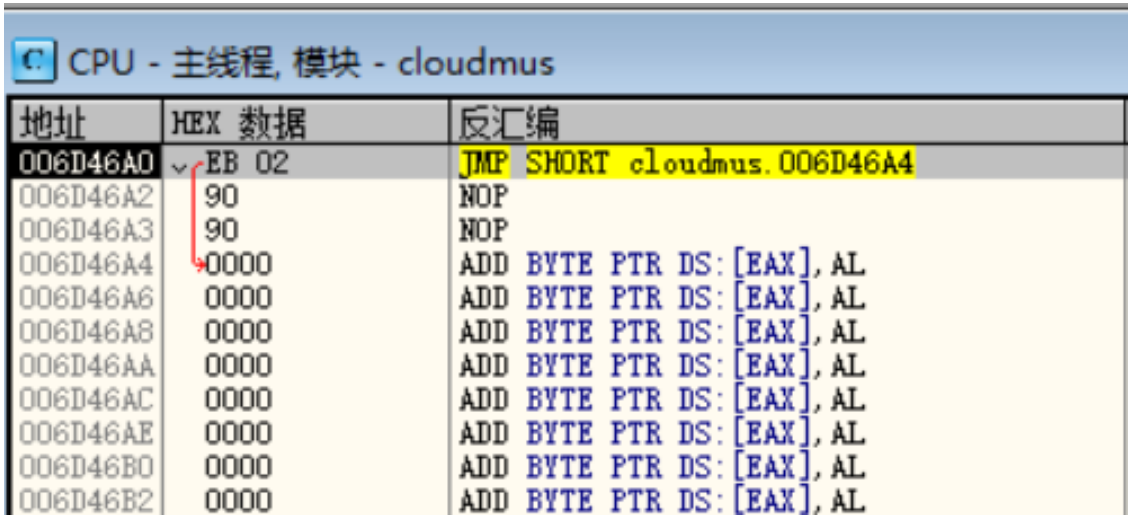
- 在可选头中找到入口点RVA字段的文件偏移
- 用UE把其值改为寄生代码的RVA

  = 81000（reloc节起始RVA） + 36A0 = 846A0









实验要求：选择一个exe文件进行以上的寄生实验
建议选用：C:\Windows\System32\notepad.exe
FTP文件目录下有