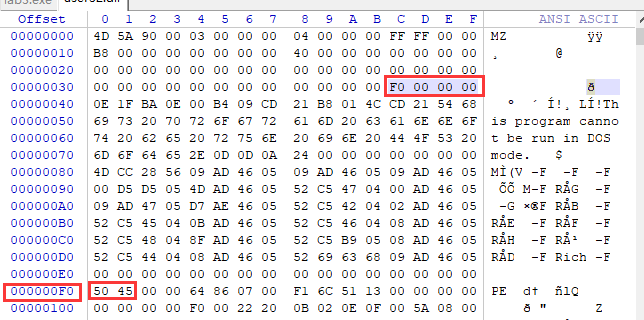
PE文件结构解析作业

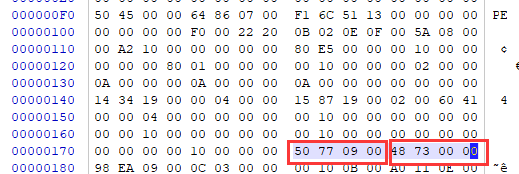
1. 请解析USER32.DLL前5个导出函数的信息，要求列举AddressOfNams、AddressOfOrdinals、AddressOfFunctions的详细数据。

用WinHex工具打开user32.dll文件,找到PE头



找到数据目录项 进而找到导出表信息

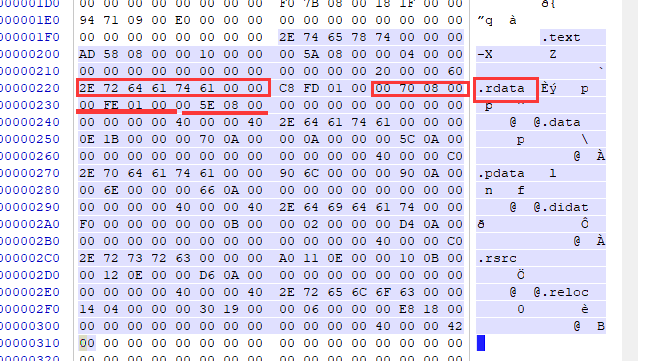
* 导出表的RVA为97750h
* 导出表的大小=7348h



分析节表信息,在7个节表中找到.rdata

* Name = 61746164722Eh ;节区的名字为.rdata
* VirtualSize = 0001FD8Ch ;节区的尺寸
* VirtualAddress = 00087000h;节区的RVA地址
* SizeOfRawData = 0001FE00h ;在文件中对齐后的尺寸
* PointerToRawData = 00085E00h ;在文件中的偏移

根据RVA与FOA的换算关系,所以导出表在文件中的偏移地址为 97750h - 87000h + 85e00h = 96550h



分析导出目录 IMAGE\_EXPORT\_DIRECTORY

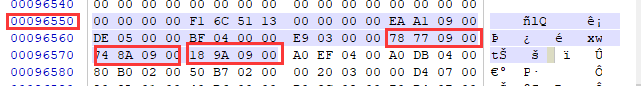
* AddressOfFunctions = 00097778h ;导出函数地址表RVA
* AddressOfNames = 00098A74h ;函数名称地址表RVA
* AddressOfOrdinals = 00099A18h;函数序号地址表

根据RVA与FOA的换算关系,所以:

(1)导出函数地址表在文件中的偏移地址为 97778h - 87000h + 85e00h = 96578h,

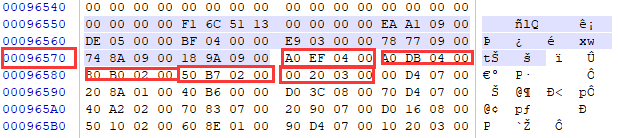
(2)函数名称地址表在文件中的偏移地址为98A74h- 87000h + 85e00h = 97874h,

(3)函数序号地址表在文件中的偏移地址为99A18h- 87000h + 85e00h = 98818h



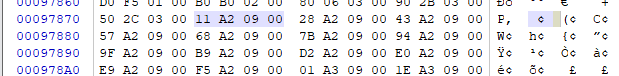
1. 从96578h前五个导出函数的地址:

红框圈出的就是前五个导出函数的RVA

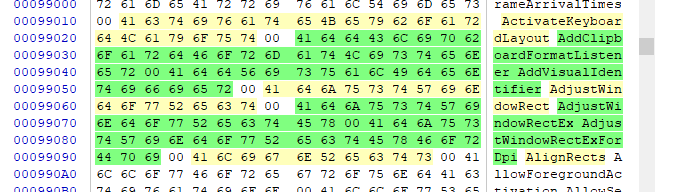


1. 在97874h前五个导出函数名称的RVA

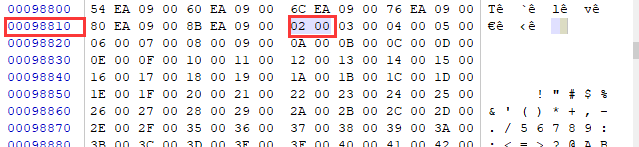
第一个函数名字偏移地址为9A211h- 87000h + 85e00h = 99011h



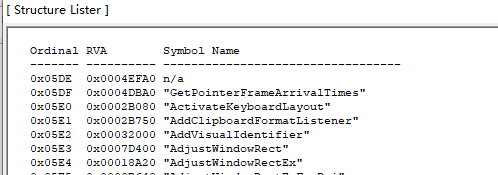
所以从99011h处开始依次找到前五个函数的名字



1. 在98818h处查看函数序号



或者使用lordPE查看前五个导出函数的信息:



1. 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。

代码如下:

.386

.model flat,stdcall

.stack

include irvine32.inc

includelib irvine32.lib

includelib user32.lib

includelib kernel32.lib

includelib masm32.lib

.data

array512 word 512 dup(?)

lpText byte "I am text",0

lpCaption byte "I am caption",0

minWord word 0

.code

main proc

mov esi,offset array512

mov ecx,lengthof array512

mov bx,0ffffh

L1:

mov eax,10000h

;invoke Random,10000h

call RandomRange

mov [esi],ax

add esi,type array512

cmp bx,ax ;如果bx中的值大，交换

jae xc

;jmp L1

loop L1

mov minWord,bx

jmp L3

xc:

xchg ax,bx ;交换

loop L1

L3:

invoke MessageBox,NULL,addr lpText,addr lpCaption,MB\_OK

invoke ExitProcess,0

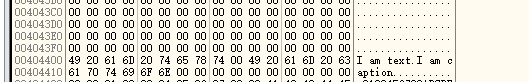
main endp

end main

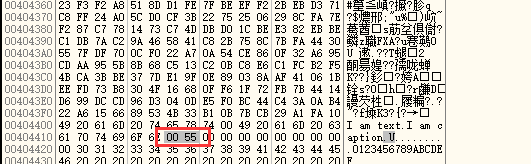
在写的程序中,将512个生成的随机数依次写入内存,然后将bx中的最小值,赋值给定义的Word类型的minWord,用OD查看,最小值为0055h,地址在定义的字符串之后.

在内存中404000h-404400h被写入了1024字节的数据

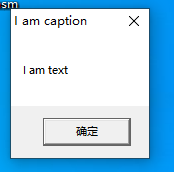
执行前该段区域为未初始化的区间:



程序执行之后:



调用了messagebox函数:

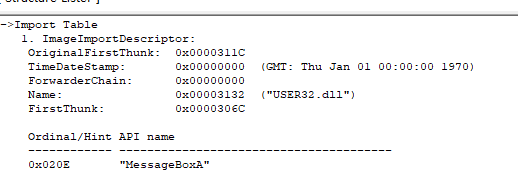


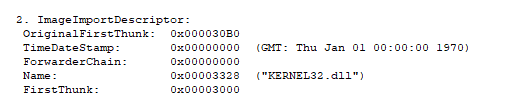
赋值完成之后,调用Messagebox函数,上图所示和预期结果一样,所以随机数赋值未赋值给非法的地址空间.

1. **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。**

导入函数信息:

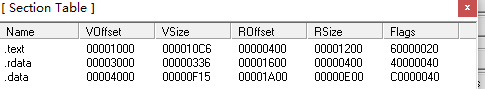
共有两个导入函数,分别位于user32.dll和kernel32.dll中



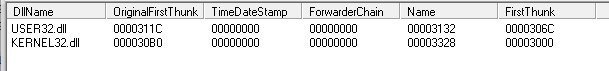


具体分析调用messagebox的过程:

**程序加载前:**

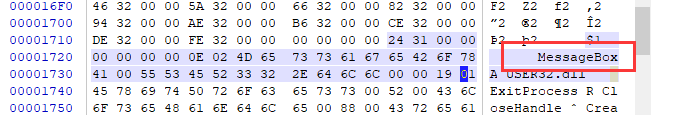


节表



导入表

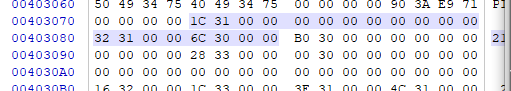
首先根据OriginalFirstThunk的值311Ch(该值是一个RVA)，计算出FOA为171Ch,然后去文件中的171Ch处，从171Ch处开始取双字，取到的依次是hint name,然后加载器会根据函数的Hint/Name从内存地址空间中查找到函数的VA,然后将找到的函数地址覆盖掉内存中的地址，当程序被加载到内存之后，jmp指令跳转到的地址的值就是调用的动态链接库的函数的代码段的地址，为上面内存中的[0040306C] = 71E93A90h。程序跳转到user32.dll的代码区继续执行。



程序加载后:

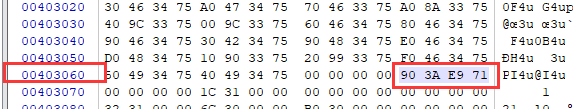
导入表数据所在RVA=3074h

导入表数据大小=3ch(分为3组,前两组代表两个动态链接库,最后一组为全0结构,表示导入表描述符已经结束)，本程度调用的动态链接库有user32.dll和kernel32.dll。



选中部分为调用messagebox的user32.dll有关的导入表数据。

最后一个双字为FirstThunk=306Ch



在内存40306Ch处的值为71E93A90h,该值为messagebox函数在内存动态链接库中的地址