编号：_____

实验	一	二	三	四	五	六	七	八	总评	教师签名
成绩										

武汉大学国家网络安全学院

课程实验(设计)报告

课程名称 ：软件安全实验

实验内容 ：PE 病毒分析与清除

专业(班) ：_____

学　　号 ：_____

姓　　名 ：_____

任课教师 ：_____

2020 年　10 月 22 日

目 录

完成报告后请更新目录。

实验 3　PE 病毒分析与清除

3.1 实验名称

PE 病毒分析与清除

3.2 实验目的

1) 了解 PE 病毒的基本原理
2) 熟悉 PE 病毒中的部分关键技术
3) 学会清除 PE 病毒

3.3 实验步骤及内容

第一阶段：

- 熟悉 Masm32
  1) 安装 masm32v11
  2) 熟悉 masm32 的基本环境
  3) 写一个最简单的 HelloWorld 程序，并编译成功
  4) 对得到的可执行文件进行反汇编，比较其反汇编代码和最初的汇编代码有哪些异同？
  5) 查看并理解 masm32\bin 下各个批处理程序，了解它们的大致功能

第二阶段：

- 熟悉病毒重定位的基本思路和方法
  - 在 HelloWorld.exe 中添加一段代码，该段代码满足以下几个条件：
    - 该段代码弹出一个对话框（标题：武大信安病毒重定位，内容：姓名+学号后四位）
    - 该段代码同时包括代码和字符串数据。
    - 该段代码可以插入到.text 节的任意指令之间，而不需要修改该段代码中的任何字节。

第三阶段：

- 搜索 API 函数地址
  - 用 ollydbg 打开 HelloWorld.exe，获取 kernel32.dll 模块基地址，定位到 kernel32.dll 模块。
  - 从内存中的 kernel32.dll 模块获取函数 LoadLibraryA 和 GetProcAddress 的函数地址，并实际检验获得的地址是否正确

第四阶段：PE 病毒感染分析与清除

- 编译教材中的感染例子程序-bookexample-old.rar

- 使用该感染例子对 HelloWorld.exe 进行感染。思考以下问题：
  - 该病毒在感染文件时具体做了哪些操作？
  - 该病毒如何返回 HOST？
- 找出该病毒程序存在的一些问题，并解决这些问题。
- 编译教材中的感染例子程序-bookexample-new.rar
- 使用该感染例子对计算器 calc.exe 进行感染。
- 手工恢复被感染的 calc.exe（功能恢复）。

课后习题思考：

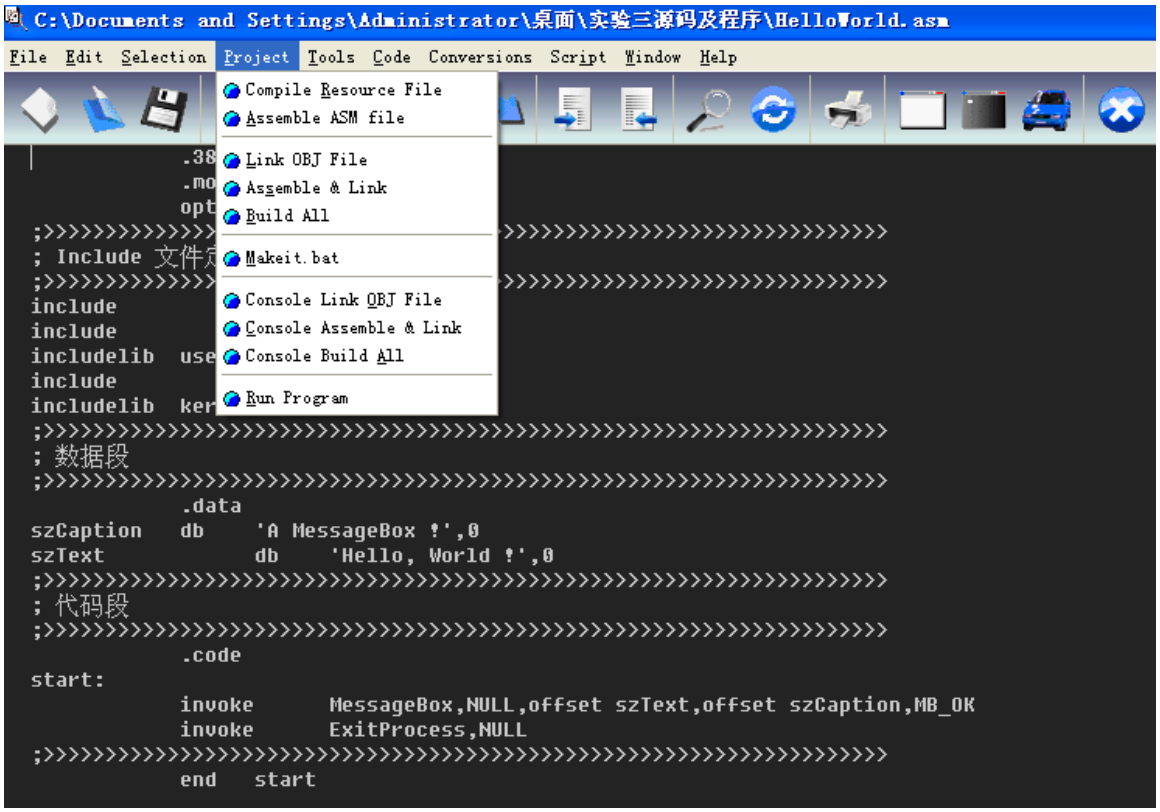
- 对 Win10 下的在 32 位及 64 位程序来说，如何通过 PEB 获取 kernel32 基地址？32 位和 64 程序有何差异？
- 尝试编写一个程序，可用来搜索指定目录下的所有 exe 文件，用 MessageBox 显示每一个被搜索到的 exe 文件名。
- 编写课本中病毒感染程序的病毒清除程序，其可以用来恢复被感染的任何文件
- 课程提供的病毒感染例子程序在 64 位系统中无法正常感染，请定位其原因，并给出解决方案。

3.4 实验关键过程、数据及其分析

熟悉 Masm32

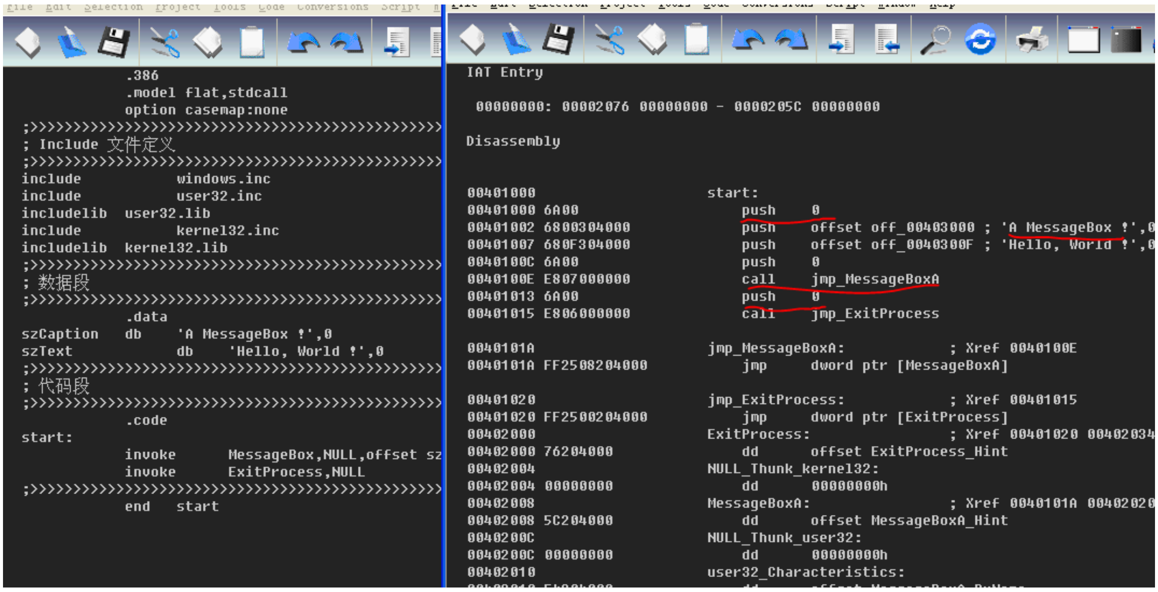
首先，根据安装程序引导，选择安装路径，执行完成后对应目录下出现 maxm32 文件夹。右键我的电脑-属性，在高级中选择环境变量，在 lib 中添加 masm32 的 lib 库文件添加进去，再加入 include 目录，最后在 Path 下添加 masm32/bin 目录

编写一个简单的 HelloWolrd.asm 汇编程序，用 masm 编辑器打开，选择 project-assemble&link，编译生成.obj 和.exe 文件



打开运行，可以看到成功进行了弹框

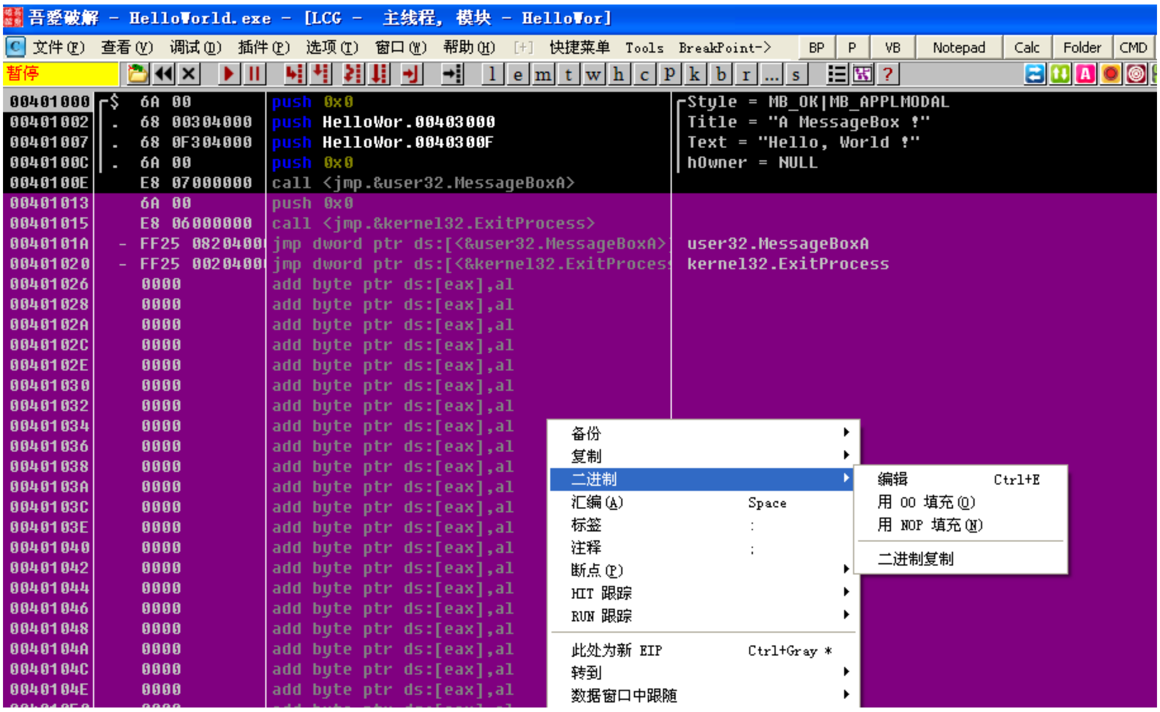
使用 masm32 自带的反汇编工具对可执行文件进行反汇编，将生成的反汇编代码与源代码进行比对。



可以看到，反汇编代码首先是 push 了 4 个值，再跳转到 MessageBox 的代码块，跳到具体函数部分，最后再 push 0,结束程序。而我们原本写的汇编代码并没有这些 push 和 jmp 跳转操作。

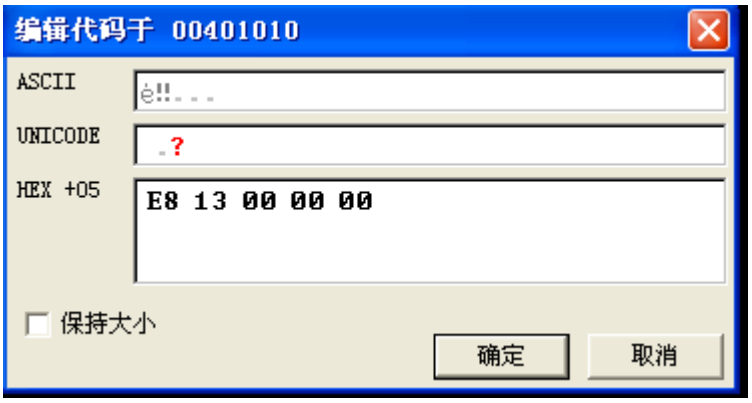
熟悉病毒重定位的基本思路和方法

使用 OllyDbg，打开之前生成的 HelloWorld.exe 文件，由于要插入一段代码，先将退出代码和一段剩余空间用空指令填充掉，选择二进制->用 NOP 填充



添加一条 push 0 指令，由于标题是“武大信安病毒重定位”9 个汉字 18 字节，字符串最后需要 00 作为结束符，因此占据 19=13H 字节。

在下一条指令处使用 ctrl+e 进行编辑，在此处进行汇编，call 指令是 E8，字节偏移要反写为 13 00 00 00（读成 00 00 00 13）



再在下个地址处填入字符串处填入字符串，注意要在末尾添上字节 00，作为结束符号

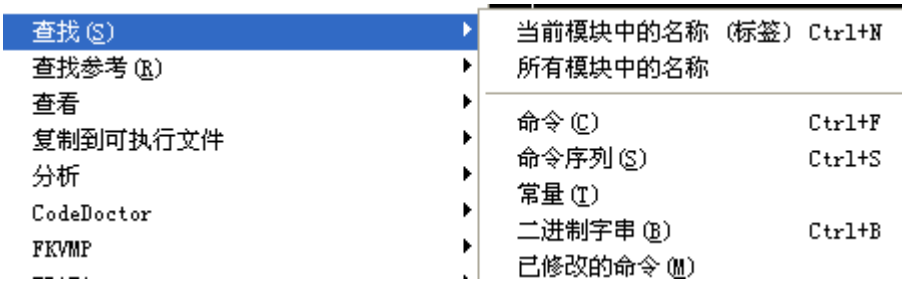
由于此时 OD 仍然将填入的字符串作为汇编代码解析，阅读不方便，使用数据窗口跟随，查看数据。

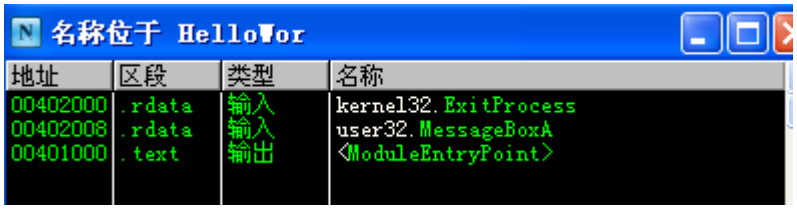
可以在数据窗口看到成功的填入的标题字符串

由于自己的内容是“沈思源 0001”，6+4 个字节，而末尾还有一个 00 字节，一共是 11=0B 个字节，因此再写入一条 call 指令如下：

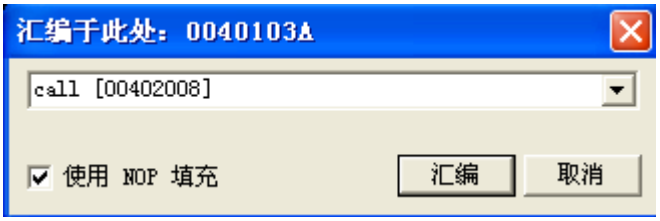
OD 中显示 call 指令是跳转到 00401038,CRTL+G 进行跳转，添加最后一个参数的命令 push 0

由于调用是要用绝对地址，右键查找->当前模块名称标签





再在最后使用绝对地址键入 call 指令，注意绝对地址要打上中括号



最后添加退出指令，复制可执行文件，保存所有修改，右键复制所有修改，右键保存文件

单击执行，可以见到正确的弹窗

再测试将代码复制到其他指令位置

```
00401060    6A 00           push 0x0
00401062    E8 13000000     call HelloWor.0040107A
00401067    CE              into
00401068    e4 b4           in al,0xb4
0040106A    f3:d0c5         rep rol ch,1
0040106D    B0 B2           mov al,0xB2
0040106F    B2 A1           mov dl,0xA1
00401071    B6 BE           mov dh,0xBE
00401073    D6              salc
00401074    D8B6 A8CEBB0(   fdiv dword ptr ds:[esi+0xBBCEA8]
0040107A    E8 0B000000     call HelloWor.0040108A
0040107F    C9              leave
00401080    f2:cb           repne retf
00401082    BC D4B43030     mov esp,0x3030B4D4
00401087    3031            xor byte ptr ds:[ecx],dh
00401089    006A 00         add byte ptr ds:[edx],ch
0040108C    FF15 0820400(   call dword ptr ds:[<&user32.MessageBoxA>  user32.MessageBoxA
00401092    6A 00           push 0x0
00401094    FF15 0020400(   call dword ptr ds:[<&kernel32.ExitProces  kernel32.ExitProcess
```

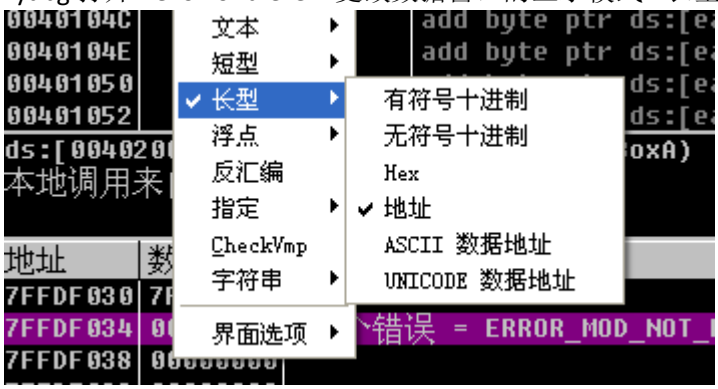
删除第一个弹窗的退出代码，保存第二个 exe 文件，执行，文件可以正确的连续弹出两个窗口

搜索 API 函数地址

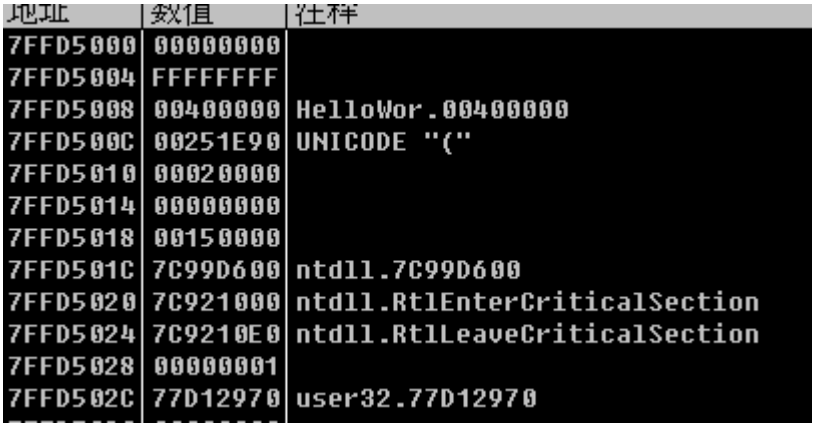
NT 内核系统中 fs 寄存器指向 TEB 结构，TEB+0x30 处指向 PEB 结构，PEB+0x0c 处指向 PEB_LDR_DATA 结构。

PEB_LDR_DATA+0x1c 处是一个叫 InInitialzationOrderModuleList 的成员，指向 LDR_MODULE 双向链表结构，存放一些动态链接库地址:第一个指向 ntdl.dll，第二个就是 kernel32.dll

使用 Ollydbg 打开 helloworld.exe。更改数据窗口的显示模式->长型->地址



找到 FS 地址，对其 30 的偏移量，跳转转到 7FFDF030,在数据窗口中跟随



现在有一个 0C 的偏移量，再进行数据窗口跟随

接下来有一个 1C 的偏移量，再次进行数据窗口跟随，找到了双向链表的结构，第一个值是指向下一个指针，第二个值是指向上一个指针。

```
00251F28  00251FD0
00251F2C  00251EAC
00251F30  7C920000  ntdll.7C920000
00251F34  7C932C28  ntdll.<ModuleEntryPoint>
00251F38  00093000
00251F3C  0208003A
00251F40  7C99D028  UNICODE "C:\WINDOWS\system32\ntdll.dll"
00251F44  00140012
00251F48  7C942158  UNICODE "ntdll.dll"
00251F4C  80084004
00251F50  0000FFFF
00251F54  7C99B2C8  ASCII 54,"■%"
```

整个双向链表的第一个值是 ntdll 模块的地址，图中是 7C920000

根据双向链表的结构，跟随第一个指针，找到可链表的下一个节点

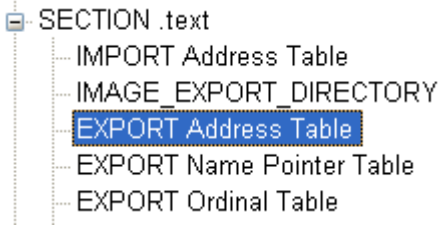
```
00251FD0  00252110
00251FD4  00251F28
00251FD8  7C800000  kernel32.7C800000
00251FDC  7C80B63E  kernel32.<ModuleEntryPoint>
00251FE0  0011E000
00251FE4  00420040
00251FE8  00251F70  UNICODE "C:\WINDOWS\system32\kernel32.dll"
00251FEC  001A0018
00251FF0  00251F98  UNICODE "kernel32.dll"
00251FF4  80084004
00251FF8  0000FFFF
00251FFC  7C99B2B0  ntdll.7C99B2B0
```

可见找到了 kernel32dll 的基地址 7C800000

使用 Ollydbg 的查找命令？LoadLibraryA 查找函数地址

？GetProcAddress

使用 PEView 查看 kernel32.dll,进入 SECTION.text EAT 表中



查找 LoadLibraryA、GetProcAddress 的 RVA

```
000020A8  0000F9ED  Function RVA  0196  GetPrivateProfileStringW
000020AC  0005CA33  Function RVA  0197  GetPrivateProfileStructA
000020B0  0005CB9D  Function RVA  0198  GetPrivateProfileStructW
000020B4  0000AE30  Function RVA  0199  GetProcAddress
000020B8  0002174D  Function RVA  019A  GetProcessAffinityMask
000020BC  000621AD  Function RVA  019B  GetProcessDEPPolicy
```

00002354	00066219	Function RVA	0241	LZRead
00002358	0006618E	Function RVA	0242	LZSeek
0000235C	0008012E	Function RVA	0243	LZStart
00002360	000091AD	Forwarded Name RVA	0244	LeaveCriticalSection -> NTDLL.RtlLeaveCr
00002364	00001D7B	Function RVA	0245	LoadLibraryA
00002368	00001D53	Function RVA	0246	LoadLibraryExA
0000236C	00001AF5	Function RVA	0247	LoadLibraryExW

与之前找到的 kernel32.dll 的基地址相加，得到：

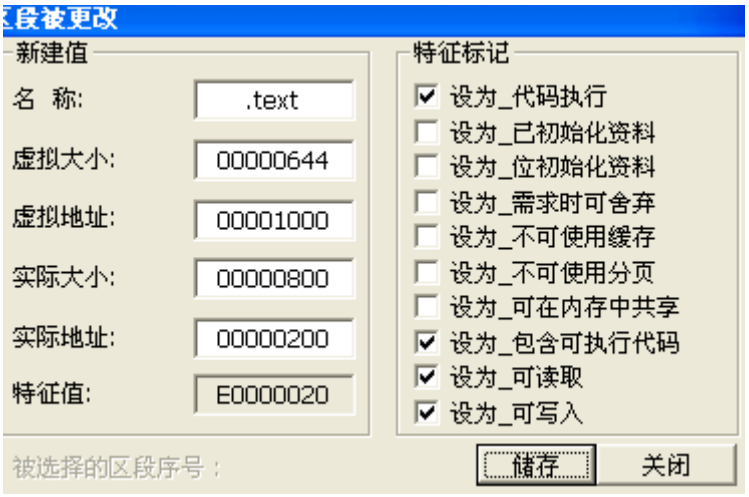
LoadLibraryA=7C80000+1D7B=7C80AD7B,

GetProcAddress=7C80000+AE30=7C80AE30

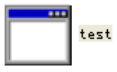
这说明我们找到的地址是正确的

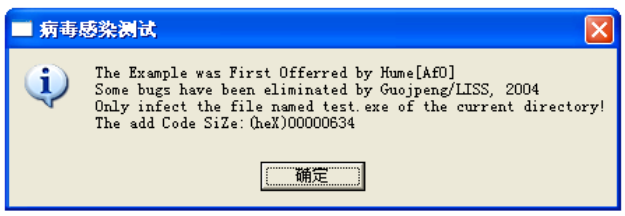
PE 病毒感染分析与清除

使用 masm32 编译教材中的感染例子程序-bookexample-old.rar，使用 Stud_PE 修改代码段属性为可读可写可执行



由于本病毒程序的感染目标是 test.exe，将 HelloWolrd.exe 改名，放到同一目录下，运行 main.exe 对 test.exe 进行感染，打开 test.exe，发现 test.exe 由原来的直接弹出 HelloWolrd 弹框变成了先弹出和病毒一样的弹框再弹出 HelloWorld



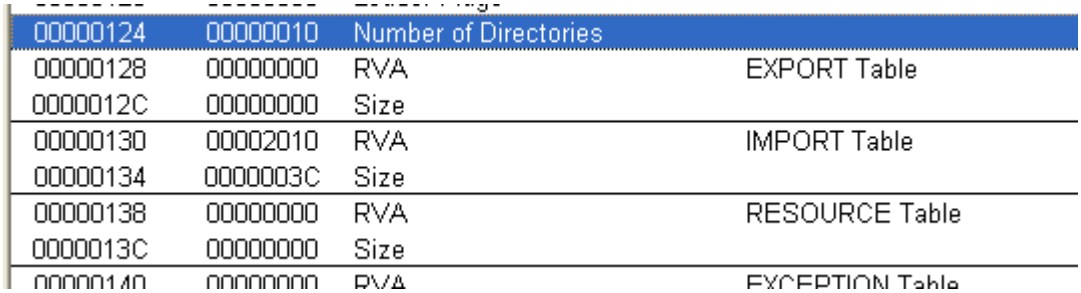


这个现象说明目标程序已经被成功感染

该病毒在感染文件时的具体操作如下：

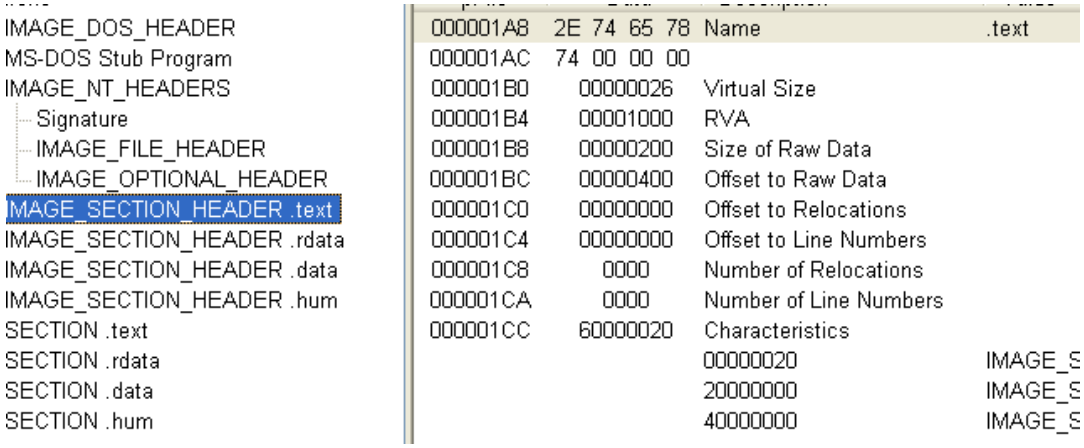
- 1.先定位 kernel32 基址
- 2.定位相关 API 函数地址
- 3.打开“test.exe”文件，获得文件大小，创建映射文件，得到映射对象句柄
- 4.判断 MZ PE 标志位确认为 PE 文件 ，判断感染标志 dark，若已被感染则跳出执行 HOST 程序，否则继续
- 5. 获得 Directory 的个数，定位 section-header 起始位置，在最后一个节区头末尾新添加.hum 节区头
- 6.保存旧的入口点（返回 HOST），修改入口点指向病毒代码起始位置
- 7.写入感染标记，写入病毒代码到.hum 节。

病毒首先定位到文件的 number of directory，可以看到 0x10 项，再从 0x168 开始偏移 0x10*8 字节到达 0x1A8



00000124	00000010	Number of Directories	
00000128	00000000	RVA	EXPORT Table
0000012C	00000000	Size	
00000130	00002010	RVA	IMPORT Table
00000134	0000003C	Size	
00000138	00000000	RVA	RESOURCE Table
0000013C	00000000	Size	
00000140	00000000	RVA	EXCEPTION Table

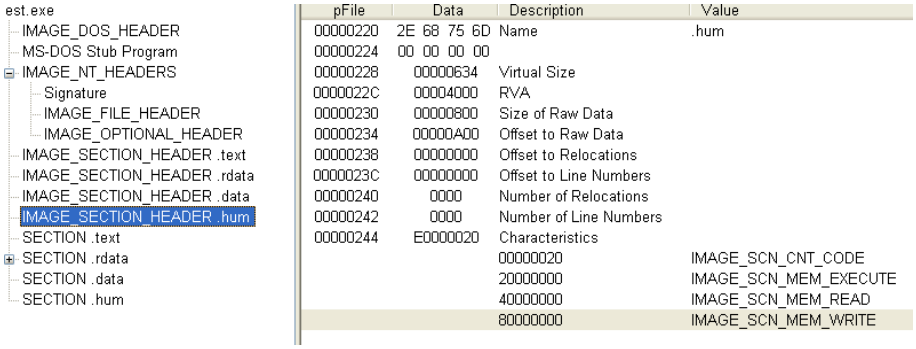
接着查看下一段的首地址进行验证，确实是 0x1A8,说明定位成功



IMAGE_DOS_HEADER
MS-DOS Stub Program
IMAGE_NT_HEADERS
Signature
IMAGE_FILE_HEADER
IMAGE_OPTIONAL_HEADER
IMAGE_SECTION_HEADER .text
IMAGE_SECTION_HEADER .rdata
IMAGE_SECTION_HEADER .data
IMAGE_SECTION_HEADER .hum
SECTION .text
SECTION .rdata
SECTION .data
SECTION .hum

000001A8	2E 74 65 78	Name	.text
000001AC	74 00 00 00		
000001B0	00000026	Virtual Size	
000001B4	00001000	RVA	
000001B8	00000200	Size of Raw Data	
000001BC	00000400	Offset to Raw Data	
000001C0	00000000	Offset to Relocations	
000001C4	00000000	Offset to Line Numbers	
000001C8	0000	Number of Relocations	
000001CA	0000	Number of Line Numbers	
000001CC	60000020	Characteristics	
		00000020	IMAGE_S
		20000000	IMAGE_S
		40000000	IMAGE_S

病毒接着利用最后一个节区头和代码段之间空闲区域，新写入了一个节区头，在这个节区头里定义了相关的名字，RVA 文件大小，将病毒内容进行复制

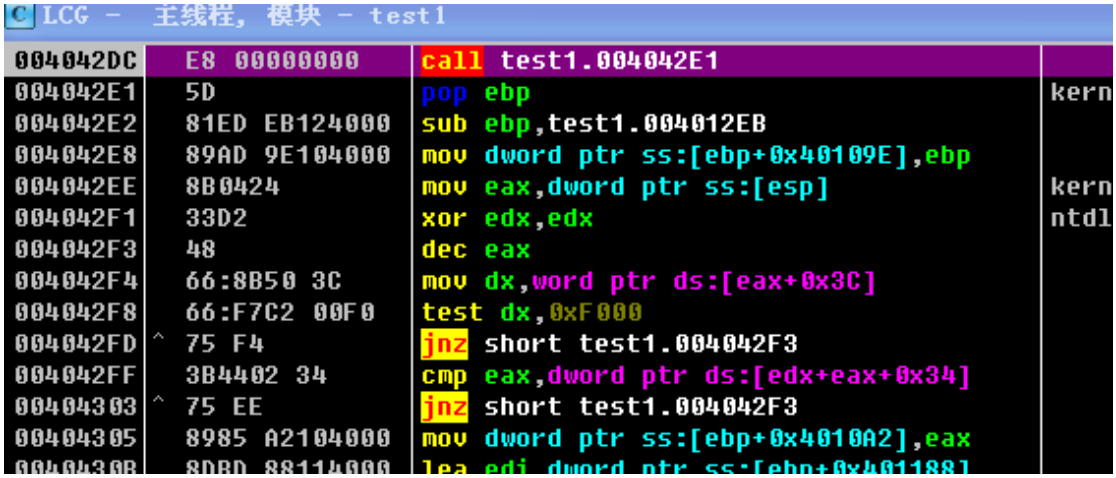


est.exe
IMAGE_DOS_HEADER
MS-DOS Stub Program
IMAGE_NT_HEADERS
Signature
IMAGE_FILE_HEADER
IMAGE_OPTIONAL_HEADER
IMAGE_SECTION_HEADER .text
IMAGE_SECTION_HEADER .rdata
IMAGE_SECTION_HEADER .data
IMAGE_SECTION_HEADER .hum
SECTION .text
SECTION .rdata
SECTION .data
SECTION .hum

pFile	Data	Description	Value
00000220	2E 68 75 6D	Name	.hum
00000224	00 00 00 00		
00000228	00000634	Virtual Size	
0000022C	00004000	RVA	
00000230	00000800	Size of Raw Data	
00000234	00000A00	Offset to Raw Data	
00000238	00000000	Offset to Relocations	
0000023C	00000000	Offset to Line Numbers	
00000240	0000	Number of Relocations	
00000242	0000	Number of Line Numbers	
00000244	E0000020	Characteristics	
		00000020	IMAGE_SCN_CNT_CODE
		20000000	IMAGE_SCN_MEM_EXECUTE
		40000000	IMAGE_SCN_MEM_READ
		80000000	IMAGE_SCN_MEM_WRITE

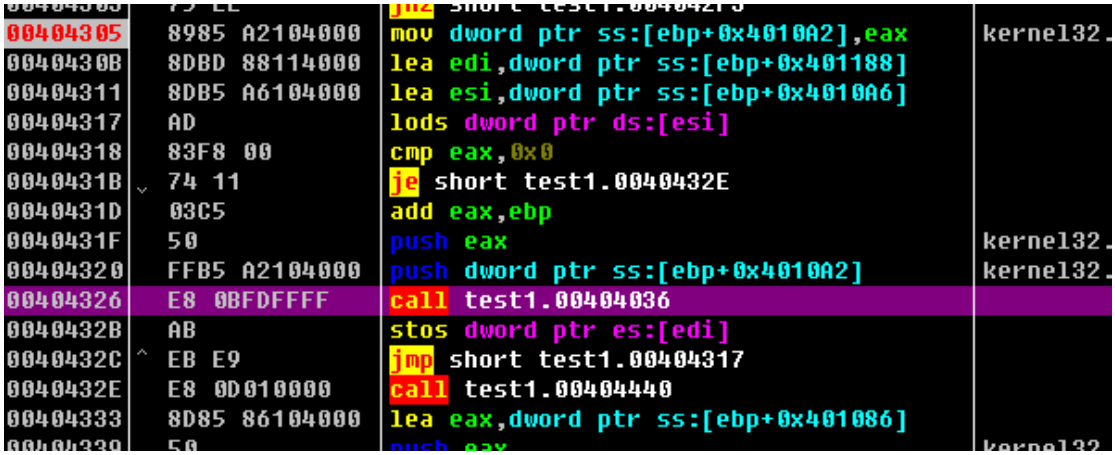
但是这个病毒程序存在一定的问题，利用这个已经感染的 test.exe 去感染计算器 exe 文件时，会报错。这是因为这个 old 版本编译的病毒程序在感染计算器的过程，感染程序用一个变量保存原来的入口点。但是 call 时保存的不是 host 的入口点，而是计算器（即被感染程序）的入口点，所以无法返回。因此造成报错。

使用 ollyDbg 打开原先被感染的 test.exe 进行查看

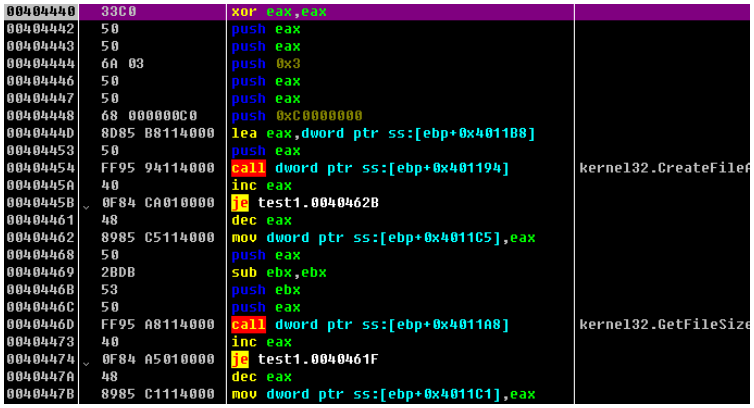


可以看到被感染程序是先 call 下一条指令，相当于把程序的入口点的地址保存到了 ebp 里面，这是病毒用于重定位寻址的。

接着通过 eax 进行循环，定位到 kernel32 的基地址，再寻找病毒定义好的 api 函数地址



再跟踪进入病毒的感染模块，打开 test.exe 文件，获取文件大小。获取内存中 test.exe 的起始地址。



接着病毒对感染标志位进行判断是否感染。如果没有被感染则进行感染，否则提示已经

感染信息。

定位到病毒往源文件写入和保存程序入口点的位置，发现其保存的外壳的程序入口点错误。

```
00404565   8946 10          mov dword ptr ds:[esi+0x10],eax
00404568   8B46 EC          mov eax,dword ptr ds:[esi-0x14]      test1.00401154
0040456B   0346 E8          add eax,dword ptr ds:[esi-0x18]      test1.00401148
0040456E   8946 14          mov dword ptr ds:[esi+0x14],eax
00404571   8985 ED114000    mov dword ptr ss:[ebp+0x4011ED],eax
00404577   8B85 D1114000    mov eax,dword ptr ss:[ebp+0x4011D1]
0040457D   66:FF40 06       inc word ptr ds:[eax+0x6]
00404581   8B58 28          mov ebx,dword ptr ds:[eax+0x28]
00404584   899D E5114000    mov dword ptr ss:[ebp+0x4011E5],ebx
0040458A   8B9D DD114000    mov ebx,dword ptr ss:[ebp+0x4011DD]
00404590   8958 28          mov dword ptr ds:[eax+0x28],ebx
00404593   8B58 50          mov ebx,dword ptr ds:[eax+0x50]
00404596   81C3 34060000    add ebx,0x634
```

这样就导致了程序感染完计算器后无法返回到外壳 helloworld 程序。这样就导致了报错。

修改这个病毒程序的源代码，新加一个入口点 oldEipTemp，相当于缓存的作用。

```
28  pe_Header       dd ?
29  sec_align       dd ?
30  file_align      dd ?
31  newEip          dd ?
32  oldEip          dd ?
33  oldEipTemp      dd ?
34  inc_size        dd ?
35  oldEnd          dd ?
```

同时修改 modipe.asm，修改存入到 oldEipTemp 偏移的位置，而不是存入到 oldEip 中。这样就不会报错了。

```
85  mov     ebx,[eax+28h]           ;eip指针偏移
86  ;---------------------------------------
87  mov     [ebp+oldEipTemp],ebx            ;保存老
88  ;---------------------------------------
89  mov     ebx,[ebp+newEip]
90  mov     [eax+28h],ebx           ;更新指针值
91  ;comment $
```

这就是 new 版本的病毒程序，使用 masm32 对 newexample 进行编译，将 calc.exe 修改为 test.exe，使用 test1.exe 进行感染。打开计算器程序，可以看到计算器也被感染了。先弹出病毒提示框，再显示计算器界面。

下面使用 UltraEdit 对被感染的计算器进行修复。找到可选头第三部分的入口点地址，位 1F2DCH，由于是小端序，反过来查找 DC F2 01 00H,这就是病毒进入的地方

```
1c160h: 6E 74 65 72 00 53 65 74 45 6E 64 4F 66 46 69 6C ; nter.SetEndOfFil
1c170h: 65 00 45 78 69 74 50 72 6F 63 65 73 73 00 31 B7 ; e.ExitProcess.1?
1c180h: 80 7C 30 AE 80 7C 7B 1D 80 7C 28 1A 80 7C EE 94 ; €|0畝|{.€|(.€|顯
1c190h: 80 7C 95 B9 80 7C 04 BA 80 7C D7 9B 80 7C 07 0B ; €|瞞€|.簽|讪€|..
1c1a0h: 81 7C 1E 0C 81 7C 5E 20 83 7C FA CA 81 7C 74 65 ; 厂..厂^ 憶  厂te
1c1b0h: 73 74 2E 65 78 65 00 00 C0 01 00 34 00 00 00 38 ; st.exe..?.4...8
1c1c0h: 00 00 00 00 00 3B 00 F0 00 3B 00 00 10 00 00 00 ; .....;.?;......
1c1d0h: 02 00 00 DC F2 01 00 75 24 01 00 75 24 01 00 00 ; ...茯?..u$..u$..
1c1e0h: 00 00 00 00 C0 01 00 4D 65 73 73 61 67 65 42 6F ; ....?.MessageBo
1c1f0h: 78 41 00 00 00 00 00 B2 A1 B6 BE B8 D0 C8 BE B2 ; xA.....病毒感染?
1c200h: E2 CA D4 00 B2 A1 B6 BE B8 D0 C8 BE B2 E2 CA D4 ; 馐?病毒感染测试
1c210h: 21 00 54 68 65 20 45 78 61 6D 70 6C 65 20 77 61 ; !.The Example wa
1c220h: 73 20 46 69 72 73 74 20 4F 66 66 65 72 72 65 64 ; s First Offerred
```

而后面找到的一个就是原来的入口地址，因此将原来的地址修改为 75 24 01 00 H，保存。再次打开计算器，可以看到不再进行弹窗。

课后习题思考：

- 对 Win10 下的在 32 位及 64 位程序来说，如何通过 PEB 获取 kernel32 基地址？32 位和 64 程序有何差异？

通过 PEB 枚举当前进程空间中用户模块列表可以获取 Kernel32 模块的基地址，fs:[0]指向 TEB，fs:[30h]指向 PEB，PEB 偏移 0ch 是 LDR 指针，可以分别通过加载顺序、内存顺序、初始化顺序获取 Kernel32 模块的基地址

对于 win10 64 位系统，gs:[0x30]指向了 TEB 结构，而[TEB+0x30]指向 PEB 结构，kernel32 在 win10 中位于动态链接库的第三项，在 Xp 中则是第二项。64 位系统寻找时需要使用 gs 寄存器，定位到双向链表的第三项即可。

- 尝试编写一个程序，可用来搜索指定目录下的所有 exe 文件，用 MessageBox 显示每一个被搜索到的 exe 文件名。

```c
#include <windows.h>
#include <string.h>
#include <stdio.h>
int main(int argc,char **argv){
    char * findPath=NULL;
    if(argc>=2){
        if(argc==2){
            findPath=argv[1];
            char* targetfile="\\*.exe";
            strcat(findPath,targetfile);
            printf("%s\n",findPath);
            WIN32_FIND_DATA p;
            HANDLE h = FindFirstFile(findPath,&p);
            char result[100000];
            char tmp[20];
            strcpy(tmp,p.cFileName);
            //printf("%s\n",tmp);
            strcat(result,tmp);
            strcat(result,"\n");
            //printf("%s\n",result);

            while(FindNextFile(h,&p)){
                char tmp[20];
                strcpy(tmp,p.cFileName);
                strcat(result,tmp);
                strcat(result,"\n");
                //printf("%s\n",result);
            }
```

```
            MessageBox(NULL,result,".exe 程序
",MB_OKCANCEL | MB_ICONINFORMATION);
        }
        else
            MessageBox(NULL,"启动参数数量不正确!","错误
",MB_ICONERROR);
    }
    else
        MessageBox(NULL,"未给定路径!","错误", MB_ICONERROR);
    return 0;
}
```

效果如下：

- 编写课本中病毒感染程序的病毒清除程序，其可以用来恢复被感染的任何文件
- 课程提供的病毒感染例子程序在 64 位系统中无法正常感染，请定位其原因，并给出解决方案。

无法正常感染的原因是 64 位系统的 kernel32 的地址不是连续的，在上文的 OD 分析中可以看到，本病毒程序是根据 EAX 递减循环查找 kernel32 基地址的，但在 win10 中这样很可能进入无效的内存区，找不到 kernel32 基地址，也就无法正常感染了。

解决方案：由于 win10 64 系统中，TEB 结构存放在 gs:[0x30]，PEB 存放在 gs:[0x60]，而动态链接库的地址可以通过 PEB_LDR_DATA 找到，因此可以在病毒中编写一个专门用于 64 位系统的定位函数。如果当前在 64 位环境下，找到 PEB_LDR_DATA+0CH 处的动态连接库地址，而 kernel32 在 64 系统中位于第三项，查找到指定位置的项即可找到 kernel32 基地址，这样感染就可以继续进行了。