电子科技大学

计算机科学与工程学院

标 准 实 验 报 告

（实验）课程名称 信息对抗综合设计实验 II

电子科技大学教务处制表

电 子 科 技 大 学

实 验 报 告

学生姓名： 刘芷溢 学 号:2020080907009 指导教师： 李忻洋

一、实验项目名称： Windows 下病毒寄生实验 2

二、实验目的：

建立起对于 Windows 下 PE 文件结构的整体印象，能灵活对硬盘上的 PE 文件和内存中的 PE 映像进行修改，从而理解恶意代码对 PE 文件的感染流程以及相应的清除方式。

熟练掌握 PE 文件所涉及的导入表机制及导出表机制，理解恶意代码攻防中常用的 API 定位机制，以及 PE 文件加载阶段对于所需调用的系统 API 的解析和定位过程。

理解 PE 导出表结构，完成对指定导出函数的定位，并在加载到内存中的 PE 文件映像中定位到对应的函数代码；理解 PE 导入表结构，通过对 PE 文件和 PE 内存映像中导入表结构中 IAT 表内存的对比， 深刻理解 PE 导入表机制；在理解 PE 导入表和导出表的基础上，以在 PE 文件代码段尾部寄生的方式完成病毒寄生，并通过 Patch API 调用指令的方式获取执行权。

三、实验原理：

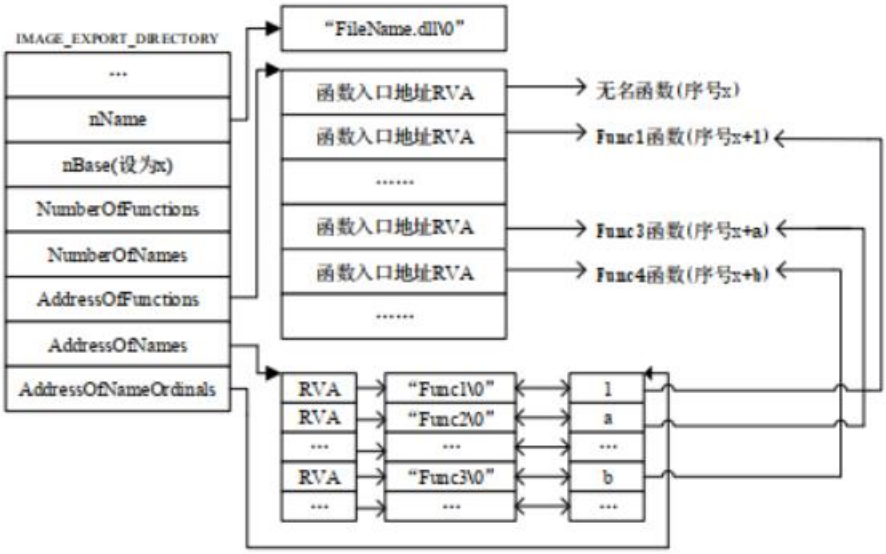
PE 结构：

动态链接库 DLL：

动态链接库（Dynamic Link Libraries）为模块化应用程序提供了一种方式，使得更新和重用程序更加方便动态链接库和可执行文件一样，作为模块被加载内存地址；DLL 是利用导出表机制向外暴露自己的函数；可执行程序（或模块）则可以通过导入表机制去调用其他模块暴露出来的函数。

导出表机制：

一个结构三张表，提供本 PE 文件导出的符号供其他 PE 文件使用。一个函数或变量既可以按名称导出也可以按序号导出；



函数地址表，存放了 DLL 中加载函数的入口地址，函数地址表是按函数序号递增的方式排列的，但是，要注意序号不等于索引，序号有最小序号，函数地址表的索引 i＝ 函数序号 － 最小序号；

函数名表，我们一般通过函数名查找函数地址，由于部分函数只有序号没有函数名，无法实现由函数名到函数地址表的直接映射，在这种情况下，采用间接映射的方式，加入函数名序号表（函数地址索引表）；

函数地址序号表，用它来实现函数名—函数地址表索引号—函数地址的间接映射，它存放的就是函数名在函数地址表中的索引号。

导出表结构：

```
typedef struct _IMAGE_EXPORT_DIRECTORY {
    DWORD   Characteristics;            //未使用，总是定义为0
    DWORD   TimeDateStamp;              //文件生成时的时间戳
    WORD    MajorVersion;               //未使用，总是定义为0
    WORD    MinorVersion;               //未使用，总是定义为0
    DWORD   Name;                       //模块的真实名称的RVA
    DWORD   Base;                       //基数，加上序数就是函数地址数组的索引值

    DWORD   NumberOfFunctions;          //导出函数的总数
    DWORD   NumberOfNames               //以名称方式导出的函数的总数
    DWORD   AddressOfFunctions;         //指向导出函数地址的RVA
    DWORD   AddressOfNames;             //指向导出函数名字的RVA
    DWORD   AddressOfNameOrdinals;      //指向导出函数序号的RVA
} IMAGE_EXPORT_DIRECTORY, *PIMAGE_EXPORT_DIRECTORY;
```

第一种方法是按函数名导出。此方法要求使用该导出函数的 PE 文件知晓本模块导出函数的名称，并在其导入表的 IMAGE_THUNK_DATA 中指明是按函数名称导入（最高位清 0）。为实现此种导出方法，既可以使用 Visual C++中定义的关键字 __declspec(dllexport)修饰对应函数，也可以采用传统方法，在模块定义文件中指明要导出的函数名。

第二种方法是按序号导出。此方法要求使用该导出函数的 PE 文件知晓本模块导出函数的序号，并在其导入表的 IMAGE_THUNK_DATA 中指明是按序号导入（最高位置 1）。要按照序号导出，必须采用传统方法，在模块定义文件中指明要导出函数名所对应的序号。

导入表原理：

导入表是动态链接机制的重要组成部分。所谓导入函数，即该函数被某模块调用， 但其并不在该模块中定义，仅将诸如函数名、其所在的 DLL 名等信息保存在该模块的导入表中。PE 文件的导入表

的位置和大小可由可选头中数据目录表结构数组第二个元素中的信息来确定。其中的 VirtualAddress 字段即指示了导入表描述符结构数组的 RVA，该结构数组的每一个成员都代表一个 DLL，表明本 PE 文件需要用到这些 DLL 中的某些导出函数，该结构数组的最后一个成员是一个空描述符；

```
typedef struct _IMAGE_IMPORT_DESCRIPTOR
{
    union
    {
        DWORD    Characteristics;
        DWORD    OriginalFirstThunk;
    } DUMMYUNIONNAME;
    DWORD    TimeDateStamp;
    DWORD    ForwarderChain;
    DWORD    Name;
    DWORD    FirstThunk;
} IMAGE_IMPORT_DESCRIPTOR;
typedef IMAGE_IMPORT_DESCRIPTOR UNALIGNED
*PIMAGE_IMPORT_DESCRIPTOR;
```

系统加载时调用模块的导入表初始化算法：

IAT 初始化算法：（预先绑定时 IAT 中是实际地址，此时只能遍历 INT）

1 从导入表目录获取 IMAGE_IMPORT_DESCRIPTOR 表入口，获取该表的当前项， 每项代表一个被引用的 DLL，从其中 DLL 名 RVA 段可获取 DLL 名，下面就引入此 DLL

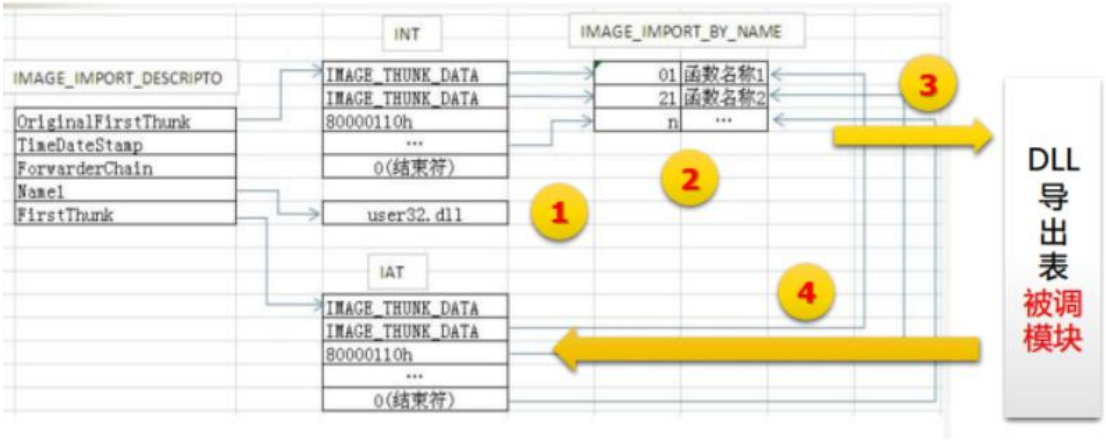
2 系统遍历编译时生成的每个 IAT（如果有 INT 则判断 INT，当预先绑定时，只能遍历 INT），比如第 2 项，其中存储的值是 IMAGE_IMPORT_BY_NAME 中对应项的 RVA

3 找到该项，获得其函数名串，以\0 结尾，通过对应 DLL 导出表找到相关函数的加载地址

4 然后将其放入 IAT 第 2 项（此时， IAT 表项的值才变为了函数的加载地址）

5 如此遍历 IAT，将所有项都填入对应函数入口地址

6 遍历 IMAGE_IMPORT_DESCRIPTOR 表，对所有 DLL 都做 2~4 步处理。



Patch 指令：

分析一些正常程序常用的 API 函数调用语句，如 kernal32.dll 中的 WriteFile 函数；对导入函数调用指令进行 Patch；这些指令的可靠度很高，可以确保大概率被执行。

CALL [xx xx xx xx] 对应的关键机器码是 FF 15 xx xx xx xx

CALL … JMP [xx xx xx xx] 对应的关键机器码是 FF 25 xx xx xx xx

四、实验环境（设备、元器件）：

个人 PC 机；Windows10 系统；notpad.exe；user32.dll

五、实验步骤：

实验一：PE 导出表实验

1.找到 DLL 导出表的入口 RVA

RVA：003900；Size：4BA9

0000014C	00000010	Number of Data Directories	
00000150	00003900	RVA	EXPORT Table
00000154	00004BA9	Size	
00000158	0005E638	RVA	IMPORT Table
0000015C	00000050	Size	

2.根据 RVA 找到导出表位置

1000 < 3900 < 61000；说明位于.text 节

RVA	Data	Description	Value
000001D0	2E 74 65 78	Name	.text
000001D4	74 00 00 00		
000001D8	0005F283	Virtual Size	
000001DC	00001000	RVA	
000001E0	0005F400	Size of Raw Data	
000001E4	00000400	Pointer to Raw Data	

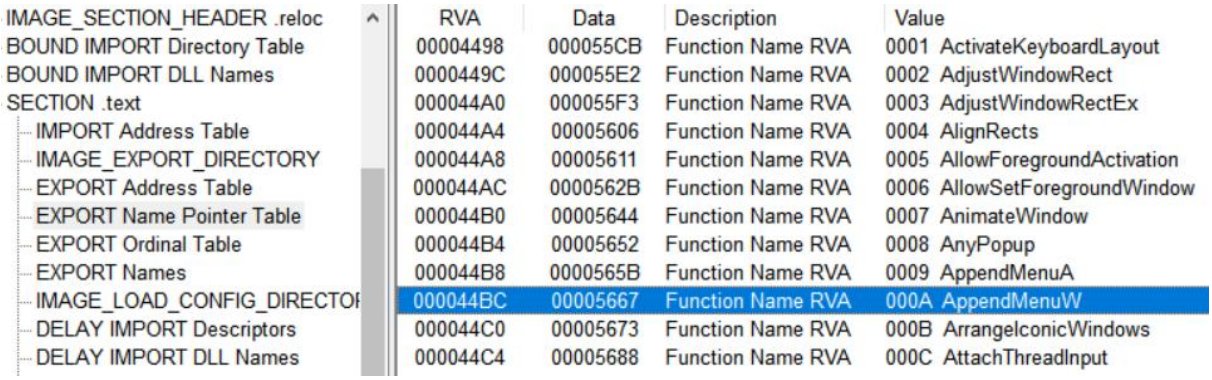
RVA	Data	Description	Value
000001F8	2E 64 61 74	Name	.data
000001FC	61 00 00 00		
00000200	00001180	Virtual Size	
00000204	00061000	RVA	
00000208	00000C00	Size of Raw Data	
0000020C	0005F800	Pointer to Raw Data	

Ordinal Base：加载位置，查看 FOA，可以验证导出表的内容

RVA	Data	Description	Value
00003900	00000000	Characteristics	
00003908	0000	Major Version	
0000390A	0000	Minor Version	
0000390C	000055C0	Name RVA	USER32.dll
00003910	00000001	Ordinal Base	
00003914	000002DC	Number of Functions	
00003918	000002DC	Number of Names	
0000391C	00003928	Address Table RVA	
00003920	00004498	Name Pointer Table RVA	
00003924	00005008	Ordinal Table RVA	

3.函数名指针表

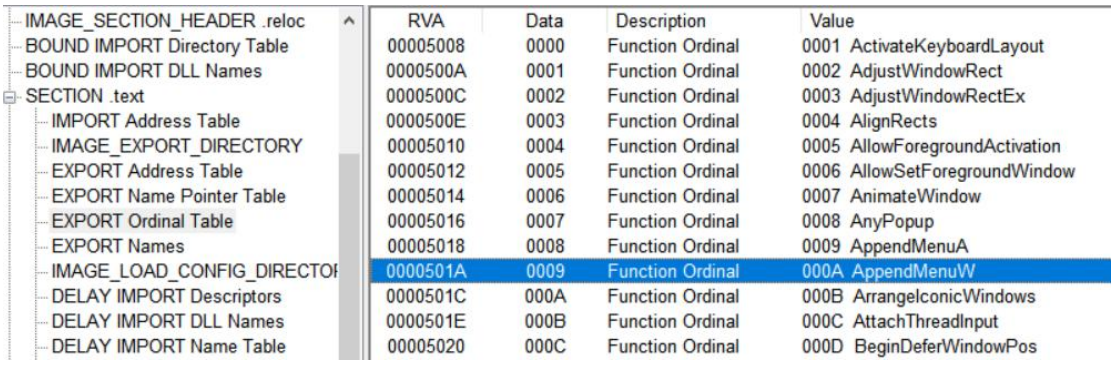
根据 Name Pointer Table RVA（4498）找到该表；找到第 10 个函数



文件偏移为 4A60 处是我们要找的函数名

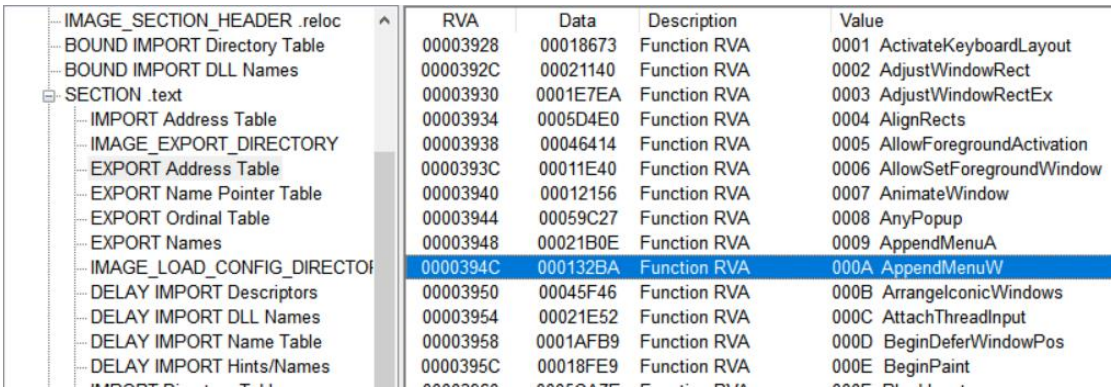
4.查找函数地址

查找 Export Ordinal Table，找到地址索引表，查看值是 AppendMenuW 的表项



	RVA	Data	Description	Value
IMAGE_SECTION_HEADER .reloc	00005008	0000	Function Ordinal	0001 ActivateKeyboardLayout
BOUND IMPORT Directory Table	0000500A	0001	Function Ordinal	0002 AdjustWindowRect
BOUND IMPORT DLL Names	0000500C	0002	Function Ordinal	0003 AdjustWindowRectEx
SECTION .text	0000500E	0003	Function Ordinal	0004 AlignRects
IMPORT Address Table	00005010	0004	Function Ordinal	0005 AllowForegroundActivation
IMAGE_EXPORT_DIRECTORY	00005012	0005	Function Ordinal	0006 AllowSetForegroundWindow
EXPORT Address Table	00005014	0006	Function Ordinal	0007 AnimateWindow
EXPORT Name Pointer Table	00005016	0007	Function Ordinal	0008 AnyPopup
EXPORT Ordinal Table	00005018	0008	Function Ordinal	0009 AppendMenuA
EXPORT Names	0000501A	0009	Function Ordinal	000A AppendMenuW
IMAGE_LOAD_CONFIG_DIRECTOR	0000501C	000A	Function Ordinal	000B ArrangeIconicWindows
DELAY IMPORT Descriptors	0000501E	000B	Function Ordinal	000C AttachThreadInput
DELAY IMPORT DLL Names	00005020	000C	Function Ordinal	000D BeginDeferWindowPos
DELAY IMPORT Name Table				

根据上面得到的索引，在 Address Table 的 RVA 找到函数地址



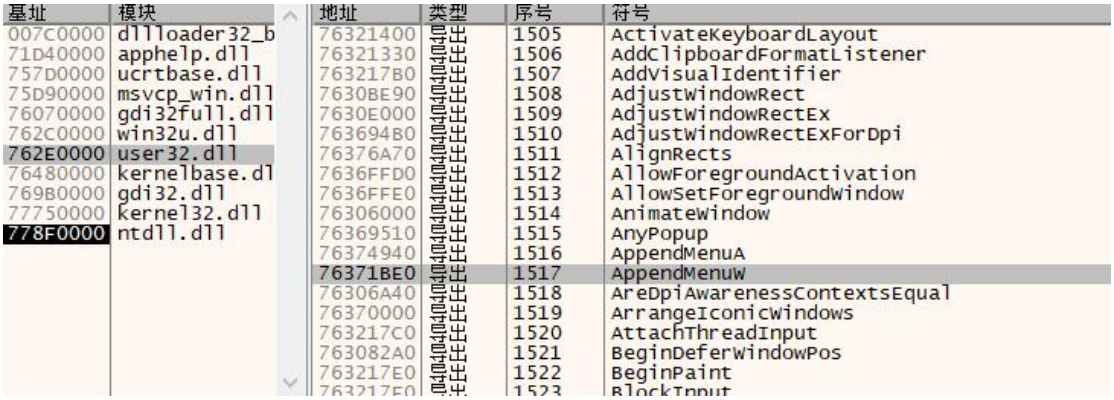
	RVA	Data	Description	Value
IMAGE_SECTION_HEADER .reloc	00003928	00018673	Function RVA	0001 ActivateKeyboardLayout
BOUND IMPORT Directory Table	0000392C	00021140	Function RVA	0002 AdjustWindowRect
BOUND IMPORT DLL Names	00003930	0001E7EA	Function RVA	0003 AdjustWindowRectEx
SECTION .text	00003934	0005D4E0	Function RVA	0004 AlignRects
IMPORT Address Table	00003938	00046414	Function RVA	0005 AllowForegroundActivation
IMAGE_EXPORT_DIRECTORY	0000393C	00011E40	Function RVA	0006 AllowSetForegroundWindow
EXPORT Address Table	00003940	00012156	Function RVA	0007 AnimateWindow
EXPORT Name Pointer Table	00003944	00059C27	Function RVA	0008 AnyPopup
EXPORT Ordinal Table	00003948	00021B0E	Function RVA	0009 AppendMenuA
EXPORT Names	0000394C	000132BA	Function RVA	000A AppendMenuW
IMAGE_LOAD_CONFIG_DIRECTOR	00003950	00045F46	Function RVA	000B ArrangeIconicWindows
DELAY IMPORT Descriptors	00003954	00021E52	Function RVA	000C AttachThreadInput
DELAY IMPORT DLL Names	00003958	0001AFB9	Function RVA	000D BeginDeferWindowPos
DELAY IMPORT Name Table	0000395C	00018FE9	Function RVA	000E BeginPaint
DELAY IMPORT Hints/Names				

5.验证找到的函数地址

函数地址表中的函数地址：

```
000126B0: FF C9 C2 10 00 90 90 90 90 90 8B FF 55 8B EC 83
000126C0: EC 30 8D 45 D0 50 FF 75 14 FF 75 10 FF 75 0C FF
000126D0: 75 08 E8 7D A8 FF FF 6A 00 8D 45 D0 50 6A 01 68
```

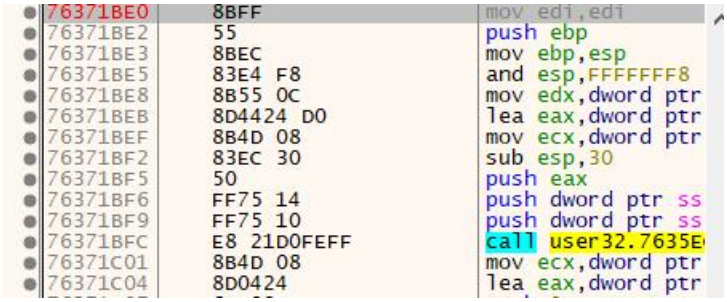
X32dbg 加载后 user32.dll



基址	模块
007C0000	dllloader32_b
71D40000	apphelp.dll
757D0000	ucrtbase.dll
75D90000	msvcp_win.dll
76070000	gdi32full.dll
762C0000	win32u.dll
762E0000	user32.dll
76480000	kernelbase.dl
769B0000	gdi32.dll
77750000	kernel32.dll
778F0000	ntdll.dll

地址	类型	序号	符号
76321400		1505	ActivateKeyboardLayout
76321330		1506	AddClipboardFormatListener
763217B0		1507	AddVisualIdentifier
7630BE90		1508	AdjustWindowRect
7630E000		1509	AdjustWindowRectEx
763694B0		1510	AdjustWindowRectExForDpi
76376A70		1511	AlignRects
7636FFD0		1512	AllowForegroundActivation
7636FFE0		1513	AllowSetForegroundWindow
76306000		1514	AnimateWindow
76369510		1515	AnyPopup
76374940		1516	AppendMenuA
76371BE0		1517	AppendMenuW
76306A40		1518	AreDpiAwarenessContextsEqual
76370000		1519	ArrangeIconicWindows
763217C0		1520	AttachThreadInput
763082A0		1521	BeginDeferWindowPos
763217E0		1522	BeginPaint
763217F0		1523	BlockInput

加载到内存中的函数地址：



结果：函数入口内容完全一致，说明我们找到了函数入口

实验二：PE 导入表机制实验（验证第 10 个函数 AbortDoc）

1.观察加载前的导入表情况

导入表入口 RVA 7604

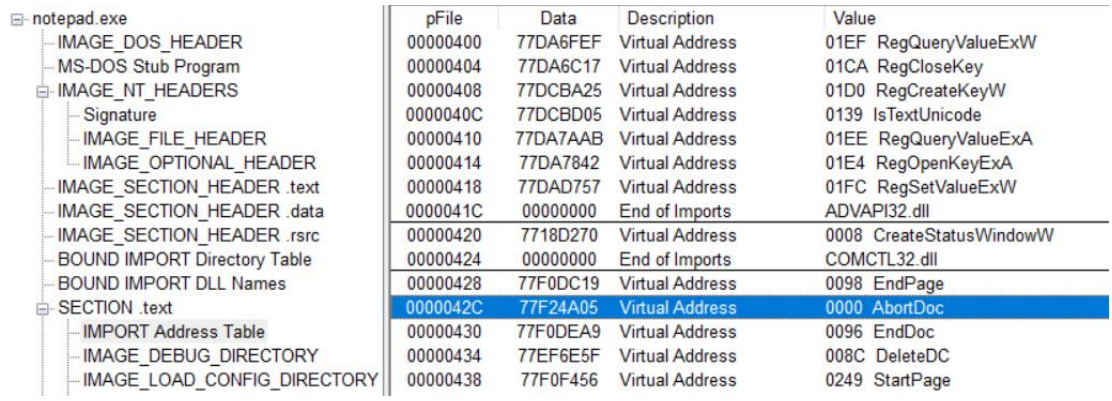
00000154	00000010	Number of Data Directories	
00000158	00000000	RVA	EXPORT Table
0000015C	00000000	Size	
00000160	00007604	RVA	IMPORT Table
00000164	000000C8	Size	

1000 < 7604 < 8748，所以位于.text 节

RVA	Data	Description	Value
000001D8	2E 74 65 78	Name	.text
000001DC	74 00 00 00		
000001E0	00007748	Virtual Size	
000001E4	00001000	RVA	
000001E8	00007800	Size of Raw Data	
000001EC	00000400	Pointer to Raw Data	

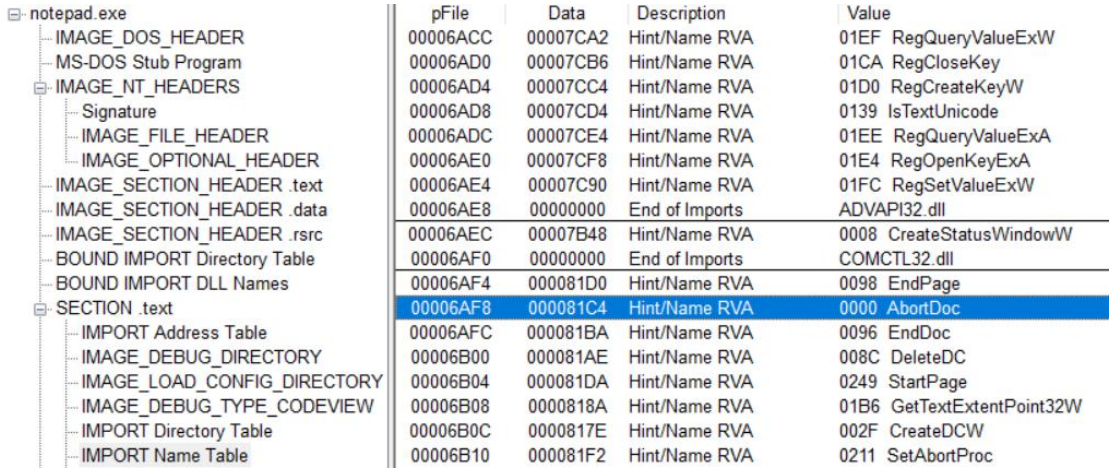
2.观察 INT 和 IAT 表

IAT：



	pFile	Data	Description	Value
notepad.exe				
IMAGE_DOS_HEADER	00000400	77DA6FEF	Virtual Address	01EF RegQueryValueExW
MS-DOS Stub Program	00000404	77DA6C17	Virtual Address	01CA RegCloseKey
IMAGE_NT_HEADERS	00000408	77DCBA25	Virtual Address	01D0 RegCreateKeyW
Signature	0000040C	77DCBD05	Virtual Address	0139 IsTextUnicode
IMAGE_FILE_HEADER	00000410	77DA7AAB	Virtual Address	01EE RegQueryValueExA
IMAGE_OPTIONAL_HEADER	00000414	77DA7842	Virtual Address	01E4 RegOpenKeyExA
IMAGE_SECTION_HEADER .text	00000418	77DAD757	Virtual Address	01FC RegSetValueExW
IMAGE_SECTION_HEADER .data	0000041C	00000000	End of Imports	ADVAPI32.dll
IMAGE_SECTION_HEADER .rsrc	00000420	7718D270	Virtual Address	0008 CreateStatusWindowW
BOUND IMPORT Directory Table	00000424	00000000	End of Imports	COMCTL32.dll
BOUND IMPORT DLL Names	00000428	77F0DC19	Virtual Address	0098 EndPage
SECTION .text	0000042C	77F24A05	Virtual Address	0000 AbortDoc
IMPORT Address Table	00000430	77F0DEA9	Virtual Address	0096 EndDoc
IMAGE_DEBUG_DIRECTORY	00000434	77EF6E5F	Virtual Address	008C DeleteDC
IMAGE_LOAD_CONFIG_DIRECTORY	00000438	77F0F456	Virtual Address	0249 StartPage

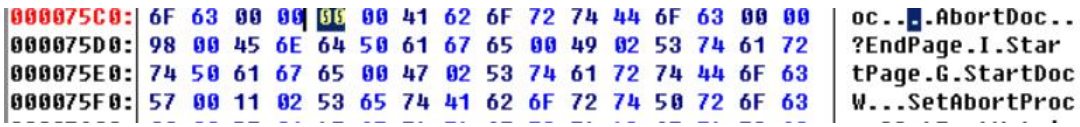
INT：



	pFile	Data	Description	Value
notepad.exe				
IMAGE_DOS_HEADER	00006ACC	00007CA2	Hint/Name RVA	01EF RegQueryValueExW
MS-DOS Stub Program	00006AD0	00007CB6	Hint/Name RVA	01CA RegCloseKey
IMAGE_NT_HEADERS	00006AD4	00007CC4	Hint/Name RVA	01D0 RegCreateKeyW
Signature	00006AD8	00007CD4	Hint/Name RVA	0139 IsTextUnicode
IMAGE_FILE_HEADER	00006ADC	00007CE4	Hint/Name RVA	01EE RegQueryValueExA
IMAGE_OPTIONAL_HEADER	00006AE0	00007CF8	Hint/Name RVA	01E4 RegOpenKeyExA
IMAGE_SECTION_HEADER .text	00006AE4	00007C90	Hint/Name RVA	01FC RegSetValueExW
IMAGE_SECTION_HEADER .data	00006AE8	00000000	End of Imports	ADVAPI32.dll
IMAGE_SECTION_HEADER .rsrc	00006AEC	00007B48	Hint/Name RVA	0008 CreateStatusWindowW
BOUND IMPORT Directory Table	00006AF0	00000000	End of Imports	COMCTL32.dll
BOUND IMPORT DLL Names	00006AF4	000081D0	Hint/Name RVA	0098 EndPage
SECTION .text	00006AF8	000081C4	Hint/Name RVA	0000 AbortDoc
IMPORT Address Table	00006AFC	000081BA	Hint/Name RVA	0096 EndDoc
IMAGE_DEBUG_DIRECTORY	00006B00	000081AE	Hint/Name RVA	008C DeleteDC
IMAGE_LOAD_CONFIG_DIRECTORY	00006B04	000081DA	Hint/Name RVA	0249 StartPage
IMAGE_DEBUG_TYPE_CODEVIEW	00006B08	0000818A	Hint/Name RVA	01B6 GetTextExtentPoint32W
IMPORT Directory Table	00006B0C	0000817E	Hint/Name RVA	002F CreateDCW
IMPORT Name Table	00006B10	000081F2	Hint/Name RVA	0211 SetAbortProc

3.查看表中的 RVA 是否指向函数名（AbortDoc）

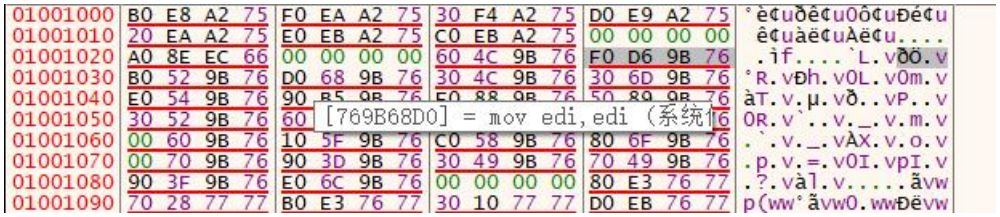
文件偏移：81C4 – 1000 + 400 = 75C4；确实指向了相应的函数名



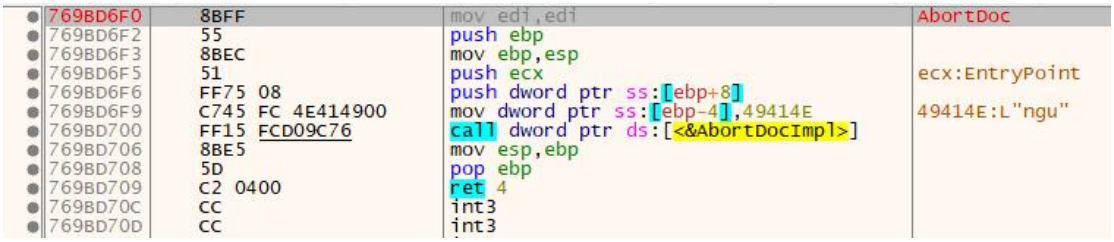
4.查看加载后的 INT 表

实际加载基地址 + 入口 RVA = 入口点实际地址

加载基地址：1000000；而 IAT 为节首先加载字段，所以加载基地址+节的 RVA = 1001000；AbortDoc 对应的字节码是 F0 D6 9B 76



函数入口确实是加载后 IAT 的内容



实验三：EPO 指令 Patch

1.生成病毒寄生代码

E8 04 00 00 00 90 90 90 90 58 BB 09 70 90 80 89 18 E9 XX XX XX XX；

最后我们需要替换原 API 函数的入口点

```
int main()
{
    _asm
    {
        call code_start
        nop
        nop
        nop
        nop
    code_start:
        pop eax
        mov ebx, 0x80907009
        mov [eax], ebx
        jmp code_start
    }
}
```

2.病毒代码寄生到.text 节

在文件最后（Pointer to Raw Data + VirtualSize）400 + 7748 = 7B48

RVA	Data	Description	Value
000001D8	2E 74 65 78	Name	.text
000001DC	74 00 00 00		
000001E0	00007748	Virtual Size	
000001E4	00001000	RVA	
000001E8	00007800	Size of Raw Data	
000001EC	00000400	Pointer to Raw Data	
000001F0	00000000	Pointer to Relocations	
000001F4	00000000	Pointer to Line Numbers	
000001F8	0000	Number of Relocations	
000001FA	0000	Number of Line Numbers	
000001FC	60000020	Characteristics	
		00000020	IMAGE_SCN_CNT_CODE
		20000000	IMAGE_SCN_MEM_EXECUTE
		40000000	IMAGE_SCN_MEM_READ

3.更新 PE 相关字段，使病毒代码可以加载到内存

修改 virtualsize（1e0h）：7748 + 16 = 775E

```
000001E0: 48 77 00 00 00 10 00 00 00 78 00 00 00 04 00 00
000001F0: 00 00 00 00 00 00 00 00 00 00 00 00 20 00 00 60
00000200: 2E 64 61 74 61 00 00 00 A8 1B 00 00 00 90 00 00
00000210: 00 08 00 00 00 7C 00 00 00 00 00 00 00 00 00 00
00000220: 00 00 00 00 40 00 00 C0 2E 72 73 72 63 00 00 00
00000230: 20 7F 00 00 00 B0 00 00 00 80 00 00 00 84 00 00
00000240: 00 00 00 00 00 00 00 00 00 00 00 00 40 00 00 40
00000250: A2 BD 02 48 58 00 00 00 B6 BD 02 48 65 00 00 00
00000260: CA BD 02 48 71 00 00 00 6C BD 02 48 7E 00 00 00
```

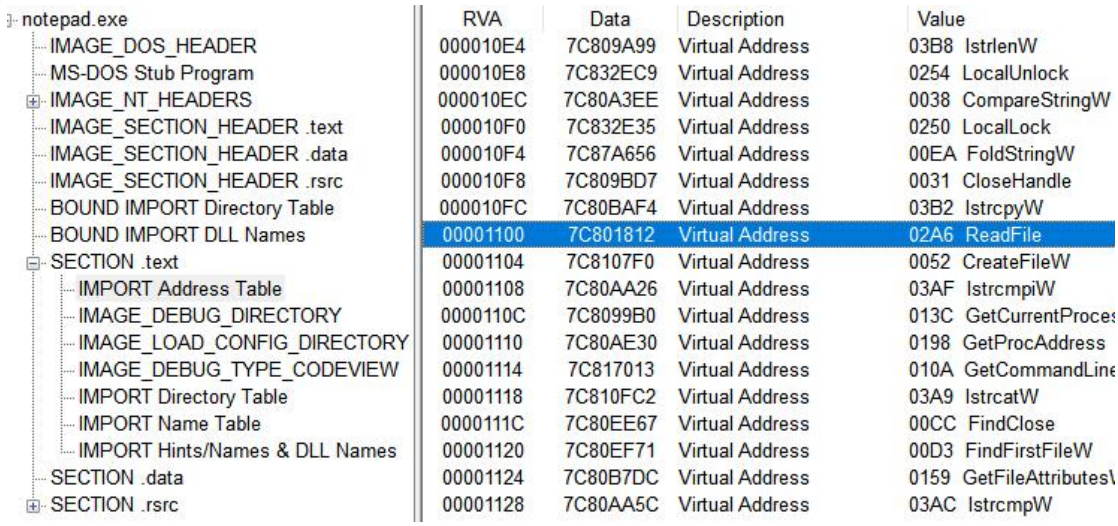
```
000001E0: 5E 77 00 00 00 10 00 00 00 78 00 00 00 04 00 00
000001F0: 00 00 00 00 00 00 00 00 00 00 00 00 20 00 00 60
00000200: 2E 64 61 74 61 00 00 00 A8 1B 00 00 00 90 00 00
00000210: 00 08 00 00 00 7C 00 00 00 00 00 00 00 00 00 00
00000220: 00 00 00 00 40 00 00 C0 2E 72 73 72 63 00 00 00
00000230: 20 7F 00 00 00 B0 00 00 00 80 00 00 00 84 00 00
00000240: 00 00 00 00 00 00 00 00 00 00 00 00 40 00 00 40
00000250: A2 BD 02 48 58 00 00 00 B6 BD 02 48 65 00 00 00
00000260: CA BD 02 48 71 00 00 00 6C BD 02 48 7E 00 00 00
```

修改属性（1fch）：E0000020（增加内存可写属性）

```
000001E0: 5E 77 00 00 00 10 00 00 00 78 00 00 00 04 00 00
000001F0: 00 00 00 00 00 00 00 00 00 00 00 00 00 20 00 00 E0
00000200: 2E 64 61 74 61 00 00 00 A8 1B 00 00 00 90 00 00
00000210: 00 08 00 00 00 7C 00 00 00 00 00 00 00 00 00 00
```

4.选用需要 Patch 的 API（kernel32.dll 的 ReadFile API）

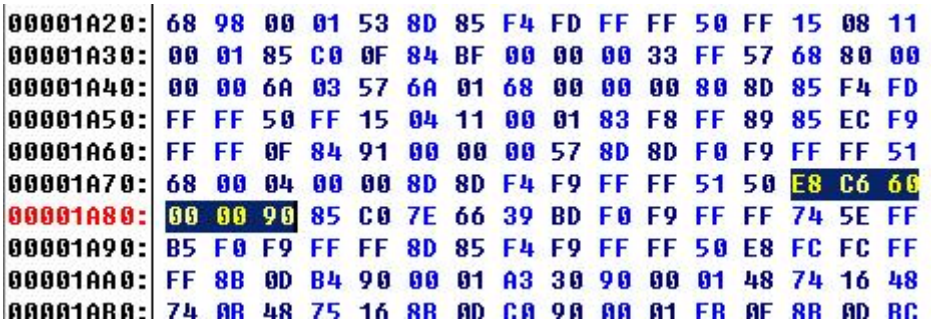
ImageBase = 1000000；ReadFile 对应 VA 为 01001100



在文件中查找 FF 15 00 11 00 01；文件地址为 1A7D；对应的 RVA 地址 = 1A7D – 400 + 1000 = 267D

5.Patch 该 API 调用指令到病毒寄生代码

相对偏移量：7748 + 1000 – 267D - 5 = 60C6；FF 15 00 11 00 01 改为 Call 指令；

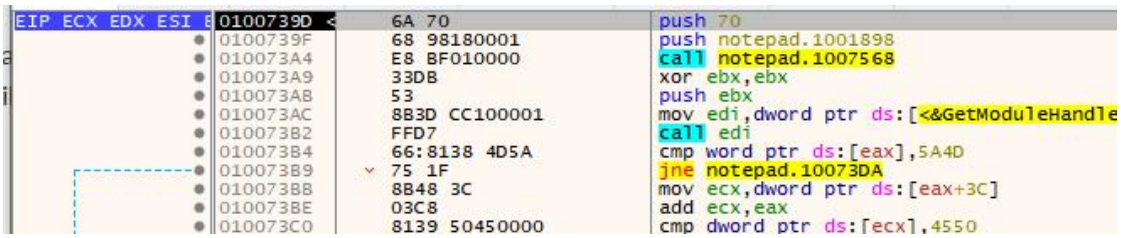


6.病毒尾部跳回原 API 函数的 JMP 指令

加载前入口点预计加载位置：ImageBase + Address of Entry Point = 01000000 + 739D = 0100739D

RVA	Data	Description	Value
000000F8	010B	Magic	IMAGE_NT_OPTIONAL_HDR32_MAGIC
000000FA	07	Major Linker Version	
000000FB	0A	Minor Linker Version	
000000FC	00007800	Size of Code	
00000100	00008800	Size of Initialized Data	
00000104	00000000	Size of Uninitialized Data	
00000108	0000739D	Address of Entry Point	
0000010C	00001000	Base of Code	
00000110	00009000	Base of Data	
00000114	01000000	Image Base	
00000118	00001000	Section Alignment	
0000011C	00000200	File Alignment	

加载后入口点的地址：说明加载后与约定的地址相同；程序没有重定位。

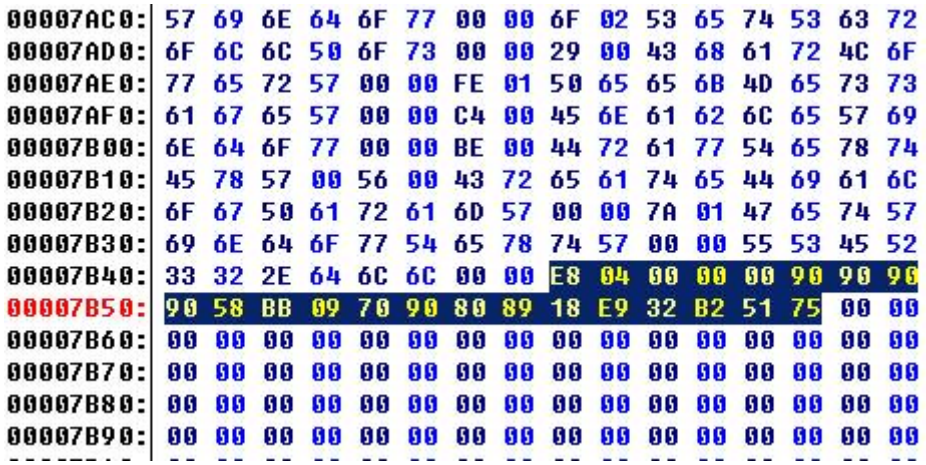


查看 API 加载地址：76523990

跳转的源地址为病毒寄生代码的尾部，即为新的 VirtualSize +节 RVA + ImageBase = 775E + 1000 + 1000000 = 100875E

最后跳转回 API 偏移：76523990 - （775E + 1000 + 1000000）= 7551B232

整体病毒代码：E8 04 00 00 00 90 90 90 90 58 BB 09 70 90 80 89 18 E9 32 B2 51 75

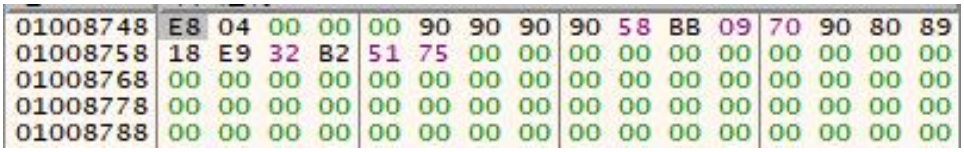


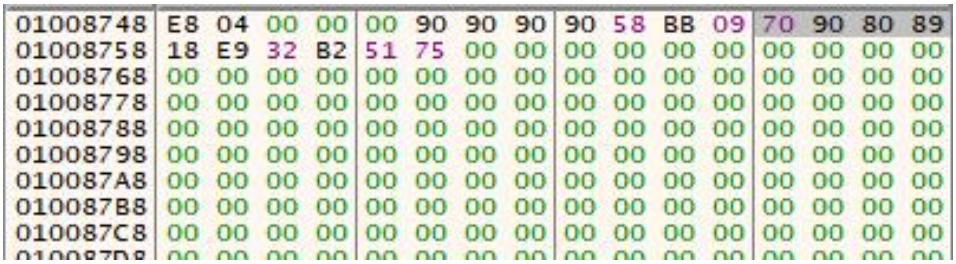
结果：

原来调用的 API 地址：

跳转回原来的 API：

内存中初始化为 90 90 90 90 的值改变为设置的值 70 90 80 89：





六、实验结论：

实验一：PE 导出表实验

PE 导出表机制主要涉及到三个表；找到函数名指针表的第 10 项，这里是 AppendMenuW；在函数地址索引表中找到这个函数的表项，得到该函数的索引；到函数地址表中按索引查找表项，最后得到函数入口地址；通过实验发现，函数地址表中存放的确实是函数入口地址。

验证找到的函数地址

```
000126B0: FF C9 C2 10 00 90 90 90 90 90 8B FF 55 8B EC 83
000126C0: EC 30 8D 45 D0 50 FF 75 14 FF 75 10 FF 75 0C FF
000126D0: 75 08 E8 7D A8 FF FF 6A 00 8D 45 D0 50 6A 01 68
```

结果：函数入口内容完全一致，说明我们找到了函数入口

```
76371BE0    8BFF            mov edi,edi
76371BE2    55              push ebp
76371BE3    8BEC            mov ebp,esp
76371BE5    83E4 F8         and esp,FFFFFFF8
76371BE8    8B55 0C         mov edx,dword ptr
76371BEB    8D4424 D0       lea eax,dword ptr
76371BEF    8B4D 08         mov ecx,dword ptr
76371BF2    83EC 30         sub esp,30
76371BF5    50              push eax
76371BF6    FF75 14         push dword ptr ss
76371BF9    FF75 10         push dword ptr ss
76371BFC    E8 21D0FEFF     call user32.7635E
76371C01    8B4D 08         mov ecx,dword ptr
76371C04    8D0424          lea eax,dword ptr
```

实验二：分析 Windows 下的导入表机制

IAT 可能出现预先绑定，此时更新导入表遍历 INT；INT 中存储的值是 IMAGE_IMPORT_BY_NAME 中对应项的 RVA 找到该项，获得其函数名串，以\0 结尾，通过对应 DLL 导出表找到相关函数的加载地址。

实验发现，加载前 IAT 表存放的是指向函数名的 RVA；加载后 IAT 表的内容为对应函数的入口地址。

加载前文件偏移：81C4 – 1000 + 400 = 75C4；确实指向了相应的函数名

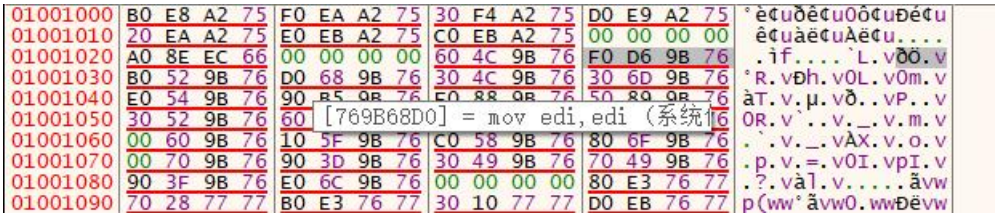
```
000075C0: 6F 63 00 00 00 00 41 62 6F 72 74 44 6F 63 00 00  oc....AbortDoc..
000075D0: 98 00 45 6E 64 50 61 67 65 00 49 02 53 74 61 72  ?EndPage.I.Star
000075E0: 74 50 61 67 65 00 47 02 53 74 61 72 74 44 6F 63  tPage.G.StartDoc
000075F0: 57 00 11 02 53 65 74 41 62 6F 72 74 50 72 6F 63  W...SetAbortProc
```

查看加载后的 INT 表

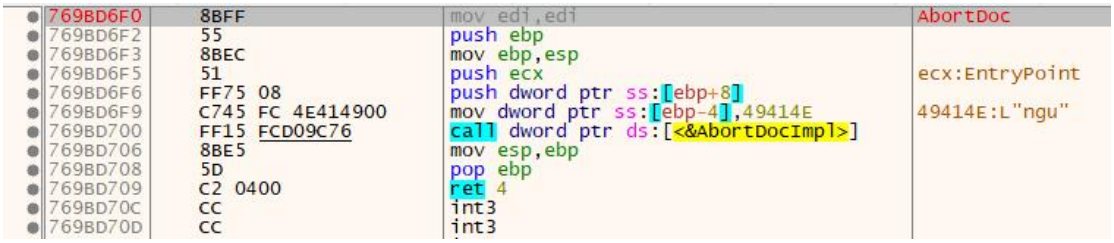
实际加载基地址 + 入口 RVA = 入口点实际地址

加载基地址：1000000；而 IAT 为节首先加载字段，所以加载基地址+

节的 RVA = 1001000；AbortDoc 对应的字节码是 F0 D6 9B 76



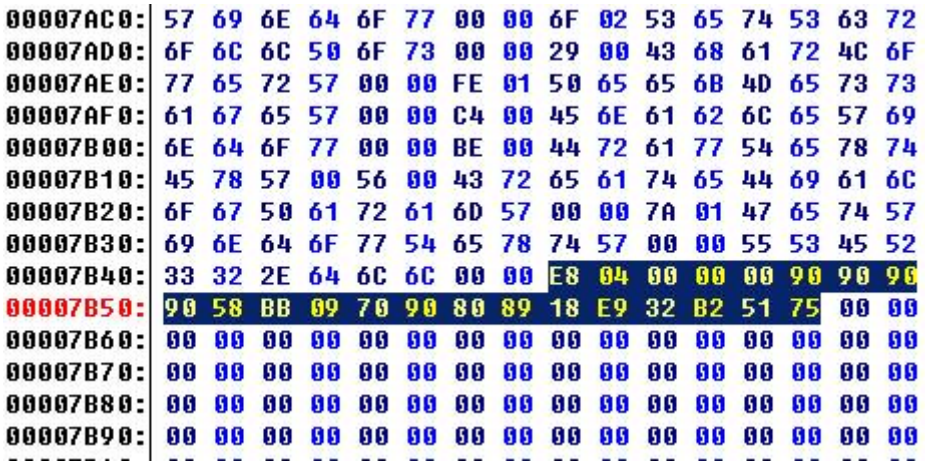
函数入口确实是加载后 IAT 的内容



实验三：EPO 之指令 Patch

在 kernel32.dll 中找到 ReadFile 的 API 函数;将系统中调用该 API 的字节码改为 call 指令，跳转到病毒代码；最后跳转回原 API 函数的入口点；实验发现内存中初始化为 90 90 90 90 的值改变为设置为的 70 90 80 89；通过替换 API 实现病毒是可行的。

整体病毒代码：E8 04 00 00 00 90 90 90 90 58 BB 09 70 90 80 89 18 E9 32 B2 51 75

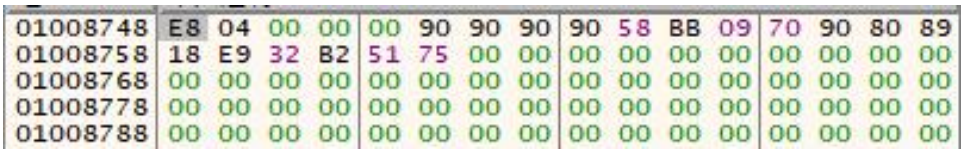


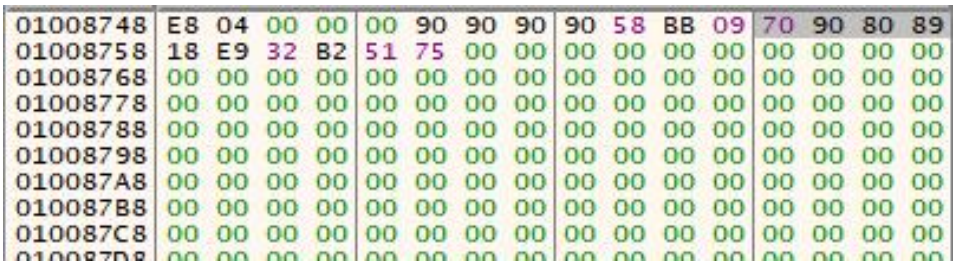
结果：

原来调用的 API 地址：

跳转回原来的 API：

内存中初始化为 90 90 90 90 的值改变为设置的值 70 90 80 89：





七、总结及心得体会：

通过这次实验，我对动态链接库的导出表机制和导入表机制有了很深的了解，明白了动态链接库如何在不同的进程中占据相同的加载基址；通过对导出表过程的模拟和对导入表过程的模拟，让我对进程加载也有了新的认识；最后修改 API 使得病毒起作用，让我感受到病毒的灵活多变。

八、对本实验过程及方法、手段的改进建议：

希望可以有一些高级语言编程的例子，最好有一些实例供学习参考。

报告评分：

指导教师签字：