Lab 09 – All analysis

1120162015 李博

Lab 9-1

1. How can you get this malware to install itself?

通过对程序的静态分析，我发现在 mian 函数中有两个分支可以进入 copyfile 函数所在的子函数，如下图。

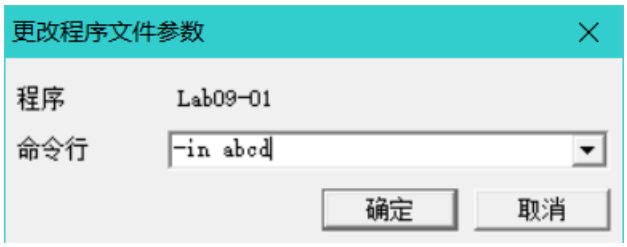
```
.text:00402B63 ; 54:     else if ( argc == 3 )
.text:00402B63                 cmp     [ebp+argc], 3
.text:00402B67                 jnz     short loc_402B9A
.text:00402B69 ; 56:       if ( sub_4025B0(&ServiceName) )          // Lab09-01
.text:00402B69                 push    400h
.text:00402B6E                 lea     ecx, [ebp+ServiceName]
.text:00402B74                 push    ecx             ; char *
.text:00402B75                 call    sub_4025B0
.text:00402B7A                 add     esp, 8
.text:00402B7D                 test    eax, eax
.text:00402B7F                 jz      short loc_402B89
.text:00402B81                 or      eax, 0FFFFFFFFh
.text:00402B84                 jmp     loc_402D78
.text:00402B89 ; ---------------------------------------------------------------------------
.text:00402B89 ; 58:       try_copy(&ServiceName);
.text:00402B89
.text:00402B89 loc_402B89:                             ; CODE XREF: _main+8F↑j
.text:00402B89                 lea     edx, [ebp+ServiceName]
.text:00402B8F                 push    edx             ; lpServiceName
.text:00402B90                 call    try_copy
.text:00402B95                 add     esp, 4
.text:00402B98                 jmp     short loc_402BC2
.text:00402B9A ; ---------------------------------------------------------------------------
.text:00402B9A ; 62:       if ( argc != 4 )
.text:00402B9A
.text:00402B9A loc_402B9A:                             ; CODE XREF: _main+77↑j
.text:00402B9A                 cmp     [ebp+argc], 4
.text:00402B9E                 jnz     short loc_402BBD
.text:00402BA0 ; 64:       try_copy(argv[2]);
.text:00402BA0                 mov     eax, [ebp+argv]
.text:00402BA3                 mov     ecx, [eax+8]
.text:00402BA6                 mov     [ebp+lpServiceName], ecx
.text:00402BAC                 mov     edx, [ebp+lpServiceName]
.text:00402BB2                 push    edx             ; lpServiceName
.text:00402BB3                 call    try_copy
.text:00402BB8                 add     esp, 4
.text:00402BBB                 jmp     short loc_402BC2
```

第一个分支进入条件为

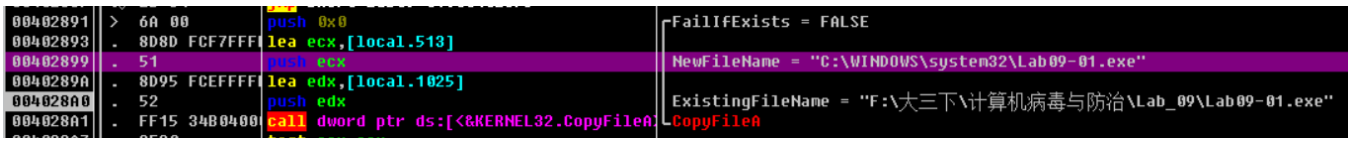
argc==3 并且 argv[argc-1]==” abcd” ,argv[1]==” -in” 。

该分支的作用即安装自身至 C:\Windows\System32。

通过 ollydbg 中**调试->参数**来输入程序的参数



如下图，为 ollydbg 动态调试跟进至复制文件函数的代码处。



2. What are the command-line options for this program? What is the pass-word requirement?

该程序的命令行参数有四种，分别为-in（安装），-re（移除），-c 和-cc。

需要的 password 为"abcd"，需输入在命令行的最后一个参数

即 argv[argc-1]=="abcd"。

3. How can you use OllyDbg to permanently patch this malware, so that it doesn't require the special command-line password?

首先在 IDA 里定位至相应代码段，关键在 **call Delltself**，位于 0x 00402B3A。

```
.text:00402B2A ; 19:     if ( !check_abcd((int)v10) )
.text:00402B2A                 mov     eax, [ebp+var_4]
.text:00402B2D                 push    eax
.text:00402B2E                 call    check_abcd
.text:00402B33                 add     esp, 4
.text:00402B36                 test    eax, eax
.text:00402B38                 jnz     short loc_402B3F
.text:00402B3A ; 20:       DelItself();
.text:00402B3A                 call    DelItself
.text:00402B3F ; ---------------------------------------------------------------
.text:00402B3F ; 21:     if ( _mbscmp((const unsigned __int8 *)argv[1], &in) )
```

由于该程序没有地址随机化，故在 Ollydbg 中可以直接定位至地址 0x0402B3A，如下图，用 nop（0x90）填充 0x0402B3A-0x0402B3E 之间的数据即可。

```
00402B36    85C0        test eax,eax
00402B38    75 05       jnz short Lab09-01.00402B3F
00402B3A    90          nop
00402B3B    90          nop
00402B3C    90          nop
00402B3D    90          nop
00402B3E    90          nop
00402B3F    8B4D 0C     mov ecx,dword ptr ss:[ebp+0xC]
00402B42    8B51 04     mov edx,dword ptr ds:[ecx+0x4]
```

4. What are the host-based indicators of this malware?

a. 该字符串用于定位至系统盘

```
.data:0040C134 aSystemrootSyst db '%SYSTEMROOT%\system32\',0
.data:0040C134                                         ; DATA XREF: try_copy:loc_402632↑o
.data:0040C134                                         ; sub_402900:loc_4029E1↑o
.data:0040C14B                 align 4
```

b. 该字符串用于调用命令行

```
.data:0040C0CC ; CHAR File[]
.data:0040C0CC File            db 'cmd.exe',0          ; DATA XREF: DelItself+D5↑o
.data:0040C0CC                                         ; __popen+21E↑o
```

c. 创建注册表项

```
; 19:   if ( RegCreateKeyExA(HKEY_LOCAL_MACHINE, SubKey, 0, 0, 0, 0xF003Fu, 0, &phkResult, 0) )
push    0               ; lpdwDisposition
lea     ecx, [ebp+phkResult]
push    ecx             ; phkResult
push    0               ; lpSecurityAttributes
push    0F003Fh         ; samDesired
push    0               ; dwOptions
push    0               ; lpClass
push    0               ; Reserved
push    offset SubKey   ; "SOFTWARE\\Microsoft \\XPS"
push    80000002h       ; hKey
call    ds:RegCreateKeyExA
test    eax, eax
jz      short loc_4011B9
```

5. What are the different actions this malware can be instructed to take via the network?

a. 休眠

```
.text:004020BA ; 23:     Sleep(1000 * v3);
.text:004020BA                 mov     ecx, [ebp+var_404]
.text:004020C0                 imul    ecx, 3E8h
.text:004020C6                 push    ecx             ; dwMilliseconds
.text:004020C7                 call    ds:Sleep
.text:004020CD                 jmp     loc_402356
```

b. 接收文件

```
.text:00401A53 loc_401A53:                             ; CODE XREF: sub_4019E0+5B↑j
.text:00401A53                                         ; sub_4019E0+D2↓j
.text:00401A53                 push    0               ; flags
.text:00401A55                 push    200h            ; len
.text:00401A5A                 lea     edx, [ebp+buf]
.text:00401A60                 push    edx             ; buf
.text:00401A61                 mov     eax, [ebp+s]
.text:00401A64                 push    eax             ; s
.text:00401A65                 call    ds:recv
.text:00401A6B                 mov     [ebp+nNumberOfBytesToWrite], eax
.text:00401A6E                 push    0               ; lpOverlapped
.text:00401A70                 push    0               ; lpNumberOfBytesWritten
.text:00401A72                 mov     ecx, [ebp+nNumberOfBytesToWrite]
.text:00401A75                 push    ecx             ; nNumberOfBytesToWrite
.text:00401A76                 lea     edx, [ebp+buf]
.text:00401A7C                 push    edx             ; lpBuffer
.text:00401A7D                 mov     eax, [ebp+hFile]
.text:00401A83                 push    eax             ; hFile
.text:00401A84                 call    ds:WriteFile
.text:00401A8A                 test    eax, eax
.text:00401A8C                 jnz     short loc_401AAE
.text:00401A8E                 lea     ecx, [ebp+s]
.text:00401A91                 push    ecx
.text:00401A92                 call    sub_401740
.text:00401A97                 add     esp, 4
.text:00401A9A                 mov     edx, [ebp+hFile]
.text:00401AA0                 push    edx             ; hObject
.text:00401AA1                 call    ds:CloseHandle
.text:00401AA7                 mov     eax, 1
.text:00401AAC                 jmp     short loc_401AE6
.text:00401AAE ; ---------------------------------------------------------------------------
```

c. 上传文件

```
.text:00401906                 call    ds:ReadFile
.text:0040190C                 test    eax, eax
.text:0040190E                 jnz     short loc_401945
.text:00401910                 call    ds:GetLastError
.text:00401916                 cmp     eax, 26h
.text:00401919                 jz      short loc_40193E
.text:0040191B                 lea     edx, [ebp+s]
.text:0040191E                 push    edx
.text:0040191F                 call    sub_401740
.text:00401924                 add     esp, 4
.text:00401927                 mov     eax, [ebp+hFile]
.text:0040192D                 push    eax             ; hObject
.text:0040192E                 call    ds:CloseHandle
.text:00401934                 mov     eax, 1
.text:00401939                 jmp     loc_4019D8
.text:0040193E ; ---------------------------------------------------------------------------
.text:0040193E
.text:0040193E loc_40193E:                             ; CODE XREF: sub_401870+A9↑j
.text:0040193E                 mov     [ebp+NumberOfBytesRead], 0
.text:00401945
.text:00401945 loc_401945:                             ; CODE XREF: sub_401870+9E↑j
.text:00401945                                         ; sub_401870+136↓j
.text:00401945                 push    0               ; flags
.text:00401947                 mov     ecx, [ebp+NumberOfBytesRead]
.text:0040194A                 push    ecx             ; len
.text:0040194B                 lea     edx, [ebp+Buffer]
.text:00401951                 push    edx             ; buf
.text:00401952                 mov     eax, [ebp+s]
.text:00401955                 push    eax             ; s
.text:00401956                 call    ds:send
.text:0040195C                 mov     [ebp+var_214], eax
.text:00401962                 cmp     [ebp+var_214], 0FFFFFFFFh
.text:00401969                 jnz     short loc_40198B
```

d. 执行 shell

```
.text:004022A6                 mov     dword ptr [ebp+hostshort], eax
.text:004022AC ; 48:     v11 = strtok(0, asc_40C0A4);
.text:004022AC                 push    offset asc_40C0A4 ; "`"
.text:004022B1                 push    0               ; char *
.text:004022B3                 call    _strtok
.text:004022B8                 add     esp, 8
.text:004022BB                 mov     [ebp+var_41C], eax
.text:004022C1 ; 49:     v13 = _popen(v11, aRb);
.text:004022C1                 push    offset aRb      ; "rb"
.text:004022C6                 mov     edx, [ebp+var_41C]
.text:004022CC                 push    edx             ; char *
.text:004022CD                 call    __popen
.text:004022D2                 add     esp, 8
.text:004022D5                 mov     [ebp+var_420], eax
.text:004022DB ; 50:     if ( !v13 )
.text:004022DB                 cmp     [ebp+var_420], 0
.text:004022E2                 jnz     short loc_4022EB
.text:004022E4                 mov     eax, 1
.text:004022E9                 jmp     short loc_402358
```

e. 无操作

```
.text:00402330 ; 61:     strncmp(&v14, aNothing, strlen(aNothing));
.text:00402330
.text:00402330 loc_402330:                             ; CODE XREF: sub_402020+242↑j
.text:00402330                 mov     edi, offset aNothing ; "NOTHING"
.text:00402335                 or      ecx, 0FFFFFFFFh
.text:00402338                 xor     eax, eax
.text:0040233A                 repne scasb
.text:0040233C                 not     ecx
.text:0040233E                 add     ecx, 0FFFFFFFFh
.text:00402341                 push    ecx             ; size_t
.text:00402342                 push    offset aNothing ; "NOTHING"
.text:00402347                 lea     edx, [ebp+var_400]
.text:0040234D                 push    edx             ; char *
.text:0040234E                 call    _strncmp
.text:00402353                 add     esp, 0Ch
```

6. Are there any useful network-based signatures for this malware?

a. 通信的目标网址

```
.data:0040C0E8 aHttpWwwPractic db 'http://www.practicalmalwareanalysis.com',0
.data:0040C0E8                                         ; DATA XREF: try_copy+2D6↑o
.data:0040C110 a80             db '80',0               ; DATA XREF: try_copy+2D1↑o
```

b. 通信方式

```
.data:0040C068 ; char asc_40C068[]
.data:0040C068 asc_40C068      db 0Dh,0Ah              ; DATA XREF: sub_401AF0+1A4↑o
.data:0040C068                 db 0Dh,0Ah,0
.data:0040C06D                 align 10h
.data:0040C070 aHttp10         db ' HTTP/1.0',0Dh,0Ah  ; DATA XREF: sub_401AF0+9F↑o
.data:0040C070                 db 0Dh,0Ah,0
.data:0040C07E                 align 10h
.data:0040C080 aGet            db 'GET ',0             ; DATA XREF: sub_401AF0:loc_401B35↑o
.data:0040C085                 align 4
```

c.

```
.rdata:0040B170 ; char aCommandCom[]
.rdata:0040B170 aCommandCom     db 'command.com',0      ; DATA XREF: __popen+217↑o
.rdata:0040B17C aC              db '/c',0               ; DATA XREF: __popen+1E8↑o
.rdata:0040B17C                                         ; __popen+228↑o
.rdata:0040B17F                 align 10h
```

Lab 9-2

1. What strings do you see statically in the binary?

a. 在 IDA 中的变量上 Y 一下，可改变变量类型，如下图

```
v13 = 49;
v14 = 113;
v15 = 97;
v16 = 122;
v17 = 50;
v18 = 119;
v19 = 115;
v20 = 120;
v21 = 51;
v22 = 101;
v23 = 100;
v24 = 99;
v25 = 0;
v26 = 111;
v27 = 99;
v28 = 108;
v29 = 46;
v30 = 101;
v31 = 120;
v32 = 101;
v33 = 0;
```

➔

```
WSAData[0] = (char *)'zaq1';
WSAData[1] = (char *)'xsw2';
WSAData[2] = (char *)'cde3';
LOBYTE(WSAData[3]) = 0;
WSAData[4] = (char *)'.lco';
WSAData[5] = (char *)'exe';
```

b.

```
.data:00405034 unk_405034      db  46h ; F          ; DATA XREF: _main+A3↑o
.data:00405035                 db   6
.data:00405036                 db  16h
.data:00405037                 db  54h ; T
.data:00405038                 db  42h ; B
.data:00405039                 db   5
.data:0040503A                 db  12h
.data:0040503B                 db  1Bh
.data:0040503C                 db  47h ; G
.data:0040503D                 db  0Ch
.data:0040503E                 db   7
.data:0040503F                 db   2
.data:00405040                 db  5Dh ; ]
.data:00405041                 db  1Ch
.data:00405042                 db   0
.data:00405043                 db  16h
.data:00405044                 db  45h ; E
.data:00405045                 db  16h
.data:00405046                 db   1
.data:00405047                 db  1Dh
.data:00405048                 db  52h ; R
.data:00405049                 db  0Bh
.data:0040504A                 db   5
.data:0040504B                 db  0Fh
.data:0040504C                 db  48h ; H
.data:0040504D                 db   2
.data:0040504E                 db   8
.data:0040504F                 db   9
.data:00405050                 db  1Ch
.data:00405051                 db  14h
.data:00405052                 db  1Ch
.data:00405053                 db  15h
```

2. What happens when you run this binary?

什么也没发生。

3. How can you get this sample to run its malicious payload?

通过 IDA 静态分析可以发现，在 main 函数中，进行了文件名校验的过程，如下。

```
.text:0040122B ; 26:    if ( strcmp((const char *)&WSAData[4], v15) )
.text:0040122B                 mov     eax, [ebp+var_4] ;          cmp - ocl.exe
.text:0040122E                 push    eax             ; char *
.text:0040122F                 lea     ecx, [ebp+WSAData.szDescription+0Ch]
.text:00401235                 push    ecx             ; char *
.text:00401236                 call    _strcmp
.text:0040123B                 add     esp, 8
.text:0040123E                 test    eax, eax
.text:00401240                 jz      short loc_40124C
.text:00401242 ; 27:      return 1;
.text:00401242                 mov     eax, 1
.text:00401247                 jmp     loc_4013D6
```

故将文件名改为 **ocl.exe** 即可。

4. What is happening at 0x00401133?

将一个字符串赋值至栈。

5. What arguments are being passed to subroutine 0x00401089?

参数一为在 0x401133 处放到栈上的字符串 1qaz2wsx3edc

参数二为

```
.data:00405034 unk_405034      db  46h ; F
.data:00405035                 db   6
.data:00405036                 db  16h
.data:00405037                 db  54h ; T
.data:00405038                 db  42h ; B
.data:00405039                 db   5
.data:0040503A                 db  12h
.data:0040503B                 db  1Bh
.data:0040503C                 db  47h ; G
.data:0040503D                 db  0Ch
.data:0040503E                 db   7
.data:0040503F                 db   2
.data:00405040                 db  5Dh ; ]
.data:00405041                 db  1Ch
.data:00405042                 db   0
.data:00405043                 db  16h
.data:00405044                 db  45h ; E
.data:00405045                 db  16h
.data:00405046                 db   1
.data:00405047                 db  1Dh
.data:00405048                 db  52h ; R
.data:00405049                 db  0Bh
.data:0040504A                 db   5
.data:0040504B                 db  0Fh
.data:0040504C                 db  48h ; H
.data:0040504D                 db   2
.data:0040504E                 db   8
.data:0040504F                 db   9
.data:00405050                 db  1Ch
.data:00405051                 db  14h
```

6. What domain name does this malware use?

通过 Ollydbg 动态调试，直接在解密函数之后查看寄存器 eax 值如下。

```
寄存器 (FPU)
EAX 0019FB24 ASCII "www.practicalmalwareanalysis.com"
ECX 00000020
```

7. What encoding routine is being used to obfuscate the domain name?

异或加密

```
.text:004010D4 loc_4010D4:                             ; CODE XREF: sub_401089+92↓j
.text:004010D4                 mov     ecx, [ebp+var_108] ; i ++
.text:004010DA                 add     ecx, 1
.text:004010DD                 mov     [ebp+var_108], ecx
.text:004010E3
.text:004010E3 loc_4010E3:                             ; CODE XREF: sub_401089+49↑j
.text:004010E3                 cmp     [ebp+var_108], 20h ; 比较i的上限0x20
.text:004010EA                 jge     short loc_40111D
.text:004010EC ; 15:     *(&v5 + i) = a1[i % v4] ^ *(_BYTE *)(i + a2);
.text:004010EC                 mov     edx, [ebp+arg_4]
.text:004010EF                 add     edx, [ebp+var_108]
.text:004010F5                 movsx   ecx, byte ptr [edx]
.text:004010F8                 mov     eax, [ebp+var_108]
.text:004010FE                 cdq
.text:004010FF                 idiv    [ebp+var_104]
.text:00401105                 mov     eax, [ebp+arg_0]
.text:00401108                 movsx   edx, byte ptr [eax+edx]
.text:0040110C                 xor     ecx, edx        ; 异或
.text:0040110E                 mov     eax, [ebp+var_108]
.text:00401114                 mov     [ebp+eax+var_100], cl
.text:0040111B                 jmp     short loc_4010D4
```

8. What is the significance of the CreateProcessAcall at 0x0040106E?

创建线程来启动命令行，并且通过设置 showWindow 标志位为 0 来隐藏 shell。

Lab 9-3

1. What DLLs are imported by Lab09-03.exe?

.rdata:0…	0000000B	C	user32.dll
.rdata:0…	0000000D	C	KERNEL32.dll
.rdata:0…	0000000D	C	NETAPI32.dll
.rdata:0…	00000009	C	DLL1.dll
.rdata:0…	00000009	C	DLL2.dll
.data:00…	00000011	C	DLL3GetStructure
.data:00…	0000000A	C	DLL3Print
.data:00…	00000009	C	DLL3.dll

2. What is the base address requested by DLL1.dll, DLL2.dll,and DLL3.dll?

基地址为 0x10000000

3. When you use OllyDbg to debug Lab09-03.exe, what is the assigned based address for: DLL1.dll, DLL2.dll, and DLL3.dll?

当程序运行至下图代码段时，DLL3 被加载。

```
.text:0040103C ; 15:   hModule = LoadLibraryA(LibFileName);
.text:0040103C                 push    offset LibFileName ; "DLL3.dll"
.text:00401041                 call    ds:LoadLibraryA
.text:00401047                 mov     [ebp+hModule], eax
```

在 Ollydbg 中查看内存

00030000	00001000	DLL1		PE 文件头	Imag	R	RWE
00031000	00006000	DLL1	.text	SFX,代码	Imag	R	RWE
00037000	00001000	DLL1	.rdata	数据,输入表	Imag	R	RWE
00038000	00005000	DLL1	.data		Imag	R	RWE
0003D000	00001000	DLL1	.reloc		Imag	R	RWE

10000000	00001000	DLL2		PE 文件头	Imag	R	RWE
10001000	00006000	DLL2	.text	SFX,代码	Imag	R	RWE
10007000	00001000	DLL2	.rdata	数据,输入表	Imag	R	RWE
10008000	00005000	DLL2	.data		Imag	R	RWE
1000D000	00001000	DLL2	.reloc		Imag	R	RWE

005C0000	00001000	DLL3		PE 文件头	Imag	R	RWE
005C1000	00006000	DLL3	.text	SFX,代码	Imag	R	RWE
005C7000	00001000	DLL3	.rdata	数据,输入表	Imag	R	RWE
005C8000	00005000	DLL3	.data		Imag	R	RWE
005CD000	00001000	DLL3	.reloc		Imag	R	RWE
005D0000	00001000				Priv	RW	RW
005E0000	00028000				Priv	RW	RW

4. When Lab09-03.exe calls an import function from DLL1.dll, what does this import function do?

输出字符串 "DLL 1 mystery data %d\n" 和一个变量。

5. When Lab09-03.exe calls WriteFile, what is the filename it writes to?

文件名为 temp.txt

```
.data:10008030 FileName        db 'temp.txt',0        ; DATA XREF: DllMain(x,x,x)+15↑o
.data:10008039                 align 4
```

赋值过程如下

```
push    offset FileName ; "temp.txt"
call    ds:CreateFileA
mov     dword_1000B078, eax
```

```
; Exported entry   2. DLL2ReturnJ


; Attributes: bp-based frame

public DLL2ReturnJ
DLL2ReturnJ proc near
push    ebp
mov     ebp, esp
mov     eax, dword_1000B078
pop     ebp
retn
DLL2ReturnJ endp
```

6. When Lab09-03.exe creates a job using NetScheduleJobAdd, where does it get the data for the second parameter?

来自栈上的局部变量

```
.text:00401078                 add     esp, 4
.text:0040107B                 lea     eax, [ebp+JobId]
.text:0040107E                 push    eax             ; JobId
.text:0040107F                 mov     ecx, [ebp+Buffer]
.text:00401082                 push    ecx             ; Buffer
.text:00401083                 push    0               ; Servername
.text:00401085                 call    NetScheduleJobAdd
```

```
-0000001C Buffer          dd ?
-00000018 hFile           dd ?
-00000014 hModule         dd ?
-00000010 var_10          dd ?
-0000000C NumberOfBytesWritten dd ?
-00000008 var_8           dd ?
-00000004 JobId           dd ?
+00000000  s              db 4 dup(?)
+00000004  r              db 4 dup(?)
+00000008 argc            dd ?
+0000000C argv            dd ?
+00000010 envp            dd ?
```

7. While running or debugging the program, you will see that it prints out three pieces of mystery data. What are the following:DLL 1 mystery data 1, DLL 2 mystery data 2, and DLL 3 mystery data 3?

a. DLL 1 mystery data 1 是 GetCurrentProcessId 的返回值，即进程标识符

```
result = GetCurrentProcessId();
dword_10008030 = result;
```

b. DLL 2 mystery data 2 是 CreateFileA 的返回值，即文件的句柄

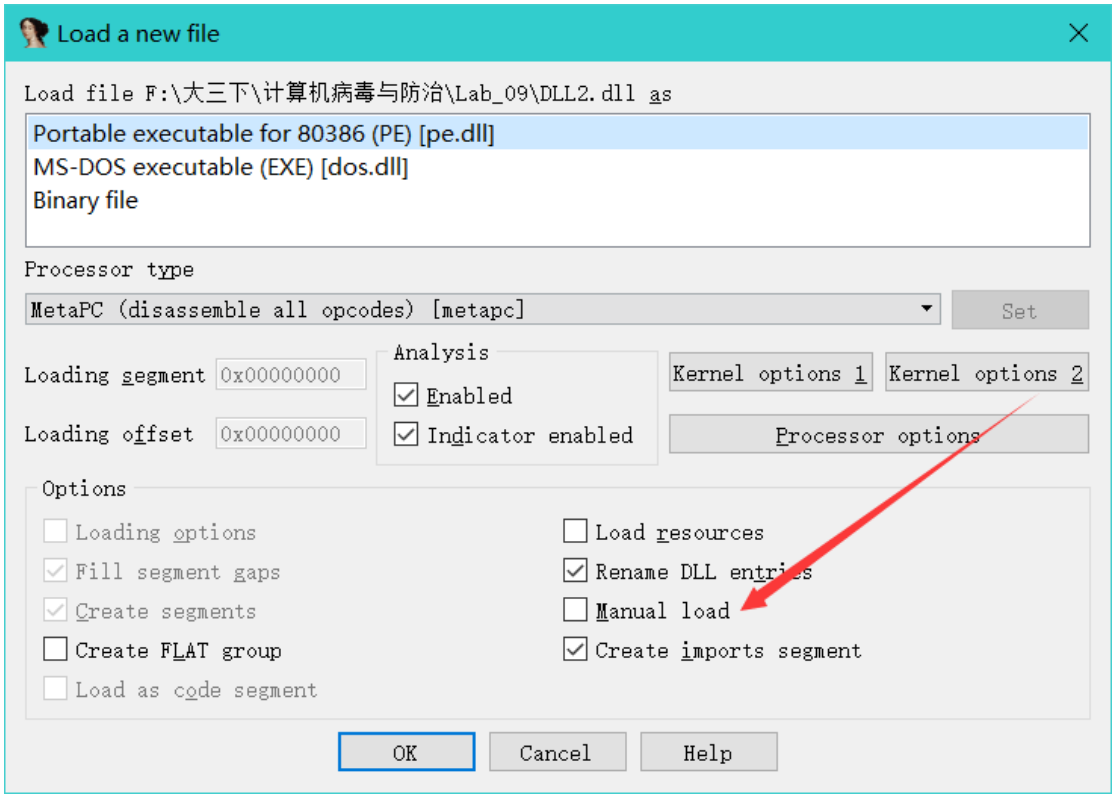
```
call    ds:CreateFileA
; 5:    dword_1000B078 = result;
mov     dword_1000B078, eax
; 6:    LOBYTE(result) = 1;
```

c. DLL 3 mystery data 3 是字符串

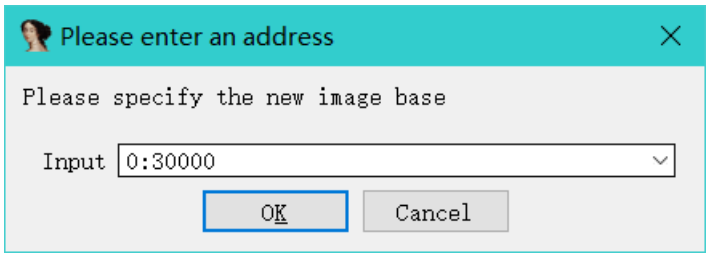
'ping www.malwareanalysisbook.com'的首地址

8. How can you load DLL2.dll into IDA Pro so that it matches the load address used by OllyDbg?

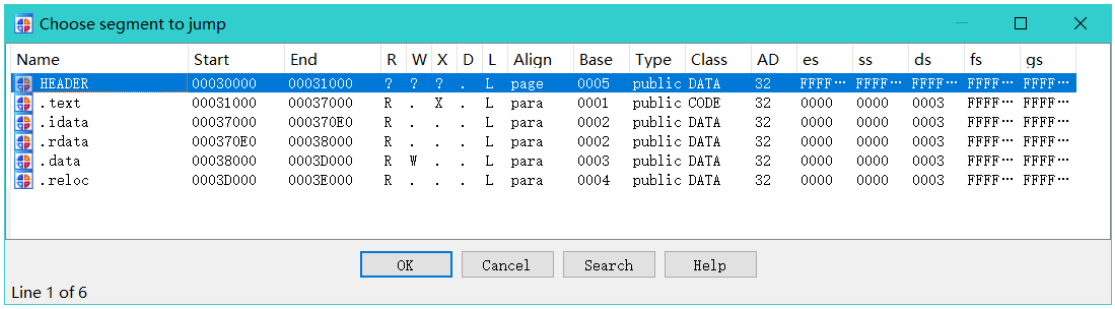
在加载 dll 时，选择 manual load



接着将 olldbg 中的 dll2 的地址 0x30000 输入即可。



结果



Choose segment to jump

Name	Start	End	R	W	X	D	L	Align	Base	Type	Class	AD	es	ss	ds	fs	gs
HEADER	00030000	00031000	?	?	?	.	L	page	0005	public	DATA	32	FFFF···	FFFF···	FFFF···	FFFF···	FFFF···
.text	00031000	00037000	R	.	X	.	L	para	0001	public	CODE	32	0000	0000	0003	FFFF···	FFFF···
.idata	00037000	000370E0	R	.	.	.	L	para	0002	public	DATA	32	0000	0000	0003	FFFF···	FFFF···
.rdata	000370E0	00038000	R	.	.	.	L	para	0002	public	DATA	32	0000	0000	0003	FFFF···	FFFF···
.data	00038000	0003D000	R	W	.	.	L	para	0003	public	DATA	32	0000	0000	0003	FFFF···	FFFF···
.reloc	0003D000	0003E000	R	.	.	.	L	para	0004	public	DATA	32	0000	0000	0003	FFFF···	FFFF···

OK Cancel Search Help

Line 1 of 6