0x01 漏洞复现

DameWare是一款强大的远控软件，用于管理大量主机设备，基于Windows NT，安装完成后，C:/WINDOWS下会生成一个dwsc的文件，其中，当DameWare服务安装完成后，会开启6129端口，此漏洞就存在于6129端口的DWSC.EXE服务中。

DWSC.EXE中有一段处理接收数据的算法逻辑，逻辑中会调用wprintf处理接收数据，但在处理buffer的过程中，没有对buffer的长度进行有效的控制，从而导致了缓冲区溢出的发生，通过构造发送payload，可反控远控主机，下面对此漏洞进行详细分析。

首先Windbg附加服务，执行payload，程序崩溃，捕捉异常点，到达漏洞现场。

```
0:013> g
(abc.424): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=01464545 ebx=0000000a ecx=0000d0d0 edx=00000000 esi=01470000
edi=0146ad66
eip=77d1aa0e esp=0146ad24 ebp=0146ad70 iopl=0         nv up ei pl nz na pe
nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000
efl=00010206
*** ERROR: Symbol file could not be found.  Defaulted to export symbols for
C:\WINDOWS\system32\USER32.dll -
USER32!wvsprintfW+0x3d:
77d1aa0e 668906          mov     word ptr [esi],ax
ds:0023:01470000=????
```

程序中断在user32.dll的wvsprintf函数中，明显现在没有处于主函数领空，通过kb回溯对战调用。

```
0:013> kb
ChildEBP RetAddr  Args to Child
WARNING: Stack unwind information not available. Following frames may be
wrong.
0146ad70 77d1a9ca 0146f94c 0146fb54 0146ad94 USER32!wvsprintfW+0x3d
*** ERROR: Symbol file could not be found.  Defaulted to export symbols for
C:\WINDOWS\dwrcs\DWRCS.EXE -
0146ad84 00423909 0146f94c 0146fb54 0146b0d8 USER32!wsprintfW+0x14
0146ad88 0146f94c 0146fb54 0146b0d8 04780058 DWRCS+0x23909
0146ad8c 0146fb54 0146b0d8 04780058 7c930208 0x146f94c
0146f94c 00200065 00650064 006b0073 006f0074 0x146fb54
0146f950 00650064 006b0073 006f0074 00200070 0x200065
0146fa8c f6faa501 81952e30 7ed2e5cc 1772a487 0x9d4a9af9
0146fa90 81952e30 7ed2e5cc 1772a487 7ac73d4e 0xf6faa501
0146fa94 7ed2e5cc 1772a487 7ac73d4e 830beb71 0x81952e30
0146fa98 1772a487 7ac73d4e 830beb71 70f31ef2 0x7ed2e5cc
0146fa9c 7ac73d4e 830beb71 70f31ef2 3df66aea 0x1772a487
0146faa0 830beb71 70f31ef2 3df66aea 2e8a87ac 0x7ac73d4e
0146faa4 70f31ef2 3df66aea 2e8a87ac 4e39a859 0x830beb71
0146faa8 3df66aea 2e8a87ac 4e39a859 dcdccb48 0x70f31ef2
0146faac 2e8a87ac 4e39a859 dcdccb48 657b2210 0x3df66aea
0146fab0 4e39a859 dcdccb48 657b2210 90903ab2 0x2e8a87ac
0146fab4 dcdccb48 657b2210 90903ab2 90909090 0x4e39a859
0146fab8 657b2210 90903ab2 90909090 90909090 0xdcdccb48
0146fabc 90903ab2 90909090 90909090 90909090 0x657b2210
0146fac0 90909090 90909090 90909090 90909090 0x90903ab2
0146fac4 90909090 90909090 90909090 90909090 0x90909090
0146fac8 90909090 90909090 90909090 90909090 0x90909090
0146facc 90909090 90909090 90909090 90909090 0x90909090
```

可以看到00423909地址位置处于主函数领空，除此之外往后的堆栈调用已经被畸形字符串覆盖，这些畸形字符串正是发送的payload部分，0146fac4的0x90909090之上的位置正是shellcode。

这里要稍微解释一下，因为原PoC所对应的操作版本不同，偏移也有所不同，这样触发异常的位置也不同，因此没有执行shellcode。

那么我就从00423909这个主程序领空的指令入手分析整个漏洞的成因。

0x02 漏洞分析

在回溯过程中，我观察了程序上下文，发现了在进入00423909函数之前，并没有调用其他函数的痕迹，在00423909指令所处函数中，我找到了一个函数块loc_42385B，下面来看一下这个块。

```
.text:0042385B loc_42385B:                             ; CODE XREF:
sub_423570+12Fj
.text:0042385B                                         ; DATA XREF:
.text:off_423F74_x0019_o
.text:0042385B                 push    430h            ; jumptable 0042369F
case 40004
.text:00423860                 lea     ecx, [esp+51E0h+var_4EA4]
.text:00423867                 push    ecx             ; buf
.text:00423868                 mov     ecx, esi
.text:0042386A                 mov     [esp+51E4h+var_4EA8], eax
.text:00423871                 call    sub_40E0B0
.text:00423876                 test    eax, eax
.text:00423878                 jz      loc_423F0D
.text:0042387E                 cmp     dword_4AB6B0, ebx
.text:00423884                 jz      loc_423F1D
.text:0042388A                 cmp     ebp, ebx
.text:0042388C                 jz      loc_423ED2      ; jumptable 0042369F
default case
.text:00423892                 push    206h            ; size_t
.text:00423897                 lea     eax, [esp+51E0h+var_626]
.text:0042389E                 xor     edx, edx
.text:004238A0                 push    ebx             ; int
.text:004238A1                 push    eax             ; void *
.text:004238A2                 mov     [esp+51E8h+String], dx
.text:004238AA                 call    _memset
.text:004238AF                 push    206h            ; size_t
.text:004238B4                 lea     edx, [esp+51ECh+var_41E]
.text:004238BB                 xor     ecx, ecx
.text:004238BD                 push    ebx             ; int
.text:004238BE                 push    edx             ; void *
.text:004238BF                 mov     [esp+51F4h+var_420], cx
.text:004238C7                 call    _memset
.text:004238CC                 mov     ecx, hModule
.text:004238D2                 add     esp, 18h
.text:004238D5                 push    104h
.text:004238DA                 lea     eax, [esp+51E0h+var_420]
.text:004238E1                 push    eax
.text:004238E2                 push    11h
.text:004238E4                 push    ecx
.text:004238E5                 call    off_4A2A24
```

首先值得关注的是004238AA和004238C7两个地址调用到的memset函数，在这两个地址下断点，重新附加Windbg进行跟踪。

```
0:013> bp 4238aa
*** ERROR: Symbol file could not be found.  Defaulted to export symbols for
C:\WINDOWS\dwrcs\DWRCS.EXE -
0:013> bl
 0 e 004238aa     0001 (0001)  0:**** DWRCS+0x238aa
0:013> g
Breakpoint 0 hit
eax=0176f94e ebx=00000000 ecx=00161c88 edx=00000000 esi=01c73388
edi=009abdb0
eip=004238aa esp=0176ad8c ebp=009a4d90 iopl=0         nv up ei pl zr na pe
nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000
efl=00000246
DWRCS+0x238aa:
004238aa e801ea0300      call    DWRCS+0x622b0 (004622b0)
```

我先对memset函数进行一个简要说明，一共三个参数，第一个参数为指定的缓冲区，第二个参数为覆盖内容，第三个参数为覆盖长度，下面来看一下到达这处call调用时候的参数情况。

```
0:013> dd esp
0176ad8c  0176f94e 00000000 00000206
```

这是一处初始化的memeset操作，会对0176f94e开辟的缓冲区进行初始化，覆盖值为0，长度为206h，接下来到达第二处call调用。

```
0:013> p
eax=0176f94e ebx=00000000 ecx=00000000 edx=0176fb56 esi=01c73388
edi=009abdb0
eip=004238be esp=0176ad84 ebp=009a4d90 iopl=0         nv up ei pl zr na pe
nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000
efl=00000246
DWRCS+0x238be:
004238be 52              push    edx
0:013> p
eax=0176f94e ebx=00000000 ecx=00000000 edx=0176fb56 esi=01c73388
edi=009abdb0
eip=004238bf esp=0176ad80 ebp=009a4d90 iopl=0         nv up ei pl zr na pe
nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000
efl=00000246
DWRCS+0x238bf:
004238bf 66898c24d44d0000 mov     word ptr [esp+4DD4h],cx
ss:0023:0176fb54=0000
0:013> p
eax=0176f94e ebx=00000000 ecx=00000000 edx=0176fb56 esi=01c73388
edi=009abdb0
eip=004238c7 esp=0176ad80 ebp=009a4d90 iopl=0         nv up ei pl zr na pe
nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000
efl=00000246
DWRCS+0x238c7:
004238c7 e8e4e90300      call    DWRCS+0x622b0 (004622b0)
```

这也是一处memset函数调用，查看一下参数情况。

```
0:013> dd esp
0176ad80  0176fb56 00000000 00000206
```

这次初始化的是0176ad80缓冲区，初始化结束后，会调用一个off_4A2A24函数，这个函数由于是定义在程序中，所以这里只是一处偏移，但不影响分析，来看一下两个memset后的函数内容。

```
.text:004238D5                push    104h
.text:004238DA                lea     eax, [esp+51E0h+var_420]
.text:004238E1                push    eax
.text:004238E2                push    11h
.text:004238E4                push    ecx
.text:004238E5                call    off_4A2A24
```

其实这里调用到的是LoadStringW函数，跟入off_4A2A24函数中，通过IDA可以看到这里的伪代码。

```
int sub_4599A1()
{
  sub_45A64A("user32.dll", (int)"LoadStringW", &off_4A2A24, dword_4B0AB4,
(LONG)sub_45AAC9);
  return ((int (*)(void))off_4A2A24)();
}
```

关于LoadStringW的函数类型如下，这个函数主要是从资源中根据资源标记，读取对应的字符串。

```
WINUSERAPI int WINAPI LoadStringA(
__in_opt HINSTANCE hInstance,
__in UINT uID,
__out_ecount(cchBufferMax) LPSTR lpBuffer,
__in int nBufferMax);
参数1: hInstance是应用程序实例句柄。
参数2: uID是资源中的字符串编号。
参数3: lpBuffer是接收从资源里拷贝字符串出来的缓冲区。
参数4: nBufferMax是指明缓冲的大小。
```

函数执行完毕后，可查看一下读取的内容。

```
0:013> dc 0176fb54 l100
0176fb54  00680054 00200065 00650064 006b0073  T.h.e. .d.e.s.k.
0176fb64  006f0074 00200070 00730075 00720065  t.o.p. .u.s.e.r.
0176fb74  00640020 00730069 006f0063 006e006e   .d.i.s.c.o.n.n.
0176fb84  00630065 00650074 00200064 00680074  e.c.t.e.d. .t.h.
0176fb94  00200065 00650073 00730073 006f0069  e. .s.e.s.s.i.o.
0176fba4  0020006e 00690076 00200061 00680074  n. .v.i.a. .t.h.
0176fbb4  00200065 0052004d 00200043 00720054  e. .M.R.C. .T.r.
0176fbc4  00790061 006d0020 006e0065 000d0075  a.y. .m.e.n.u...
0176fbd4  0055000a 00650073 00490072 003a0044  ..U.s.e.r.I.D.:.
0176fbe4  00250020 00000073 00000000 00000000   .%.s...........
```

读取的是The Desk top user disconnected the session via the MRC Tray menu UserID %s 这个字符串非常重要，接下来继续单步执行，到达漏洞触发前的位置。

```
0:013> p
eax=0000004b ebx=00000000 ecx=00000000 edx=0176b0d8 esi=01c73388
edi=009abdb0
eip=004238f2 esp=0176ad98 ebp=009a4d90 iopl=0         nv up ei pl zr na pe
nc
```

```
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000
efl=00000246
DWRCS+0x238f2:
004238f2 52              push    edx
0:013> p
eax=0000004b ebx=00000000 ecx=00000000 edx=0176b0d8 esi=01c73388
edi=009abdb0
eip=004238f3 esp=0176ad94 ebp=009a4d90 iopl=0         nv up ei pl zr na pe
nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000
efl=00000246
DWRCS+0x238f3:
004238f3 8d8424c04d0000  lea     eax,[esp+4DC0h]
0:013> p
eax=0176fb54 ebx=00000000 ecx=00000000 edx=0176b0d8 esi=01c73388
edi=009abdb0
eip=004238fa esp=0176ad94 ebp=009a4d90 iopl=0         nv up ei pl zr na pe
nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000
efl=00000246
DWRCS+0x238fa:
004238fa 50              push    eax
0:013> p
eax=0176fb54 ebx=00000000 ecx=00000000 edx=0176b0d8 esi=01c73388
edi=009abdb0
eip=004238fb esp=0176ad90 ebp=009a4d90 iopl=0         nv up ei pl zr na pe
nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000
efl=00000246
DWRCS+0x238fb:
004238fb 8d8c24bc4b0000  lea     ecx,[esp+4BBCh]
0:013> p
eax=0176fb54 ebx=00000000 ecx=0176f94c edx=0176b0d8 esi=01c73388
edi=009abdb0
eip=00423902 esp=0176ad90 ebp=009a4d90 iopl=0         nv up ei pl zr na pe
nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000
efl=00000246
DWRCS+0x23902:
00423902 51              push    ecx
0:013> p
eax=0176fb54 ebx=00000000 ecx=0176f94c edx=0176b0d8 esi=01c73388
edi=009abdb0
eip=00423903 esp=0176ad8c ebp=009a4d90 iopl=0         nv up ei pl zr na pe
nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000
efl=00000246
DWRCS+0x23903:
00423903 ff15542a4a00    call    dword ptr [DWRCS!CoInitializeEx+0x28ea4
(004a2a54)] ds:0023:004a2a54={USER32!wsprintfW (77d1a9b6)}
```

到达00423903的位置，可以看到这里调用了wsprintfW函数，查看一下参数情况。

```
0:013> dd esp
0176ad8c  0176f94c 0176fb54 0176b0d8 1154f6eb
0176ad9c  7c930208 009ad0d0 0176ffa8 009ad0d0
0176adac  01c73388 00009c44 00000000 00000000
0176adbc  00000000 00000000 00000000 00000000
0176adcc  00000000 00000000 00000000 00000000
0176addc  00000000 00000000 00000000 00000000
0176adec  00000000 00000000 00000000 00000000
0176adfc  00000000 00000000 00000000 00000000
```

根据wsprintfW的函数特性，0176f94c是待拷贝的缓冲区，之前初始化提到，0176fb54是要拷贝的内容，这个地址的值之前提到过，而那个字符串里存在一个%s，也就是说涉及到第三个值0176b0d8缓冲区的值，那么0176b0d8缓冲区中的值内容是多少呢。

```
0:013> dc 0176b0d8 l1000
0176b0d8  90909090 90909090 4fa801ba d9cad99e  ..........O....
0176b0e8  5ef42474 31b1c929 03135631 ee831356  t$.^)..11V..V...
0176b0f8  62ba4afd 9b450815 7ecf6de5 0babadd4  .J.b..E..m.~....
0176b108  5ebf1e46 4aedd56a 7c399bf9 b31c114a  F..^j..J..9|J...
0176b118  d25c0a4b 34b151cf 35c499ee 6725c737  K.\..Q.4...57.%g
0176b128  989883e0 1220de85 c720cfd5 5601eead  ..... ... ....V
0176b138  5881a8a6 428bc16b f842ec68 28549a5a  ...Xk..Bh.B.Z.T(
0176b148  15fa6393 5102961c ab71499a 6882f4d9  .c.....Q.Iq....h
0176b158  6b0622a0 57b0a002 132665b3 7b2cc2bf  ."k...W.e&...,{
0176b168  f7e1d5a3 d8045edf fc232456 a54afe33  .....^..V$#.3.J.
0176b178  b5725199 bdd60d42 9c6b5a6e 9af99de4  .Qr.B...nZk.....
0176b188  a5019d4a 2e30f6fa e5cc8195 a4877ed2  J.....0......~..
0176b198  3d4e1772 eb717ac7 1ef2830b 6aea70f3  r.N=.zq......p.j
0176b1a8  87ac3df6 a8592e8a cb484e39 2210dcdc  .=....Y.9NH...."
0176b1b8  3ab2657b 90909090 90909090 90909090  {e.:............
0176b1c8  90909090 90909090 90909090 90909090  ................
0176b1d8  90909090 90909090 90909090 90909090  ................
0176b1e8  90909090 90909090 90909090 90909090  ................
0176b1f8  90909090 90909090 90909090 90909090  ................
0176b208  90909090 90909090 90909090 90909090  ................
0176b218  90909090 90909090 90909090 90909090  ................
0176b228  90909090 90909090 90909090 90909090  ................
0176b238  90909090 90909090 90909090 90909090  ................
0176b248  90909090 90909090 90909090 90909090  ................
0176b258  90909090 90909090 90909090 90909090  ................
0176b268  90909090 90909090 90909090 90909090  ................
0176b278  90909090 90909090 90909090 90909090  ................
0176b288  90909090 90909090 90909090 90909090  ................
```

正是畸形字符串，紧接着执行到漏洞位置。

```
0:013> p
eax=0176fb54 ebx=00000000 ecx=0176f94c edx=0176b0d8 esi=01c73388
edi=009abdb0
eip=00423903 esp=0176ad8c ebp=009a4d90 iopl=0         nv up ei pl zr na pe
nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000
efl=00000246
DWRCS+0x23903:
00423903 ff15542a4a00    call    dword ptr [DWRCS!CoInitializeEx+0x28ea4
(004a2a54)] ds:0023:004a2a54={USER32!wsprintfW (77d1a9b6)}
0:013> p
(4a8.570): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=01764545 ebx=0000000a ecx=0000d0d0 edx=00000000 esi=01770000
edi=0176ad66
eip=77d1aa0e esp=0176ad24 ebp=0176ad70 iopl=0         nv up ei pl nz na pe
nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000
efl=00010206
USER32!wvsprintfW+0x3d:
77d1aa0e 668906          mov     word ptr [esi],ax
ds:0023:01770000=????
```

查看一下待拷贝的区域。

```
0:013> dc 0176f94c l100
0176f94c  00680054 00200065 00650064 006b0073  T.h.e. .d.e.s.k.
0176f95c  006f0074 00200070 00730075 00720065  t.o.p. .u.s.e.r.
0176f96c  00640020 00730069 006f0063 006e006e   .d.i.s.c.o.n.n.
0176f97c  00630065 00650074 00200064 00680074  e.c.t.e.d. .t.h.
0176f98c  00200065 00650073 00730073 006f0069  e. .s.e.s.s.i.o.
0176f99c  0020006e 00690076 00200061 00680074  n. .v.i.a. .t.h.
0176f9ac  00200065 0052004d 00200043 00720054  e. .M.R.C. .T.r.
0176f9bc  00790061 006d0020 006e0065 000d0075  a.y. .m.e.n.u...
0176f9cc  0055000a 00650073 00490072 003a0044  ..U.s.e.r.I.D.:.
0176f9dc  90900020 90909090 01ba9090 d99e4fa8   ............O..
0176f9ec  2474d9ca c9295ef4 563131b1 13560313  ..t$.^)..11V..V.
0176f9fc  4afdee83 081562ba 6de59b45 add47ecf  ...J.b..E..m.~..
0176fa0c  1e460bab d56a5ebf 9bf94aed 114a7c39  ..F..^j..J..9|J.
0176fa1c  0a4bb31c 51cfd25c 99ee34b1 c73735c4  ..K.\..Q.4...57.
0176fa2c  83e06725 de859898 cfd51220 eeadc720  %g..... ... ...
0176fa3c  a8a65601 c16b5881 ec68428b 9a5af842  .V...Xk..Bh.B.Z.
0176fa4c  63932854 961c15fa 499a5102 f4d9ab71  T(.c.....Q.Iq...
0176fa5c  22a06882 a0026b06 65b357b0 c2bf1326  .h."..k...W.e&...
0176fa6c  d5a37b2c 5edff7e1 2456d804 fe33fc23  ,{.....^..V$#.3.
0176fa7c  5199a54a 0d42b572 5a6ebdd6 9de49c6b  J..Qr.B...nZk...
0176fa8c  9d4a9af9 f6faa501 81952e30 7ed2e5cc  ..J.....0......~
0176fa9c  1772a487 7ac73d4e 830beb71 70f31ef2  ..r.N=.zq......p
0176faac  3df66aea 2e8a87ac 4e39a859 dcdccb48  .j.=....Y.9NH...
0176fabc  657b2210 90903ab2 90909090 90909090  ."{e.:..........
0176facc  90909090 90909090 90909090 90909090  ................
0176fadc  90909090 90909090 90909090 90909090  ................
0176faec  90909090 90909090 90909090 90909090  ................
0176fafc  90909090 90909090 90909090 90909090  ................
0176fb0c  90909090 90909090 90909090 90909090  ................
0176fb1c  90909090 90909090 90909090 90909090  ................
```

原来的%s已经被畸形字符串覆盖了，通过IDA来看一下这一段的函数逻辑。

```
          if ( v1 )
          {
            String = 0;
            memset(&v70, 0, 0x206u);
            v71 = 0;
            memset(&v72, 0, 0x206u);
            ((void (__stdcall *)(HMODULE, signed int, __int16 *, signed
int))off_4A2A24)(hModule, 17, &v71, 260);
            ((void (__cdecl *)(WCHAR *, __int16 *, char *))off_4A2A54)
(&String, &v71, &v63);


int sub_4599A1()
{
  sub_45A64A("user32.dll", (int)"LoadStringW", &off_4A2A24, dword_4B0AB4,
(LONG)sub_45AAC9);
  return ((int (*)(void))off_4A2A24)();
}
```

由此可见，v70为大小为206h的缓冲区，v72也是大小为206的缓冲区，而在LoadString中，会读取下面要用到的v71，最后在wsprintf中，由于对v63没有进行严格的控制，导致v63拷贝至String时，发生了缓冲区溢出，从而可以远程执行代码，反控远控主机。