A Debugging Primer with CVE-2019–0708

Bruce Lee Follow
May 30 · 10 min read

By: @straight_blast ; straightblast426@gmail.com

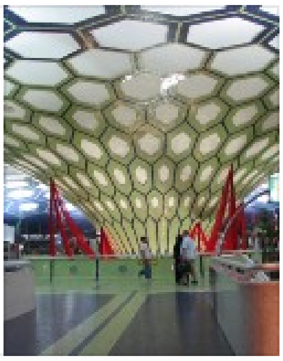
The purpose of this post is to share how one would use a debugger to identify the relevant code path that can trigger the crash. I hope this post will be educational to people that are excited to learning how to use debugger for vulnerability analysis.

This post will not visit details on RDP communication basics and MS_T120. Interested readers should refer to the following blogs that sum up the need to know basis:

CVE-2019-0708: A Comprehensive Analysis of a Remote Desktop Services Vulnerability

In the May 2019 patch cycle, Microsoft released a patch for a remote code execution bug in their…

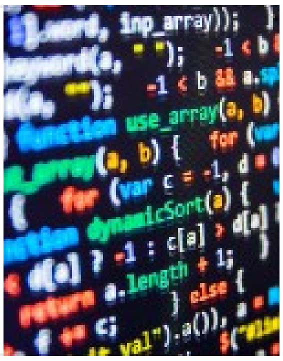
www.zerodayinitiative.com



RDP Stands for "Really DO Patch!" - Understanding the Wormable RDP Vulnerabili…

During Microsoft's May Patch Tuesday cycle, a security advisory was released for a vulnerability…

securingtomorrow.mcafee.com



Furthermore, no PoC code will be provided in this post, as the purpose is to show vulnerability analysis with a debugger.

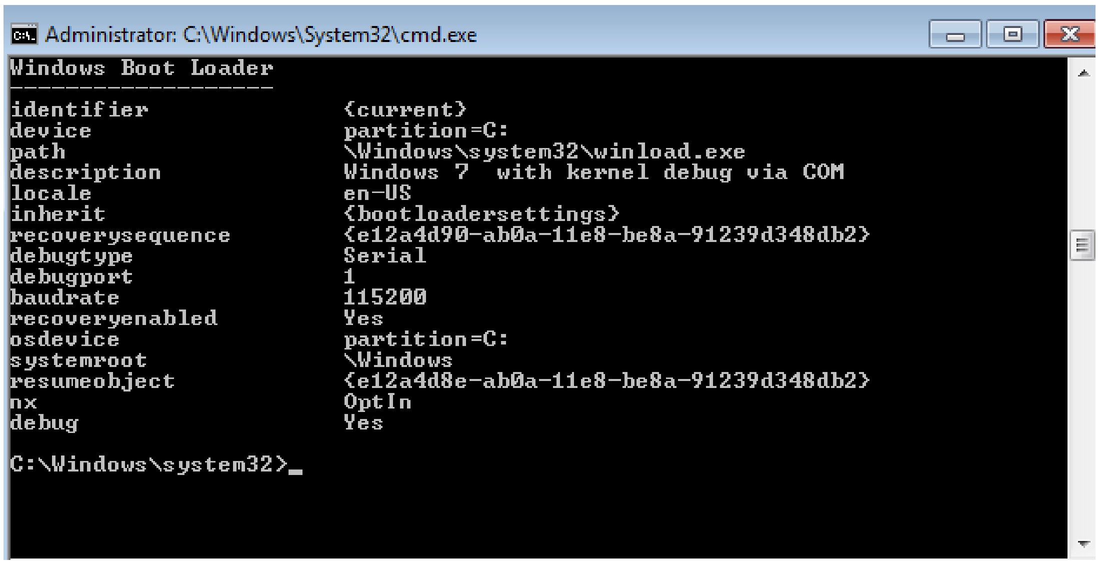
The target machine (debuggee) will be a Windows 7 x64 and the debugger machine will be a Windows 10 x64. Both the debugger and debuggee will run within VirtualBox.

Setting up the kernel debugging environment with VirtualBox

1. On the target machine, run cmd.exe with administrative privilege. Use the bcdedit command to enable kernel debugging.

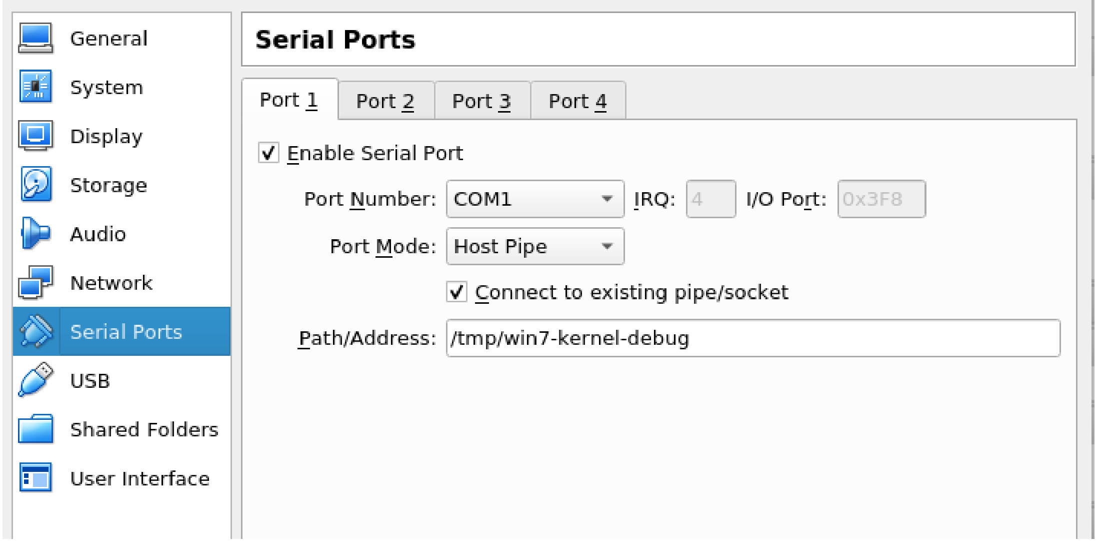
```
bcdedit /set {current} debug yes
bcdedit /set {current} debugtype serial
bcdedit /set {current} debugport 1
bcdedit /set {current} baudrate 115200
bcdedit /set {current} description "Windows 7 with kernel
debug via COM"
```

When you type bcdedit again, something similar to the following screenshot should display:



```
Administrator: C:\Windows\System32\cmd.exe
Windows Boot Loader
-------------------
identifier              {current}
device                  partition=C:
path                    \Windows\system32\winload.exe
description             Windows 7  with kernel debug via COM
locale                  en-US
inherit                 {bootloadersettings}
recoverysequence        {e12a4d90-ab0a-11e8-be8a-91239d348db2}
debugtype               Serial
debugport               1
baudrate                115200
recoveryenabled         Yes
osdevice                partition=C:
systemroot              \Windows
resumeobject            {e12a4d8e-ab0a-11e8-be8a-91239d348db2}
nx                      OptIn
debug                   Yes

C:\Windows\system32>_
```

2. Shutdown the target machine (debuggee) and right click on the target image in the VirtualBox Manager. Select “Settings” and then “Serial Ports”. Copy the settings as illustrated in the following image and click “OK”:



☑ Enable Serial Port

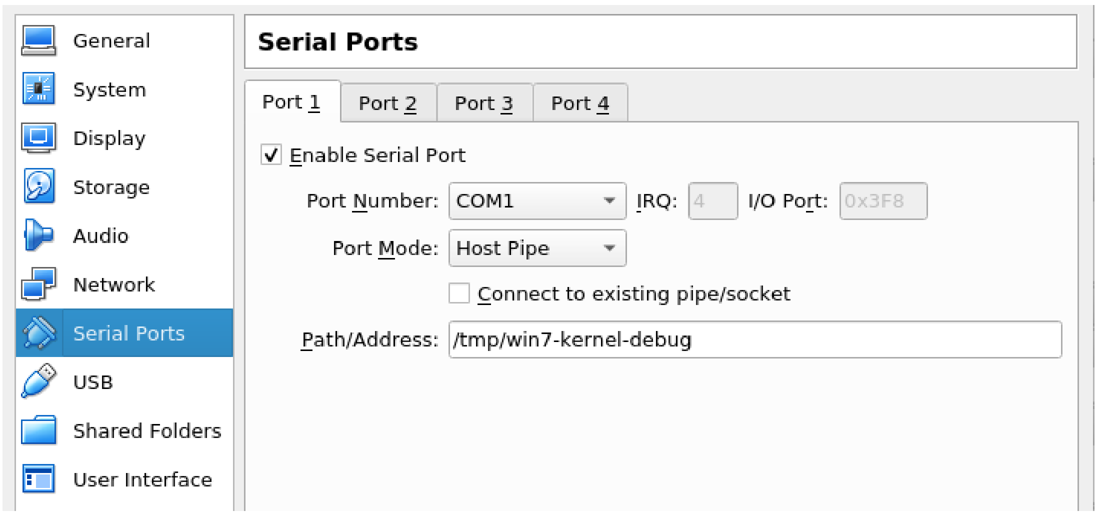
Port Number: COM1 IRQ: 4 I/O Port: 0x3F8

Port Mode: Host Pipe

☑ Connect to existing pipe/socket

Path/Address: /tmp/win7-kernel-debug

3. Right click on the image that will host the debugger, and go to the "Serial Ports" setting and copy the settings as shown and click "OK":



☑ Enable Serial Port

Port Number: COM1 IRQ: 4 I/O Port: 0x3F8

Port Mode: Host Pipe

☐ Connect to existing pipe/socket

Path/Address: /tmp/win7-kernel-debug

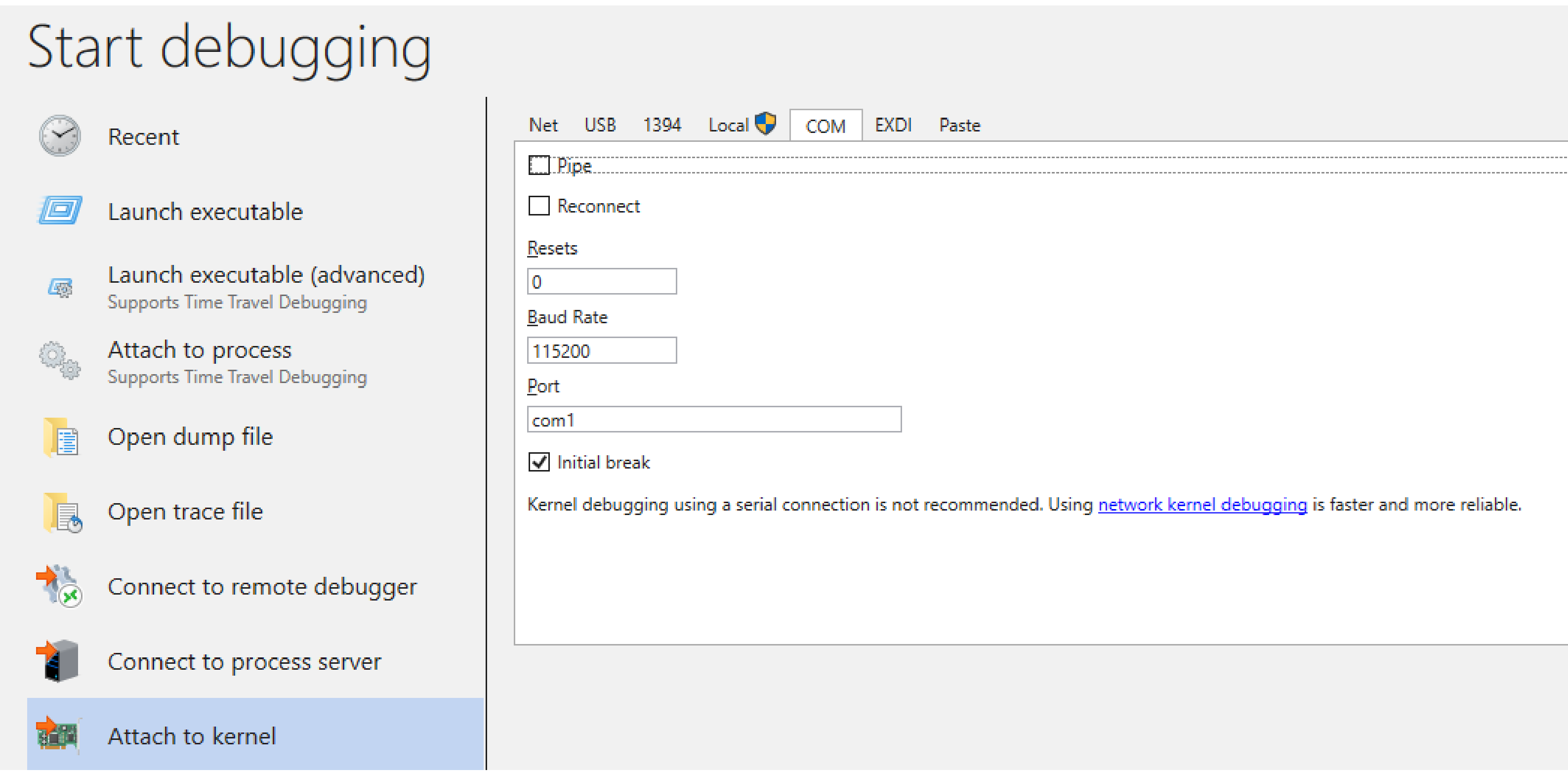
4. Keep the debuggee VM shutdown, and boot up the debugger VM. On the debugger VM, download and install WinDBG. I will be using the WinDBG Preview edition.

Download Debugging Tools for Windows - WinDbg - Windows drivers

This page provides downloads for the Windows Debugging tools, such as WinDbg.

docs.microsoft.com

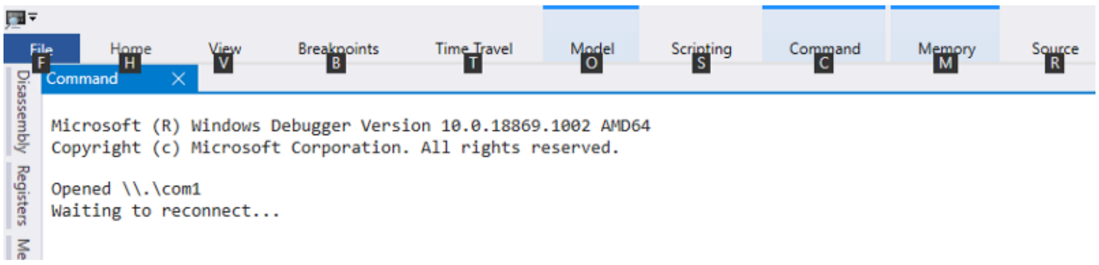
5. Once the debugger is installed, select "Attach to kernel", set the "Baud Rate" to "115200" and "Port" to "com1". Click on the "initial break" as well.


Start debugging
Recent
Launch executable
Launch executable (advanced)
Supports Time Travel Debugging
Attach to process
Supports Time Travel Debugging
Open dump file
Open trace file
Connect to remote debugger
Connect to process server
Attach to kernel
Net
USB
1394
Local
COM
EXDI
Paste

☐ Pipe
☐ Reconnect
Resets: 0
Baud Rate: 115200
Port: com1
☑ Initial break

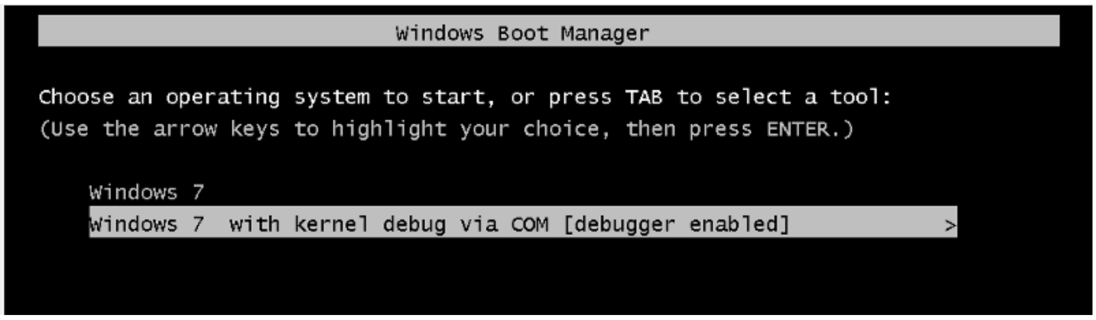
Kernel debugging using a serial connection is not recommended. Using network kernel debugging is faster and more reliable.

Click "OK" and the debugger is now ready to attach to the debuggee.



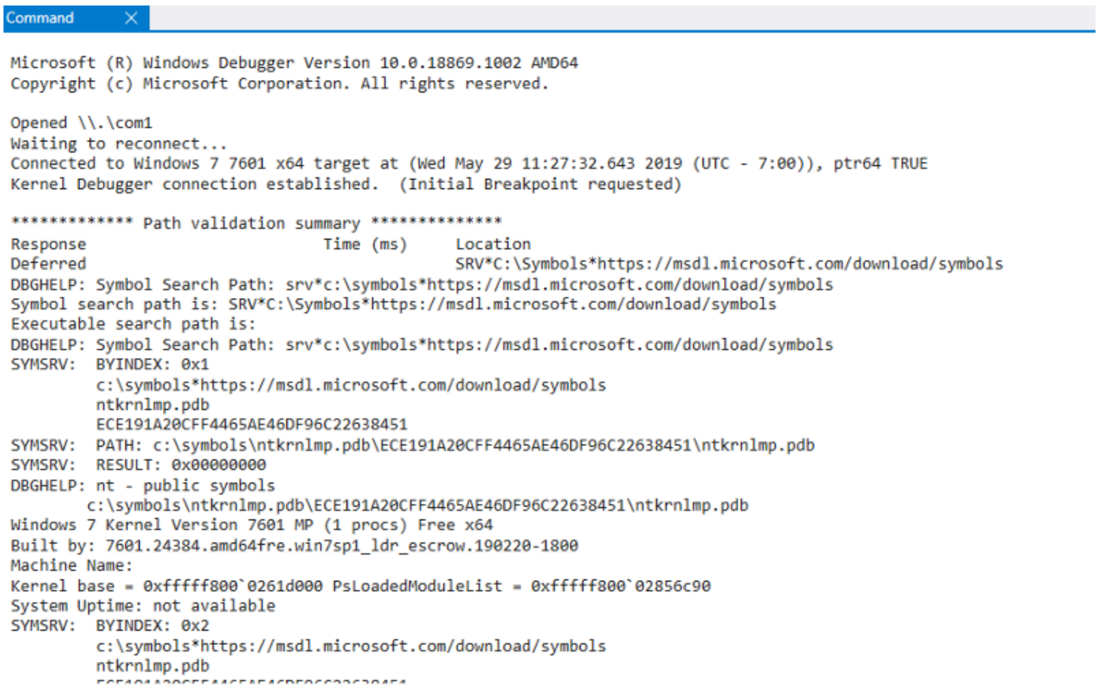
```
Microsoft (R) Windows Debugger Version 10.0.18869.1002 AMD64
Copyright (c) Microsoft Corporation. All rights reserved.

Opened \\.\com1
Waiting to reconnect...
```

6. Fire up the target "debuggee" machine, and the following prompt will be displayed. Select the one with "debugger enabled" and proceed.



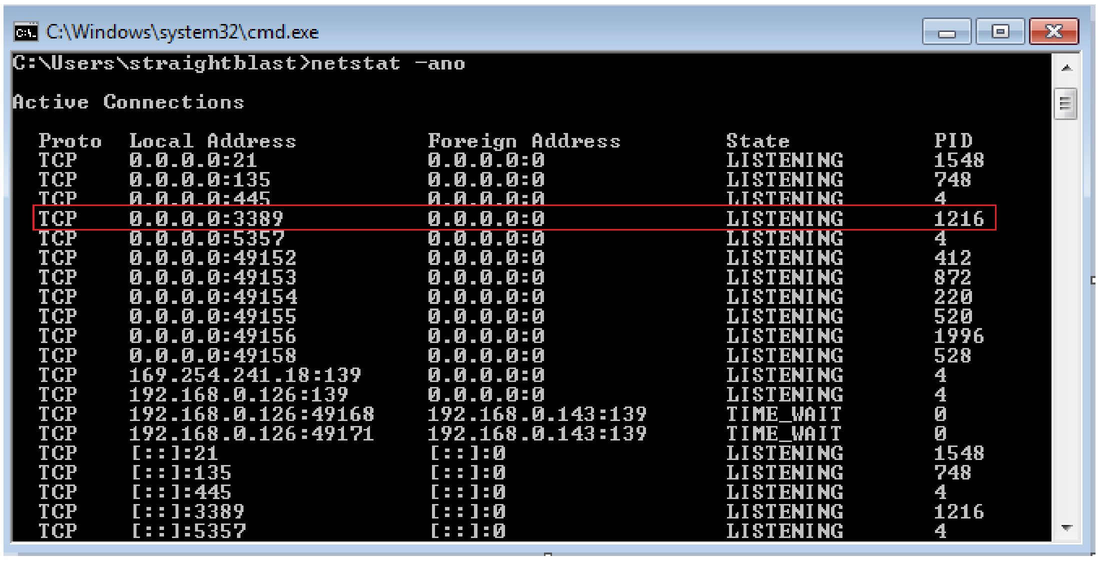
```
                    Windows Boot Manager

Choose an operating system to start, or press TAB to select a tool:
(Use the arrow keys to highlight your choice, then press ENTER.)

    Windows 7
    Windows 7  with kernel debug via COM [debugger enabled]          >
```

On the debugger end, the WinDBG will have established a connection with the debuggee. It is going to require a few manual enter of "g" into the "debugger command prompt" to have the debuggee completely loaded up. Also, because the debugging action is handled through "com", the initial start up will take a bit of time.



```
Microsoft (R) Windows Debugger Version 10.0.18869.1002 AMD64
Copyright (c) Microsoft Corporation. All rights reserved.

Opened \\.\com1
Waiting to reconnect...
Connected to Windows 7 7601 x64 target at (Wed May 29 11:27:32.643 2019 (UTC - 7:00)), ptr64 TRUE
Kernel Debugger connection established.  (Initial Breakpoint requested)

************* Path validation summary **************
Response                         Time (ms)     Location
Deferred                                       SRV*C:\Symbols*https://msdl.microsoft.com/download/symbols
DBGHELP: Symbol Search Path: srv*c:\symbols*https://msdl.microsoft.com/download/symbols
Symbol search path is: SRV*C:\Symbols*https://msdl.microsoft.com/download/symbols
Executable search path is:
DBGHELP: Symbol Search Path: srv*c:\symbols*https://msdl.microsoft.com/download/symbols
SYMSRV:  BYINDEX: 0x1
         c:\symbols*https://msdl.microsoft.com/download/symbols
         ntkrnlmp.pdb
         ECE191A20CFF4465AE46DF96C22638451
SYMSRV:  PATH: c:\symbols\ntkrnlmp.pdb\ECE191A20CFF4465AE46DF96C22638451\ntkrnlmp.pdb
SYMSRV:  RESULT: 0x00000000
DBGHELP: nt - public symbols
        c:\symbols\ntkrnlmp.pdb\ECE191A20CFF4465AE46DF96C22638451\ntkrnlmp.pdb
Windows 7 Kernel Version 7601 MP (1 procs) Free x64
Built by: 7601.24384.amd64fre.win7sp1_ldr_escrow.190220-1800
Machine Name:
Kernel base = 0xfffff800`0261d000 PsLoadedModuleList = 0xfffff800`02856c90
System Uptime: not available
SYMSRV:  BYINDEX: 0x2
         c:\symbols*https://msdl.microsoft.com/download/symbols
         ntkrnlmp.pdb
```

7. Once the debuggee is loaded, fire up "cmd.exe" and type "netstat -ano". Locate the PID that runs port 3389, as following:

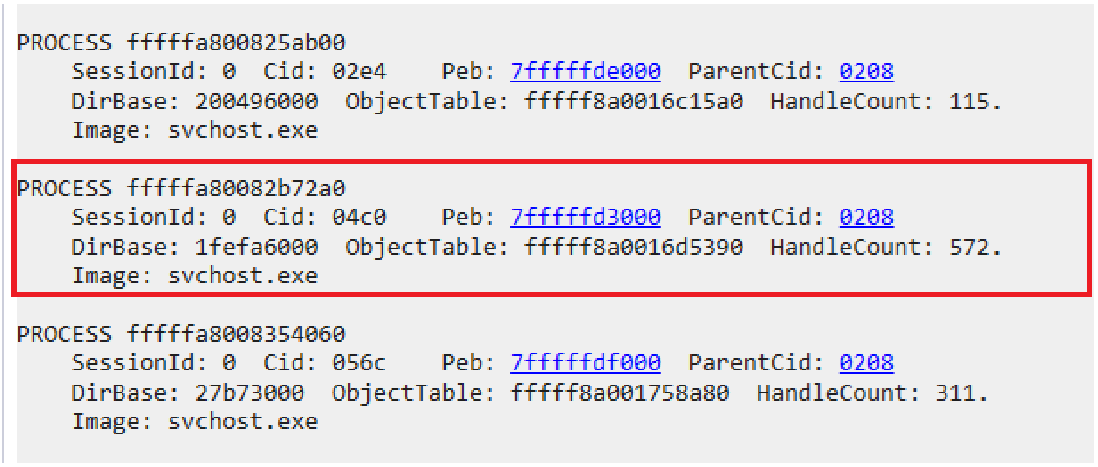


```
C:\Users\straightblast>netstat -ano

Active Connections

  Proto  Local Address          Foreign Address        State           PID
  TCP    0.0.0.0:21             0.0.0.0:0              LISTENING       1548
  TCP    0.0.0.0:135            0.0.0.0:0              LISTENING       748
  TCP    0.0.0.0:445            0.0.0.0:0              LISTENING       4
  TCP    0.0.0.0:3389           0.0.0.0:0              LISTENING       1216
  TCP    0.0.0.0:5357           0.0.0.0:0              LISTENING       4
  TCP    0.0.0.0:49152          0.0.0.0:0              LISTENING       412
  TCP    0.0.0.0:49153          0.0.0.0:0              LISTENING       872
  TCP    0.0.0.0:49154          0.0.0.0:0              LISTENING       220
  TCP    0.0.0.0:49155          0.0.0.0:0              LISTENING       520
  TCP    0.0.0.0:49156          0.0.0.0:0              LISTENING       1996
  TCP    0.0.0.0:49158          0.0.0.0:0              LISTENING       528
  TCP    169.254.241.18:139     0.0.0.0:0              LISTENING       4
  TCP    192.168.0.126:139      0.0.0.0:0              LISTENING       4
  TCP    192.168.0.126:49168    192.168.0.143:139      TIME_WAIT       0
  TCP    192.168.0.126:49171    192.168.0.143:139      TIME_WAIT       0
  TCP    [::]:21                [::]:0                 LISTENING       1548
  TCP    [::]:135               [::]:0                 LISTENING       748
  TCP    [::]:445               [::]:0                 LISTENING       4
  TCP    [::]:3389              [::]:0                 LISTENING       1216
  TCP    [::]:5357              [::]:0                 LISTENING       4
```

8. Go back to the debugger and click on "Home" -> "Break" to enable the debugger command prompt and type:

```
!process 0 0 svchost.exe
```

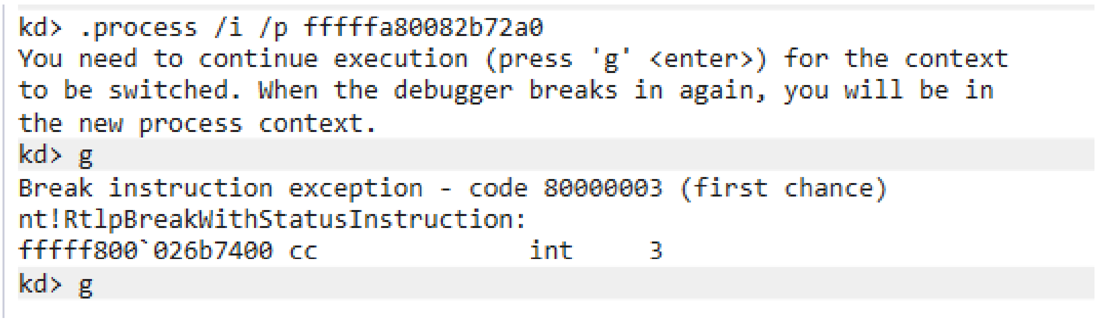
This will list a bunch of process that is associated with svchost.exe. We're interested in the process that has PID 1216 (0x4C0).



```
PROCESS fffffa800825ab00
    SessionId: 0  Cid: 02e4    Peb: 7fffffde000  ParentCid: 0208
    DirBase: 200496000  ObjectTable: fffff8a0016c15a0  HandleCount: 115.
    Image: svchost.exe

PROCESS fffffa80082b72a0
    SessionId: 0  Cid: 04c0    Peb: 7fffffd3000  ParentCid: 0208
    DirBase: 1fefa6000  ObjectTable: fffff8a0016d5390  HandleCount: 572.
    Image: svchost.exe

PROCESS fffffa8008354060
    SessionId: 0  Cid: 056c    Peb: 7fffffdf000  ParentCid: 0208
    DirBase: 27b73000  ObjectTable: fffff8a001758a80  HandleCount: 311.
    Image: svchost.exe
```

9. We will now switch into the context of svchost.exe that runs RDP. In the debugger command prompt, type:

```
.process /i /p fffffa80082b72a0
```



```
kd> .process /i /p fffffa80082b72a0
You need to continue execution (press 'g' <enter>) for the context
to be switched. When the debugger breaks in again, you will be in
the new process context.
kd> g
Break instruction exception - code 80000003 (first chance)
nt!RtlpBreakWithStatusInstruction:
fffff800`026b7400 cc              int     3
kd> g
```

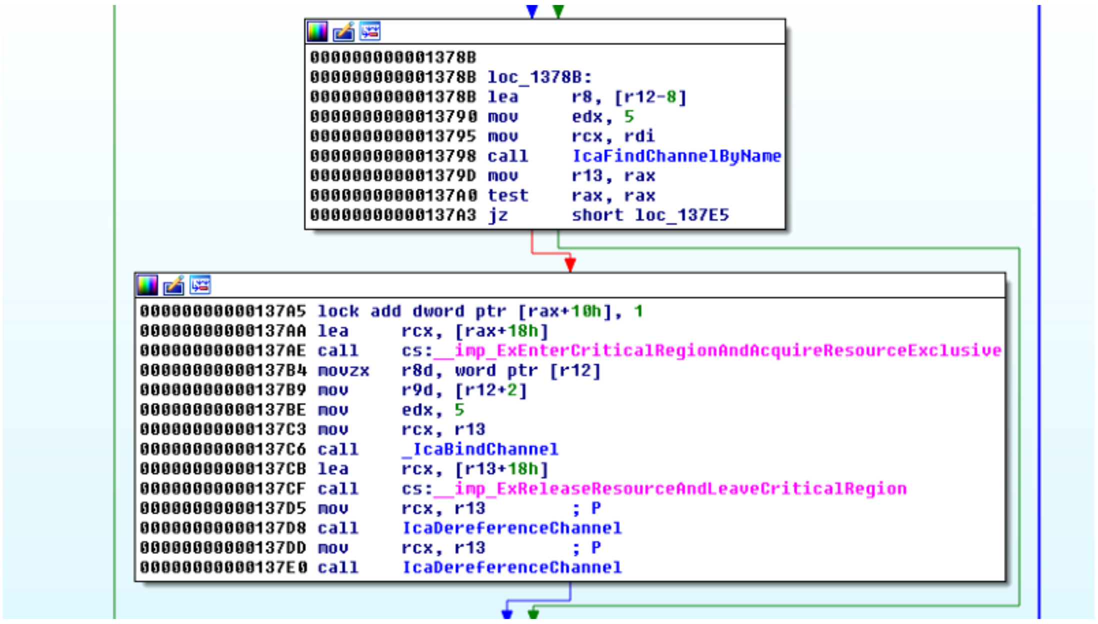
After the context switched, pause the debugger and run the command ".reload" to reload all the symbols that the process will use.

Identifying the relevant code path

Without repeating too much of the public information, the patched vulnerability have code changed in the IcaBindVirtualChannels. We know that if IcaFindChannelByName finds the string "MS_T120", it calls IcaBindchannel such as:

```
_IcaBindChannel(ChannelControlStructure*, 5, index,
dontcare)
```

The following screenshots depicts the relevant unpatched code in IcaBindVirtualChannels:

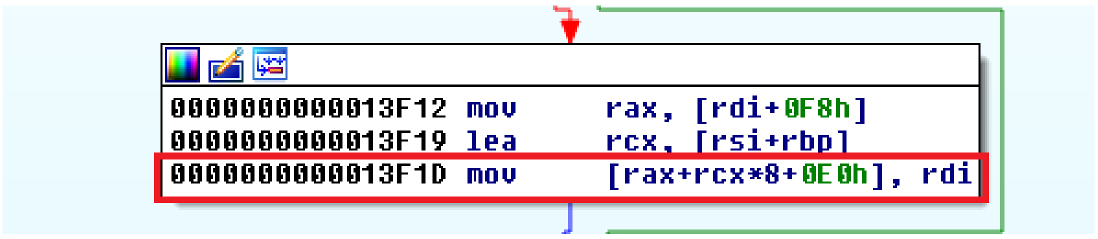


```
000000000001378B
000000000001378B loc_1378B:
000000000001378B lea     r8, [r12-8]
0000000000013790 mov     edx, 5
0000000000013795 mov     rcx, rdi
0000000000013798 call    IcaFindChannelByName
000000000001379D mov     r13, rax
00000000000137A0 test    rax, rax
00000000000137A3 jz      short loc_137E5
```

```
00000000000137A5 lock add dword ptr [rax+10h], 1
00000000000137AA lea     rcx, [rax+18h]
00000000000137AE call    cs:__imp_ExEnterCriticalRegionAndAcquireResourceExclusive
00000000000137B4 movzx   r8d, word ptr [r12]
00000000000137B9 mov     r9d, [r12+2]
00000000000137BE mov     edx, 5
00000000000137C3 mov     rcx, r13
00000000000137C6 call    _IcaBindChannel
00000000000137CB lea     rcx, [r13+18h]
00000000000137CF call    cs:__imp_ExReleaseResourceAndLeaveCriticalRegion
00000000000137D5 mov     rcx, r13        ; P
00000000000137D8 call    IcaDereferenceChannel
00000000000137DD mov     rcx, r13        ; P
00000000000137E0 call    IcaDereferenceChannel
```

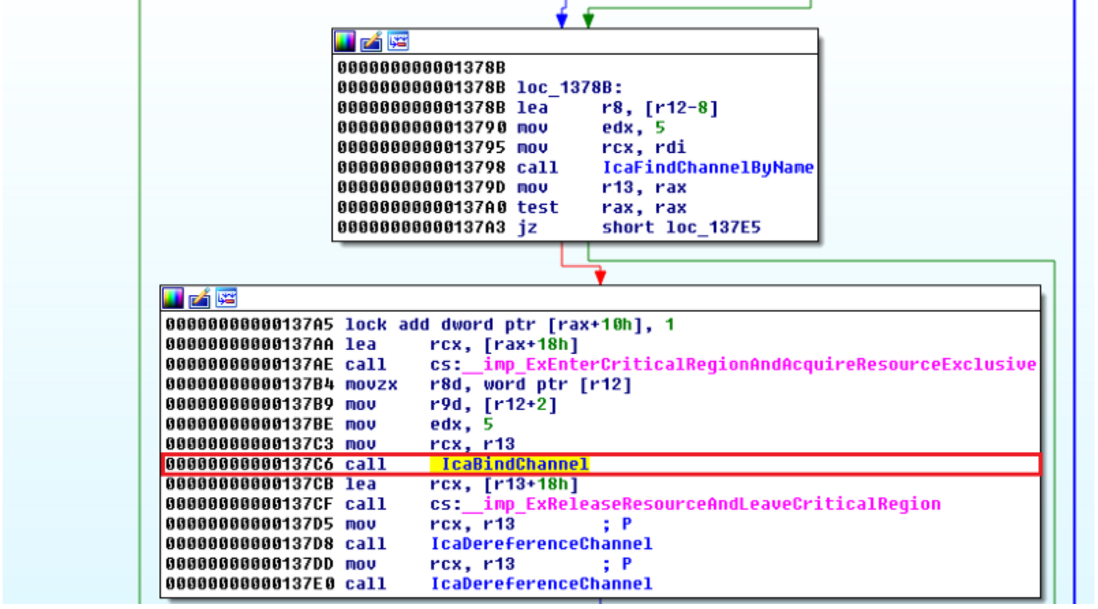
We're going to set two breakpoints.

One will be on _IcaBindChannel where the channel control structure is stored into the channel pointer table. The index of where the channel control structure is stored is based on the index of where the Virtual Channel name is declared within the clientNetworkData of the MCS Initial Connect and GCC Create packet.



```
0000000000013F12 mov     rax, [rdi+0F8h]
0000000000013F19 lea     rcx, [rsi+rbp]
0000000000013F1D mov     [rax+rcx*8+0E0h], rdi
```

and the other one on the "call _IcaBindChannel" within the IcaBindVirtualChannels.

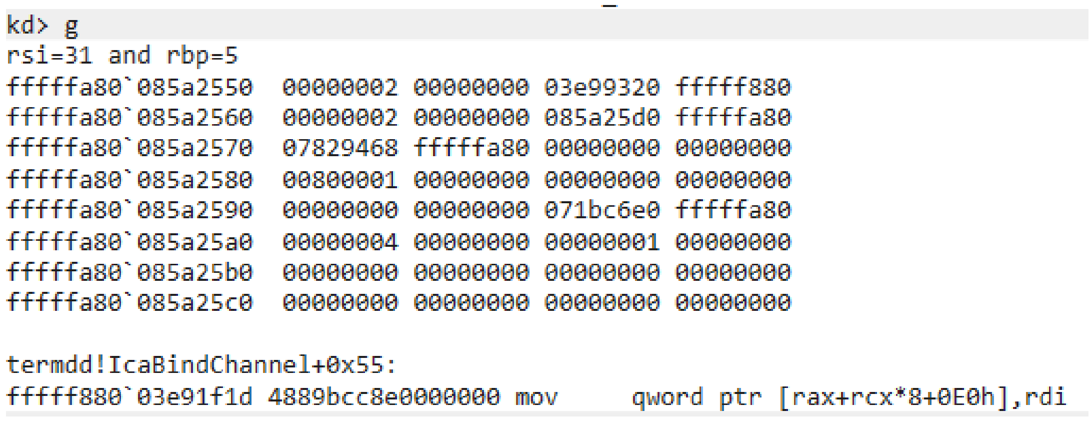

```
000000000001378B
000000000001378B loc_1378B:
000000000001378B lea     r8, [r12-8]
0000000000013790 mov     edx, 5
0000000000013795 mov     rcx, rdi
0000000000013798 call    IcaFindChannelByName
000000000001379D mov     r13, rax
00000000000137A0 test    rax, rax
00000000000137A3 jz      short loc_137E5
```

```
00000000000137A5 lock add dword ptr [rax+10h], 1
00000000000137AA lea     rcx, [rax+18h]
00000000000137AE call    cs:__imp_ExEnterCriticalRegionAndAcquireResourceExclusive
00000000000137B4 movzx   r8d, word ptr [r12]
00000000000137B9 mov     r9d, [r12+2]
00000000000137BE mov     edx, 5
00000000000137C3 mov     rcx, r13
00000000000137C6 call    IcaBindChannel
00000000000137CB lea     rcx, [r13+18h]
00000000000137CF call    cs:__imp_ExReleaseResourceAndLeaveCriticalRegion
00000000000137D5 mov     rcx, r13        ; P
00000000000137D8 call    IcaDereferenceChannel
00000000000137DD mov     rcx, r13        ; P
00000000000137E0 call    IcaDereferenceChannel
```

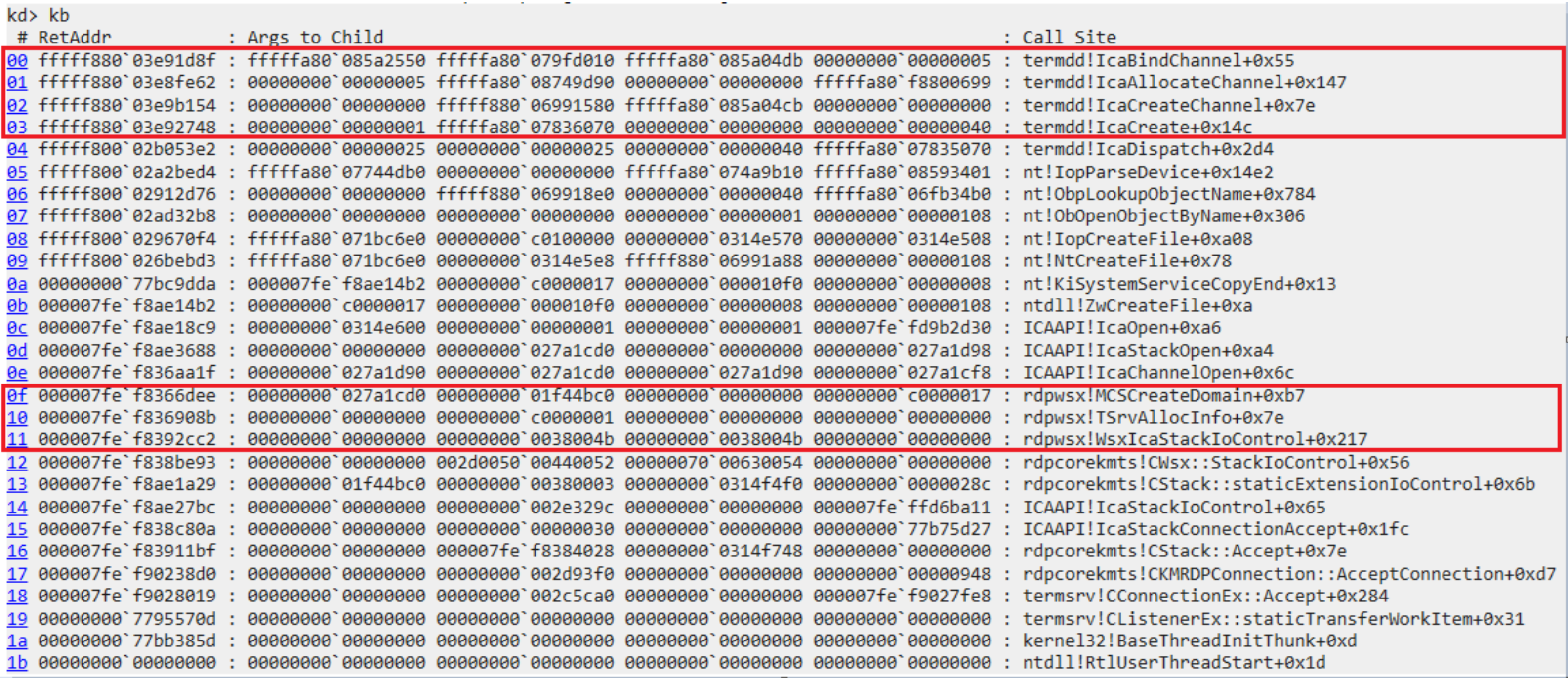
The purpose of these breakpoints areto observe the creation of virtual channels and the orders these channels are created.

```
bp termdd!IcaBindChannel+0x55 ".printf \"rsi=%d and rbp=%d\\n\", rsi, rbp;dd rdi;.echo"

bp termdd!IcaBindVirtualChannels+0x19e ".printf \"We got a MS_T120, r8=%d\\n\",r8;dd rcx;r $t0=rcx;.echo"
```

The breakpoint first hits the following, with an index value of “31”:

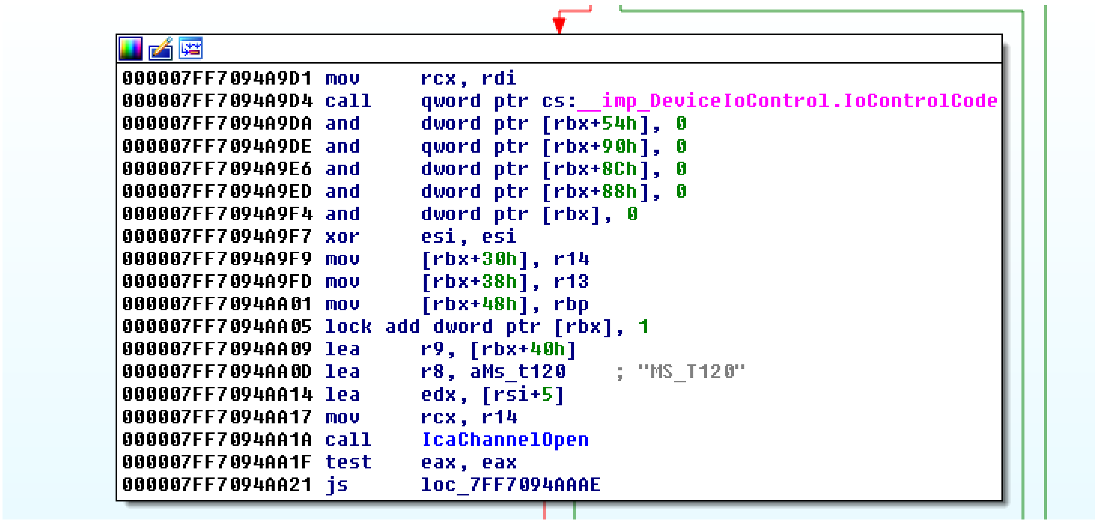

```
kd> g
rsi=31 and rbp=5
fffffa80`085a2550  00000002 00000000 03e99320 fffff880
fffffa80`085a2560  00000002 00000000 085a25d0 fffffa80
fffffa80`085a2570  07829468 fffffa80 00000000 00000000
fffffa80`085a2580  00800001 00000000 00000000 00000000
fffffa80`085a2590  00000000 00000000 071bc6e0 fffffa80
fffffa80`085a25a0  00000004 00000000 00000001 00000000
fffffa80`085a25b0  00000000 00000000 00000000 00000000
fffffa80`085a25c0  00000000 00000000 00000000 00000000

termdd!IcaBindChannel+0x55:
fffff880`03e91f1d 4889bcc8e0000000 mov     qword ptr [rax+rcx*8+0E0h],rdi
```

Listing the call stack with “kb” shows the following:



```
kd> kb
 # RetAddr           : Args to Child                                                           : Call Site
00 fffff880`03e91d8f : fffffa80`085a2550 fffffa80`079fd010 fffffa80`085a04db 00000000`00000005 : termdd!IcaBindChannel+0x55
01 fffff880`03e8fe62 : 00000000`00000005 fffffa80`08749d90 00000000`00000000 fffffa80`f8800699 : termdd!IcaAllocateChannel+0x147
02 fffff880`03e9b154 : 00000000`00000000 fffff880`06991580 fffffa80`085a04cb 00000000`00000000 : termdd!IcaCreateChannel+0x7e
03 fffff880`03e92748 : 00000000`00000001 fffffa80`07836070 00000000`00000000 00000000`00000040 : termdd!IcaCreate+0x14c
04 fffff800`02b053e2 : 00000000`00000025 00000000`00000025 00000000`00000040 fffffa80`07835070 : termdd!IcaDispatch+0x2d4
05 fffff800`02a2bed4 : fffffa80`07744db0 00000000`00000000 fffffa80`074a9b10 fffffa80`08593401 : nt!IopParseDevice+0x14e2
06 fffff800`02912d76 : 00000000`00000000 fffff880`069918e0 00000000`00000040 fffffa80`06fb34b0 : nt!ObpLookupObjectName+0x784
07 fffff800`02ad32b8 : 00000000`00000000 00000000`00000000 00000000`00000001 00000000`00000108 : nt!ObOpenObjectByName+0x306
08 fffff800`029670f4 : fffffa80`071bc6e0 00000000`c0100000 00000000`0314e570 00000000`0314e508 : nt!IopCreateFile+0xa08
09 fffff800`026bebd3 : fffffa80`071bc6e0 00000000`0314e5e8 fffff880`06991a88 00000000`00000108 : nt!NtCreateFile+0x78
0a 00000000`77bc9dda : 000007fe`f8ae14b2 00000000`c0000017 00000000`000010f0 00000000`00000008 : nt!KiSystemServiceCopyEnd+0x13
0b 000007fe`f8ae14b2 : 00000000`c0000017 00000000`000010f0 00000000`00000008 00000000`00000108 : ntdll!ZwCreateFile+0xa
0c 000007fe`f8ae18c9 : 00000000`0314e600 00000000`00000001 00000000`00000001 000007fe`fd9b2d30 : ICAAPI!IcaOpen+0xa6
0d 000007fe`f8ae3688 : 00000000`00000000 00000000`027a1cd0 00000000`00000000 00000000`027a1d98 : ICAAPI!IcaStackOpen+0xa4
0e 000007fe`f836aa1f : 00000000`027a1d90 00000000`027a1cd0 00000000`027a1d90 00000000`027a1cf8 : ICAAPI!IcaChannelOpen+0x6c
0f 000007fe`f8366dee : 00000000`027a1cd0 00000000`01f44bc0 00000000`00000000 00000000`c0000017 : rdpwsx!MCSCreateDomain+0xb7
10 000007fe`f836908b : 00000000`00000000 00000000`c0000001 00000000`00000000 00000000`00000000 : rdpwsx!TSrvAllocInfo+0x7e
11 000007fe`f8392cc2 : 00000000`00000000 00000000`0038004b 00000000`0038004b 00000000`00000000 : rdpwsx!WsxIcaStackIoControl+0x217
12 000007fe`f838be93 : 00000000`00000000 002d0050`00440052 00000070`00630054 00000000`00000000 : rdpcorekmts!CWsx::StackIoControl+0x56
13 000007fe`f8ae1a29 : 00000000`01f44bc0 00000000`00380003 00000000`0314f4f0 00000000`0000028c : rdpcorekmts!CStack::staticExtensionIoControl+0x6b
14 000007fe`f8ae27bc : 00000000`00000000 00000000`002e329c 00000000`00000000 000007fe`ffd6ba11 : ICAAPI!IcaStackIoControl+0x65
15 000007fe`f838c80a : 00000000`00000000 00000000`00000030 00000000`00000000 00000000`77b75d27 : ICAAPI!IcaStackConnectionAccept+0x1fc
16 000007fe`f83911bf : 00000000`00000000 000007fe`f8384028 00000000`0314f748 00000000`00000000 : rdpcorekmts!CStack::Accept+0x7e
17 000007fe`f90238d0 : 00000000`00000000 00000000`002d93f0 00000000`00000000 00000000`00000948 : rdpcorekmts!CKMRDPConnection::AcceptConnection+0xd7
18 000007fe`f9028019 : 00000000`00000000 00000000`002c5ca0 00000000`00000000 000007fe`f9027fe8 : termsrv!CConnectionEx::Accept+0x284
19 00000000`7795570d : 00000000`00000000 00000000`00000000 00000000`00000000 00000000`00000000 : termsrv!CListenerEx::staticTransferWorkItem+0x31
1a 00000000`77bb385d : 00000000`00000000 00000000`00000000 00000000`00000000 00000000`00000000 : kernel32!BaseThreadInitThunk+0xd
1b 00000000`00000000 : 00000000`00000000 00000000`00000000 00000000`00000000 00000000`00000000 : ntdll!RtlUserThreadStart+0x1d
```

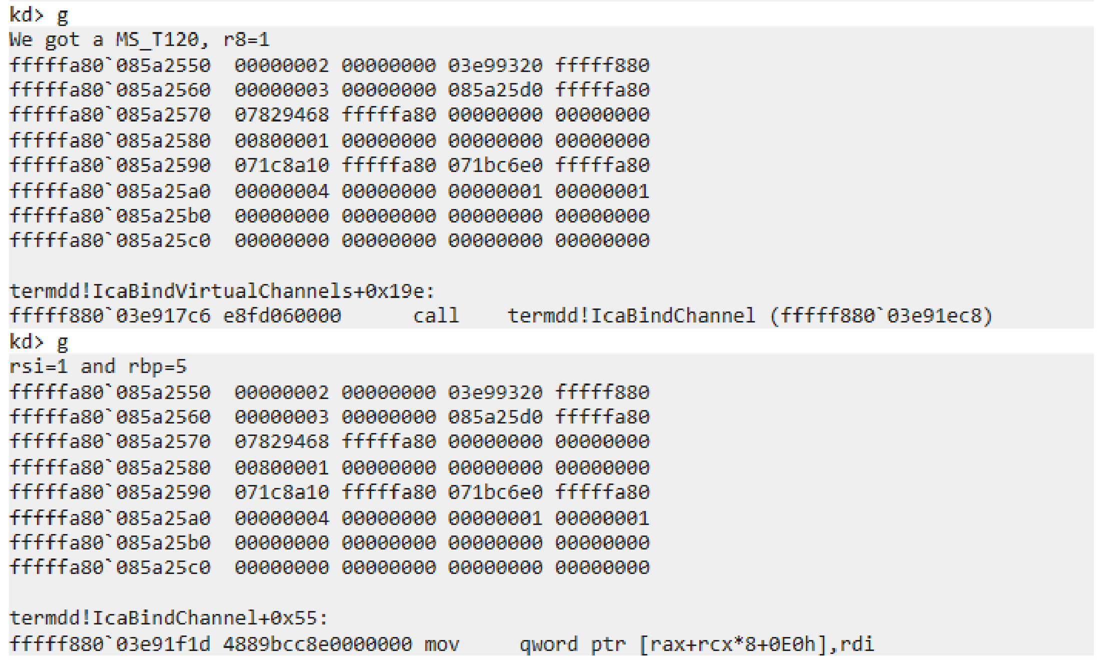
We can see the IcaBindChannel is called from a IcaCreateChannel, which can be traced all the way to the rdpwsx!MSCreateDomain. If we take a look at that function under a disassembler, we noticed it is creating the MS_T120 channel:



```
000007FF7094A9D1 mov     rcx, rdi
000007FF7094A9D4 call    qword ptr cs:__imp_DeviceIoControl.IoControlCode
000007FF7094A9DA and     dword ptr [rbx+54h], 0
000007FF7094A9DE and     qword ptr [rbx+90h], 0
000007FF7094A9E6 and     dword ptr [rbx+8Ch], 0
000007FF7094A9ED and     dword ptr [rbx+88h], 0
000007FF7094A9F4 and     dword ptr [rbx], 0
000007FF7094A9F7 xor     esi, esi
000007FF7094A9F9 mov     [rbx+30h], r14
000007FF7094A9FD mov     [rbx+38h], r13
000007FF7094AA01 mov     [rbx+48h], rbp
000007FF7094AA05 lock add dword ptr [rbx], 1
000007FF7094AA09 lea     r9, [rbx+40h]
000007FF7094AA0D lea     r8, aMs_t120     ; "MS_T120"
000007FF7094AA14 lea     edx, [rsi+5]
000007FF7094AA17 mov     rcx, r14
000007FF7094AA1A call    IcaChannelOpen
000007FF7094AA1F test    eax, eax
000007FF7094AA21 js      loc_7FF7094AAAE
```

Also, but looking at the patched termdd.sys, we know that the patched code enforces the index for MS_T120 virtual channel to be 31, this first breakpoint indicates the first channel that gets created is the MS_T120 channel.

The next breakpoint hit is the 2nd breakpoint (within the IcaBindVirtualChannel), followed by the 1st breakpoint (within IcaBindChannel) again:



```
kd> g
We got a MS_T120, r8=1
fffffa80`085a2550  00000002 00000000 03e99320 fffff880
fffffa80`085a2560  00000003 00000000 085a25d0 fffffa80
fffffa80`085a2570  07829468 fffffa80 00000000 00000000
fffffa80`085a2580  00800001 00000000 00000000 00000000
fffffa80`085a2590  071c8a10 fffffa80 071bc6e0 fffffa80
fffffa80`085a25a0  00000004 00000000 00000001 00000001
fffffa80`085a25b0  00000000 00000000 00000000 00000000
fffffa80`085a25c0  00000000 00000000 00000000 00000000

termdd!IcaBindVirtualChannels+0x19e:
fffff880`03e917c6 e8fd060000      call    termdd!IcaBindChannel (fffff880`03e91ec8)
kd> g
rsi=1 and rbp=5
fffffa80`085a2550  00000002 00000000 03e99320 fffff880
fffffa80`085a2560  00000003 00000000 085a25d0 fffffa80
fffffa80`085a2570  07829468 fffffa80 00000000 00000000
fffffa80`085a2580  00800001 00000000 00000000 00000000
fffffa80`085a2590  071c8a10 fffffa80 071bc6e0 fffffa80
fffffa80`085a25a0  00000004 00000000 00000001 00000001
fffffa80`085a25b0  00000000 00000000 00000000 00000000
fffffa80`085a25c0  00000000 00000000 00000000 00000000

termdd!IcaBindChannel+0x55:
fffff880`03e91f1d 4889bcc8e0000000 mov     qword ptr [rax+rcx*8+0E0h],rdi
```

This gets hit as it observed the MS_T120 value from the clientNetworkData. If we compared the address and content displayed in above image with the one way, way above, we can see they're identical. This means both are referring to the same channel control structure. However, the reference to this structure is being stored at two different locations:

```
rsi = 31, rbp = 5;
[rax + (31 + 5) * 8 + 0xe0] = MST_120_structure

rsi = 1, rbp = 5;
[rax + (1 + 5) * 8 + 0xe0] = MS_T120_structure
```

In another words, there are two entries in the channel pointer table that have references to the MS_T120 structure.

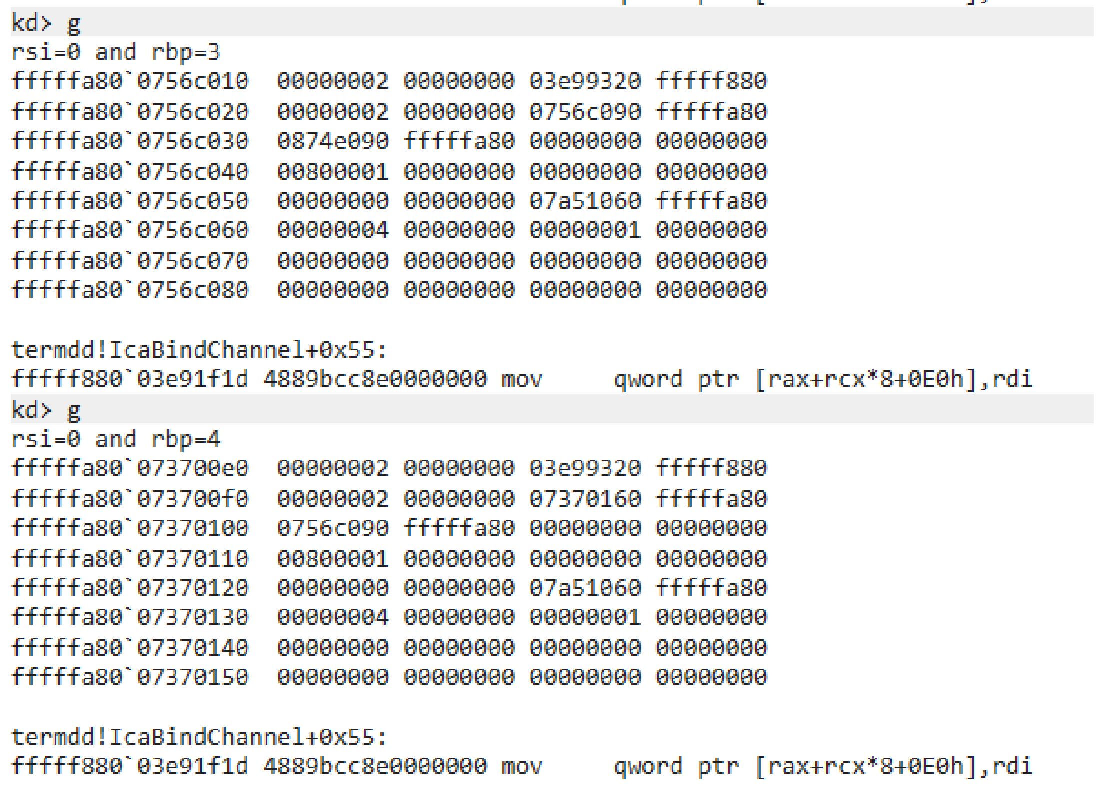
Afterwards, a few more channels are created which we don't care about:

```
kd> g
rsi=7 and rbp=5
fffffa80`07818b10  00000002 00000000 03e99320 fffff880
fffffa80`07818b20  00000002 00000000 07818b90 fffffa80
fffffa80`07818b30  085a25d0 fffffa80 00000000 00000000
fffffa80`07818b40  00800001 00000000 00000000 00000000
fffffa80`07818b50  00000000 00000000 071bc6e0 fffffa80
fffffa80`07818b60  00000004 00000000 00000001 00000000
fffffa80`07818b70  00000000 00000000 00000000 00000000
fffffa80`07818b80  00000000 00000000 00000000 00000000

termdd!IcaBindChannel+0x55:
fffff880`03e91f1d 4889bcc8e0000000 mov     qword ptr [rax+rcx*8+0E0h],rdi
```

index 7 with offset 5

```
kd> g
rsi=0 and rbp=0
fffffa80`07578b50  00000002 00000000 03e99320 fffff880
fffffa80`07578b60  00000002 00000000 07578bd0 fffffa80
fffffa80`07578b70  073d09d0 fffffa80 00000000 00000000
fffffa80`07578b80  00800001 00000000 00000000 00000000
fffffa80`07578b90  00000000 00000000 07a51060 fffffa80
fffffa80`07578ba0  00000004 00000000 00000001 00000000
fffffa80`07578bb0  00000000 00000000 00000000 00000000
fffffa80`07578bc0  00000000 00000000 00000000 00000000

termdd!IcaBindChannel+0x55:
fffff880`03e91f1d 4889bcc8e0000000 mov     qword ptr [rax+rcx*8+0E0h],rdi
kd> g
rsi=0 and rbp=1
fffffa80`0874e010  00000002 00000000 03e99320 fffff880
fffffa80`0874e020  00000002 00000000 0874e090 fffffa80
fffffa80`0874e030  07578bd0 fffffa80 00000000 00000000
fffffa80`0874e040  00800001 00000000 00000000 00000000
fffffa80`0874e050  00000000 00000000 07a51060 fffffa80
fffffa80`0874e060  00000004 00000000 00000001 00000000
fffffa80`0874e070  00000000 00000000 00000000 00000000
fffffa80`0874e080  00000000 00000000 00000000 00000000

termdd!IcaBindChannel+0x55:
fffff880`03e91f1d 4889bcc8e0000000 mov     qword ptr [rax+rcx*8+0E0h],rdi
```

index 0 with offset 0 and 1

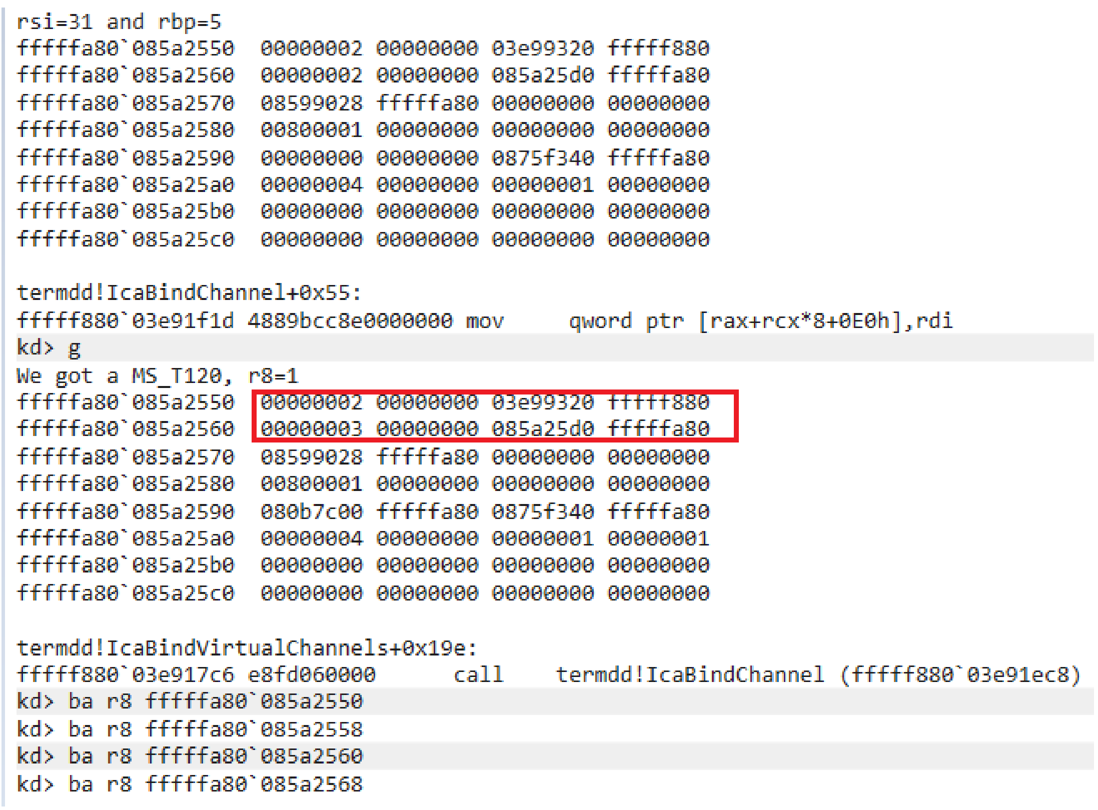


```
kd> g
rsi=0 and rbp=3
fffffa80`0756c010  00000002 00000000 03e99320 fffff880
fffffa80`0756c020  00000002 00000000 0756c090 fffffa80
fffffa80`0756c030  0874e090 fffffa80 00000000 00000000
fffffa80`0756c040  00800001 00000000 00000000 00000000
fffffa80`0756c050  00000000 00000000 07a51060 fffffa80
fffffa80`0756c060  00000004 00000000 00000001 00000000
fffffa80`0756c070  00000000 00000000 00000000 00000000
fffffa80`0756c080  00000000 00000000 00000000 00000000

termdd!IcaBindChannel+0x55:
fffff880`03e91f1d 4889bcc8e0000000 mov     qword ptr [rax+rcx*8+0E0h],rdi
kd> g
rsi=0 and rbp=4
fffffa80`073700e0  00000002 00000000 03e99320 fffff880
fffffa80`073700f0  00000002 00000000 07370160 fffffa80
fffffa80`07370100  0756c090 fffffa80 00000000 00000000
fffffa80`07370110  00800001 00000000 00000000 00000000
fffffa80`07370120  00000000 00000000 07a51060 fffffa80
fffffa80`07370130  00000004 00000000 00000001 00000000
fffffa80`07370140  00000000 00000000 00000000 00000000
fffffa80`07370150  00000000 00000000 00000000 00000000

termdd!IcaBindChannel+0x55:
fffff880`03e91f1d 4889bcc8e0000000 mov     qword ptr [rax+rcx*8+0E0h],rdi
```

index 0 with offset 3 and 4

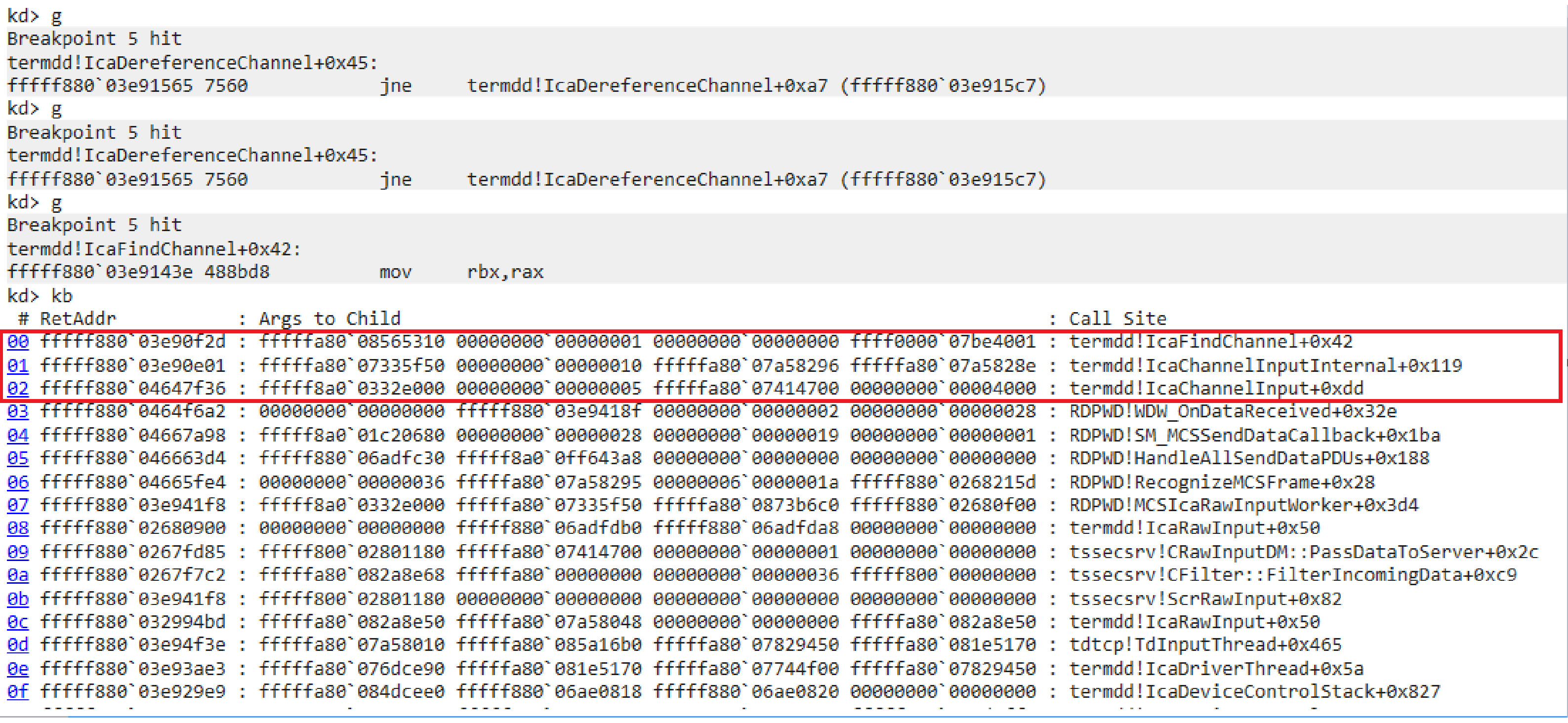
The next step into finding other relevant code to look at will be to set a break read/write on the MS_T120 structure. It is with certain the MS_T120 structure will be 'touch' in the future.

I set the break read/write breakpoint on the data within the red box, as shown in the following:



```
rsi=31 and rbp=5
fffffa80`085a2550  00000002 00000000 03e99320 fffff880
fffffa80`085a2560  00000002 00000000 085a25d0 fffffa80
fffffa80`085a2570  08599028 fffffa80 00000000 00000000
fffffa80`085a2580  00800001 00000000 00000000 00000000
fffffa80`085a2590  00000000 00000000 0875f340 fffffa80
fffffa80`085a25a0  00000004 00000000 00000001 00000000
fffffa80`085a25b0  00000000 00000000 00000000 00000000
fffffa80`085a25c0  00000000 00000000 00000000 00000000

termdd!IcaBindChannel+0x55:
fffff880`03e91f1d 4889bcc8e0000000 mov     qword ptr [rax+rcx*8+0E0h],rdi
kd> g
We got a MS_T120, r8=1
fffffa80`085a2550  00000002 00000000 03e99320 fffff880
fffffa80`085a2560  00000003 00000000 085a25d0 fffffa80
fffffa80`085a2570  08599028 fffffa80 00000000 00000000
fffffa80`085a2580  00800001 00000000 00000000 00000000
fffffa80`085a2590  080b7c00 fffffa80 0875f340 fffffa80
fffffa80`085a25a0  00000004 00000000 00000001 00000001
fffffa80`085a25b0  00000000 00000000 00000000 00000000
fffffa80`085a25c0  00000000 00000000 00000000 00000000

termdd!IcaBindVirtualChannels+0x19e:
fffff880`03e917c6 e8fd060000      call    termdd!IcaBindChannel (fffff880`03e91ec8)
kd> ba r8 fffffa80`085a2550
kd> ba r8 fffffa80`085a2558
kd> ba r8 fffffa80`085a2560
kd> ba r8 fffffa80`085a2568
```

As we proceed with the execution, we get calls to IcaDereferenceChannel, which we're not interested in. Then, we hit termdd!IcaFindChannel, with some more information to look into from the call stack:

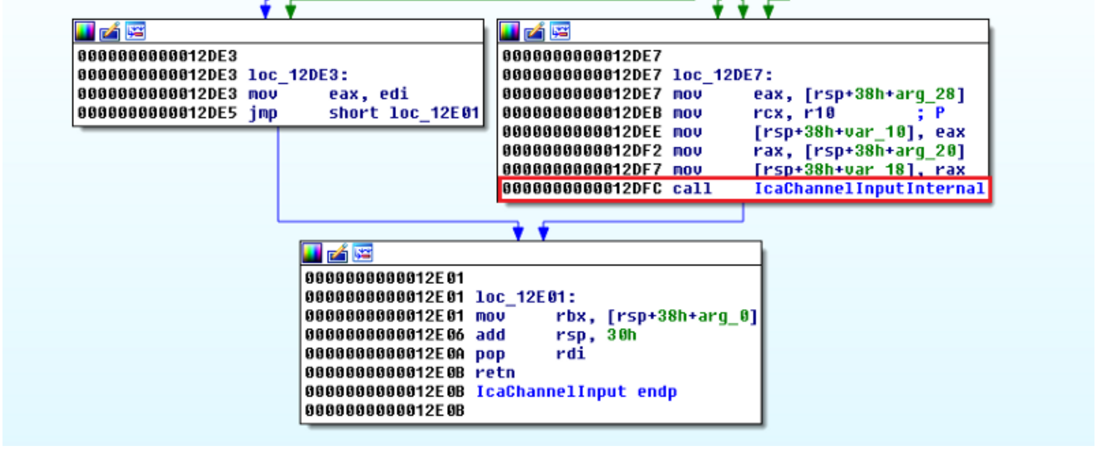


```
kd> g
Breakpoint 5 hit
termdd!IcaDereferenceChannel+0x45:
fffff880`03e91565 7560            jne     termdd!IcaDereferenceChannel+0xa7 (fffff880`03e915c7)
kd> g
Breakpoint 5 hit
termdd!IcaDereferenceChannel+0x45:
fffff880`03e91565 7560            jne     termdd!IcaDereferenceChannel+0xa7 (fffff880`03e915c7)
kd> g
Breakpoint 5 hit
termdd!IcaFindChannel+0x42:
fffff880`03e9143e 488bd8          mov     rbx,rax
kd> kb
 # RetAddr           : Args to Child                                                           : Call Site
00 fffff880`03e90f2d : fffffa80`08565310 00000000`00000001 00000000`00000000 ffff0000`07be4001 : termdd!IcaFindChannel+0x42
01 fffff880`03e90e01 : fffffa80`07335f50 00000000`00000010 fffffa80`07a58296 fffffa80`07a5828e : termdd!IcaChannelInputInternal+0x119
02 fffff880`04647f36 : fffff8a0`0332e000 00000000`00000005 fffffa80`07414700 00000000`00004000 : termdd!IcaChannelInput+0xdd
03 fffff880`0464f6a2 : 00000000`00000000 fffff880`03e9418f 00000000`00000002 00000000`00000028 : RDPWD!WDW_OnDataReceived+0x32e
04 fffff880`04667a98 : fffff8a0`01c20680 00000000`00000028 00000000`00000019 00000000`00000001 : RDPWD!SM_MCSSendDataCallback+0x1ba
05 fffff880`046663d4 : fffff880`06adfc30 fffff8a0`0ff643a8 00000000`00000000 00000000`00000000 : RDPWD!HandleAllSendDataPDUs+0x188
06 fffff880`04665fe4 : 00000000`00000036 fffffa80`07a58295 00000006`0000001a fffff880`0268215d : RDPWD!RecognizeMCSFrame+0x28
07 fffff880`03e941f8 : fffff8a0`0332e000 fffffa80`07335f50 fffffa80`0873b6c0 fffff880`02680f00 : RDPWD!MCSIcaRawInputWorker+0x3d4
08 fffff880`02680900 : 00000000`00000000 fffff880`06adfdb0 fffff880`06adfda8 00000000`00000000 : termdd!IcaRawInput+0x50
09 fffff880`0267fd85 : fffff800`02801180 fffffa80`07414700 00000000`00000001 00000000`00000000 : tssecsrv!CRawInputDM::PassDataToServer+0x2c
0a fffff880`0267f7c2 : fffffa80`082a8e68 fffffa80`00000000 00000000`00000036 fffff800`00000000 : tssecsrv!CFilter::FilterIncomingData+0xc9
0b fffff880`03e941f8 : fffff800`02801180 00000000`00000000 00000000`00000000 00000000`00000000 : tssecsrv!ScrRawInput+0x82
0c fffff880`032994bd : fffffa80`082a8e50 fffffa80`07a58048 00000000`00000000 fffffa80`082a8e50 : termdd!IcaRawInput+0x50
0d fffff880`03e94f3e : fffffa80`07a58010 fffffa80`085a16b0 fffffa80`07829450 fffffa80`081e5170 : tdtcp!TdInputThread+0x465
0e fffff880`03e93ae3 : fffffa80`076dce90 fffffa80`081e5170 fffffa80`07744f00 fffffa80`07829450 : termdd!IcaDriverThread+0x5a
0f fffff880`03e929e9 : fffffa80`084dcee0 fffff880`06ae0818 fffff880`06ae0820 00000000`00000000 : termdd!IcaDeviceControlStack+0x827
```

The termdd!IcaChannelInput and termdd!IcaChannelInputInternal sounds like something that might process data sent to the virtual channel.

A pro tip is to set breakpoint before a function call, to see if the registers or stacks (depending how data are passed to a function) could contain recognizable or readable data.

I will set a breakpoint on the call to IcaChannelInputInternal, within the IcaChannelInput function:

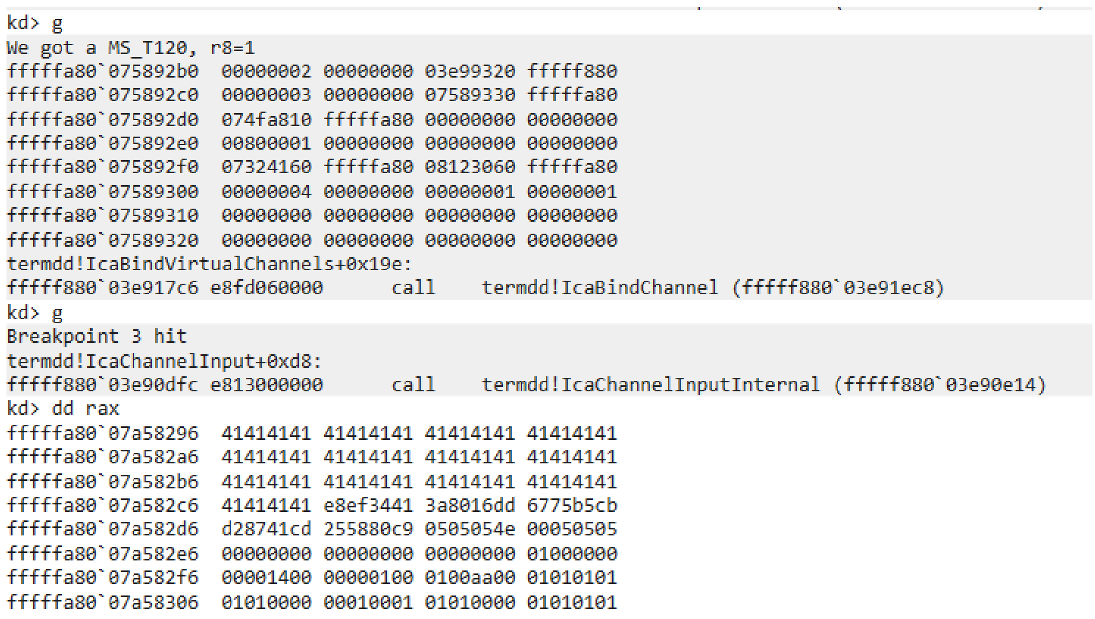


```
0000000000012DE3
0000000000012DE3 loc_12DE3:
0000000000012DE3 mov     eax, edi
0000000000012DE5 jmp     short loc_12E01
```

```
0000000000012DE7
0000000000012DE7 loc_12DE7:
0000000000012DE7 mov     eax, [rsp+38h+arg_28]
0000000000012DEB mov     rcx, r10        ; P
0000000000012DEE mov     [rsp+38h+var_10], eax
0000000000012DF2 mov     rax, [rsp+38h+arg_20]
0000000000012DF7 mov     [rsp+38h+var_18], rax
0000000000012DFC call    IcaChannelInputInternal
```

```
0000000000012E01
0000000000012E01 loc_12E01:
0000000000012E01 mov     rbx, [rsp+38h+arg_0]
0000000000012E06 add     rsp, 30h
0000000000012E0A pop     rdi
0000000000012E0B retn
0000000000012E0B IcaChannelInput endp
0000000000012E0B
```

```
bp termdd!IcaChannelInput+0xd8
```



```
kd> g
We got a MS_T120, r8=1
fffffa80`075892b0  00000002 00000000 03e99320 fffff880
fffffa80`075892c0  00000003 00000000 07589330 fffffa80
fffffa80`075892d0  074fa810 fffffa80 00000000 00000000
fffffa80`075892e0  00800001 00000000 00000000 00000000
fffffa80`075892f0  07324160 fffffa80 08123060 fffffa80
fffffa80`07589300  00000004 00000000 00000001 00000001
fffffa80`07589310  00000000 00000000 00000000 00000000
fffffa80`07589320  00000000 00000000 00000000 00000000
termdd!IcaBindVirtualChannels+0x19e:
fffff880`03e917c6 e8fd060000      call    termdd!IcaBindChannel (fffff880`03e91ec8)
kd> g
Breakpoint 3 hit
termdd!IcaChannelInput+0xd8:
fffff880`03e90dfc e813000000      call    termdd!IcaChannelInputInternal (fffff880`03e90e14)
kd> dd rax
fffffa80`07a58296  41414141 41414141 41414141 41414141
fffffa80`07a582a6  41414141 41414141 41414141 41414141
fffffa80`07a582b6  41414141 41414141 41414141 41414141
fffffa80`07a582c6  41414141 e8ef3441 3a8016dd 6775b5cb
fffffa80`07a582d6  d28741cd 255880c9 0505054e 00050505
fffffa80`07a582e6  00000000 00000000 00000000 01000000
fffffa80`07a582f6  00001400 00000100 0100aa00 01010101
fffffa80`07a58306  01010000 00010001 01010000 01010101
```

We're interested in calls to the IcaChannelInput breakpoint after IcaBindVirtualChannels has been called. From the above image, just right before the call to IcaChannelInputInternal, the rax register holds an address that references to the **"A"s** I passed over as data through the virtual channel.

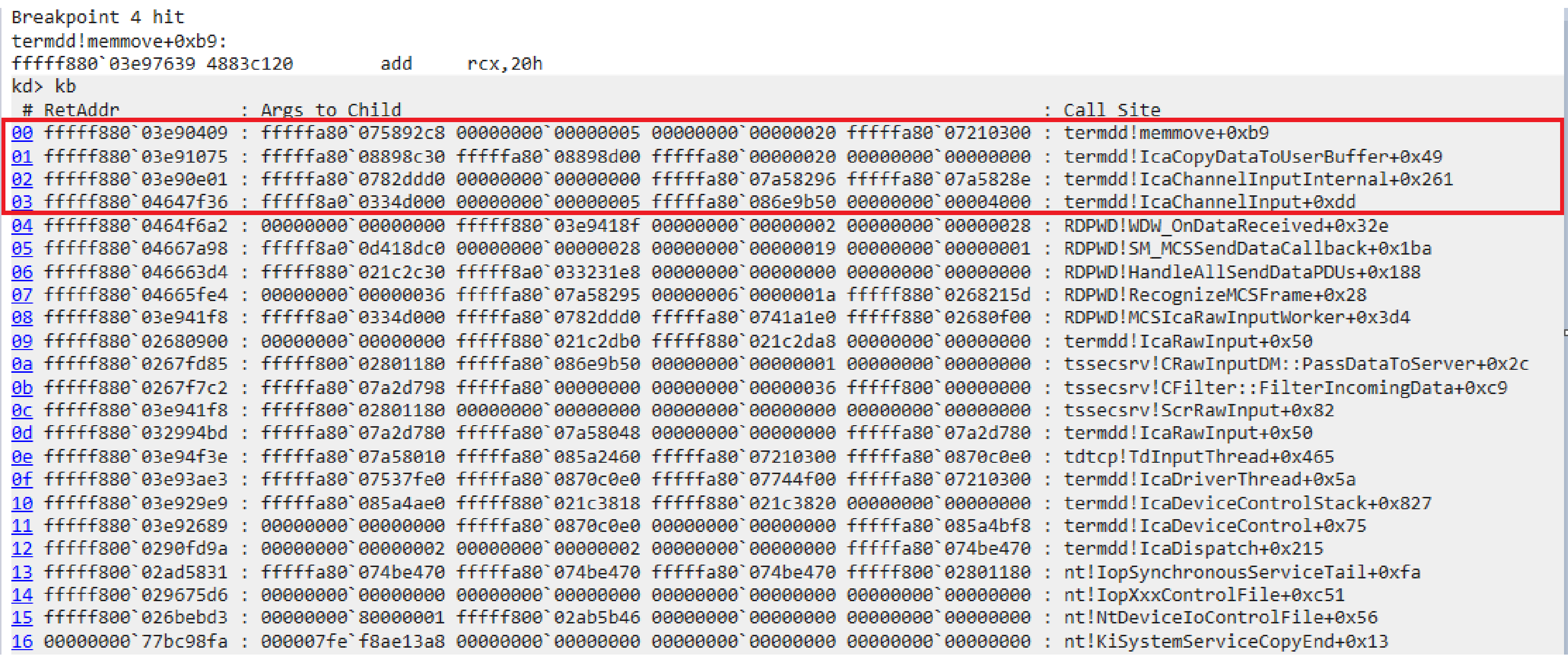
I will now set another set of break on read/write on the **"A"s** to see what code will 'touch' them.

```
ba r8 rax+0xa
```

The reason I had to add 0xA to the rax register is because the break on read/write requires an align address (ends in0x0 or 0x8 for x64 env)

```
kd> dd rax
fffffa80`07a58296  41414141 41414141 41414141 41414141
fffffa80`07a582a6  41414141 41414141 41414141 41414141
fffffa80`07a582b6  41414141 41414141 41414141 41414141
fffffa80`07a582c6  41414141 e8ef3441 3a8016dd 6775b5cb
fffffa80`07a582d6  d28741cd 255880c9 0505054e 00050505
fffffa80`07a582e6  00000000 00000000 00000000 01000000
fffffa80`07a582f6  00001400 00000100 0100aa00 01010101
fffffa80`07a58306  01010000 00010001 01010000 01010101
kd> dd rax+0xa
fffffa80`07a582a0  41414141 41414141 41414141 41414141
fffffa80`07a582b0  41414141 41414141 41414141 41414141
fffffa80`07a582c0  41414141 41414141 34414141 16dde8ef
fffffa80`07a582d0  b5cb3a80 41cd6775 80c9d287 054e2558
fffffa80`07a582e0  05050505 00000005 00000000 00000000
fffffa80`07a582f0  00000000 14000100 01000000 aa000000
fffffa80`07a58300  01010100 00000101 00010101 00000001
fffffa80`07a58310  01010101 01010101 01010100 00000000
kd> ba r8 rax+0xa
kd> g
Breakpoint 4 hit
termdd!memmove+0xb9:
fffff880`03e97639 4883c120        add     rcx,20h
```

So the **"A"s** are now being worked in a "memmove" function. Looking at the call stack, the "memmove" is called from the "IcaCopyDataToUserBuffer".

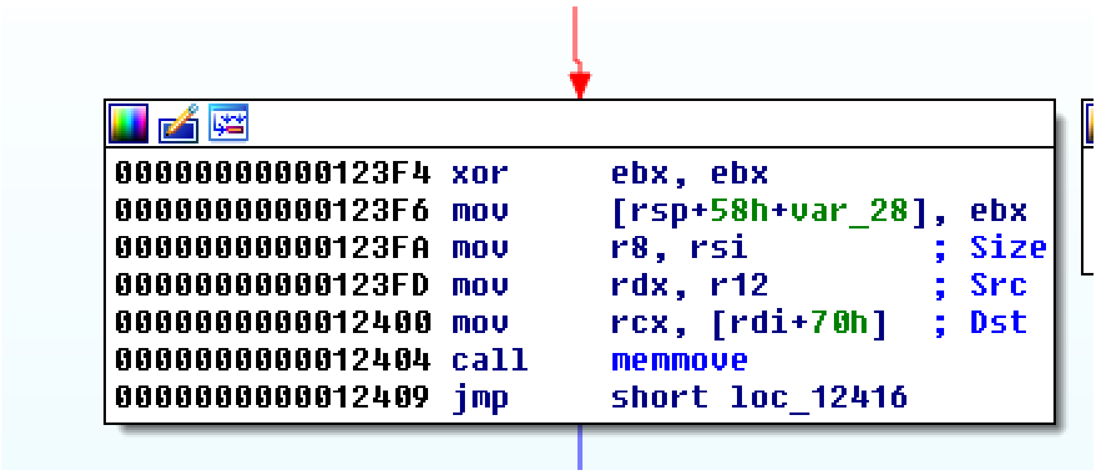


```
Breakpoint 4 hit
termdd!memmove+0xb9:
fffff880`03e97639 4883c120        add     rcx,20h
kd> kb
 # RetAddr           : Args to Child                                                           : Call Site
00 fffff880`03e90409 : fffffa80`075892c8 00000000`00000005 00000000`00000020 fffffa80`07210300 : termdd!memmove+0xb9
01 fffff880`03e91075 : fffffa80`08898c30 fffffa80`08898d00 fffffa80`00000020 00000000`00000000 : termdd!IcaCopyDataToUserBuffer+0x49
02 fffff880`03e90e01 : fffffa80`0782ddd0 00000000`00000000 fffffa80`07a58296 fffffa80`07a5828e : termdd!IcaChannelInputInternal+0x261
03 fffff880`04647f36 : fffff8a0`0334d000 00000000`00000005 fffffa80`086e9b50 00000000`00004000 : termdd!IcaChannelInput+0xdd
04 fffff880`0464f6a2 : 00000000`00000000 fffff880`03e9418f 00000000`00000002 00000000`00000028 : RDPWD!WDW_OnDataReceived+0x32e
05 fffff880`04667a98 : fffff8a0`0d418dc0 00000000`00000028 00000000`00000019 00000000`00000001 : RDPWD!SM_MCSSendDataCallback+0x1ba
06 fffff880`046663d4 : fffff880`021c2c30 fffff8a0`033231e8 00000000`00000000 00000000`00000000 : RDPWD!HandleAllSendDataPDUs+0x188
07 fffff880`04665fe4 : 00000000`00000036 fffffa80`07a58295 00000006`0000001a fffff880`0268215d : RDPWD!RecognizeMCSFrame+0x28
08 fffff880`03e941f8 : fffff8a0`0334d000 fffffa80`0782ddd0 fffffa80`0741a1e0 fffff880`02680f00 : RDPWD!MCSIcaRawInputWorker+0x3d4
09 fffff880`02680900 : 00000000`00000000 fffff880`021c2db0 fffff880`021c2da8 00000000`00000000 : termdd!IcaRawInput+0x50
0a fffff880`0267fd85 : fffff800`02801180 fffffa80`086e9b50 00000000`00000001 00000000`00000000 : tssecsrv!CRawInputDM::PassDataToServer+0x2c
0b fffff880`0267f7c2 : fffffa80`07a2d798 fffffa80`00000000 00000000`00000036 fffff800`00000000 : tssecsrv!CFilter::FilterIncomingData+0xc9
0c fffff880`03e941f8 : fffff800`02801180 00000000`00000000 00000000`00000000 00000000`00000000 : tssecsrv!ScrRawInput+0x82
0d fffff880`032994bd : fffffa80`07a2d780 fffffa80`07a58048 00000000`00000000 fffffa80`07a2d780 : termdd!IcaRawInput+0x50
0e fffff880`03e94f3e : fffffa80`07a58010 fffffa80`085a2460 fffffa80`07210300 fffffa80`0870c0e0 : tdtcp!TdInputThread+0x465
0f fffff880`03e93ae3 : fffffa80`07537fe0 fffffa80`0870c0e0 fffffa80`07744f00 fffffa80`07210300 : termdd!IcaDriverThread+0x5a
10 fffff880`03e929e9 : fffffa80`085a4ae0 fffff880`021c3818 fffff880`021c3820 00000000`00000000 : termdd!IcaDeviceControlStack+0x827
11 fffff880`03e92689 : 00000000`00000000 fffffa80`0870c0e0 00000000`00000000 fffffa80`085a4bf8 : termdd!IcaDeviceControl+0x75
12 fffff800`0290fd9a : 00000000`00000002 00000000`00000002 00000000`00000000 fffffa80`074be470 : termdd!IcaDispatch+0x215
13 fffff800`02ad5831 : fffffa80`074be470 fffffa80`074be470 fffffa80`074be470 fffff800`02801180 : nt!IopSynchronousServiceTail+0xfa
14 fffff800`029675d6 : 00000000`00000000 00000000`00000000 00000000`00000000 00000000`00000000 : nt!IopXxxControlFile+0xc51
15 fffff800`026bebd3 : 00000000`80000001 fffff800`02ab5b46 00000000`00000000 00000000`00000000 : nt!NtDeviceIoControlFile+0x56
16 00000000`77bc98fa : 000007fe`f8ae13a8 00000000`00000000 00000000`00000000 00000000`00000000 : nt!KiSystemServiceCopyEnd+0x13
```

Lets step out (gu) of the “memmove” to see where is the destination address that the **“A”s** are moving to.

```
kd> gu
Breakpoint 4 hit
termdd!memmove+0xca:
fffff880`03e9764a 4c8b540af8       mov      r10,qword ptr [rdx+rcx-8]
kd> gu
termdd!IcaCopyDataToUserBuffer+0x49:
fffff880`03e90409 eb0b             jmp      termdd!IcaCopyDataToUserBuffer+0x56 (fffff880`03e90416)
```

Which is here looking at it from the disassembler:

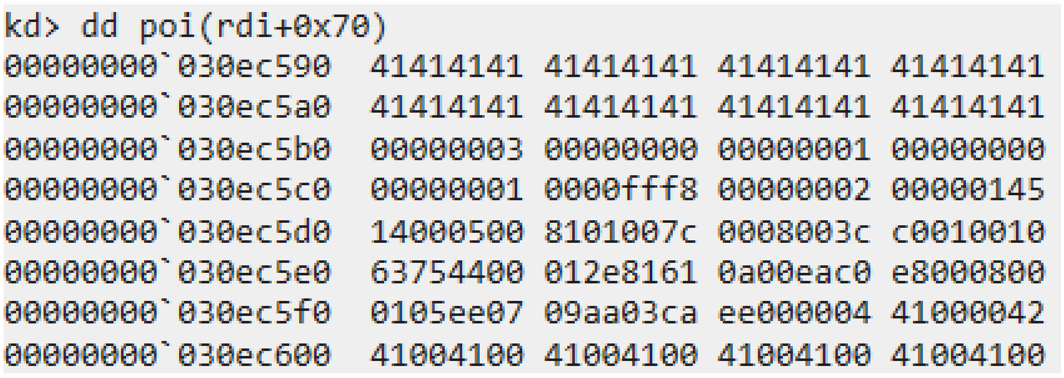


```
00000000000123F4 xor     ebx, ebx
00000000000123F6 mov     [rsp+58h+var_28], ebx
00000000000123FA mov     r8, rsi             ; Size
00000000000123FD mov     rdx, r12            ; Src
0000000000012400 mov     rcx, [rdi+70h]      ; Dst
0000000000012404 call    memmove
0000000000012409 jmp     short loc_12416
```

The values for “Src”, “Dst” and “Size” are as follow:

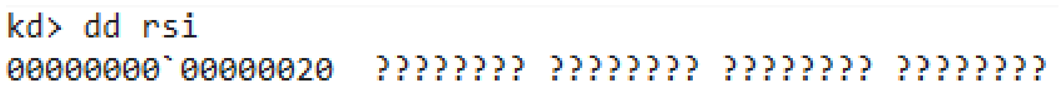
```
kd> dd r12
fffffa80`07a58296  41414141 41414141 41414141 41414141
fffffa80`07a582a6  41414141 41414141 41414141 41414141
fffffa80`07a582b6  41414141 41414141 41414141 41414141
fffffa80`07a582c6  41414141 e8ef3441 3a8016dd 6775b5cb
fffffa80`07a582d6  d28741cd 255880c9 0505054e 00050505
fffffa80`07a582e6  00000000 00000000 00000000 01000000
fffffa80`07a582f6  00001400 00000100 0100aa00 01010101
fffffa80`07a58306  01010000 00010001 01010000 01010101
```

Src



```
kd> dd poi(rdi+0x70)
00000000`030ec590  41414141 41414141 41414141 41414141
00000000`030ec5a0  41414141 41414141 41414141 41414141
00000000`030ec5b0  00000003 00000000 00000001 00000000
00000000`030ec5c0  00000001 0000fff8 00000002 00000145
00000000`030ec5d0  14000500 8101007c 0008003c c0010010
00000000`030ec5e0  63754400 012e8161 0a00eac0 e8000800
00000000`030ec5f0  0105ee07 09aa03ca ee000004 41000042
00000000`030ec600  41004100 41004100 41004100 41004100
```

Dst



```
kd> dd rsi
00000000`00000020  ???????? ???????? ???????? ????????
```

Size (0x20)

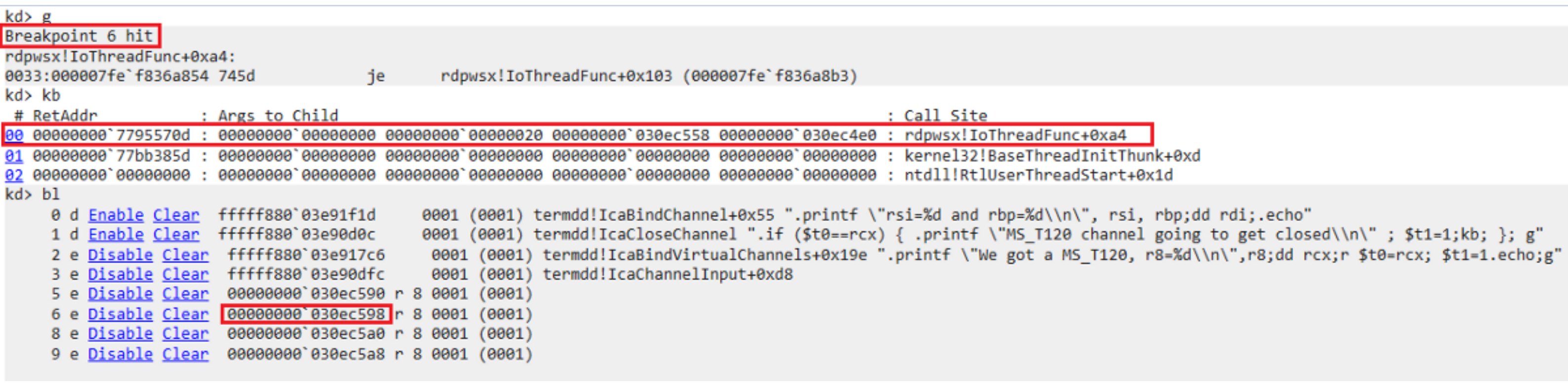
So the “memmove” copy **“A”s** from an kernel’s address space into a user’s address space.

We will now set another groups of break on read/write on the user’s address space to see how these values are ‘touched’

```
ba r8 00000000`030ec590
ba r8 00000000`030ec598
ba r8 00000000`030ec5a0
ba r8 00000000`030ec5a8
```

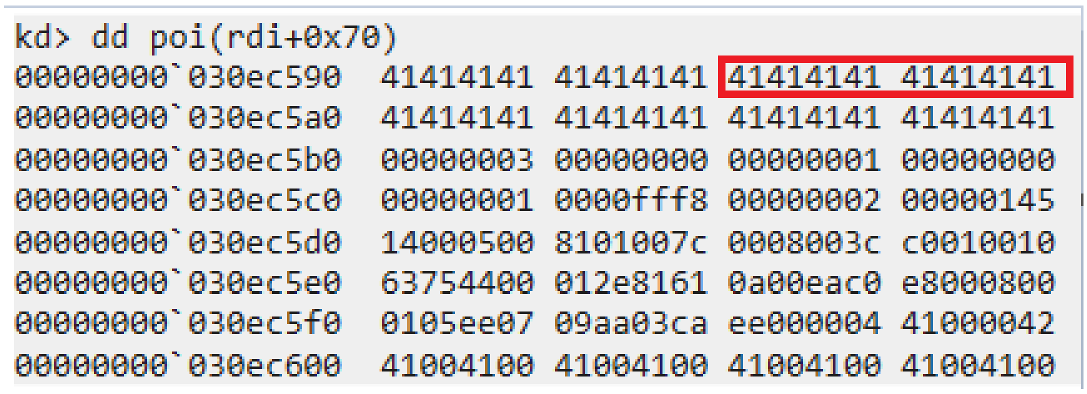
(side note: If you get a message “Too many data breakpoints for processor 0…”, remove some of the older breakpoints you set then enter “g” again)

We then get a hit on rdpwsx!IoThreadFunc:



```
kd> g
Breakpoint 6 hit
rdpwsx!IoThreadFunc+0xa4:
0033:000007fe`f836a854 745d            je      rdpwsx!IoThreadFunc+0x103 (000007fe`f836a8b3)
kd> kb
 # RetAddr           : Args to Child                                                           : Call Site
00 00000000`7795570d : 00000000`00000000 00000000`00000020 00000000`030ec558 00000000`030ec4e0 : rdpwsx!IoThreadFunc+0xa4
01 00000000`77bb385d : 00000000`00000000 00000000`00000000 00000000`00000000 00000000`00000000 : kernel32!BaseThreadInitThunk+0xd
02 00000000`00000000 : 00000000`00000000 00000000`00000000 00000000`00000000 00000000`00000000 : ntdll!RtlUserThreadStart+0x1d
kd> bl
     0 d Enable Clear  fffff880`03e91f1d     0001 (0001) termdd!IcaBindChannel+0x55 ".printf \"rsi=%d and rbp=%d\\n\", rsi, rbp;dd rdi;.echo"
     1 d Enable Clear  fffff880`03e90d0c     0001 (0001) termdd!IcaCloseChannel ".if ($t0==rcx) { .printf \"MS_T120 channel going to get closed\\n\" ; $t1=1;kb; }; g"
     2 e Disable Clear  fffff880`03e917c6     0001 (0001) termdd!IcaBindVirtualChannels+0x19e ".printf \"We got a MS_T120, r8=%d\\n\",r8;dd rcx;r $t0=rcx; $t1=1.echo;g"
     3 e Disable Clear  fffff880`03e90dfc     0001 (0001) termdd!IcaChannelInput+0xd8
     5 e Disable Clear  00000000`030ec590 r 8 0001 (0001)
     6 e Disable Clear  00000000`030ec598 r 8 0001 (0001)
     8 e Disable Clear  00000000`030ec5a0 r 8 0001 (0001)
     9 e Disable Clear  00000000`030ec5a8 r 8 0001 (0001)
```

The breakpoint touched the memory section in the highlighted red box:

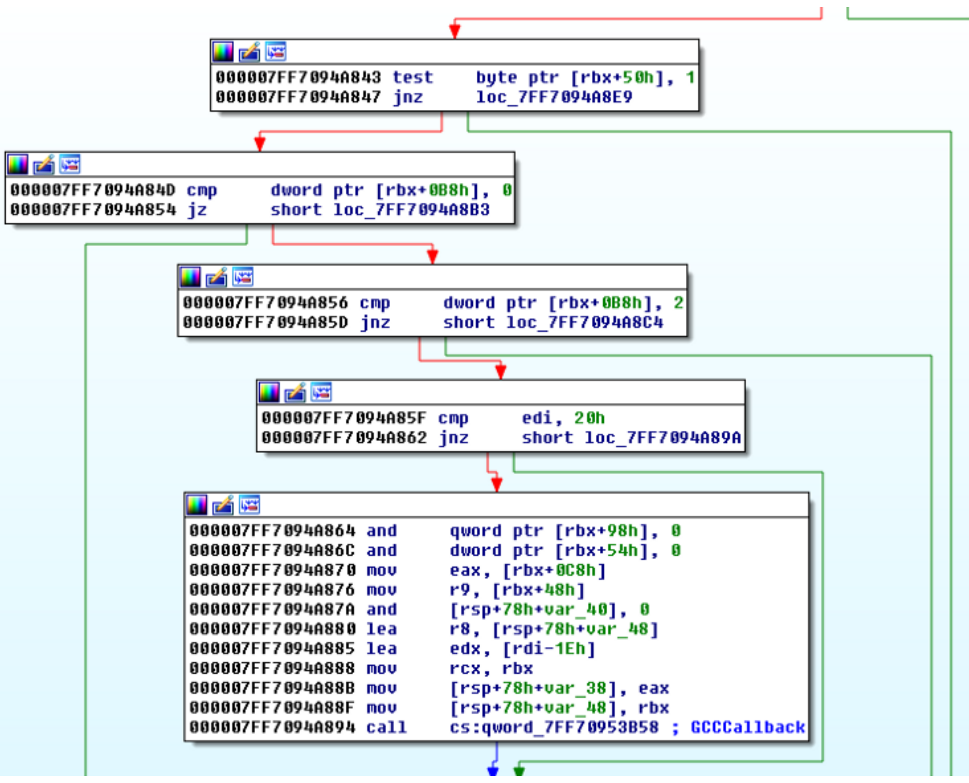


```
kd> dd poi(rdi+0x70)
00000000`030ec590  41414141 41414141 41414141 41414141
00000000`030ec5a0  41414141 41414141 41414141 41414141
00000000`030ec5b0  00000003 00000000 00000001 00000000
00000000`030ec5c0  00000001 0000fff8 00000002 00000145
00000000`030ec5d0  14000500 8101007c 0008003c c0010010
00000000`030ec5e0  63754400 012e8161 0a00eac0 e8000800
00000000`030ec5f0  0105ee07 09aa03ca ee000004 41000042
00000000`030ec600  41004100 41004100 41004100 41004100
```

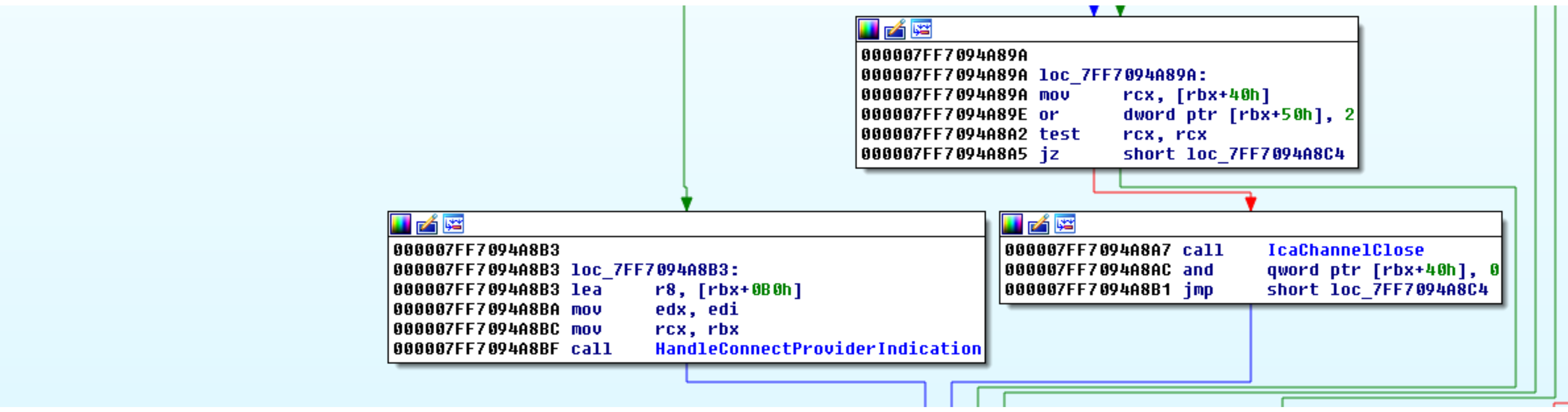
The rdpwsx!IoThreadFunc appears to be the code that parses and handle the MS_T120 data content.

```
kd> u
rdpwsx!IoThreadFunc+0xa4:
000007fe`f836a854 745d             je      rdpwsx!IoThreadFunc+0x103 (000007fe`f836a8b3)
000007fe`f836a856 83bbb800000002   cmp     dword ptr [rbx+0B8h],2
000007fe`f836a85d 7565             jne     rdpwsx!IoThreadFunc+0x114 (000007fe`f836a8c4)
000007fe`f836a85f 83ff20           cmp     edi,20h
000007fe`f836a862 7536             jne     rdpwsx!IoThreadFunc+0xea (000007fe`f836a89a)
000007fe`f836a864 4883a39800000000 and      qword ptr [rbx+98h],0
000007fe`f836a86c 83635400         and     dword ptr [rbx+54h],0
000007fe`f836a870 8b83c8000000     mov     eax,dword ptr [rbx+0C8h]
```

Using a disassembler will provide a greater view:



```
000007FF7094A843 test    byte ptr [rbx+50h], 1
000007FF7094A847 jnz     loc_7FF7094A8E9

000007FF7094A84D cmp     dword ptr [rbx+0B8h], 0
000007FF7094A854 jz      short loc_7FF7094A8B3

000007FF7094A856 cmp     dword ptr [rbx+0B8h], 2
000007FF7094A85D jnz     short loc_7FF7094A8C4

000007FF7094A85F cmp     edi, 20h
000007FF7094A862 jnz     short loc_7FF7094A89A

000007FF7094A864 and     qword ptr [rbx+98h], 0
000007FF7094A86C and     dword ptr [rbx+54h], 0
000007FF7094A870 mov     eax, [rbx+0C8h]
000007FF7094A876 mov     r9, [rbx+48h]
000007FF7094A87A and     [rsp+78h+var_40], 0
000007FF7094A880 lea     r8, [rsp+78h+var_48]
000007FF7094A885 lea     edx, [rdi-1Eh]
000007FF7094A888 mov     rcx, rbx
000007FF7094A88B mov     [rsp+78h+var_38], eax
000007FF7094A88F mov     [rsp+78h+var_48], rbx
000007FF7094A894 call    cs:qword_7FF70953B58 ; GCCCallback
```

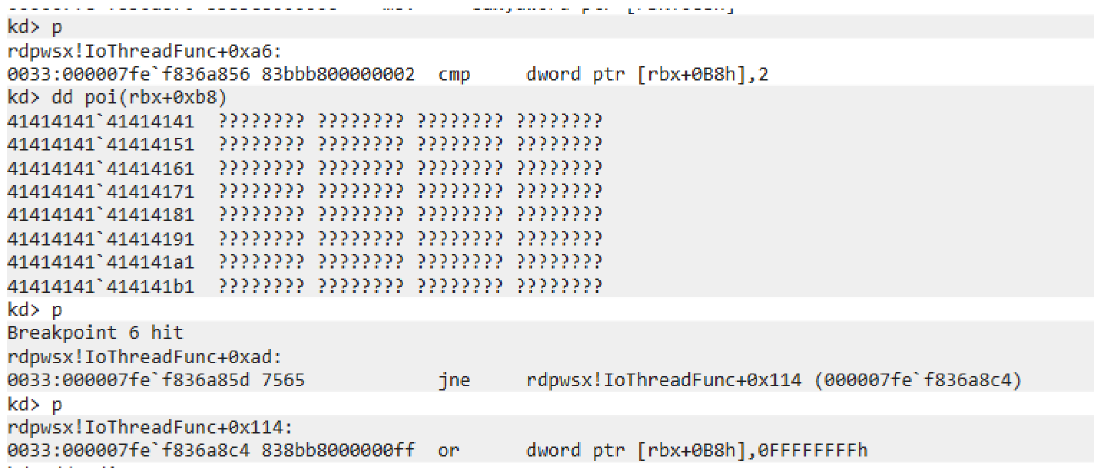


```
000007FF7094A89A
000007FF7094A89A loc_7FF7094A89A:
000007FF7094A89A mov     rcx, [rbx+40h]
000007FF7094A89E or      dword ptr [rbx+50h], 2
000007FF7094A8A2 test    rcx, rcx
000007FF7094A8A5 jz      short loc_7FF7094A8C4
```

```
000007FF7094A8B3
000007FF7094A8B3 loc_7FF7094A8B3:
000007FF7094A8B3 lea     r8, [rbx+0B0h]
000007FF7094A8BA mov     edx, edi
000007FF7094A8BC mov     rcx, rbx
000007FF7094A8BF call    HandleConnectProviderIndication
```

```
000007FF7094A8A7 call    IcaChannelClose
000007FF7094A8AC and     qword ptr [rbx+40h], 0
000007FF7094A8B1 jmp     short loc_7FF7094A8C4
```

We will now use “p” command to step over each instruction.



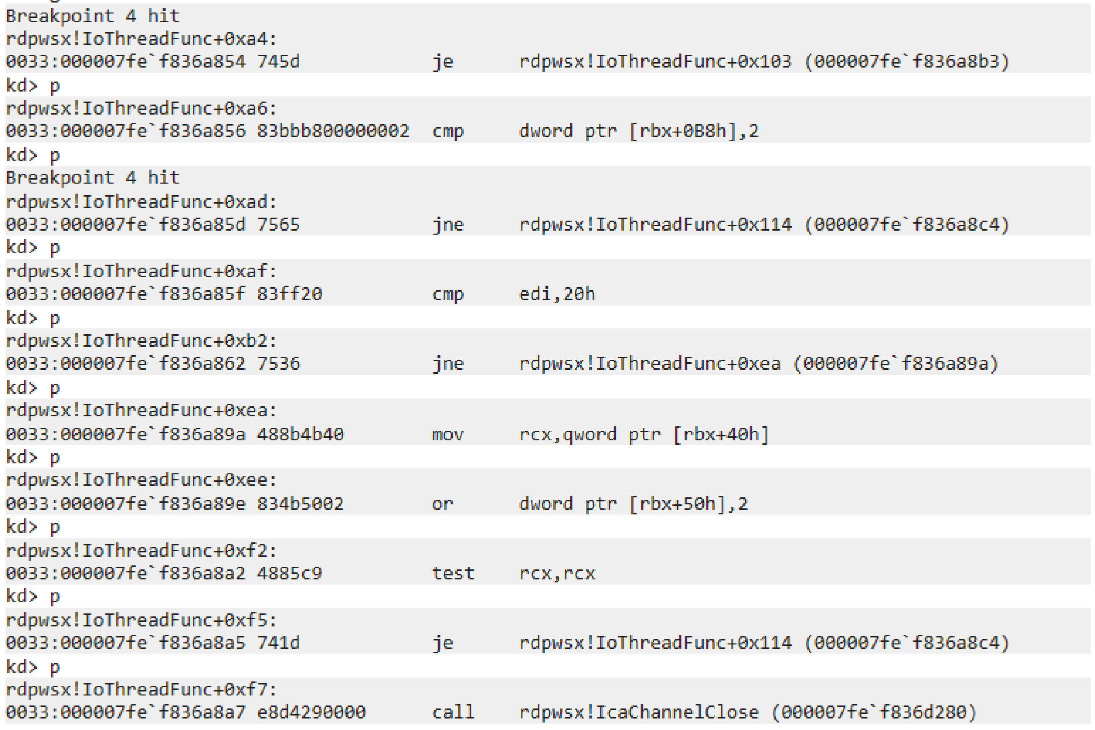
```
kd> p
rdpwsx!IoThreadFunc+0xa6:
0033:000007fe`f836a856 83bbb800000002  cmp     dword ptr [rbx+0B8h],2
kd> dd poi(rbx+0xb8)
41414141`41414141  ???????? ???????? ???????? ????????
41414141`41414151  ???????? ???????? ???????? ????????
41414141`41414161  ???????? ???????? ???????? ????????
41414141`41414171  ???????? ???????? ???????? ????????
41414141`41414181  ???????? ???????? ???????? ????????
41414141`41414191  ???????? ???????? ???????? ????????
41414141`414141a1  ???????? ???????? ???????? ????????
41414141`414141b1  ???????? ???????? ???????? ????????
kd> p
Breakpoint 6 hit
rdpwsx!IoThreadFunc+0xad:
0033:000007fe`f836a85d 7565            jne     rdpwsx!IoThreadFunc+0x114 (000007fe`f836a8c4)
kd> p
rdpwsx!IoThreadFunc+0x114:
0033:000007fe`f836a8c4 838bb8000000ff  or      dword ptr [rbx+0B8h],0FFFFFFFFh
```

It looks like because I supplied ‘AAAA’, it took a different path.

According to the blog post from ZDI, we need to send crafted data to the MS_T120 channel (over our selected index), so it will terminate the channel (free the MS_T120 channel control structure), such that when the RDPWD!SignalBrokenConnection tries to reach out to the MS_T120 channel again over index 31 from the channel pointer structure, it will Use a Freed MS_T120 channel control structure, leading to the crash.

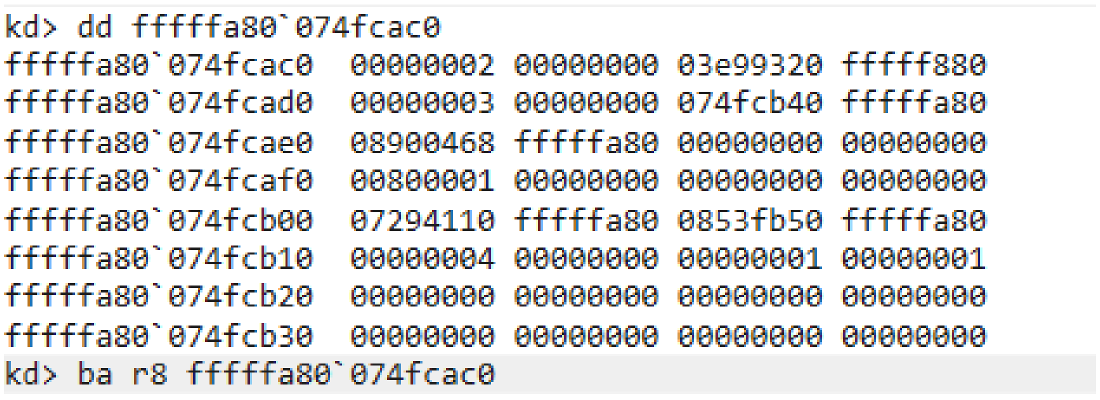
Based on the rdpwsx!IoThreadFunc, it appears to make sense to create crafted data that will hit the IcaChannelClose function.

When the crafted data is correct, it will hit the rdpwsx!IcaChannelClose



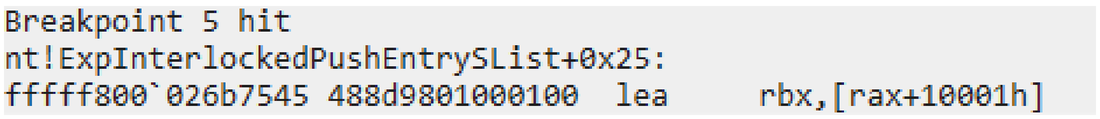
```
Breakpoint 4 hit
rdpwsx!IoThreadFunc+0xa4:
0033:000007fe`f836a854 745d            je      rdpwsx!IoThreadFunc+0x103 (000007fe`f836a8b3)
kd> p
rdpwsx!IoThreadFunc+0xa6:
0033:000007fe`f836a856 83bbb800000002  cmp     dword ptr [rbx+0B8h],2
kd> p
Breakpoint 4 hit
rdpwsx!IoThreadFunc+0xad:
0033:000007fe`f836a85d 7565            jne     rdpwsx!IoThreadFunc+0x114 (000007fe`f836a8c4)
kd> p
rdpwsx!IoThreadFunc+0xaf:
0033:000007fe`f836a85f 83ff20          cmp     edi,20h
kd> p
rdpwsx!IoThreadFunc+0xb2:
0033:000007fe`f836a862 7536            jne     rdpwsx!IoThreadFunc+0xea (000007fe`f836a89a)
kd> p
rdpwsx!IoThreadFunc+0xea:
0033:000007fe`f836a89a 488b4b40        mov     rcx,qword ptr [rbx+40h]
kd> p
rdpwsx!IoThreadFunc+0xee:
0033:000007fe`f836a89e 834b5002        or      dword ptr [rbx+50h],2
kd> p
rdpwsx!IoThreadFunc+0xf2:
0033:000007fe`f836a8a2 4885c9          test    rcx,rcx
kd> p
rdpwsx!IoThreadFunc+0xf5:
0033:000007fe`f836a8a5 741d            je      rdpwsx!IoThreadFunc+0x114 (000007fe`f836a8c4)
kd> p
rdpwsx!IoThreadFunc+0xf7:
0033:000007fe`f836a8a7 e8d4290000      call    rdpwsx!IcaChannelClose (000007fe`f836d280)
```

Before stepping through the IcaChannelClose, lets set a breakpoint on the MS_T120 control channel structure to see how does it get affected



```
kd> dd fffffa80`074fcac0
fffffa80`074fcac0  00000002 00000000 03e99320 fffff880
fffffa80`074fcad0  00000003 00000000 074fcb40 fffffa80
fffffa80`074fcae0  08900468 fffffa80 00000000 00000000
fffffa80`074fcaf0  00800001 00000000 00000000 00000000
fffffa80`074fcb00  07294110 fffffa80 0853fb50 fffffa80
fffffa80`074fcb10  00000004 00000000 00000001 00000001
fffffa80`074fcb20  00000000 00000000 00000000 00000000
fffffa80`074fcb30  00000000 00000000 00000000 00000000
kd> ba r8 fffffa80`074fcac0
```

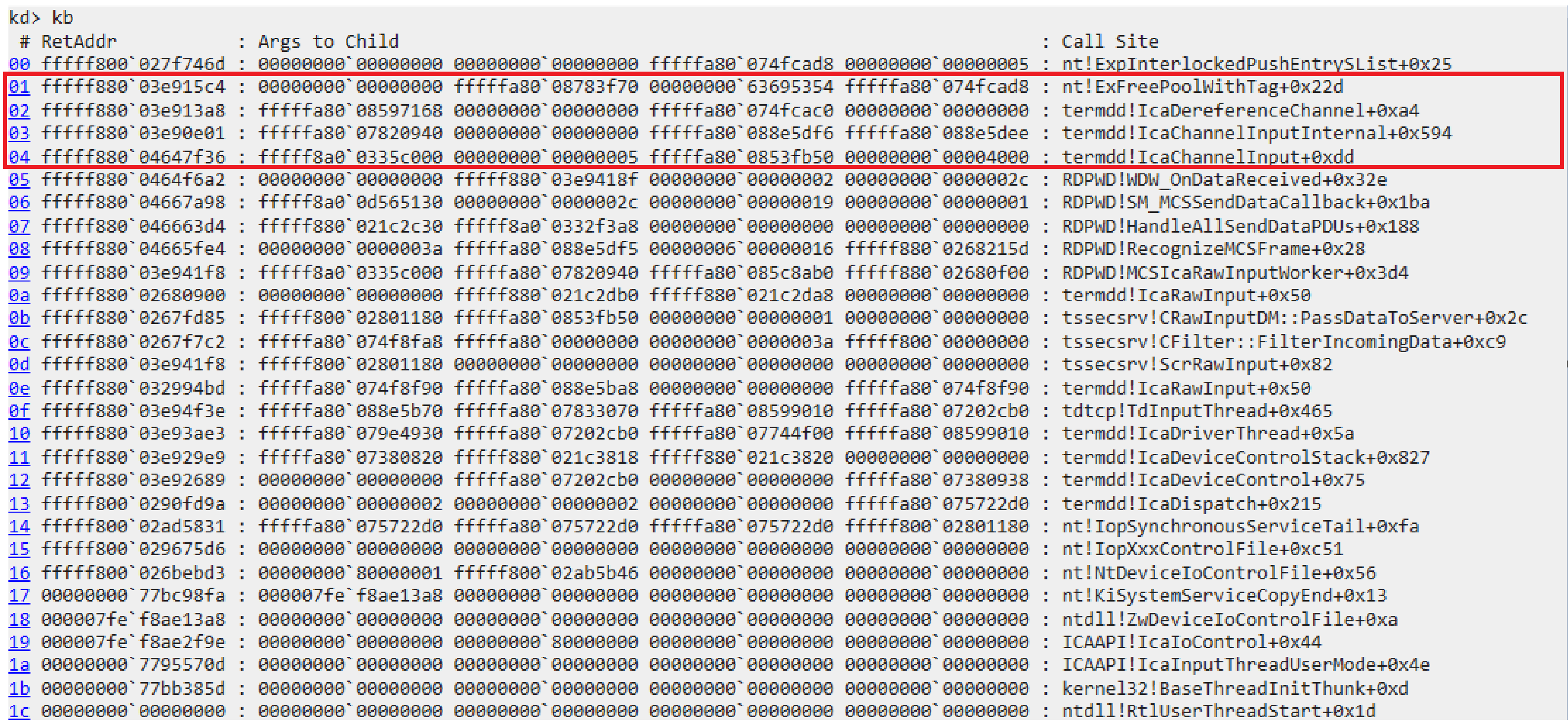
fffffa80`074fcac0 is the current address for the MS_T120 structure



```
Breakpoint 5 hit
nt!ExpInterlockedPushEntrySList+0x25:
fffff800`026b7545 488d9801000100  lea     rbx,[rax+10001h]
```

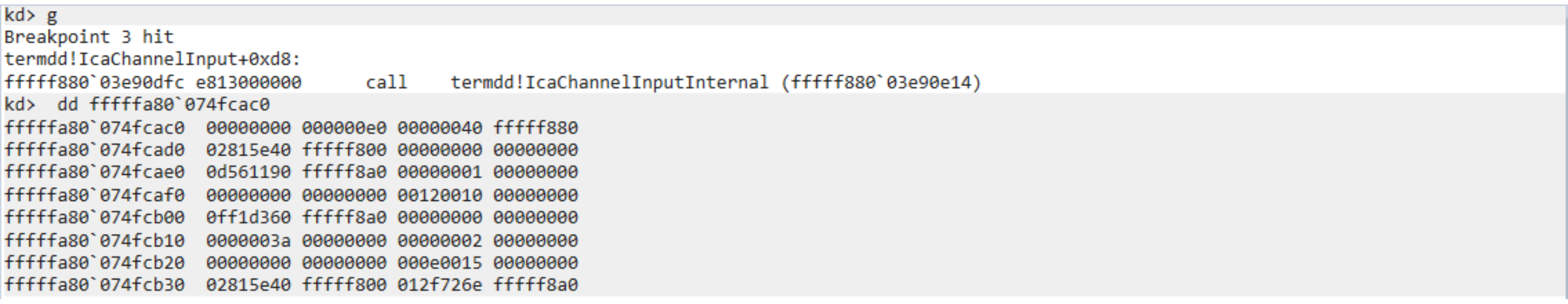
A breakpoint read is hit on fffffa80`074fcac0

The following picture shows the call stack when the breakpoint read is hit. A call is made to ExFreePoolWithTag, which frees the MS_T120 channel control structure.



```
kd> kb
 # RetAddr           : Args to Child                                                           : Call Site
00 fffff800`027f746d : 00000000`00000000 00000000`00000000 fffffa80`074fcad8 00000000`00000005 : nt!ExpInterlockedPushEntrySList+0x25
01 fffff880`03e915c4 : 00000000`00000000 fffffa80`08783f70 00000000`63695354 fffffa80`074fcad8 : nt!ExFreePoolWithTag+0x22d
02 fffff880`03e913a8 : fffffa80`08597168 00000000`00000000 fffffa80`074fcac0 00000000`00000000 : termdd!IcaDereferenceChannel+0xa4
03 fffff880`03e90e01 : fffffa80`07820940 00000000`00000000 fffffa80`088e5df6 fffffa80`088e5dee : termdd!IcaChannelInputInternal+0x594
04 fffff880`04647f36 : fffff8a0`0335c000 00000000`00000005 fffffa80`0853fb50 00000000`00004000 : termdd!IcaChannelInput+0xdd
05 fffff880`0464f6a2 : 00000000`00000000 fffff880`03e9418f 00000000`00000002 00000000`0000002c : RDPWD!WDW_OnDataReceived+0x32e
06 fffff880`04667a98 : fffff8a0`0d565130 00000000`0000002c 00000000`00000019 00000000`00000001 : RDPWD!SM_MCSSendDataCallback+0x1ba
07 fffff880`046663d4 : fffff880`021c2c30 fffff8a0`0332f3a8 00000000`00000000 00000000`00000000 : RDPWD!HandleAllSendDataPDUs+0x188
08 fffff880`04665fe4 : 00000000`0000003a fffffa80`088e5df5 00000006`00000016 fffff880`0268215d : RDPWD!RecognizeMCSFrame+0x28
09 fffff880`03e941f8 : fffff8a0`0335c000 fffffa80`07820940 fffffa80`085c8ab0 fffff880`02680f00 : RDPWD!MCSIcaRawInputWorker+0x3d4
0a fffff880`02680900 : 00000000`00000000 fffff880`021c2db0 fffff880`021c2da8 00000000`00000000 : termdd!IcaRawInput+0x50
0b fffff880`0267fd85 : fffff800`02801180 fffffa80`0853fb50 00000000`00000001 00000000`00000000 : tssecsrv!CRawInputDM::PassDataToServer+0x2c
0c fffff880`0267f7c2 : fffffa80`074f8fa8 fffffa80`00000000 00000000`0000003a fffff800`00000000 : tssecsrv!CFilter::FilterIncomingData+0xc9
0d fffff880`03e941f8 : fffff800`02801180 00000000`00000000 00000000`00000000 00000000`00000000 : tssecsrv!ScrRawInput+0x82
0e fffff880`032994bd : fffffa80`074f8f90 fffffa80`088e5ba8 00000000`00000000 fffffa80`074f8f90 : termdd!IcaRawInput+0x50
0f fffff880`03e94f3e : fffffa80`088e5b70 fffffa80`07833070 fffffa80`08599010 fffffa80`07202cb0 : tdtcp!TdInputThread+0x465
10 fffff880`03e93ae3 : fffffa80`079e4930 fffffa80`07202cb0 fffffa80`07744f00 fffffa80`08599010 : termdd!IcaDriverThread+0x5a
11 fffff880`03e929e9 : fffffa80`07380820 fffff880`021c3818 fffff880`021c3820 00000000`00000000 : termdd!IcaDeviceControlStack+0x827
12 fffff880`03e92689 : 00000000`00000000 fffffa80`07202cb0 00000000`00000000 fffffa80`07380938 : termdd!IcaDeviceControl+0x75
13 fffff800`0290fd9a : 00000000`00000002 00000000`00000002 00000000`00000000 fffffa80`075722d0 : termdd!IcaDispatch+0x215
14 fffff800`02ad5831 : fffffa80`075722d0 fffffa80`075722d0 fffffa80`075722d0 fffff800`02801180 : nt!IopSynchronousServiceTail+0xfa
15 fffff800`029675d6 : 00000000`00000000 00000000`00000000 00000000`00000000 00000000`00000000 : nt!IopXxxControlFile+0xc51
16 fffff800`026bebd3 : 00000000`80000001 fffff800`02ab5b46 00000000`00000000 00000000`00000000 : nt!NtDeviceIoControlFile+0x56
17 00000000`77bc98fa : 000007fe`f8ae13a8 00000000`00000000 00000000`00000000 00000000`00000000 : nt!KiSystemServiceCopyEnd+0x13
18 000007fe`f8ae13a8 : 00000000`00000000 00000000`00000000 00000000`00000000 00000000`00000000 : ntdll!ZwDeviceIoControlFile+0xa
19 000007fe`f8ae2f9e : 00000000`00000000 00000000`80000000 00000000`00000000 00000000`00000000 : ICAAPI!IcaIoControl+0x44
1a 00000000`7795570d : 00000000`00000000 00000000`00000000 00000000`00000000 00000000`00000000 : ICAAPI!IcaInputThreadUserMode+0x4e
1b 00000000`77bb385d : 00000000`00000000 00000000`00000000 00000000`00000000 00000000`00000000 : kernel32!BaseThreadInitThunk+0xd
1c 00000000`00000000 : 00000000`00000000 00000000`00000000 00000000`00000000 00000000`00000000 : ntdll!RtlUserThreadStart+0x1d
```

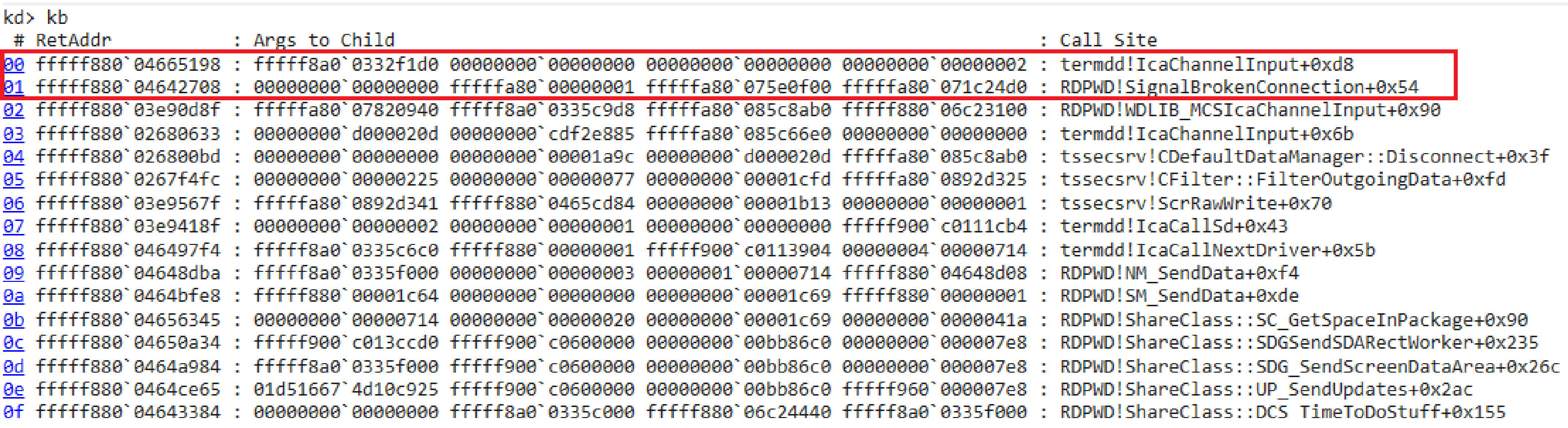
We can proceed with “g” until we hit the breakpoint in termdd!IcaChannelInput:



```
kd> g
Breakpoint 3 hit
termdd!IcaChannelInput+0xd8:
fffff880`03e90dfc e813000000      call    termdd!IcaChannelInputInternal (fffff880`03e90e14)
kd>  dd fffffa80`074fcac0
fffffa80`074fcac0  00000000 000000e0 00000040 fffff880
fffffa80`074fcad0  02815e40 fffff800 00000000 00000000
fffffa80`074fcae0  0d561190 fffff8a0 00000001 00000000
fffffa80`074fcaf0  00000000 00000000 00120010 00000000
fffffa80`074fcb00  0ff1d360 fffff8a0 00000000 00000000
fffffa80`074fcb10  0000003a 00000000 00000002 00000000
fffffa80`074fcb20  00000000 00000000 000e0015 00000000
fffffa80`074fcb30  02815e40 fffff800 012f726e fffff8a0
```

Taking a look at the address that holds the MS_T120 channel control structure, the content looks pretty different.

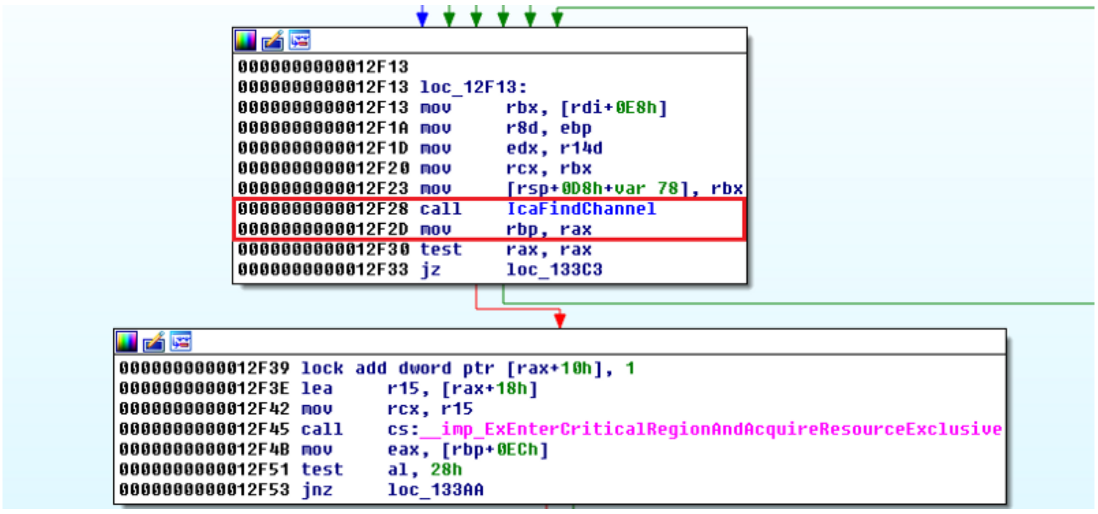
Furthermore, the call stack shows the call to IcaChannelInput comes from RDPWD!SignalBrokenConnection. The ZDI blog noted this function gets called when the connection terminates.



```
kd> kb
 # RetAddr           : Args to Child                                                           : Call Site
00 fffff880`04665198 : fffff8a0`0332f1d0 00000000`00000000 00000000`00000000 00000000`00000002 : termdd!IcaChannelInput+0xd8
01 fffff880`04642708 : 00000000`00000000 fffffa80`00000001 fffffa80`075e0f00 fffffa80`071c24d0 : RDPWD!SignalBrokenConnection+0x54
02 fffff880`03e90d8f : fffffa80`07820940 fffff8a0`0335c9d8 fffffa80`085c8ab0 fffff880`06c23100 : RDPWD!WDLIB_MCSIcaChannelInput+0x90
03 fffff880`02680633 : 00000000`d000020d 00000000`cdf2e885 fffffa80`085c66e0 00000000`00000000 : termdd!IcaChannelInput+0x6b
04 fffff880`026800bd : 00000000`00000000 00000000`00001a9c 00000000`d000020d fffffa80`085c8ab0 : tssecsrv!CDefaultDataManager::Disconnect+0x3f
05 fffff880`0267f4fc : 00000000`00000225 00000000`00000077 00000000`00001cfd fffffa80`0892d325 : tssecsrv!CFilter::FilterOutgoingData+0xfd
06 fffff880`03e9567f : fffffa80`0892d341 fffff880`0465cd84 00000000`00001b13 00000000`00000001 : tssecsrv!ScrRawWrite+0x70
07 fffff880`03e9418f : 00000000`00000002 00000000`00000001 00000000`00000000 fffff900`c0111cb4 : termdd!IcaCallSd+0x43
08 fffff880`046497f4 : fffff8a0`0335c6c0 fffff880`00000001 fffff900`c0113904 00000004`00000714 : termdd!IcaCallNextDriver+0x5b
09 fffff880`04648dba : fffff8a0`0335f000 00000000`00000003 00000001`00000714 fffff880`04648d08 : RDPWD!NM_SendData+0xf4
0a fffff880`0464bfe8 : fffff880`00001c64 00000000`00000000 00000000`00001c69 fffff880`00000001 : RDPWD!SM_SendData+0xde
0b fffff880`04656345 : 00000000`00000714 00000000`00000020 00000000`00001c69 00000000`0000041a : RDPWD!ShareClass::SC_GetSpaceInPackage+0x90
0c fffff880`04650a34 : fffff900`c013ccd0 fffff900`c0600000 00000000`00bb86c0 00000000`000007e8 : RDPWD!ShareClass::SDGSendSDARectWorker+0x235
0d fffff880`0464a984 : fffff8a0`0335f000 fffff900`c0600000 00000000`00bb86c0 00000000`000007e8 : RDPWD!ShareClass::SDG_SendScreenDataArea+0x26c
0e fffff880`0464ce65 : 01d51667`4d10c925 fffff900`c0600000 00000000`00bb86c0 fffff960`000007e8 : RDPWD!ShareClass::UP_SendUpdates+0x2ac
0f fffff880`04643384 : 00000000`00000000 fffff8a0`0335c000 fffff880`06c24440 fffff8a0`0335f000 : RDPWD!ShareClass::DCS TimeToDoStuff+0x155
```

We will use "t" command to step into the IcaChannelInputInternal function. Once we're inside the function, we will set a new breakpoint:

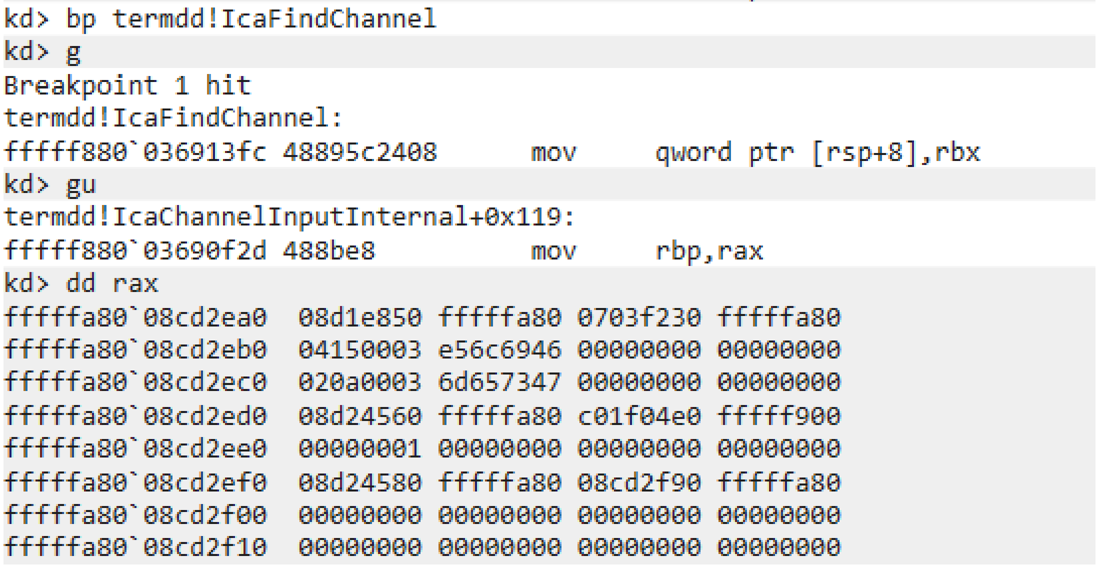
```
bp termdd!IcaFindChannel
```



```
0000000000012F13
0000000000012F13 loc_12F13:
0000000000012F13 mov     rbx, [rdi+0E8h]
0000000000012F1A mov     r8d, ebp
0000000000012F1D mov     edx, r14d
0000000000012F20 mov     rcx, rbx
0000000000012F23 mov     [rsp+0D8h+var_78], rbx
0000000000012F28 call    IcaFindChannel
0000000000012F2D mov     rbp, rax
0000000000012F30 test    rax, rax
0000000000012F33 jz      loc_133C3
```

```
0000000000012F39 lock add dword ptr [rax+10h], 1
0000000000012F3E lea     r15, [rax+18h]
0000000000012F42 mov     rcx, r15
0000000000012F45 call    cs:__imp_ExEnterCriticalRegionAndAcquireResourceExclusive
0000000000012F4B mov     eax, [rbp+0ECh]
0000000000012F51 test    al, 28h
0000000000012F53 jnz     loc_133AA
```

Once we're inside the IcaFindChannel function, use "gu" to step out of it to return back to the IcaChannelInputInternal function:

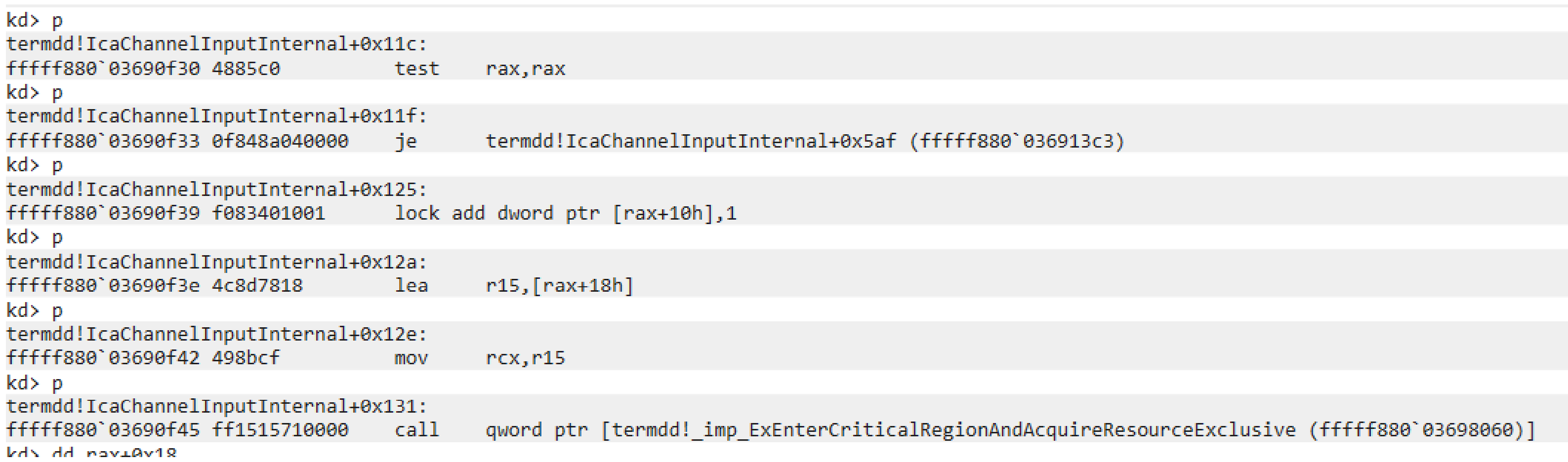


```
kd> bp termdd!IcaFindChannel
kd> g
Breakpoint 1 hit
termdd!IcaFindChannel:
fffff880`036913fc 48895c2408      mov     qword ptr [rsp+8],rbx
kd> gu
termdd!IcaChannelInputInternal+0x119:
fffff880`03690f2d 488be8          mov     rbp,rax
kd> dd rax
fffffa80`08cd2ea0  08d1e850 fffffa80 0703f230 fffffa80
fffffa80`08cd2eb0  04150003 e56c6946 00000000 00000000
fffffa80`08cd2ec0  020a0003 6d657347 00000000 00000000
fffffa80`08cd2ed0  08d24560 fffffa80 c01f04e0 fffff900
fffffa80`08cd2ee0  00000001 00000000 00000000 00000000
fffffa80`08cd2ef0  08d24580 fffffa80 08cd2f90 fffffa80
fffffa80`08cd2f00  00000000 00000000 00000000 00000000
fffffa80`08cd2f10  00000000 00000000 00000000 00000000
```

The MS_T120 object address is different to other MS_T120 object shown above, as these images are taken aross different debugging session

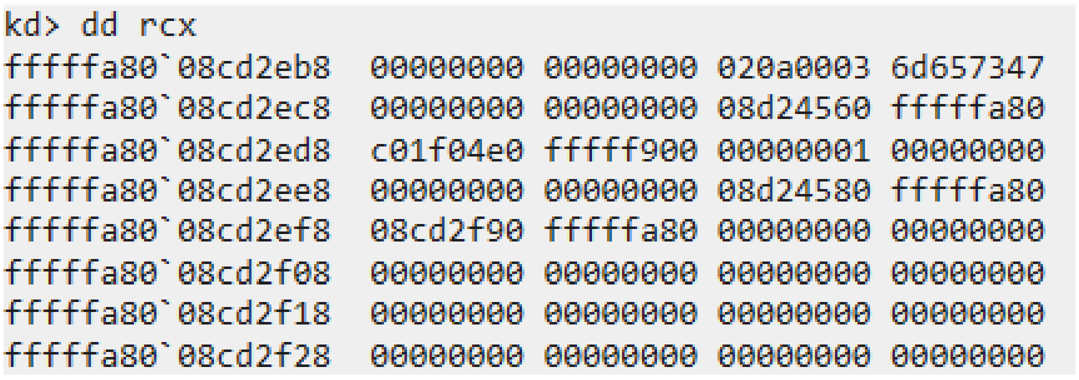
The rax registers holds the reference to the freed MS_T120 control channel structure.

As we continue to step through the code, the address at MS_T120+0x18 is being used as an parameter (rcx) to the ExEnterCriticalRegionAndAcquireResourceExclusive function.



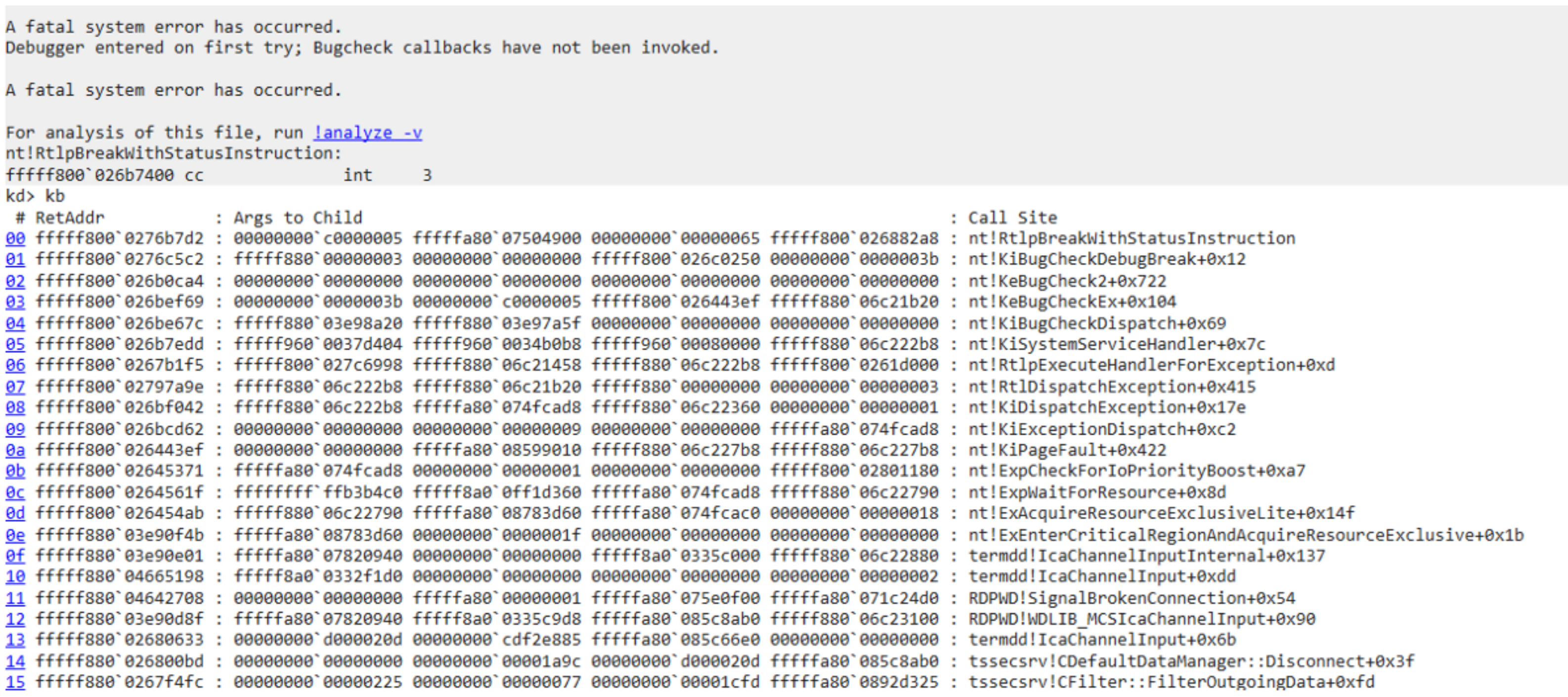
```
kd> p
termdd!IcaChannelInputInternal+0x11c:
fffff880`03690f30 4885c0          test    rax,rax
kd> p
termdd!IcaChannelInputInternal+0x11f:
fffff880`03690f33 0f848a040000    je      termdd!IcaChannelInputInternal+0x5af (fffff880`036913c3)
kd> p
termdd!IcaChannelInputInternal+0x125:
fffff880`03690f39 f083401001      lock add dword ptr [rax+10h],1
kd> p
termdd!IcaChannelInputInternal+0x12a:
fffff880`03690f3e 4c8d7818        lea     r15,[rax+18h]
kd> p
termdd!IcaChannelInputInternal+0x12e:
fffff880`03690f42 498bcf          mov     rcx,r15
kd> p
termdd!IcaChannelInputInternal+0x131:
fffff880`03690f45 ff1515710000    call    qword ptr [termdd!_imp_ExEnterCriticalRegionAndAcquireResourceExclusive (fffff880`03698060)]
kd> dd rax+0x18
```

Lets take a look at rcx:



```
kd> dd rcx
fffffa80`08cd2eb8  00000000 00000000 020a0003 6d657347
fffffa80`08cd2ec8  00000000 00000000 08d24560 fffffa80
fffffa80`08cd2ed8  c01f04e0 fffff900 00000001 00000000
fffffa80`08cd2ee8  00000000 00000000 08d24580 fffffa80
fffffa80`08cd2ef8  08cd2f90 fffffa80 00000000 00000000
fffffa80`08cd2f08  00000000 00000000 00000000 00000000
fffffa80`08cd2f18  00000000 00000000 00000000 00000000
fffffa80`08cd2f28  00000000 00000000 00000000 00000000
```

And there we go, if we dereference rcx, it is nothing! So lets step over ExEnterCriticalRegionAndAcquireResourceExclusive and see the result:



```
A fatal system error has occurred.
Debugger entered on first try; Bugcheck callbacks have not been invoked.

A fatal system error has occurred.

For analysis of this file, run !analyze -v
nt!RtlpBreakWithStatusInstruction:
fffff800`026b7400 cc              int     3
kd> kb
 # RetAddr           : Args to Child                                                           : Call Site
00 fffff800`0276b7d2 : 00000000`c0000005 fffffa80`07504900 00000000`00000065 fffff800`026882a8 : nt!RtlpBreakWithStatusInstruction
01 fffff800`0276c5c2 : fffff880`00000003 00000000`00000000 fffff800`026c0250 00000000`0000003b : nt!KiBugCheckDebugBreak+0x12
02 fffff800`026b0ca4 : 00000000`00000000 00000000`00000000 00000000`00000000 00000000`00000000 : nt!KeBugCheck2+0x722
03 fffff800`026bef69 : 00000000`0000003b 00000000`c0000005 fffff800`026443ef fffff880`06c21b20 : nt!KeBugCheckEx+0x104
04 fffff800`026be67c : fffff880`03e98a20 fffff880`03e97a5f 00000000`00000000 00000000`00000000 : nt!KiBugCheckDispatch+0x69
05 fffff800`026b7edd : fffff960`0037d404 fffff960`0034b0b8 fffff960`00080000 fffff880`06c222b8 : nt!KiSystemServiceHandler+0x7c
06 fffff800`0267b1f5 : fffff800`027c6998 fffff880`06c21458 fffff880`06c222b8 fffff800`0261d000 : nt!RtlpExecuteHandlerForException+0xd
07 fffff800`02797a9e : fffff880`06c222b8 fffff880`06c21b20 fffff880`00000000 00000000`00000003 : nt!RtlDispatchException+0x415
08 fffff800`026bf042 : fffff880`06c222b8 fffffa80`074fcad8 fffff880`06c22360 00000000`00000001 : nt!KiDispatchException+0x17e
09 fffff800`026bcd62 : 00000000`00000000 00000000`00000009 00000000`00000000 fffffa80`074fcad8 : nt!KiExceptionDispatch+0xc2
0a fffff800`026443ef : 00000000`00000000 fffffa80`08599010 fffff880`06c227b8 fffff880`06c227b8 : nt!KiPageFault+0x422
0b fffff800`02645371 : fffffa80`074fcad8 00000000`00000001 00000000`00000000 fffff800`02801180 : nt!ExpCheckForIoPriorityBoost+0xa7
0c fffff800`0264561f : ffffffff`ffb3b4c0 fffff8a0`0ff1d360 fffffa80`074fcad8 fffff880`06c22790 : nt!ExpWaitForResource+0x8d
0d fffff800`026454ab : fffff880`06c22790 fffffa80`08783d60 fffffa80`074fcac0 00000000`00000018 : nt!ExAcquireResourceExclusiveLite+0x14f
0e fffff880`03e90f4b : fffffa80`08783d60 00000000`0000001f 00000000`00000000 00000000`00000000 : nt!ExEnterCriticalRegionAndAcquireResourceExclusive+0x1b
0f fffff880`03e90e01 : fffffa80`07820940 00000000`00000000 fffff8a0`0335c000 fffff880`06c22880 : termdd!IcaChannelInputInternal+0x137
10 fffff880`04665198 : fffff8a0`0332f1d0 00000000`00000000 00000000`00000000 00000000`00000002 : termdd!IcaChannelInput+0xdd
11 fffff880`04642708 : 00000000`00000000 fffffa80`00000001 fffffa80`075e0f00 fffffa80`071c24d0 : RDPWD!SignalBrokenConnection+0x54
12 fffff880`03e90d8f : fffffa80`07820940 fffff8a0`0335c9d8 fffffa80`085c8ab0 fffff880`06c23100 : RDPWD!WDLIB_MCSIcaChannelInput+0x90
13 fffff880`02680633 : 00000000`d000020d 00000000`cdf2e885 fffffa80`085c66e0 00000000`00000000 : termdd!IcaChannelInput+0x6b
14 fffff880`026800bd : 00000000`00000000 00000000`00001a9c 00000000`d000020d fffffa80`085c8ab0 : tssecsrv!CDefaultDataManager::Disconnect+0x3f
15 fffff880`0267f4fc : 00000000`00000225 00000000`00000077 00000000`00001cfd fffffa80`0892d325 : tssecsrv!CFilter::FilterOutgoingData+0xfd
```