This work was originally done on Windows 7 Ultimate SP1 64-bit.

The versions of the libraries used in the tutorial are:

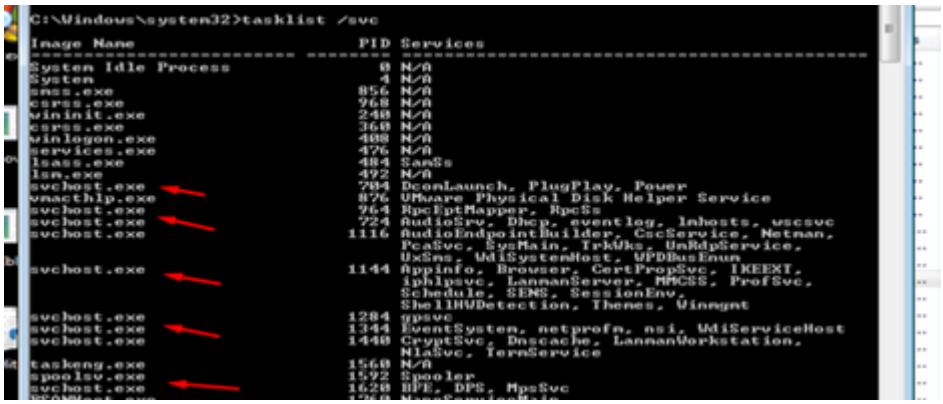
- termdd.sys version 6.1.7601.17514
- rdpwsx.dll version 6.1.7601.17828
- rdpwd.sys version 6.1.7601.17830
- icaapi.dll version 6.1.7600.16385
- rdpcorekmts.dll version 6.1.7601.17828

The Svchost.exe process

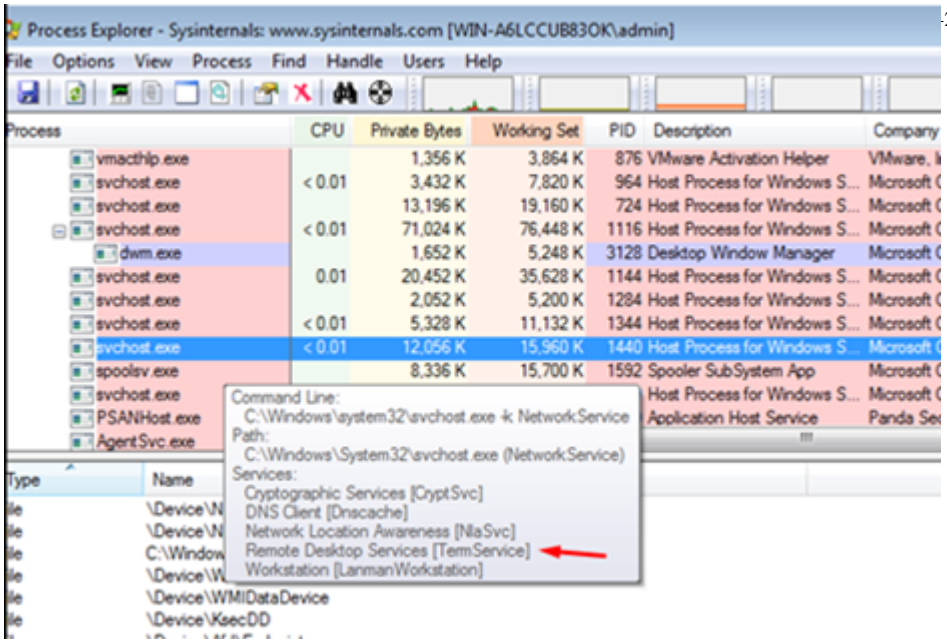
In the Windows NT operating system family, svchost.exe ('Service Host) is a system process that serves or hosts multiple Windows services.

It runs on multiple instances, each hosting one or more services. It's indispensable in the execution of so-called shared services processes, where a grouping of services can share processes in order to reduce the use of system resources.

The tasklist /svc command on a console with administrator permission shows us the different svchost processes and their associated services.

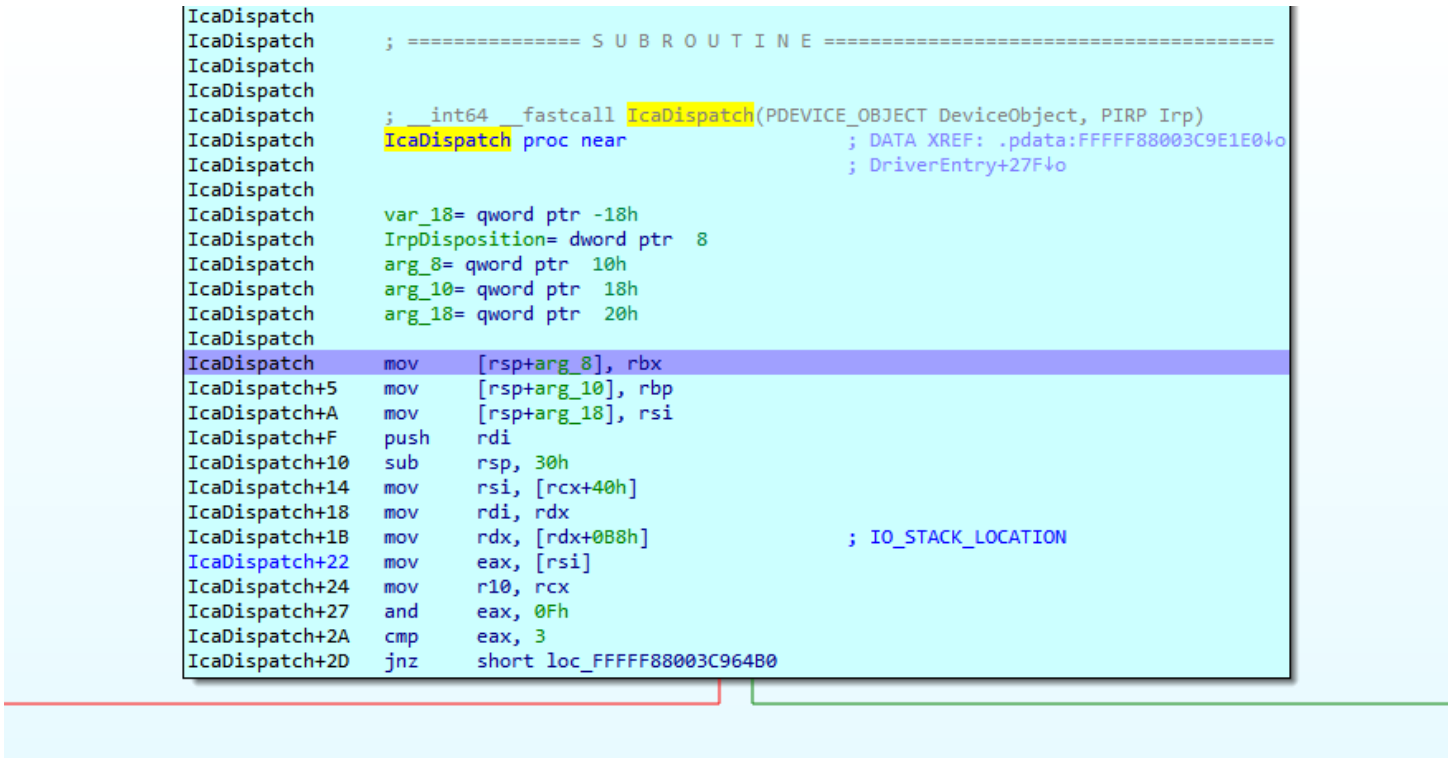


Also in PROCESS EXPLORER you can easily identify which of the SVChosts is the one that handles RDP connections.(Remote Desktop Services)



STEP 1) Initial reversing to find the point where the program starts to parse my data decrypted

The first thing we'll do is try to see where the driver is called from, for that, once we're debugging the remote kernel with Windbg or IDA, we put a breakpoint in the driver dispatch i.e. in the IcaDispatch function of termdd.sys.



```
IcaDispatch
IcaDispatch      ; =============== S U B R O U T I N E =======================================
IcaDispatch
IcaDispatch
IcaDispatch      ; __int64 __fastcall IcaDispatch(PDEVICE_OBJECT DeviceObject, PIRP Irp)
IcaDispatch      IcaDispatch proc near                ; DATA XREF: .pdata:FFFFF88003C9E1E0↓o
IcaDispatch                                           ; DriverEntry+27F↓o
IcaDispatch
IcaDispatch      var_18= qword ptr -18h
IcaDispatch      IrpDisposition= dword ptr  8
IcaDispatch      arg_8= qword ptr  10h
IcaDispatch      arg_10= qword ptr  18h
IcaDispatch      arg_18= qword ptr  20h
IcaDispatch
IcaDispatch      mov     [rsp+arg_8], rbx
IcaDispatch+5    mov     [rsp+arg_10], rbp
IcaDispatch+A    mov     [rsp+arg_18], rsi
IcaDispatch+F    push    rdi
IcaDispatch+10   sub     rsp, 30h
IcaDispatch+14   mov     rsi, [rcx+40h]
IcaDispatch+18   mov     rdi, rdx
IcaDispatch+1B   mov     rdx, [rdx+0B8h]              ; IO_STACK_LOCATION
IcaDispatch+22   mov     eax, [rsi]
IcaDispatch+24   mov     r10, rcx
IcaDispatch+27   and     eax, 0Fh
IcaDispatch+2A   cmp     eax, 3
IcaDispatch+2D   jnz     short loc_FFFFF88003C964B0
```

In windbg bar I type

.reload /f

!process 1 0

PROCESS ffffa8006598b30

SessionId: 0 Cid: **0594** Peb: 7ffffd7000 ParentCid: 01d4

DirBase: 108706000 ObjectTable: ffff8a000f119a0 HandleCount: 662.

Image: svchost.exe

The call stack is

WINDBG>**k**

Child-SP RetAddr Call Site

ffff880`05c14728 ffff800`02b95b35 termdd!IcaDispatch

ffff880`05c14730 ffff800`02b923d8 nt!IopParseDevice+0x5a5

ffff880`05c148c0 ffff800`02b935f6 nt!ObpLookupObjectName+0x588

ffff880`05c149b0 ffff800`02b94efc nt!ObOpenObjectByName+0x306

ffff880`05c14a80 ffff800`02b9fb54 nt!IopCreateFile+0x2bc

ffff880`05c14b20 ffff800`0289b253 nt!NtCreateFile+0x78

ffff880`05c14bb0 00000000`7781186a nt!KiSystemServiceCopyEnd+0x13

00000000`06d0f6c8 000007fe`f95014b2 ntdll!NtCreateFile+0xa

00000000`06d0f6d0 000007fe`f95013f3 ICAAPI!IcaOpen+0xa6

00000000`06d0f790 000007fe`f7dbd2b6 ICAAPI!IcaOpen+0x13

00000000`06d0f7c0 000007fe`f7dc04bd rdpcorekmts!CKMRDPConnection::InitializeInstance+0x1da

00000000`06d0f830 000007fe`f7dbb58a rdpcorekmts!CKMRDPConnection::Listen+0xf9

00000000`06d0f8d0 000007fe`f7dba8ea rdpcorekmts!CKMRDPListener::ListenThreadWorker+0xae

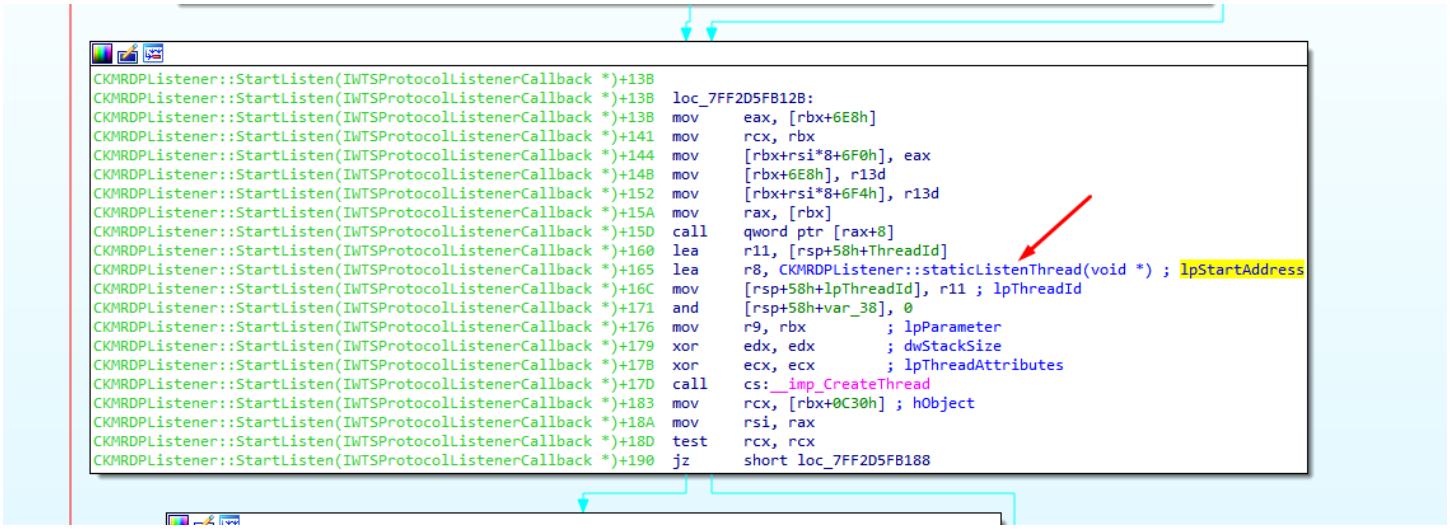
00000000`06d0f910 00000000`7755652d rdpcorekmts!CKMRDPListener::staticListenThread+0x12

00000000`06d0f940 00000000`777ec521 kernel32!BaseThreadInitThunk+0xd

00000000`06d0f970 00000000`00000000 ntdll!RtlUserThreadStart+0x1d

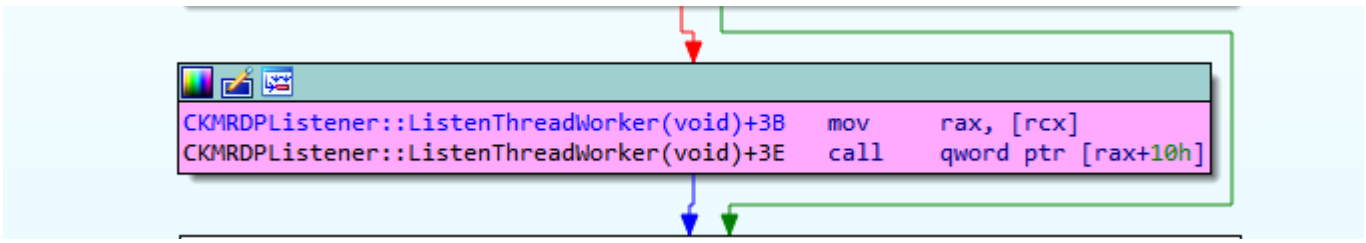
An instance of CKMRDPListener class is created.

This thread is created, the start address of the thread is the method CKMRDPListener::staticListenThread

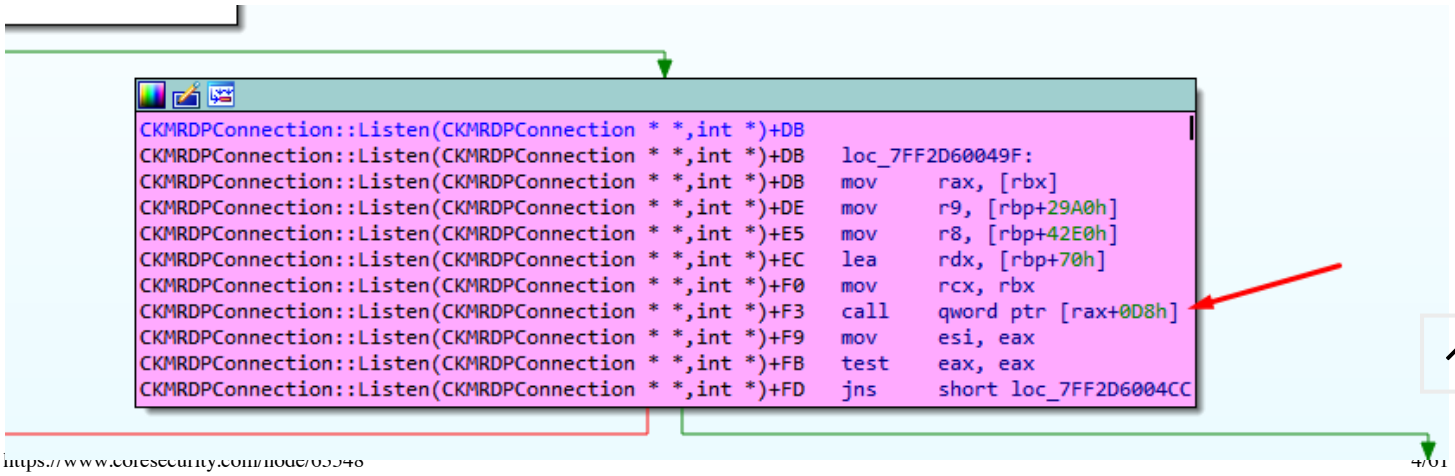


the execution continues here

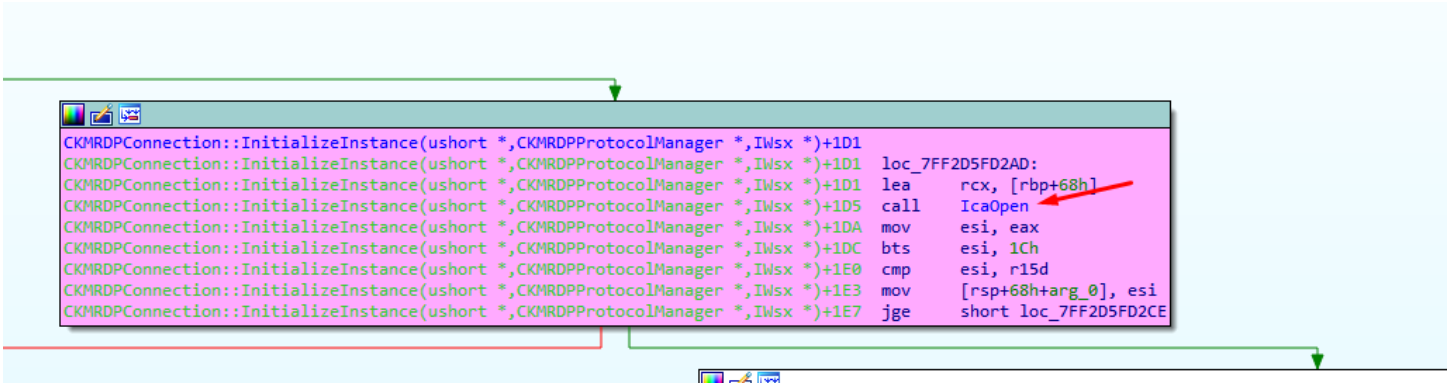
here

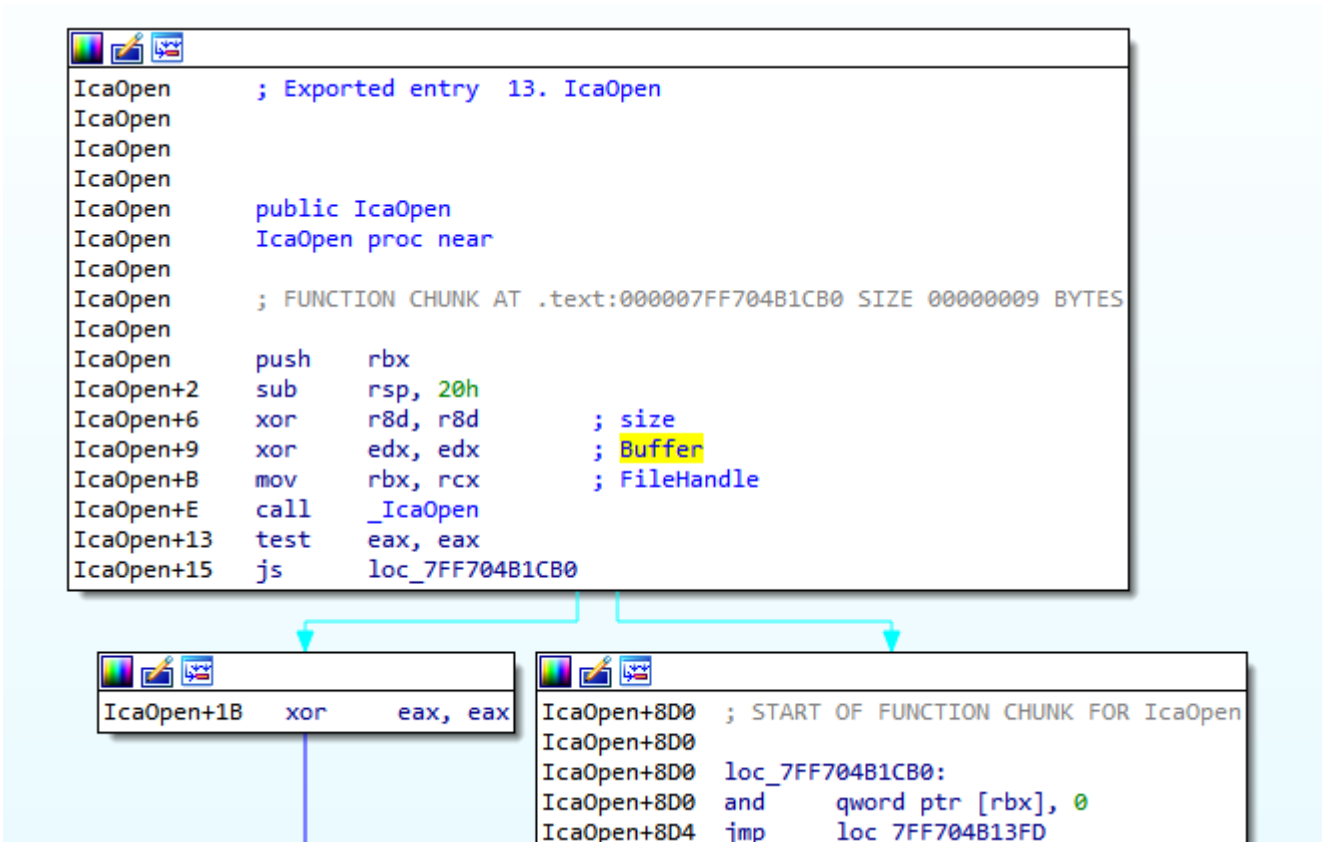


here



IcaOpen is called





We can see RDX (buffer) and r8d (size of buffer) both are equal to zero in this first call to IcaOpen.

Next the driver termdd is opened using the call to ntCreateFile

```
_IcaOpen
_IcaOpen
_IcaOpen        _IcaOpen proc near
_IcaOpen
_IcaOpen        var_98= qword ptr -98h
_IcaOpen        var_90= dword ptr -90h
_IcaOpen        var_88= dword ptr -88h
_IcaOpen        var_80= dword ptr -80h
_IcaOpen        var_78= dword ptr -78h
_IcaOpen        var_70= qword ptr -70h
_IcaOpen        var_68= dword ptr -68h
_IcaOpen        DestinationString= _UNICODE_STRING ptr -58h
_IcaOpen        var_48= byte ptr -48h
_IcaOpen        var_38= dword ptr -38h
_IcaOpen        var_30= qword ptr -30h
_IcaOpen        var_28= qword ptr -28h
_IcaOpen        var_20= dword ptr -20h
_IcaOpen        var_18= qword ptr -18h
_IcaOpen        var_10= qword ptr -10h
_IcaOpen        var_8= byte ptr -8
_IcaOpen        arg_0= qword ptr  8
_IcaOpen        arg_8= qword ptr  10h
_IcaOpen
_IcaOpen        mov     [rsp+arg_0], rbx
_IcaOpen+5      mov     [rsp+arg_8], rsi
_IcaOpen+A      push    rdi
_IcaOpen+B      sub     rsp, 0B0h
_IcaOpen+12     mov     rdi, rdx
_IcaOpen+15     mov     rsi, rcx
_IcaOpen+18     lea     rdx, SourceString ; "\\Device\\Termdd"
_IcaOpen+1F     lea     rcx, [rsp+0B8h+DestinationString] ; DestinationString
_IcaOpen+24     mov     ebx, r8d
_IcaOpen+27     call    cs:__imp_RtlInitUnicodeString
_IcaOpen+2D     xor     ecx, ecx
_IcaOpen+2F     mov     [rsp+0B8h+var_68], ebx
_IcaOpen+33     mov     [rsp+0B8h+var_70], rdi
_IcaOpen+38     mov     [rsp+0B8h+var_78], ecx
_IcaOpen+3C     lea     rax, [rsp+0B8h+DestinationString]
_IcaOpen+41     mov     [rsp+0B8h+var_28], rax
_IcaOpen+49     lea     eax, [rcx+3]
_IcaOpen+4C     mov     [rsp+0B8h+var_30], rcx
_IcaOpen+54     mov     [rsp+0B8h+var_80], eax
_IcaOpen+58     mov     [rsp+0B8h+var_88], eax
_IcaOpen+5C     mov     [rsp+0B8h+var_90], ecx
_IcaOpen+60     mov     [rsp+0B8h+var_98], rcx
_IcaOpen+65     mov     [rsp+0B8h+var_18], rcx
_IcaOpen+6D     mov     [rsp+0B8h+var_10], rcx
_IcaOpen+75     lea     r9, [rsp+0B8h+var_48]
_IcaOpen+7A     lea     r8, [rsp+0B8h+var_38]
_IcaOpen+82     mov     rcx, rsi
_IcaOpen+85     mov     edx, 0C0100000h
_IcaOpen+8A     mov     [rsp+0B8h+var_38], 30h ; '0'
_IcaOpen+95     mov     [rsp+0B8h+var_20], 40h ; '@'
_IcaOpen+A0     call    cs:__imp_NtCreateFile
_IcaOpen+A6     lea     r11, [rsp+0B8h+var_8]
_IcaOpen+AE     mov     rbx, [r11+10h]
_IcaOpen+B2     mov     rsi, [r11+18h]
_IcaOpen+B6     mov     rsp, r11
_IcaOpen+B9     pop     rdi
_IcaOpen+BA     retn
_IcaOpen+BA     _IcaOpen endp
_IcaOpen+BA
```

We arrived to IcaDispatch when opening the driver.

```
IcaDispatch
IcaDispatch      ; ============== S U B R O U T I N E =======================================
IcaDispatch
IcaDispatch
IcaDispatch      ; __int64 __fastcall IcaDispatch(PDEVICE_OBJECT DeviceObject, PIRP Irp)
IcaDispatch      IcaDispatch proc near                ; DATA XREF: .pdata:FFFFF88003C9E1E0↓o
IcaDispatch                                           ; DriverEntry+27F↓o
IcaDispatch
IcaDispatch      var_18= qword ptr -18h
IcaDispatch      IrpDisposition= dword ptr  8
IcaDispatch      arg_8= qword ptr  10h
IcaDispatch      arg_10= qword ptr  18h
IcaDispatch      arg_18= qword ptr  20h
IcaDispatch
IcaDispatch      mov     [rsp+arg_8], rbx
IcaDispatch+5    mov     [rsp+arg_10], rbp
IcaDispatch+A    mov     [rsp+arg_18], rsi
IcaDispatch+F    push    rdi
IcaDispatch+10   sub     rsp, 30h
IcaDispatch+14   mov     rsi, [rcx+40h]
IcaDispatch+18   mov     rdi, rdx
IcaDispatch+1B   mov     rdx, [rdx+0B8h]             ; IO_STACK_LOCATION
IcaDispatch+22   mov     eax, [rsi]
IcaDispatch+24   mov     r10, rcx
IcaDispatch+27   and     eax, 0Fh
IcaDispatch+2A   cmp     eax, 3
IcaDispatch+2D   jnz     short loc_FFFFF88003C964B0
```

Reversing we can see

```
0
1  DeviceExtension_rsi = DeviceObject->DeviceExtension;
2  _IRP_rdi = Irp;
3  IO_STACK_LOCATION_rdx = Irp->Tail.Overlay.CurrentStackLocation;
4  _DeviceObject = DeviceObject;
5  v6 = *( DWORD *)DeviceExtension_rsi & 0xF;
```

```
IcaDispatch      IrpDisposition= dword ptr  8
IcaDispatch      rbx_stored= qword ptr  10h
IcaDispatch      rbp_stored= qword ptr  18h
IcaDispatch      rsi_stored= qword ptr  20h
IcaDispatch
IcaDispatch      mov     [rsp+rbx_stored], rbx
IcaDispatch+5    mov     [rsp+rbp_stored], rbp
IcaDispatch+A    mov     [rsp+rsi_stored], rsi
IcaDispatch+F    push    rdi
IcaDispatch+10   sub     rsp, 30h
IcaDispatch+14   mov     rsi, [rcx+_DEVICE_OBJECT.DeviceExtension] ; RSI=DeviceExtension_rsi
IcaDispatch+18   mov     rdi, rdx                    ; RDI=_IRP_rdi
IcaDispatch+1B   mov     rdx, qword ptr [rdx+(_IRP.Tail+40h)] ; RDX=IO_STACK_LOCATION_rdx
IcaDispatch+22   mov     eax, dword ptr [rsi+_DEVICE_EXTENSION.___u0.Flags]
IcaDispatch+24   mov     r10, rcx
IcaDispatch+27   and     eax, 0Fh
IcaDispatch+2A   cmp     eax, 3
IcaDispatch+2D   jnz     short loc_FFFFF88003C964B0
```

The MajorFunction value is read here

```
51  {
52    if ( IO_STACK_LOCATION_rdx->MajorFunction )
53    {
54      v10 = v8 - 2;
55      if ( v10 )
56      {
```

```
IcaDispatch+3C   ; ---------------------------------------------------------------------------
IcaDispatch+3C
IcaDispatch+3C   loc_FFFFF88003C964B0:                   ; CODE XREF: IcaDispatch+2D↑j
IcaDispatch+3C   movzx   ecx, [rdx+_IO_STACK_LOCATION.MajorFunction]
IcaDispatch+3F   xor     ebx, ebx
IcaDispatch+41   cmp     eax, 1
IcaDispatch+44   jz      loc_FFFFF88003C96612
```

As MajorFuncion equals 0 it takes us to IcaCreate

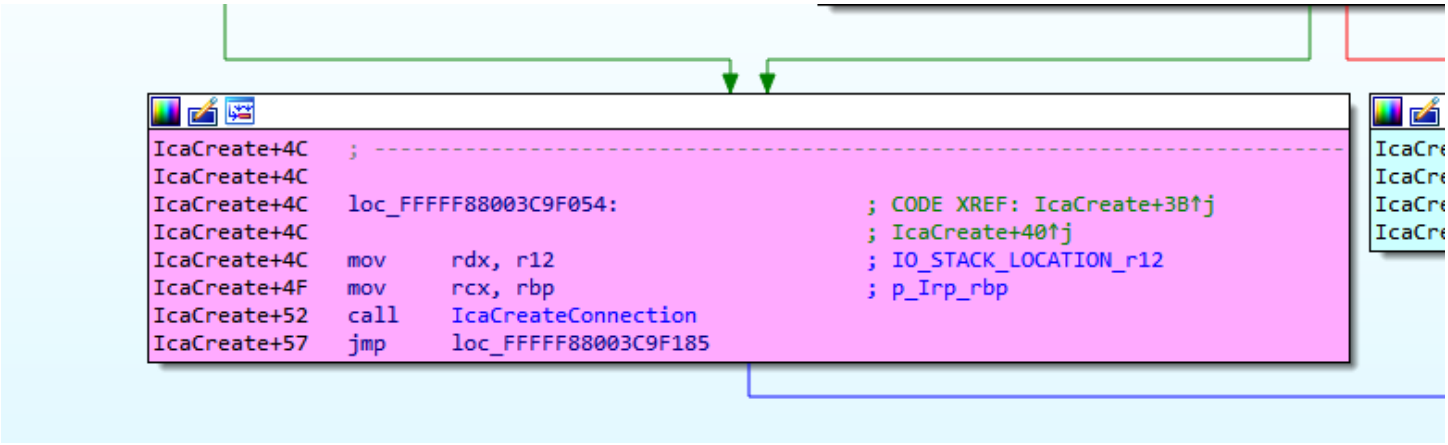
```
}
if ( !IO_STACK_LOCATION_rdx->MajorFunction )
{
  v27 = IcaCreate((__int64)_IRP_rdi, (__int64)IO_STACK_LOCATION_rdx);
  goto LABEL_64;
}
```

```
IcaDispatch+2CC  ; ---------------------------------------------------------------------------
IcaDispatch+2CC
IcaDispatch+2CC  loc_FFFFF88003C96740:                   ; CODE XREF: IcaDispatch+1A0↑j
IcaDispatch+2CC  mov     rcx, rdi                        ; p_Irp
IcaDispatch+2CF  call    IcaCreate
```

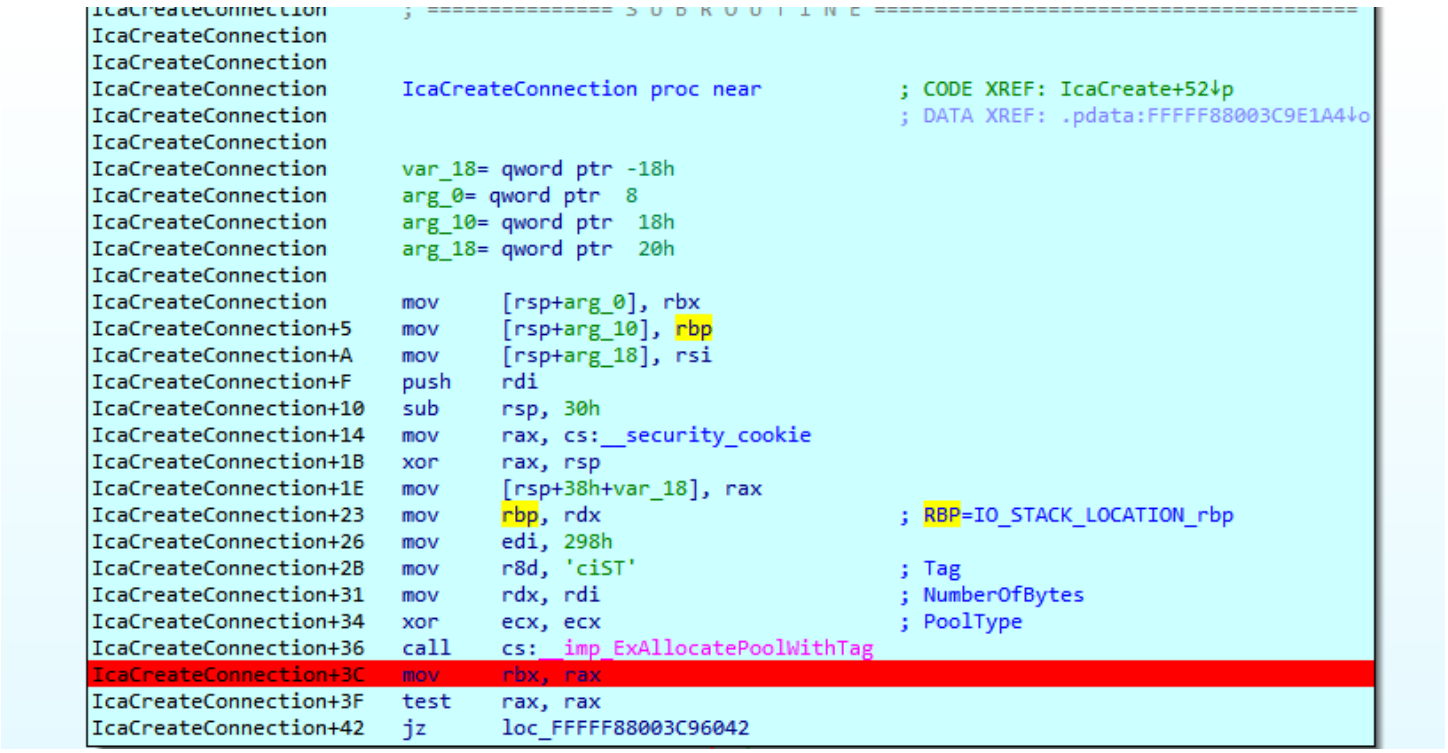
Inside IcaCreate, SystemBuffer is equal to 0

```
IcaCreate
IcaCreate       mov     [rsp+rbx_stored], rbx
IcaCreate+5     mov     [rsp+rbp_stored], rbp
IcaCreate+A     push    rsi
IcaCreate+B     push    rdi
IcaCreate+C     push    r12
IcaCreate+E     sub     rsp, 30h
IcaCreate+12    lea     r8, [rsp+48h+byte]              ; p_byte
IcaCreate+17    mov     r12, rdx                        ; R12=IO_STACK_LOCATION_r12
IcaCreate+1A    mov     rbp, rcx                        ; RBP=p_Irp_rbp
IcaCreate+1D    call    IcaIsSystemProcessRequest
IcaCreate+22    movzx   r11d, al                        ; R11= FsContext2_r11
IcaCreate+26    mov     rax, [r12+_IO_STACK_LOCATION.FileObject]
IcaCreate+2B    mov     [rax+_FILE_OBJECT.FsContext2], r11
IcaCreate+2F    mov     rcx, [rbp+_IRP.AssociatedIrp.SystemBuffer] ; RCX=AssociatedIrp.SystemBuffer_rcx
IcaCreate+33    test    rcx, rcx
IcaCreate+36    jnz     short loc_FFFFF88003C9F064
```

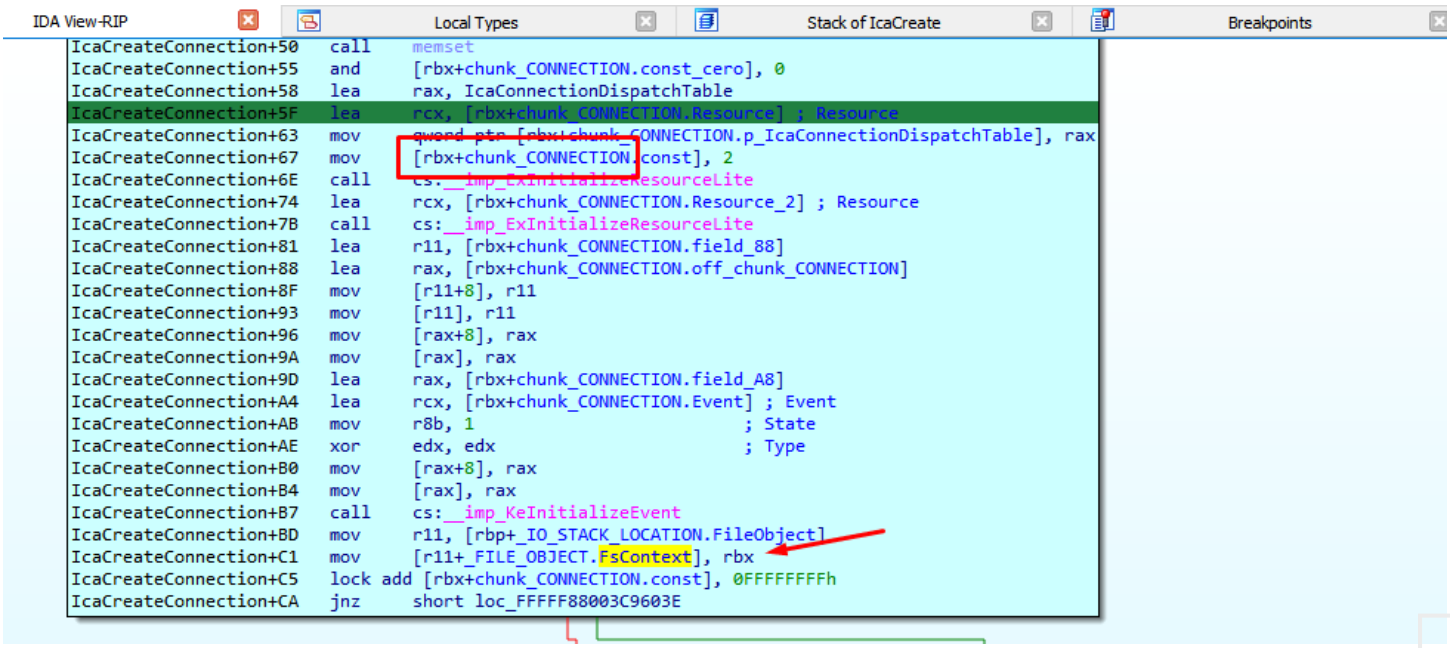
```
106     return -1073741790;
107   return IcaCreateConnection(p_Irp_rbp, IO_STACK_LOCATION_r12);
```



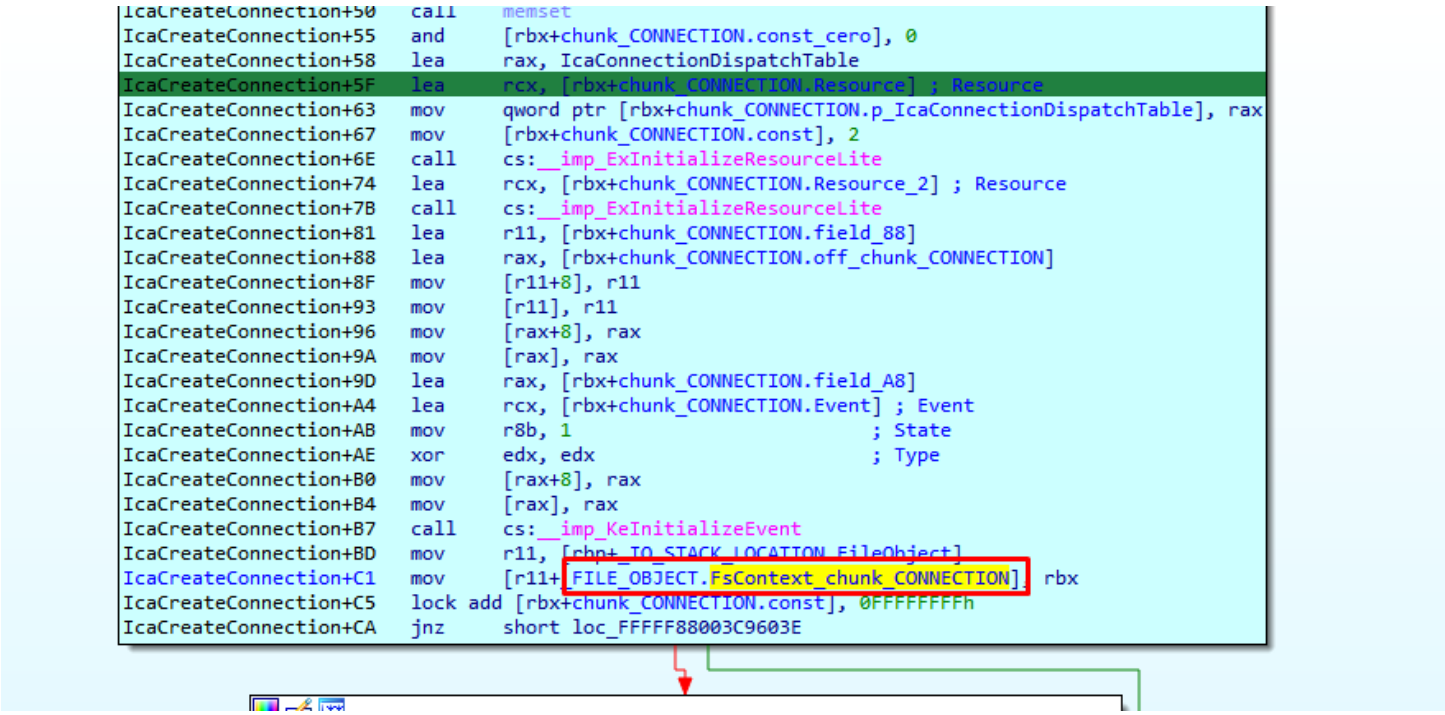
A chunk of size 0x298 and tag ciST is created, and I call it chunk_CONNECTION.



chunk_CONNECTION is stored in FILE_OBJECT.FsContext



I rename FsContext to **FsContext_chunk_CONNECTION**.



IcaDispatch is called for second time

Child-SP RetAddr Call Site

ffff880`05c146a0 ffff880`03c96748 termdd!IcaCreate+0x36

ffff880`05c146f0 ffff800`02b95b35 termdd!IcaDispatch+0x2d4

ffff880`05c14730 ffff800`02b923d8 nt!IopParseDevice+0x5a5

ffff880`05c148c0 ffff800`02b935f6 nt!ObpLookupObjectName+0x588

ffff880`05c149b0 ffff800`02b94efc nt!ObOpenObjectByName+0x306

ffff880`05c14a80 ffff800`02b9fb54 nt!IopCreateFile+0x2bc

ffff880`05c14b20 ffff800`0289b253 nt!NtCreateFile+0x78

ffff880`05c14bb0 00000000`7781186a nt!KiSystemServiceCopyEnd+0x13

00000000`06d0f618 000007fe`f95014b2 ntdll!NtCreateFile+0xa

00000000`06d0f620 000007fe`f95018c9 ICAAPI!IcaOpen+0xa6

00000000`06d0f6e0 000007fe`f95017e8 ICAAPI!IcaStackOpen+0xa4

00000000`06d0f710 000007fe`f7dbc015 ICAAPI!IcaStackOpen+0x83

00000000`06d0f760 000007fe`f7dbd2f9 rdpcorekmts!CStack::CStack+0x189

00000000`06d0f7c0 000007fe`f7dc04bd rdpcorekmts!CKMRDPConnection::InitializeInstance+0x21d

00000000`06d0f830 000007fe`f7dbb58a rdpcorekmts!CKMRDPConnection::Listen+0xf9

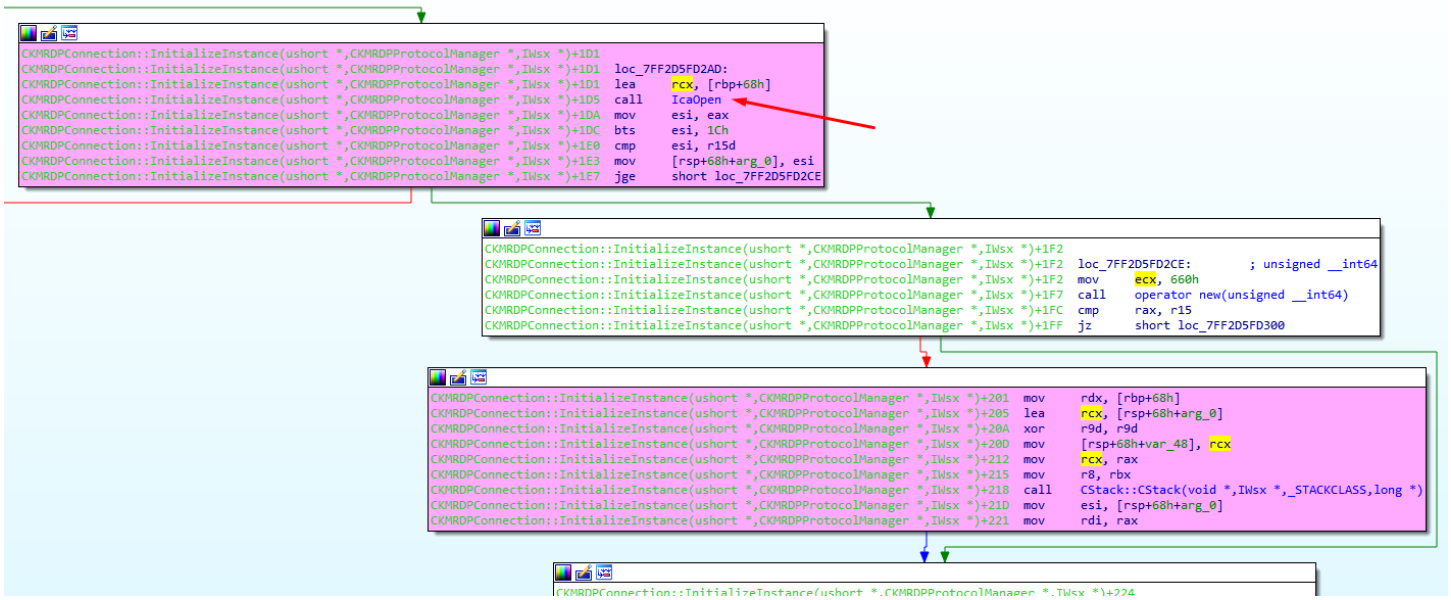
00000000`06d0f8d0 000007fe`f7dba8ea rdpcorekmts!CKMRDPListener::ListenThreadWorker+0xae

00000000`06d0f910 00000000`7755652d rdpcorekmts!CKMRDPListener::staticListenThread+0x12

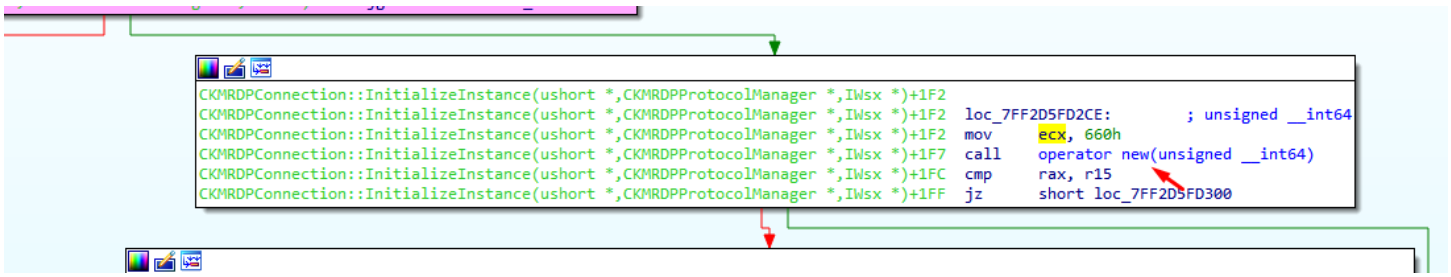
00000000`06d0f940 00000000`777ec521 kernel32!BaseThreadInitThunk+0xd

00000000`06d0f970 00000000`00000000 ntdll!RtlUserThreadStart+0x1d

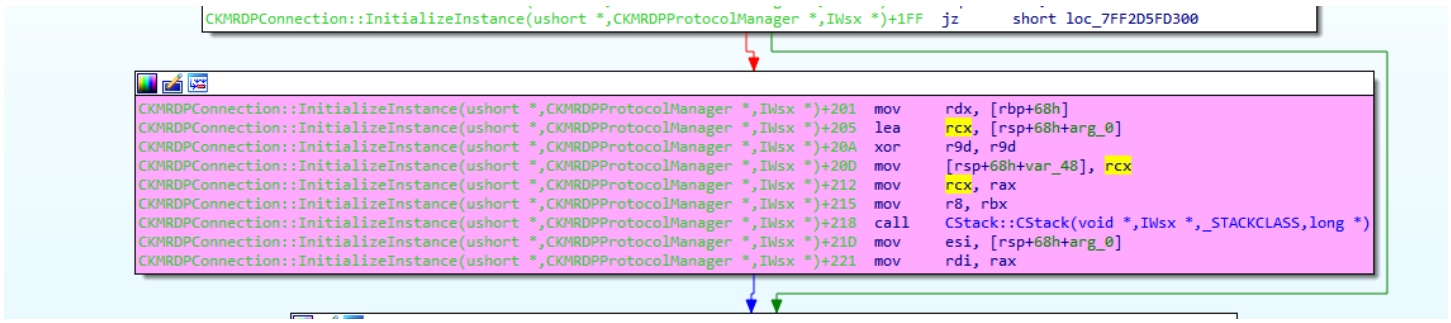
We had seen that the previous call to the driver had been generated here



When that call ends an instance of the class Cstack is created



And the class constructor is called.



this matches the current call stack

ffff880`05c146a0 ffff880`03c96748 termdd!IcaCreate+0x36

ffff880`05c146f0 ffff800`02b95b35 termdd!IcaDispatch+0x2d4

ffff880`05c14730 ffff800`02b923d8 nt!IopParseDevice+0x5a5

ffff880`05c148c0 ffff800`02b935f6 nt!ObpLookupObjectName+0x588

ffff880`05c149b0 ffff800`02b94efc nt!ObOpenObjectByName+0x306

ffff880`05c14a80 ffff800`02b9fb54 nt!IopCreateFile+0x2bc

ffff880`05c14b20 ffff800`0289b253 nt!NtCreateFile+0x78

ffff880`05c14bb0 00000000`7781186a nt!KiSystemServiceCopyEnd+0x13

00000000`06d0f618 000007fe`f95014b2 ntdll!NtCreateFile+0xa

00000000`06d0f620 000007fe`f95018c9 ICAAPI!IcaOpen+0xa6

00000000`06d0f6e0 000007fe`f95017e8 ICAAPI!IcaStackOpen+0xa4

00000000`06d0f710 000007fe`f7dbc015 ICAAPI!IcaStackOpen+0x83

00000000`06d0f760 000007fe`f7dbd2f9 rdpcorekmts!CStack::CStack+0x189

00000000`06d0f7c0 000007fe`f7dc04bd rdpcorekmts!CKMRDPConnection::InitializeInstance+0x21d

00000000`06d0f830 000007fe`f7dbb58a rdpcorekmts!CKMRDPConnection::Listen+0xf9

00000000`06d0f8d0 000007fe`f7dba8ea rdpcorekmts!CKMRDPListener::ListenThreadWorker+0xae

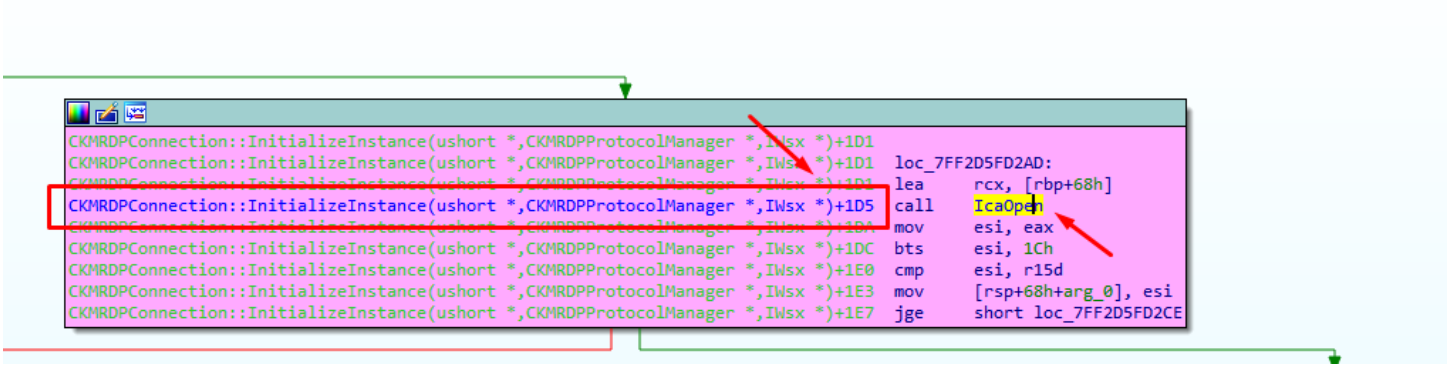
00000000`06d0f910 00000000`7755652d rdpcorekmts!CKMRDPListener::staticListenThread+0x12

00000000`06d0f940 00000000`777ec521 kernel32!BaseThreadInitThunk+0xd

00000000`06d0f970 00000000`00000000 ntdll!RtlUserThreadStart+0x1d

The highlighted text is the same for both calls, the difference is the red line and the upper lines

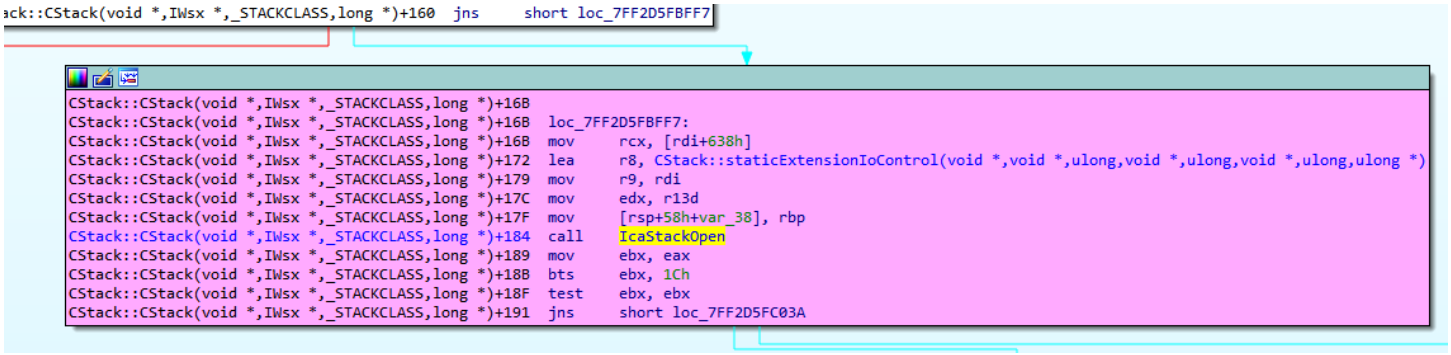
00000000`06d0f7c0 000007fe`f7dc04bd rdpcorekmts!CKMRDPConnection::InitializeInstance+0x1da



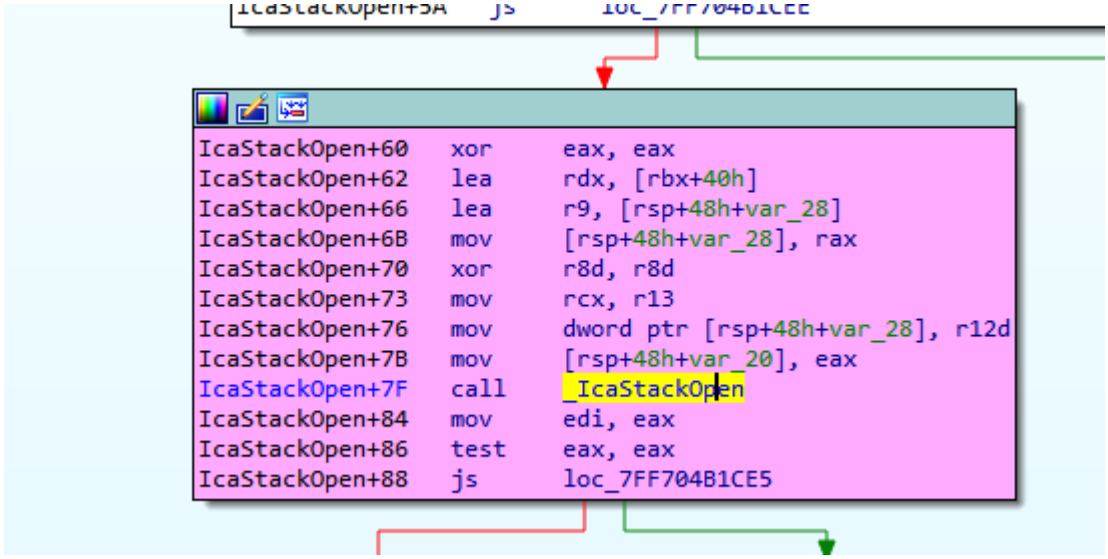
The second call returns to

00000000`06d0f7c0 000007fe`f7dc04bd rdpcorekmts!CKMRDPConnection::InitializeInstance+0x21d

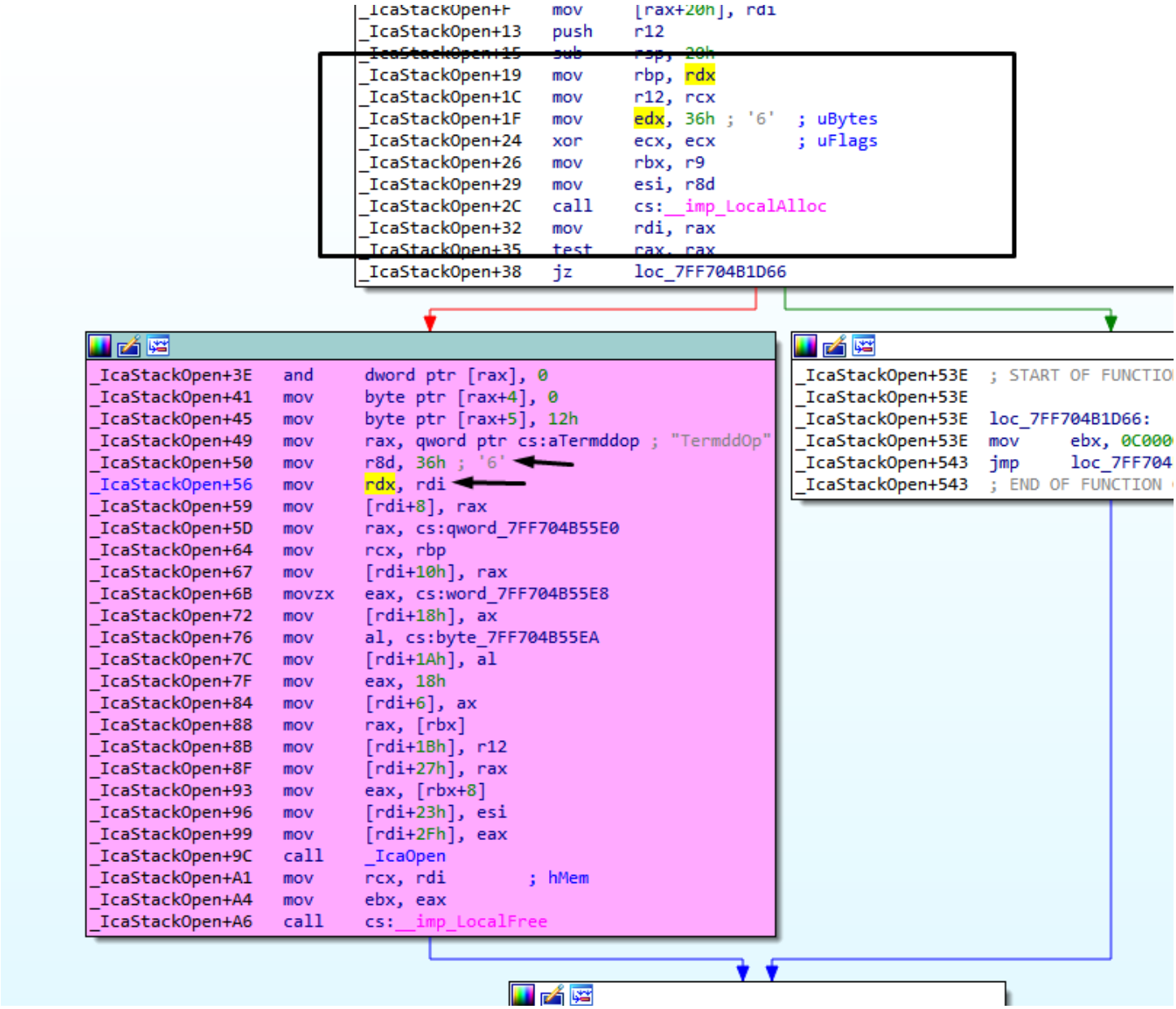
And this second call continues to


ack::CStack(void *,IWsx *,_STACKCLASS,long *)+160 jns short loc_7FF2D5FBFF7
CStack::CStack(void *,IWsx *,_STACKCLASS,long *)+16B
CStack::CStack(void *,IWsx *,_STACKCLASS,long *)+16B loc_7FF2D5FBFF7:
CStack::CStack(void *,IWsx *,_STACKCLASS,long *)+16B mov rcx, [rdi+638h]
CStack::CStack(void *,IWsx *,_STACKCLASS,long *)+172 lea r8, CStack::staticExtensionIoControl(void *,void *,ulong,void *,ulong,void *,ulong,ulong *)
CStack::CStack(void *,IWsx *,_STACKCLASS,long *)+179 mov r9, rdi
CStack::CStack(void *,IWsx *,_STACKCLASS,long *)+17C mov edx, r13d
CStack::CStack(void *,IWsx *,_STACKCLASS,long *)+17F mov [rsp+58h+var_38], rbp
CStack::CStack(void *,IWsx *,_STACKCLASS,long *)+184 call IcaStackOpen
CStack::CStack(void *,IWsx *,_STACKCLASS,long *)+189 mov ebx, eax
CStack::CStack(void *,IWsx *,_STACKCLASS,long *)+18B bts ebx, 1Ch
CStack::CStack(void *,IWsx *,_STACKCLASS,long *)+18F test ebx, ebx
CStack::CStack(void *,IWsx *,_STACKCLASS,long *)+191 jns short loc_7FF2D5FC03A

Next


IcaStackOpen+60 xor eax, eax
IcaStackOpen+62 lea rdx, [rbx+40h]
IcaStackOpen+66 lea r9, [rsp+48h+var_28]
IcaStackOpen+6B mov [rsp+48h+var_28], rax
IcaStackOpen+70 xor r8d, r8d
IcaStackOpen+73 mov rcx, r13
IcaStackOpen+76 mov dword ptr [rsp+48h+var_28], r12d
IcaStackOpen+7B mov [rsp+48h+var_20], eax
IcaStackOpen+7F call _IcaStackOpen
IcaStackOpen+84 mov edi, eax
IcaStackOpen+86 test eax, eax
IcaStackOpen+88 js loc_7FF704B1CE5

Next



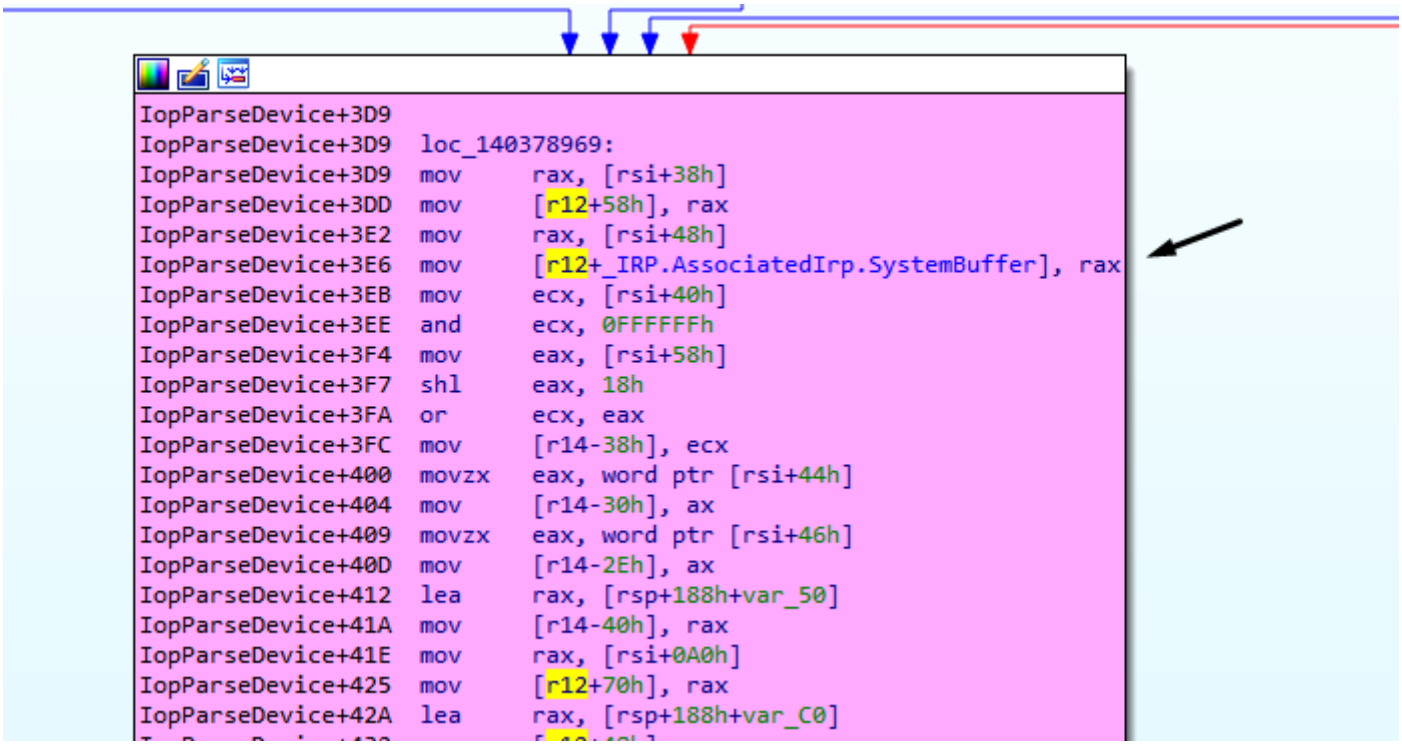
We arrived to _IcaOpen, calling ntCreafile for the second time, but now Buffer is a chunk in user allocated with a size different than zero, its size is 0x36.

```
_IcaOpen
_IcaOpen
_IcaOpen
_IcaOpen        _IcaOpen proc near
_IcaOpen
_IcaOpen        var_98= qword ptr -98h
_IcaOpen        var_90= dword ptr -90h
_IcaOpen        var_88= dword ptr -88h
_IcaOpen        var_80= dword ptr -80h
_IcaOpen        var_78= dword ptr -78h
_IcaOpen        var_70= qword ptr -70h
_IcaOpen        var_68= dword ptr -68h
_IcaOpen        DestinationString= _UNICODE_STRING ptr -58h
_IcaOpen        var_48= byte ptr -48h
_IcaOpen        var_38= dword ptr -38h
_IcaOpen        var_30= qword ptr -30h
_IcaOpen        var_28= qword ptr -28h
_IcaOpen        var_20= dword ptr -20h
_IcaOpen        var_18= qword ptr -18h
_IcaOpen        var_10= qword ptr -10h
_IcaOpen        var_8= byte ptr -8
_IcaOpen        arg_0= qword ptr  8
_IcaOpen        arg_8= qword ptr  10h
_IcaOpen
_IcaOpen        mov     [rsp+arg_0], rbx
_IcaOpen+5      mov     [rsp+arg_8], rsi
_IcaOpen+A      push    rdi
_IcaOpen+B      sub     rsp, 0B0h
_IcaOpen+12     mov     rdi, rdx
_IcaOpen+15     mov     rsi, rcx
_IcaOpen+18     lea     rdx, SourceString ; "\\Device\\Termdd"
_IcaOpen+1F     lea     rcx, [rsp+0B8h+DestinationString] ; DestinationString
_IcaOpen+24     mov     ebx, r8d
_IcaOpen+27     call    cs:__imp_RtlInitUnicodeString
_IcaOpen+2D     xor     ecx, ecx
_IcaOpen+2F     mov     [rsp+0B8h+var_68], ebx
_IcaOpen+33     mov     [rsp+0B8h+var_70], rdi
_IcaOpen+38     mov     [rsp+0B8h+var_78], ecx
_IcaOpen+3C     lea     rax, [rsp+0B8h+DestinationString]
_IcaOpen+41     mov     [rsp+0B8h+var_28], rax
_IcaOpen+49     lea     eax, [rcx+3]
_IcaOpen+4C     mov     [rsp+0B8h+var_30], rcx
_IcaOpen+54     mov     [rsp+0B8h+var_80], eax
_IcaOpen+58     mov     [rsp+0B8h+var_88], eax
_IcaOpen+5C     mov     [rsp+0B8h+var_90], ecx
_IcaOpen+60     mov     [rsp+0B8h+var_98], rcx
_IcaOpen+65     mov     [rsp+0B8h+var_18], rcx
_IcaOpen+6D     mov     [rsp+0B8h+var_10], rcx
_IcaOpen+75     lea     r9, [rsp+0B8h+var_48]
_IcaOpen+7A     lea     r8, [rsp+0B8h+var_38]
_IcaOpen+82     mov     rcx, rsi
_IcaOpen+85     mov     edx, 0C0100000h
_IcaOpen+8A     mov     [rsp+0B8h+var_38], 30h ; '0'
_IcaOpen+95     mov     [rsp+0B8h+var_20], 40h ; '@'
_IcaOpen+A0     call    cs:__imp_NtCreateFile
_IcaOpen+A6     lea     r11, [rsp+0B8h+var_8]
_IcaOpen+AE     mov     rbx, [r11+10h]
_IcaOpen+B2     mov     rsi, [r11+18h]
_IcaOpen+B6     mov     rsp, r11
_IcaOpen+B9     pop     rdi
_IcaOpen+BA     retn
_IcaOpen+BA     _IcaOpen endp
_IcaOpen+BA
```

This second call reaches IcaDispath and IcaCreate in similar way to the first call.

But now SystemBuffer is different than zero, I suppose that SystemBuffer is created, if the buffer size is different to zero.(in the first call buffer=0 → SystemBuffer=0 now buffer!=0 → SystemBuffer is !=0).

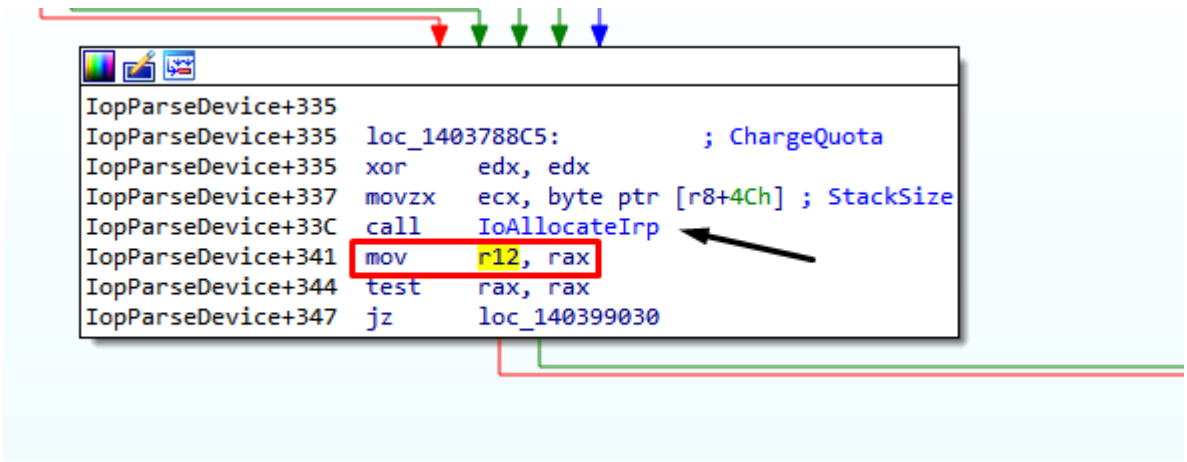
SystemBuffer is stored in **_IRP.AssociatedIrp.SystemBuffer** here



in the decompiled code

```
    if ( !(a5 & 0x40) )
      v54[-1].Flags = v56 | 0x80;
  }
  *(_QWORD *)(v52 + 88) = *(_QWORD *)(v14 + 56);
  *(_QWORD *)(v52 + offsetof(_IRP, AssociatedIrp)) = *(_QWORD *)(v14 + 72);
  v54[-1].Parameters.Create.Options = (*(_DWORD *)(v14 + 88) << 24) | *(_DWORD *)(v14 + 6
  v54[-1].Parameters.Create.FileAttributes = *(_WORD *)(v14 + 68);
  v54[-1].Parameters.Create.ShareAccess = *(_WORD *)(v14 + 70);
  v54[-1].Parameters.WMI.ProviderId = (unsigned __int64)&v210;
  *(_QWORD *)(v52 + 112) = *(_QWORD *)(v14 + 160);
  *(_QWORD *)(v52 + 72) = &v195;
```

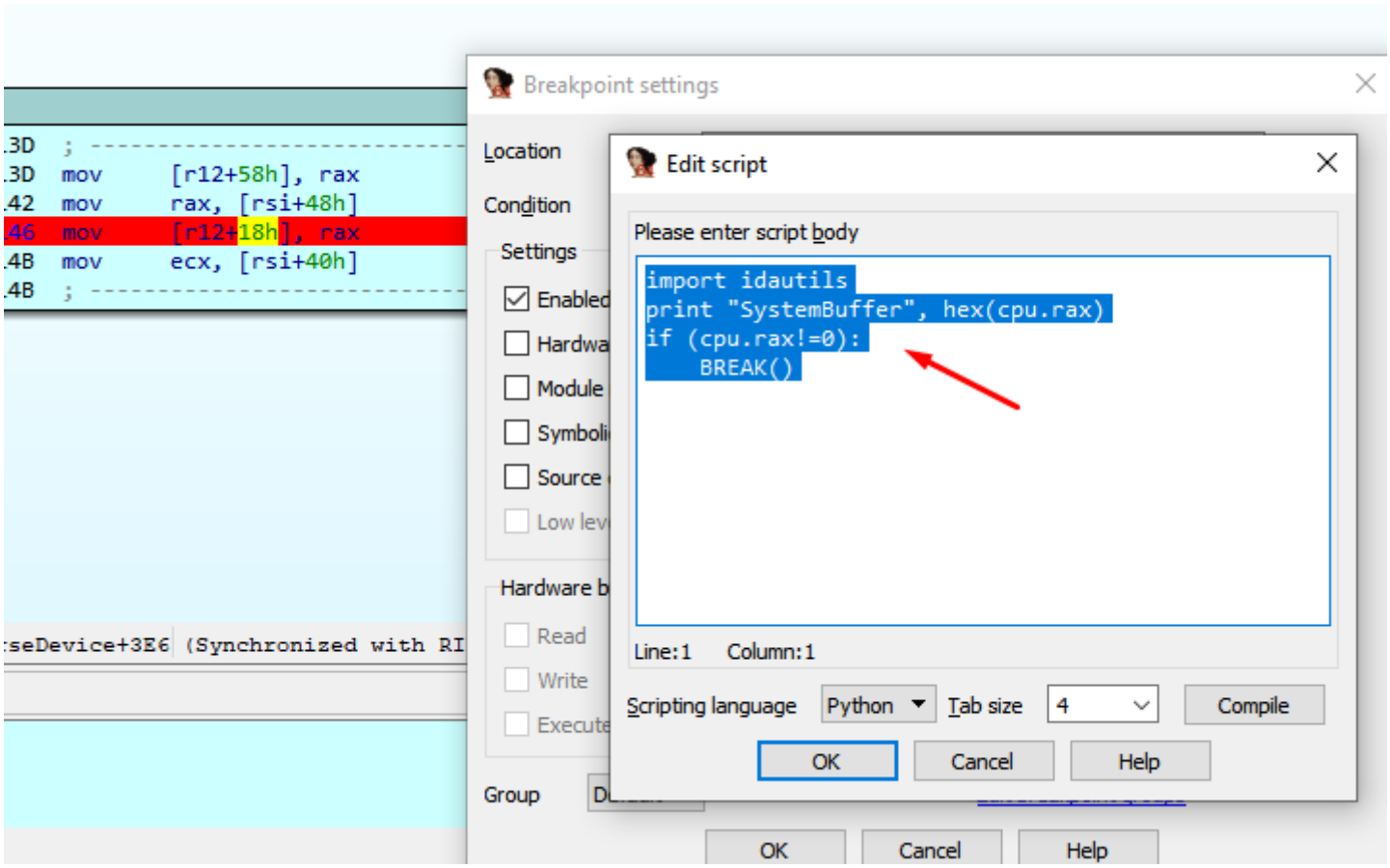
Previously IRP is moved to r12



=

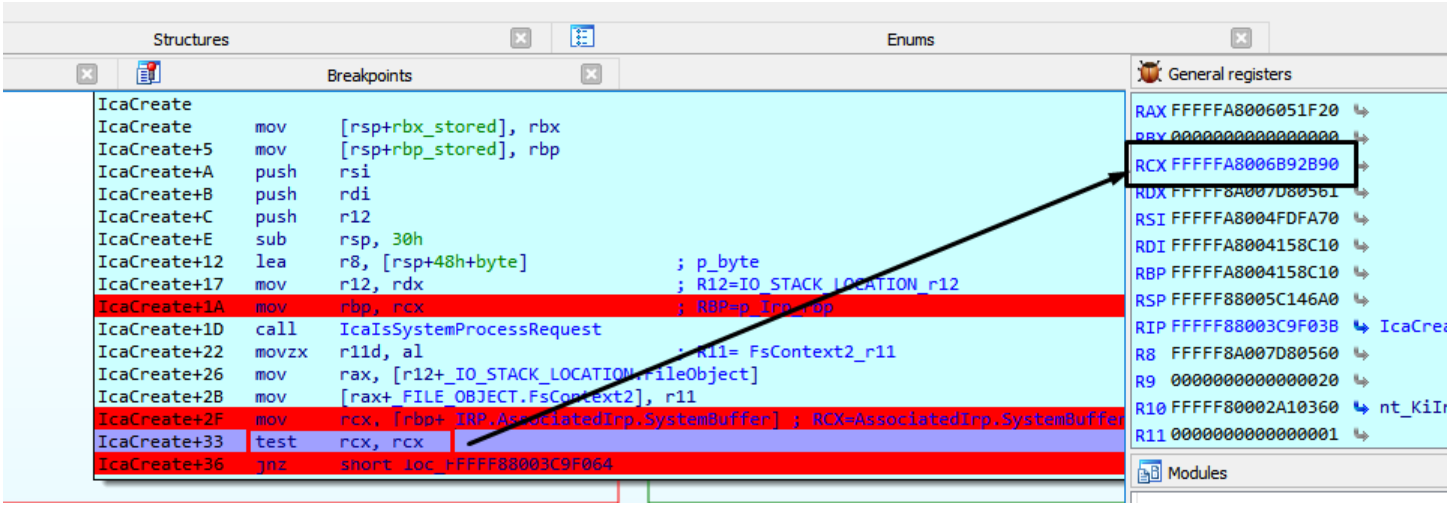
```
IcaCreate
IcaCreate       ; NTSTATUS __fastcall IcaCreate(__int64 p_Irp, __int64 IO_STACK_LOCATION)
IcaCreate       IcaCreate proc near                     ; CODE XREF: IcaDispatch+2CF↑p
IcaCreate                                               ; HandleMonitorCall+96↑p
IcaCreate                                               ; DATA XREF: ...
IcaCreate
IcaCreate       p_Object= qword ptr -28h
IcaCreate       const_cero= qword ptr -20h
IcaCreate       byte= byte ptr  8
IcaCreate       _Object= qword ptr  10h
IcaCreate       rbx_stored= qword ptr  18h
IcaCreate       rbp_stored= qword ptr  20h
IcaCreate
IcaCreate       mov     [rsp+rbx_stored], rbx
IcaCreate+5     mov     [rsp+rbp_stored], rbp
IcaCreate+A     push    rsi
IcaCreate+B     push    rdi
IcaCreate+C     push    r12
IcaCreate+E     sub     rsp, 30h
IcaCreate+12    lea     r8, [rsp+48h+byte]              ; p_byte
IcaCreate+17    mov     r12, rdx                        ; R12=IO_STACK_LOCATION_r12
IcaCreate+1A    mov     rbp, rcx                        ; RBP=p_Irp_rbp
IcaCreate+1D    call    IcaIsSystemProcessRequest
IcaCreate+22    movzx   r11d, al                        ; R11= FsContext2_r11
IcaCreate+26    mov     rax, [r12+_IO_STACK_LOCATION.FileObject]
IcaCreate+2B    mov     [rax+_FILE_OBJECT.FsContext2], r11
IcaCreate+2F    mov     rcx, [rbp+_IRP.AssociatedIrp.SystemBuffer] ; RCX=AssociatedIrp.SystemBuffer_rcx
IcaCreate+33    test    rcx, rcx
IcaCreate+36    jnz     short loc_FFFFF88003C9F064
```

That address is accessed many times over there, so the only way to stop when it is nonzero is to use a conditional breakpoint.

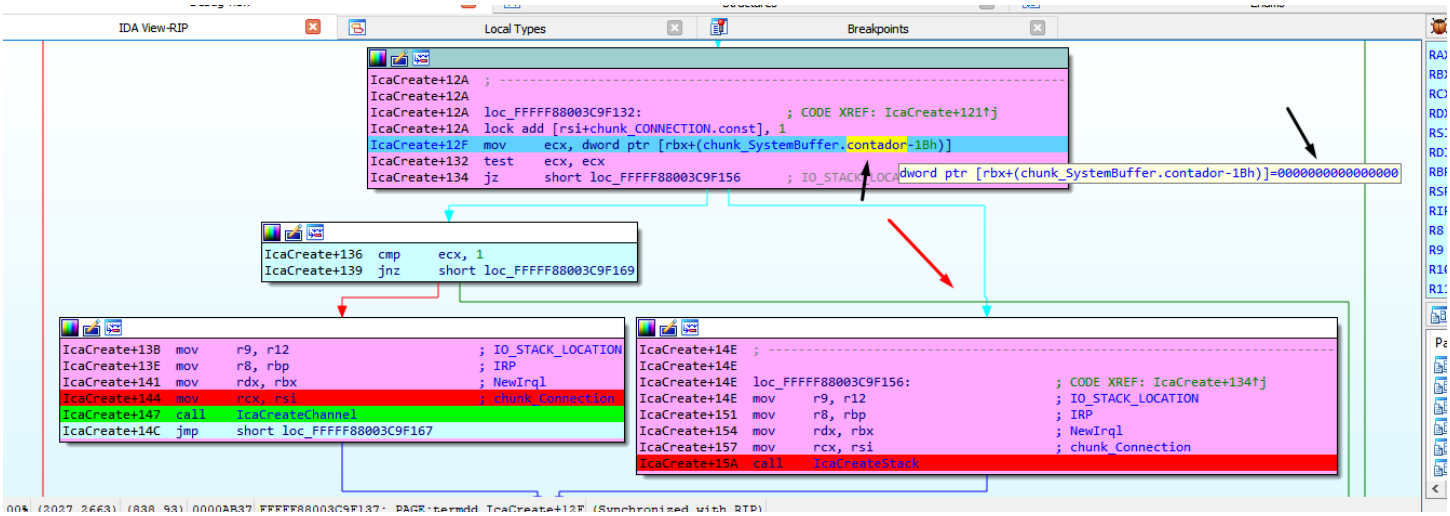


```
SystemBuffer 0x0L
SystemBuffer 0x0L
CREATE
IRP 0xfffffa8004158c10L
chunk_SystemBuffer  0x0L
IRP  0xfffffa8004158c10L
chunk_CONNECTION 0xfffffa80048bbb40L
SystemBuffer 0xfffffa8006b92b90L
CREATE
IRP 0xfffffa8004158c10L
chunk_SystemBuffer  0xfffffa8006b92b90L
```

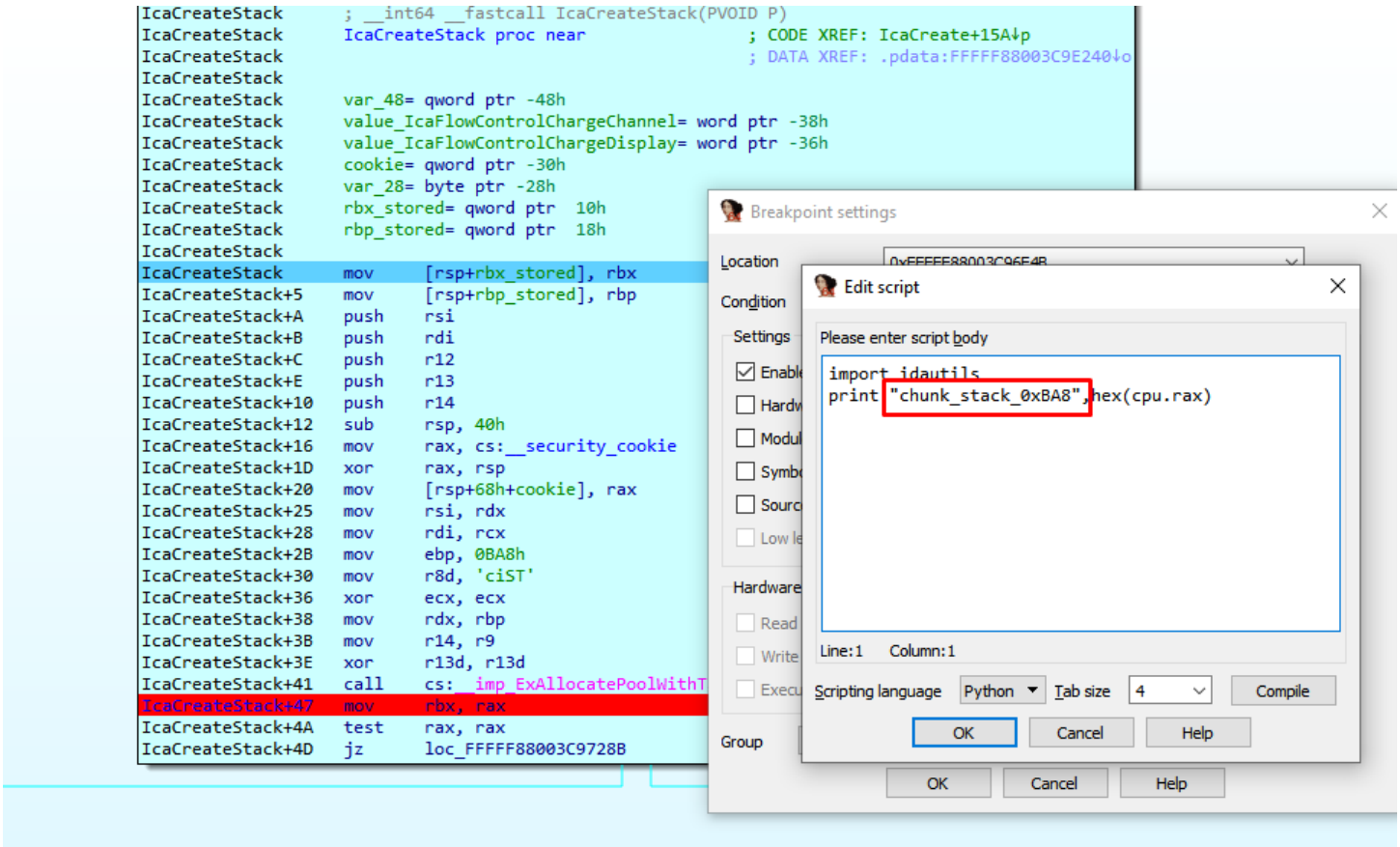
The first time that RAX is different from zero it stops before the second call to CREATE, and if I continue executing, I reach IcaCreate with that new value of SystemBuffer.



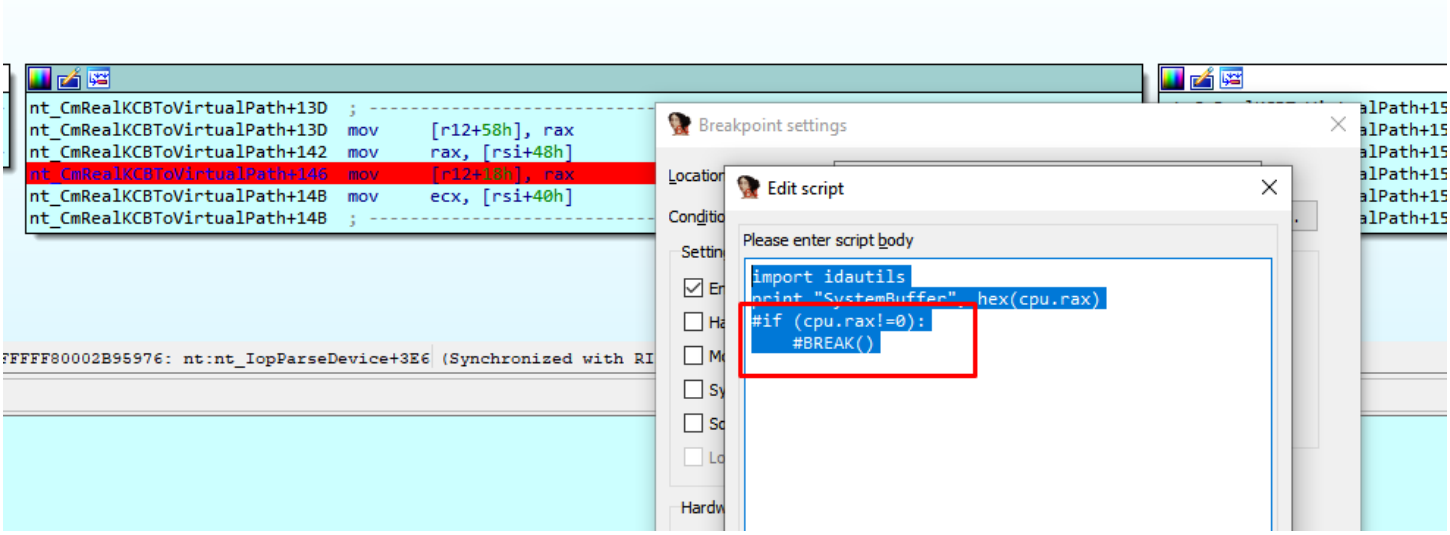
We arrived at this code, the variable named "contador" is zero, for this reason, we landed in IcaCreateStack.



In IcaCreateStack a new fragment of size 0xBA8 is allocated, I call it chunk_stack_0xBA8.



I comment the conditional breakpoint part, to avoid stopping and only keep logging.



I repeat the process to get a new fresh log.

```
SystemBuffer 0x0L
SystemBuffer 0x0L
SystemBuffer 0x0L
SystemBuffer 0x0L
SystemBuffer 0x0L
SystemBuffer 0x0L
SystemBuffer 0x0L
SystemBuffer 0x0L
CREATE
IRP 0xfffffa8004158c10L
chunk_SystemBuffer  0x0L
IRP  0xfffffa8004158c10L
chunk_CONNECTION 0xfffffa8006a2c6e0L
SystemBuffer 0xfffffa80067fca00L
CREATE
IRP 0xfffffa8004158c10L
chunk_SystemBuffer  0xfffffa80067fca00L
IRP  0xfffffa8004158c10L
FILE_OBJECT  0xfffffa80070fa820L
chunk_CONNECTION  0xfffffa8006a2c6e0L
chunk_stack_0xBA8 0xfffffa8004711450L
```

Summarizing by just executing this two lines of code to create a connection, and even without sending data, we have access to the driver

The most relevent part of the log when connecting is this.

```
MAJOR_FUNCTION 0x0L
CREATE
chunk_CONNECTION 0xfffffa800538bd60L
MAJOR_FUNCTION 0x0L
CREATE
chunk_stack_0xBA8 0xfffffa80046c0450L
```

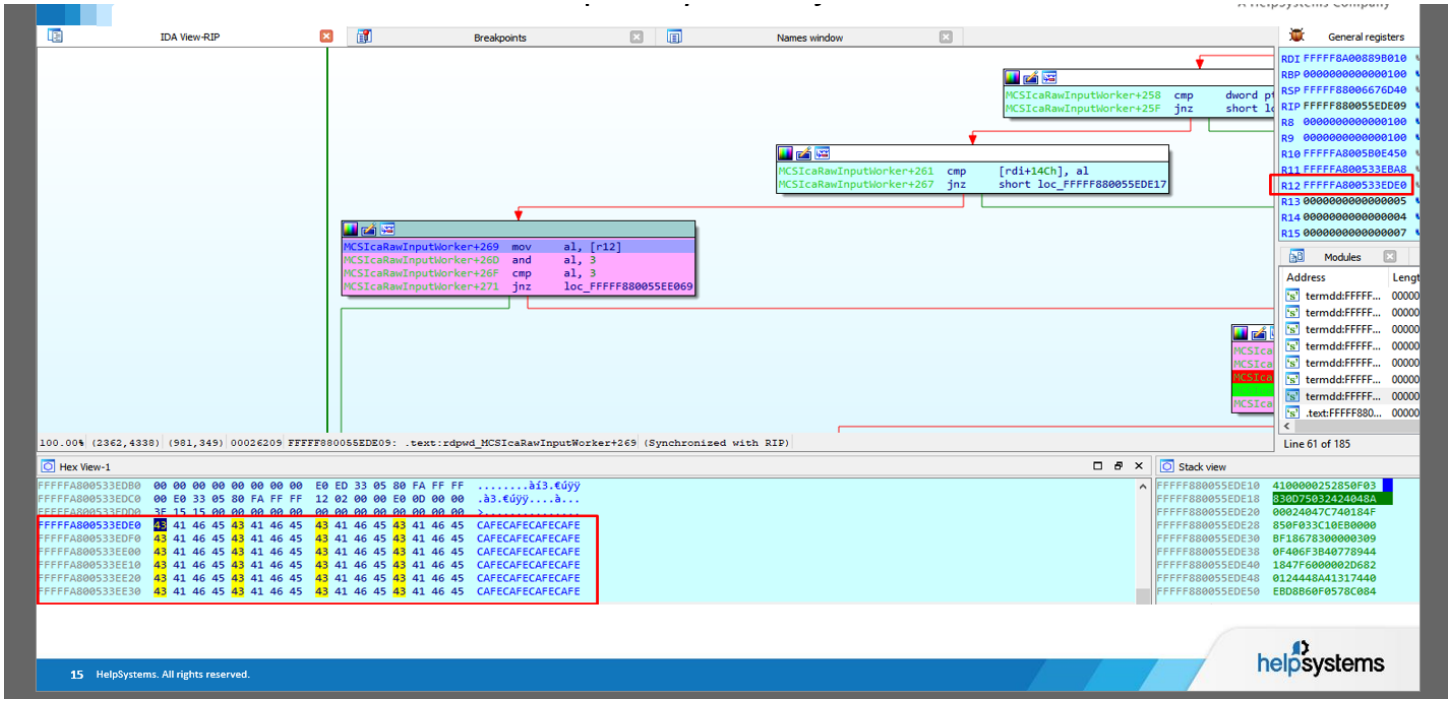
IcaCreate was called two times, with MajorFunction = 0x0.
The first call allocates CHUNK_CONNECTION, the second call allocates chunk_stack_0xBA8.

We will begin to reverse the data that it receives, for it would be convenient to be able to use Wireshark to analyze the data, although as the connection is encrypted with SSL, in Wireshark we could only see that the encrypted data which does not help us much.

```
if self.__isRemoteTargetXP():
    self.__negotitateRequest(useSSL=False)
else:
    self.__negotitateRequest(useSSL=True)
    self.__initiateSSLConnection()
```

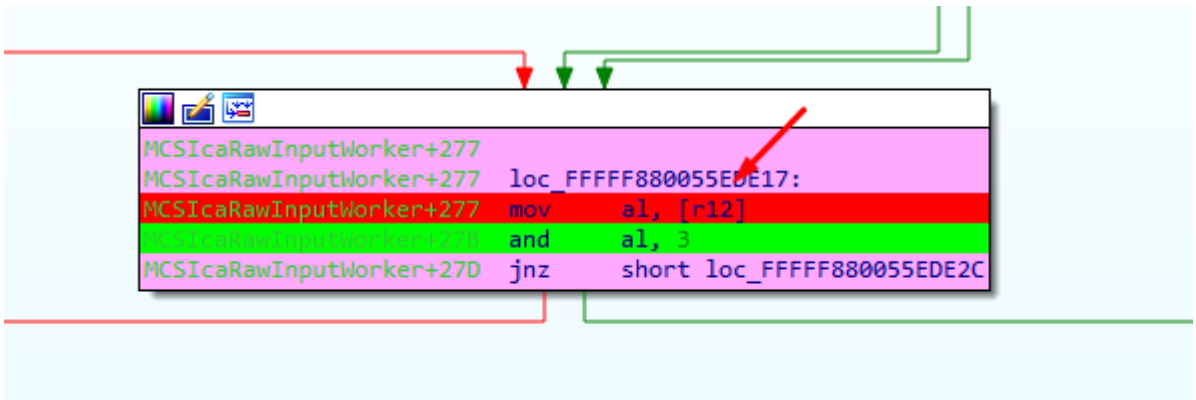
The data travels encrypted and thus the Wireshark receives it, but we will try to use it all the same.

For this purpose we need to detect the point where the program begins to parse data already decrypted.



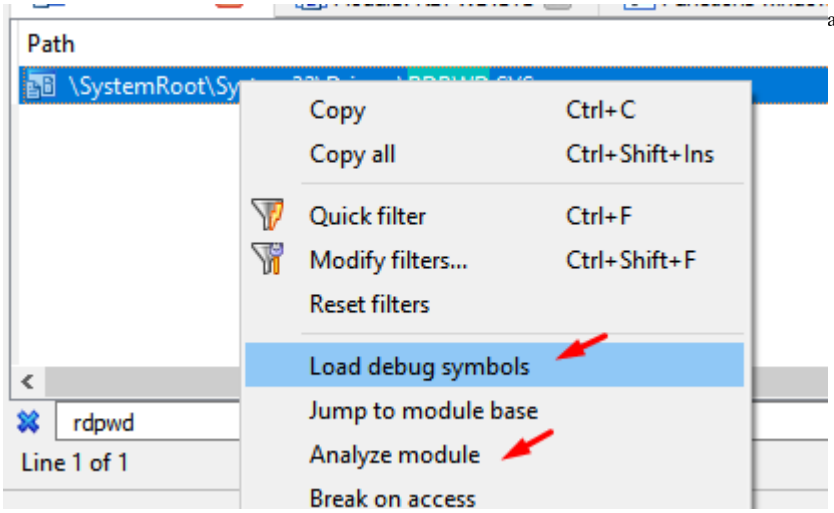
The driver rdpwd.sys is in charge of starting to parse the data already decrypted.

The important point for us is in the function MCSIcaRawInputWorker, where the program started to parse the decrypted code.

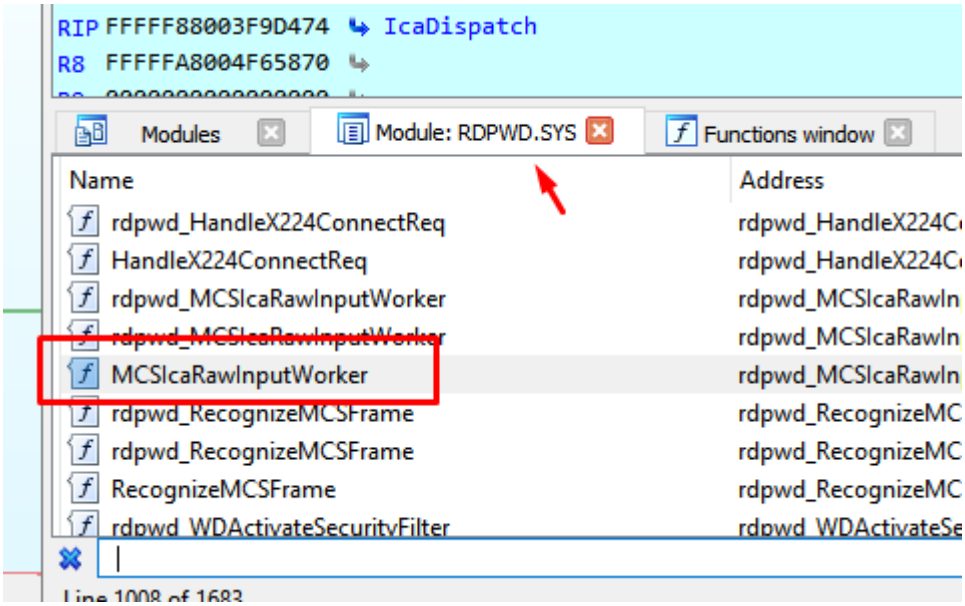


STEP 2) Put some conditional breakpoints in IDA PRO to dump to a file the data decrypted

The idea is place a conditional breakpoint in that point, so that each time the execution passes there, it will save the data it has already decrypted in a file, then use that file and load it in Wireshark.



This will analyze the module rdpwd.sys and I can find its functions in IDA, debugging from my database of termdd.sys, when it stops at any breakpoint of this driver.



I already found the important point: if the module rdpwd.sys changes its location by ASLR, I will have to repeat these steps to relocate the breakpoint correctly.

address = 0xFFFFF880034675E8

filename=r"C:\Users\ricardo\Desktop\pepe.txt"

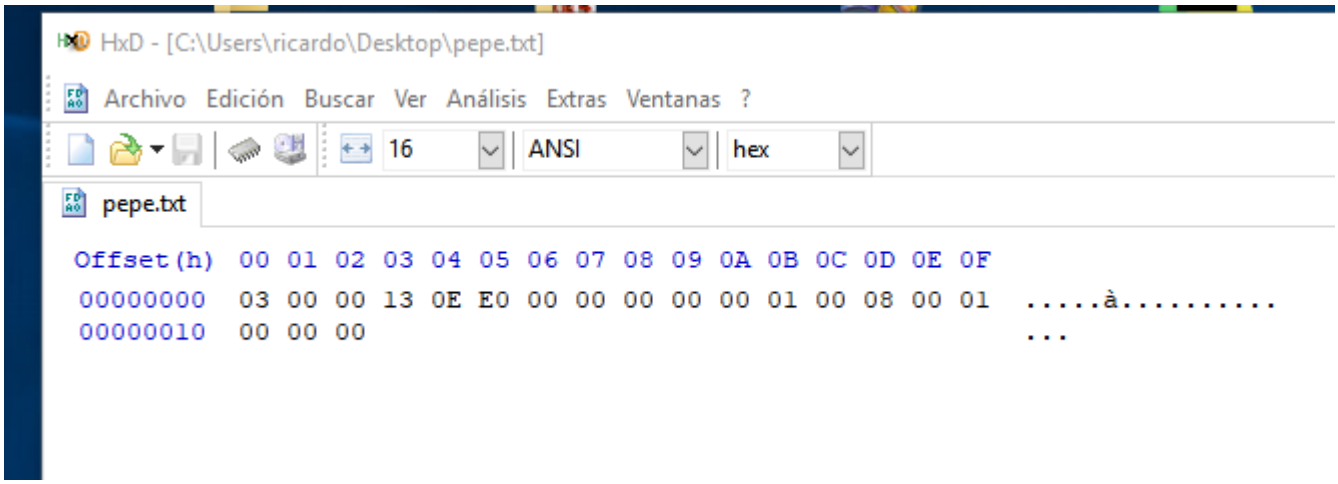
```
size=0x40
out=open(filename, "wb"
dbgr =True
data = GetManyBytes(address, size, use_dbg=dbgr)
out.write(data)
out.close()
```

This script saves in a file the bytes pointed by variable "address", the amount saved will be given by the variable "size", and saves it in a file on my desktop, I will adapt it to read the address and size from the registers at the point of breakpoint.

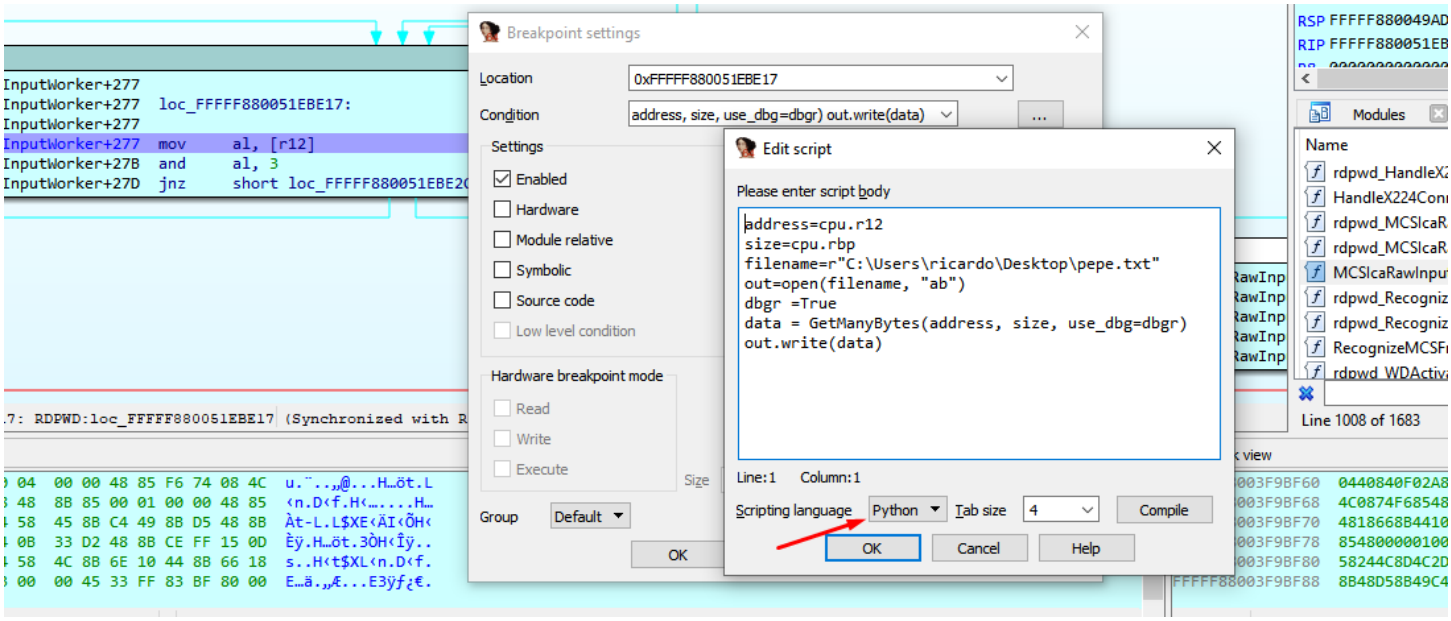
```
address=cpu.r12
size=cpu.rbp
filename=r"C:\Users\ricardo\Desktop\pepe.txt"
out=open(filename, "ab")
dbgr =True
data = GetManyBytes(address, size, use_dbg=dbgr)
out.write(data)
```

```
WINDBG>db r12
fffffa80`07350430  03 00 00 13 0e e0 00 00-00 00 00 01 00 08 00 01  ................
fffffa80`07350440  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
fffffa80`07350450  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
fffffa80`07350460  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
fffffa80`07350470  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
fffffa80`07350480  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
fffffa80`07350490  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
fffffa80`073504a0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
```

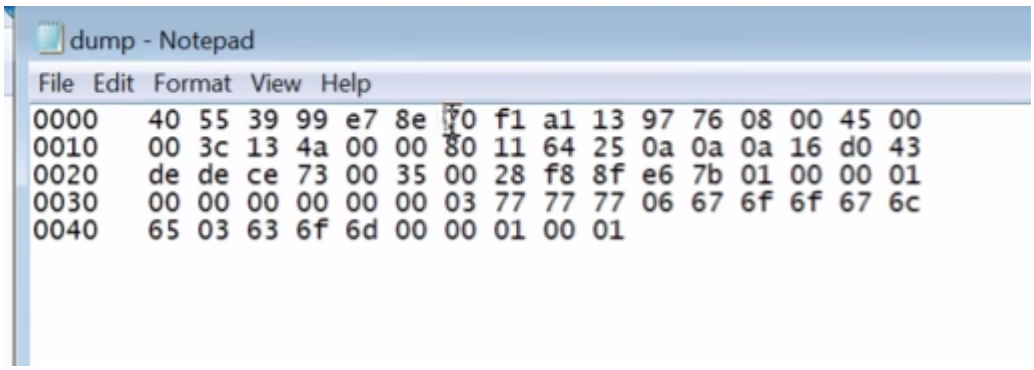
This will dump the bytes perfectly to the file.



I will use this script in the conditional breakpoint.

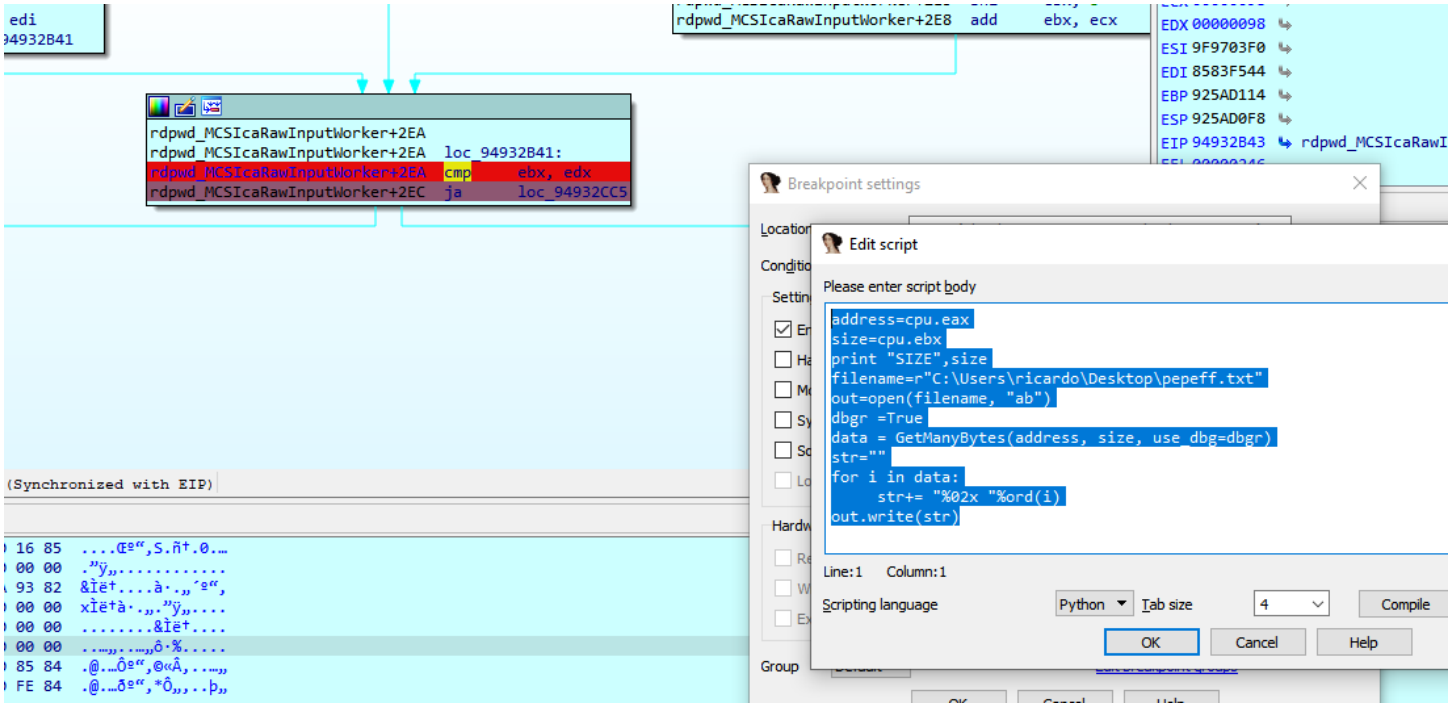


This script made a raw dump, but wireshark only imports in this format.




```
address=cpu.r12
size=cpu.rbp
filename=r"C:\Users\ricardo\Desktop\pepe.txt"
out=open(filename, "ab")
dbgr =True
data = GetManyBytes(address, size, use_dbg=dbgr)

str=""
for i in data:
    str+= "%02x "%ord(i)
out.write(str)
```

in Windows 7 32 bits version this is the important point where the decrypted code is parsed, and we can use this script to dump to a file.



Windows 7 32 script

address=cpu.eax

size=cpu.ebx

filename=r"C:\Users\ricardo\Desktop\pepefff.txt"

out=open(filename, "ab")

dbgr =True

data = GetManyBytes(address, size, use_dbg=dbgr)

str=""

for i in data:

    str+= "%02x "%ord(i)

out.write(str)

Windows XP 32 bits script

address=cpu.eax

size=cpu.edi

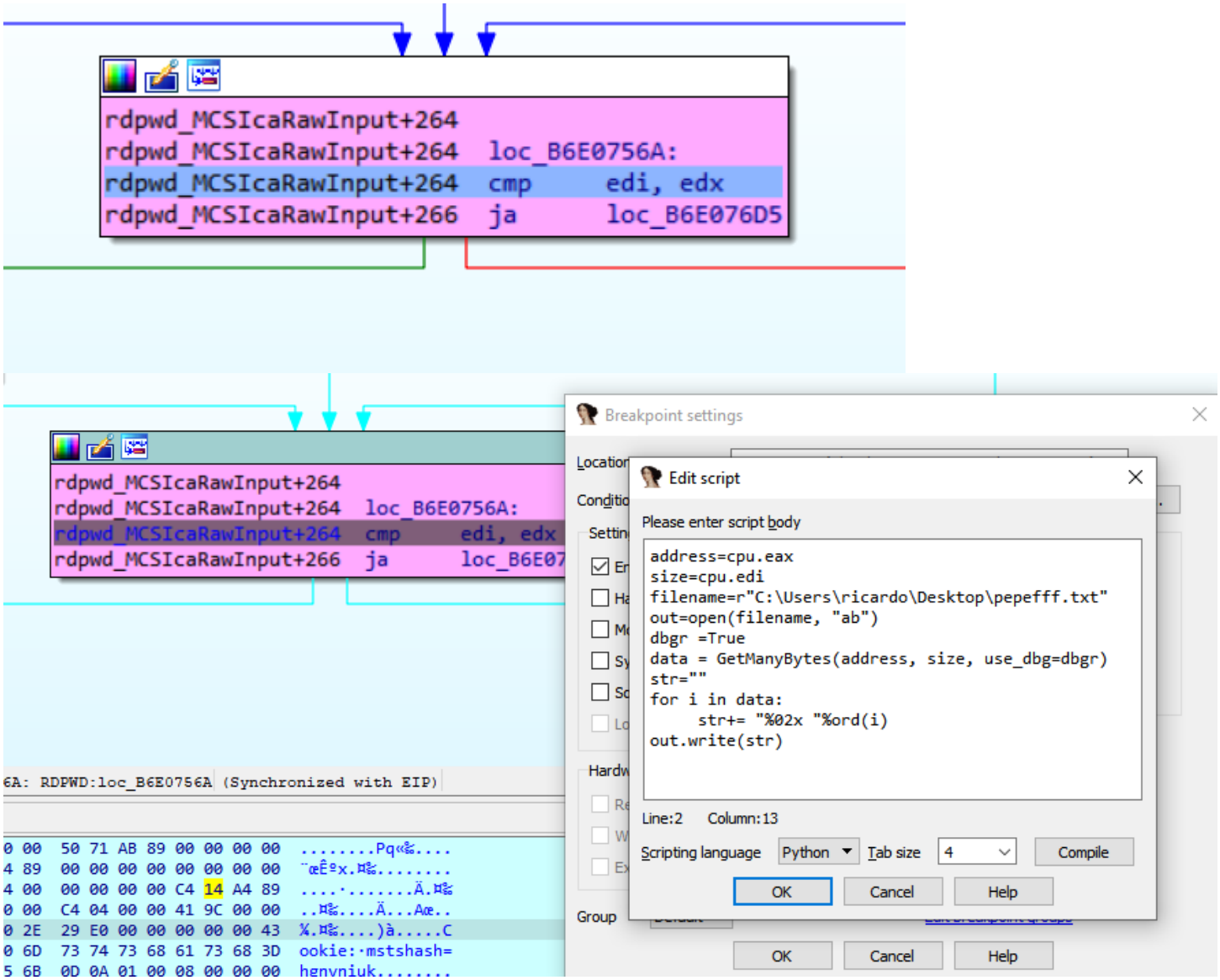
filename=r"C:\Users\ricardo\Desktop\pepefff.txt"

out=open(filename, "ab")

dbgr =True

```
data = GetManyBytes(address, size, use_dbg=dbgr)

str=""

for i in data:

    str+= "%02x "%ord(i)

out.write(str)
```

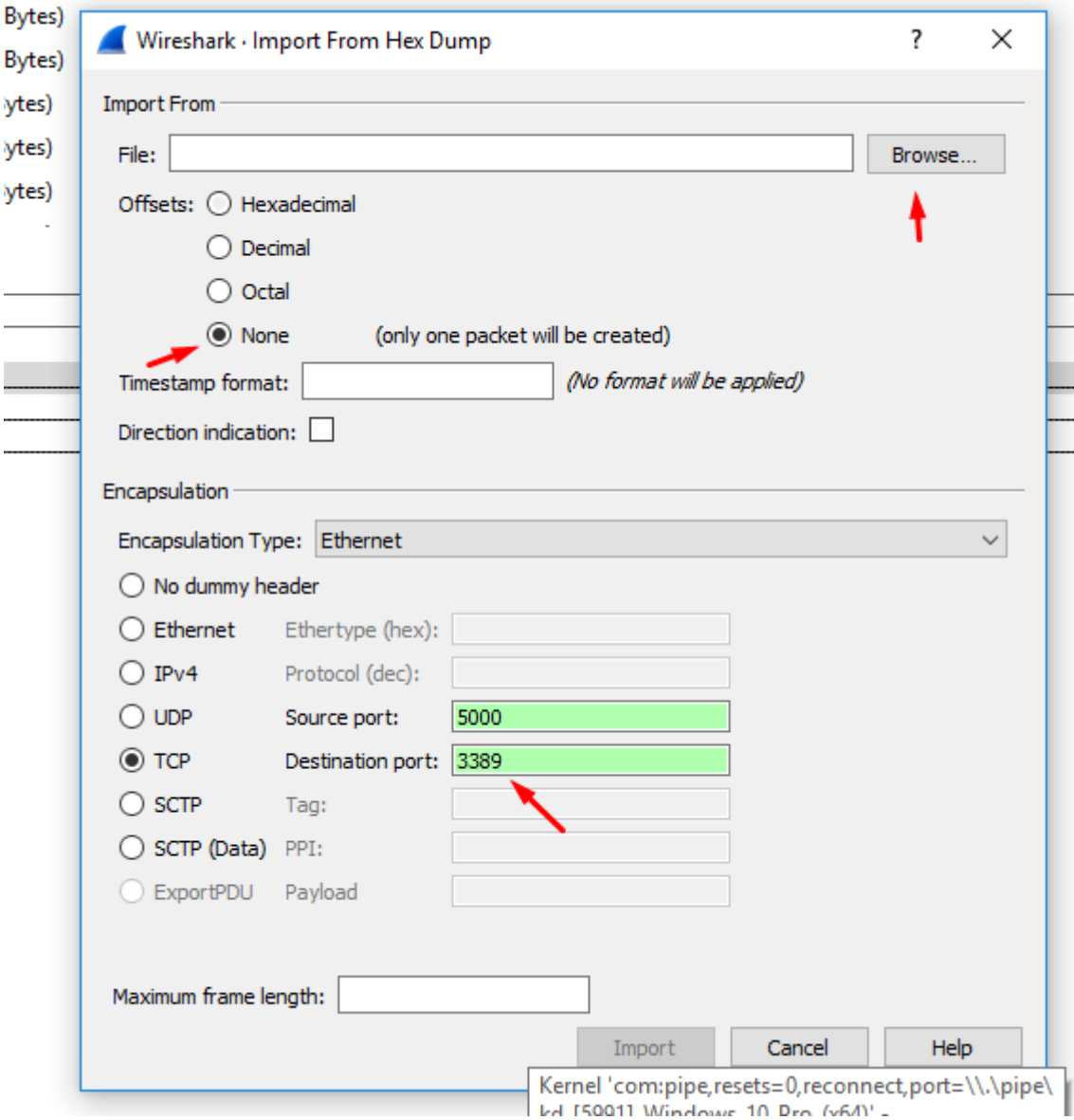
This is the similar point in Windows XP.



STEP 3) Importing to Wireshark

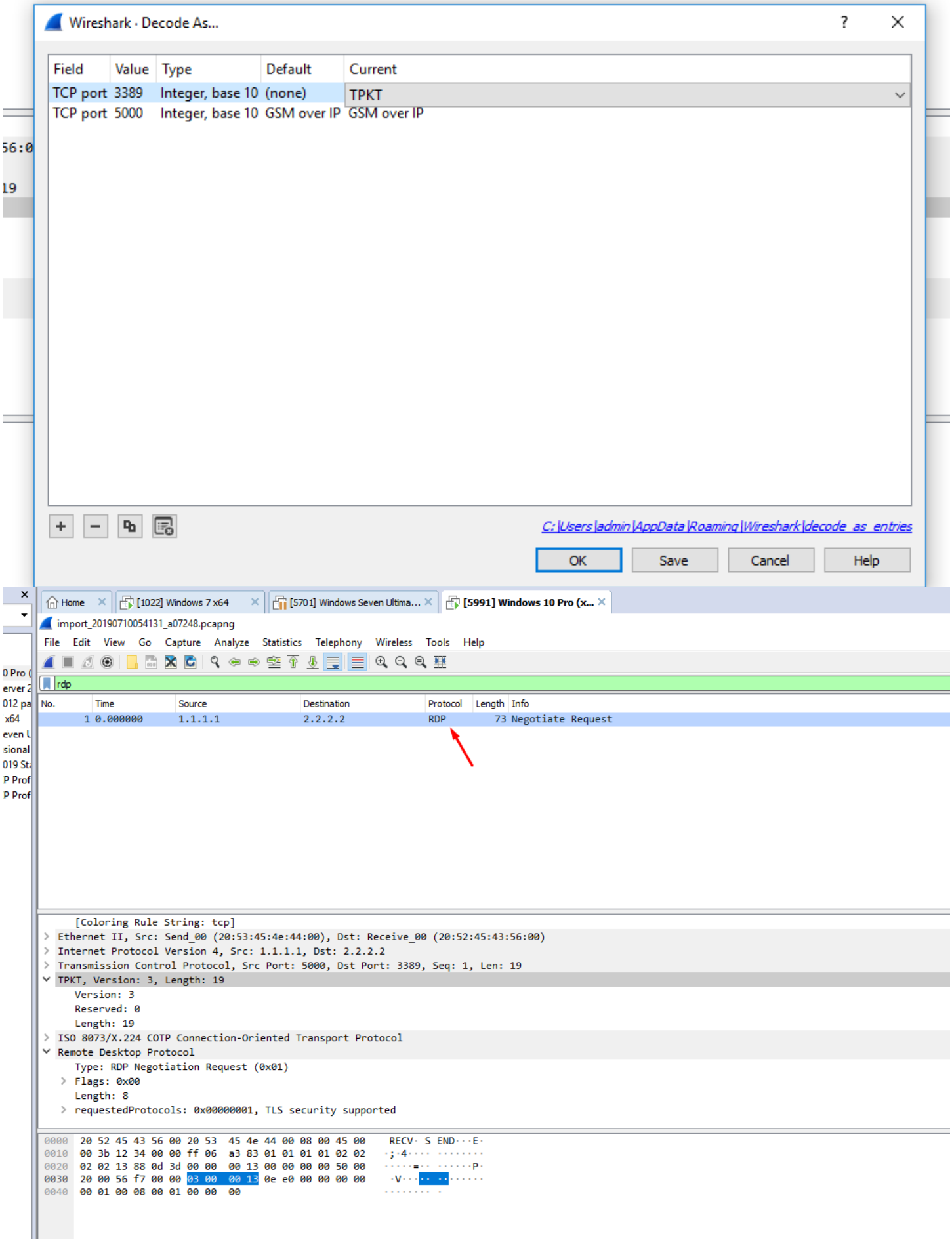
This script will save the bytes in the format that wireshark will understand.

When I I"Import from hex dump", I will use the port "3389/tcp" : "msrdp"



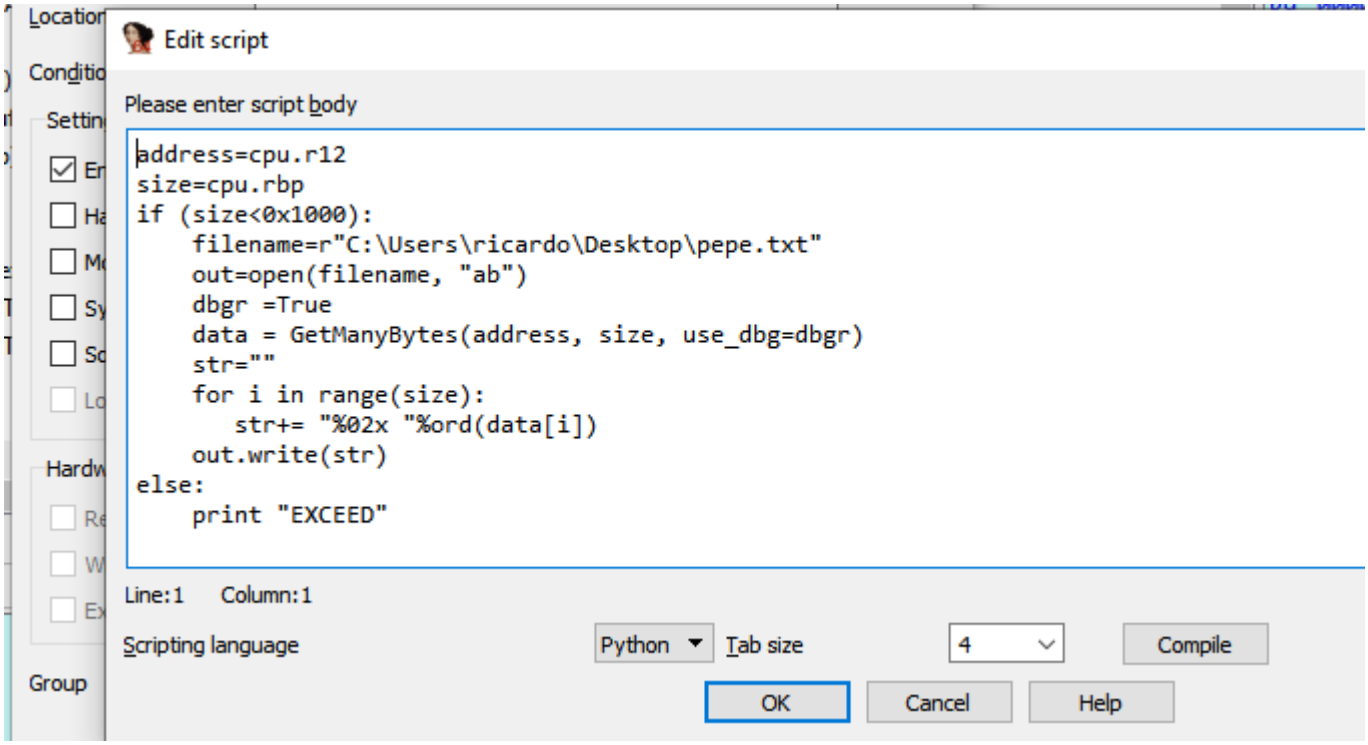
We load our dump file and put the destination port as 3389, the source port is not important.

I add a rule to decode port 3389 as TPKT

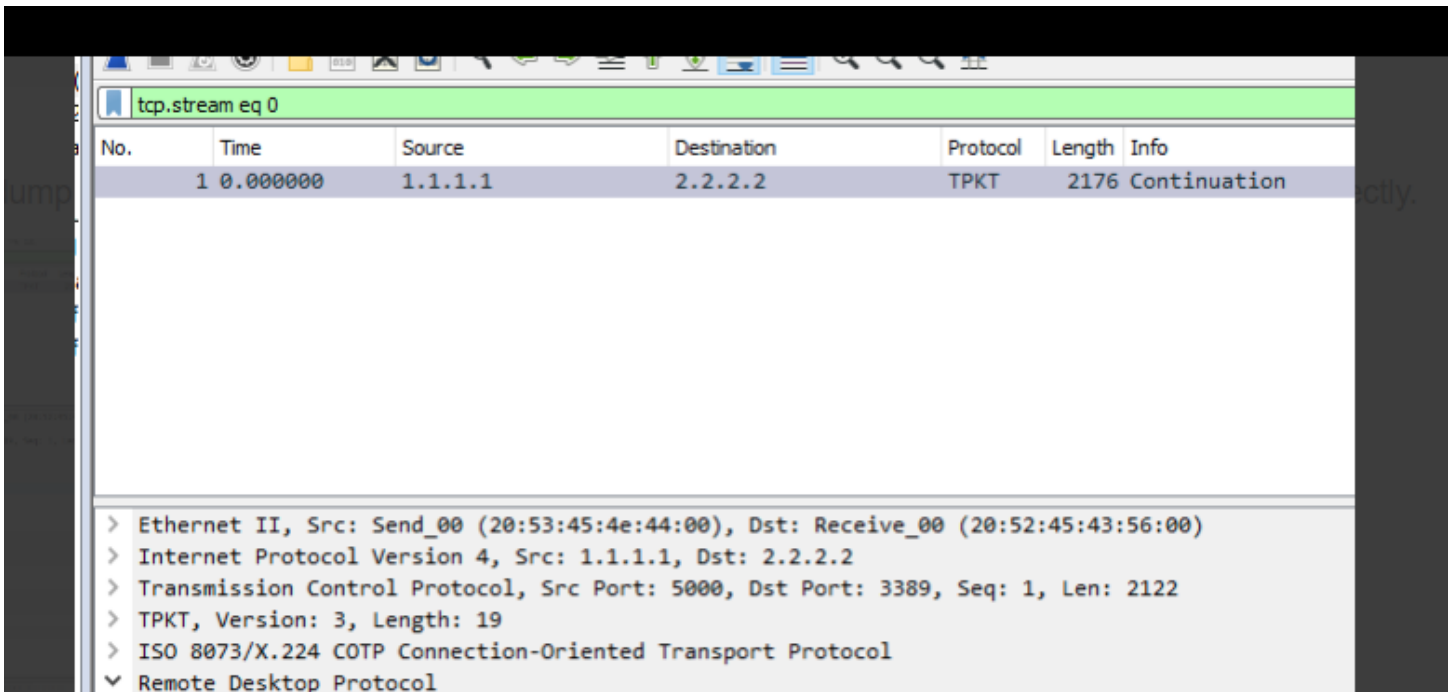


That file will be decoded as TPKT in wireshark.

That's the complete script.



Using that script, the dump file is created, when it is imported as a hexadecimal file in Wireshark, it is displayed perfectly.



This work can be done also by importing the SSL private key in wireshark, but I like to do it in the most manual way, old school type.

STEP 4) More reversing

We are ready to receive and analyze our first package, but first we must complete and analyze some more tasks that the program performs after what we saw before receiving the first package of our data.

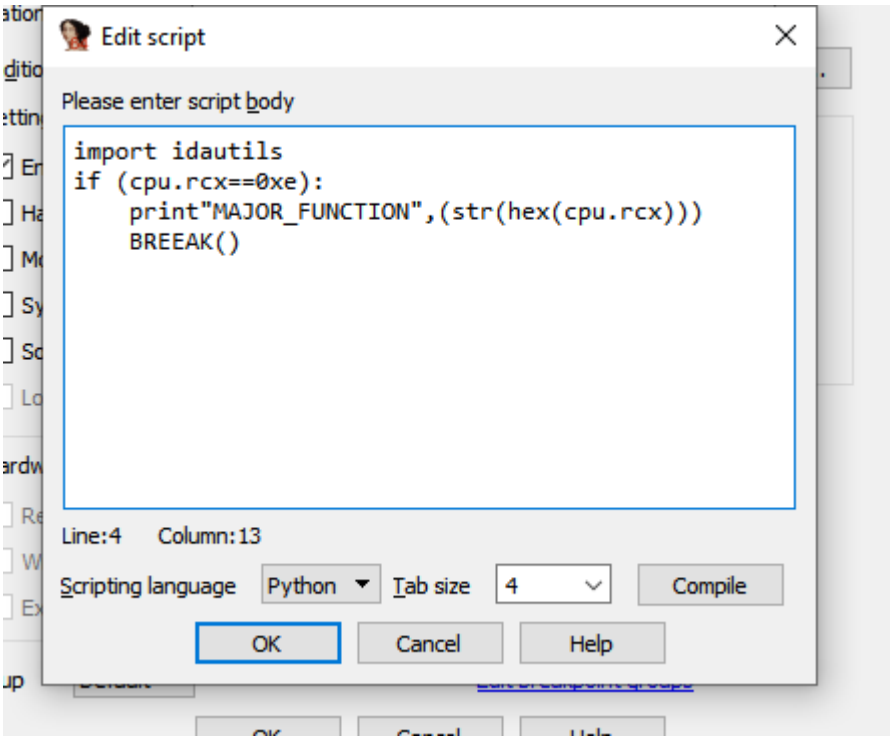
```
MAJOR_FUNCTION 0x0L
chunk_CONNECTION 0xfffffa8004491d60L
MAJOR_FUNCTION 0x0L
chunk_stack 0xBA8 0xfffffa8006ba7010L
MAJOR_FUNCTION 0x0L
chunk_channel 0xfffffa8005597ea0L
guarda RDI DESTINATION 0xfffffa8005597ea0L
MAJOR_FUNCTION 0x3L
```

We can see that there are a few more calls before starting to receive data, a couple more calls to the driver.

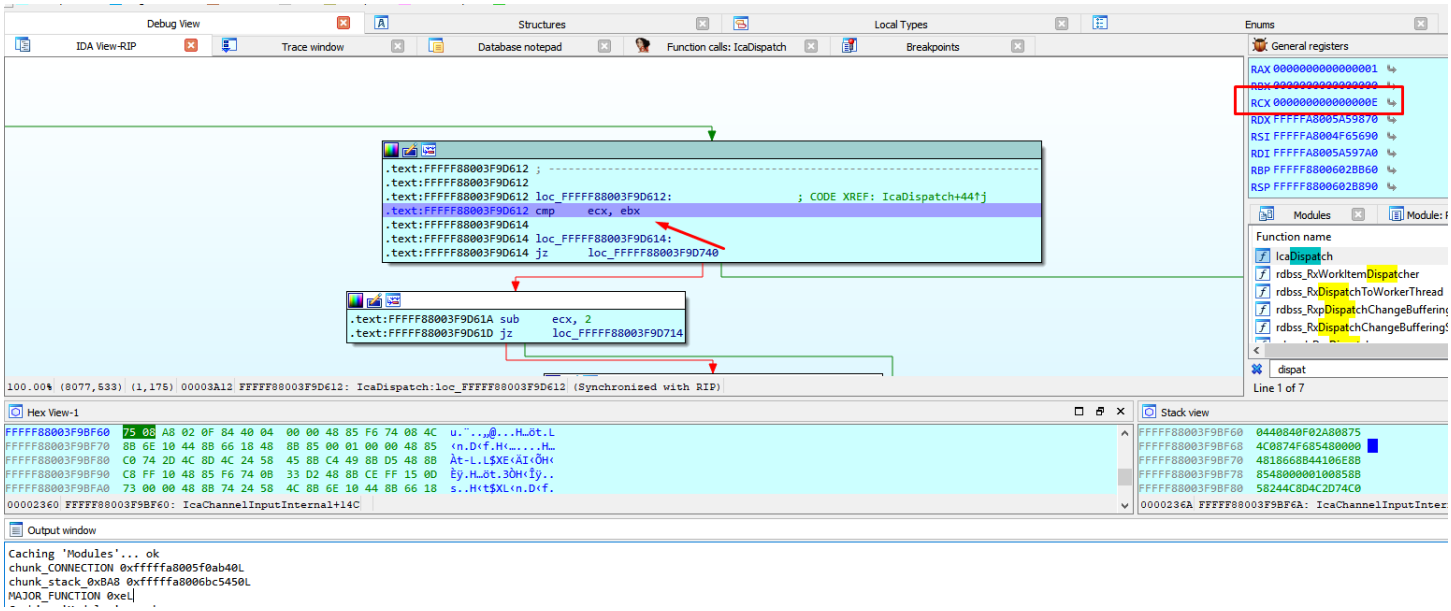
```
MAJOR_FUNCTION 0x0L
chunk_CONNECTION 0xfffffa80071cf010L
MAJOR_FUNCTION 0x0L
chunk_stack 0xBA8 0xfffffa8006b00450L
MAJOR_FUNCTION 0xeL
MAJOR_FUNCTION 0xeL
MAJOR_FUNCTION 0xeL
MAJOR_FUNCTION 0xeL
MAJOR_FUNCTION 0xeL
MAJOR_FUNCTION 0xeL
MAJOR_FUNCTION 0xeL
MAJOR_FUNCTION 0xeL
MAJOR_FUNCTION 0xeL
MAJOR_FUNCTION 0xeL
MAJOR_FUNCTION 0xeL
MAJOR_FUNCTION 0x0L
chunk_channel 0xfffffa800461c3c0L
guarda RDI DESTINATION 0xfffffa800461c3c0L
MAJOR_FUNCTION 0xeL
MAJOR_FUNCTION 0x3L
```

The part marked in red is what we have left to analyze from the first connection without sending data.

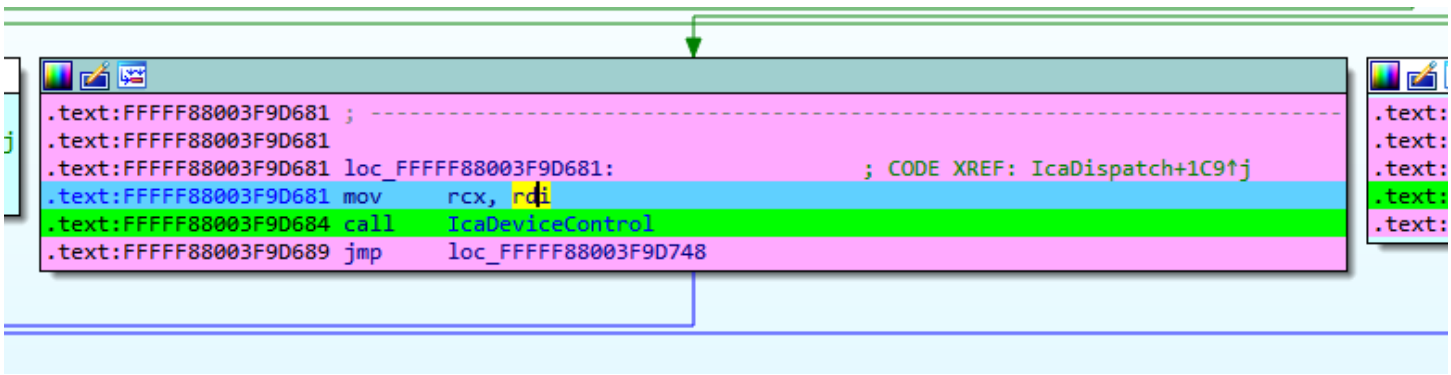
I will modify the conditional breakpoint to stop at the first MajorFunction = 0xE.



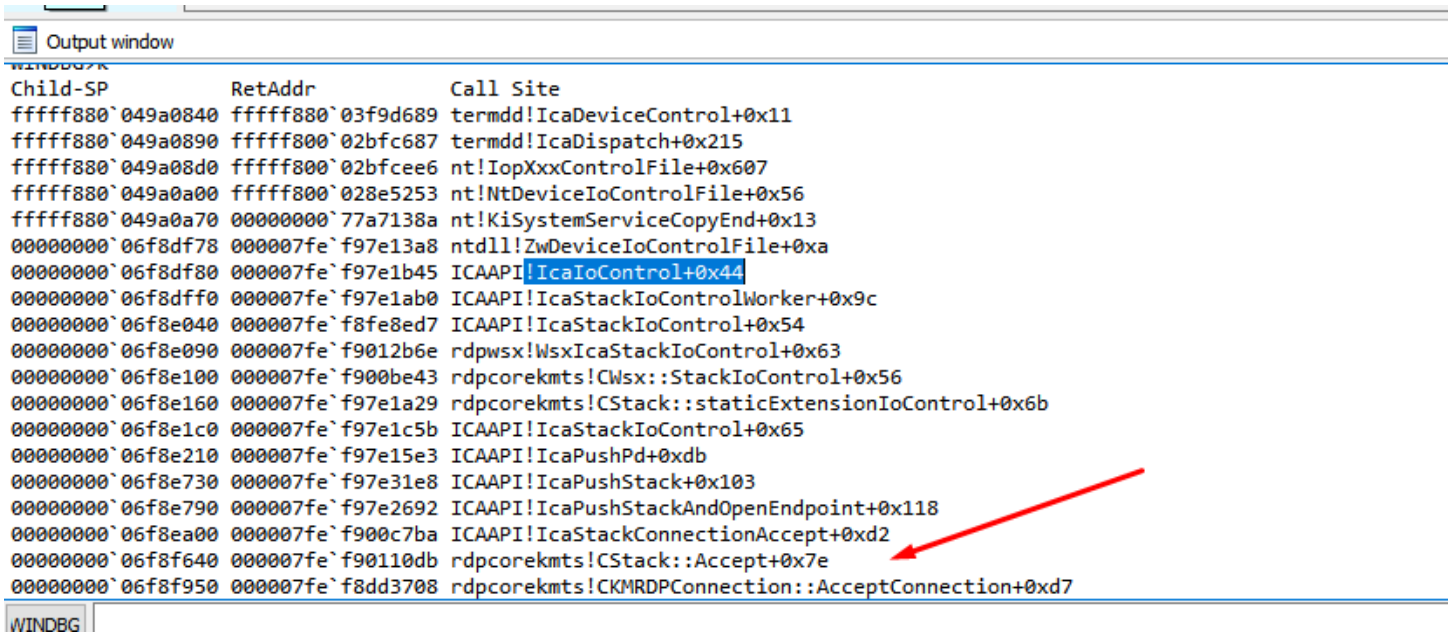
It will stop when MajorFunction = 0xE



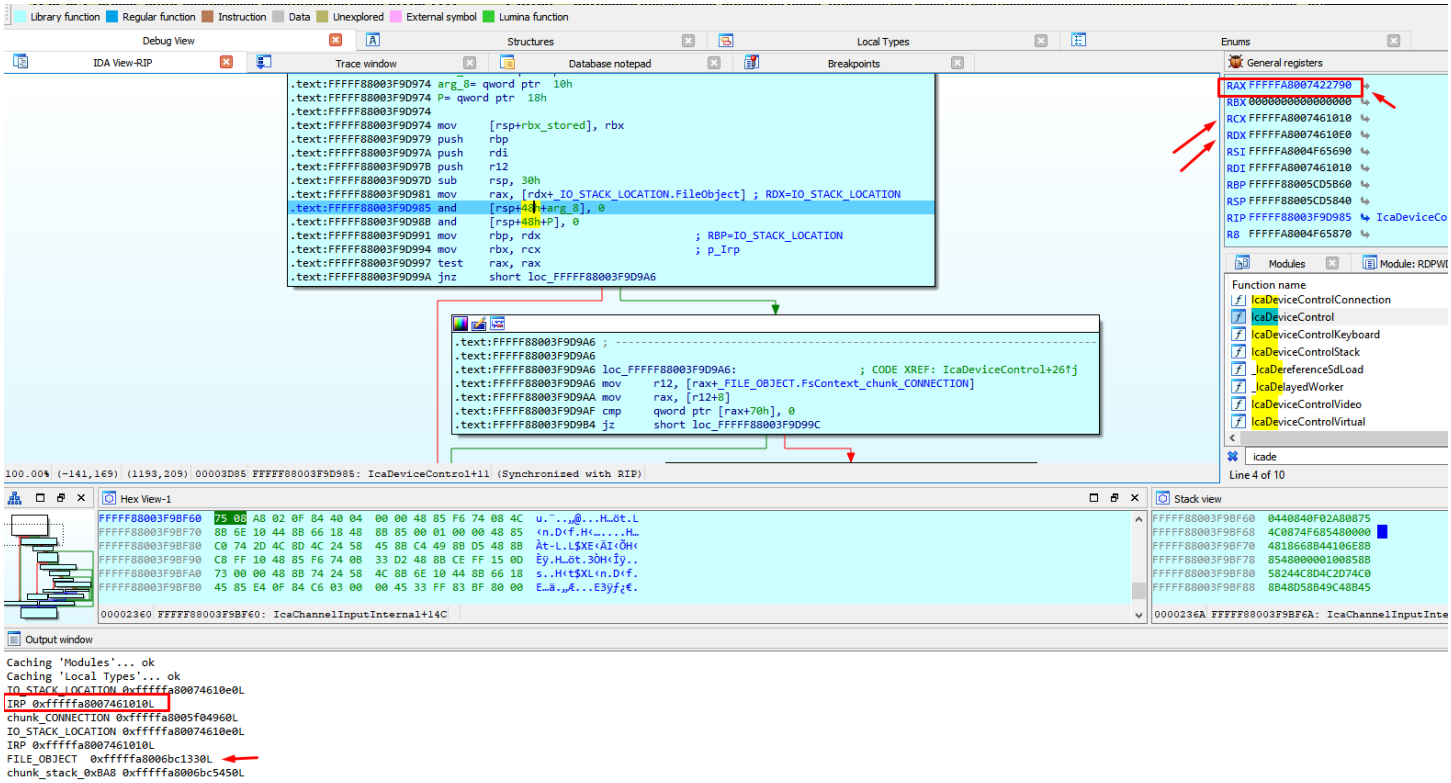
We arrived at IcaDeviceControl.



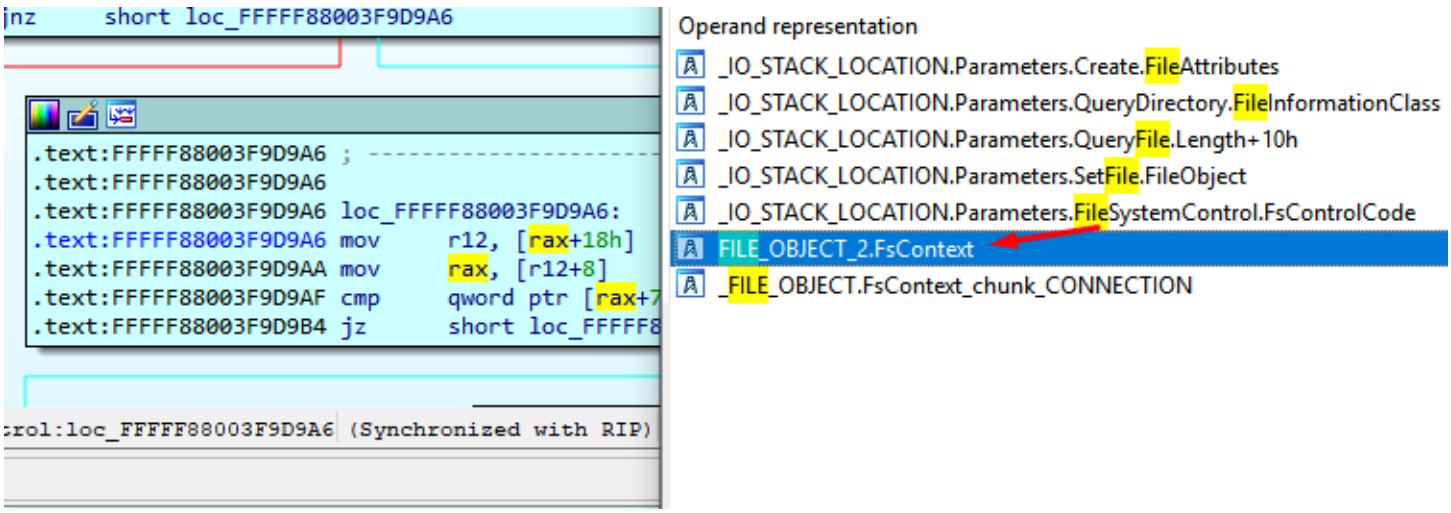
We can see that this call is generated when the program accepts the connection, calling ZwDeviceIoControlFile next.



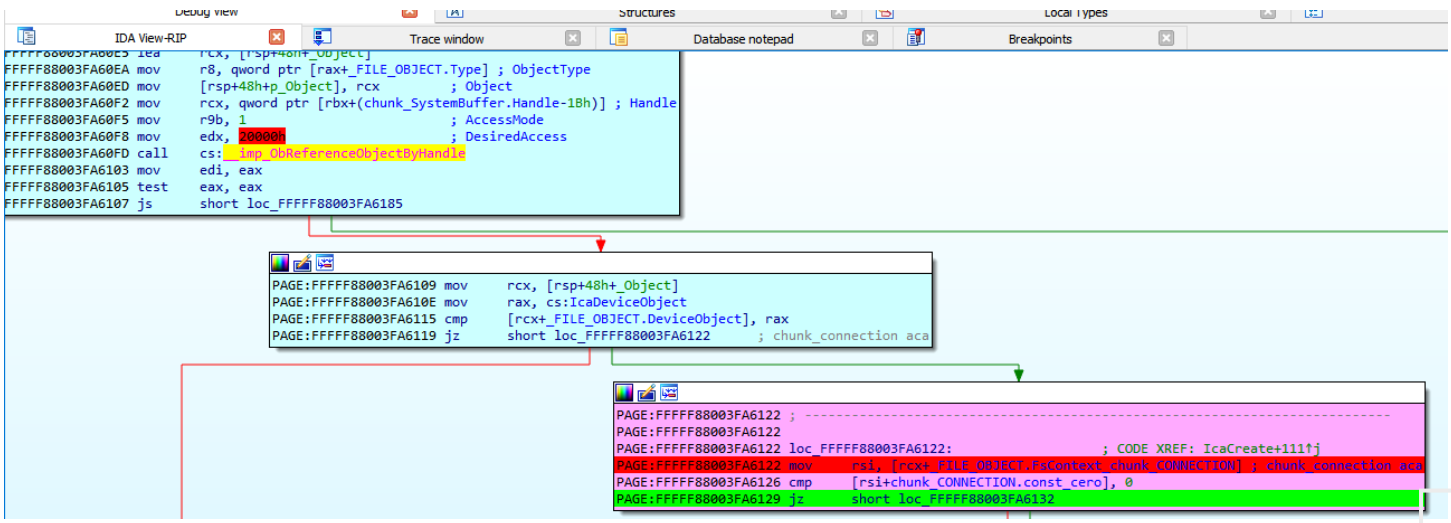
We can see that IRP and IO_STACK_LOCATION are maintained with the same value, fileobject has changed.



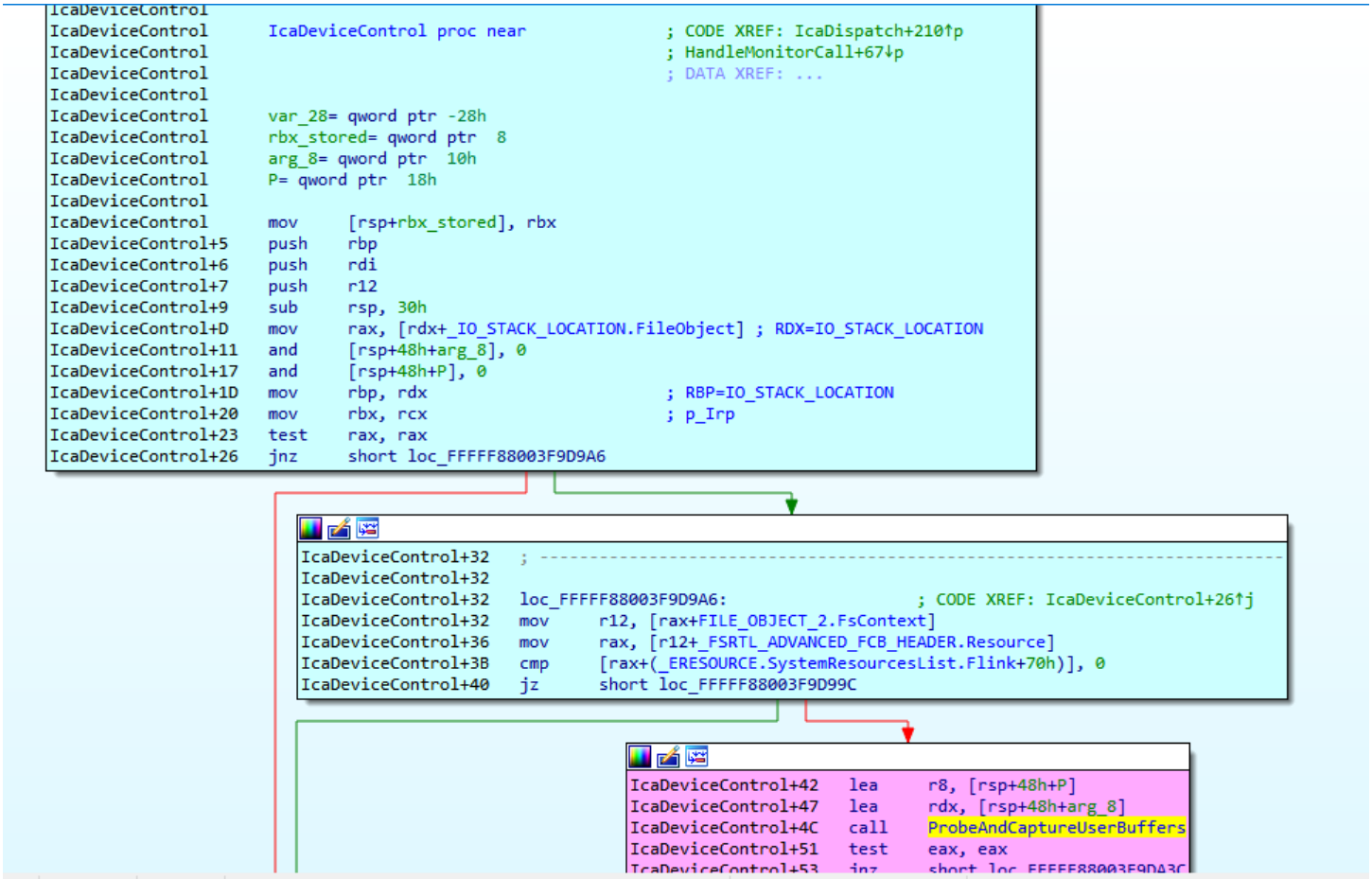
We will leave the previous structure called FILE_OBJECT for the previous call, and we will make a copy with the original fields called FILE_OBJECT_2, to be used in this call.



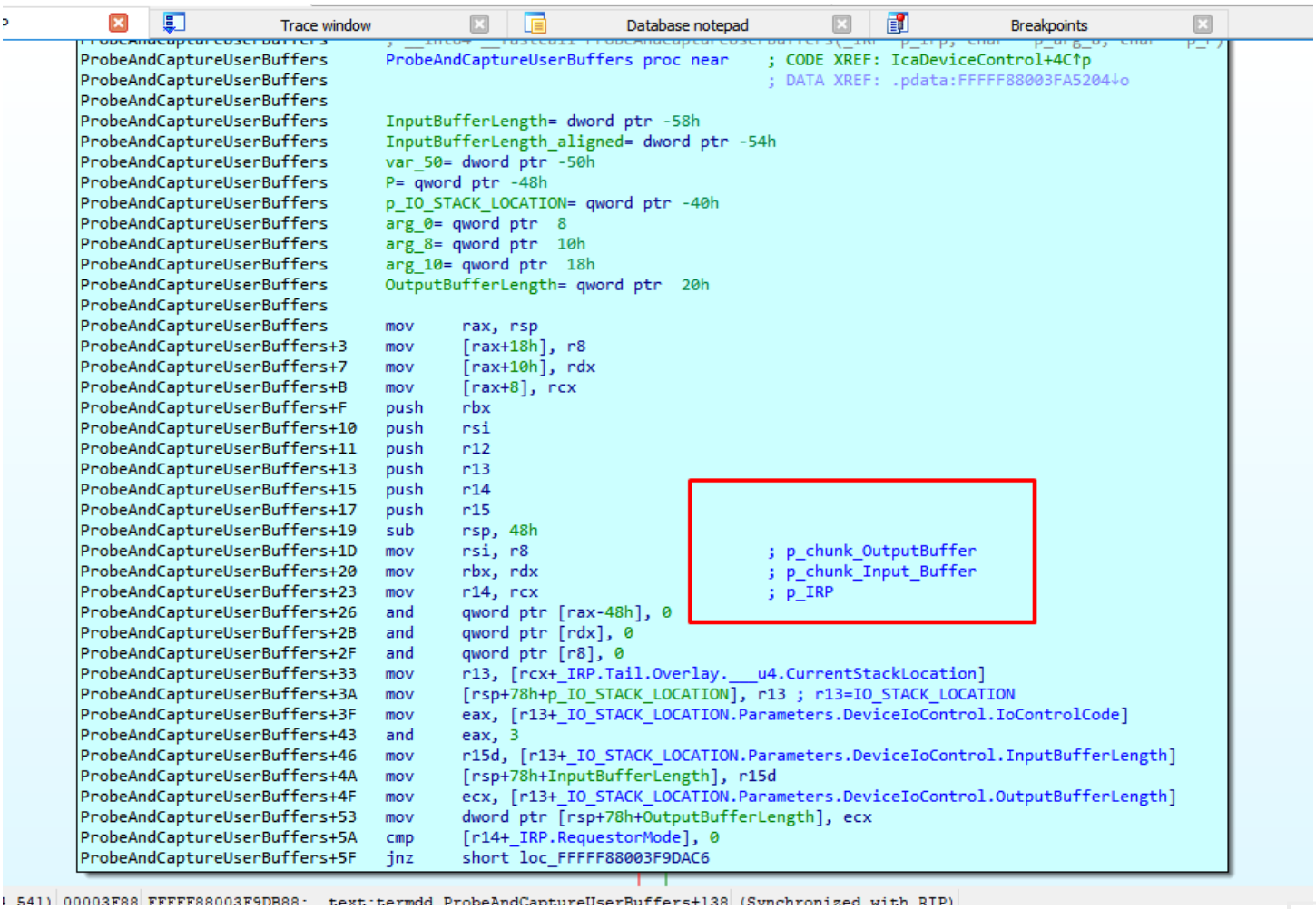
The previous FILE_OBJECT was an object that was obtained from ObReferenceObjectByHandle.



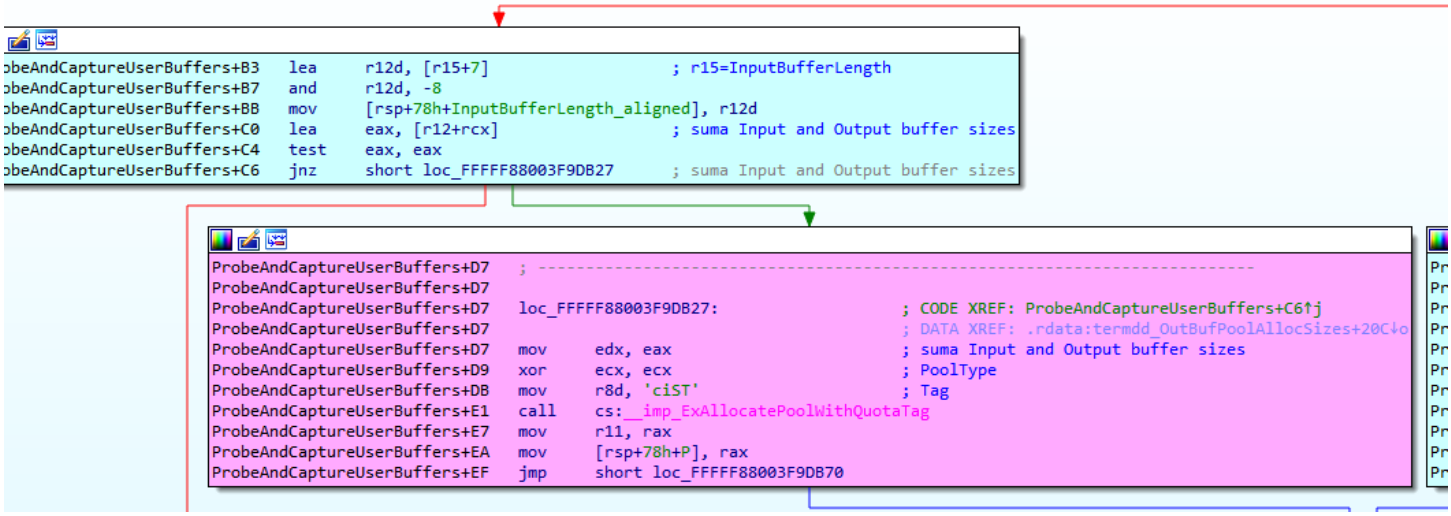
The new FILE_OBJECT has the same structure but is a different object, for that reason we create a new structure for this.



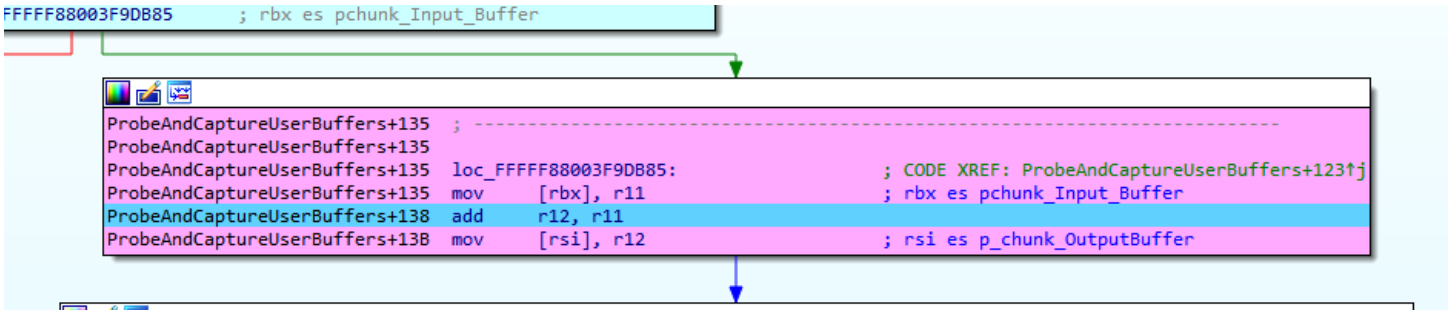
We continue reversing ProbeAndCaptureUserBuffers



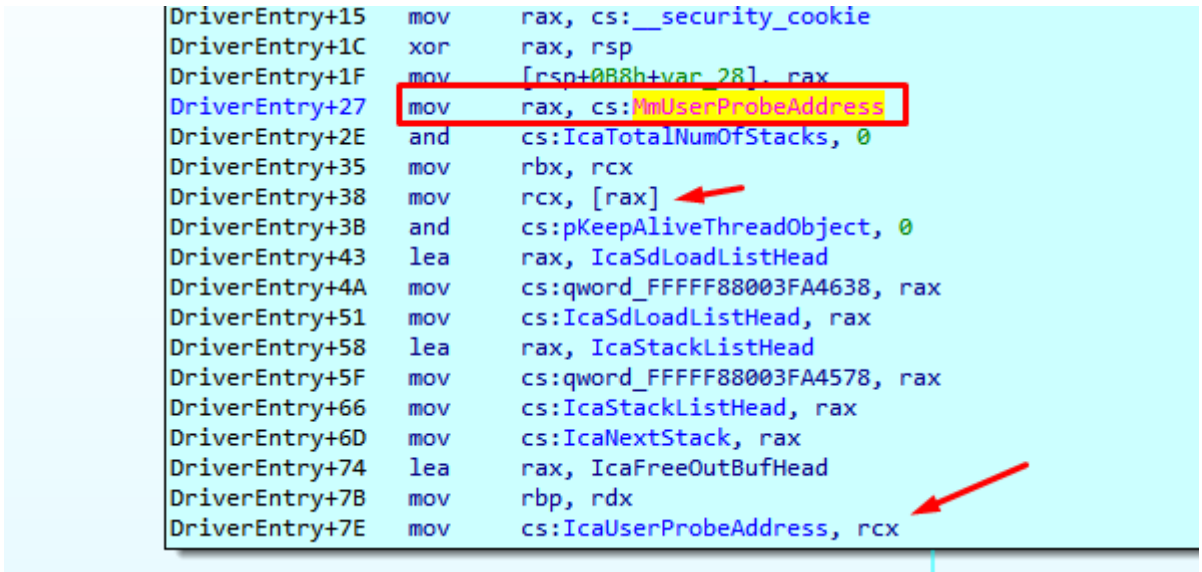
A new chunk with the size (InputBufferLenght + OutputBufferLenght) is created.



Stores the pointers to the Input and Output buffers chunks.

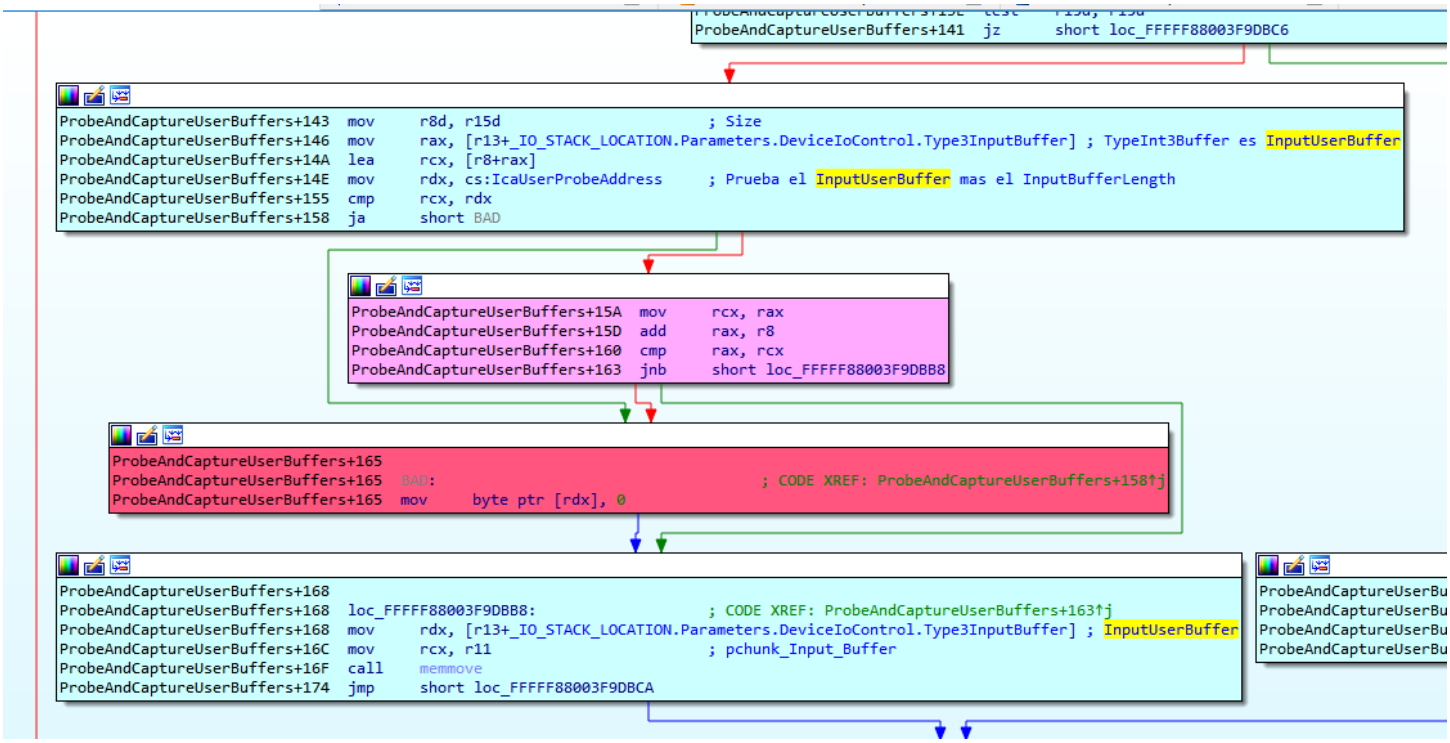


We can see that IcaUserProbeAddress is similar to nt! MmUserProbeAddress value



That's used to verify whether a user-specified address resides within user-mode memory areas, or not.

If the address is lower than IcaUserProbeAddress resides in User mode memory areas, and a second check is performed to ensure than the InputUserBuffer + InputBufferLenght address is bigger than InputUserBuffer address.(size not negative)

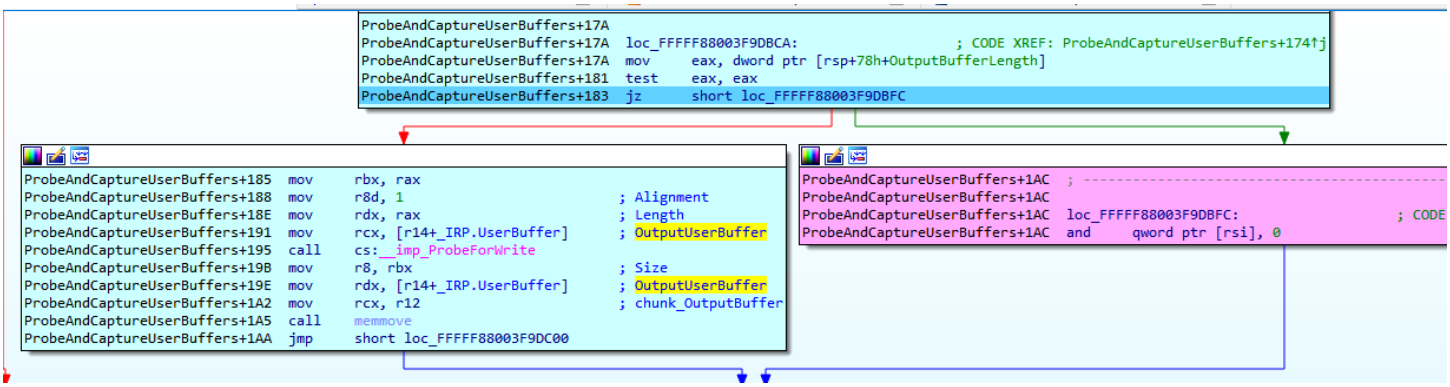


Then the data is copied from the InputUserBuffer to the chunk_Input_Buffer that has just allocated for this purpose.

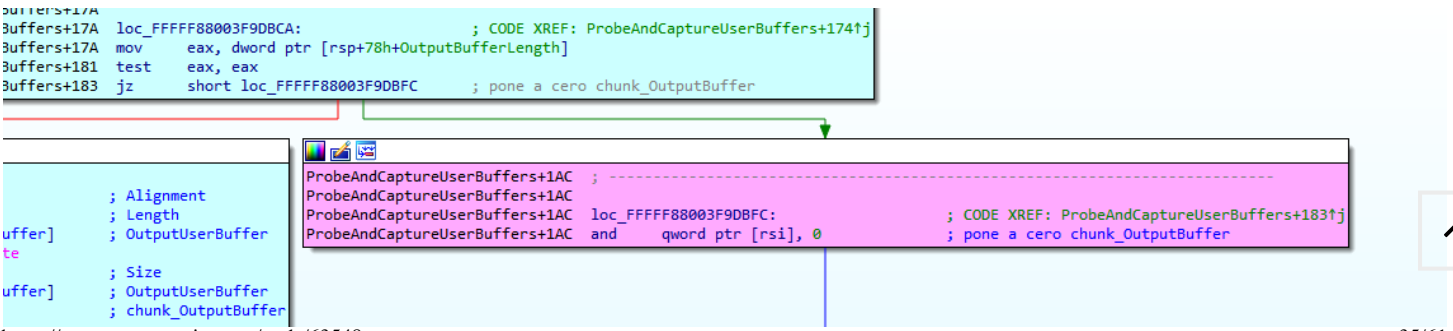
We can see the data that the program copies from InputUserBuffer, it's not data that we send yet.

```
WINDBG>db rdx
00000000`06f8e210  00 00 00 00 74 00 64 00-74 00 63 00 70 00 00 00  ....t.d.t.c.p...
00000000`06f8e220  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
00000000`06f8e230  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
00000000`06f8e240  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
00000000`06f8e250  00 00 00 00 00 00 00 00-00 00 00 00 4d 00 69 00  ............M.i.
00000000`06f8e260  63 00 72 00 6f 00 73 00-6f 00 66 00 74 00 20 00  c.r.o.s.o.f.t. .
00000000`06f8e270  52 00 44 00 50 00 20 00-37 00 2e 00 31 00 00 00  R.D.P. .7...1...
00000000`06f8e280  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
```

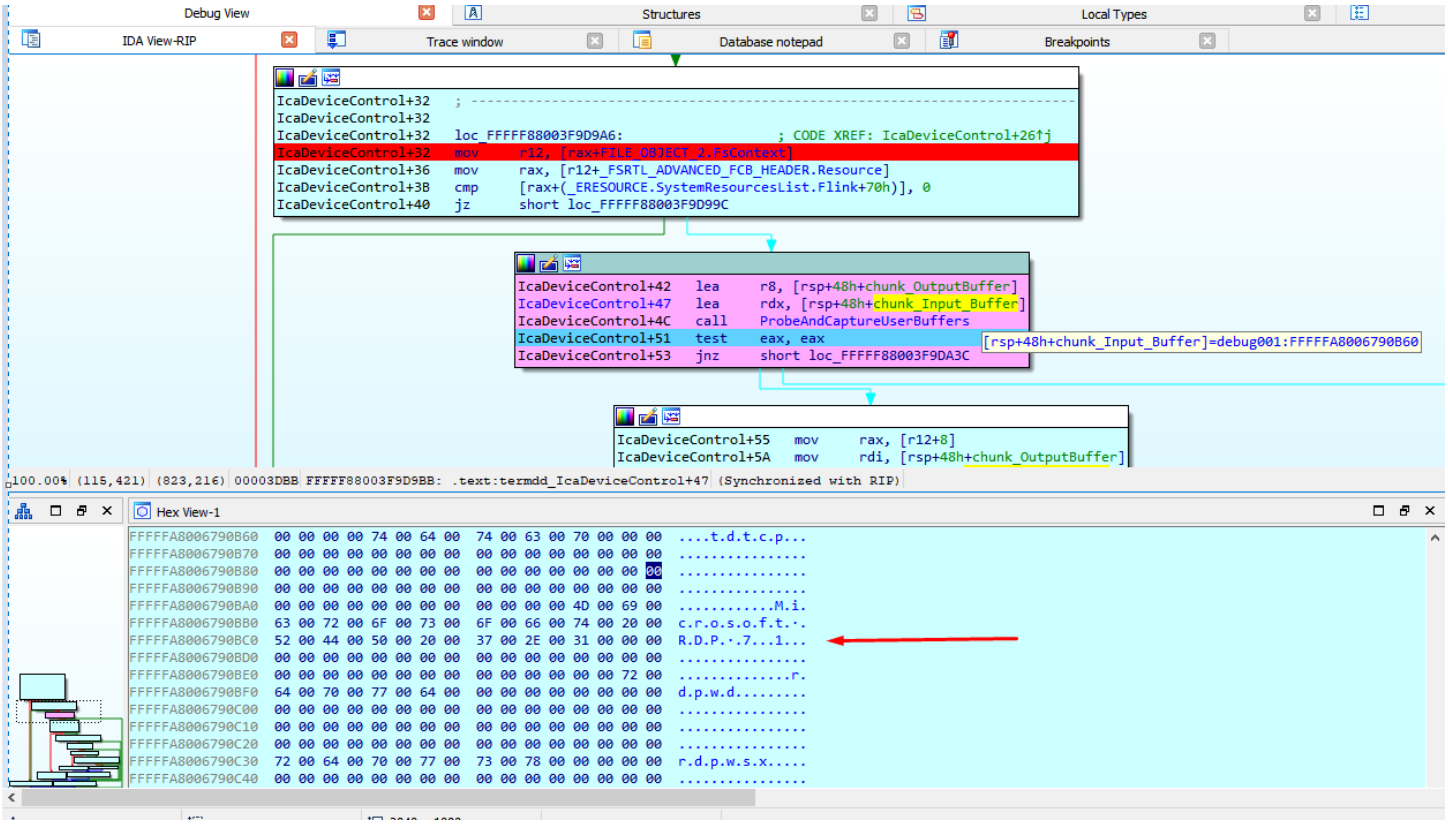
Since the OutputBufferLength is zero, it will not copy from OutputUserBuffer to the chunk_OutputBuffer.



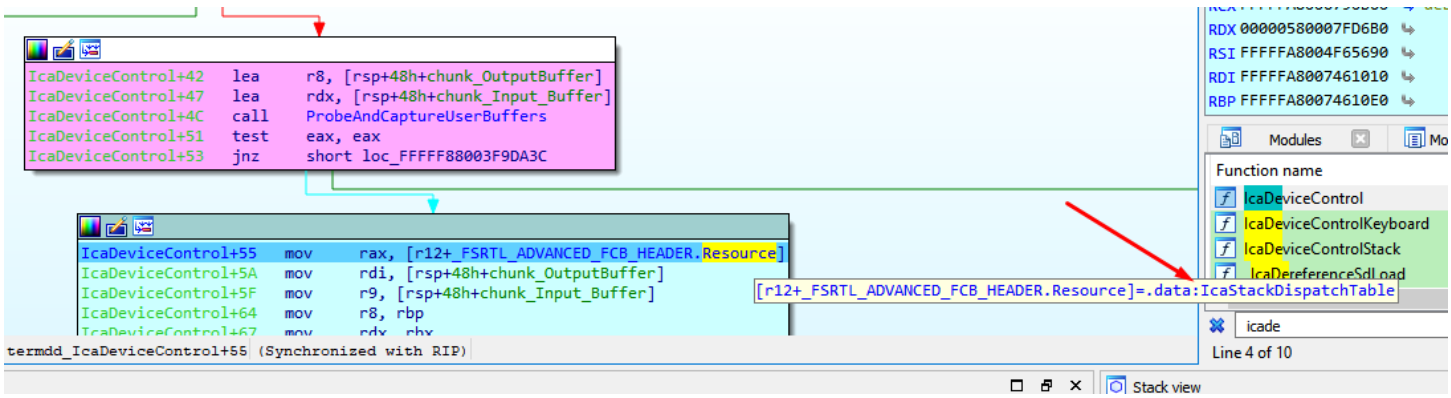
Clears chunk_OutputBuffer and return.



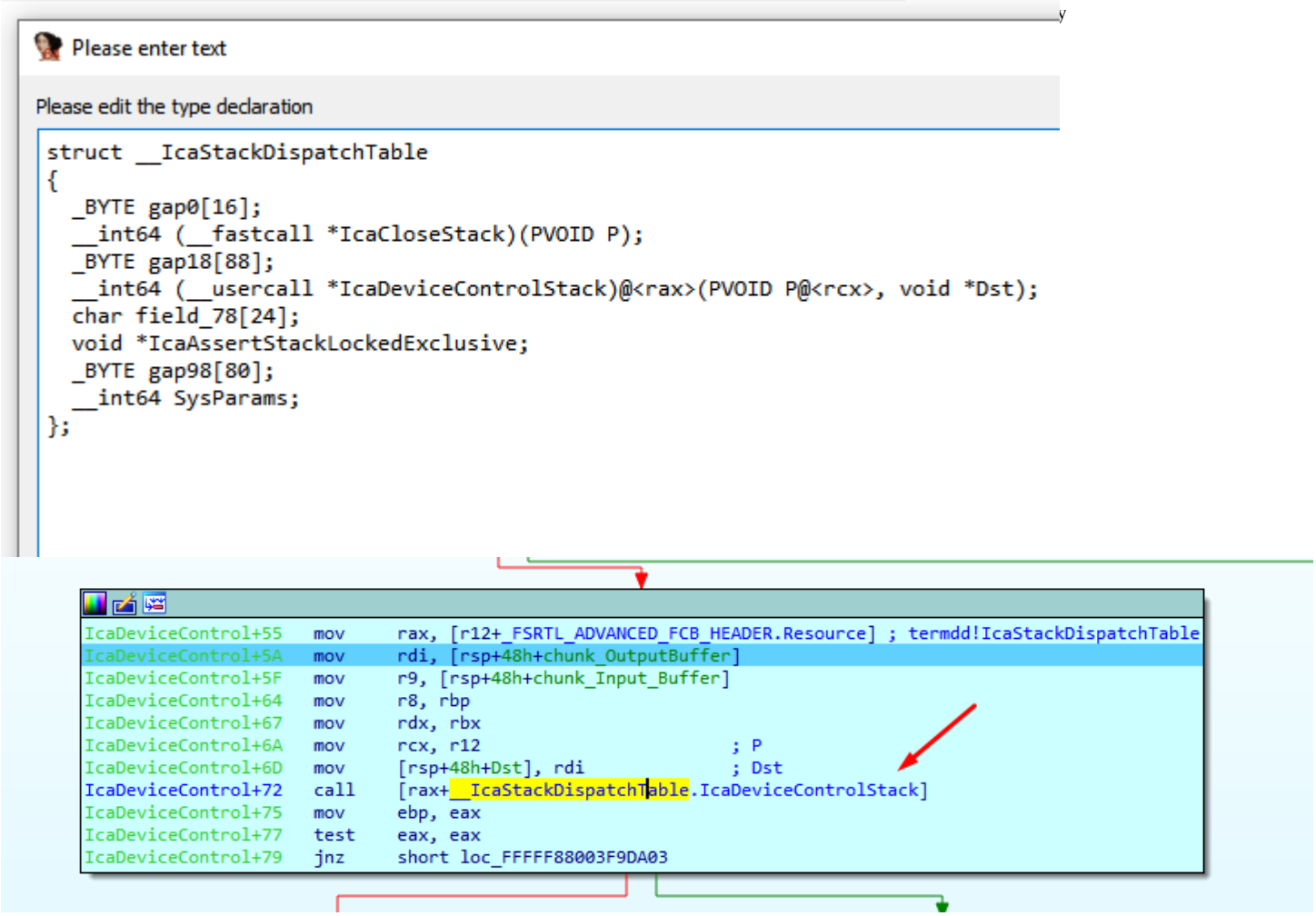
Returning from ProbeAndCaptureUserBuffers, we can see that this function copies the input and output buffer of the user mode memory to the new chunks allocated in the kernel memory, for the handling of said data by the driver



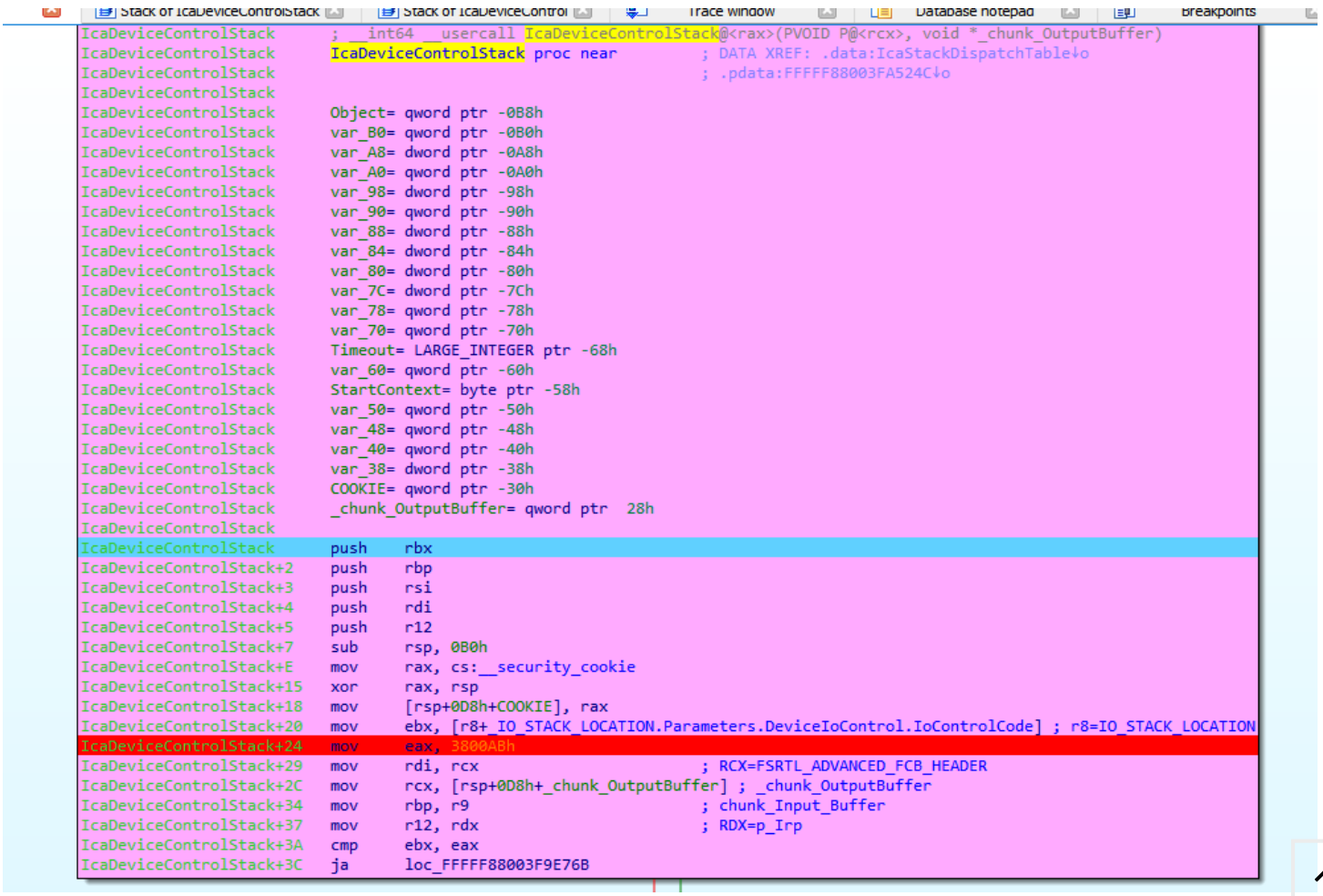
The variable "resource" points to IcaStackDispatchTable.



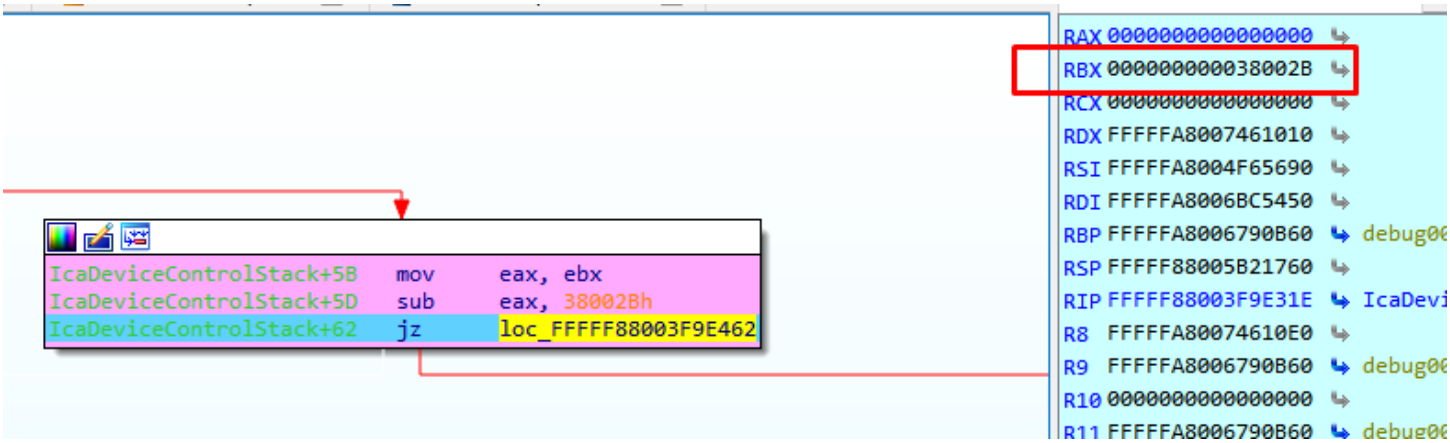
I frame the area of the table and create a structure from memory which I call _IcaStackDispatchTable.



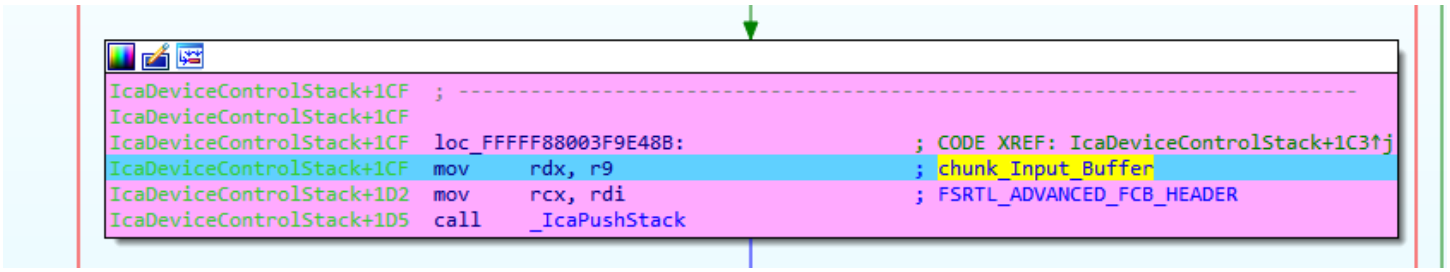
I entered and started to reverse this function.



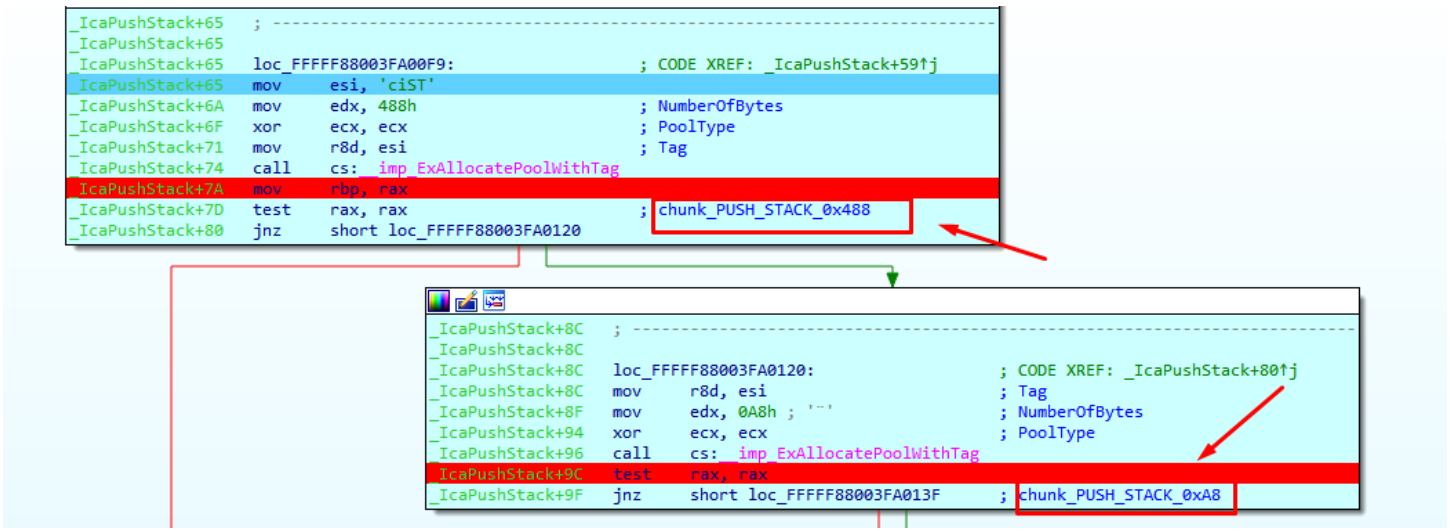
The first time we arrived here, the IOCTL value is 38002b.



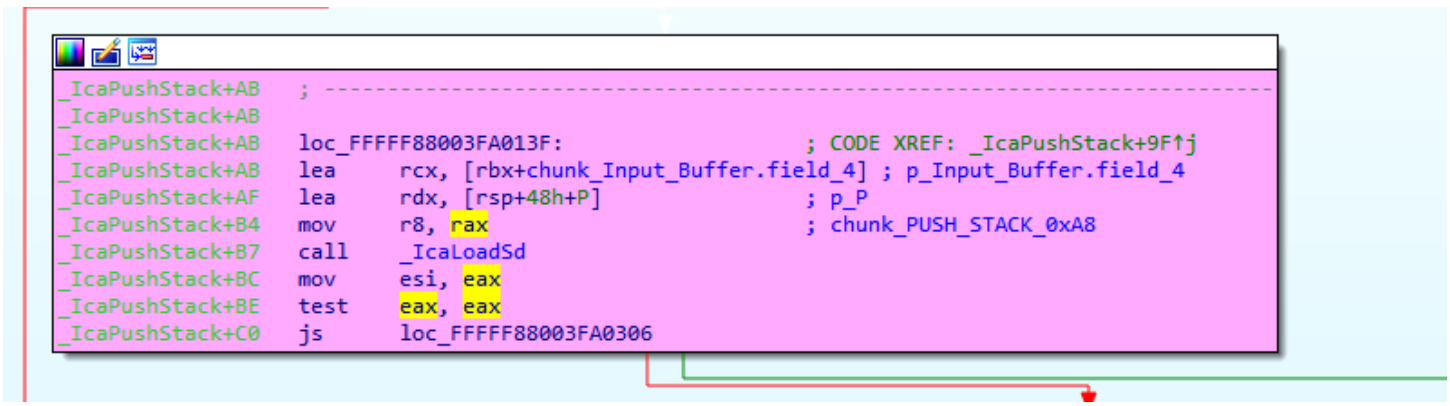
We arrived to a call to _IcaPushStack.



Inside two allocations are performed, i named them **chunk_PUSH_STACK_0x488** and **chunk_PUSH_STACK_0xA8**



When IOCTL value 0x38002b is used, we reach _IcaLoadSd



We can see the complete log of the calls to the driver with different IOCTL only in the connection without sending data yet.

IO_STACK_LOCATION 0xfffffa80061bea90L

IRP 0xfffffa80061be9c0L

chunk_CONNECTION 0xfffffa8006223510L

IO_STACK_LOCATION 0xfffffa80061bea90L

IRP 0xfffffa80061be9c0L

FILE_OBJECT 0xfffffa8004231860L

chunk_stack_0xBA8 0xfffffa80068d63d0L

FILE_OBJECT_2 0xfffffa80063307b0L

IOCTL 0x380047L

FILE_OBJECT_2 0xfffffa8006335ae0L

IOCTL 0x38002bL

chunk_PUSH_STACK_0x488 0xfffffa8006922a20L

chunk_PUSH_STACK_0xa8 0xfffffa8005ce0570L

FILE_OBJECT_2 0xfffffa8006335ae0L

IOCTL 0x38002bL

chunk_PUSH_STACK_0x488 0xfffffa8005f234e0L

chunk_PUSH_STACK_0xa8 0xfffffa8006875ba0L

FILE_OBJECT_2 0xfffffa8006335ae0L

IOCTL 0x38002bL

chunk_PUSH_STACK_0x488 0xfffffa8005daf010L

chunk_PUSH_STACK_0xa8 0xfffffa8006324c40L

FILE_OBJECT_2 0xfffffa8006335ae0L

IOCTL 0x38003bL

FILE_OBJECT_2 0xfffffa8006335ae0L

IOCTL 0x3800c7L

FILE_OBJECT_2 0xffffa8006335ae0L

IOCTL 0x38244fL

FILE_OBJECT_2 0xffffa8006335ae0L

IOCTL 0x38016fL

FILE_OBJECT_2 0xffffa8006335ae0L

IOCTL 0x380173L

FILE_OBJECT_2 0xffffa8006334c90L

FILE_OBJECT_2 0xffffa8006335ae0L

IOCTL 0x38004bL

IO_STACK_LOCATION 0xffffa8004ceb9d0L

IRP 0xffffa8004ceb900L

FILE_OBJECT 0xffffa8006334c90L

chunk_channel 0xffffa8006923240L

guarda RDI DESTINATION 0xffffa8006923240L

FILE_OBJECT_2 0xffffa8006335ae0L

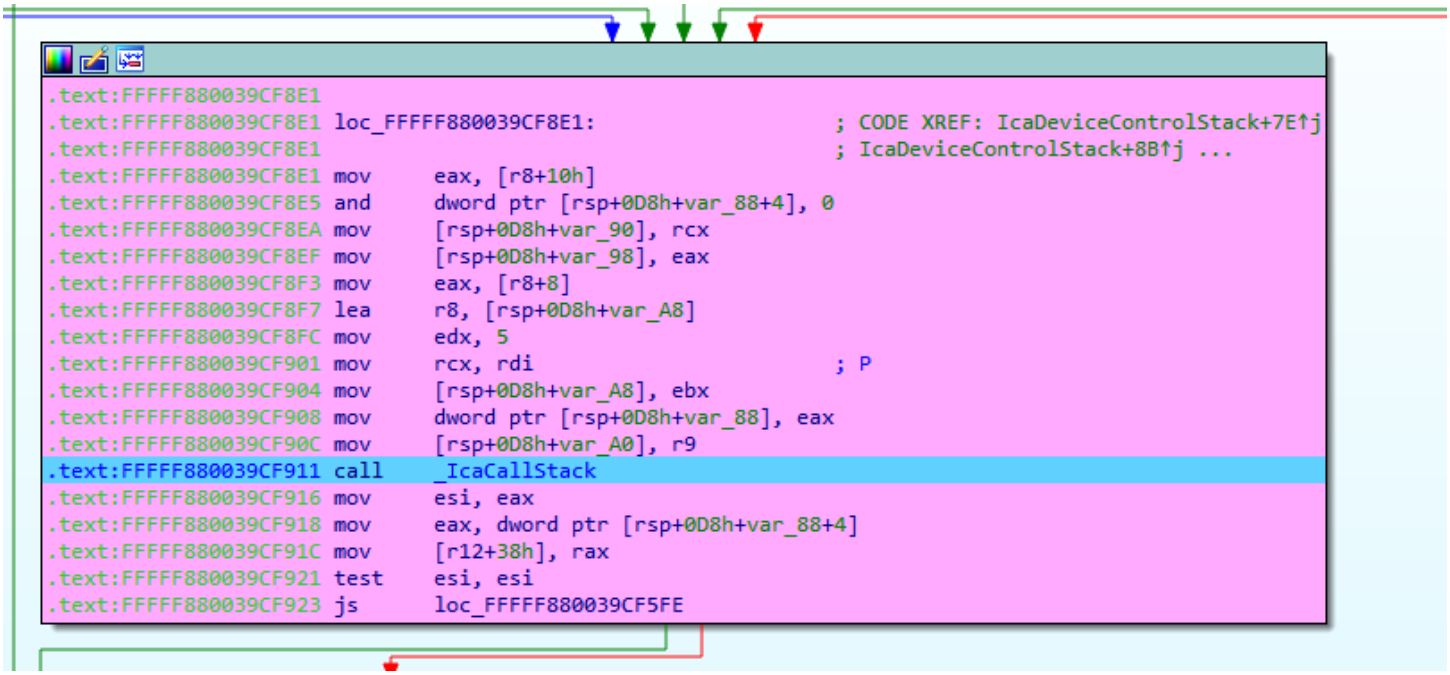
IOCTL 0x381403L

FILE_OBJECT_2 0xffffa8006335ae0L

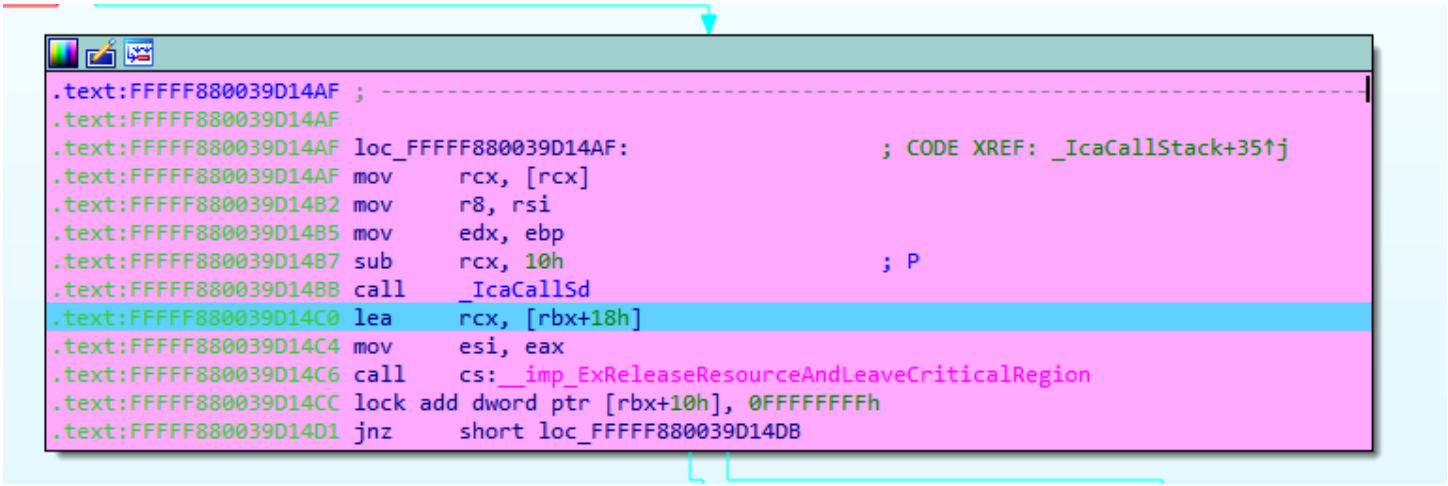
IOCTL 0x380148L

I will put conditional breakpoints in each different IOCTL, to list the functions where each one ends up.

The IOCTLs 0x380047, 0x38003b, 0x3800c7, 0x38244f, 0x38016f, 0x38004b, 0x381403 end in _IcaCallStack

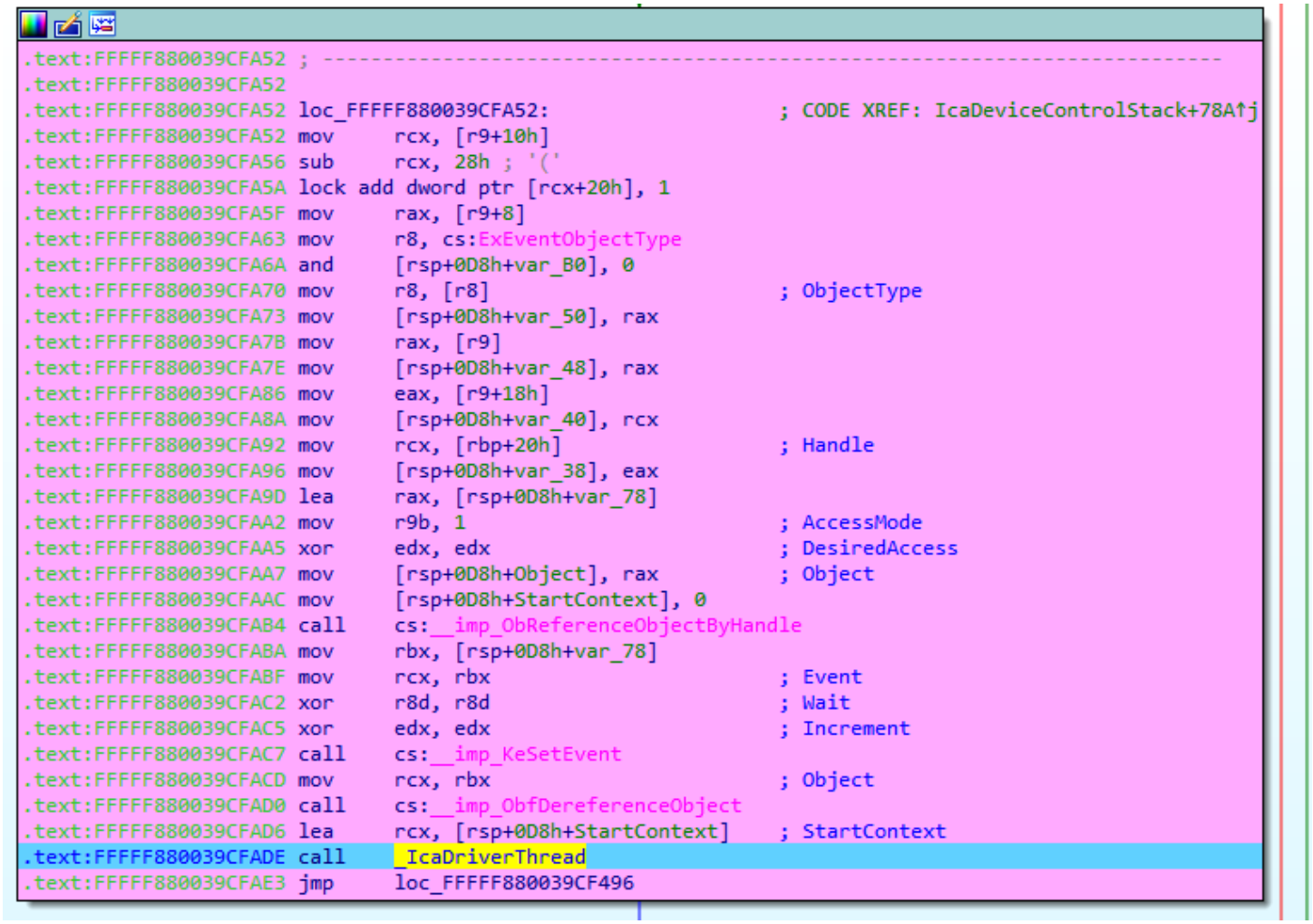


These IOCTLs also reach _IcaCallSd



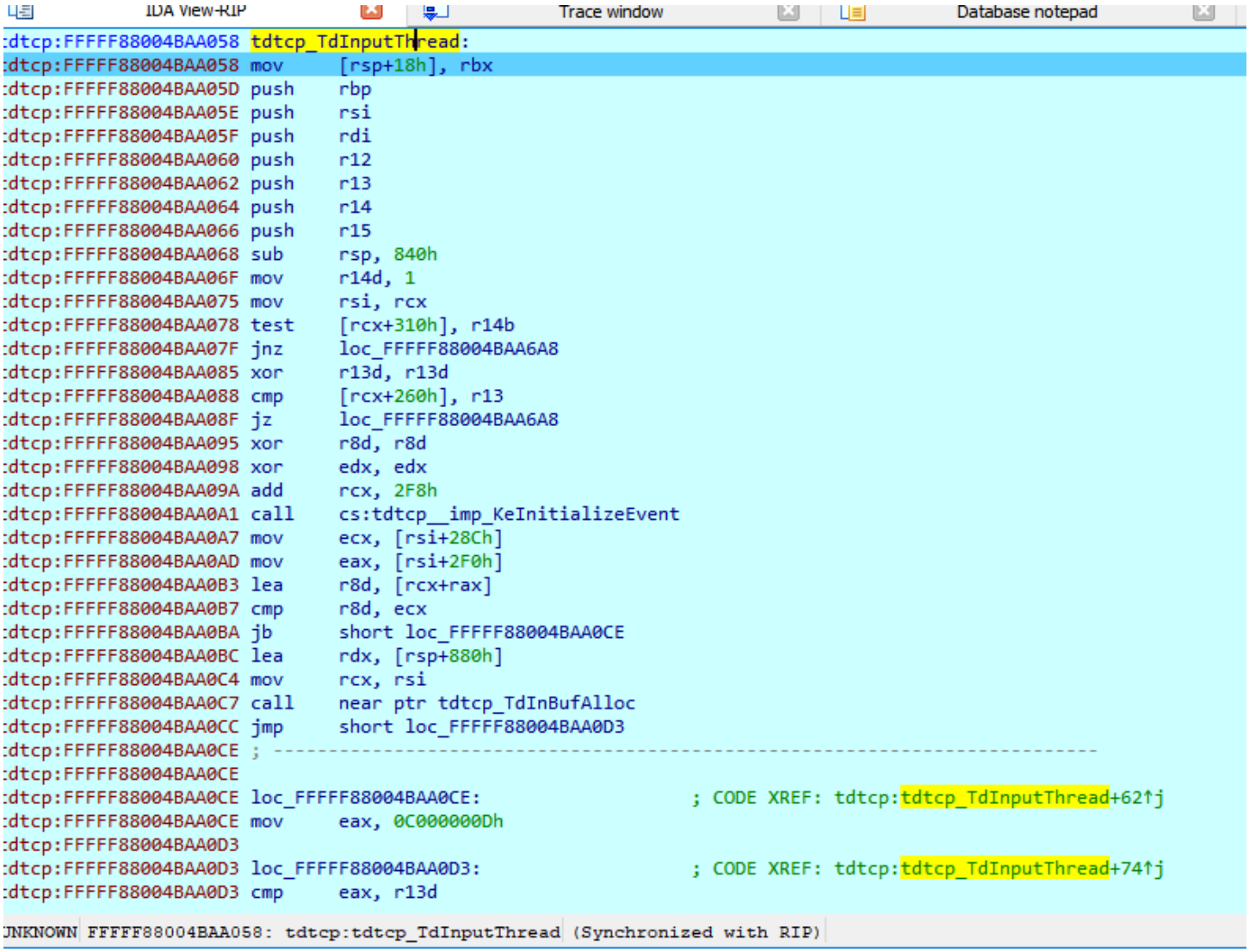
IOCTL **0x380148** does nothing

IOCTL **0x380173 reaches _IcaDriverThread**



```
.text:FFFFF880039CFA52 ; ---------------------------------------------------------------------------
.text:FFFFF880039CFA52
.text:FFFFF880039CFA52 loc_FFFFF880039CFA52:                   ; CODE XREF: IcaDeviceControlStack+78A↑j
.text:FFFFF880039CFA52 mov     rcx, [r9+10h]
.text:FFFFF880039CFA56 sub     rcx, 28h ; '('
.text:FFFFF880039CFA5A lock add dword ptr [rcx+20h], 1
.text:FFFFF880039CFA5F mov     rax, [r9+8]
.text:FFFFF880039CFA63 mov     r8, cs:ExEventObjectType
.text:FFFFF880039CFA6A and     [rsp+0D8h+var_B0], 0
.text:FFFFF880039CFA70 mov     r8, [r8]                        ; ObjectType
.text:FFFFF880039CFA73 mov     [rsp+0D8h+var_50], rax
.text:FFFFF880039CFA7B mov     rax, [r9]
.text:FFFFF880039CFA7E mov     [rsp+0D8h+var_48], rax
.text:FFFFF880039CFA86 mov     eax, [r9+18h]
.text:FFFFF880039CFA8A mov     [rsp+0D8h+var_40], rcx
.text:FFFFF880039CFA92 mov     rcx, [rbp+20h]                  ; Handle
.text:FFFFF880039CFA96 mov     [rsp+0D8h+var_38], eax
.text:FFFFF880039CFA9D lea     rax, [rsp+0D8h+var_78]
.text:FFFFF880039CFAA2 mov     r9b, 1                          ; AccessMode
.text:FFFFF880039CFAA5 xor     edx, edx                        ; DesiredAccess
.text:FFFFF880039CFAA7 mov     [rsp+0D8h+Object], rax          ; Object
.text:FFFFF880039CFAAC mov     [rsp+0D8h+StartContext], 0
.text:FFFFF880039CFAB4 call    cs:__imp_ObReferenceObjectByHandle
.text:FFFFF880039CFABA mov     rbx, [rsp+0D8h+var_78]
.text:FFFFF880039CFABF mov     rcx, rbx                        ; Event
.text:FFFFF880039CFAC2 xor     r8d, r8d                        ; Wait
.text:FFFFF880039CFAC5 xor     edx, edx                        ; Increment
.text:FFFFF880039CFAC7 call    cs:__imp_KeSetEvent
.text:FFFFF880039CFACD mov     rcx, rbx                        ; Object
.text:FFFFF880039CFAD0 call    cs:__imp_ObfDereferenceObject
.text:FFFFF880039CFAD6 lea     rcx, [rsp+0D8h+StartContext]    ; StartContext
.text:FFFFF880039CFADE call    _IcaDriverThread
.text:FFFFF880039CFAE3 jmp     loc_FFFFF880039CF496
```

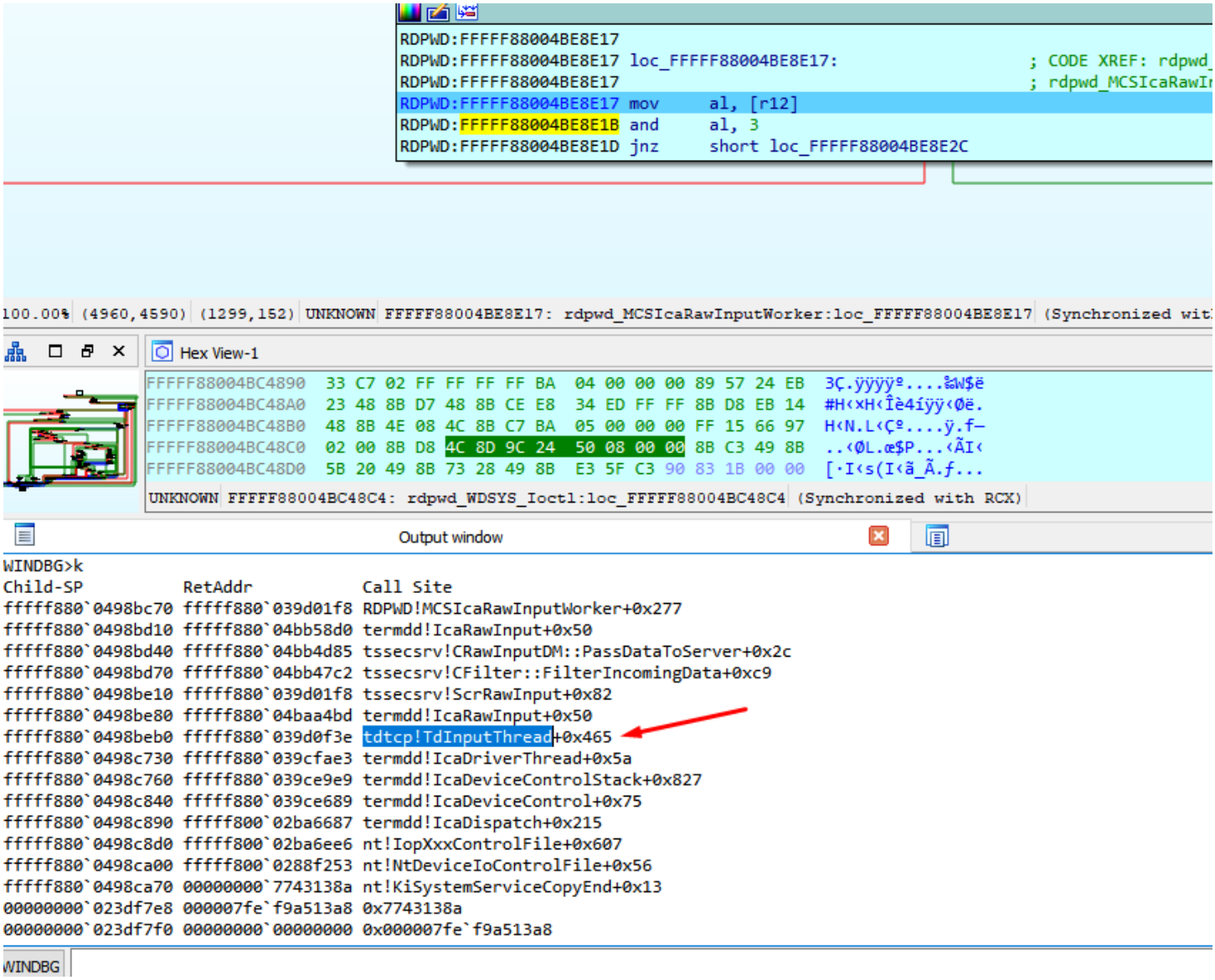
And this last one reaches tdtcp_TdInputThread also.



This function is used to receive the data sended by the user.

STEP 5) Receiving data

If we continue running to the point of data entry breakpoint, we can see in the call stack that it comes from **tdtcp! TdInputThread.**



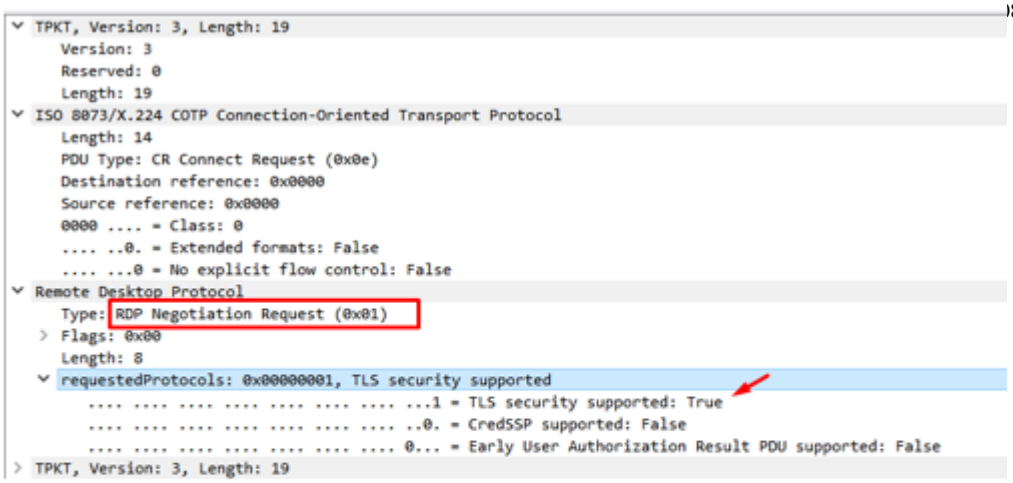
The server is ready now, and waiting for our first send.

We will analyze the packages and next we will return to the reversing.

STEP 6) Analyzing Packets

Negotiate Request package

03 00 00 13 0e e0 00 00 00 00 00 01 00 08 00 01 00 00 00



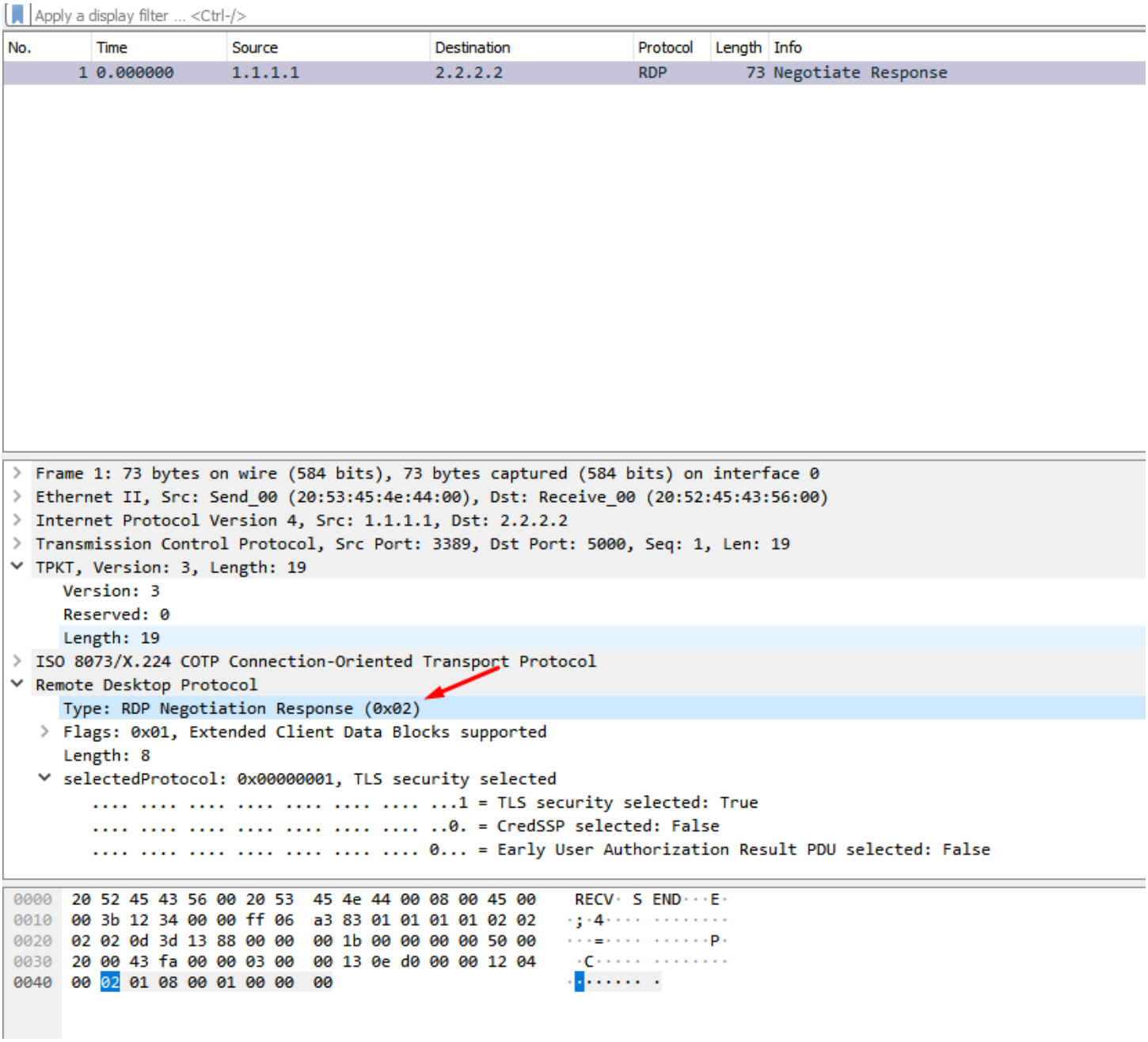

```
Header
 03 -> TPKT: TPKT version = 3
 00 -> TPKT: Reserved = 0
 00 -> TPKT: Packet length - high part
 13 -> TPKT: Packet length - low part
X.224
 0e -> X.224: Length indicator
 e0 -> X.224: CRC connect request
 00 00 -> X.224: Destination reference = 0
 00 00 -> X.224: Source reference = 0
 00 -> X.224: Class and options = 0
PDU
 01 -> RDP_NEG_REQ::Type=0x1 RDP Negotiation Request
 00 -> RDP_NEG_REQ::flags (0)
 08 00 -> RDP_NEG_REQ::length (8 bytes)
 01 00 00 00 -> 0x1 TLS security supported(SSL)
```

Requested Protocol

```
requestedProtocols: 0x00000001, TLS security supported
    .... .... .... .... .... .... .... ...1 = TLS security supported: True
    .... .... .... .... .... .... .... ..0. = CredSSP supported: False
    .... .... .... .... .... .... .... 0... = Early User Authorization Result PDU supported: False
```

Negotiation Response package

The Response package was similar only with **Type=0x2 RDP Negotiation Response**



Connect Initial Package

The package starts with

"\x03\x00\xFF\xFF\x02\xf0\x80" #\xFF\xFF are sizes to be calculated and smashed at the end

Header

03 -> TPKT: TPKT version = 3 00 -> TPKT: Reserved = 0 FF -> TPKT: Packet length - high part FF -> TPKT: Packet length - low part

X.224

02 -> X.224: Length indicator = 2 f0 -> X.224: Type = 0xf0 = Data TPDU 80 -> X.224: EOT

PDU

"7f 65" .. -- BER: Application-Defined Type = APPLICATION 101,

"82 FF FF" .. -- BER: Type Length = will be calculated and smashed at the end in the Dos sample will be 0x1b2

"04 01 01" .. -- Connect-Initial::callingDomainSelector

"04 01 01" .. -- Connect-Initial::calledDomainSelector

"01 01 ff" .. -- Connect-Initial::upwardFlag = TRUE

"30 19" .. -- Connect-Initial::targetParameters (25 bytes)

"02 01 22" .. -- DomainParameters::maxChannelIds = 34

"02 01 02" .. -- DomainParameters::maxUserIds = 2

"02 01 00" .. -- DomainParameters::maxTokenIds = 0

"02 01 01" .. -- DomainParameters::numPriorities = 1

"02 01 00" .. -- DomainParameters::minThroughput = 0

"02 01 01" .. -- DomainParameters::maxHeight = 1

"02 02 ff ff" .. -- DomainParameters::maxMCSPDUsize = 65535

"02 01 02" .. -- DomainParameters::protocolVersion = 2

"30 19" .. -- Connect-Initial::minimumParameters (25 bytes)

"02 01 01" .. -- DomainParameters::maxChannelIds = 1

"02 01 01" .. -- DomainParameters::maxUserIds = 1

"02 01 01" .. -- DomainParameters::maxTokenIds = 1

"02 01 01" .. -- DomainParameters::numPriorities = 1

"02 01 00" .. -- DomainParameters::minThroughput = 0

"02 01 01" .. -- DomainParameters::maxHeight = 1

"02 02 04 20" .. -- DomainParameters::maxMCSPDUsize = 1056

"02 01 02" .. -- DomainParameters::protocolVersion = 2

"30 1c" .. -- Connect-Initial::maximumParameters (28 bytes)

"02 02 ff ff" .. -- DomainParameters::maxChannelIds = 65535

"02 02 fc 17" .. -- DomainParameters::maxUserIds = 64535

"02 02 ff ff" .. -- DomainParameters::maxTokenIds = 65535

"02 01 01" .. -- DomainParameters::numPriorities = 1

"02 01 00" .. -- DomainParameters::minThroughput = 0

"02 01 01" .. -- DomainParameters::maxHeight = 1

"02 02 ff ff" .. -- DomainParameters::maxMCSPDUsize = 65535

"02 01 02" .. -- DomainParameters::protocolVersion = 2

"04 82 FF FF" .. -- Connect-Initial::userData (calculated at the end in the DoS example will be 0x151 bytes)

"00 05" .. -- object length = 5 bytes
"00 14 7c 00 01" .. -- object
"81 48" .. -- ConnectData::connectPDU length = 0x48 bytes
"00 08 00 10 00 01 c0 00 44 75 63 61" .. -- PER encoded (ALIGNED variant of BASIC-PER) GCC Conference Create Request PDU

"81 FF" .. -- UserData::value length (calculated at the end in the DoS example will be 0x13a bytes)

#-------------
"01 c0 ea 00" .. -- TS_UD_HEADER::type = CS_CORE (0xc001), length = 0xea bytes
"04 00 08 00" .. -- TS_UD_CS_CORE::version = 0x0008004
"00 05" .. -- TS_UD_CS_CORE::desktopWidth = 1280
"20 03" .. -- TS_UD_CS_CORE::desktopHeight = 1024
"01 ca" .. -- TS_UD_CS_CORE::colorDepth = RNS_UD_COLOR_8BPP (0xca01)
"03 aa" .. -- TS_UD_CS_CORE::SASSequence
"09 04 00 00" .. -- TS_UD_CS_CORE::keyboardLayout = 0x409 = 1033 = English (US)
"28 0a 00 00" .. -- TS_UD_CS_CORE::clientBuild = 2600

"45 00 4d 00 50 00 2d 00 4c 00 41 00 50 00 2d 00 " ..
"30 00 30 00 31 00 34 00 00 00 00 00 00 00 00 00 " .. -- TS_UD_CS_CORE::clientName = EMP-LAP-0014

"04 00 00 00" .. -- TS_UD_CS_CORE::keyboardType
"00 00 00 00" .. -- TS_UD_CS_CORE::keyboardSubtype
"0c 00 00 00" .. -- TS_UD_CS_CORE::keyboardFunctionKey

"00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 " ..
"00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 " ..
"00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 " ..
"00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 " .. -- TS_UD_CS_CORE::imeFileName = ""

"01 ca" .. -- TS_UD_CS_CORE::postBeta2ColorDepth = RNS_UD_COLOR_8BPP (0xca01)
"01 00" .. -- TS_UD_CS_CORE::clientProductId
"00 00 00 00" .. -- TS_UD_CS_CORE::serialNumber
"18 00" .. -- TS_UD_CS_CORE::highColorDepth = 24 bpp
"07 00" .. -- TS_UD_CS_CORE::supportedColorDepths = 24 bpp

"01 00" .. -- TS_UD_CS_CORE::earlyCapabilityFlags

"00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 " ..
"00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 " ..
"00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 " ..
"00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 " .. -- TS_UD_CS_CORE::clientDigProductId

07 -> TS_UD_CS_CORE::connectionType = 7
00 -> TS_UD_CS_CORE::pad1octet

01 00 00 00 -> TS_UD_CS_CORE::serverSelectedProtocol
#---------------

04 c0 0c 00 -> TS_UD_HEADER::type = CS_CLUSTER (0xc004), length = 12 bytes

"15 00 00 00" .. -- TS_UD_CS_CLUSTER::Flags = 0x15 f (REDIRECTION_SUPPORTED | REDIRECTION_VERSION3)
"00 00 00 00" .. -- TS_UD_CS_CLUSTER::RedirectedSessionID

#------------

"02 c0 0c 00" -- TS_UD_HEADER::type = CS_SECURITY (0xc002), length = 12 bytes
"1b 00 00 00" .. -- TS_UD_CS_SEC::encryptionMethods

"00 00 00 00" .. -- TS_UD_CS_SEC::extEncryptionMethods

"03 c0 38 00" .. -- TS_UD_HEADER::type = CS_NET (0xc003), length = 0x38 bytes

In this package we need to set the user channels, and a MS_T120 channel needs to be included in the list.

Erect Domain Package

```
Header
 03 -> TPKT: TPKT version = 3
 00 -> TPKT: Reserved = 0
 00 -> TPKT: Packet length - high part
 0c -> TPKT: Packet length - low part
X.224
 02 -> X.224: Length indicator = 2
 f0 -> X.224: Type = 0xf0 = Data TPDU
 80 -> X.224: EOT
PDU
```

```
> Transmission Control Protocol, Src Port: 5000, Dst Port: 3389, Seq: 1, Len: 1889
v TPKT, Version: 3, Length: 12
    Version: 3
    Reserved: 0
    Length: 12
v ISO 8073/X.224 COTP Connection-Oriented Transport Protocol
    Length: 2
    PDU Type: DT Data (0x0f)
    [Destination reference: 0x0000]
    .000 0000 = TPDU number: 0x00
    1... .... = Last data unit: Yes
v MULTIPOINT-COMMUNICATION-SERVICE T.125
  v DomainMCSPDU: erectDomainRequest (1)
    v erectDomainRequest
        subHeight: 0
        subInterval: 0
> TPKT, Version: 3, Length: 19
> ISO 8073/X.224 COTP Connection-Oriented Transport Protocol
0030  20 00 db 06 00 00 03 00  00 0c 02 f0 80 04 01 00    ........ ........
0040  01 00 03 00 00 13 0e e0  00 00 00 00 00 01 00 08    ........ ........
0050  00 01 00 00 00 03 00 00  13 0e e0 00 00 00 00 00    ........ ........
```

0x04: type ErectDomainRequest

0x01: subHeight length = 1 byte

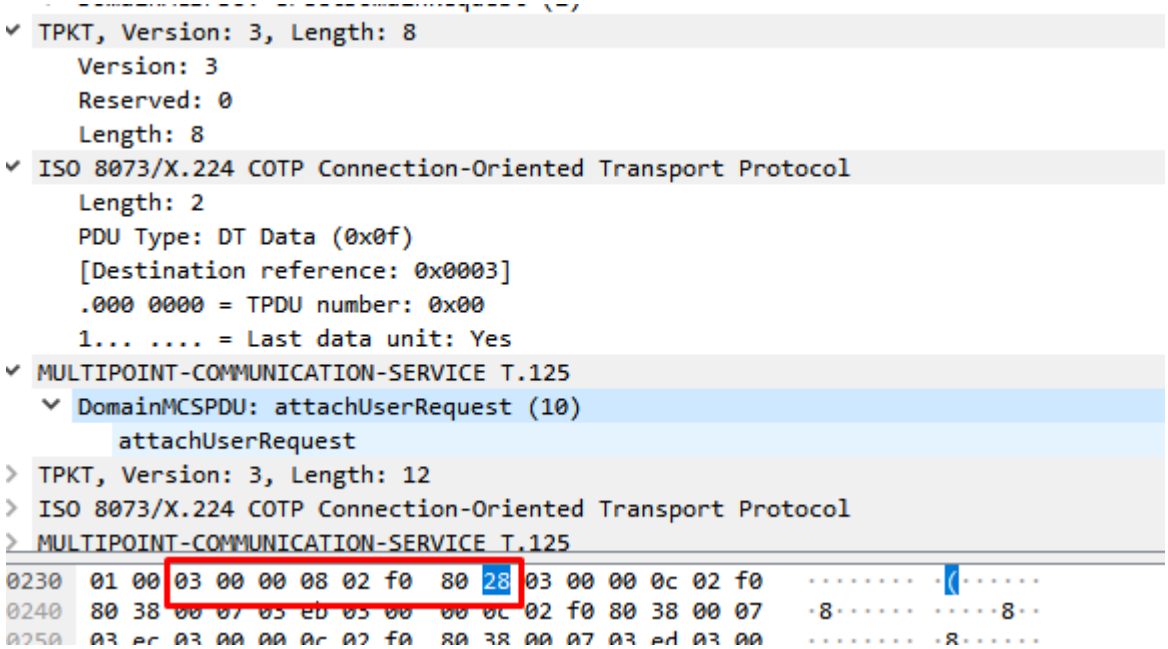
0x00 : subHeight = 0

0x01: subInterval length = 1 byte

0x00: subInterval = 0

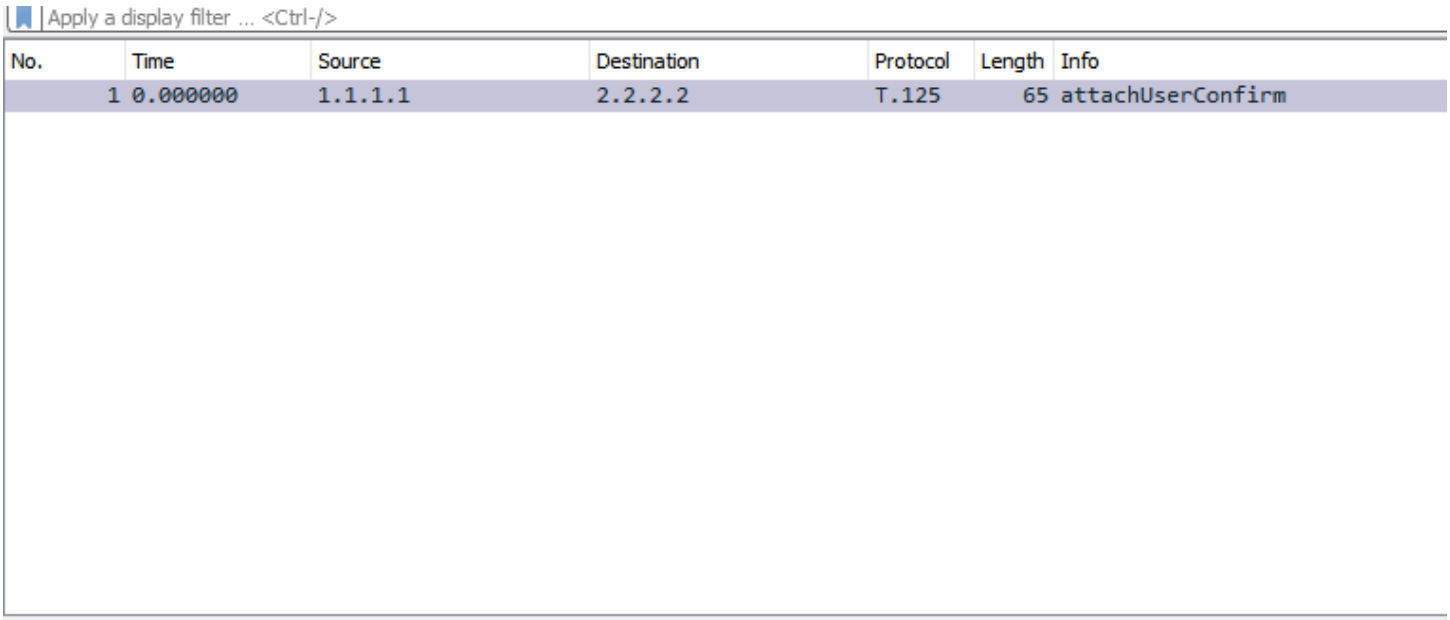
User Attach Packet package

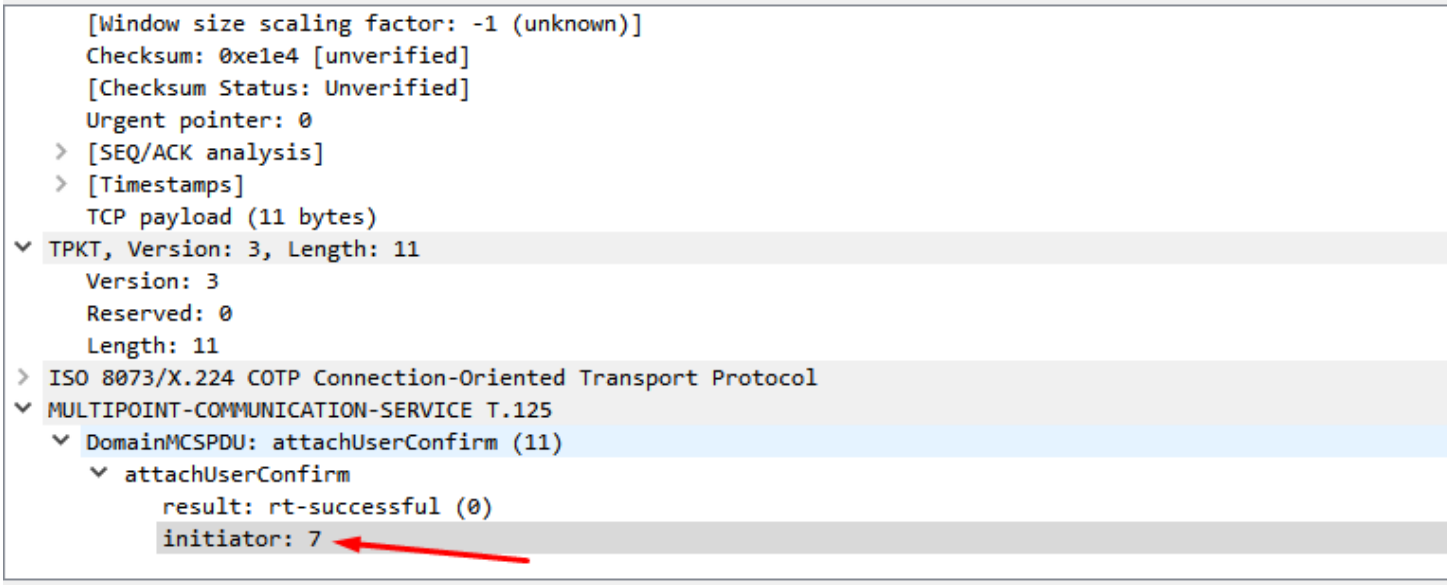
```
00000000 03 00 00 08 02 f0 80 28                        .......(

03 00 00 08 -> TPKT Header (length = 8 bytes)
02 f0 80 -> X.224 Data TPDU

PER encoded (ALIGNED variant of BASIC-PER) PDU contents:
28

0x28:
0 - --\
0 -   |
1 -   | CHOICE: From DomainMCSPDU select attachUserRequest (10)
0 -   | of type AttachUserRequest
1 -   |
0 - --/
0 - padding
0 - padding
```

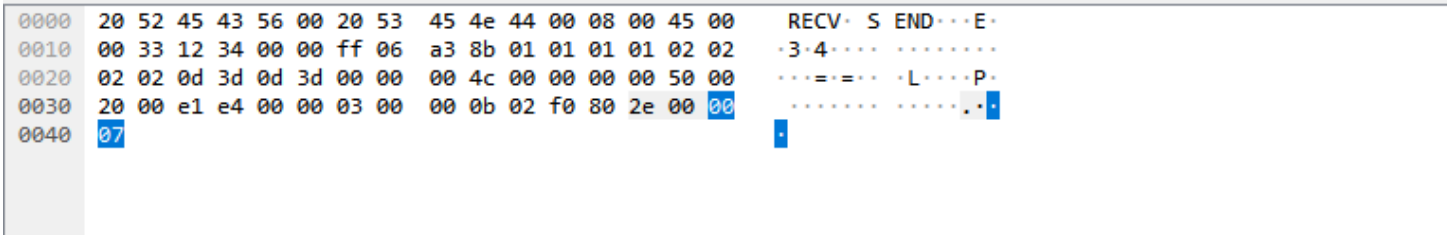


We need to analyze the response.

03 00 00 0b 02 f0 80 2e 00 00 07







The last byte is the initiator, we need to strip from the response to use in the next packet.

Channel Join request package

Building the package

xv1 = (chan_num) / 256

val = (chan_num) % 256

'\x03\x00\x00\x0c\x02\xf0\x80\x38\x00' + initiator + chr(xv1) + chr(val)

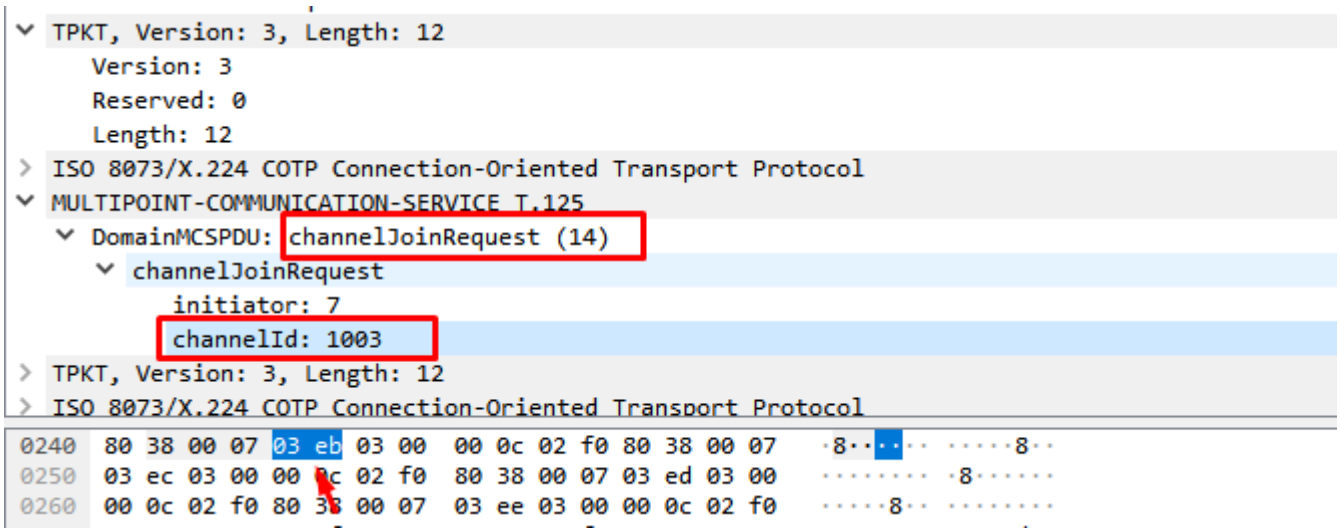
For channel 1003 by example

xv1 = (1003) / 256 = 3

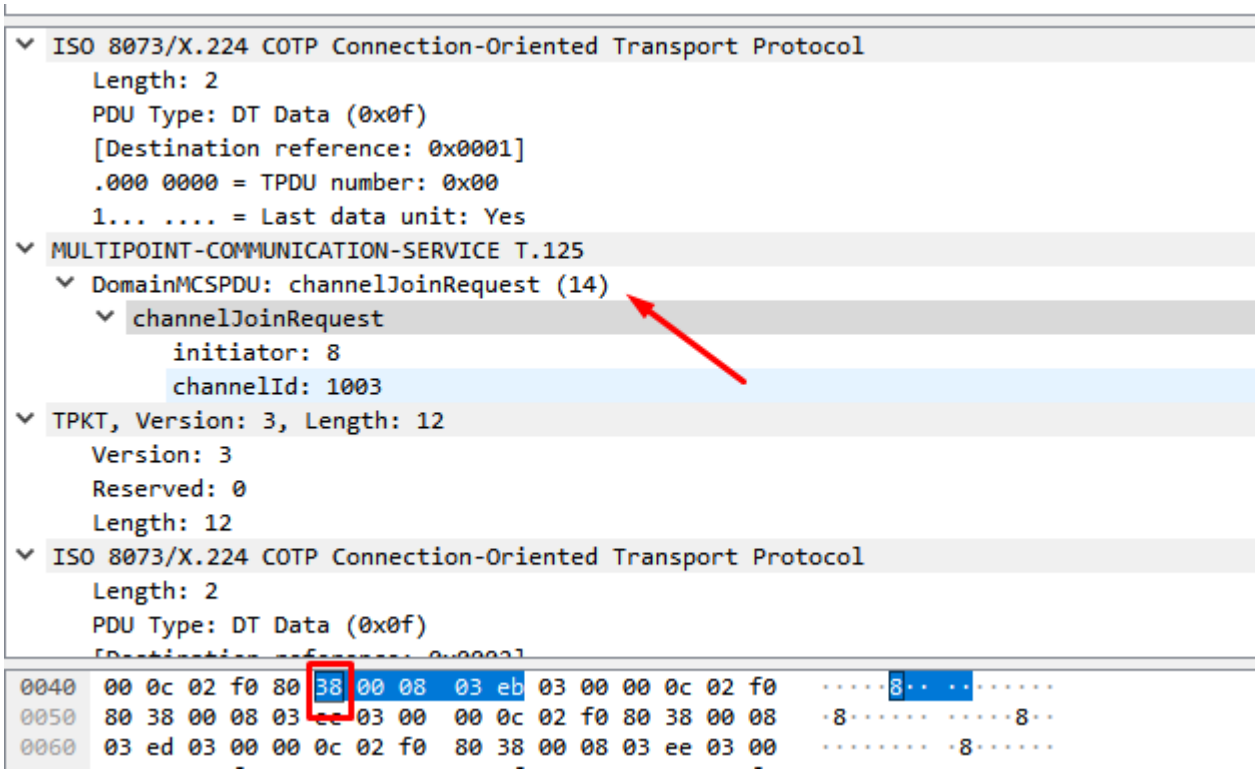
val = (1003) % 256 = 235

'\x03\x00\x00\x0c\x02\xf0\x80\x38\x00' + initiator + chr(3) + chr(235)

```
>>> chr(3)
'\x03'
>>> chr(235)
'\xeb'
```



0x38: channelJoinRequest (14)



All channel join packages are similar, the only thing that changes are the last two bytes that correspond to the channel number.

Channel Join Confirm Response package

The response was

03 00 00 0f 02 f0 80 **3e** 00 00 07 03 eb 03 eb

0x3e:channelJoinConfirm (15)

```
TPKT, Version: 3, Length: 15
    Version: 3
    Reserved: 0
    Length: 15
ISO 8073/X.224 COTP Connection-Oriented Transport Protocol
    Length: 2
    PDU Type: DT Data (0x0f)
    [Destination reference: 0x0000]
    .000 0000 = TPDU number: 0x00
    1... .... = Last data unit: Yes
MULTIPOINT-COMMUNICATION-SERVICE T.125
  DomainMCSPDU: channelJoinConfirm (15)
    channelJoinConfirm
      result: rt-successful (0)
      initiator: 7
      requested: 1003
      channelId: 1003
```

```
0000  20 52 45 43 56 00 20 53  45 4e 44 00 08 00 45 00    RECV· S END···E·
0010  00 37 12 34 00 00 ff 06  a3 87 01 01 01 01 02 02   ·7·4···· ········
0020  02 02 0d 3d 0d 3d 00 00  00 00 00 00 00 00 50 00   ···=·=·· ·····P·
0030  20 00 0c 11 00 00 03 00  00 0f 02 f0 80 3e 00 00    ······ ·····>··
0040  07 03 eb 03 eb                                    ·····
```

result: rt_succesful (0x0)

The packet has the same initiator and channelid values than the request to the same channel.

When all the channels response the Join Request, the next package sended is send Data Request.

```
  DomainMCSPDU: channelJoinRequest (14)
TPKT, Version: 3, Length: 12
ISO 8073/X.224 COTP Connection-Oriented Transport Protocol
MULTIPOINT-COMMUNICATION-SERVICE T.125
  DomainMCSPDU: channelJoinRequest (14)
    channelJoinRequest
      initiator: 7
      channelId: 1007
TPKT, Version: 3, Length: 353
ISO 8073/X.224 COTP Connection-Oriented Transport Protocol
MULTIPOINT-COMMUNICATION-SERVICE T.125
  DomainMCSPDU: sendDataRequest (25)
```

Client Info PDU or Send Data Request Package

```
TPKT, Version: 3, Length: 353
    Version: 3
    Reserved: 0
    Length: 353
ISO 8073/X.224 COTP Connection-Oriented Transport Protocol
    Length: 2
    PDU Type: DT Data (0x0f)
    [Destination reference: 0x0009]
    .000 0000 = TPDU number: 0x00
    1... .... = Last data unit: Yes
MULTIPOINT-COMMUNICATION-SERVICE T.125
  DomainMCSPDU: sendDataRequest (25)
    sendDataRequest
        initiator: 7
        channelId: 1003
        dataPriority: high (1)
```

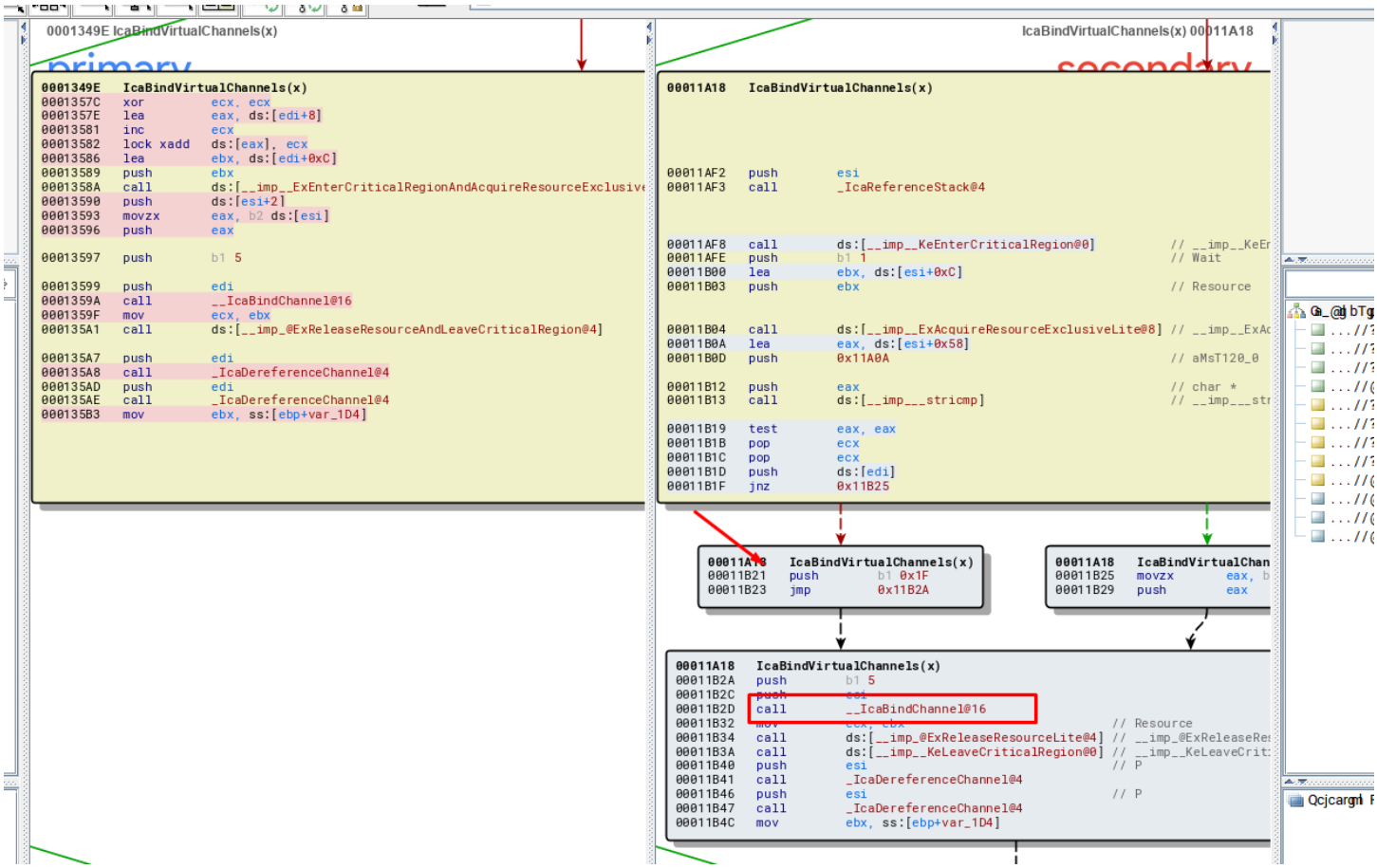
```
0270  80 38 00 07 03 ef 03 00  01 61 02 f0 80 64 00 07   ·8······ ·a···d··
0280  03 eb 70 81 52 40 00 a1  a5 09 04 09 04 bb 47 03   ··p·R@·· ·····G·
0290  00 00 00 0e 00 08 00 00  00 00 00 00 00 41 00 41   ········ ·····A·A
```

The remaining packages are important for the exploitation, so for now we will not show them in this first delivery.

STEP 7) The vulnerability

The program allocate a channel **MS_T120** by default, the user can set different channels in the packages.

This is the diff of the function named IcabindVirtualChannels



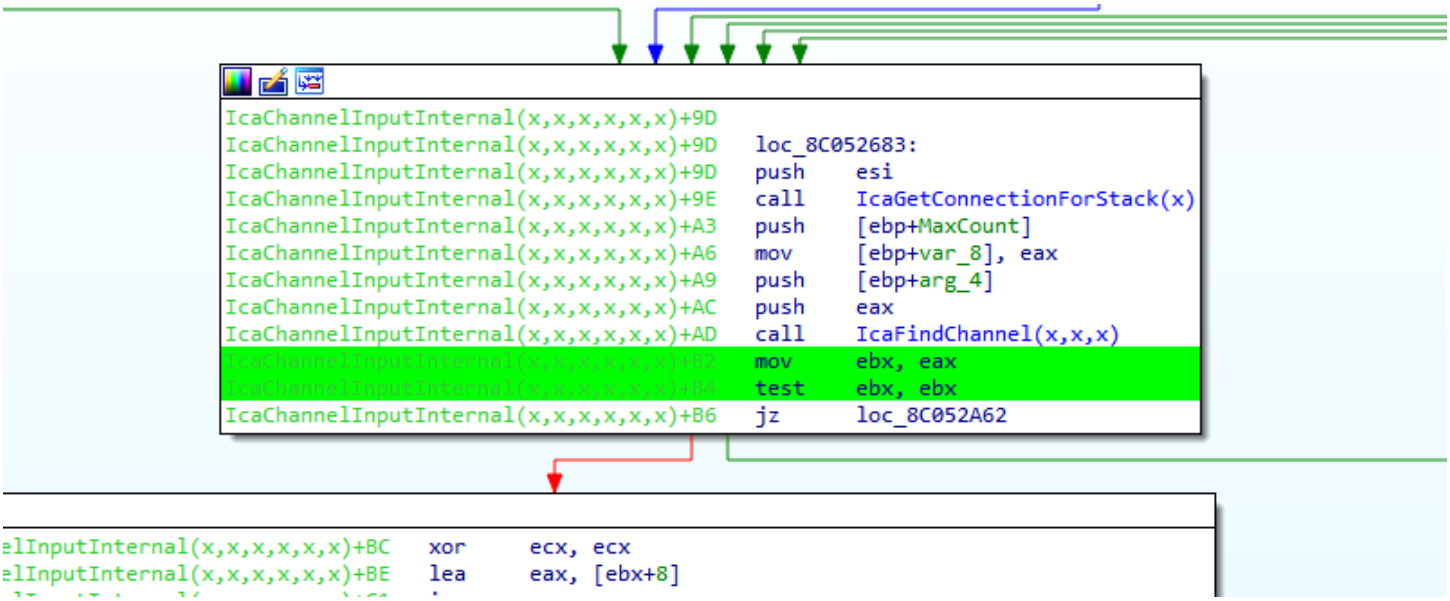
This is the patch for the Windows XP version, which its logic is similar for every vulnerable windows version, when the program compares the string MS_T120 with the name of each channel, the pointer is forced to be stored in a fixed position of the table, forcing to use the value 0x1f to calculate the place to save it .

In the vulnerable version, the pointer is stored using the channel number to calculate the position in the channel table, and we will have two pointers stored in different locations, pointing to the same chunk.

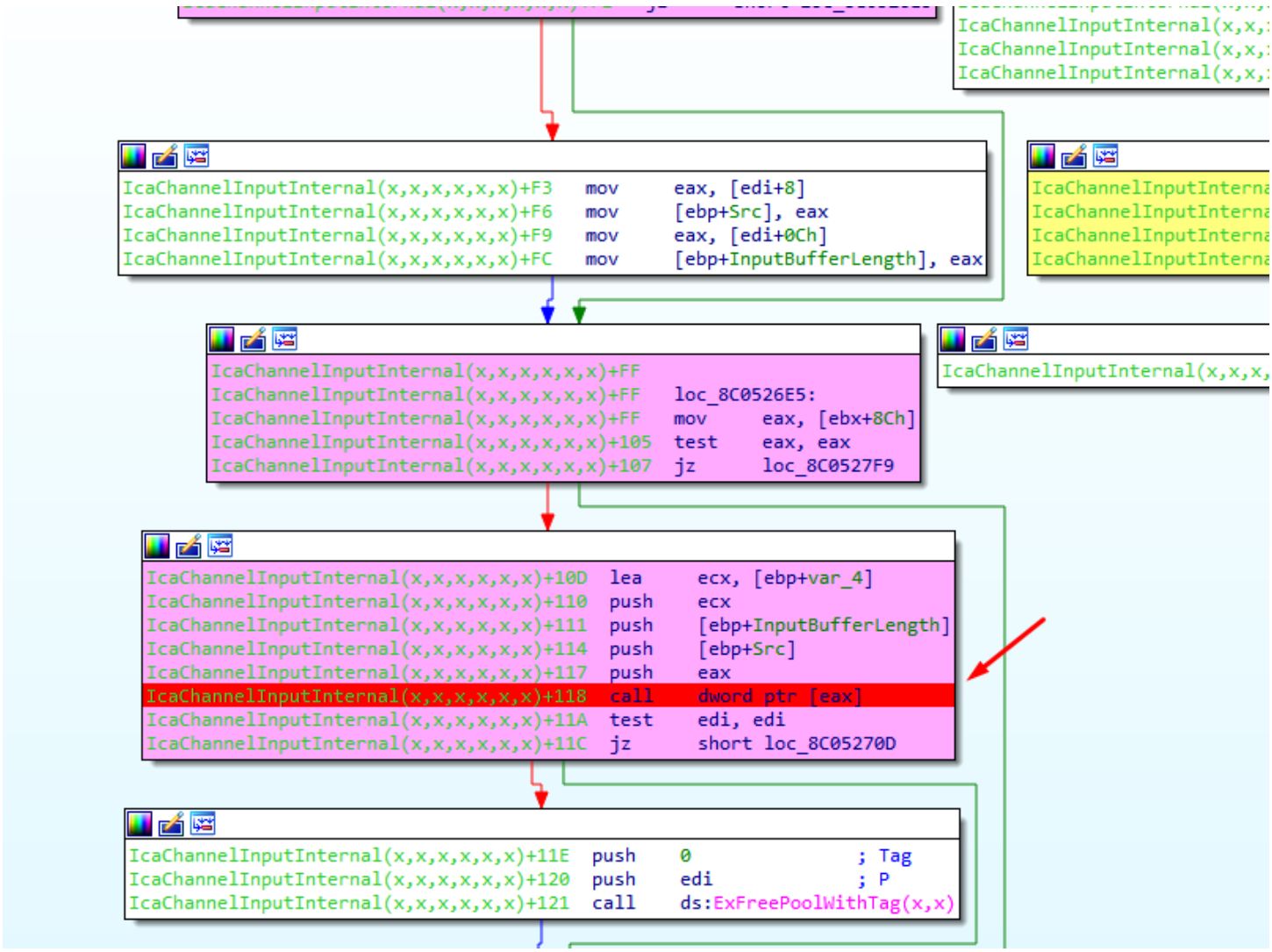
If the user set a channel MS_T120 and send crafted data to that channel, the program will allocate a chunk for that, but will store two different pointers to that chunk, after that the program frees the chunk, but the data of the freed chunk is incorrectly accessed, performing a USE AFTER FREE vulnerability.

The chunk is freed here

Then the chunk is accessed after the free here, EBX will point to the freed chunk.

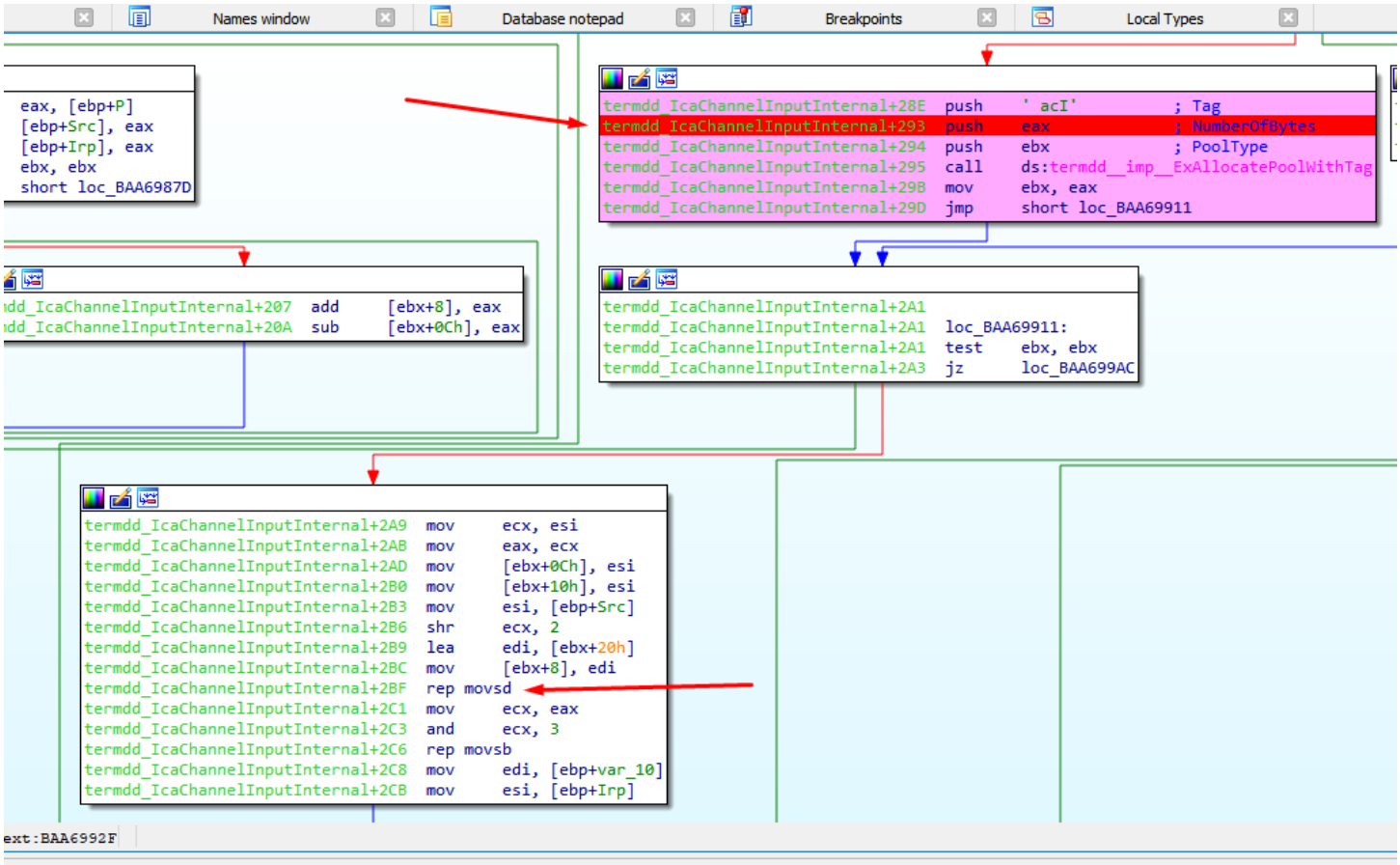


If a perfect pool spray is performed, using the correct chunk size, we can control the execution flow, the value of EAX controlled by us, EBX point to our chunk, EAX =[EBX+0x8c] is controlled by us too.



STEP 8) Pool spray

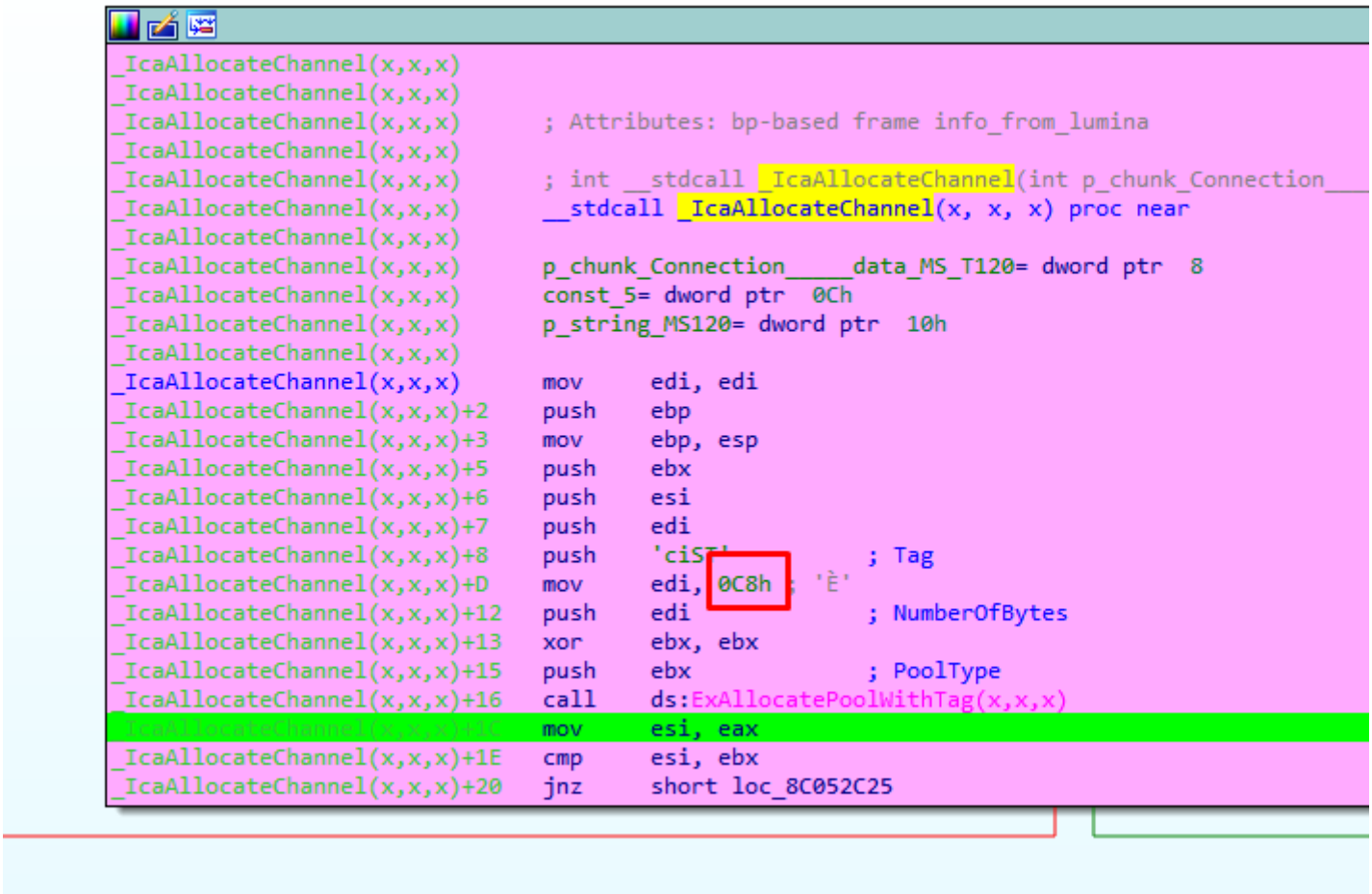
There is a point in the code that let us allocate our data with size controlled, and the same type of pool.



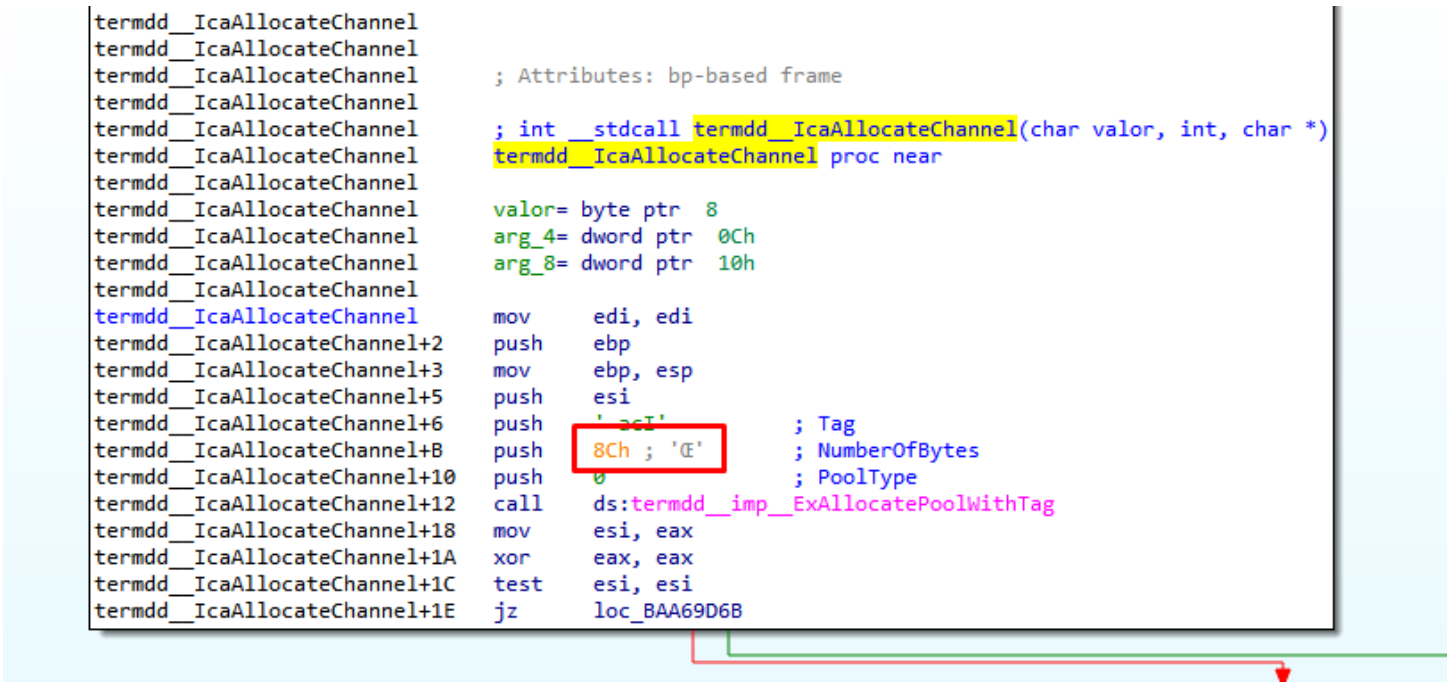
We can send bunch of crafted packages to reach this point, if this packages have the right size can fill the freed chunk, with our data.

In order to get the right size is necessary look at the function IcaAllocateChannel.

In Windows 7 32 bits, the size of each chunk of the pool spray should be 0xc8.



For Windows XP 32 bits that size should be 0x8c.



This pool spray remain in this loop allocating with the right size, and we can fill the freed chunk with our own data to control the code execution in the CALL (IcaChannelInputInternal + 0x118)

Be happy

Ricardo Narvaja

(//www.addthis.com/bookmark.php?v=300) (//www.addthis.com/bookmark.php?v=300)
(//www.addthis.com/bookmark.php?v=300) (//www.addthis.com/bookmark.php?v=300)