Analyzing CVE-2018-8453 : An interesting tale of UAF and Double Free in Windows Kernel

Himanshu Khokhar

Shivam Trivedi

#whoweare

- Vulnerability Researchers
- Occasional Speaker at local infosec communities
- Interested in:
  - ➢ Exploit Development
  - ➢ Reverse Engineering
  - ➢ Malware Analysis
  - ➢ Anime

- Blog: pwnrip.com
- Twitter: @shivamtrivedi18
- Twitter: @pwnrip

Agenda

- Motivation
- Introduction to Kernel Exploitation
- Patch Diffing
- Analysing the patch
- Overview of the bug
- Triggering the bug
- Forcing Double Free
- Obtaining a R/W Primitive
- Final Exploit
- Demo

Motivation

Abuse of CVE-2018-8453

Initially used by an APT to target highly specific people in Middle East.

Now, reports have told that Sodinokibi Ransomware has also leveraged this exploit.

Intro to Kernel Exploitation

Basically exploitation of vulnerabilities residing in kernel-land.

As more and more mitigations are added, it is getting harder by the day.

In this talk, we will be discussing a Use-After-Free vulnerability in Windows kernel, affecting Windows 7 to Windows 10.

Patch Diffing

“Patch diffing is a common technique of comparing two binary builds of the same code – a known-vulnerable one and one containing a security fix.” - Mateusz “j00ru” Jurczyk

Plethora of tools available for the task, both paid as well as free ones.

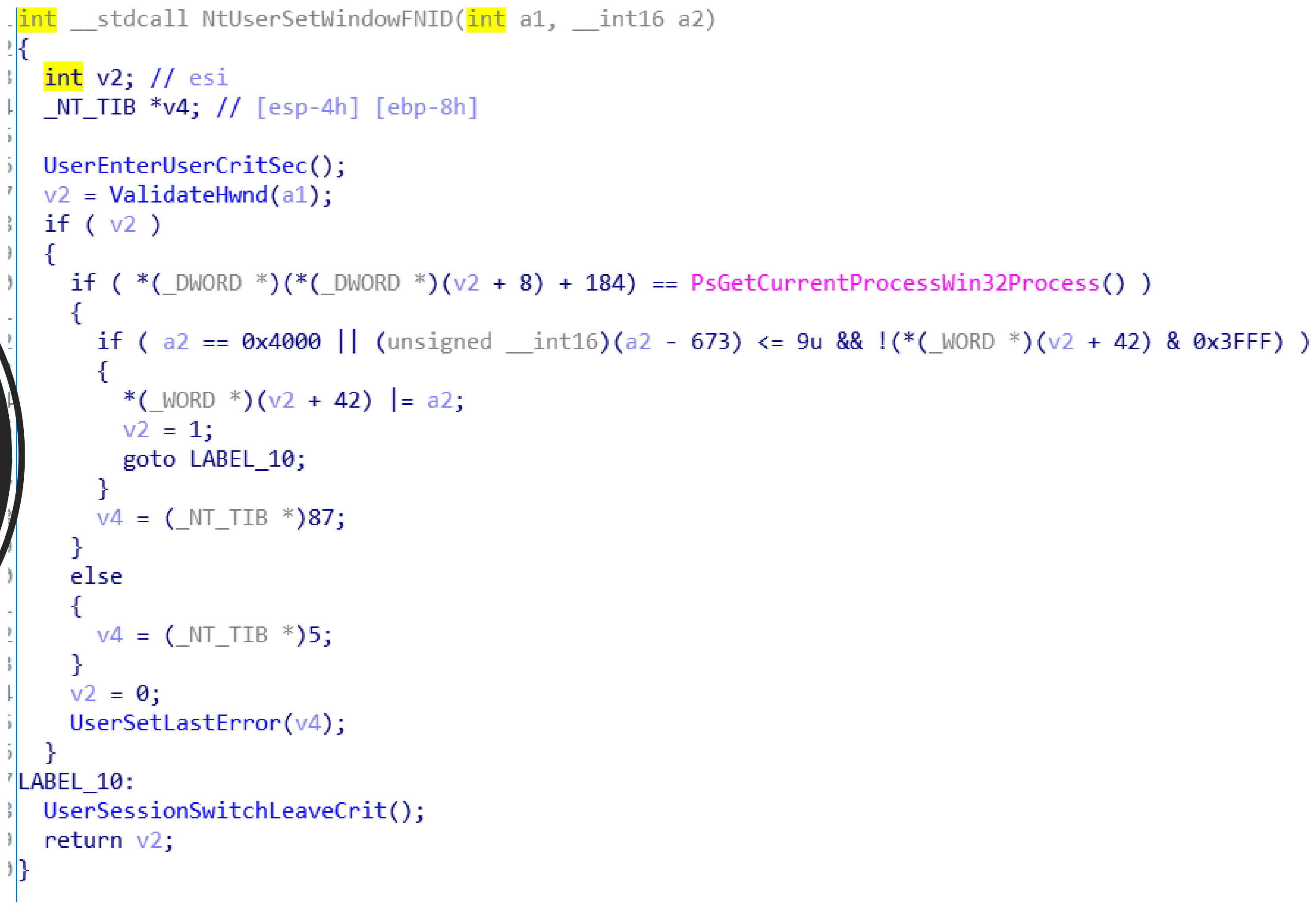
Paid Tools: IDA Pro with Binary Diffing plugins, such as BinDiff or Diaphora or DarumDrim.

Usually the tools/plugins that use IDA are much better as they simply make use of IDA’s decompiler/disassembler and leverage that.

Diffing Patch for CVE-2018-8453

Demo

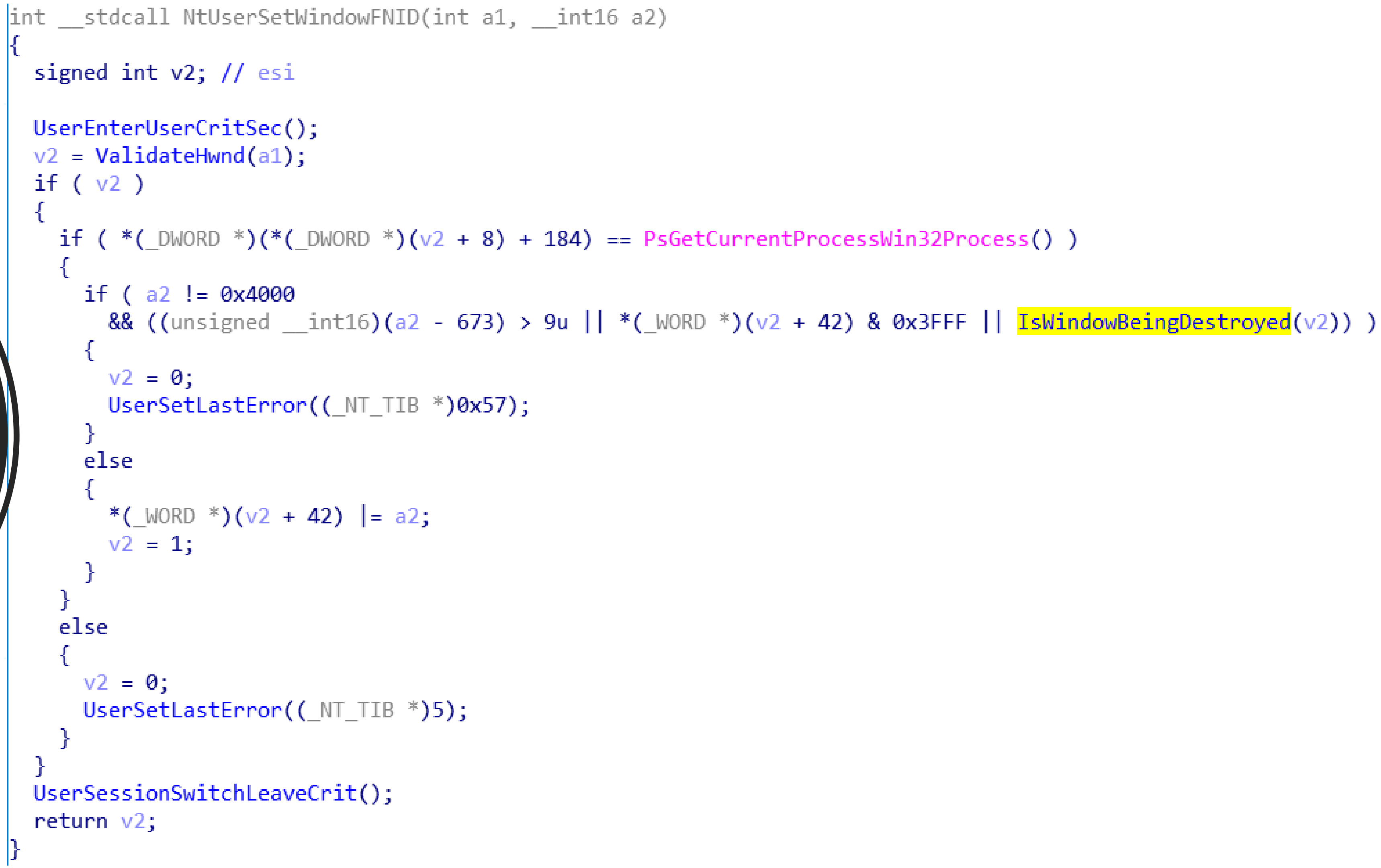
Unpatched



```
int __stdcall NtUserSetWindowFNID(int a1, __int16 a2)
{
  int v2; // esi
  _NT_TIB *v4; // [esp-4h] [ebp-8h]

  UserEnterUserCritSec();
  v2 = ValidateHwnd(a1);
  if ( v2 )
  {
    if ( *(_DWORD *)(*(_DWORD *)(v2 + 8) + 184) == PsGetCurrentProcessWin32Process() )
    {
      if ( a2 == 0x4000 || (unsigned __int16)(a2 - 673) <= 9u && !(*(_WORD *)(v2 + 42) & 0x3FFF) )
      {
        *(_WORD *)(v2 + 42) |= a2;
        v2 = 1;
        goto LABEL_10;
      }
      v4 = (_NT_TIB *)87;
    }
    else
    {
      v4 = (_NT_TIB *)5;
    }
    v2 = 0;
    UserSetLastError(v4);
  }
LABEL_10:
  UserSessionSwitchLeaveCrit();
  return v2;
}
```

Patched



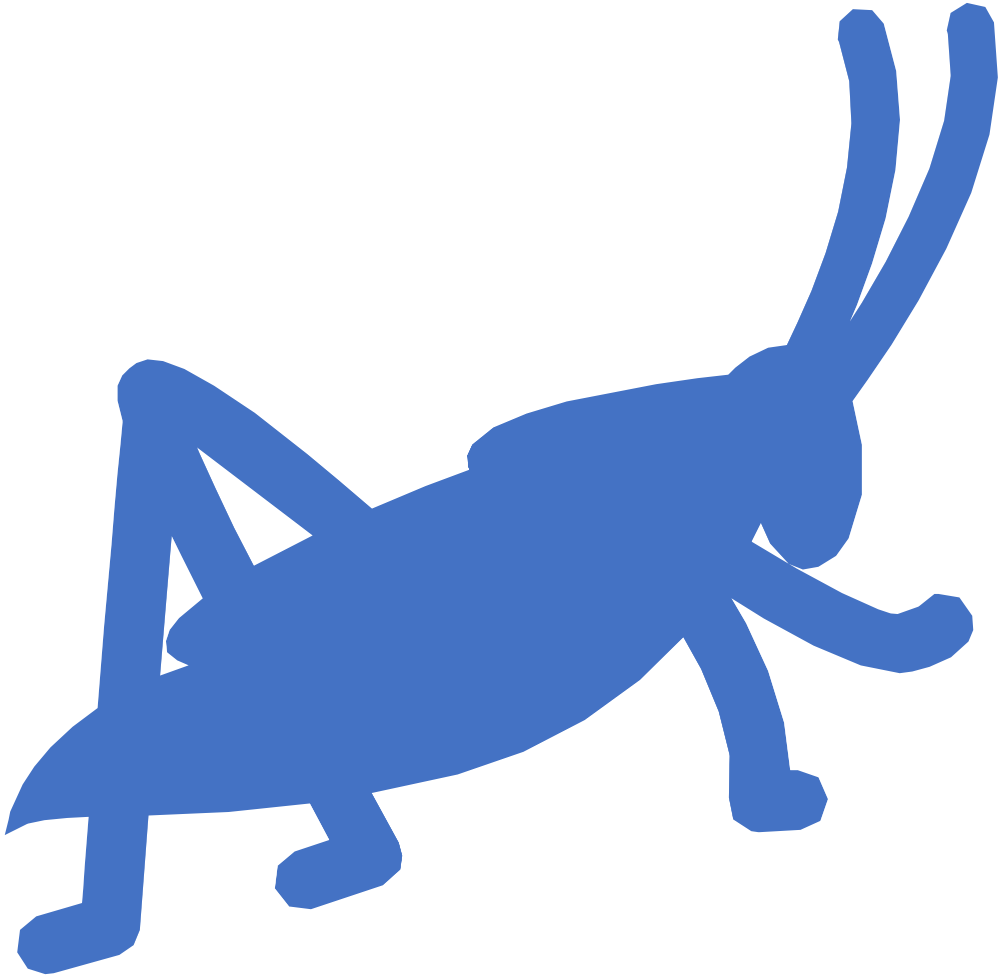
```
int __stdcall NtUserSetWindowFNID(int a1, __int16 a2)
{
  signed int v2; // esi

  UserEnterUserCritSec();
  v2 = ValidateHwnd(a1);
  if ( v2 )
  {
    if ( *(_DWORD *)(*(_DWORD *)(v2 + 8) + 184) == PsGetCurrentProcessWin32Process() )
    {
      if ( a2 != 0x4000
        && ((unsigned __int16)(a2 - 673) > 9u || *(_WORD *)(v2 + 42) & 0x3FFF || IsWindowBeingDestroyed(v2)) )
      {
        v2 = 0;
        UserSetLastError((_NT_TIB *)0x57);
      }
      else
      {
        *(_WORD *)(v2 + 42) |= a2;
        v2 = 1;
      }
    }
    else
    {
      v2 = 0;
      UserSetLastError((_NT_TIB *)5);
    }
  }
  UserSessionSwitchLeaveCrit();
  return v2;
}
```

Overview of the Bug

- As reported by Kaspersky, it is a *Use-After-Free* inside **win32kfull!xxxDestroyWindow**
- Windows does not properly check the FNID to decide whether a window is free or not.
- It can be made to re-use an already free window by setting FNID and forcing the UAF.
- The exploitation, however, requires converting this UAF into a Double-Free and exploiting that to obtain a R/W primitive.

Triggering the bug



Approach for triggering the bug

- Hook Kernel Callback Table for user-mode callback functions – *fnDWORD, fnNCDESTROY* and *fnINLPCREATESTRUCT*.
- Initialize a *SysShadow* window.
- Call *DestroyWindow* on the main window.
- In *fnNCDESTROY*, change the FNID of parent window by using *NtUserSetWindowFNID* syscall.

Hooking the Kernel Callback Table

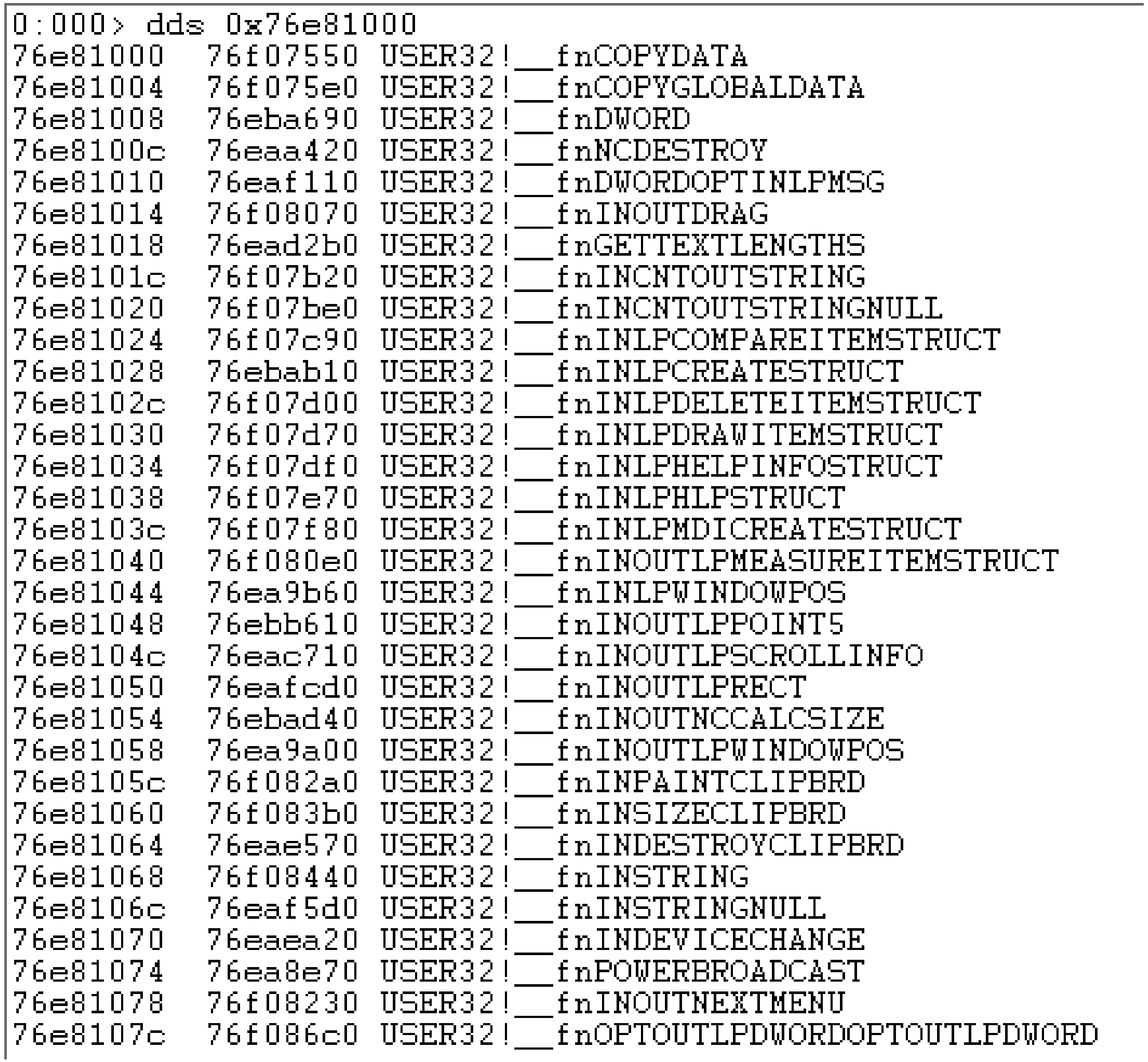
- Allows win32k to make Callback to the user mode.
- We can hook the functions define in *nt!KeUserModeCallback*.
- Need to trigger and exploit the vulnerability successfully.
- Here we can hook 3 Callback functions:

➢ fnDWORD

➢ fnINLPCREATESTRUCT

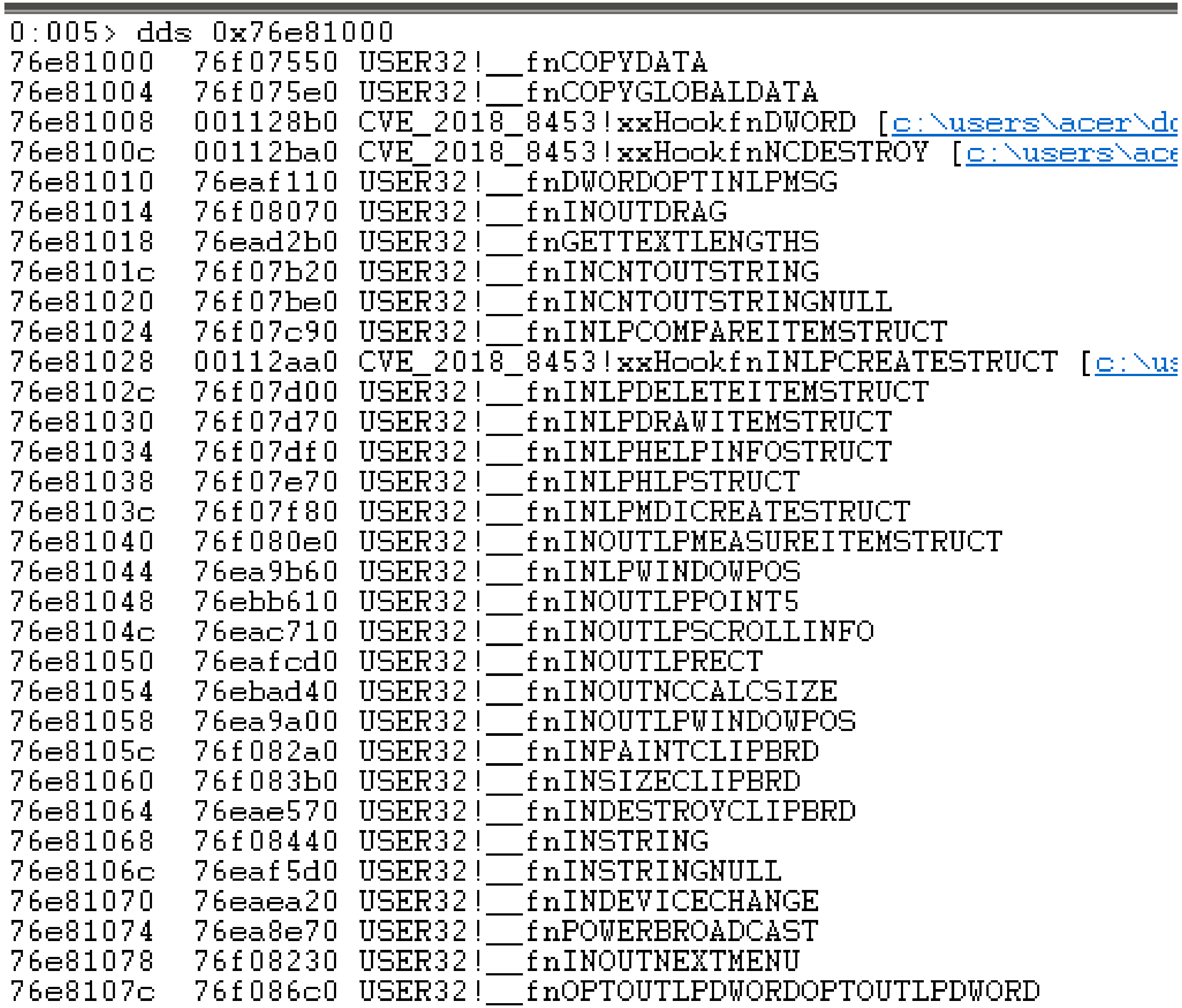
➢ fnNCDESTROY

Kernel Callback Table – Before Hooking



```
0:000> dds 0x76e81000
76e81000  76f07550 USER32!__fnCOPYDATA
76e81004  76f075e0 USER32!__fnCOPYGLOBALDATA
76e81008  76eba690 USER32!__fnDWORD
76e8100c  76eaa420 USER32!__fnNCDESTROY
76e81010  76eaf110 USER32!__fnDWORDOPTINLPMSG
76e81014  76f08070 USER32!__fnINOUTDRAG
76e81018  76ead2b0 USER32!__fnGETTEXTLENGTHS
76e8101c  76f07b20 USER32!__fnINCNTOUTSTRING
76e81020  76f07be0 USER32!__fnINCNTOUTSTRINGNULL
76e81024  76f07c90 USER32!__fnINLPCOMPAREITEMSTRUCT
76e81028  76ebab10 USER32!__fnINLPCREATESTRUCT
76e8102c  76f07d00 USER32!__fnINLPDELETEITEMSTRUCT
76e81030  76f07d70 USER32!__fnINLPDRAWITEMSTRUCT
76e81034  76f07df0 USER32!__fnINLPHELPINFOSTRUCT
76e81038  76f07e70 USER32!__fnINLPHLPSTRUCT
76e8103c  76f07f80 USER32!__fnINLPMDICREATESTRUCT
76e81040  76f080e0 USER32!__fnINOUTLPMEASUREITEMSTRUCT
76e81044  76ea9b60 USER32!__fnINLPWINDOWPOS
76e81048  76ebb610 USER32!__fnINOUTLPPOINT5
76e8104c  76eac710 USER32!__fnINOUTLPSCROLLINFO
76e81050  76eafcd0 USER32!__fnINOUTLPRECT
76e81054  76ebad40 USER32!__fnINOUTNCCALCSIZE
76e81058  76ea9a00 USER32!__fnINOUTLPWINDOWPOS
76e8105c  76f082a0 USER32!__fnINPAINTCLIPBRD
76e81060  76f083b0 USER32!__fnINSIZECLIPBRD
76e81064  76eae570 USER32!__fnINDESTROYCLIPBRD
76e81068  76f08440 USER32!__fnINSTRING
76e8106c  76eaf5d0 USER32!__fnINSTRINGNULL
76e81070  76eaea20 USER32!__fnINDEVICECHANGE
76e81074  76ea8e70 USER32!__fnPOWERBROADCAST
76e81078  76f08230 USER32!__fnINOUTNEXTMENU
76e8107c  76f086c0 USER32!__fnOPTOUTLPDWORDOPTOUTLPDWORD
```

Kernel Callback Table – After Hooking



```
0:005> dds 0x76e81000
76e81000  76f07550 USER32!__fnCOPYDATA
76e81004  76f075e0 USER32!__fnCOPYGLOBALDATA
76e81008  001128b0 CVE_2018_8453!xxHookfnDWORD [c:\users\acer\d
76e8100c  00112ba0 CVE_2018_8453!xxHookfnNCDESTROY [c:\users\ace
76e81010  76eaf110 USER32!__fnDWORDOPTINLPMSG
76e81014  76f08070 USER32!__fnINOUTDRAG
76e81018  76ead2b0 USER32!__fnGETTEXTLENGTHS
76e8101c  76f07b20 USER32!__fnINCNTOUTSTRING
76e81020  76f07be0 USER32!__fnINCNTOUTSTRINGNULL
76e81024  76f07c90 USER32!__fnINLPCOMPAREITEMSTRUCT
76e81028  00112aa0 CVE_2018_8453!xxHookfnINLPCREATESTRUCT [c:\us
76e8102c  76f07d00 USER32!__fnINLPDELETEITEMSTRUCT
76e81030  76f07d70 USER32!__fnINLPDRAWITEMSTRUCT
76e81034  76f07df0 USER32!__fnINLPHELPINFOSTRUCT
76e81038  76f07e70 USER32!__fnINLPHLPSTRUCT
76e8103c  76f07f80 USER32!__fnINLPMDICREATESTRUCT
76e81040  76f080e0 USER32!__fnINOUTLPMEASUREITEMSTRUCT
76e81044  76ea9b60 USER32!__fnINLPWINDOWPOS
76e81048  76ebb610 USER32!__fnINOUTLPPOINT5
76e8104c  76eac710 USER32!__fnINOUTLPSCROLLINFO
76e81050  76eafcd0 USER32!__fnINOUTLPRECT
76e81054  76ebad40 USER32!__fnINOUTNCCALCSIZE
76e81058  76ea9a00 USER32!__fnINOUTLPWINDOWPOS
76e8105c  76f082a0 USER32!__fnINPAINTCLIPBRD
76e81060  76f083b0 USER32!__fnINSIZECLIPBRD
76e81064  76eae570 USER32!__fnINDESTROYCLIPBRD
76e81068  76f08440 USER32!__fnINSTRING
76e8106c  76eaf5d0 USER32!__fnINSTRINGNULL
76e81070  76eaea20 USER32!__fnINDEVICECHANGE
76e81074  76ea8e70 USER32!__fnPOWERBROADCAST
76e81078  76f08230 USER32!__fnINOUTNEXTMENU
76e8107c  76f086c0 USER32!__fnOPTOUTLPDWORDOPTOUTLPDWORD
```

Creating a Window

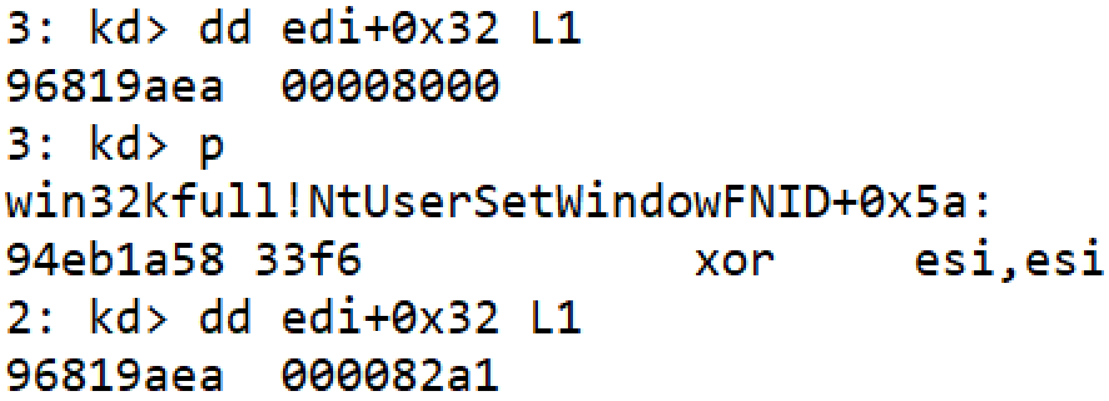
- Create a window using *CreateWindowEx*.
- Add the CS_DROPSHADOW for initialize the *SysShadow* class.
- Add a scroll bar to the window that is being created.
- Send WM_LBUTTONDOWN message to the ScrollBar.

fnDWORD Hook Execution

- Here we compare the class to get the *ScrollBar* class.
- If found then call the *DestroyWindow* function to destroy the main window.
- DestroyWindow calls the fnNCDESTROY hook.
- Here, compare the classname with the *SysShadow* in fnNCDESTROY hook.
- At this point we set the FNID of the Freed Window to FNID of the button.
- This is the point where the vulnerability gets triggered.

FNID Changes from FNID_FREED to FNID_BUTTON

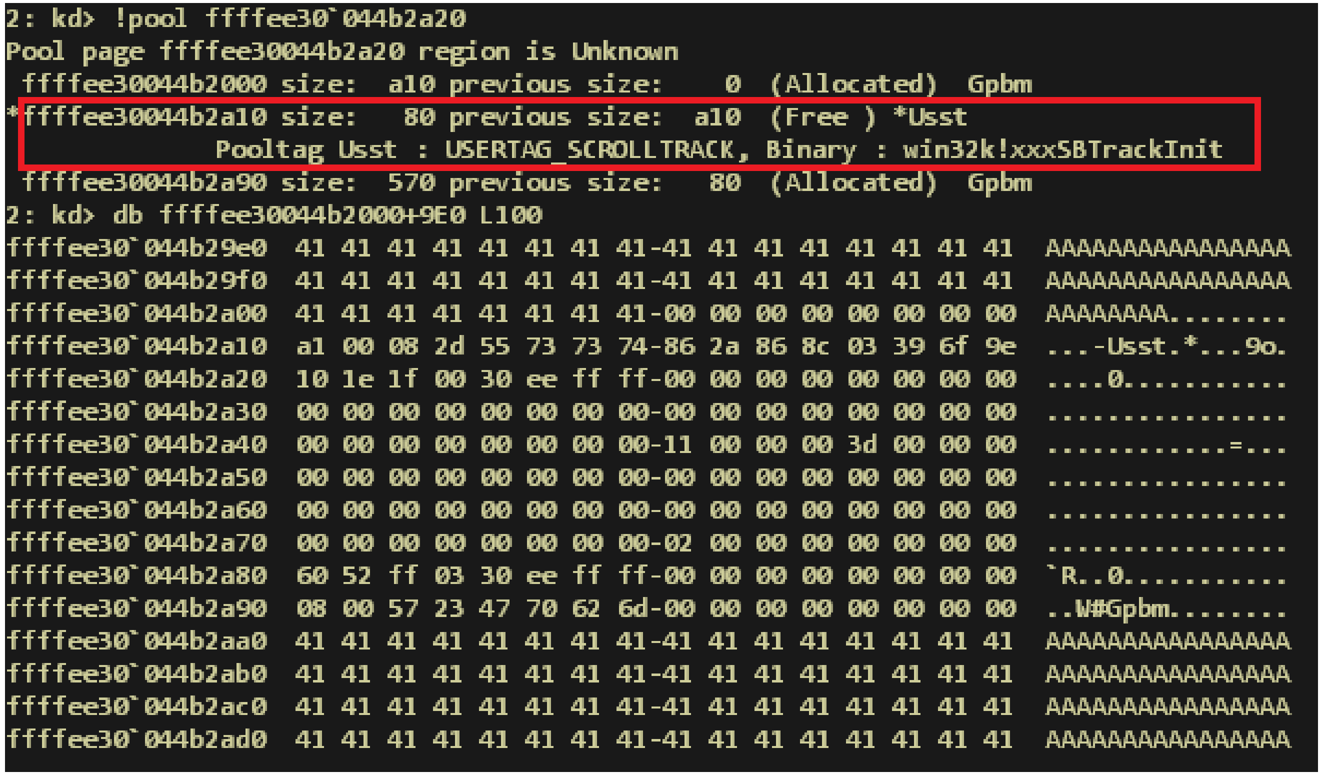
- FNID of a window that has been freed – 0x8000 (FNID_FREED).
- FNID of a button – 0x02A1 (FNID_BUTTON).
- We can observe here that the FNID is getting changed.



```
3: kd> dd edi+0x32 L1
96819aea  00008000
3: kd> p
win32kfull!NtUserSetWindowFNID+0x5a:
94eb1a58 33f6            xor     esi,esi
2: kd> dd edi+0x32 L1
96819aea  000082a1
```

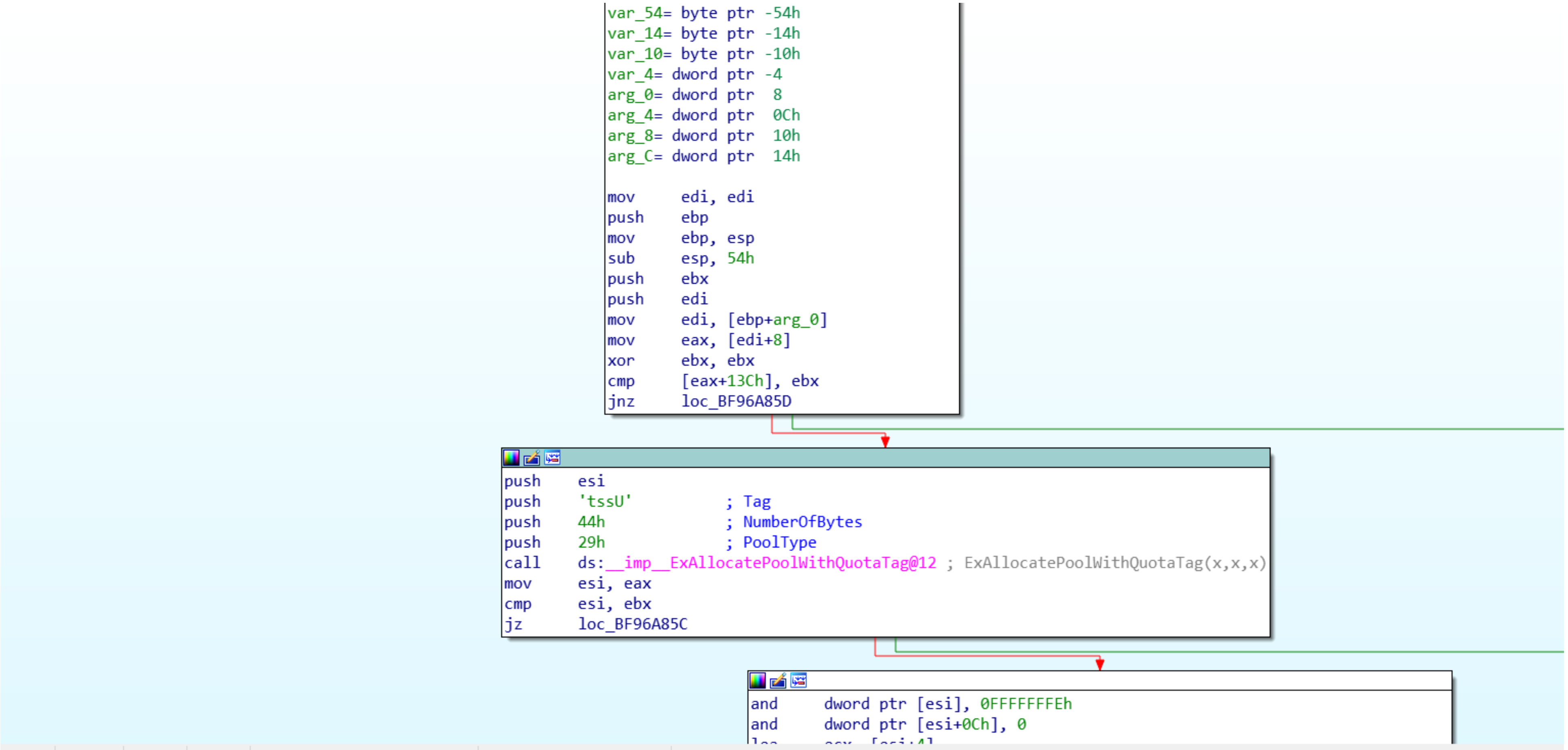
Kaspersky comes to the Rescue!!

- USERTAG_SCROLLTRACK is freed, which indicates usage of a scrollbar.
- *xxxSbtrackInit* initializes the object. This function is called when the scrollbar is being moved.
- When we send the WM_LBUTTONDOWN message to the scrollbar it calls the fnDWORD hook and initialize the USST object.



```
2: kd> !pool ffffee30`044b2a20
Pool page ffffee30044b2a20 region is Unknown
 ffffee30044b2000 size:  a10 previous size:    0  (Allocated)  Gpbm
*ffffee30044b2a10 size:   80 previous size:  a10  (Free ) *Usst
            Pooltag Usst : USERTAG_SCROLLTRACK, Binary : win32k!xxxSBTrackInit
 ffffee30044b2a90 size:  570 previous size:   80  (Allocated)  Gpbm
2: kd> db ffffee30044b2000+9E0 L100
ffffee30`044b29e0  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
ffffee30`044b29f0  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
ffffee30`044b2a00  41 41 41 41 41 41 41 41-00 00 00 00 00 00 00 00  AAAAAAAA........
ffffee30`044b2a10  a1 00 08 2d 55 73 73 74-86 2a 86 8c 03 39 6f 9e  ...-Usst.*...9o.
ffffee30`044b2a20  10 1e 1f 00 30 ee ff ff-00 00 00 00 00 00 00 00  ....0...........
ffffee30`044b2a30  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
ffffee30`044b2a40  00 00 00 00 00 00 00 00-11 00 00 00 3d 00 00 00  ............=...
ffffee30`044b2a50  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
ffffee30`044b2a60  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
ffffee30`044b2a70  00 00 00 00 00 00 00 00-02 00 00 00 00 00 00 00  ................
ffffee30`044b2a80  60 52 ff 03 30 ee ff ff-00 00 00 00 00 00 00 00  `R..0...........
ffffee30`044b2a90  08 00 57 23 47 70 62 6d-00 00 00 00 00 00 00 00  ..W#Gpbm........
ffffee30`044b2aa0  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
ffffee30`044b2ab0  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
ffffee30`044b2ac0  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
ffffee30`044b2ad0  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
```

xxxSbtrackInit allocating with tag Usst



```
var_54= byte ptr -54h
var_14= byte ptr -14h
var_10= byte ptr -10h
var_4= dword ptr -4
arg_0= dword ptr  8
arg_4= dword ptr  0Ch
arg_8= dword ptr  10h
arg_C= dword ptr  14h

mov     edi, edi
push    ebp
mov     ebp, esp
sub     esp, 54h
push    ebx
push    edi
mov     edi, [ebp+arg_0]
mov     eax, [edi+8]
xor     ebx, ebx
cmp     [eax+13Ch], ebx
jnz     loc_BF96A85D
```

```
push    esi
push    'tssU'          ; Tag
push    44h             ; NumberOfBytes
push    29h             ; PoolType
call    ds:__imp__ExAllocatePoolWithQuotaTag@12 ; ExAllocatePoolWithQuotaTag(x,x,x)
mov     esi, eax
cmp     esi, ebx
jz      loc_BF96A85C
```

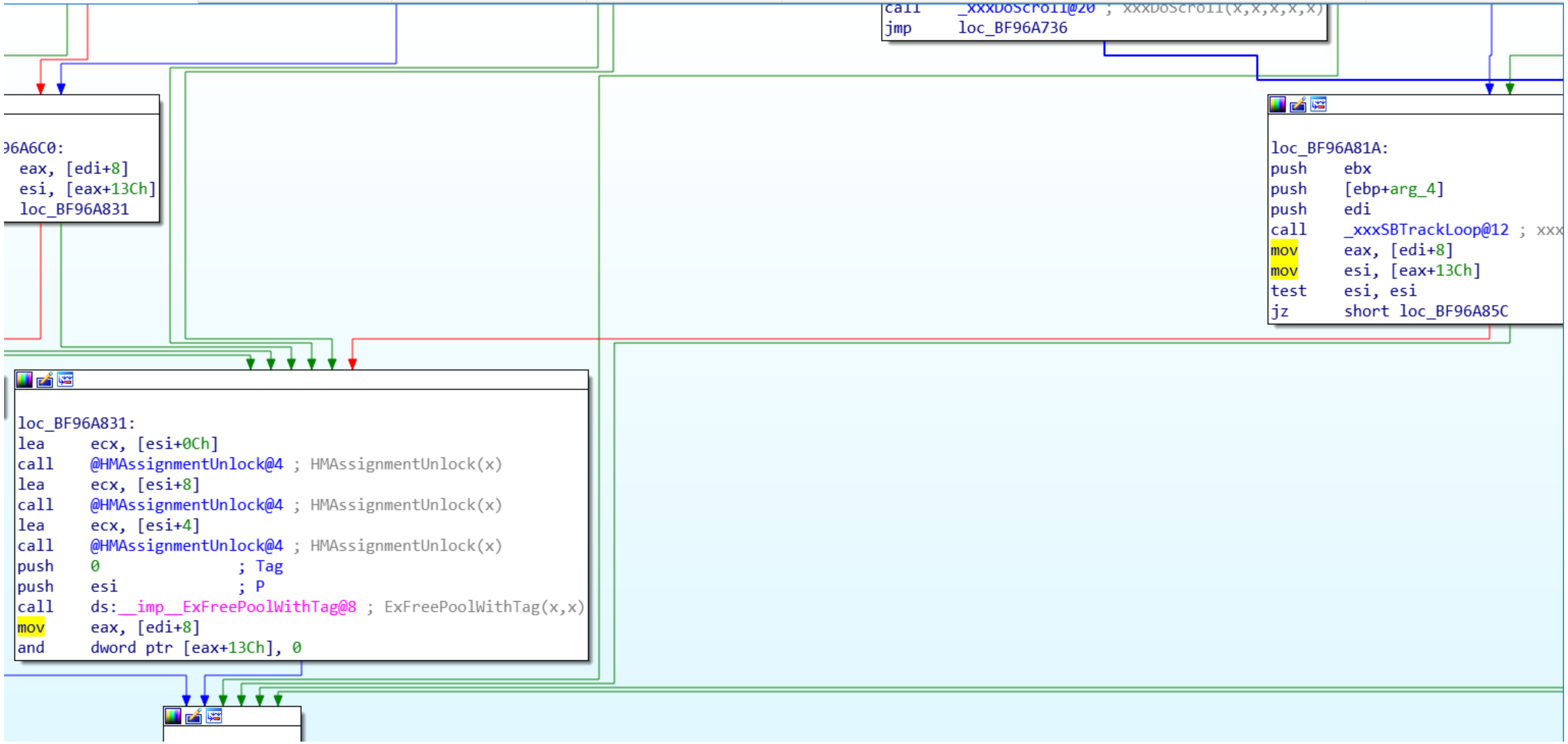
```
and     dword ptr [esi], 0FFFFFFFEh
and     dword ptr [esi+0Ch], 0
lea     ecx, [esi+4]
```

Freeing the ScrollBar Object

- Thanks to ze0r article for this. ☺
- The ScrollBar object gets freed in two ways:

1. User lifts the mouse button (after xxxSbTrackLoop).
2. Calling xxxEndScroll function.

Freeing ScrollBar Window after exiting xxxSBTrackLoop



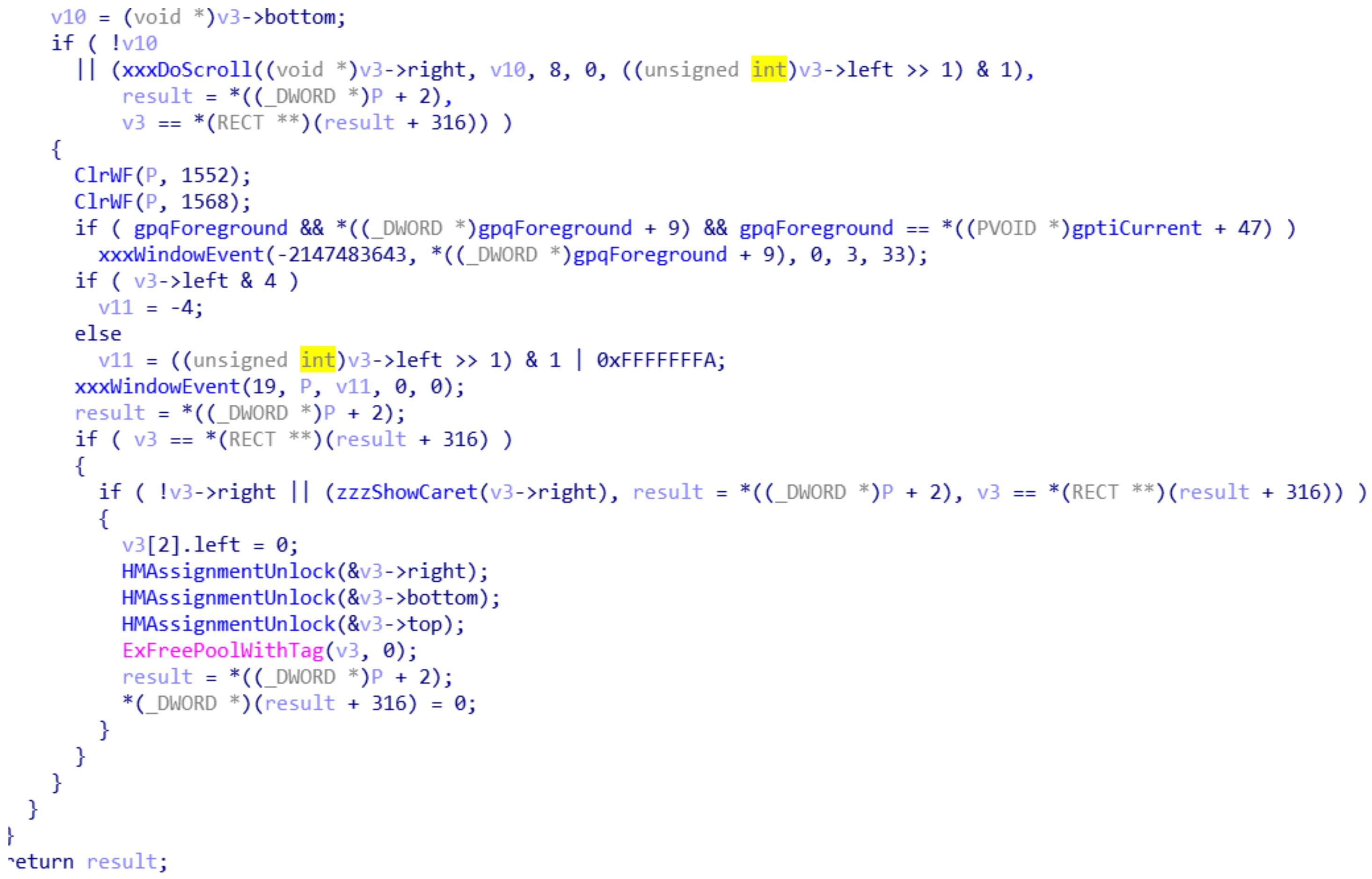
```
call    _xxxDoScroll@20 ; xxxDoScroll(x,x,x,x,x)
jmp     loc_BF96A736
```

```
96A6C0:
   eax, [edi+8]
   esi, [eax+13Ch]
   loc_BF96A831
```

```
loc_BF96A81A:
push    ebx
push    [ebp+arg_4]
push    edi
call    _xxxSBTrackLoop@12 ; xxx
mov     eax, [edi+8]
mov     esi, [eax+13Ch]
test    esi, esi
jz      short loc_BF96A85C
```

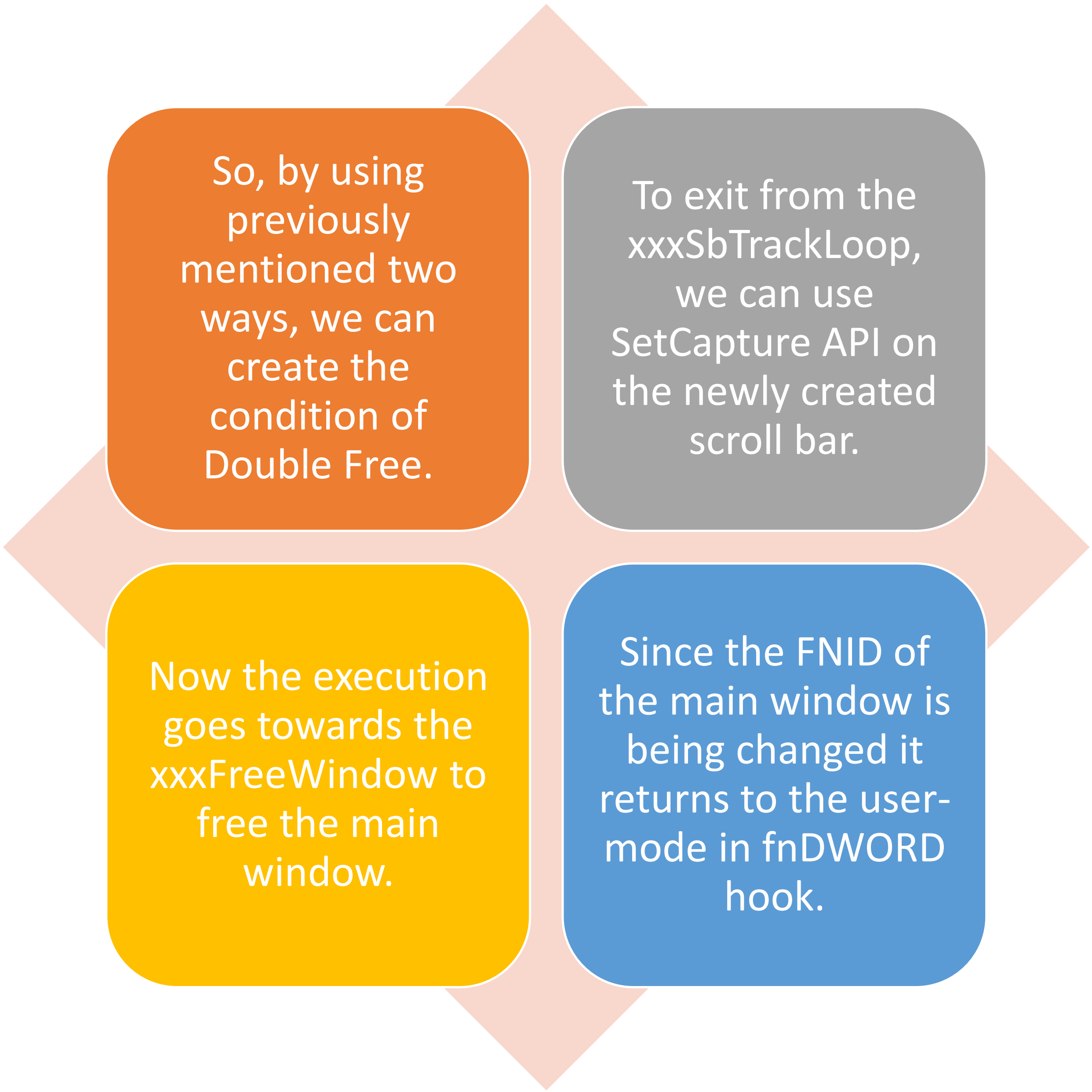
```
loc_BF96A831:
lea     ecx, [esi+0Ch]
call    @HMAssignmentUnlock@4 ; HMAssignmentUnlock(x)
lea     ecx, [esi+8]
call    @HMAssignmentUnlock@4 ; HMAssignmentUnlock(x)
lea     ecx, [esi+4]
call    @HMAssignmentUnlock@4 ; HMAssignmentUnlock(x)
push    0                ; Tag
push    esi              ; P
call    ds:__imp__ExFreePoolWithTag@8 ; ExFreePoolWithTag(x,x)
mov     eax, [edi+8]
and     dword ptr [eax+13Ch], 0
```

Freeing ScrollBar Window through xxxEndScroll Function



```
  v10 = (void *)v3->bottom;
  if ( !v10
   || (xxxDoScroll((void *)v3->right, v10, 8, 0, ((unsigned int)v3->left >> 1) & 1),
       result = *((_DWORD *)P + 2),
       v3 == *(RECT **)(result + 316)) )
  {
   ClrWF(P, 1552);
   ClrWF(P, 1568);
   if ( gpqForeground && *((_DWORD *)gpqForeground + 9) && gpqForeground == *((PVOID *)gptiCurrent + 47) )
    xxxWindowEvent(-2147483643, *((_DWORD *)gpqForeground + 9), 0, 3, 33);
   if ( v3->left & 4 )
    v11 = -4;
   else
    v11 = ((unsigned int)v3->left >> 1) & 1 | 0xFFFFFFFA;
   xxxWindowEvent(19, P, v11, 0, 0);
   result = *((_DWORD *)P + 2);
   if ( v3 == *(RECT **)(result + 316) )
   {
    if ( !v3->right || (zzzShowCaret(v3->right), result = *((_DWORD *)P + 2), v3 == *(RECT **)(result + 316)) )
    {
     v3[2].left = 0;
     HMAssignmentUnlock(&v3->right);
     HMAssignmentUnlock(&v3->bottom);
     HMAssignmentUnlock(&v3->top);
     ExFreePoolWithTag(v3, 0);
     result = *((_DWORD *)P + 2);
     *(_DWORD *)(result + 316) = 0;
    }
   }
  }
 }
}
return result;
```

Forcing a Double Free



So, by using previously mentioned two ways, we can create the condition of Double Free.

To exit from the xxxSbTrackLoop, we can use SetCapture API on the newly created scroll bar.

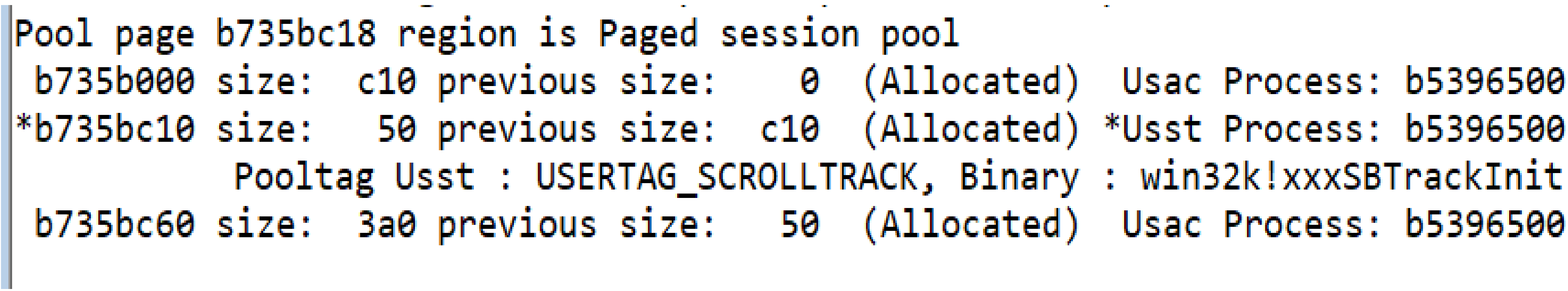
Now the execution goes towards the xxxFreeWindow to free the main window.

Since the FNID of the main window is being changed it returns to the user-mode in fnDWORD hook.

Forcing a Double Free

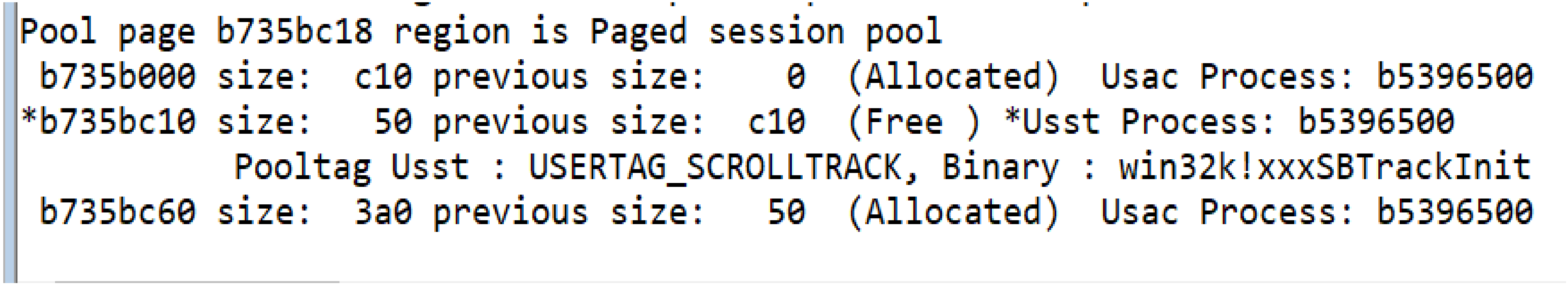
Here we call our *xxxEndScroll* via *SendMessage* API by sending WM_CANCEL mode message to the newly created ScrollBar, which frees the ScrollBar object.

When the *xxxFreeWindow* is executed, it tries to free the already freed ScrollBar object, leads to the Double Free condition.



```
Pool page b735bc18 region is Paged session pool
 b735b000 size:  c10 previous size:    0  (Allocated)  Usac Process: b5396500
*b735bc10 size:   50 previous size:  c10  (Allocated) *Usst Process: b5396500
           Pooltag Usst : USERTAG_SCROLLTRACK, Binary : win32k!xxxSBTrackInit
 b735bc60 size:  3a0 previous size:   50  (Allocated)  Usac Process: b5396500
```

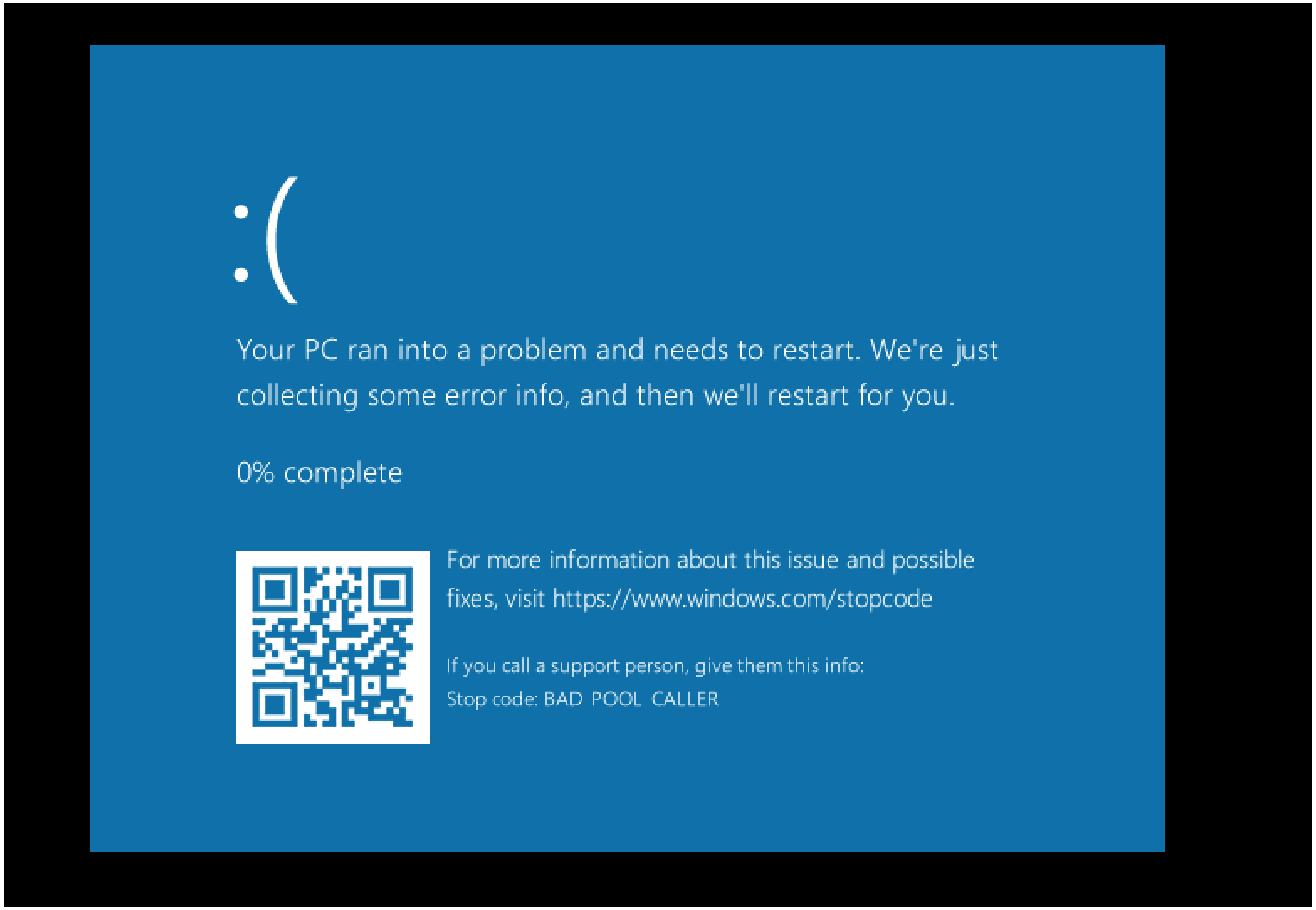
Pool state – Before free()



```
Pool page b735bc18 region is Paged session pool
 b735b000 size:  c10 previous size:    0  (Allocated)  Usac Process: b5396500
*b735bc10 size:   50 previous size:  c10  (Free ) *Usst Process: b5396500
           Pooltag Usst : USERTAG_SCROLLTRACK, Binary : win32k!xxxSBTrackInit
 b735bc60 size:  3a0 previous size:   50  (Allocated)  Usac Process: b5396500
```

Pool state – After free()

What do we get? A BSOD!



:(

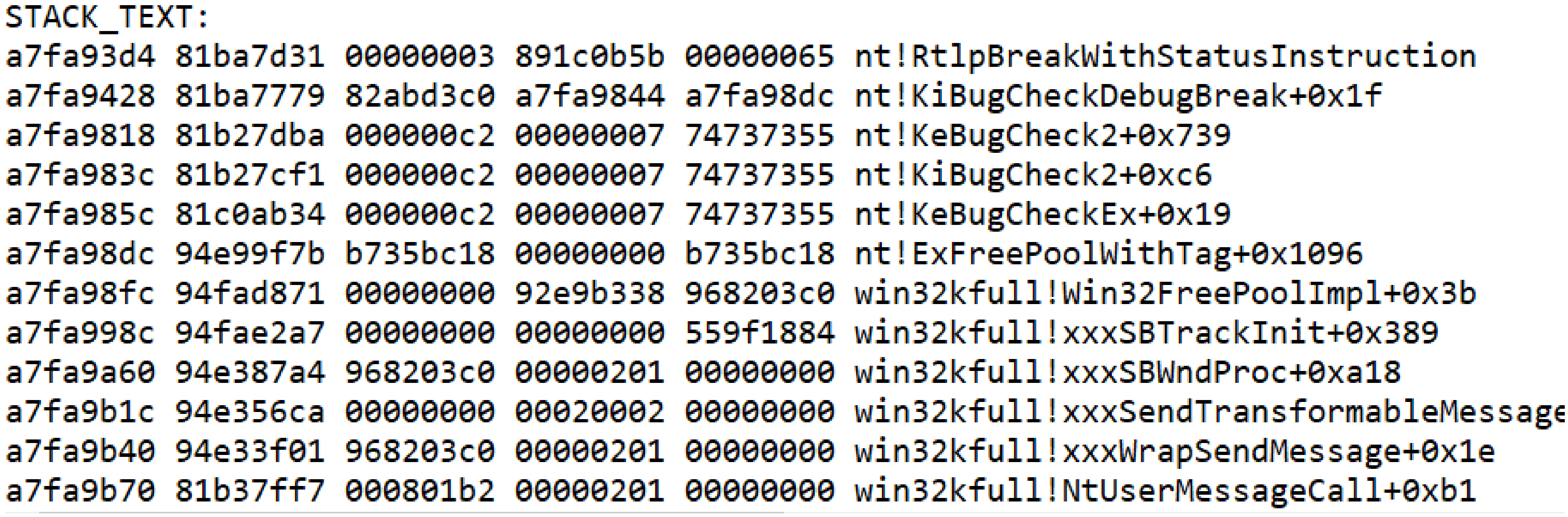
Your PC ran into a problem and needs to restart. We're just collecting some error info, and then we'll restart for you.

0% complete

For more information about this issue and possible fixes, visit https://www.windows.com/stopcode

If you call a support person, give them this info:
Stop code: BAD POOL CALLER

What happened?



```
STACK_TEXT:
a7fa93d4 81ba7d31 00000003 891c0b5b 00000065 nt!RtlpBreakWithStatusInstruction
a7fa9428 81ba7779 82abd3c0 a7fa9844 a7fa98dc nt!KiBugCheckDebugBreak+0x1f
a7fa9818 81b27dba 000000c2 00000007 74737355 nt!KeBugCheck2+0x739
a7fa983c 81b27cf1 000000c2 00000007 74737355 nt!KiBugCheck2+0xc6
a7fa985c 81c0ab34 000000c2 00000007 74737355 nt!KeBugCheckEx+0x19
a7fa98dc 94e99f7b b735bc18 00000000 b735bc18 nt!ExFreePoolWithTag+0x1096
a7fa98fc 94fad871 00000000 92e9b338 968203c0 win32kfull!Win32FreePoolImpl+0x3b
a7fa998c 94fae2a7 00000000 00000000 559f1884 win32kfull!xxxSBTrackInit+0x389
a7fa9a60 94e387a4 968203c0 00000201 00000000 win32kfull!xxxSBWndProc+0xa18
a7fa9b1c 94e356ca 00000000 00020002 00000000 win32kfull!xxxSendTransformableMessage
a7fa9b40 94e33f01 968203c0 00000201 00000000 win32kfull!xxxWrapSendMessage+0x1e
a7fa9b70 81b37ff7 000801b2 00000201 00000000 win32kfull!NtUserMessageCall+0xb1
```

Weaponizing Double Free For Arbitrary R/W Primitive

Designing the exploit

First arrange the heap using feng shui.

Basic idea is to make the memory predictable, allocated by the Windows kernel.

The problematic part is to take read write primitive in only 0x50 bytes of space.

0x50 is the size of usst object used by scrollbar.

Demo

References

Securelist blog - https://securelist.com/cve-2018-8453-used-in-targeted-attacks/88151/

ze0r’s blog - https://paper.seebug.org/798/

Questions?

Thank you