MWR Labs

++

# A Window into Ring0

Sam Brown

Securi-Tay 2017


MWR LABS

++

## Alternative Title

# Please stop using Windows 7, what year is this? Why are you doing that?

MWR LABS

++

# whoami

+ Sam Brown - @_samdb_
+ Consultant in the research practice @ MWR
+ Worky worky – Secure Dev, Code Review, Product Teardowns, Pentesting
+ Research/home time – poking at Windows/driver internals, playing with Angr and Z3

++

# Introduction

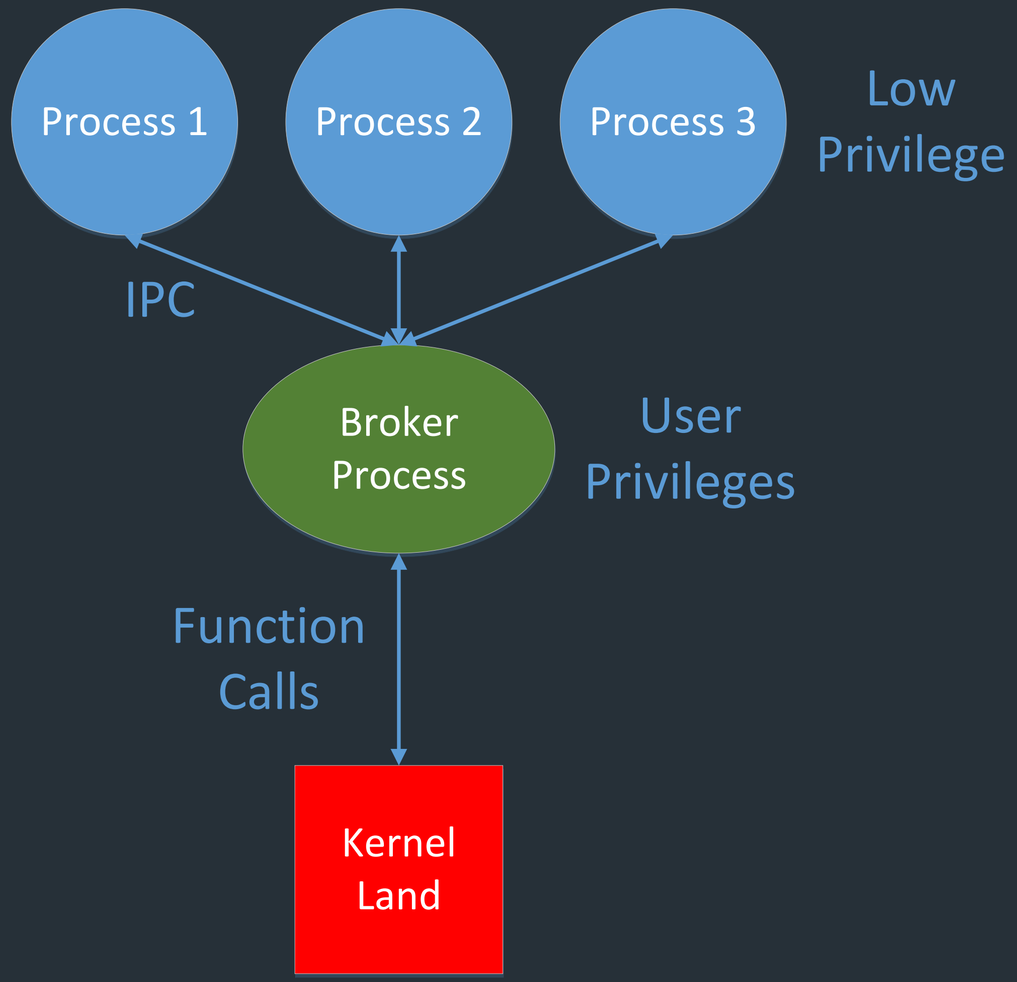
+ Survey style – no 1337 0day
+ Focused on concepts
+ Based off past year of reading, reversing and poking at kernel/driver bugs
+ References at end but all of the things here: https://github.com/sam-b/windows_kernel_resources

# Outline

1. Motivation
2. The Attack Surface
3. Bug Hunting
4. Mitigations
5. CVE-2016-7255
6. Conclusions & Questions

MWR LABS

# ++ Motivation – Sandboxes




"a virtual space in which new or untested software or coding can be run securely."

++

# Motivation – Sandboxes

+ Started appearing in 2006 with IE 7 protected mode
+ Low Integrity processes
+ Increasingly prevalent

Firefox takes the next step towards rolling out multi-process to everyone

Firefox gets closer to offering the same security and stability as competition.

PETER BRIGHT (US) - 22/12/2016, 05:15

++

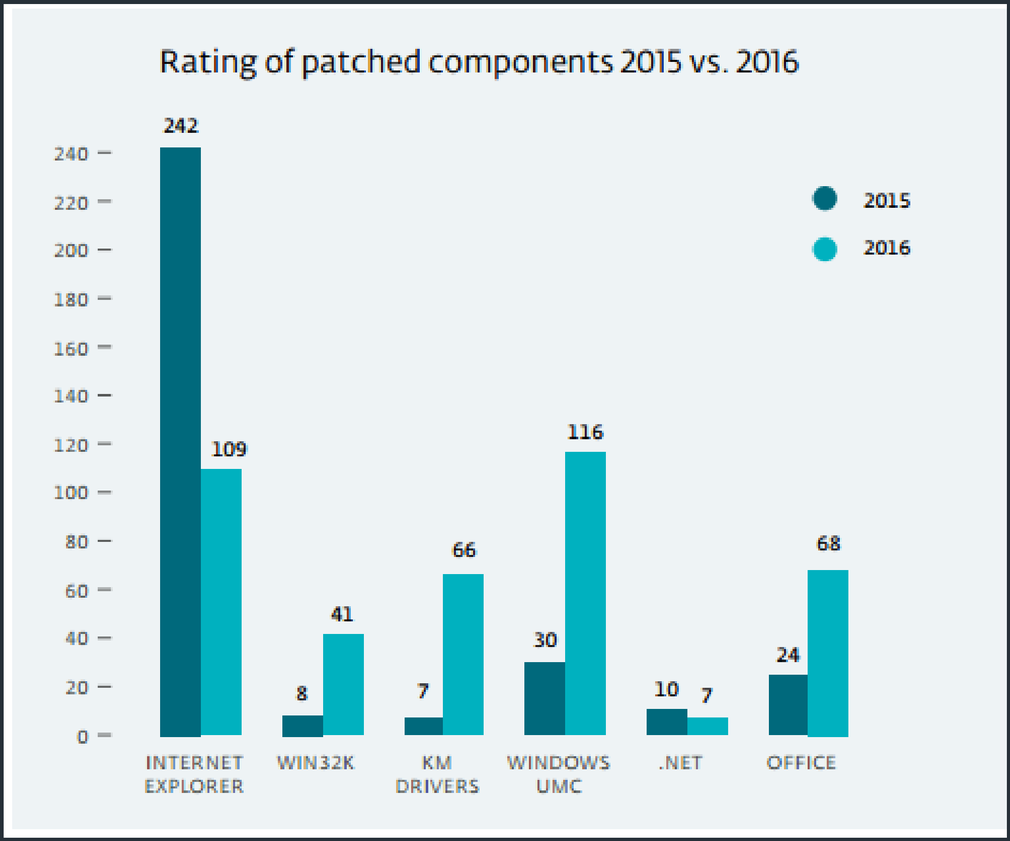
# Motivation – Sandbox Escapes

+ Compromised a client but sandbox containing us
+ EoP exploit required
+ Sandbox broker exploit – limited attack surface but possible

MWR LABS

# ++ Motivation – Sandbox Escapes

+ Kernel – straight to the core, massive attack surface




http://www.welivesecurity.com/wp-content/uploads/2017/01/Windows-Exploitation-2016-A4.pdf

++

# Background

+ We want to escalate our privileges
+ Low Integrity to SYSTEM
+ How?

++

# Background

+ Windows has Access Token objects
+ Think cookies for users
+ Many methods of privescing
+ Steal the Access Token from a process running as SYSTEM
+ Modify users token to have permissions to inject code into a process running as SYSTEM
+ Overwrite a SYSTEM processes security descriptor with NULL

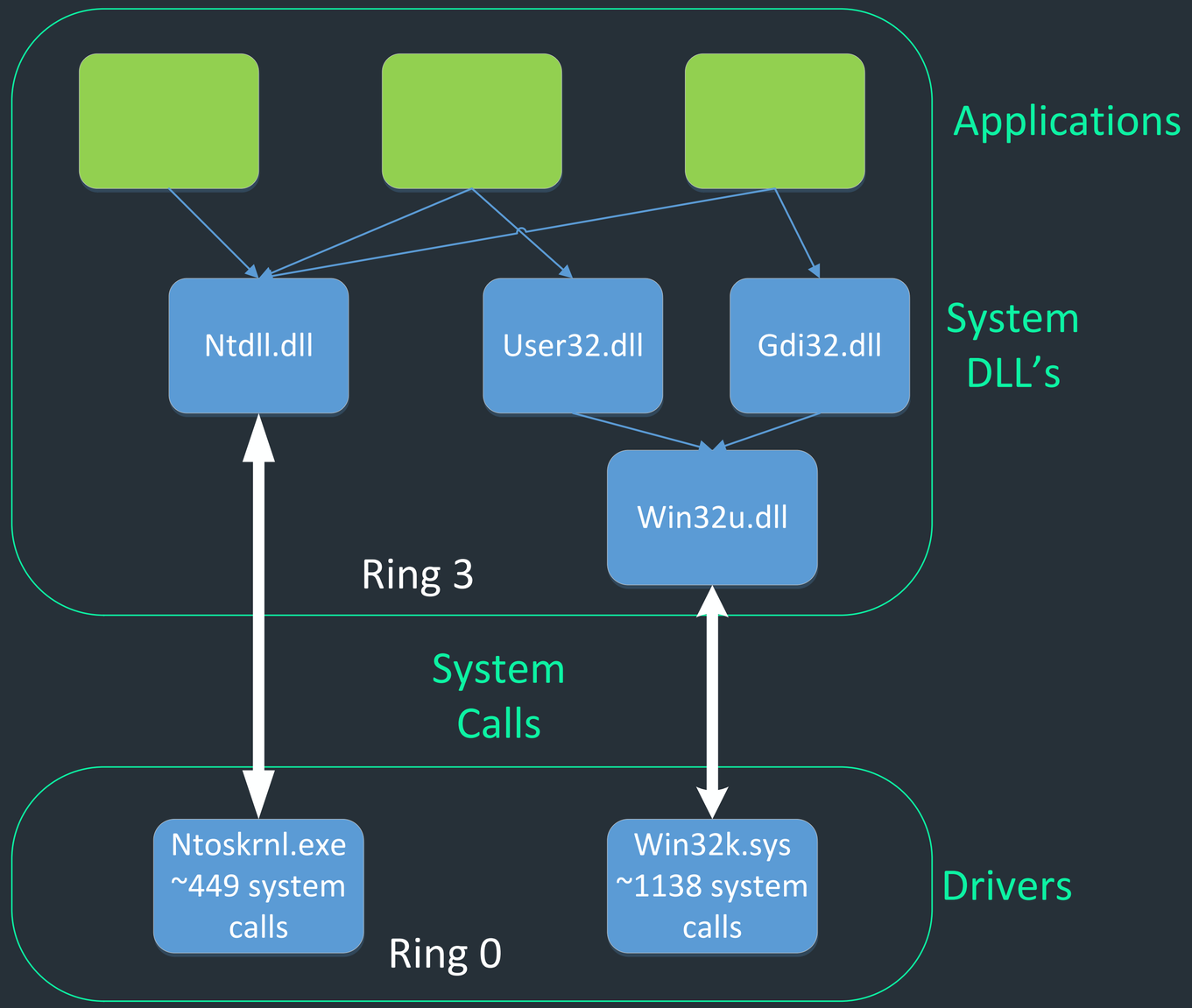
# Outline

1. Motivation
2. The Attack Surface
3. Bug Hunting
4. Mitigations
5. CVE-2016-7255
6. Conclusions & Questions

++

# The Attack Surface

+ System calls
+ Drivers
+ Font Parsing



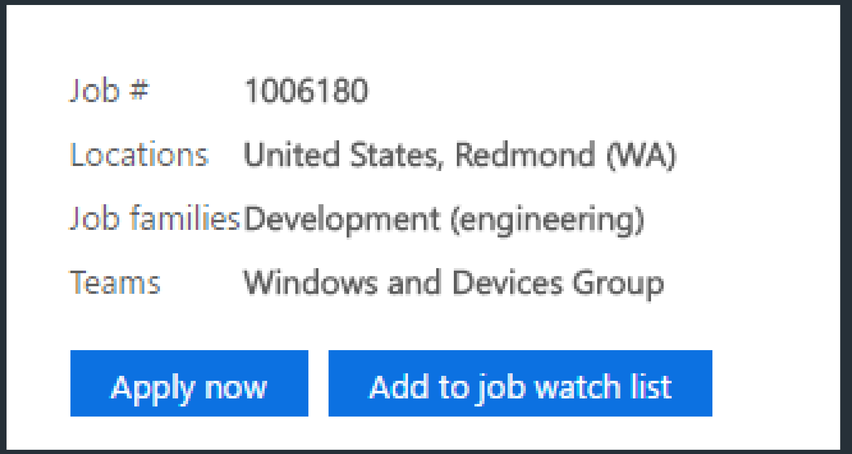

https://github.com/sam-b/windows_syscalls_dumper

MWR LABS

# ++ win32k

- Main Windows graphics driver
- Lots of complex functionality
- Written in the 90's
- All in kernel mode
- "How bad design decisions created the least secure driver on Windows" by Thomas Garnier[1]

tools & processes. One of our components, win32k*.sys, is the vector for 60% or more of all kernel-mode Windows exploits, and is the topic of papers at Black Hat every year, so we are at the cutting edge of fixing vulnerabilities as well as developing mitigations. To



++

# ntoskrnl

+ Windows kernel executive
+ Implements core functionality:
+ Processes, Threads
+ Virtual Memory
+ The registry

++

## ntoskrnl

+ A fraction of the system call count Win32k has
+ Less than half the number of CVE’s
+ Still lots of bugs to be found

# ++ Drivers

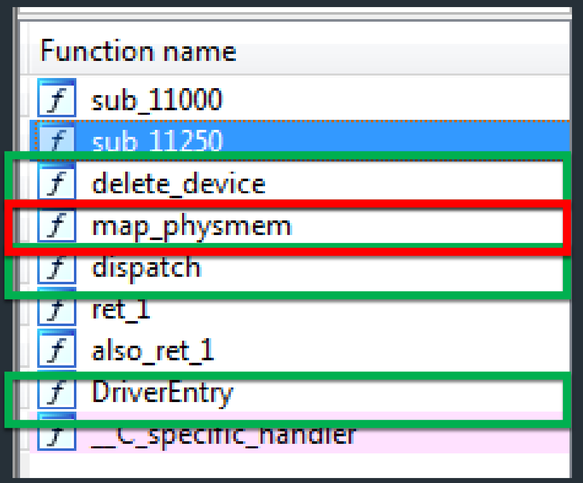
+ Interact with hardware
+ Firmware updaters
+ Antivirus
+ Anti-Cheat

# ++ Driver Communications

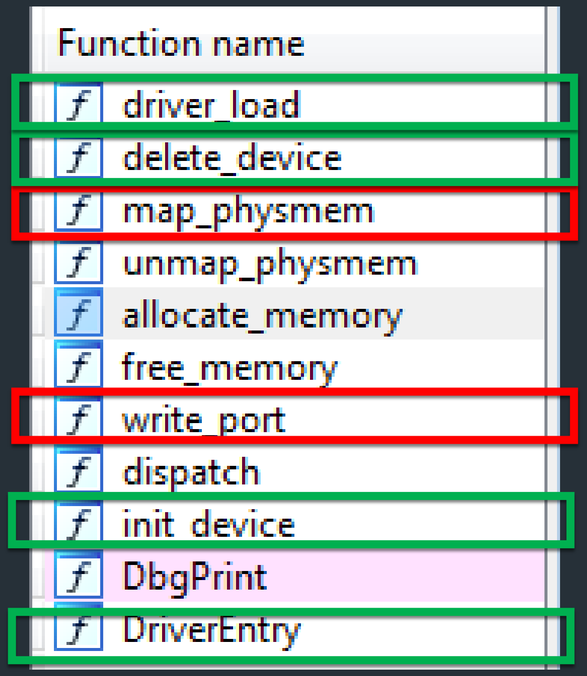
+ Many ways, bugs mostly in…
+ IOCTL codes – triggers a function within the driver, identified by a number – input buffer pointer and size and output buffer pointer and size sent
+ Shared memory – mapped memory shared between user mode and kernel mode, allows for fast data exchange

MWR LABS

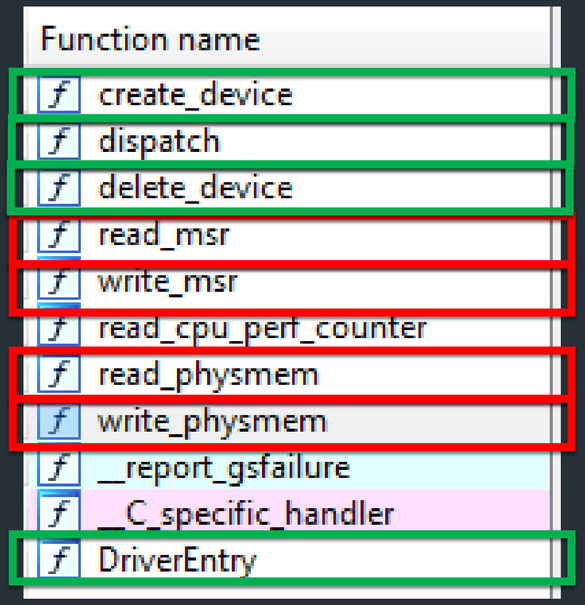
++ Third party drivers do terrible things



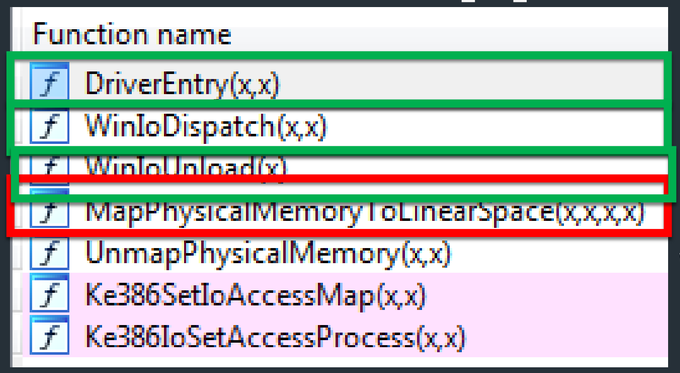

RTCore64.sys
RivaTuner[5]




ASMMAP.sys – ASUS[6]




NTIO.sys - MSI[5]




WinIo.sys - internals.com[5]

++

# Font Parsing

+ Font’s are actually super complex
+ Include small instruction sets
+ Win32k is responsible for parsing TrueType and OpenType fonts

# Outline

1. Motivation
2. The Attack Surface
3. Bug Hunting
4. Mitigations
5. CVE-2016-7255
6. Conclusions & Questions

MWR LABS

++

# Kernel Fuzzing

+ MWR <3's kernel fuzzing
+ https://github.com/mwrlabs/KernelFuzzer

### Windows Kernel Fuzzing

Nils presented Windows Kernel Fuzzing at T2.fi 2015.

### Platform Agnostic Kernel Fuzzing

James Loureiro and Georgi Geshev presented 'Platform Agnostic Kernel Fuzzing' at Def Con 24.

### Fuzzing the Windows kernel

Yong Chuan Koh presented 'Windows kernel fuzzing' HITB GSEC, Singapore

++

## Kernel Fuzzing – general work flow:

1. Select library/system call from catalogue
2. Generate fuzzed values for primitives
3. Grab random Handles from HandleDB if needed
4. Log arguments and call
5. Execute
6. Saves any returned Handles in HandleDB
7. GOTO 1;

MWR Labs

++

# Kernel Fuzzing

## - All of the bugs:

Windows kernel: use-after-free in bitmap handling CCProjectZeroMembers

Windows kernel: NULL pointer dereference with window station and clipboard CCProjectZeroMembers

Windows kernel: use-after-free in WindowStation CCProjectZeroMembers

Windows kernel: Brush object Use-after-free vulnerability CCProjectZeroMembers

Window kernel: use-after-free in bitmap handling #2 CCProjectZeroMembers

Windows kernel: possible NULL pointer dereference of a SURFOBJ CCProjectZeroMembers

Windows kernel: buffer overflow in win32k!vSolidFillRect CCProjectZeroMembers

Windows kernel: use-after-free in HmgAllocateObjectAttr CCProjectZeroMembers

Windows kernel: pool buffer overflow drawing caption bar CCProjectZeroMembers

Windows kernel: use-after-free with UserCommitDesktopMemory CCProjectZeroMembers

Windows kernel: DeferWindowPos use-after-free CCProjectZeroMembers

Windows kernel: pool buffer overflows in NtGdiStretchBlt CCProjectZeroMembers

Windows kernel: use-after-free with printer device contexts CCProjectZeroMembers

Windows kernel: use-after-free with cursor object CCProjectZeroMembers

Windows kernel: use-after-free in bGetRealizedBrush CCProjectZeroMembers

Windows kernel: buffer overflow in NtGdiBitBlt CCProjectZeroMembers

Windows kernel: FlashWindowEx memory corruption CCProjectZeroMembers

Windows kernel use-after-free with device contexts and NtGdiSelectBitmap CCProjectZeroMembers

Windows kernel NtUserScrollDC memory corruption CCProjectZeroMembers

Windows race condition leading to use after free in DestroySMWP CCProjectZeroMembers

Windows Cursor object potential memory leak CCProjectZeroMembers

Windows ndis.sys IOCTL 0x170034 (ndis!NdisNsiGetIfNameForIfIndex) pool buffer overflow CCProjectZeroMembers

win32k clipboard Bitmap use-after-free vulnerability CCProjectZeroMembers

win32k null pointer derefence with Desktop and Clipboard CCProjectZeroMembers

Windows kernel null pointer dereference in win32k!OffsetChildren CCProjectZeroMembers

Windows kernel: NtGdiGetTextExtentExW out-of-bounds memory read CCProjectZeroMembers

Windows kernel: bitmap use-after-free CCProjectZeroMembers

Windows kernel: DrawMenuBarTemp wild-write on 64-bit CCProjectZeroMembers

Windows 7 win32k bitmap use-after-free (#1) CCProjectZeroMembers

Windows 7 win32k bitmap use-after-free (#2) CCProjectZeroMembers

MWR LABS

MWR LABS

++

# Code Review

+ Generally everything's closed source
+ A few exceptions…

```
422     case GWL_ID:
423         /*
424          * Win95 does a TestWF(pwnd, WFCHILD) here, but we'll do the same
425          * check we do everywhere else or it'll cause us trouble.
426          */
427         if (TestwndChild(pwnd)) {
428
429             /*
430              * pwnd->spmenu is an id in this case.
431              */
432             dwOld = (DWORD)pwnd->spmenu;
433             pwnd->spmenu = (struct tagMENU *)dwData;
```

```
#define TestWF(hwnf, f) (GetWindowLong(hwnd, GWL_STYLE) & (f))
#define TestwndChild(hwnd)    (TestWF(hwnd, WFTYPEMASK) == LOBYTE(WFCHILD))
```

The Windows vulnerability is a local privilege escalation in the Windows kernel that can be used as a security sandbox escape. It can be triggered via the win32k.sys system call NtSetWindowLongPtr() for the index GWLP_ID on a window handle with GWL_STYLE set to WS_CHILD. Chrome's sandbox blocks win32k.sys system calls using

**Alex Ionescu** @aionescu · 2 Nov 2016

Here's your Microsoft "actively exploited" Google-disclosed bug right here. Source: GitHub, where people post NT4 source (try Google Search)
pic.twitter.com/touOnvjact

++

## Reverse Engineering

+ Supports other techniques
+ A lot of Windows binaries have debugging symbols on Microsoft's symbol server which helps
+ ReactOS helps
+ Narrowly targeted might be successful
+ Kernel is huge, fuzzers still easily find bugs, why bother?

++

# Reverse Engineering

+ Reversing Third Party drivers has been a good source of bugs
+ Much smaller binaries, lower code quality
+ Tools to help:
+ My IDA plugin: https://github.com/mwrlabs/win_driver_plugin
+ NCC Group's: https://github.com/nccgroup/DriverBuddy

++

# Driver Fuzzing

+ Reverse driver to find IOCTL codes
+ Randomly fuzz them
+ iSEC's driver fuzzer: https://github.com/iSECPartners/DIBF

++

## Font Fuzzing/j00ru is a machine

+ J00ru has been hitting this heavily for years[2]
+ Specs are publically available
+ Targeted fuzzing with custom fuzzers

MWR LABS

++

# Patch Diffing

+ One day bugs
+ Diff kernel code before/after patch Tuesday
+ CVE details and patch notes give hints[7]

CVE-2014-4113
New pointer check

```
-test    ebx, ebx
-jz      short loc_BF937F73
-cmp     ebx, 0FFFFFFFBh
-jz      short loc_BF937F73
-push    dword ptr [ebp-4]
-push    dword ptr [ebp+10h]
-push    1EFh
 push    ebx                                    3214130783   push    ebx
                                                3214130784  +call    _IsMFMWFPWindow@4; IsMFMWFPWindow(x)
                                                3214130789  +test    eax, eax
                                                3214130791  +jz      short loc_BF93BE7F
                                                3214130793  +push    [ebp+Address] ; Address
                                                3214130796  +push    dword ptr [ebp+UnicodeString]; UnicodeString
                                                3214130799  +push    1EFh ; MbString
                                                3214130804  +push    ebx ; P
 call    _xxxSendMessage@16; xxxSendMessage(x,x,x,x)   3214130805   call    _xxxSendMessage@16; xxxSendMessage(x,x,x,x)
```

# Outline

1. Motivation
2. The Attack Surface
3. Bug Hunting
4. Mitigations
5. CVE-2016-7255
6. Conclusions & Questions

++

# Mitigations

- **Type 0 - Strong Mitigation**
  *End a bug class.*
- **Type 1 - Weak Mitigation**
  *End an exploitation technique.*
- **Type 2 - Attack Surface Reduction**
  *Remove a set of exposed functionality.*
- **Type 3 - Chain Extension**
  *Increase the number of bugs required in an exploit.*

Ben Hawkes, USENIX Enigma 2016 - What Makes Software Exploitation Hard?

++

# Mitigations

+ Many mitigations in modern Windows
+ Only covering a few key/interesting ones
+ Being added to Windows 10 rapidly

++

# Once upon a time…

+ Kernel memory marked NX
+ Map shellcode in usermode
+ Control flow hijacking exploit? Jump to it
+ Write-What-Where? Overwrite an entry in a function table to point at it

++

# SMEP

+ Supervisor Mode Execution Prevention
+ Introduced with Intel Ivy Bridge Processors ~April 2012
+ First supported in Windows 8
+ Causes a BSOD on kernel mode attempting to execute user mode memory
+ Type 1 Mitigation

++

## Bypasses

+ Data only attacks
+ Return Oriented Programming
+ Or…

MWR LABS

++

# Just have a friendly driver disable it…

**Double KO! Capcom's *Street Fighter V* installs hidden rootkit on PCs**

Fatality – wait, no, what? That's the other game

```
lea     rax, disable_smep
lea     rcx, [rsp+48h+var_28]
call    rax ; disable_smep
mov     rcx, [rsp+48h+var_18]
call    [rsp+48h+var_20] ; execute shellcode \o/
lea     rax, enable_smep
lea     rcx, [rsp+48h+var_28]
call    rax ; enable_smep
```

- deobfuscate_device_name
- init_device
- unimplemented_handler
- disable_smep_and_execute
- dispatch
- DriverEntry
- disable_smep
- enable_smep

++

# KASLR

+ Kernel Address Space Layout Randomisation
+ Randomizes addresses objects are loaded at
+ Introduced in Vista, potentially a type 3 mitigation
+ Randomness++ since

++

# KASLR – Address Leaks

+ NtQuerySystemInformation

+ Undocumented function for getting information about the system

++

# KASLR – Address Leaks

## SystemHandleInformation

```
PID: 3072       Object 0x84B50AE8       Handle 0xB8
PID: 3072       Object 0x845DFFF0       Handle 0xBC
PID: 3072       Object 0x847E68B8       Handle 0xC0
PID: 3072       Object 0x85A65C18       Handle 0xC4
PID: 3072       Object 0xA519F558       Handle 0xC8
PID: 3072       Object 0x963047E0       Handle 0xCC
PID: 3072       Object 0x8463B8A0       Handle 0xD0
PID: 264        Object 0x8EE9F838       Handle 0x4
PID: 264        Object 0x8544DAF8       Handle 0x8
PID: 264        Object 0x85A9DD90       Handle 0xC
PID: 264        Object 0x85A99F00       Handle 0x10
PID: 264        Object 0x84A9B038       Handle 0x14
PID: 264        Object 0x98265898       Handle 0x18
PID: 264        Object 0xB2AA7030       Handle 0x1C

C:\Users\sam\Documents\Visual Studio 2015\Projects\NtQuerySysInfo_SystemHandleIn
formation\Debug>NtQuerySysInfo_SystemHandleInformation.exe_
```

MWR LABS

++

# KASLR – Address Leaks

## SystemModuleInformation

```
Module name \SystemRoot\System32\DRIVERS\srv.sys          Base Address 0x96F85000
Module name \SystemRoot\system32\drivers\spsys.sys        Base Address 0xA2E02000
Module name \SystemRoot\System32\Drivers\BTHUSB.sys       Base Address 0xA2E6C000
Module name \SystemRoot\System32\Drivers\bthport.sys      Base Address 0xA2E7E000
Module name \SystemRoot\system32\DRIVERS\rfcomm.sys       Base Address 0xA2EE2000
Module name \SystemRoot\system32\DRIVERS\BthEnum.sys      Base Address 0xA2F06000
Module name \SystemRoot\system32\DRIVERS\bthpan.sys       Base Address 0xA2F13000
Module name \Windows\System32\ntdll.dll Base Address 0x77810000
Module name \Windows\System32\smss.exe  Base Address 0x47AF0000
Module name \Windows\System32\apisetschema.dll  Base Address 0x77A50000
Module name \Windows\System32\autochk.exe       Base Address 0x330000

C:\Users\sam\Documents\Visual Studio 2015\Projects\NtQuerySysInfo_SystemModuleIn
formation\Debug>NtQuerySysInfo_SystemModuleInformation.exe_
```

++

# KASLR – Address Leaks

Windows 8.1, Low Integrity ☹

```
C:\Users\sam\Desktop\windows_kernel_address_leaks\windows_kernel_address_leaks\N
tQuerySysInfo_SystemLockInformation\x64\Debug>NtQuerySysInfo_SystemLockInformati
on.exe
NtQuerySystemInformation failed with error code 0xC0000022
```

| Technique | Windows 7 | Windows 8 | Windows 8.1 Low Integrity | Windows 8.1 Medium Integrity | Windows 10 Low Integrity | Windows 10 Medium Integrity |
|---|---|---|---|---|---|---|
| NtQuerySystemInformation (SystemHandleInformation) | ✓ | ✓ | ✗ | ✓ | ✗ | ✓ |
| NtQuerySystemInformation (SystemLockInformation) | ✓ | ✓ | ✗ | ✓ | ✗ | ✓ |
| NtQuerySystemInformation (SystemModuleInformation) | ✓ | ✓ | ✗ | ✓ | ✗ | ✓ |
| NtQuerySystemInformation (SystemProcessInformation) | ✓ | ✓ | ✗ | ✓ | ✗ | ✓ |
| NtQuerySystemInformation (SystemBigPoolInformation) | ✓ | ✓ | ✗ | ✓ | ✗ | ✓ |
| System Call Return Values | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ |
| Win32k Shared Info User Handle Table | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Descriptor Tables | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| HMValidateHandle | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |

MWR LABS

https://samdb.xyz/revisiting-windows-security-hardening-through-kernel-address-protection/ https://github.com/sam-b/windows_kernel_address_leaks

++

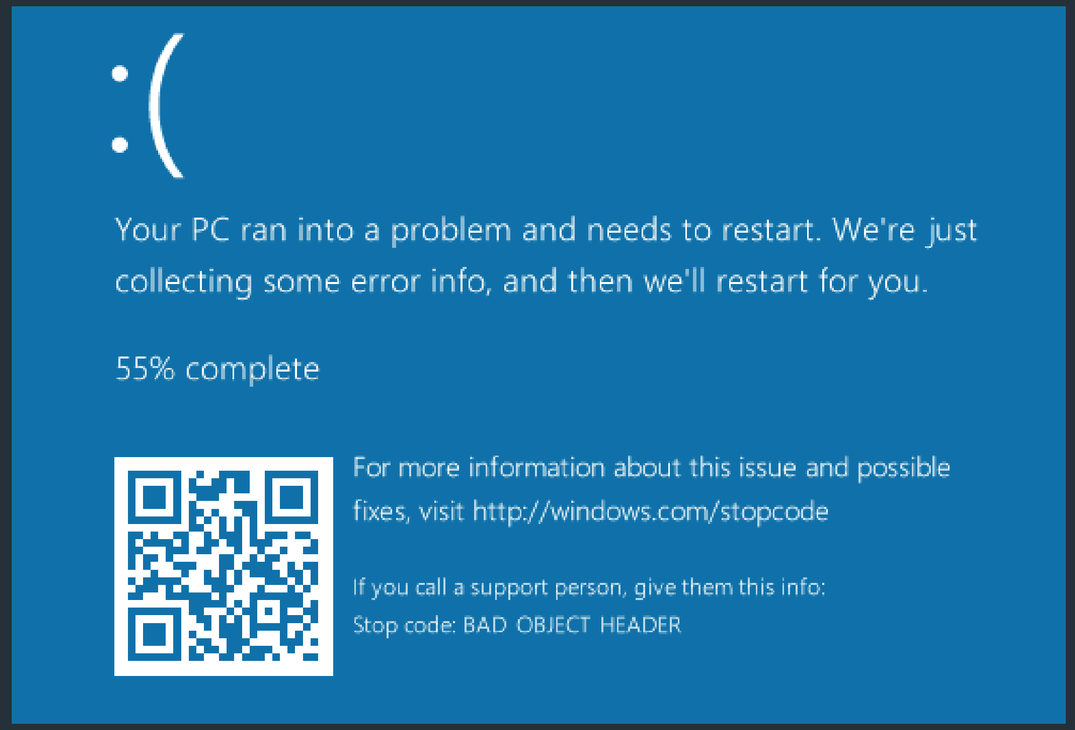
# NULL Page Mapping

+ NULL pointer deference's
+ Super common C/C++ coding error
+ Map the NULL page from user mode
+ Manipulate kernel control flow by customising the data you control
+ Gone as of Windows 7 64 bit
+ Type 0 mitigation

++

# NULL Security Descriptor Protection

+ SecurityDescriptor field header == NULL?
+ Is it a process object?
+ SecurityRequired flag set?
+ Nettitude did an awesome writeup[3]
+ Type 1 mitigation



++

# Moving Font Parsing out of the kernel

+ Windows 10 anniversary update
+ Font parsing now done in an AppContainer[4][9]
+ Type 2 mitigation

++

# Win32k Lockdown

+ Stop processes using win32k[8]
+ Type 2 mitigation

# Outline

1. Motivation
2. The Attack Surface
3. Bug Hunting
4. Mitigations
5. CVE-2016-7255
6. Conclusions & Questions

# ++ CVE-2016-7255/MS16-135

## Disclosing vulnerabilities to protect users

October 31, 2016

Posted by Neel Mehta and Billy Leonard, Threat Analysis Group

On Friday, October 21st, we reported 0-day vulnerabilities — previously publicly-unknown vulnerabilities — to Adobe and Microsoft. Adobe updated Flash on October 26th to address CVE-2016-7855; this update is available via Adobe's updater and Chrome auto-update.

After 7 days, per our published policy for actively exploited critical vulnerabilities, we are today disclosing the existence of a remaining critical vulnerability in Windows for which no advisory or fix has yet been released. This vulnerability is particularly serious because we know it is being actively exploited.

https://securingtomorrow.mcafee.com/mcafee-labs/digging-windows-kernel-privilege-escalation-vulnerability-cve-2016-7255/

http://blog.trendmicro.com/trendlabs-security-intelligence/one-bit-rule-system-analyzing-cve-2016-7255-exploit-wild/

++

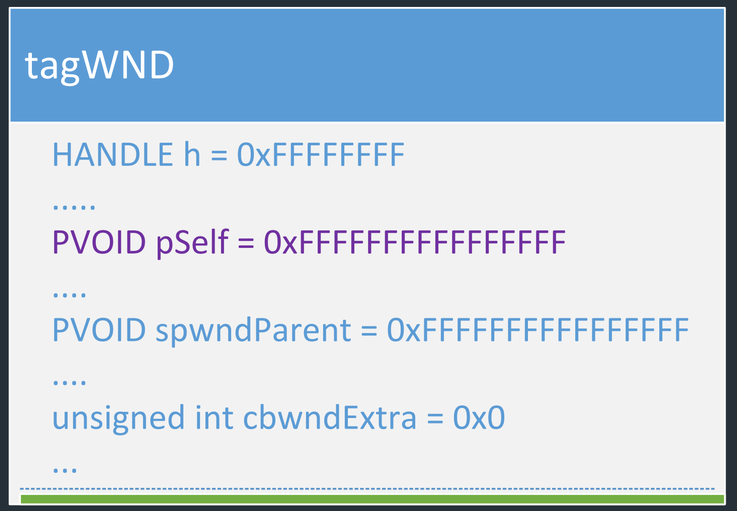
# Primitives

+ One kernel structure leak
+ One kernel memory corruption vulnerability – ‘or’ any value with 4
+ Combined for SYSTEM code exec on Windows 7 to 10, 32 + 64 bit
+ Source: https://github.com/mwrlabs/CVE-2016-7255

++

# Data Leak

+ void* HMValidateHandle(HANDLE h, int type);
+ Undocumented/unexported function in user32
+ Copies entire tagWND structure into user memory
+ Helpfully tagWND includes a pointer to itself :D



++

# Corruption Primitive

+ Window object
+ NtUserSetWindowLongPtr, can modify spmenu with no checks
+ xxxNextWindow takes this value and uses it as a pointer to a tagMenu
+ Sets a single bit the address + 0x28 using an ‘or’ with 4
+ Allows a byte at any address in memory to have it’s 6th bit set

++

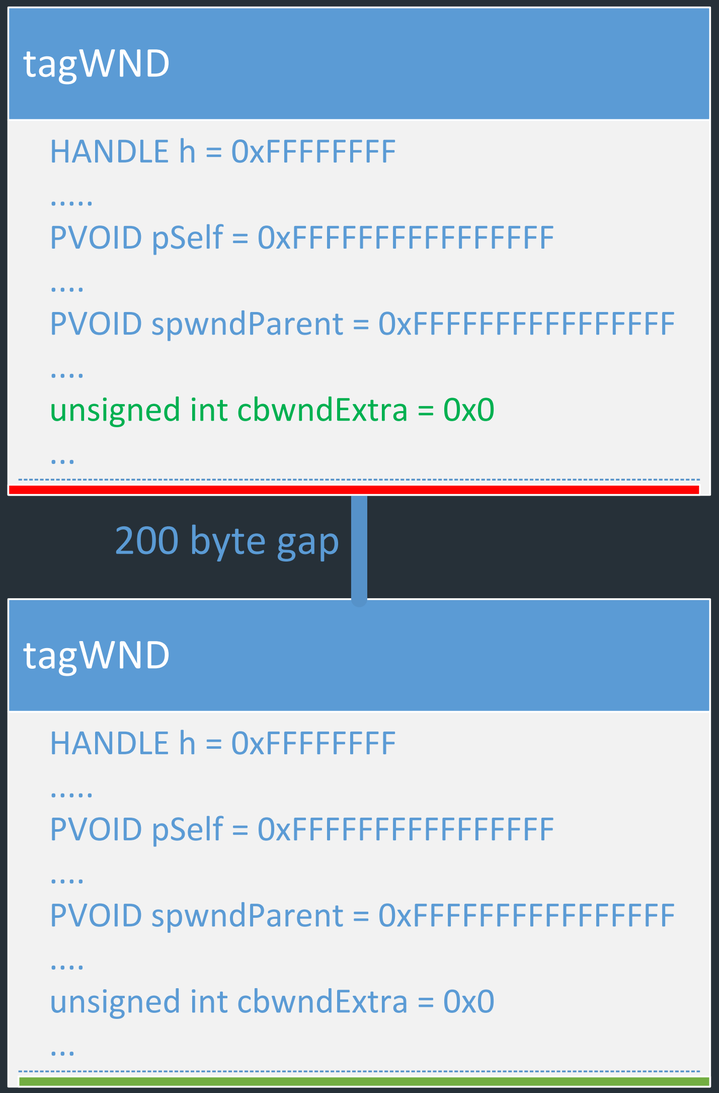
# Exploitation – setup

+ Create 0x100 Window objects
+ HMValidateHandle to leak locations in kernel memory
+ Find two that are < 0x3fd00 apart
+ Destroy spares

++

# Exploitation – Initial corruption

+ Extra memory after a tagWND
+ Size == cbwndExtra

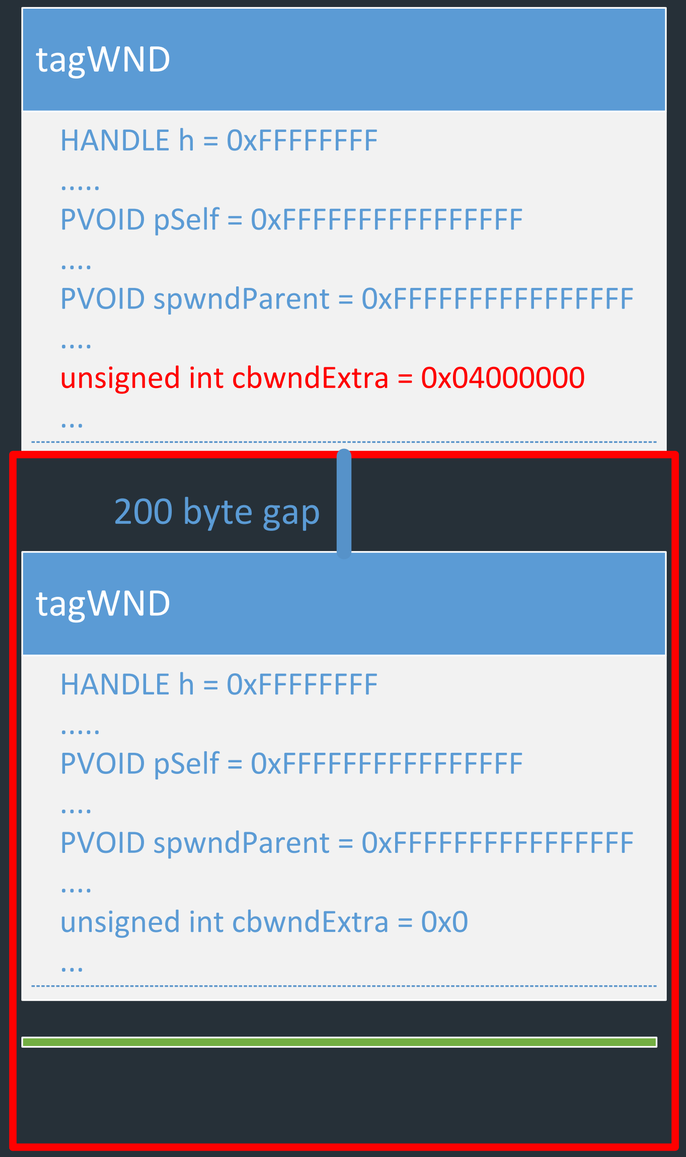


MWR LABS

++

# Exploitation – Initial corruption

+ Use the corruption primitive to ‘or’ highest byte of cbWndExtra with 4
+ 0 -> 0x04000000
+ Extra memory now includes the secondary tagWND structure

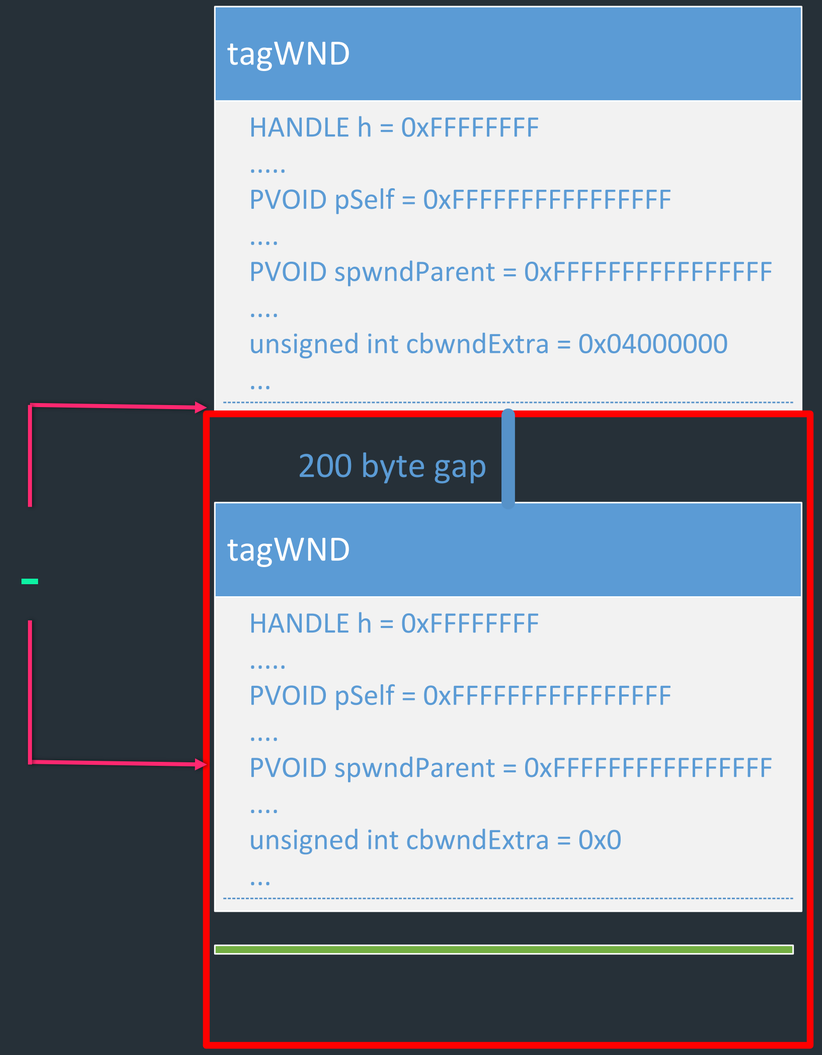


MWR LABS

++

# Exploitation – Read primitive

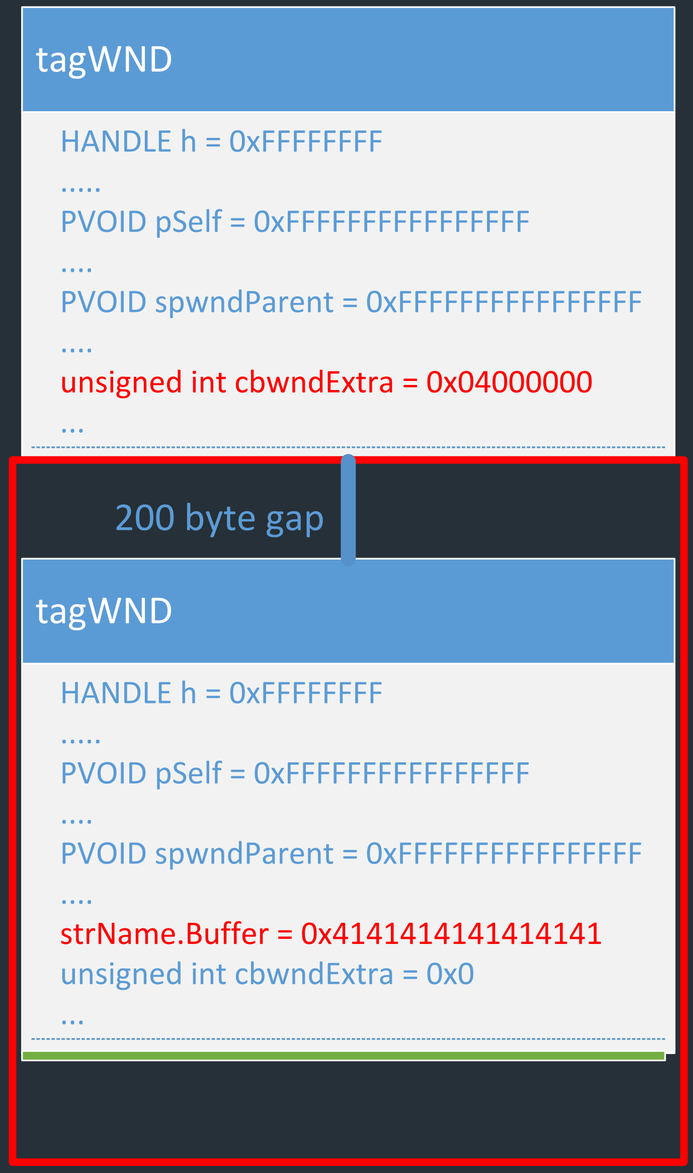
+ Corrupt tagWND -> any address read
+ spwndParent field – pointer to parent window
+ NtUserGetAncestor reads 32 bit int at spwndParent
+ End of tagWND 1 – start of tagWND 2 spwndParent



++

# Exploitation – Read primitive

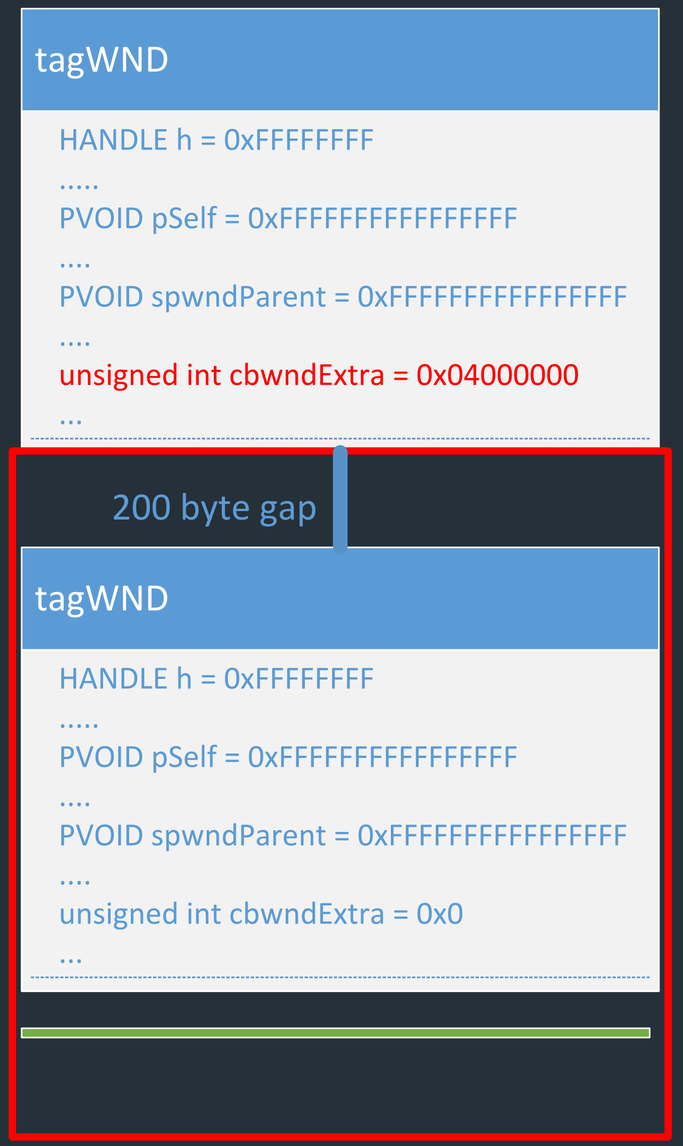
+ Call NtUserSetWindowLongPtr(primaryWindow, diff, TARGET_ADDRESS)
+ NtUserGetAncestor  to read it



++

# Exploitation – Read primitive

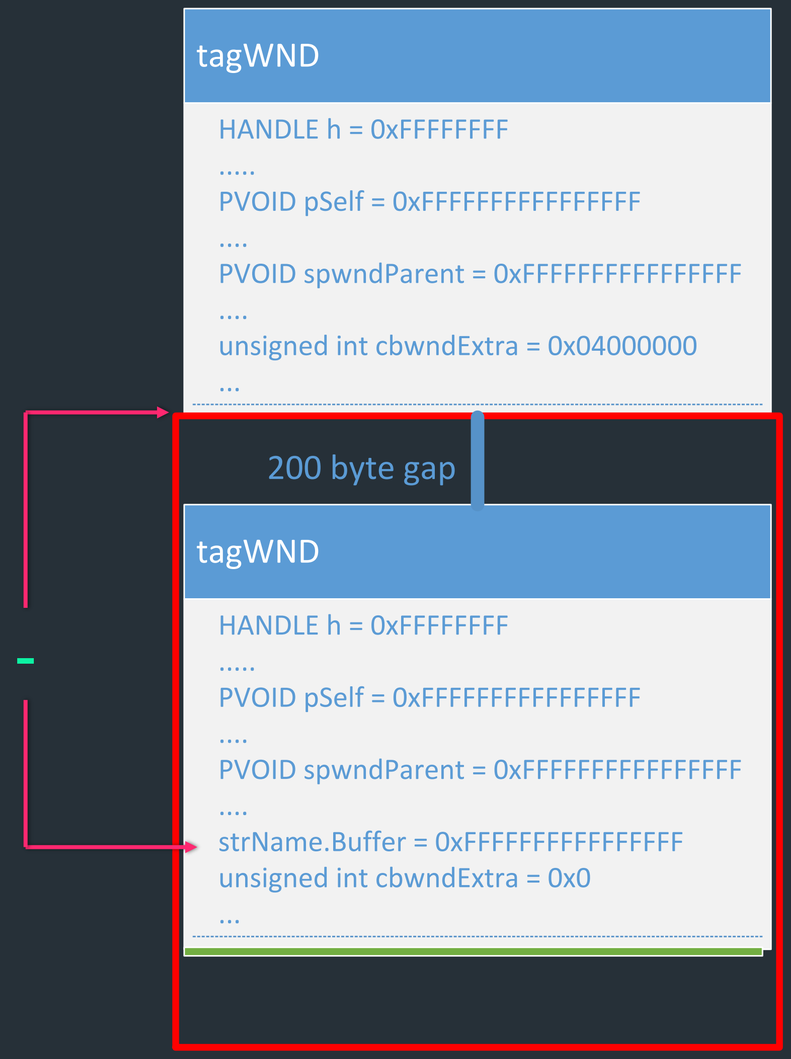
+ Call NtUserSetWindowLongPtr(primaryWindow, diff, TARGET_ADDRESS)
+ NtUserGetAncestor  to read it



++

# Exploitation – Write primitive

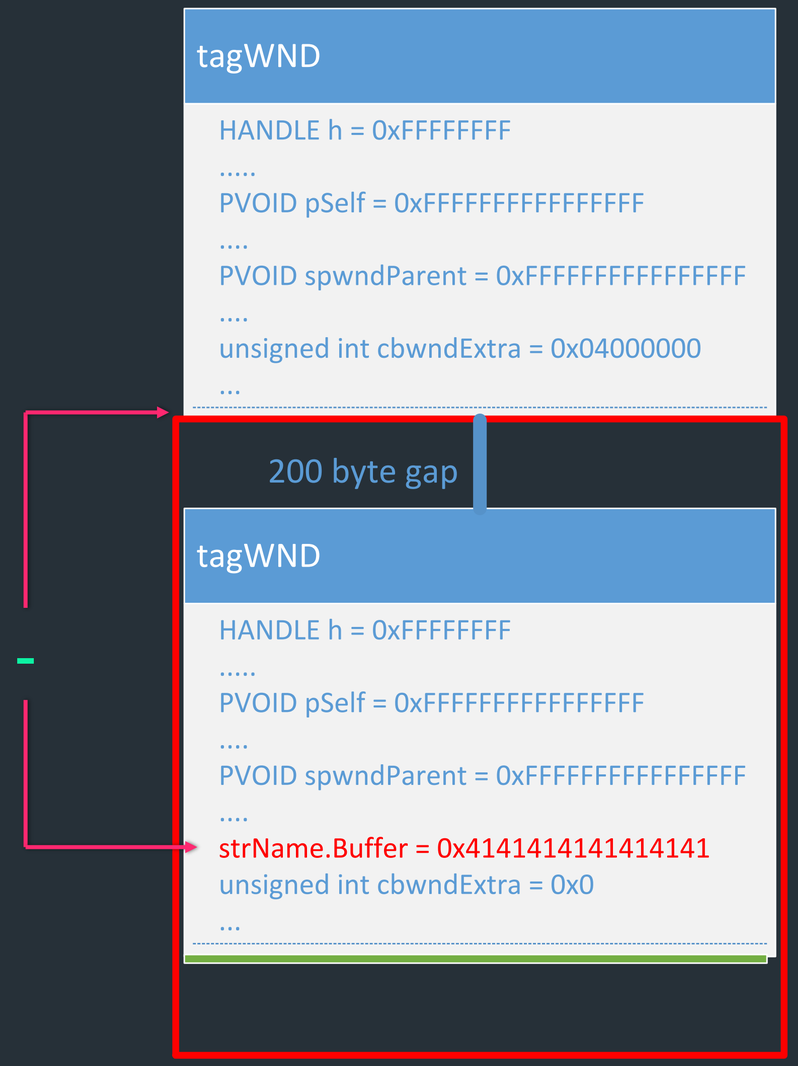
+ Turn corrupting a tagWND into an any address write
+ tagWND has a name field – overwrite it's buffer pointer with the address we want to write
+ Call SetWindowText to write arbitrary data to it



++

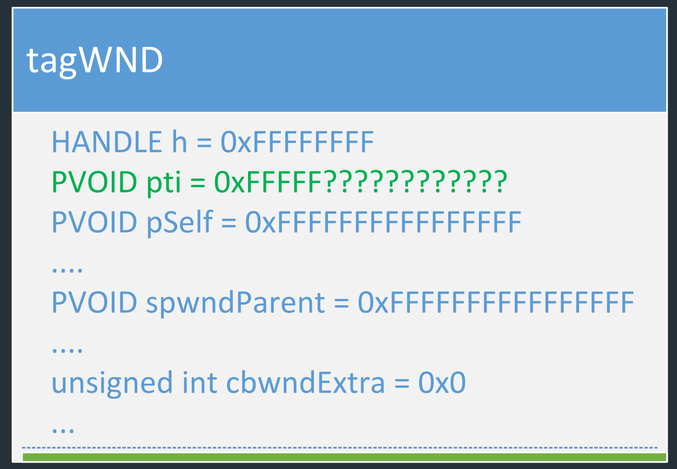
# Exploitation – Write primitive

+ Turn corrupting a tagWND into an any address write
+ tagWND has a name field – overwrite it's buffer pointer with the address we want to write
+ Call SetWindowText to write arbitrary data to it



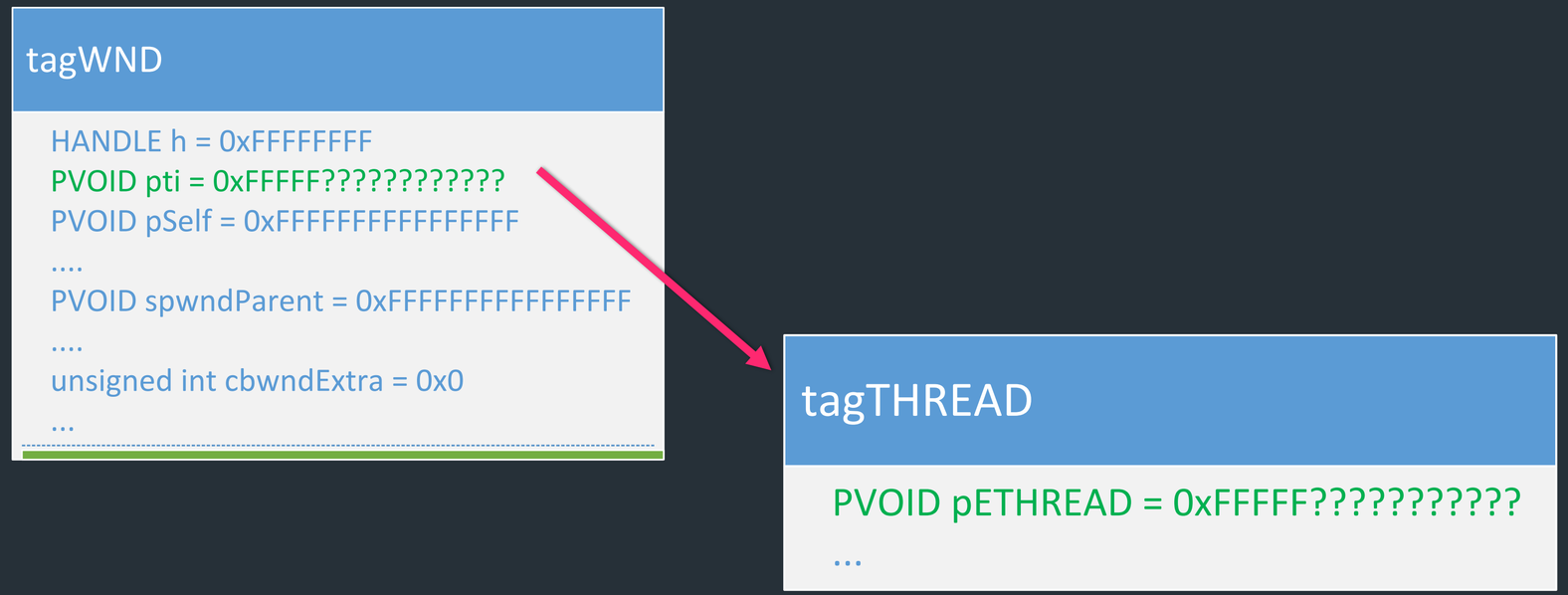
++

# Exploitation – Privesc



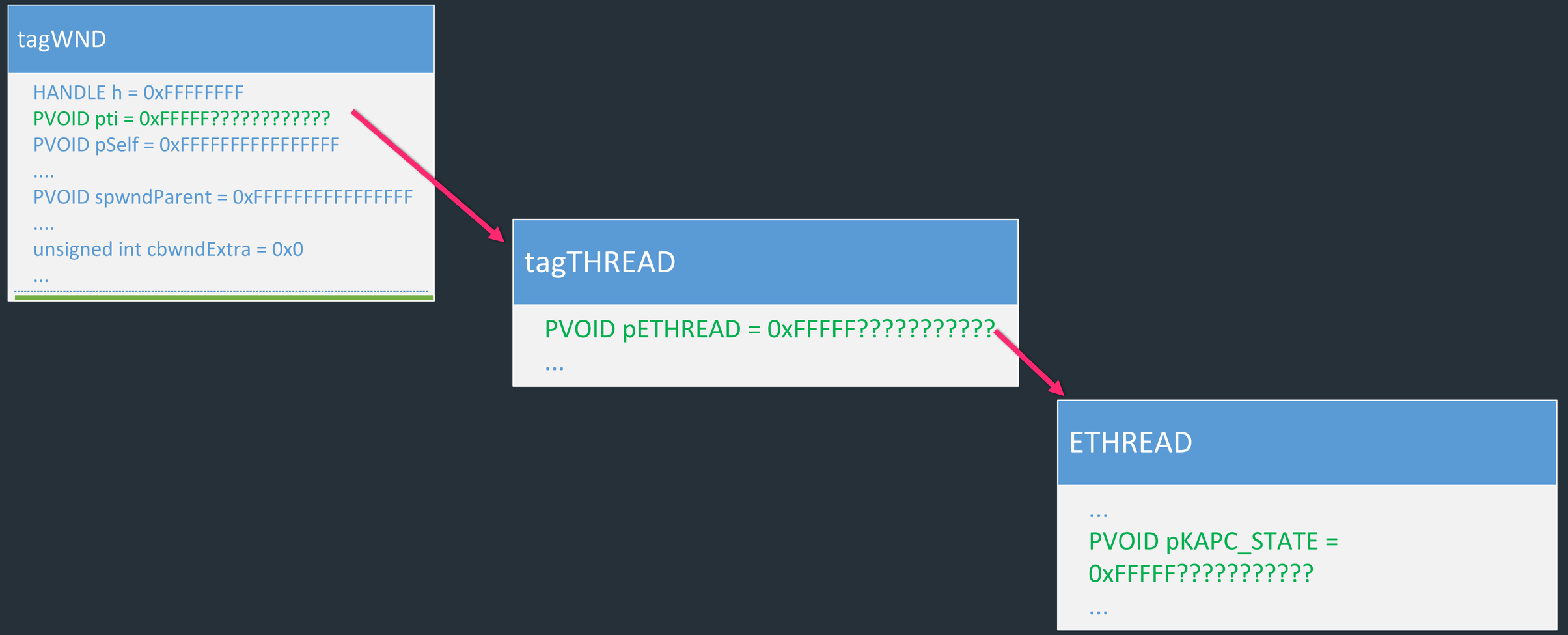
++

# Exploitation – Privesc



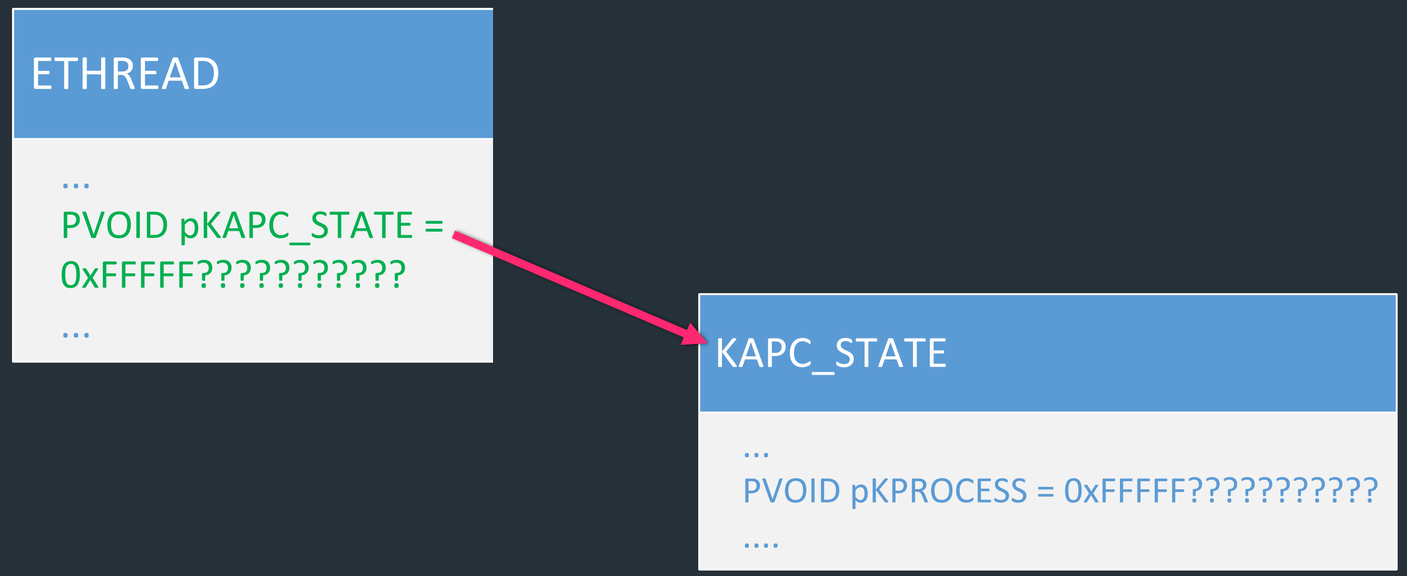
++

# Exploitation – Privesc



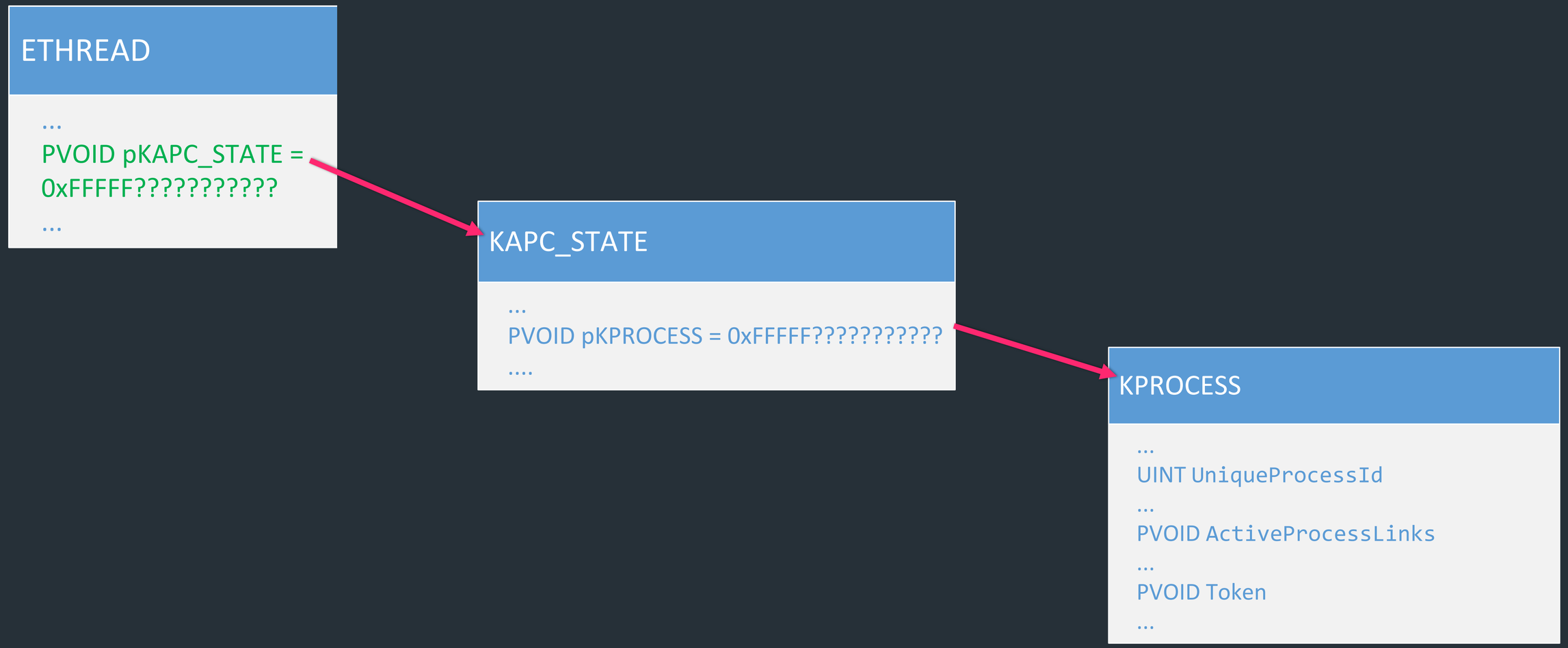
++

# Exploitation – Privesc



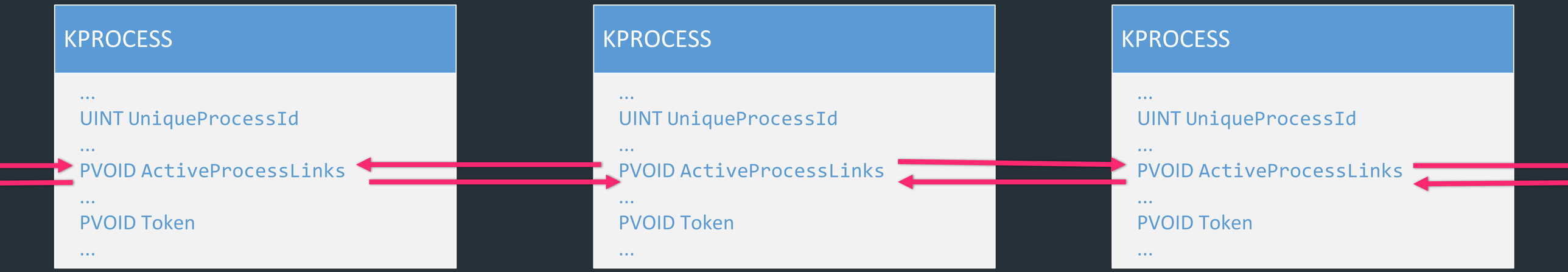
++

# Exploitation – Privesc



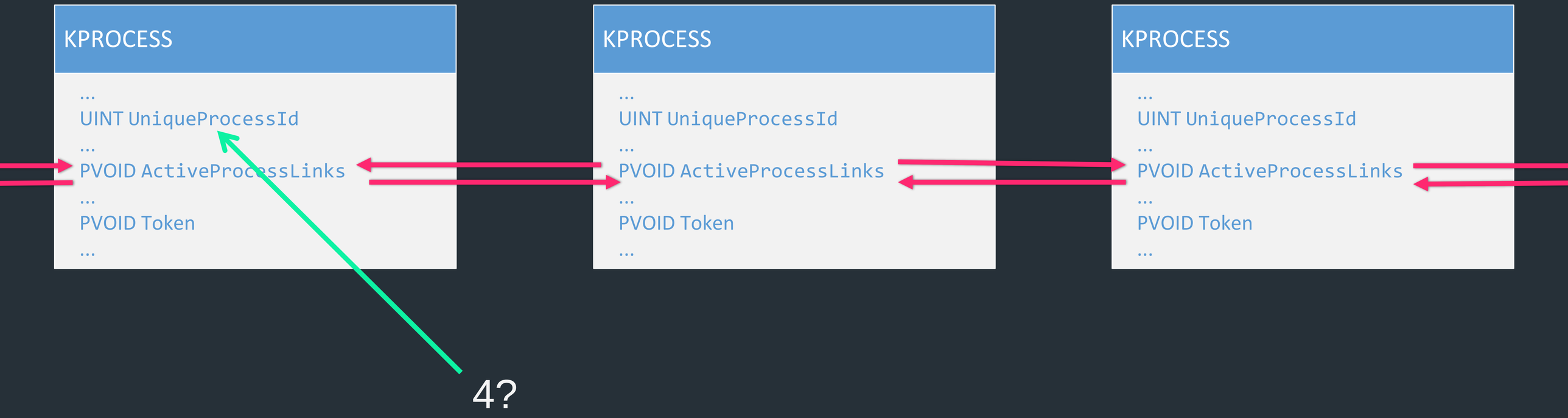
++

# Exploitation – Privesc



++

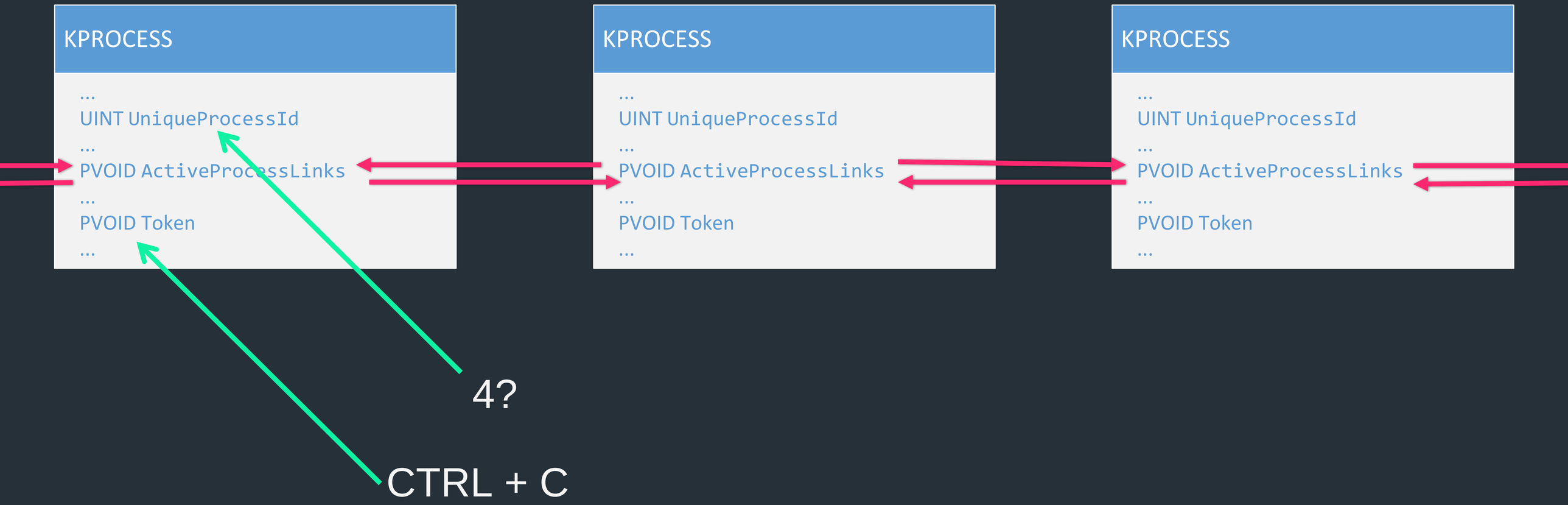
# Exploitation – Privesc



MWR LABS

++

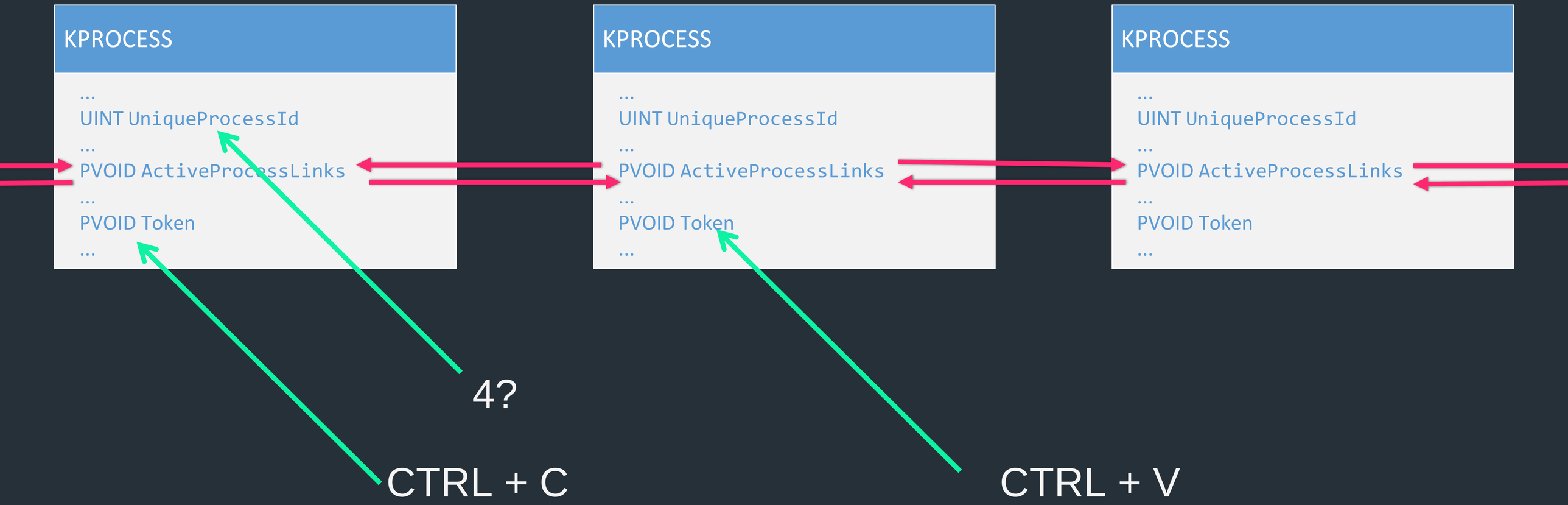
# Exploitation – Privesc



MWR LABS

++

# Exploitation – Privesc



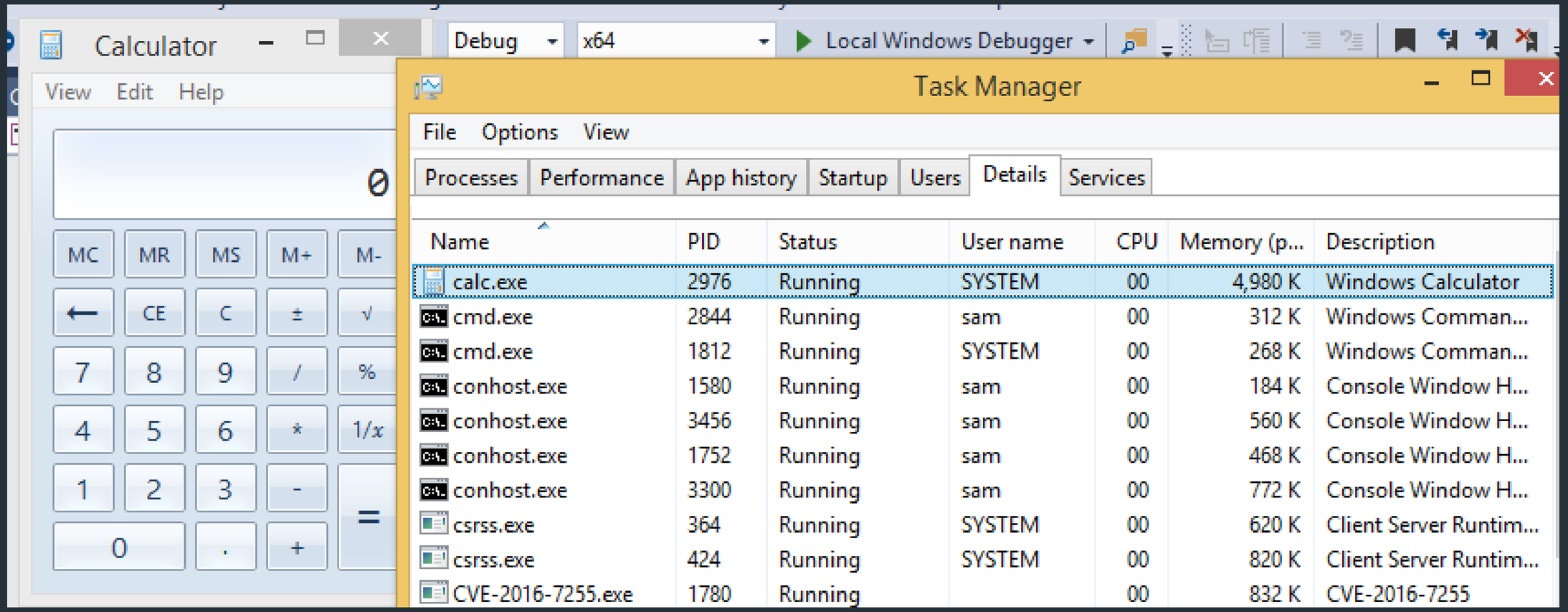
++

# Exploitation – Privesc

MWR LABS

++

# Exploitation – Privesc



++

# Caveats

Hardening Windows 10 with zero-day exploit mitigations

https://blogs.technet.microsoft.com/mmpc/2017/01/13/hardening-windows-10-with-zero-day-exploit-mitigations/

January 30, 2017

Hardening Windows 10 With Zero Day Exploit Mitigations Under The Microscope

https://improsec.com/blog//hardening-windows-10-with-zero-day-exploit-mitigations-under-the-microscope

++

# Conclusions

+ Windows kernel has a massive complex attack surface
+ Exploit development rapidly becoming harder
+ Not going away anytime soon

MWR Labs

MWR LABS

++

# Questions?

++

# References


1. https://medium.com/@mxatone/how-bad-design-decisions-created-the-least-secure-driver-on-windows-33e662a502fe#.a527m4bvt
2. https://googleprojectzero.blogspot.co.uk/2016/06/a-year-of-windows-kernel-font-fuzzing-1_27.html
3. https://labs.nettitude.com/blog/analysing-the-null-securitydescriptor-kernel-exploitation-mitigation-in-the-latest-windows-10-v1607-build-14393/
4. https://blogs.technet.microsoft.com/mmpc/2017/01/13/hardening-windows-10-with-zero-day-exploit-mitigations/
5. http://blog.rewolf.pl/blog/?p=1630
6. http://rol.im/asux/
7. https://whitehatters.academy/diffing-with-kam1n0/
8. https://msdn.microsoft.com/en-us/library/windows/desktop/hh871472(v=vs.85).aspx
9. https://msdn.microsoft.com/en-us/library/windows/desktop/mt706244(v=vs.85).aspx