The Adventures of a KeyStroke

An in-depth look into Keyloggers on Windows

Emre TINAZTEPE

What you will learn?

- Completing this training, you will be able to:
  - Use a kernel debugger for malware analysis,
  - Understand the threats posed by keyloggers,
  - Detect / Remove all kinds of keyloggers,
  - Understand how a keylogger works in greatest detail,
  - Be prepared to Advanced Persistent Threats!
- We will cover a lot of OS Internal structures.
- Without dealing with OS Internals, you can’t be sure that your system is clean.

Who am I?

- Emre TINAZTEPE
- Ex military:
  - Maltepe Military High School (21 / 421)
  - Turkish War Academy (8 / 838)
  - Passed half of his life in the army (First Lieutenant)
  - Resigned 3 years ago.
- Low level guy who likes to deal with OS Internals
- Currently leading a Malware Analysis Team
- Responsible of malware analysis and mobile av dev.

Methodology

- Hard to easy because it all starts at hardware ☹
- If you have question, just interrupt me.
- Hands on labs combined with theory.
  - Labs are made in a Win 7 x32 machine.

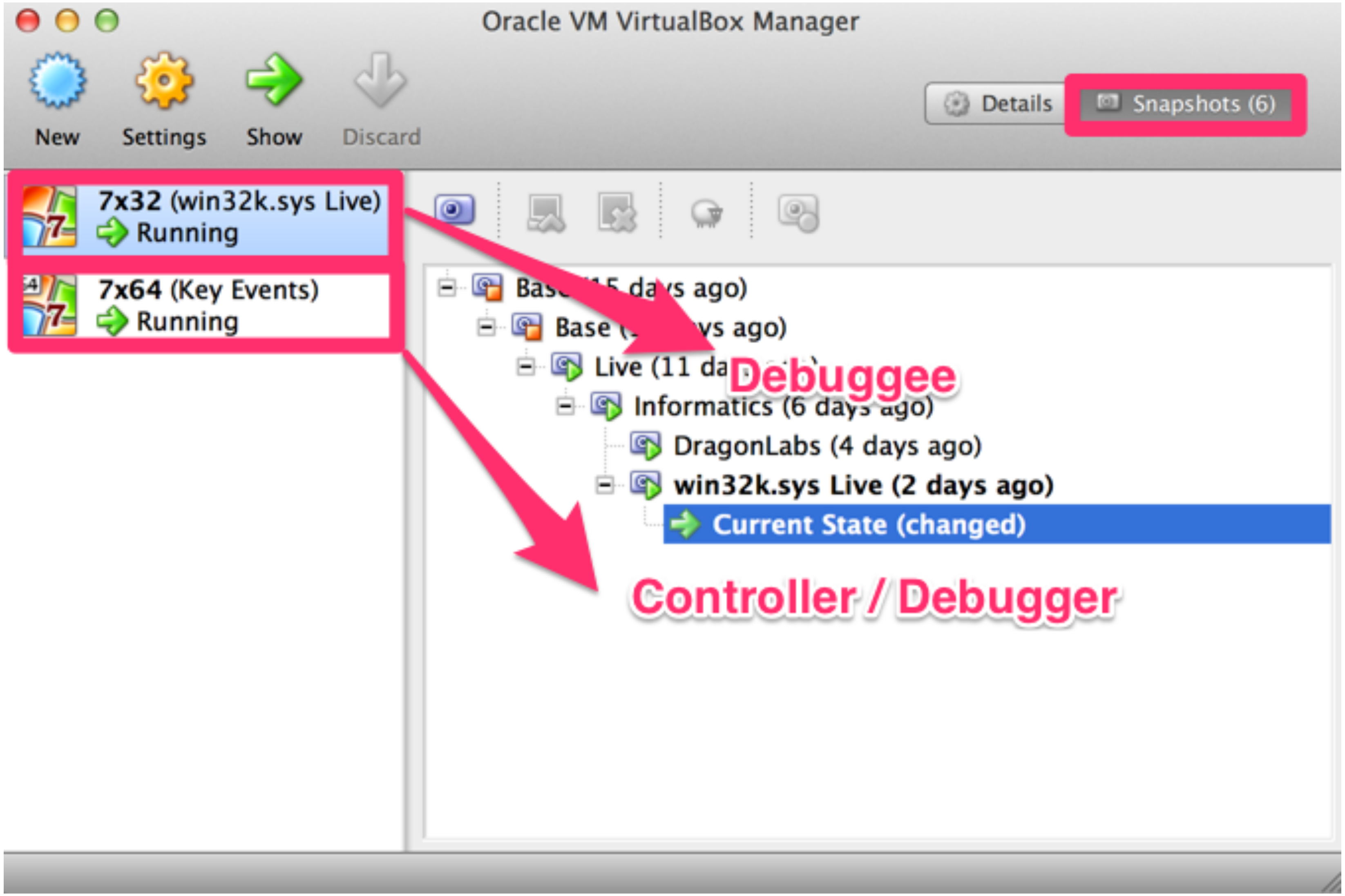
Why keyloggers?

- Because keyboard is the device you command your computers.
- Logging keys from a PC provides the malware authors with great power.
- Best way to gather intelligence.
  - Russia is said to be switching to “typing machines” in critical institutions.
- Best way to get rich ☺

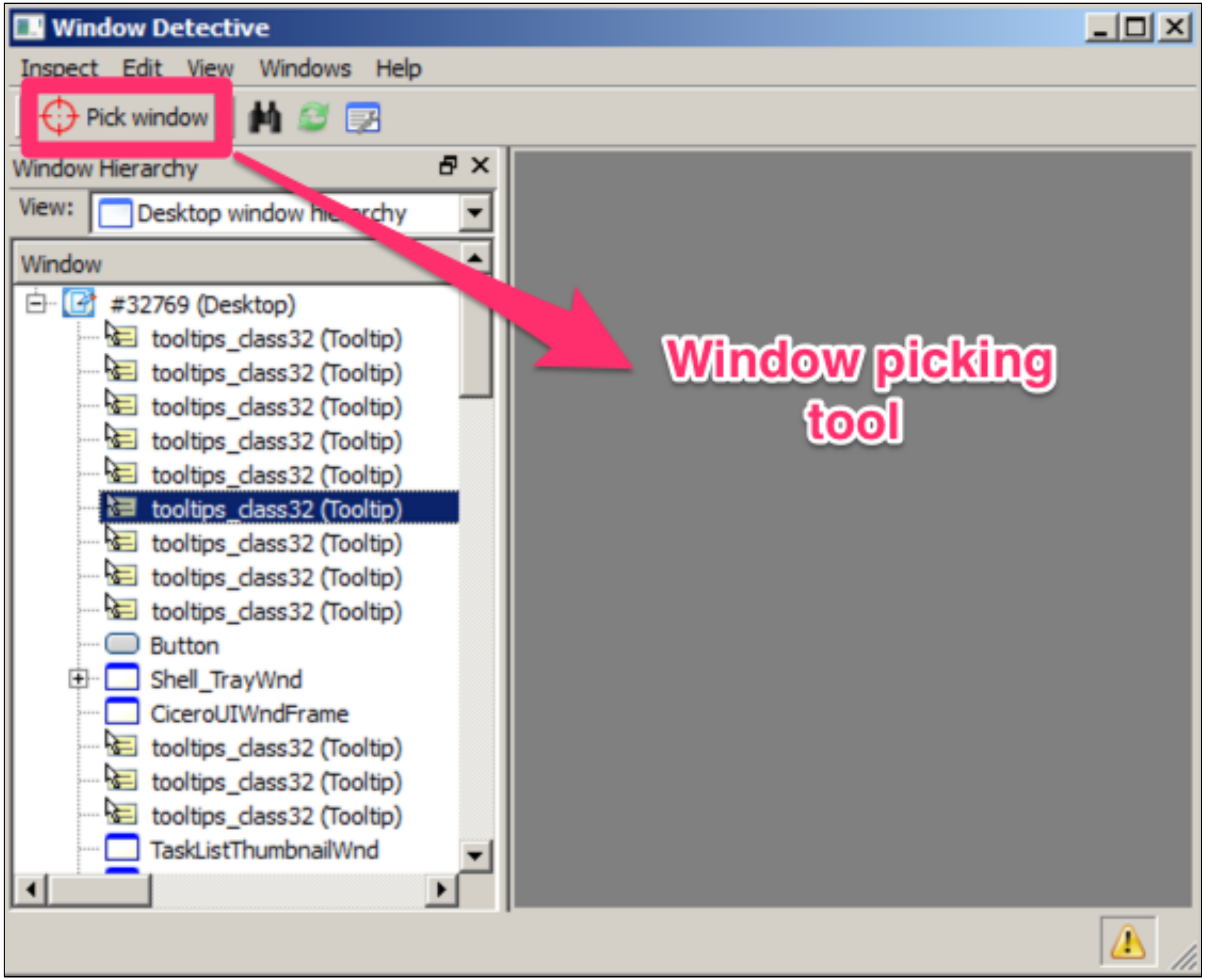
Before we begin

- Please download these files:
  - **Materials: http://bit.ly/1aLVnOI (pass: infected)**
  - **Labs: http://bit.ly/16FZ73t**
- **Please turn your AV/Windows Defender OFF!**

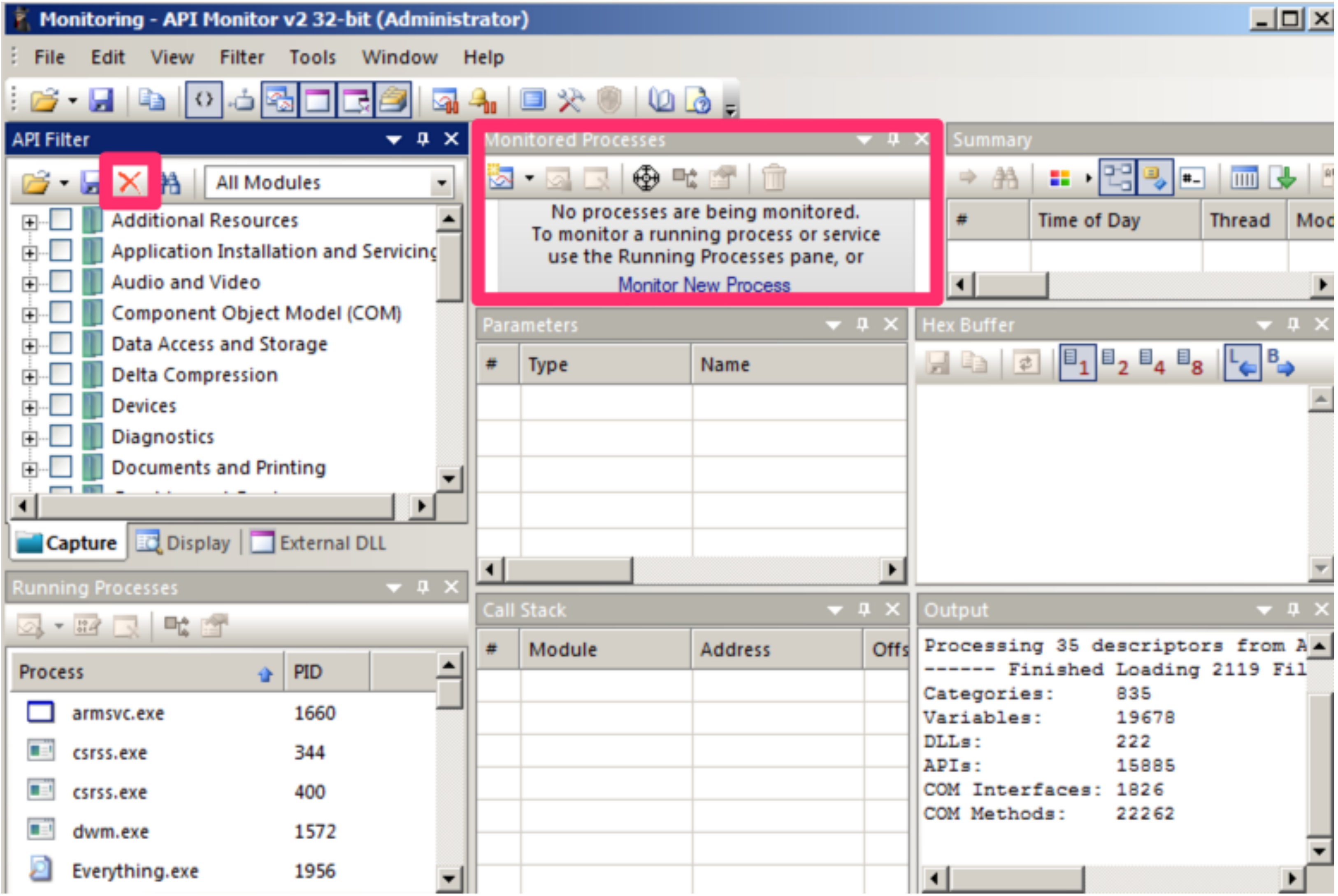
VirtualBox


Oracle VM VirtualBox Manager
New
Settings
Show
Discard
Details
Snapshots (6)
7x32 (win32k.sys Live)
Running
7x64 (Key Events)
Running
Informatics (6 days ago)
DragonLabs (4 days ago)
win32k.sys Live (2 days ago)
Current State (changed)
Debuggee
Controller / Debugger

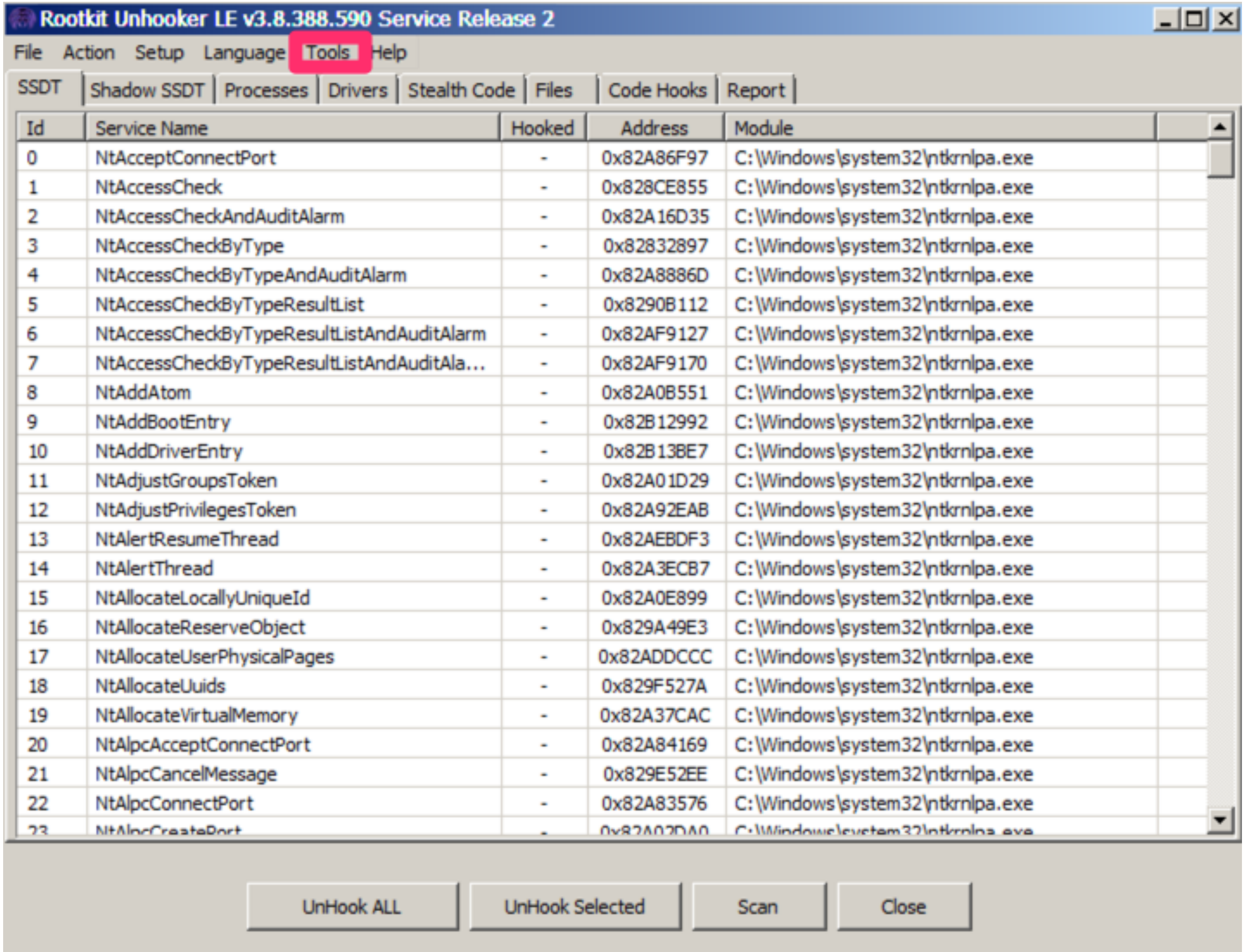
Window Detective



API Monitor

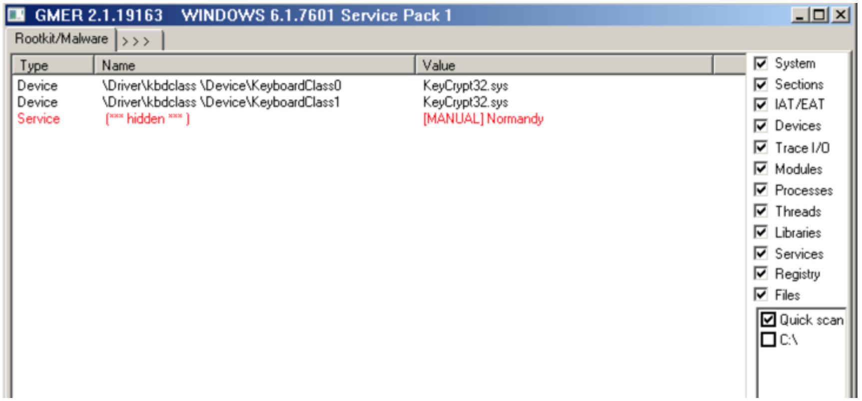

Monitoring - API Monitor v2 32-bit (Administrator)
File Edit View Filter Tools Window Help
API Filter
All Modules
Additional Resources
Application Installation and Servicing
Audio and Video
Component Object Model (COM)
Data Access and Storage
Delta Compression
Devices
Diagnostics
Documents and Printing
Capture
Display
External DLL
Monitored Processes
No processes are being monitored.
To monitor a running process or service
use the Running Processes pane, or
Monitor New Process
Summary
#
Time of Day
Thread
Parameters
#
Type
Name
Hex Buffer
Running Processes
Process
PID
armsvc.exe 1660
csrss.exe 344
csrss.exe 400
dwm.exe 1572
Everything.exe 1956
Call Stack
#
Module
Address
Output
Processing 35 descriptors from A
------ Finished Loading 2119 Fil
Categories: 835
Variables: 19678
DLLs: 222
APIs: 15885
COM Interfaces: 1826
COM Methods: 22262

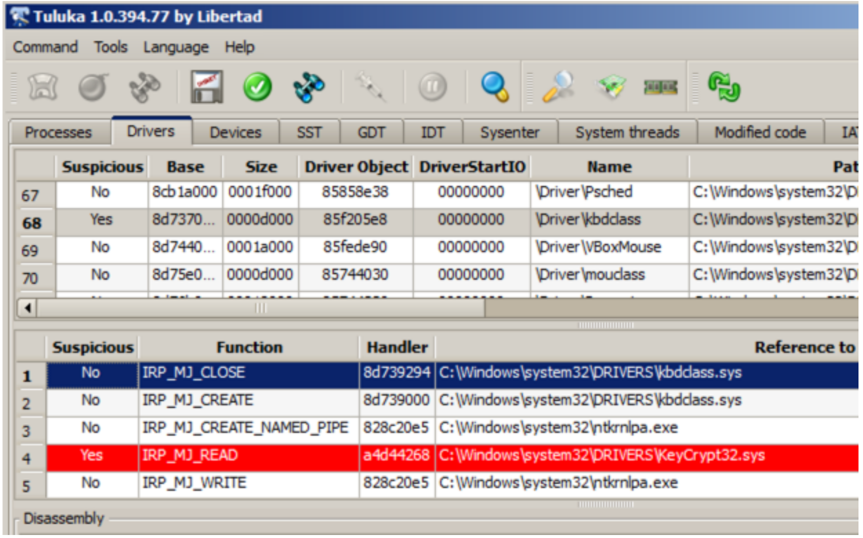
Rootkit Unhooker



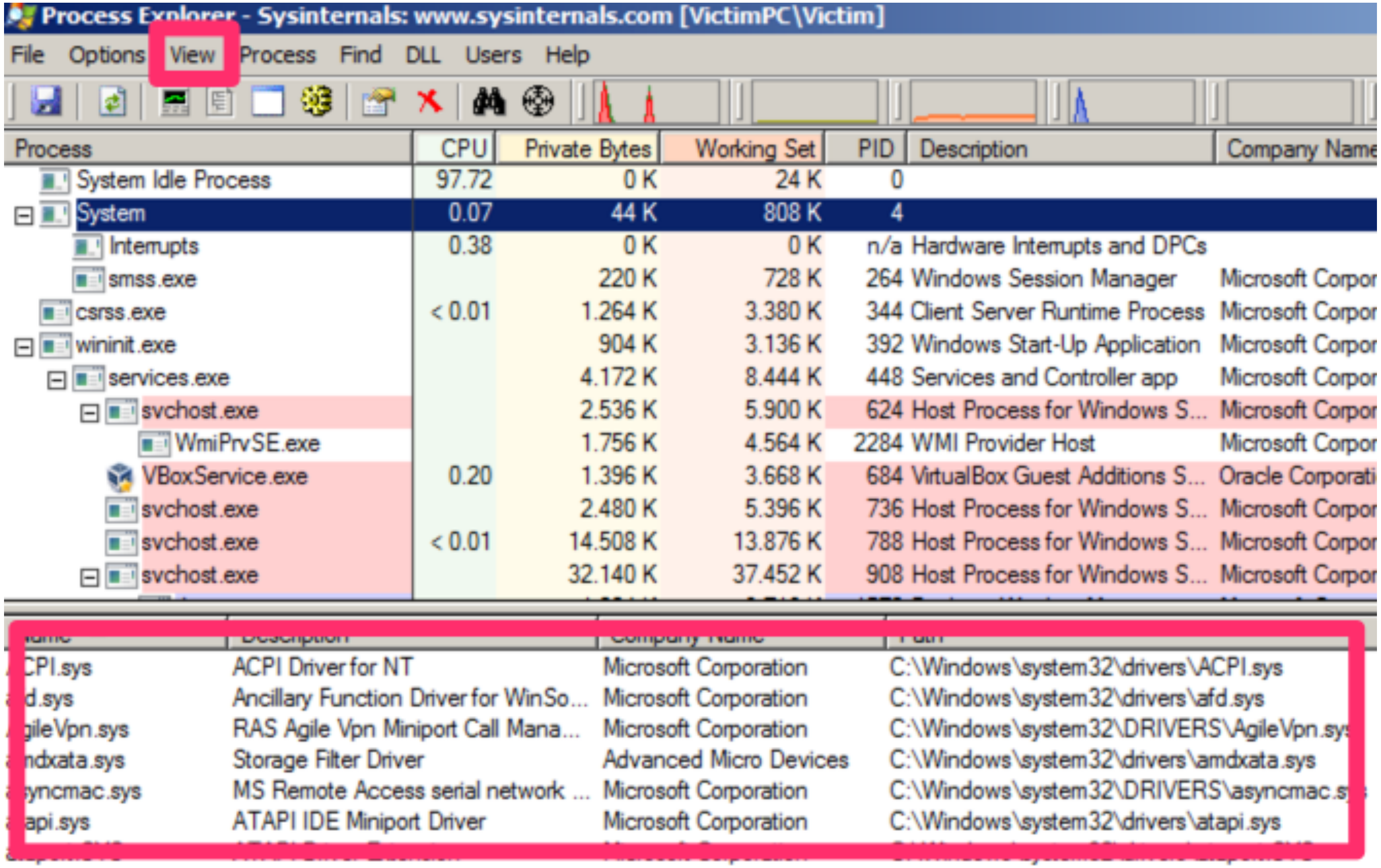
Id	Service Name	Hooked	Address	Module
0	NtAcceptConnectPort	-	0x82A86F97	C:\Windows\system32\ntkrnlpa.exe
1	NtAccessCheck	-	0x828CE855	C:\Windows\system32\ntkrnlpa.exe
2	NtAccessCheckAndAuditAlarm	-	0x82A16D35	C:\Windows\system32\ntkrnlpa.exe
3	NtAccessCheckByType	-	0x82832897	C:\Windows\system32\ntkrnlpa.exe
4	NtAccessCheckByTypeAndAuditAlarm	-	0x82A8886D	C:\Windows\system32\ntkrnlpa.exe
5	NtAccessCheckByTypeResultList	-	0x8290B112	C:\Windows\system32\ntkrnlpa.exe
6	NtAccessCheckByTypeResultListAndAuditAlarm	-	0x82AF9127	C:\Windows\system32\ntkrnlpa.exe
7	NtAccessCheckByTypeResultListAndAuditAla...	-	0x82AF9170	C:\Windows\system32\ntkrnlpa.exe
8	NtAddAtom	-	0x82A0B551	C:\Windows\system32\ntkrnlpa.exe
9	NtAddBootEntry	-	0x82B12992	C:\Windows\system32\ntkrnlpa.exe
10	NtAddDriverEntry	-	0x82B13BE7	C:\Windows\system32\ntkrnlpa.exe
11	NtAdjustGroupsToken	-	0x82A01D29	C:\Windows\system32\ntkrnlpa.exe
12	NtAdjustPrivilegesToken	-	0x82A92EAB	C:\Windows\system32\ntkrnlpa.exe
13	NtAlertResumeThread	-	0x82AEBDF3	C:\Windows\system32\ntkrnlpa.exe
14	NtAlertThread	-	0x82A3ECB7	C:\Windows\system32\ntkrnlpa.exe
15	NtAllocateLocallyUniqueId	-	0x82A0E899	C:\Windows\system32\ntkrnlpa.exe
16	NtAllocateReserveObject	-	0x829A49E3	C:\Windows\system32\ntkrnlpa.exe
17	NtAllocateUserPhysicalPages	-	0x82ADDCCC	C:\Windows\system32\ntkrnlpa.exe
18	NtAllocateUuids	-	0x829F527A	C:\Windows\system32\ntkrnlpa.exe
19	NtAllocateVirtualMemory	-	0x82A37CAC	C:\Windows\system32\ntkrnlpa.exe
20	NtAlpcAcceptConnectPort	-	0x82A84169	C:\Windows\system32\ntkrnlpa.exe
21	NtAlpcCancelMessage	-	0x829E52EE	C:\Windows\system32\ntkrnlpa.exe
22	NtAlpcConnectPort	-	0x82A83576	C:\Windows\system32\ntkrnlpa.exe

GMER / Tuluka

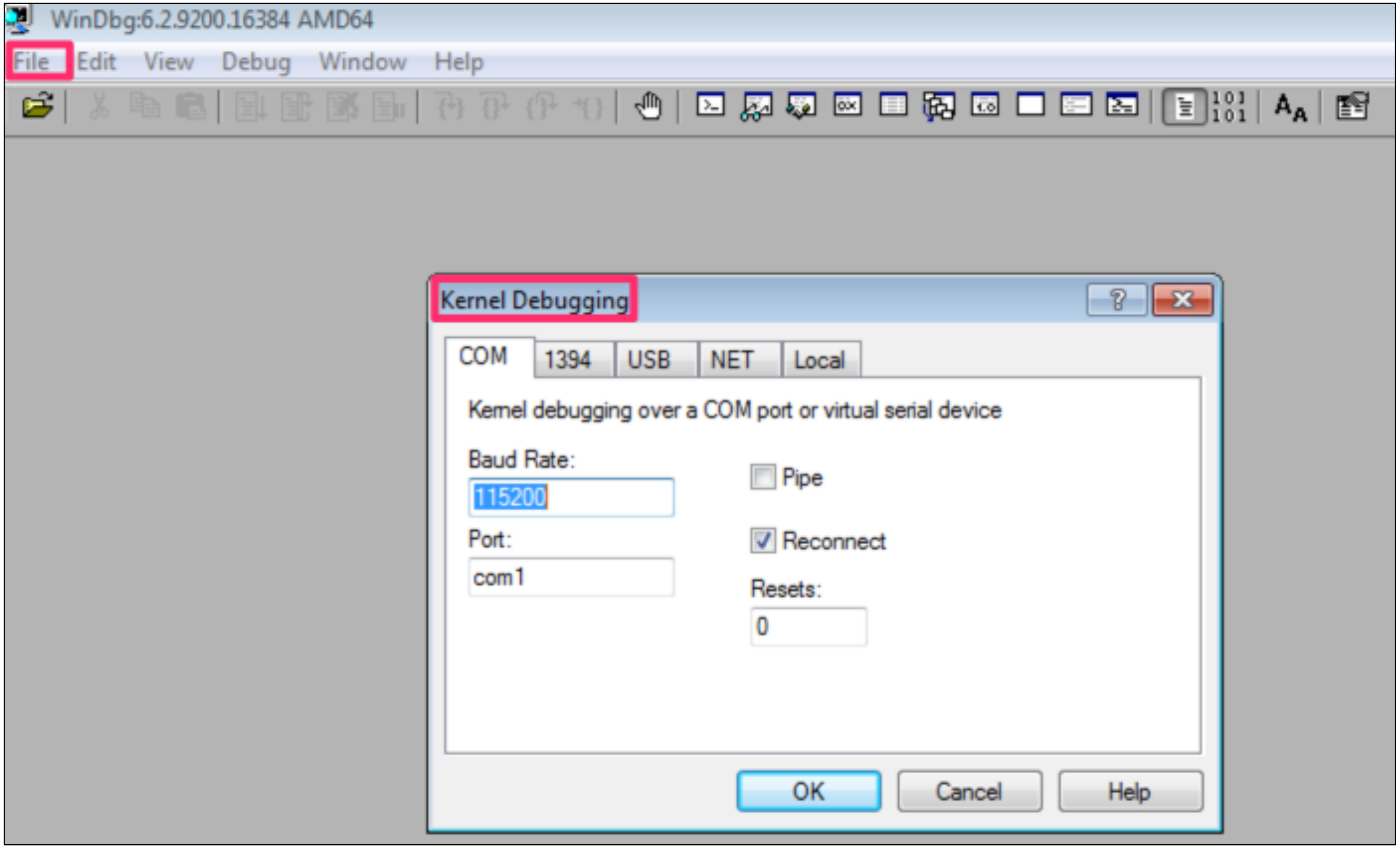

GMER 2.1.19163 WINDOWS 6.1.7601 Service Pack 1
Rootkit/Malware
Type
Name
Value
Device
\Driver\kbdclass \Device\KeyboardClass0
KeyCrypt32.sys
Device
\Driver\kbdclass \Device\KeyboardClass1
KeyCrypt32.sys
Service
(*** hidden ***)
[MANUAL] Normandy
System
Sections
IAT/EAT
Devices
Trace I/O
Modules
Processes
Threads
Libraries
Services
Registry
Files
Quick scan
C:\


Tuluka 1.0.394.77 by Libertad
Command Tools Language Help
Processes
Drivers
Devices
SST
GDT
IDT
Sysenter
System threads
Modified code
Suspicious
Base
Size
Driver Object
DriverStartIO
Name
67 No 8cb1a000 0001f000 85858e38 00000000 \Driver\Psched C:\Windows\system32\D
68 Yes 8d7370... 0000d000 85f205e8 00000000 \Driver\kbdclass C:\Windows\system32\D
69 No 8d7440... 0001a000 85fede90 00000000 \Driver\VBoxMouse C:\Windows\system32\D
70 No 8d75e0... 0000d000 85744030 00000000 \Driver\mouclass C:\Windows\system32\D
Suspicious
Function
Handler
Reference to
1 No IRP_MJ_CLOSE 8d739294 C:\Windows\system32\DRIVERS\kbdclass.sys
2 No IRP_MJ_CREATE 8d739000 C:\Windows\system32\DRIVERS\kbdclass.sys
3 No IRP_MJ_CREATE_NAMED_PIPE 828c20e5 C:\Windows\system32\ntkrnlpa.exe
4 Yes IRP_MJ_READ a4d44268 C:\Windows\system32\DRIVERS\KeyCrypt32.sys
5 No IRP_MJ_WRITE 828c20e5 C:\Windows\system32\ntkrnlpa.exe
Disassembly

Process Explorer



Windbg


WinDbg:6.2.9200.16384 AMD64
File Edit View Debug Window Help
Kernel Debugging
COM
1394
USB
NET
Local
Kernel debugging over a COM port or virtual serial device
Baud Rate:
115200
Port:
com1
Pipe
Reconnect
Resets:
0
OK
Cancel
Help

Windbg Cheat Sheet

- lm : Lists loaded modules (drivers , dlls)
- !process -1 0 : Displays current process
- !process 0 0 winlogon.exe : Displays info for the process
- .process EPROCESS : Switches to the process (implicit)
- bp ADDRESS : Puts a breakpoint at the address
- g,p,t : Go, Step, Trace
- bl : Lists the breakpoints
- bc INDEX : Clears the BP indicated by the index
- bd INDEX : Disables BP temporarily
- .echo : Outputs a string

Windbg Cheat Sheet

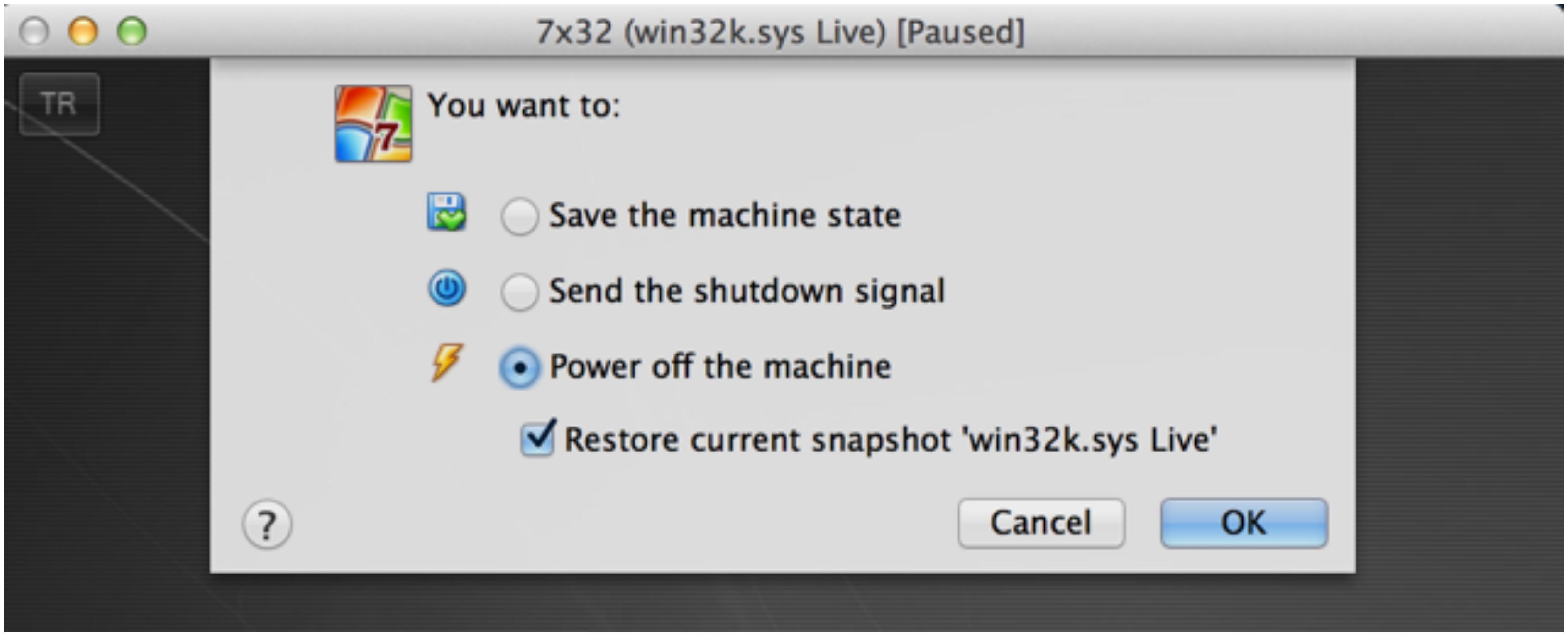
- .cls : Clears the screen
- u ADDRESS / SYMBOL : Unassembles the address
- uf ADRESS OF FUNCTION : Unassembles the whole func.
- db ADDRESS : Dumps the address.
- ? poi(ADDRESS) : Displays the address pointed by.
- !pic / !ioapic : Displays information about interrupt controllers.
- !drvobj \Driver\kbdclass 0x7: Display the specified driver.
- !devobj OBJECT : Display information about device obj.

Let’s infect ourselves

- Restart RED VM, make sure it is not in “KERNEL DEBUG” mode.
- Go to Materials/Keyloggers directory
- Double click “Elite Keylogger.exe”
- Install with default settings (Click NEXT multiple times)
- Choose “Allow” in case Windows Defender consents.
- Restart the VM in non debug mode.
- Write “unhide” on start menu and provide a password at least 3 chars long.
- Fire up a “Notepad” and write your name in it.
- Please also provide your Credit Card number ☺☺☺
- Do not save it please, it is safer ???

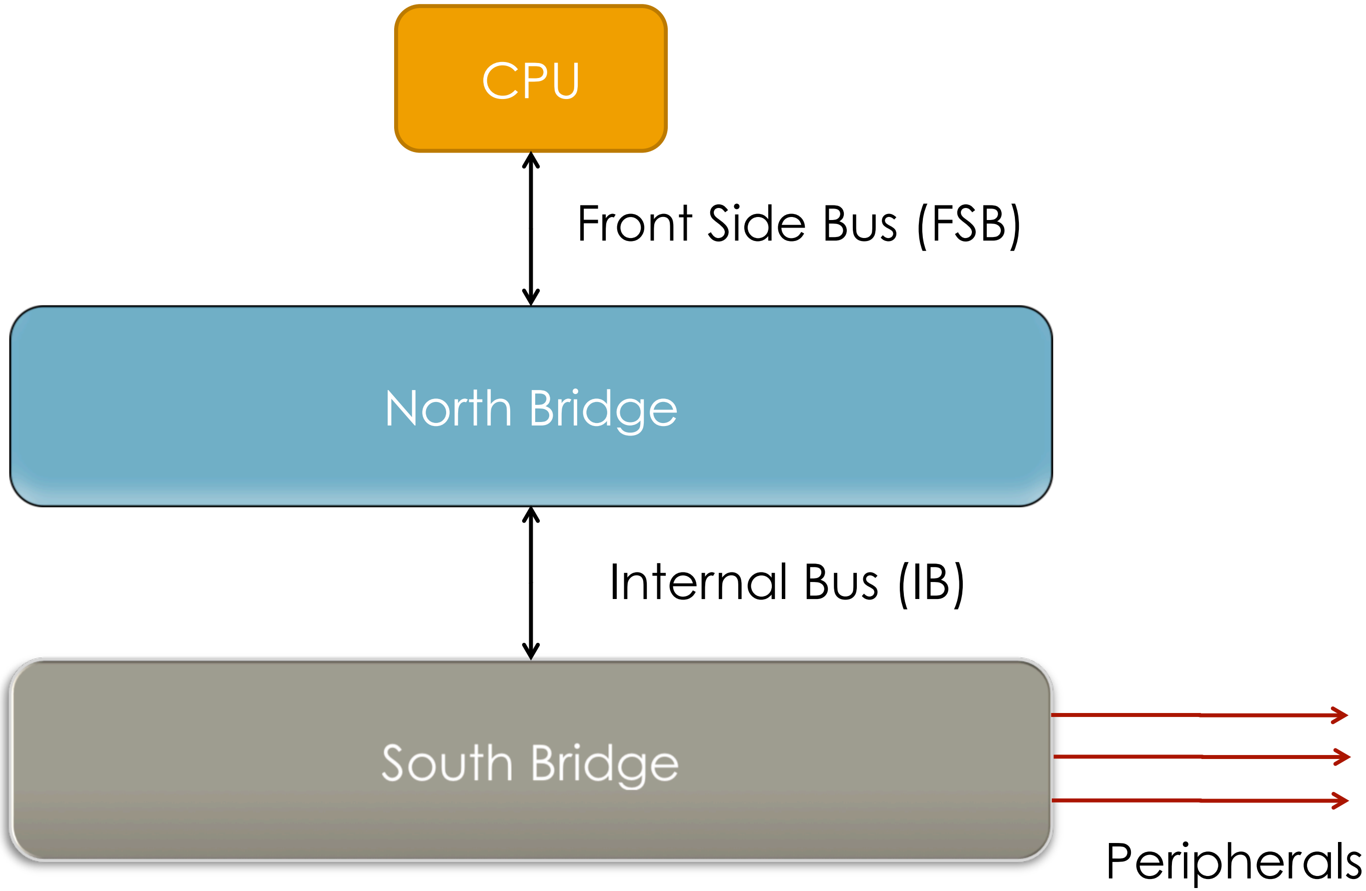
You are infected now ☹

- We will see how to detect keyloggers in the following ours.
- For the moment, please restore your VM to snapshot "Informatics" and start your VM in "Kernel Debugging" Mode.



Ready to dive?

An overview of a mother board



An overview of a mother board

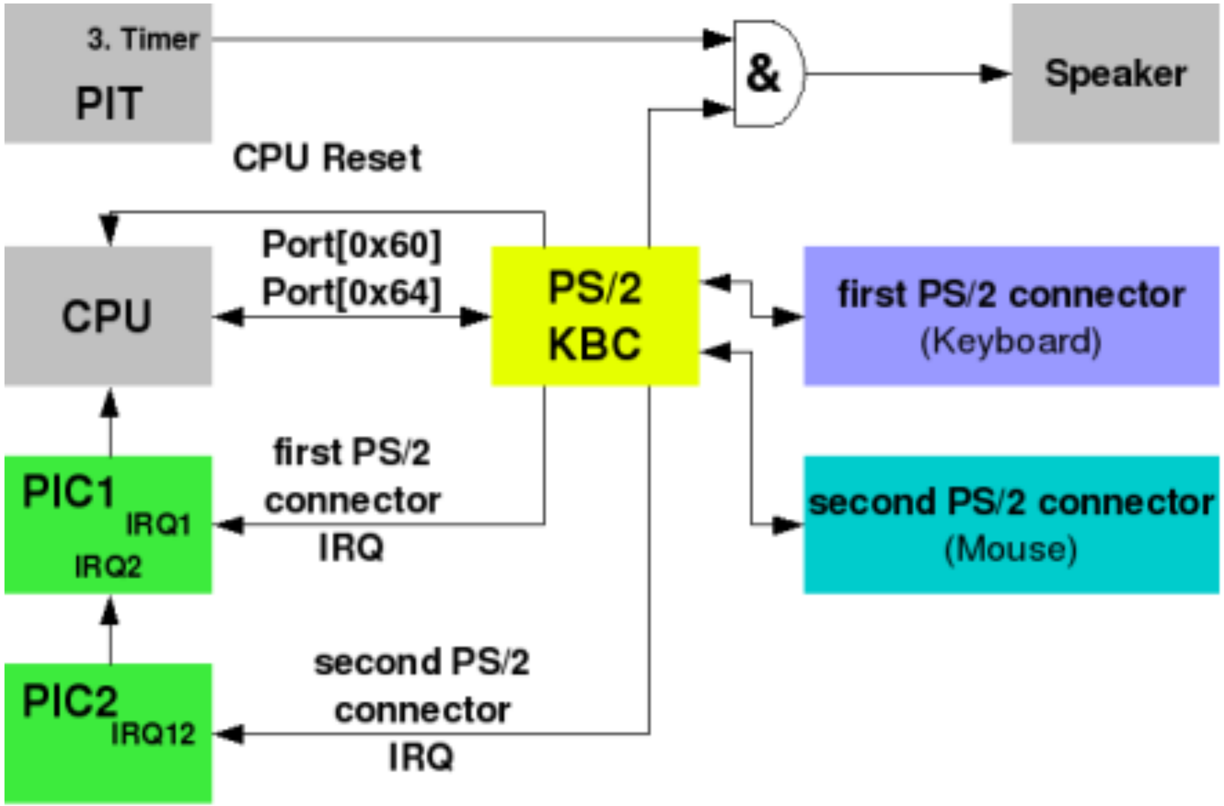
- Bus is a communication system that transfers data between components inside a computer,
- FSB is the CPU's connection to the North Bridge and through it to rest of the system,
- North Bridge is a high-speed hub that in most systems connects the CPU to the graphics card and to RAM,
- South Bridge is a slower-speed hub that connects the CPU to the rest of the system.

South Bridge (SB)

- It is also named as “Input/Output Controller Hub”.
- Responsible from the peripheral device connections such as USB, PCI, PS/2, Sound and etc.
- Why two bridges?
  - Same as the idea of having RAM, Cache, Register
  - Simpler design which is easy to modify in terms of IO capabilities.
- It is what you actually connect your keyboard to.

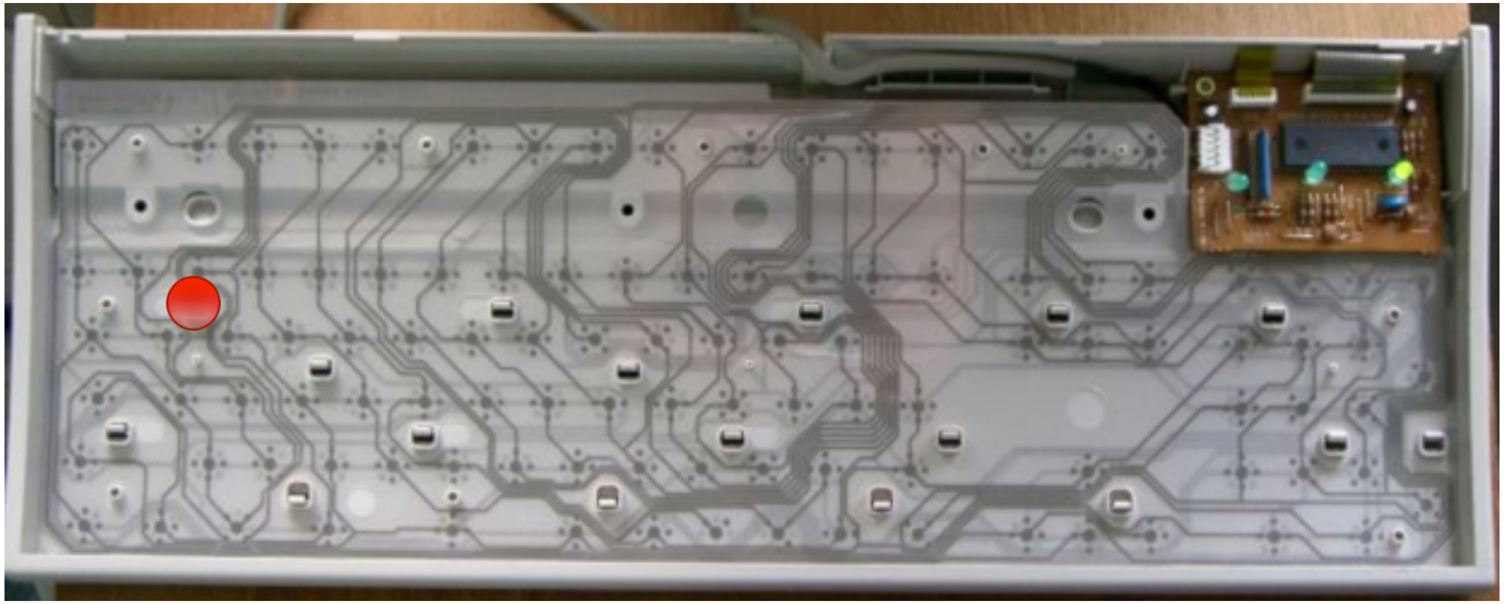
PS/2 Keyboard Controller

- A component of a mainboard which handles the connection between a motherboard and a PS/2 keyboard.



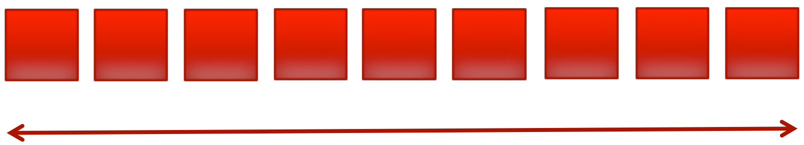
PS/2 Keyboard

- Just a limited computer system which scans a wireframe continuously for finding a closed/opened circuit.



PS/2 Keyboard

- The PS/2 Keyboard is a device that talks to a PS/2 controller using serial communication.
- The PS/2 Keyboard accepts commands and sends responses to those commands, and also sends **scancodes** indicating when a key was pressed or released.
- The keyboards processor includes its own timer, 33 instruction set, *and can even access 128K of external memory.*

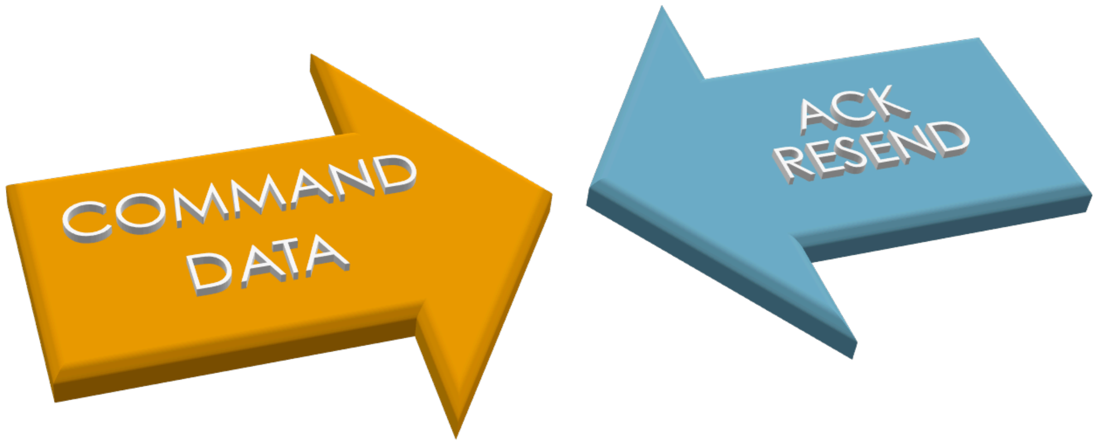


16 Byte Buffer

Talking to a Keyboard?

- A PS/2 Keyboard accepts many types of commands,
- Each command is one byte,
- Some commands have data byte/s which must be sent after the command byte,
- The keyboard typically responds to a command by sending either an "ACK" (to acknowledge the command) or a "Resend" (to say something was wrong with the previous command) back.

Talking to a Keyboard?



- Commands must be sent one at a time (IN/OUT),
- Some commands have data byte/s which must be sent after the command byte,
- 0xFE (resend) expects a command to be sent again,
  while 0xFA (ACK) means command is successfully processed.

PS/2 Keyboard Controller/Encoder Ports

IO Port	Access Type	Purpose
	Keyboard Encoder	
0x60	Read	Read Input Buffer
0x60	Write	Send Command
	Keyboard Controller	
0x64	Read	Status Register
0x64	Write	Send Command

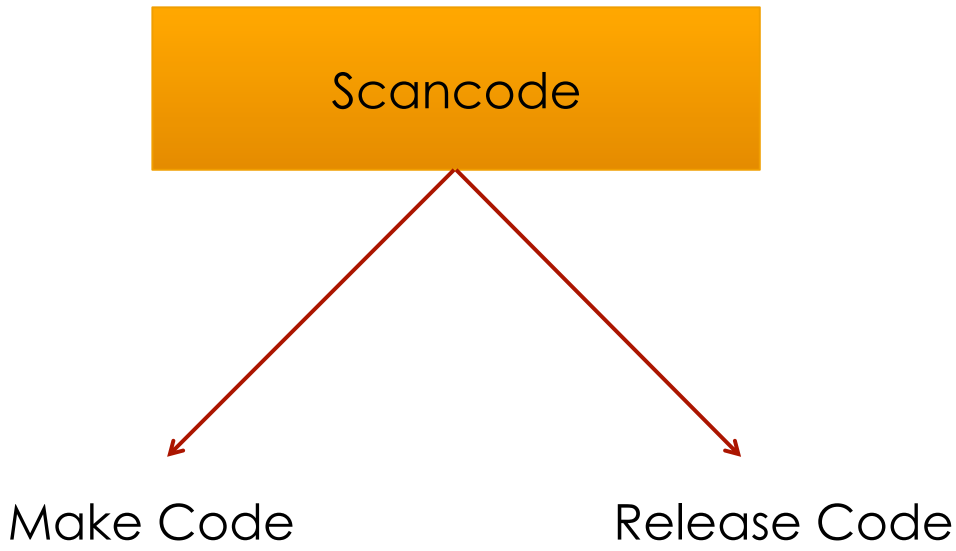
- Port 0x60 is what we use for reading and writing data to/from the keyboard device,
- The Status Register contains various flags that indicate the state of the PS/2 controller such as the state of input/output buffers,
- The Command Port (0x64) is used for sending commands to the PS/2 Controller (not to PS/2 Devices).

Some of the PS/2 Keyboard Encoder Commands

Command	Description	Data
0xED	Set LEDs	Bit0: ScrollLock Bit1: NumberLock Bit2: CapsLock
0xEE	Echo	For diagnostic purposes.
0xF0	Get/set current scan code set	0: Get current scan code set 1: Set scan code set 1 2: Set scan code set 2 3: Set scan code set 3
0xF4	Enable scanning	-
0xF5	Disable scanning	Discard key presses or mouse movements. Used especially while identifying the attached PS/2 device in order to prevent messing up the identification process.

Scancodes and Code Sets

- A scan code set is a set of codes that determine when a key is pressed or repeated, or released.



Scancodes and Code Sets

- There are 3 scan code sets, normally on PC compatible systems the keyboard itself uses scan code set 2 and the keyboard controller translates this into scan code set 1 for compatibility.

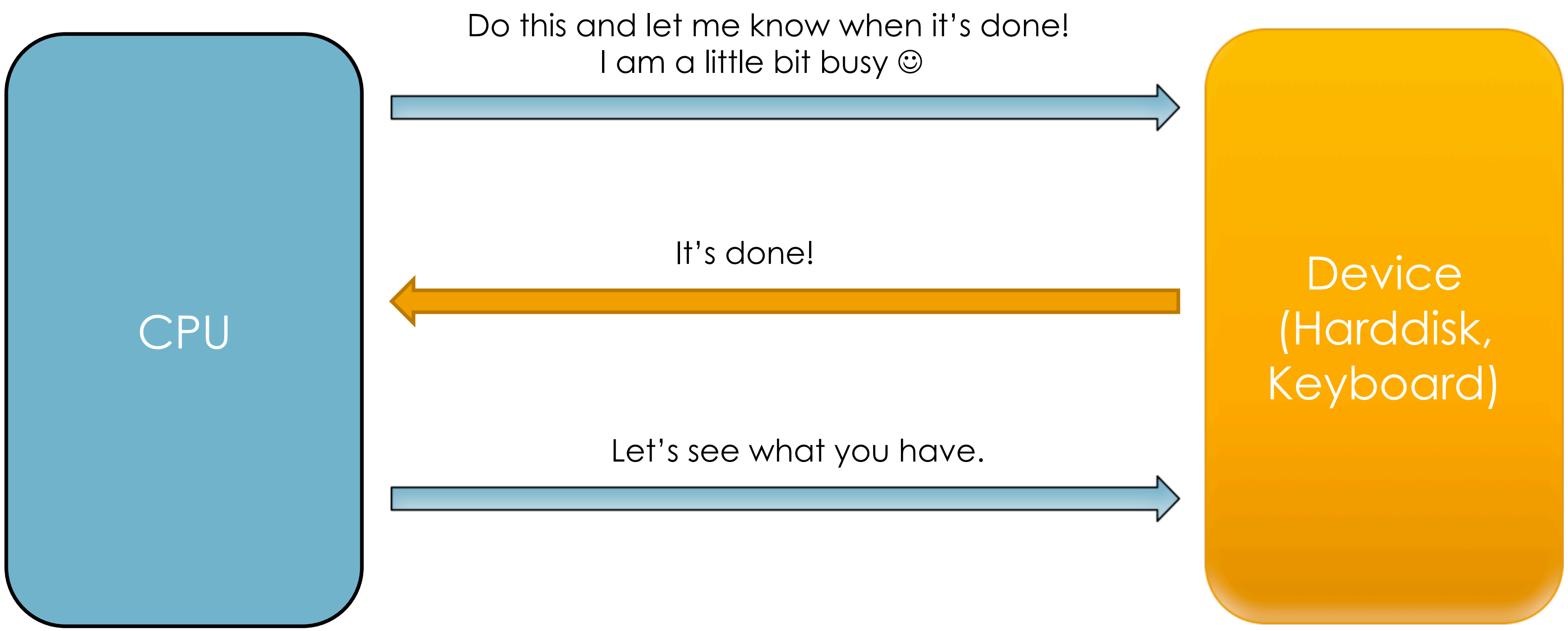
Microsoft Keyboard Scan Code Specification Document

How to read scancodes?

- Poll the Bit 0 of status register and then read the data from port 0x60
  - To much CPU time!
  - Multiple PS/2 devices lead to problems for differentiating the data.
- Wait for an interrupt to occur
  - Much better!
  - Wait for an IRQ 1 / IRQ 12 (wait for the next slide☺)

What is an interrupt?

- Interrupt is a signal to the processor emitted by hardware or software indicating an event that needs immediate attention.

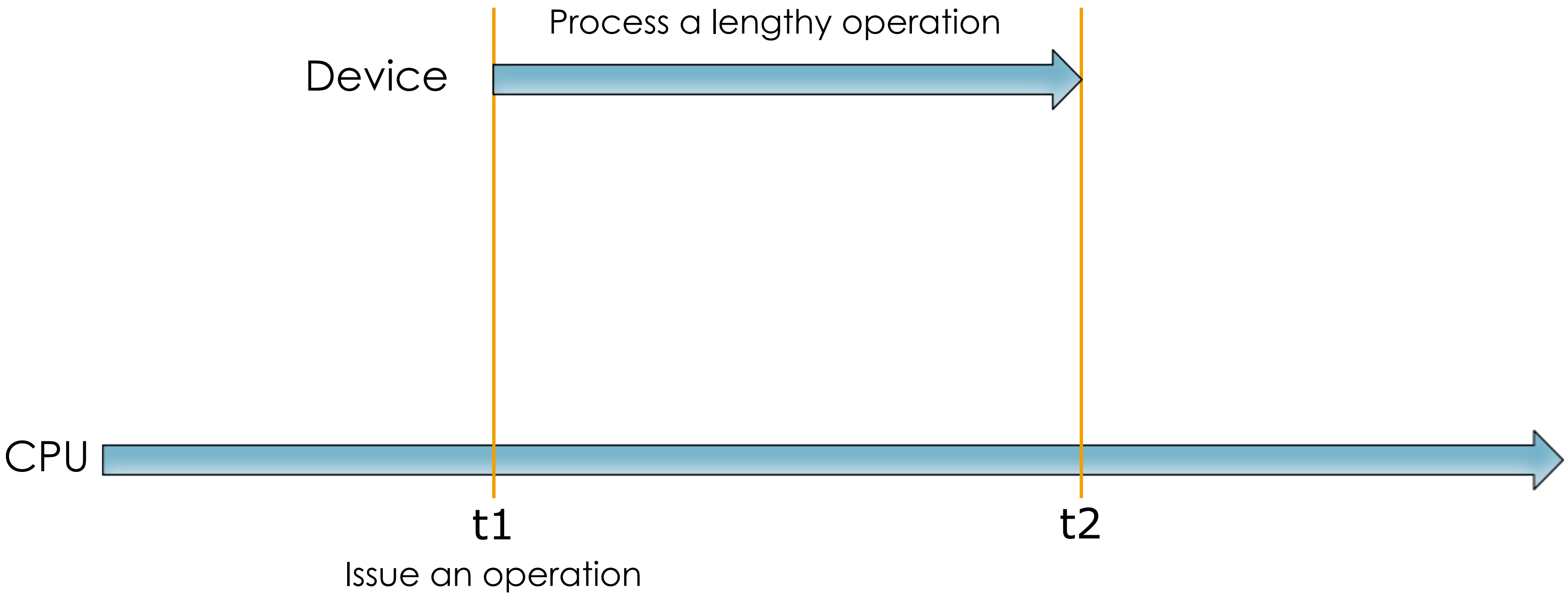


Why called as "IRQ"?

- Each peripheral device requests to "Interrupt the CPU" this is why it is a "Request" which may or may not be handled by the CPU.
- Question: What happens when multiple devices send an IRQ at the same time?
- Answer: The one with a higher IRQL gets processed while the others keep waiting.

Interrupt Handling

- One of the best advantages of an interrupt driven device is the ability to overlap device's processing time with the CPU's activity.



Where do I connect my device?

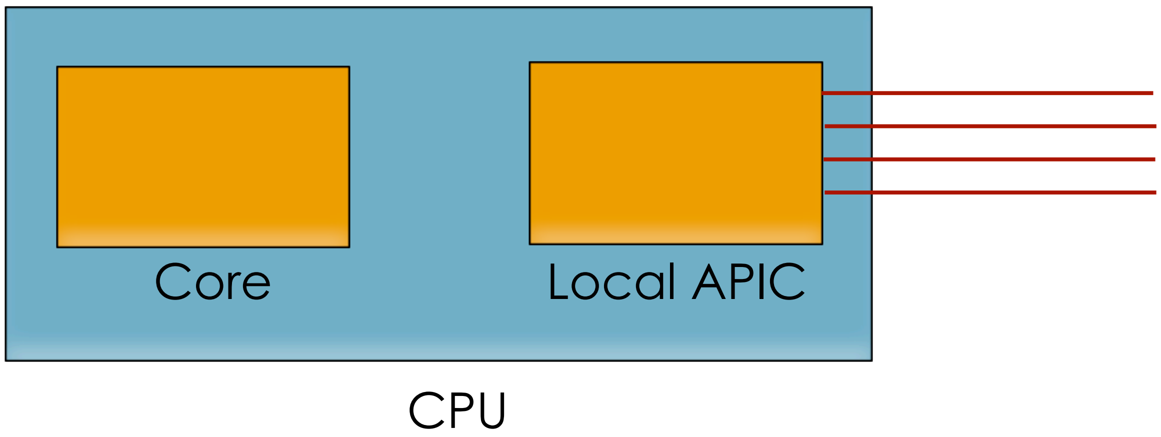
- Question: If we have 2 or more devices attached to our mainboard, how will we differentiate one device’s interrupt from the other?
- Answer: Each motherboard has an at least one Programmable Interrupt Controller (PIC / APIC) into which your external devices get connected. You do not have to do anything, all is done seamlessly by this electronic circuit.

Programmable Interrupt Controller

- OMG! What is an interrupt controller?
- One of the most important chips making up the x86 architecture,
- Without it, the x86 architecture would not be an interrupt driven architecture,
- The function of the PIC is to manage hardware interrupts and send them to the appropriate system interrupt.
- This way, no polling needed ☺

APIC

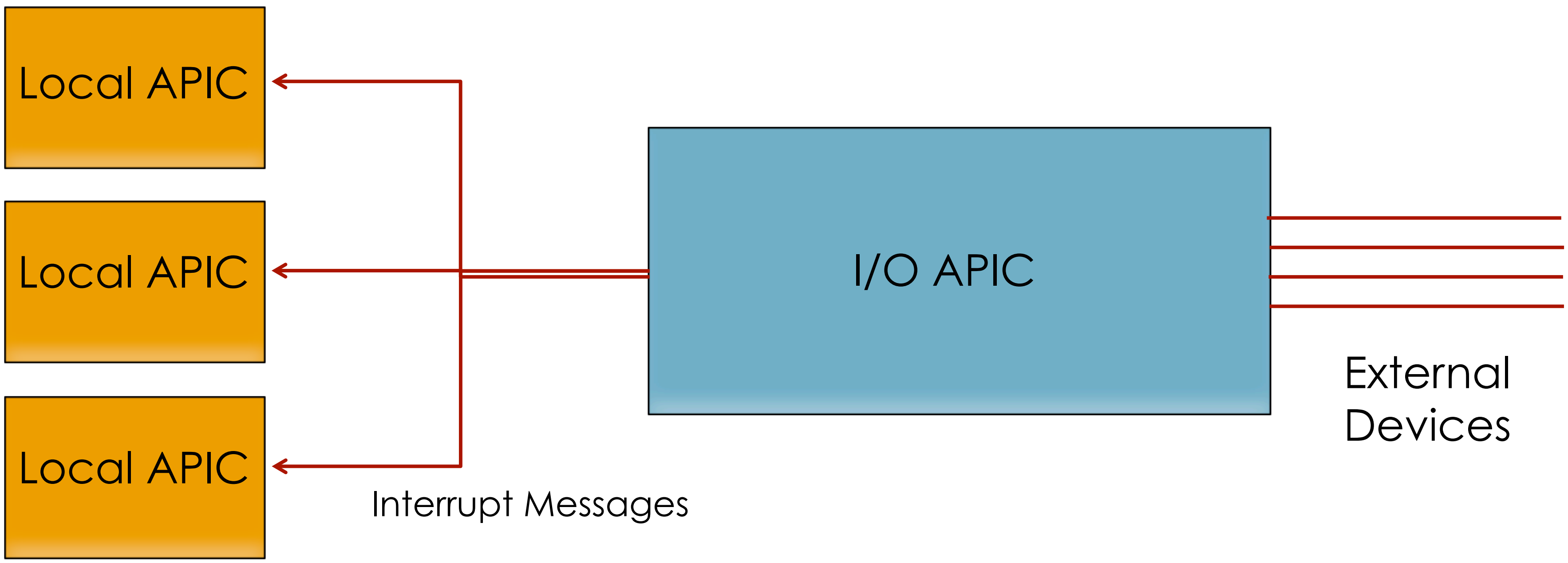
- More sophisticated interrupt handling and the ability to send interrupts between processors.
- In an APIC-based system, each CPU is made of a "core" and a "local APIC".



I/O APIC

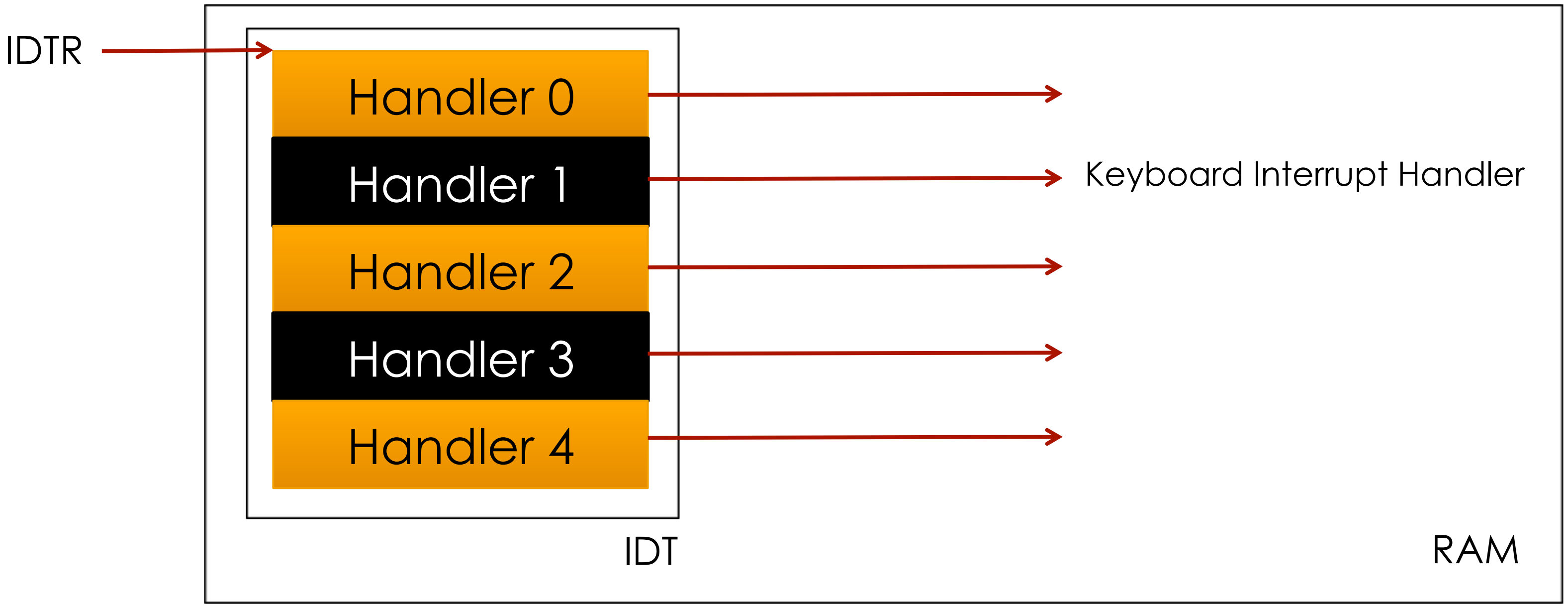
- The external I/O APIC is part of Intel's system chip set. Its primary function is to receive external interrupt events from the system and its associated I/O devices and relay them to the local APIC as interrupt messages.

- It is programmed by the OS before enabling interrupt handling mechanism.



What magic CPU does to handle IRQs?

- There is no magic, we tell it what to do.
- We create a table of function pointers and tell the CPU where it resides.
- This table is called as "Interrupt Descriptor Table" and the address for this table is hold by a register called IDTR (IDT register).



Intel x86 CPU Modes

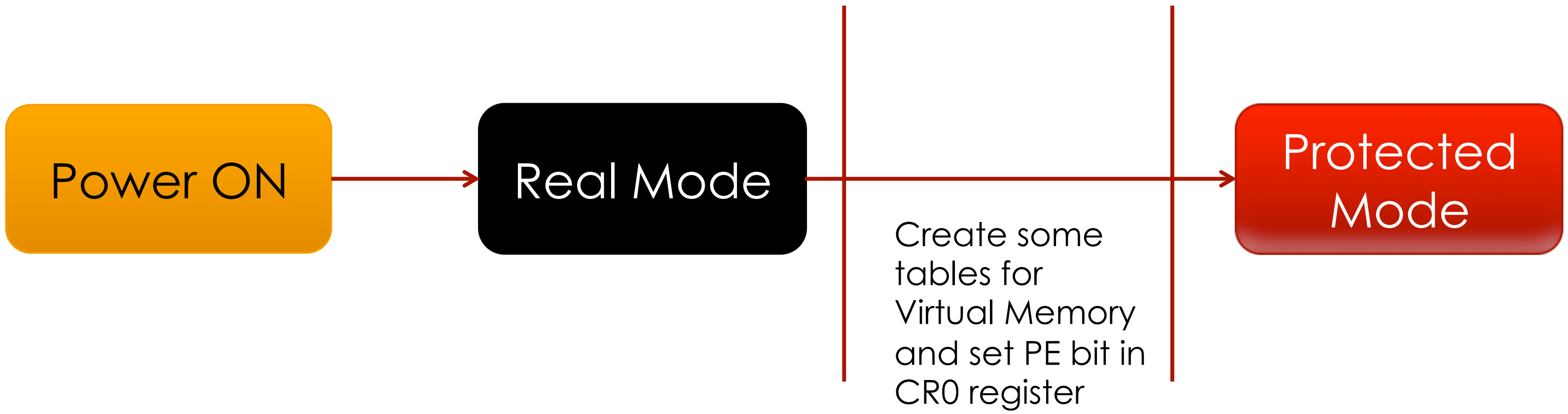
- 3 + 1 Modes of operation is supported by CPU.
  - Real Mode
  - Virtual 8086 Mode
  - Protected Mode
  - System Management Mode

Real Mode

- Also called **real address mode.**
- Real mode is characterized by a 20-bit segmented memory address space and unlimited direct software access to all memory, I/O addresses and peripheral hardware.
- Real mode provides no support for memory protection, multitasking, or code privilege levels.
- Before the release of the 80286, which introduced Protected mode, real mode was the only available mode for x86 CPUs.
- In the interests of backwards compatibility, ***all x86 CPUs start in real mode*** when reset.

Protected Mode

- Also called **protected virtual address mode.**
- It allows system software to use features such as virtual memory, paging and safe multi-tasking designed to increase an operating system's control over application software.



Virtual 8086 Mode

- Also called **virtual real mode.**
- Allows the execution of real mode applications that are incapable of running directly in protected mode while the processor is running a protected mode operating system.

System Management Mode

- Is an operating mode in which all normal execution (including the operating system) is suspended, and special separate software (usually firmware or a hardware-assisted debugger) is executed in high-privilege mode.

- SMM is a special-purpose operating mode provided for handling system-wide functions like:
  - Handle system events like memory or chipset errors,
  - Manage system safety functions, such as shutdown on high CPU temperature and turning the fans on and off,
  - Emulate motherboard hardware that is unimplemented or buggy.

More on SMM

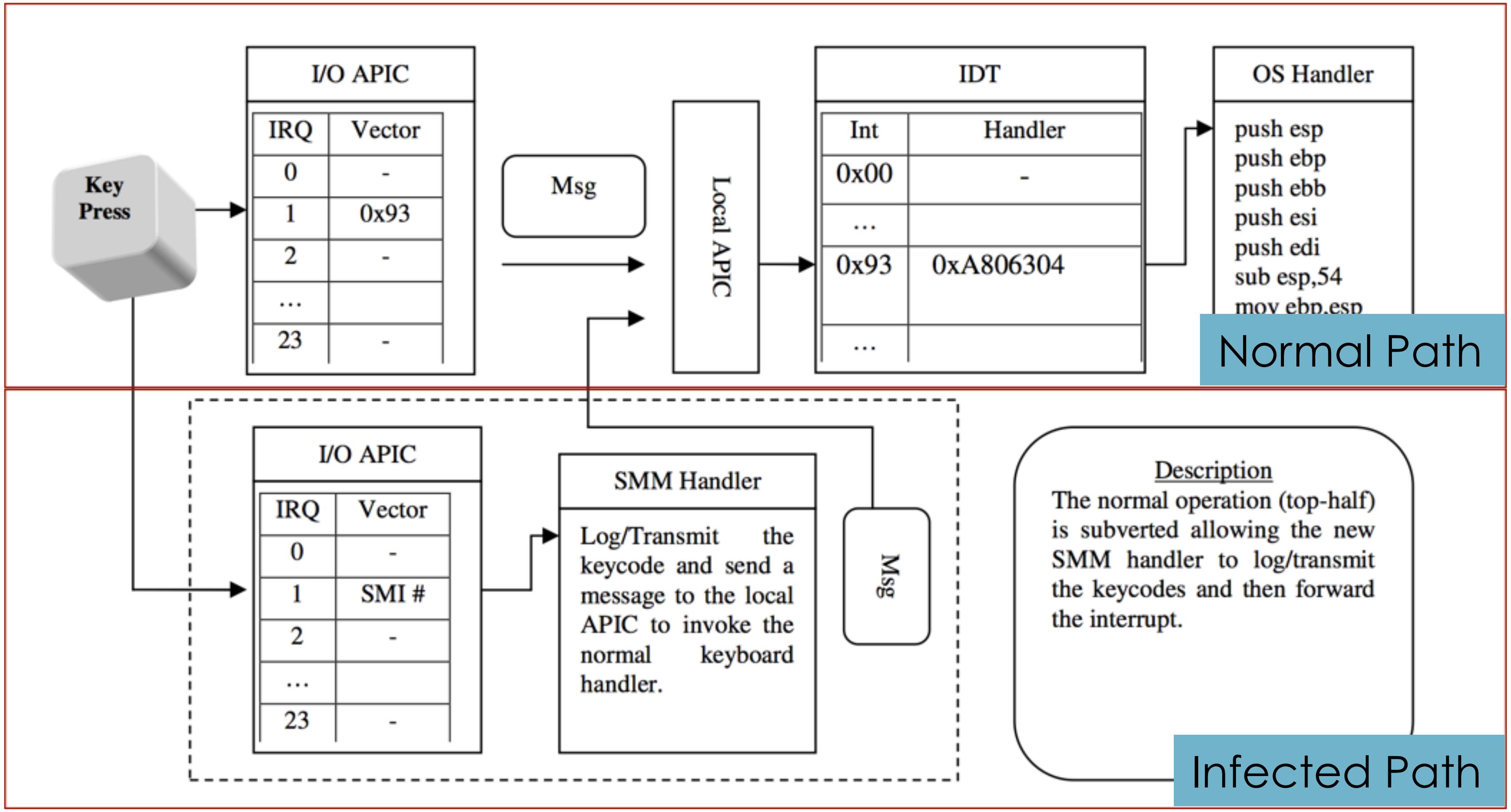
- A powerful mode of CPU which can even preempt the whole OS!!!
- SMM is entered via the SMI (system management interrupt)
- SMM is a really good place to execute malicious software **without modifying the structures created by OS.**

Here comes the karate kick!

#1 SMM Rootkits

An overview of SMM Rootkits

- Did you know that you can see the keystrokes even before they are handled by “Interrupt Handler”?



The implementation

1. Use SMRAM Control Register (SMRAMC)
   - Check bit D_OPEN (is SMRAM visible to outside code)
   - Check bit D_LCK (is SMRAMC is read-only, if yes a reset is needed)

2. If D_LCK bit is clear:
   1. Set D_OPEN bit to make SMRAM visible to protected mode code,
   2. Copy the SMM Handler code to the handler portion of SMRAM defined by Intel Docs,
   3. Clear D_OPEN bit and set D_LCK bit to protect our evil code ☺

3. We are invisible!

Routing IRQ 1 to Malicious SMM Handler

1. Modify the I/O APIC in such a way that when ever a user presses a key, our SMM code is executed,
2. SMM Handler reads the scan code, logs it and sends a special command to keyboard for overcoming the problem of a popped up scancode.
3. This in turn makes the next data written into the keyboard buffer available for OS Keyboard Interrupt handler,
4. Send an IPI to ourself for handling an emulated IRQ 1!
5. Let the OS think it is a real scancode generated by the keyboard encoder ☺

Pros & Cons

1. Pros
   1. Totally invisible to the OS!
   2. No need to change any OS created structures.
   3. Very hard to detect.

2. Cons
   1. Works only with PS/2
   2. Limited to single processor system
   3. D_LCK bit is already set on modern systems ☹

#2 IDT Hooking

Structure of an Interrupt Descriptor Table

1. Protected Mode counterpart of Real Mode Interrupt Vector Table (IVT),
2. Contains at most 256 entries.
3. Each entry is 8 bytes long and they are structured as defined below:

```
nt!_KIDTENTRY
   +0x000 Offset           : Uint2B
   +0x002 Selector         : Uint2B
   +0x004 Access           : Uint2B
   +0x006 ExtendedOffset   : Uint2B
```

Keyboard Interrupt is not mapped to IDT#1???

1. Where is IRQ 1 mapped? Which IDT Entry???
   - "IOAPIC makes IRQ and remaps IRQ to IDT."

```
kd> !ioapic
IoApic @ FEC00000  ID:1 (11)  Arb:0
Inti00.: 00000000`000100ff   Vec:FF  FixedDel   Ph:00000000     edg high     m
Inti01.: 01000000`00000991   Vec:91  LowestDl   Lg:01000000     edg high
Inti02.: 00000000`000100ff   Vec:FF  FixedDel   Ph:00000000     edg high     m
Inti03.: 00000000`000100ff   Vec:FF  FixedDel   Ph:00000000     edg high     m

kd> !idt –a
31: 84866058 i8042prt!I8042KeyboardInterruptService (KINTERRUPT 84866000) NO I/O APIC
91: 84864058 i8042prt!I8042KeyboardInterruptService (KINTERRUPT 84864000)
```

2. Methods for retrieving the vector address:
   - Use APIC
   - Scan kernel memory
   - Use the kernel API function (HalGetInterruptVector)

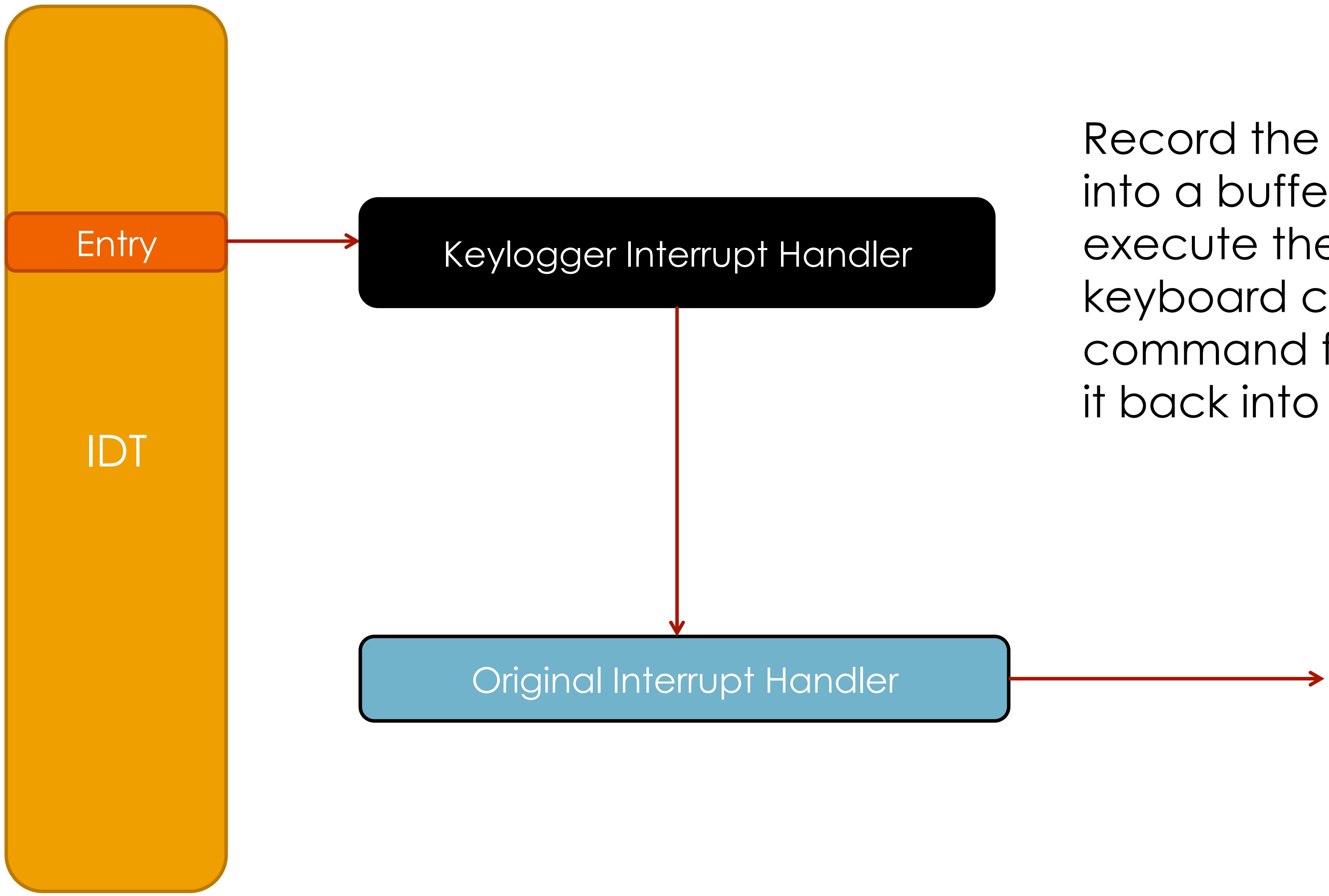
How to read scancode?

1. It’s as easy as executing an “in al,60h” instruction ☺
   - IN instruction empties the data, we need to put it back into its place for system’s use.
2. Here is an excerpt from the Keyboard Controller command set:

   Command 0xd2: Write keyboard output buffer

   Write the keyboard controllers output buffer with the byte next written to port 0x60, **and act as if this is a keyboard generated data.**

Here is the method



Record the keystroke into a buffer and execute the special keyboard controller command for putting it back into place

#3 Hacking KINTERRUPT

Structure of a KINTERRUPT

```
kd> !idt 91

Dumping IDT: 80b95400

91:     84864058 i8042prt!I8042KeyboardInterruptService (KINTERRUPT 84864000)

kd> dt nt!_KINTERRUPT 84864000
   +0x000 Type             : 0n22
   +0x002 Size             : 0n632
   +0x004 InterruptListEntry : _LIST_ENTRY [ 0x84864004 - 0x84864004 ]
   +0x00c ServiceRoutine   : 0x8a71d49a     unsigned char  i8042prt!I8042KeyboardInterruptService+0
   +0x010 MessageServiceRoutine : (null)
   +0x014 MessageIndex     : 0
   +0x018 ServiceContext   : 0x860252a8 Void
   +0x01c SpinLock         : 0
   +0x020 TickCount        : 0xffffffff
   +0x024 ActualLock       : 0x86025368  -> 0
   +0x028 DispatchAddress  : 0x8284adb0     void  nt!KiInterruptDispatch+0
   +0x02c Vector           : 0x91
   +0x030 Irql             : 0x8 ''
   +0x031 SynchronizeIrql  : 0x8 ''
   +0x032 FloatingSave     : 0 ''
   +0x033 Connected        : 0x1 ''
   +0x034 Number           : 0
   +0x038 ShareVector      : 0 ''
   +0x039 Pad              : [3]  ""
   +0x03c Mode             : 1 ( Latched )
   +0x040 Polarity         : 0 ( InterruptPolarityUnknown )
   +0x044 ServiceCount     : 0
   +0x048 DispatchCount    : 0xffffffff
   +0x050 Rsvd1            : 0
   +0x058 DispatchCode     : [135] 0x56535554
```

IDT Entry is actually pointing into a structure called KINTERRUPT

Where does this code come from?

1. KINTERRUPT->DispatchCode is actually a modified version of KiInterruptTemplate.

KiInterruptTemplate

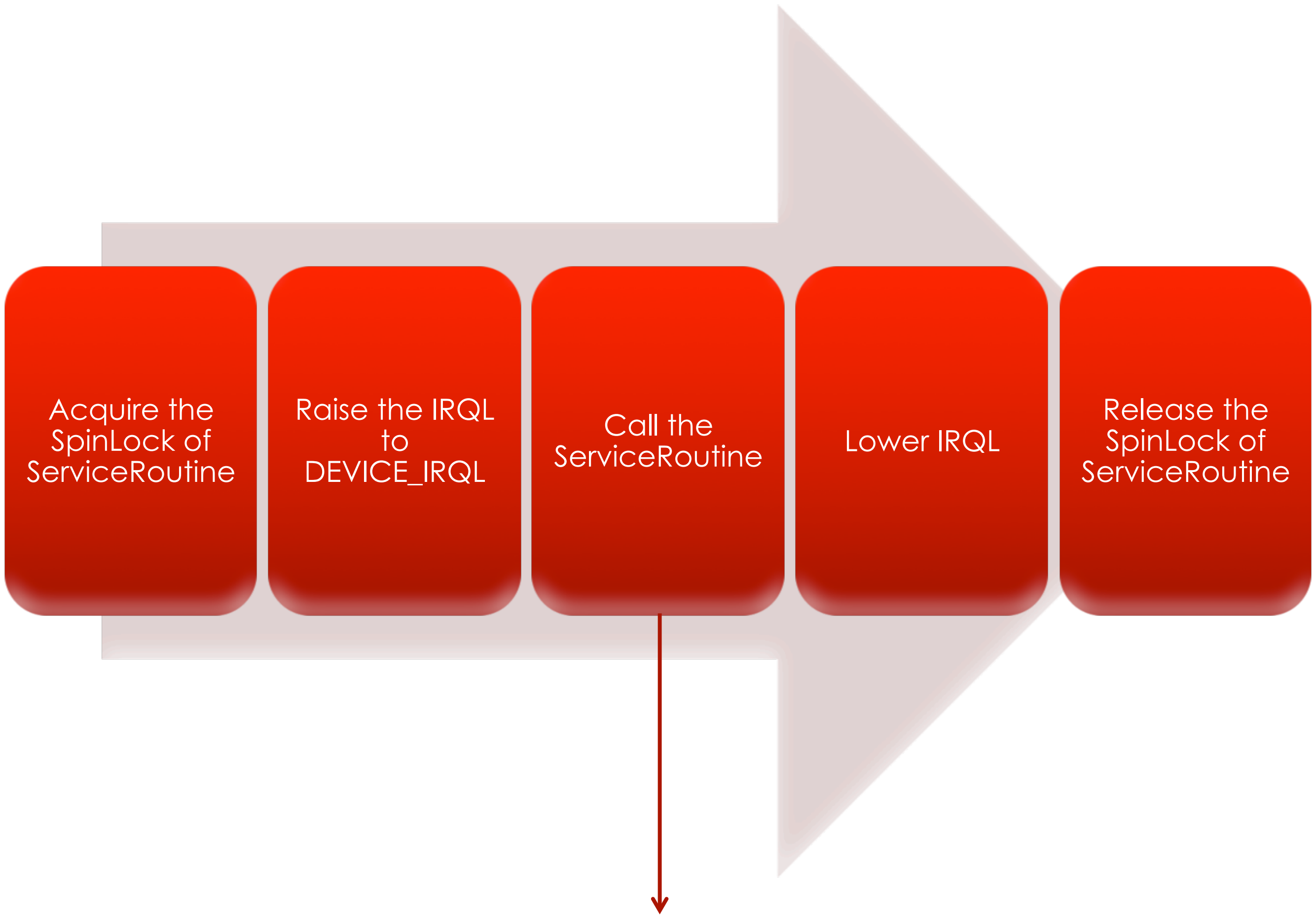
```
8284b00c 7522            jne     nt!Dr_kit_a (8284b030)
8284b00e 8b5d60                  [ebp+60h]
8284b011 8b7d68                  [ebp+68h]
8284b014 89550c          mov     dword ptr [ebp+0Ch],edx
8284b017 c74508000ddbba  mov     dword ptr [ebp+8],0BADB0D00h
8284b01e 895d00          mov     dword ptr [ebp],ebx
8284b021 897d04          mov     dword ptr [ebp+4],edi
nt!KiInterruptTemplate2ndDispatch:
8284b024 bf00000000      mov     edi,0
nt!KiInterruptTemplateObject:
8284b029 e922faffff      jmp     nt!KeSynchronizeExecution (8284aa50)
nt!KiInterruptTemplateDispatch:
8284b02e 8bff            mov     edi,edi
nt!Dr_kit_a:
8284b030 f7457000000200  test    dword ptr [ebp+70h],20000h
8284b037 7506            jne     nt!Dr_kit_a+0xf (8284b03f)
8284b039 f6456c01        test    byte ptr [ebp+6Ch],1
8284b03d 74cf            je      nt!KiInterruptTemplate+0xcb (8284b00e
```

KINTERRUPT->DispatchCode

```
8486411d f64103df        test    byte ptr [ecx+3],0DFh
84864121 7522
84864123 8b5d60
84864126 8b7d68          mov     edi,dword ptr [ebp+68h]
84864129 89550c          mov     dword ptr [ebp+0Ch],edx
8486412c c74508000ddbba  mov     dword ptr [ebp+8],0BADB0D00h
84864133 895d00          mov     dword ptr [ebp],ebx
84864136 897d04          mov     dword ptr [ebp+4],edi
84864139 bf00408684      mov     edi,84864000h
8486413e e96d6cfefd      jmp     nt!KiInterruptDispatch (8284adb0)
84864143 8bff            mov     edi,edi
84864145 f7457000000200  test    dword ptr [ebp+70h],20000h
8486414c 7506            jne     84864154
8486414e f6456c01        test    byte ptr [ebp+6Ch],1
84864152 74cf            je      84864123
84864154 0f21c3          mov     ebx,dr0
84864157 0f21c9          mov     ecx,dr1
8486415a 0f21d7          mov     edi,dr2
```

2. Can be easily modified for different kinds of interrupts such as KiChainedDispatch, KiFloatingDispatch.

What does a DispatchCode do?



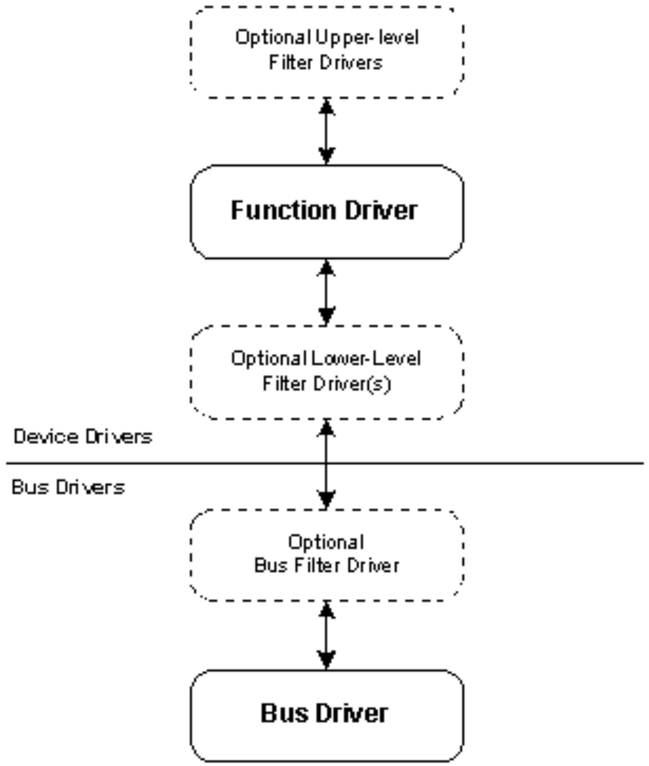
This is the point where “Interrupt Servicing” takes place!
i8042KeyboardInterruptService

How to intercept

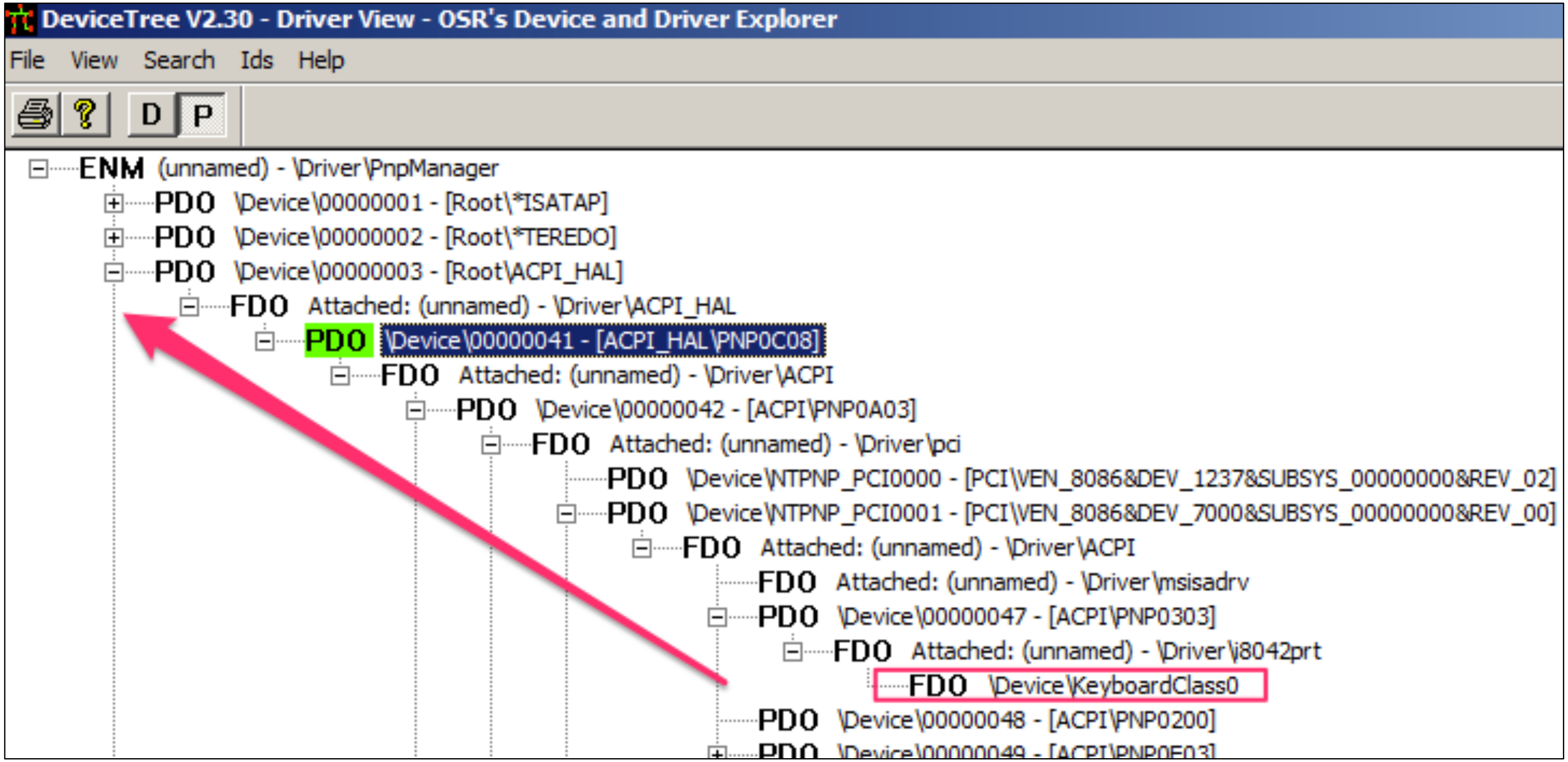
1. Put an inline hook into DispatchCode's prolog,
2. Create a new KINTERRUPT object and make EDI point to it,
3. Replace the ServiceRoutine field of KINTERRUPT,
4. Inline hook the ServiceRoutine.

Windows Driver Model

- A layered design with support for adding drivers into the stack dynamically.
- Great design for management.
- Allows another driver to filter some other driver's packets.

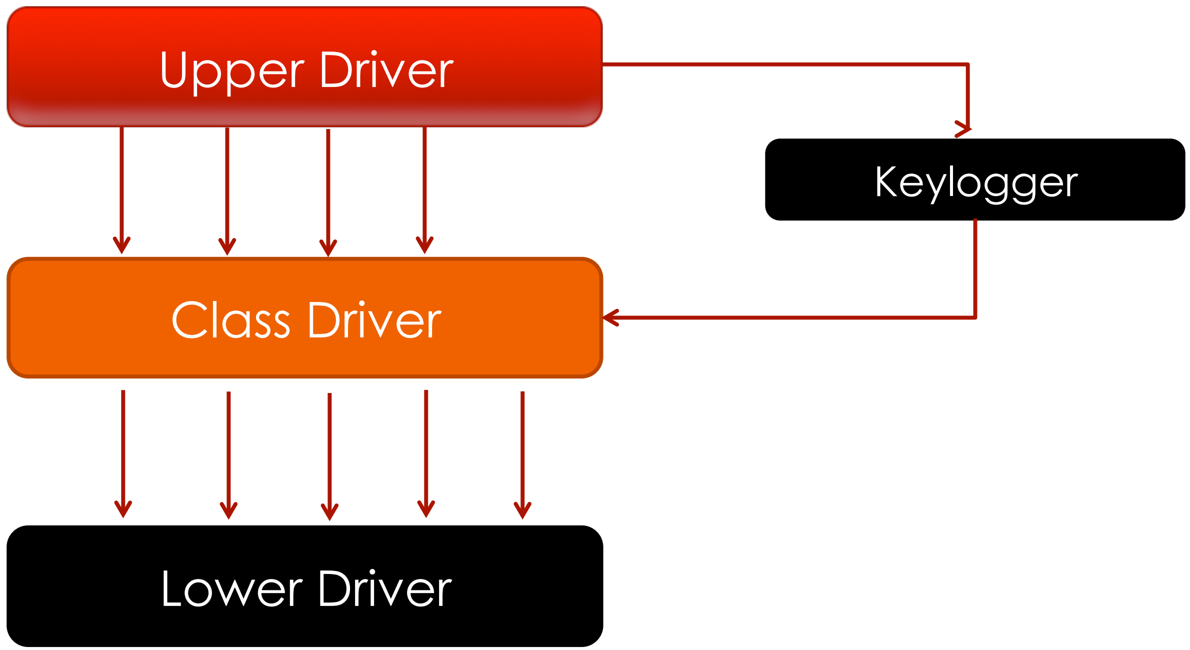


Keyboard Device Stack



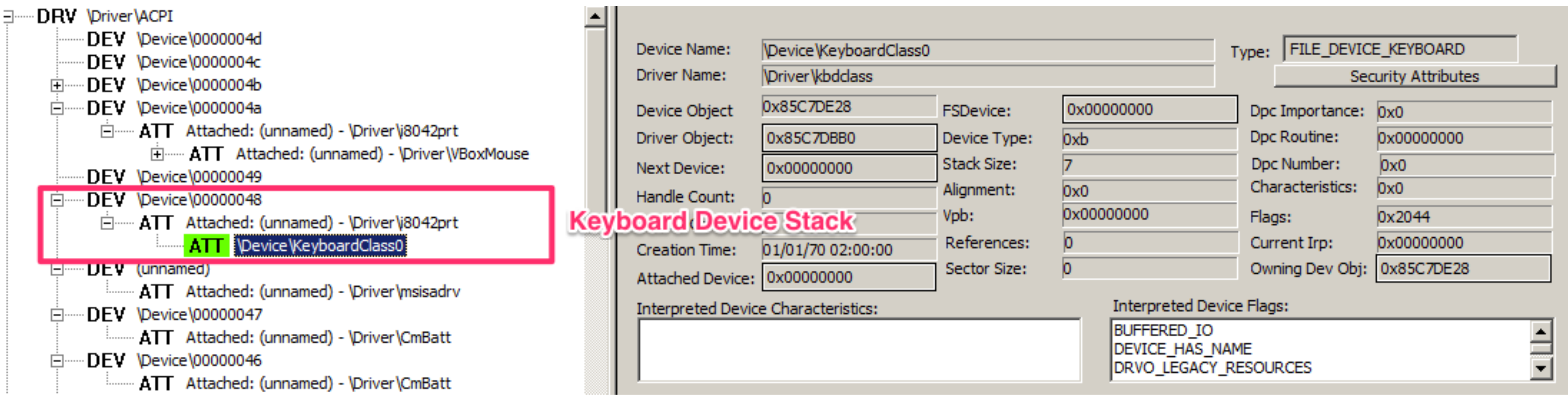
What is an IRP?

- A structure which is used by the I/O manager for defining a request targeted to a device.
- Reading a file, writing to a file and much more operation is handled with IRPs.
- Each IRP has a Major code which makes it possible to call appropriate handler for that IRP.

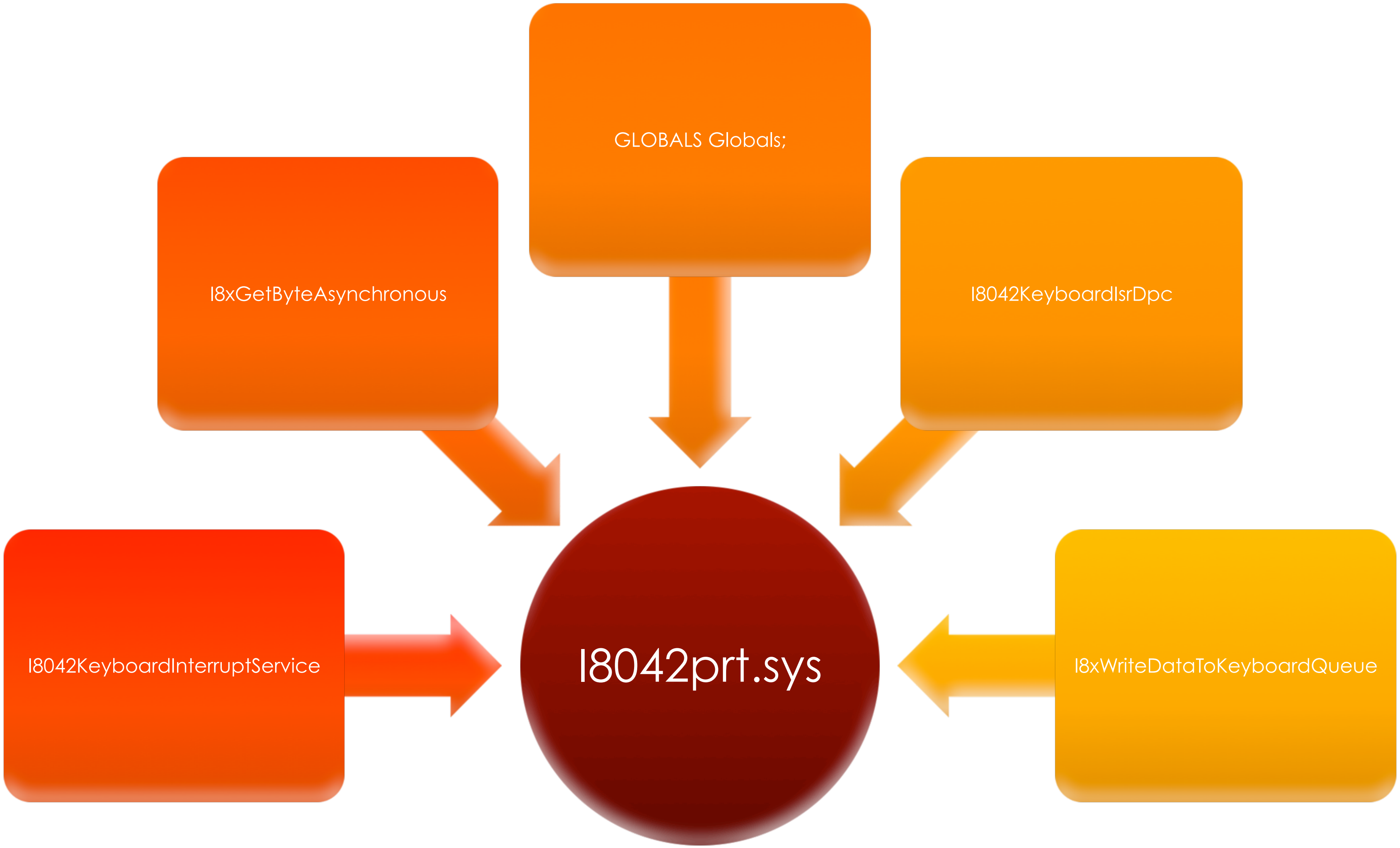


i8042prt.sys

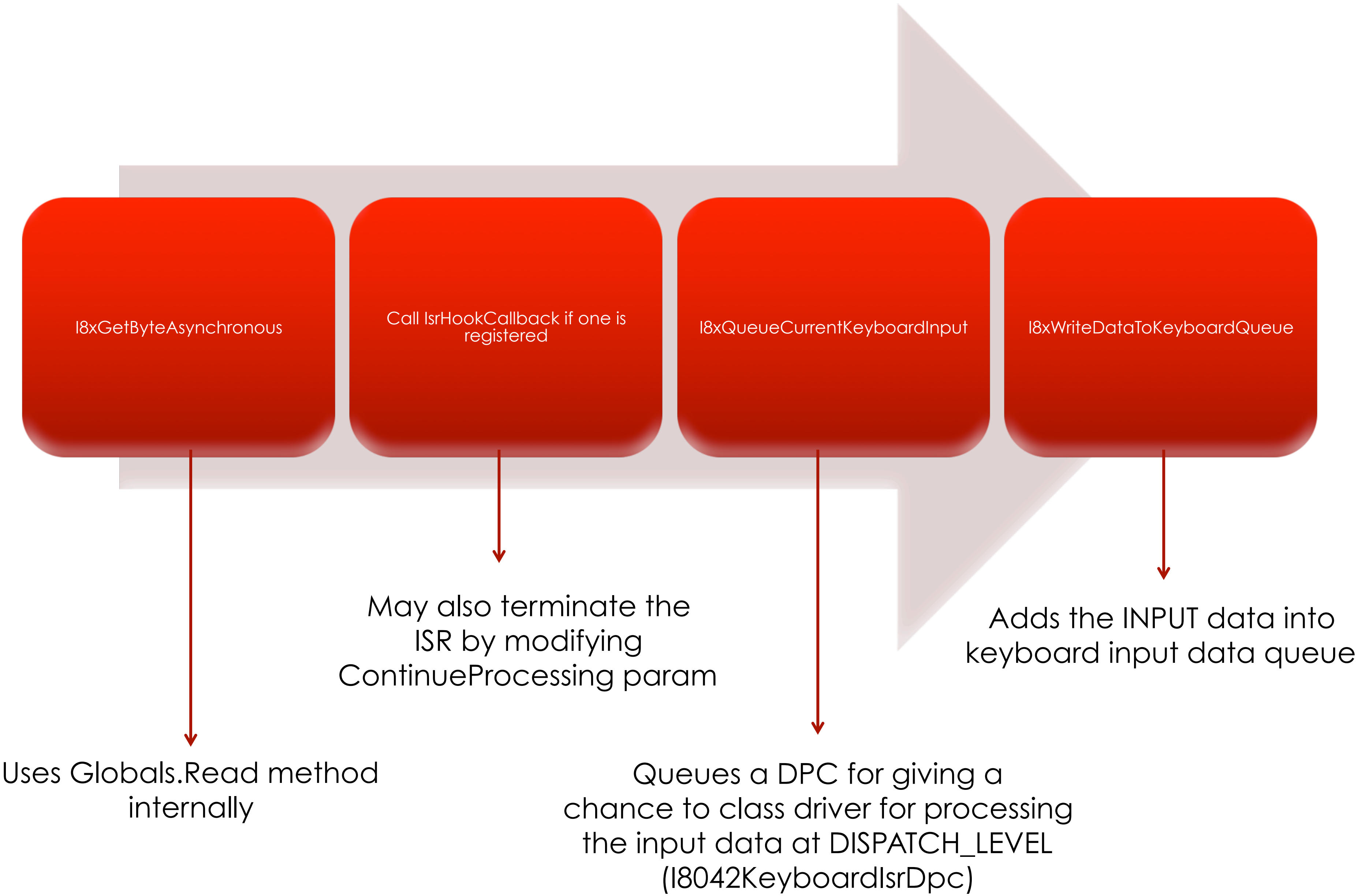
1. Port driver for 8042 compatible keyboard and mouse devices.
2. Handles the interrupt for a keyboard device and delivers it to the system.
3. Contains good candidates for a keylogger.



i8042prt.sys



i8042prt.sys Overview



#4 i8042prt!Globals Hack

i8042prt.sys GLOBALS structure

A look into i8042prt!Globals


```
8d94c601 ffb0a0000000    push    dword ptr [eax+0A0h]
8d94c607 ff15cc40958d    call    dword ptr [i8042prt!Globals+0xc (8d9540cc)]
8d94c60d 8807            mov     byte ptr [edi],al
8d94c60f 0fb6c0          movzx   eax,al
```

```
kd> dps i8042prt!Globals
8d9540c0  85799cd8
8d9540c4  8594cab8
8d9540c8  85a52c88
8d9540cc  8281a094 hal!READ_PORT_UCHAR
8d9540d0  8281a0fc  hal!WRITE_PORT_UCHAR
8d9540d4  00720070
8d9540d8  859b3c80
```

Replace it with your own ☺

Globals Read Data Hook

```
kd> bl *
 0 d 8d94c57c     0001 (0001) i8042prt!I8xGetByteAsynchronous+0x81   "r al;g;"
 1 d 8d94c599     0001 (0001) i8042prt!I8xGetByteAsynchronous+0x9e   "r al;g;"
 2 e 8d94c60d     0001 (0001) i8042prt!I8xGetByteAsynchronous+0x112 "r al;g;"
```

0 3d 0 3d 0 3d 9 1d 1e 1d 9e 1d 1f 1d 9f 1d 20 1d a0 1d 21 1d a1 1d 22 1d a2 1d 23 1d

Here we have the keystrokes, also little noisy but can be parsed with a simple script.

#5 I8xGetByteAsynchronous

I8xGetByteAsynchronous

- Defined as

  I8xGetByteAsynchronous(CCHAR KeyboardType,UCHAR*ScanCode)

- Pretty good place to hook.
- Internally uses Global.Read

#6 Hacking IsrHookCallback

IsrHookCallback

- Used by upper level drivers to modify the scancode in the ISR routine.
- Gets called right after scan code is retrieved from the keyboard controller.

PI8042_KEYBOARD_ISR function pointer

This topic has not yet been rated – Rate this topic

A PI8042_KEYBOARD_ISR-typed callback routine customizes the operation of the I8042prt keyboard ISR.

Syntax

C++

```
typedef BOOLEAN ( *PI8042_KEYBOARD_ISR)(
  _In_    PVOID IsrContext,
  _In_    PKEYBOARD_INPUT_DATA CurrentInput,
  _In_    POUTPUT_PACKET CurrentOutput,
  _In_    UCHAR StatusByte,
  _In_    PUCHAR Byte,
  _Out_   PBOOLEAN ContinueProcessing,
  _In_    PKEYBOARD_SCAN_STATE ScanState
);
```

Hack IsrHookCallback

- As easy as modifying DEVICE_EXTENSION of port device:
  - DeviceObject->DeviceExtension->IsrHookCallback
- Right after that, keys will start flowing into our callback!
- Callback can even stop the ISR’s processing.

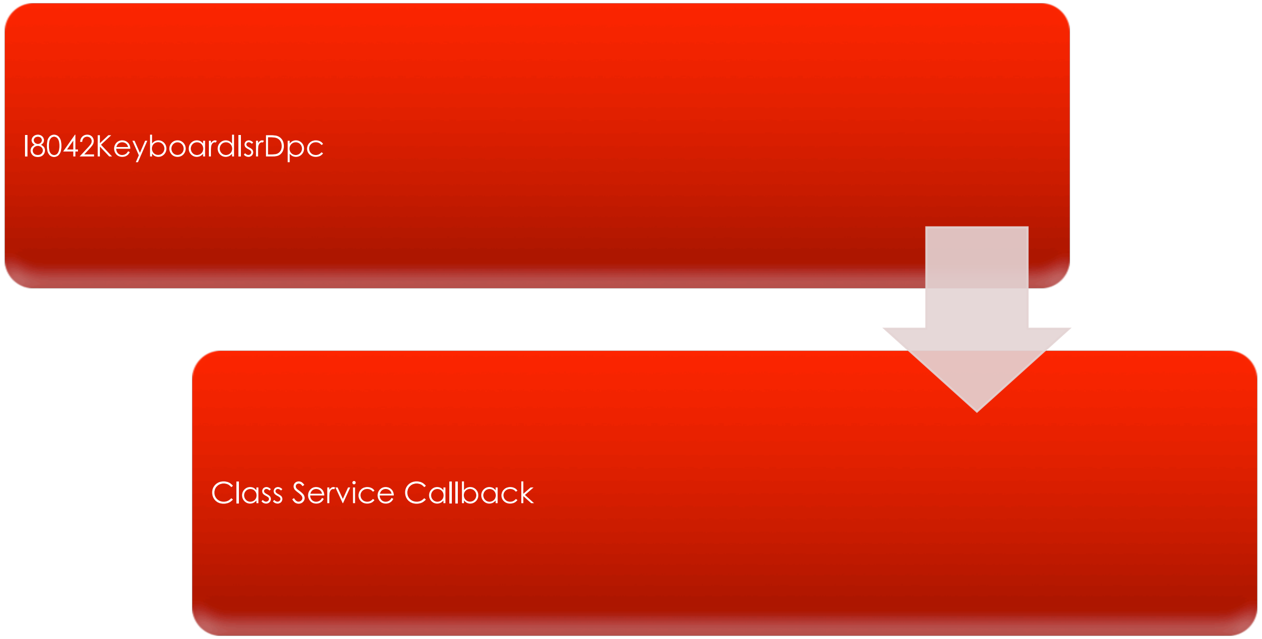
#7 Hacking ClassService

What does I8xQueueCurrentKeyboardInput do?

- Queues a DPC for further processing.
- DPC calls DeviceExtension->ConnectData.ClassService function for delivering the scan code information to the class driver.
- Question: Can’t we hook that?
- Answer: Definitely yes!
- How: Replace the ClassService function with your own ☺

I8xQueueCurrentKeyboardInput

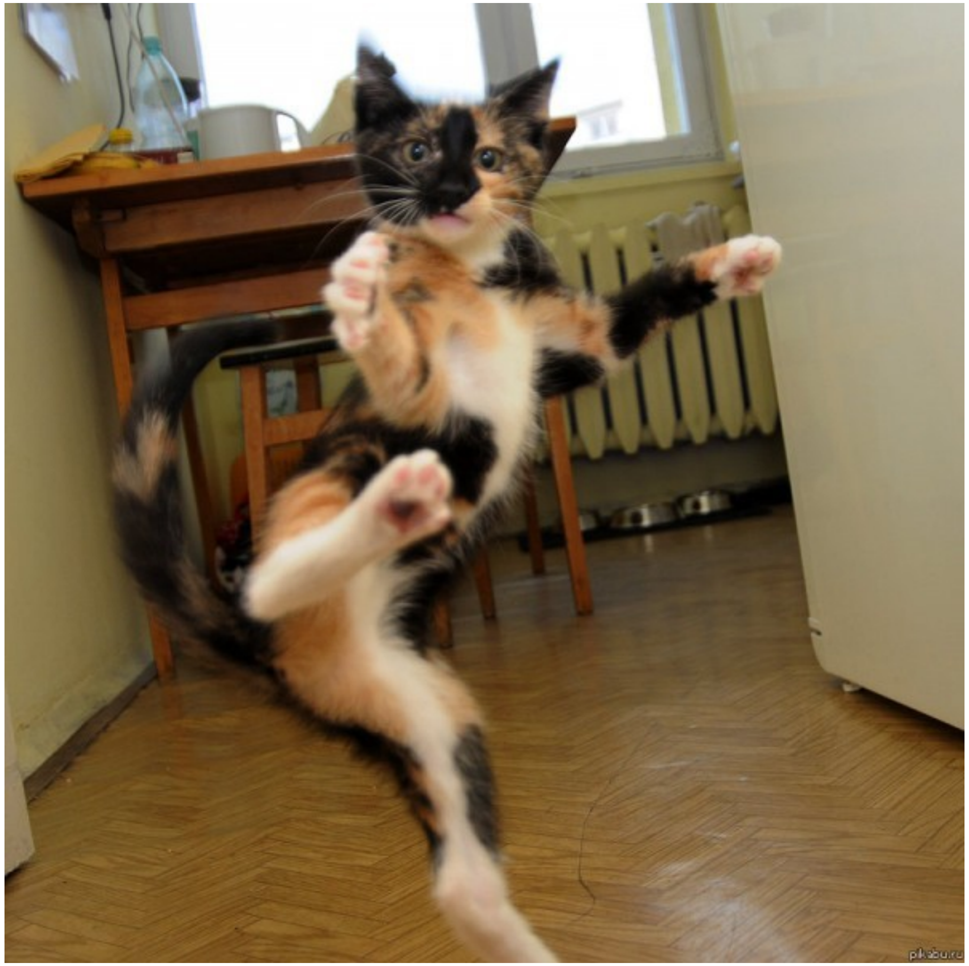
- Queues a DPC object for further processing the input data.
- This gives class drivers or any upper level drivers a chance to process the input data structure, even modify it!
- As soon as IRQL drops to DISPATCH_LEVEL, DPC gets executed and calls the callback supplied by Class Driver.



DPC – Deferred Procedure Call

- Time is a precious thing!
- Do what ever you can to make hardware feel better and queue a procedure to be called when everything is OK.
- This prevents keeping a CPU at a high IRQL level for a long time.

#8 I8xWriteDataToKeyboardQueue



I8xWriteDataToKeyboardQueue

- A great candidate for hooking!
- Gets the INPUT data as it’s second parameter and writes that into it’s internal data queue.
- Flags describe whether the key is down or up.

KEYBOARD_INPUT_DATA structure

0 out of 1 rated this helpful – Rate this topic

KEYBOARD_INPUT_DATA contains one packet of keyboard input data.

Syntax

C++

```
typedef struct _KEYBOARD_INPUT_DATA {
  USHORT UnitId;
  USHORT MakeCode;
  USHORT Flags;
  USHORT Reserved;
  ULONG  ExtraInformation;
} KEYBOARD_INPUT_DATA, *PKEYBOARD_INPUT_DATA;
```

This is the scan code!

#9 Filter Drivers

How to filter?

- Meaning of layer in malware authors slang:
  - "A point for injecting evil"

- Two methods:
  - IoAttachDevice API: The **IoAttachDevice** routine attaches the caller's device object to a named target device object, so that I/O requests bound for the target device are routed first to the caller.

```
NTSTATUS IoAttachDevice(
  _In_   PDEVICE_OBJECT SourceDevice,
  _In_   PUNICODE_STRING TargetDevice,
  _Out_  PDEVICE_OBJECT *AttachedDevice
);
```

  - Registry hacks for devices. Set UpperFilter and LowerFilters. Upper filter drivers go between the operating system and the main driver, and lower filter drivers go between the main driver and the hardware.

Let’s check for Keyboard Filters

1. Go to Materials/Applications copy RegShot directory to your Desktop.
2. Execute “regshot.exe”
3. Set output path to “Desktop”
4. Click on “1st Shot” -> “Shot”
5. Install “Zemana AntiLogger Free.exe”
6. Go to regshot again and click “2nd Shot” -> “Shot”
7. Click “compare”
8. Search for “UpperFilters” (Upper filters for keyboard device)
9. Copy the GUID and google it. Guess what does it define?
10. Restart the machine in DEBUG MODE and execute:
    1. !drvobj \Device\kbdclass
    2. !devstack SECOND OBJECT ADDRESS

#10 IRP Handler Hooking

Keyboard Class Driver

- \Driver\kbdclass
- Represents a Keyboard Device either USB or PS/2.
- Used **exclusively** by the Raw Input Thread (RIT) (coming next).

```
kd> !drvobj 0x85768f08 7
Driver object (85768f08) is for:
 \Driver\kbdclass
Driver Extension List: (id , addr)

Device Object list:
85861030  857687d8

DriverEntry:   8e1419f2 kbdclass!GsDriverEntry
DriverStartIo: 00000000
DriverUnload:  00000000
AddDevice:     8e13fdee kbdclass!KeyboardAddDevice

Dispatch routines:
[00] IRP_MJ_CREATE                      8e13a000      kbdclass!KeyboardClassCreate
[01] IRP_MJ_CREATE_NAMED_PIPE           828d20e5      nt!IopInvalidDeviceRequest
[02] IRP_MJ_CLOSE                       8e13a294      kbdclass!KeyboardClassClose
[03] IRP_MJ_READ                        8e13b0ba      kbdclass!KeyboardClassRead
[04] IRP_MJ_WRITE                       828d20e5      nt!IopInvalidDeviceRequest
[05] IRP_MJ_QUERY_INFORMATION           828d20e5      nt!IopInvalidDeviceRequest
[06] IRP_MJ_SET_INFORMATION             828d20e5      nt!IopInvalidDeviceRequest
[07] IRP_MJ_QUERY_EA                    828d20e5      nt!IopInvalidDeviceRequest
[08] IRP_MJ_SET_EA                      828d20e5      nt!IopInvalidDeviceRequest
[09] IRP_MJ_FLUSH_BUFFERS               8e139f78      kbdclass!KeyboardClassFlush
```

Look at the difference

- KbdClass has a READ routine while the Port Driver doesn't! Why?

Command

```
kd> !devobj 0x857689D0
Device object (857689d0) is for:
 \Driver\i8042prt DriverObject 857a5a28
Current Irp 00000000 RefCount 0 Type 00000027 Flags 00002004
DevExt 85768a88 DevObjExt 85768d18
ExtensionFlags (0x00000800)  DOE_DEFAULT_SD_PRESENT
Characteristics (0000000000)
AttachedDevice (Upper) 857687d8 \Driver\kbdclass
AttachedTo (Lower) 84870030 \Driver\ACPI
Device queue is not busy.
kd> !drvobj 857a5a28 7
Driver object (857a5a28) is for:
 \Driver\i8042prt
Driver Extension List: (id , addr)

Device Object list:
85785020  857689d0

DriverEntry:   8e132138 i8042prt!GsDriverEntry
DriverStartIo: 8e1227bc i8042prt!I8xStartIo
DriverUnload:  8e12ea31 i8042prt!I8xUnload
AddDevice:     8e12dfe3 i8042prt!I8xAddDevice

Dispatch routines:
[00] IRP_MJ_CREATE                      8e12b96b    i8042prt!I8xCreate
[01] IRP_MJ_CREATE_NAMED_PIPE           828d20e5    nt!IopInvalidDeviceRequest
[02] IRP_MJ_CLOSE                       8e12e3c1    i8042prt!I8xClose
[03] IRP_MJ_READ                        828d20e5    nt!IopInvalidDeviceRequest
[04] IRP_MJ_WRITE                       828d20e5    nt!IopInvalidDeviceRequest
[05] IRP_MJ_QUERY_INFORMATION           828d20e5    nt!IopInvalidDeviceRequest
[06] IRP_MJ_SET_INFORMATION             828d20e5    nt!IopInvalidDeviceRequest
[07] IRP_MJ_QUERY_EA                    828d20e5    nt!IopInvalidDeviceRequest
[08] IRP_MJ_SET_EA                      828d20e5    nt!IopInvalidDeviceRequest
[09] IRP_MJ_FLUSH_BUFFERS               8e125f54    i8042prt!I8xFlush
```

Here is why

- Port driver doesn't provide a read routine because it expects a "Keyboard Class Service Callback" to be registered by a class driver.
- Class driver gets the requests from the RIT and waits for KeyboardClassServiceCallback to get called by the keyboard port driver's ISR DPC.
- This callback is registered by sending an IRP carrying a structure called as CONNECT_DATA with an IOCTL_INTERNAL_KEYBOARD_CONNECT code.
- This in turn makes the port driver record this callback routine for calling whenever an interrupt occurs.
- When ever the service callback gets called by port driver's DPC, class driver completes the request of RIT which makes the RIT send another request.

KeyboardClassServiceCallback

- Routine which dequeues an IRP each time it gets called by the port driver's ISR DPC.
- As soon as data is copied to the IRP, it completes the IRP with STATUS_SUCCESS.

#12 Inline hooking for ClassCallback

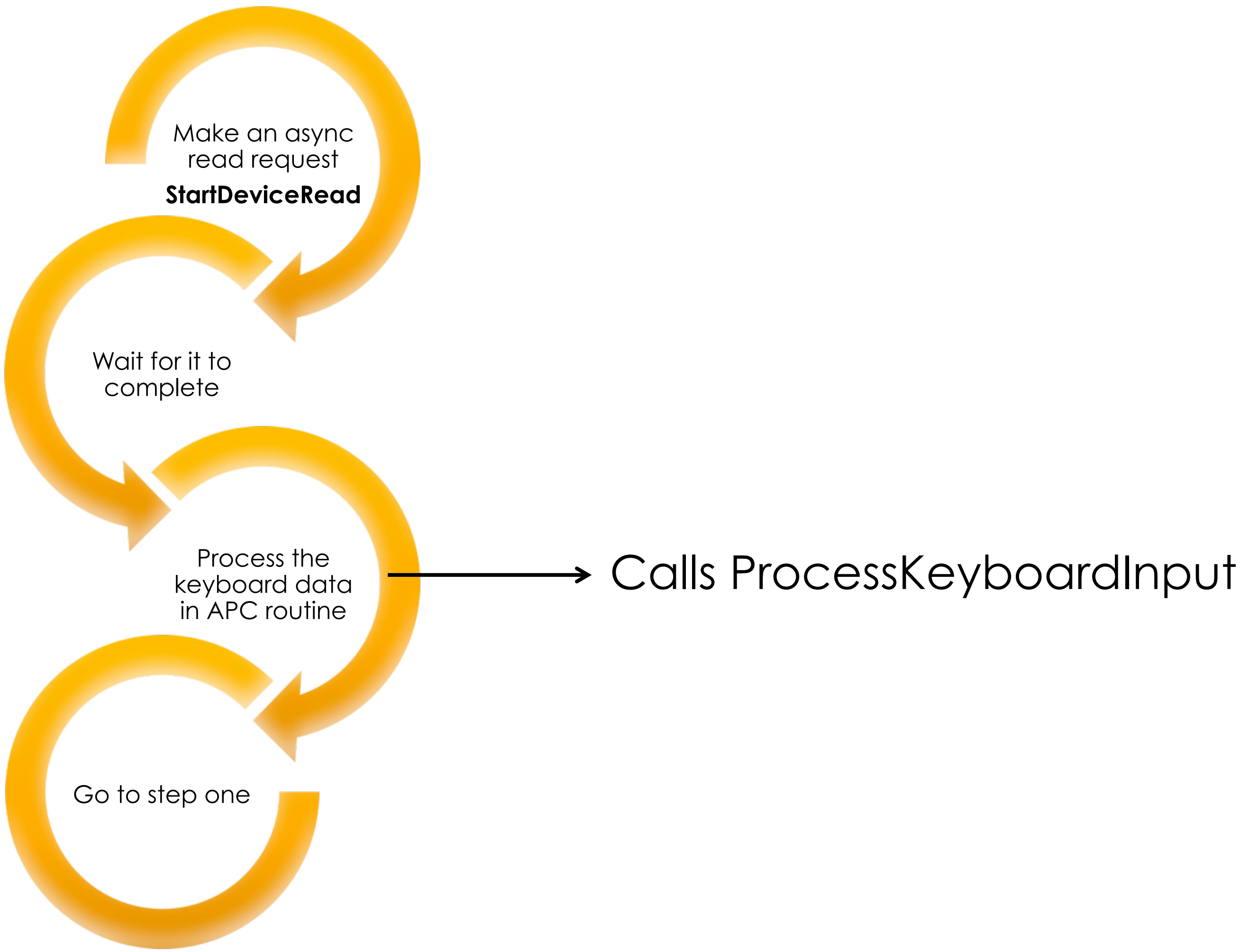
Hook the class callback

- We have already hacked this callback routine but in a different way. It was just a replacement of a pointer in ConnectData structure residing in port driver's DeviceExtension.
- This time, another approach.
- Put an inline hook into KeyboardClassServiceCallback which will make us the king of scancodes ☺
- As easy as putting a 5 byte prolog into the routine.

Let's talk about "Raw Input Thread"

- A thread of ***csrss.exe*** which continuously makes a read request to keyboard class device.
- It is the guy who retrieves keystrokes from the class driver and posts them to appropriate queues.
- It's mainly a loop which makes a request and waits for that request to complete which in turn makes another request and so forth…
- Key method here is StartDeviceRead which sends a read request to class driver asynchronously with an APC object.

How it functions?



#13 Hacking Device Templates

What is a Device Template?

- A structure for keeping device specific attributes such as keyboard and mouse.
- This is where the word “KbdClass” comes from ☺
- Also contains a function pointer which is responsible for processing the Keyboard or Mouse input hence the name : “ProcessKeyboardInput”

Device Template

Command

```
kd> uf win32k!InputApc
win32k!InputApc:
91631747 8bff            mov     edi,edi
91631749 55              push    ebp
9163174a 8bec            mov     ebp,esp
9163174c 56              push    esi
9163174d 8b7508          mov     esi,dword ptr [ebp+8]
91631750 ff4e48          dec     dword ptr [esi+48h]
91631753 f6460e80        test    byte ptr [esi+0Eh],80h
91631757 7420            je      win32k!InputApc+0x32 (91631779)

win32k!InputApc+0x12:
91631759 e83b1e0a00      call    win32k!EnterCrit (916d3599)
9163175e e8f3510500      call    win32k!EnterDeviceInfoListCrit_ (91686956)
91631763 80660dfd        and     byte ptr [esi+0Dh],0FDh
91631767 56              push    esi
91631768 e8ef34ffff      call    win32k!FreeDeviceInfo (91624c5c)
9163176d e8d3510500      call    win32k!LeaveDeviceInfoListCrit_ (91686945)
91631772 e8401e0a00      call    win32k!UserSessionSwitchLeaveCrit (916d35b7)
91631777 eb22            jmp     win32k!InputApc+0x54 (9163179b)

win32k!InputApc+0x32:
91631779 8b450c          mov     eax,dword ptr [ebp+0Ch]
9163177c 833800          cmp     dword ptr [eax],0
9163177f 7c14            jl      win32k!InputApc+0x4e (91631795)

win32k!InputApc+0x3a:
91631781 837e1c00        cmp     dword ptr [esi+1Ch],0
91631785 740e            je      win32k!InputApc+0x4e (91631795)

win32k!InputApc+0x40:
91631787 0fb6460c        movzx   eax,byte ptr [esi+0Ch]
9163178b 6bc03c          imul    eax,eax,3Ch
9163178e 56              push    esi
9163178f ff902cc78191    call    dword ptr win32k!aDeviceTemplate+0x2c (9181c72c)[eax]

win32k!InputApc+0x4e:
91631795 56              push    esi
91631796 e87afdffff      call    win32k!StartDeviceRead (91631515)

win32k!InputApc+0x54:
9163179b 5e              pop     esi
9163179c 5d              pop     ebp
9163179d c20c00          ret     0Ch
```

What do we have here?

It’s dump time

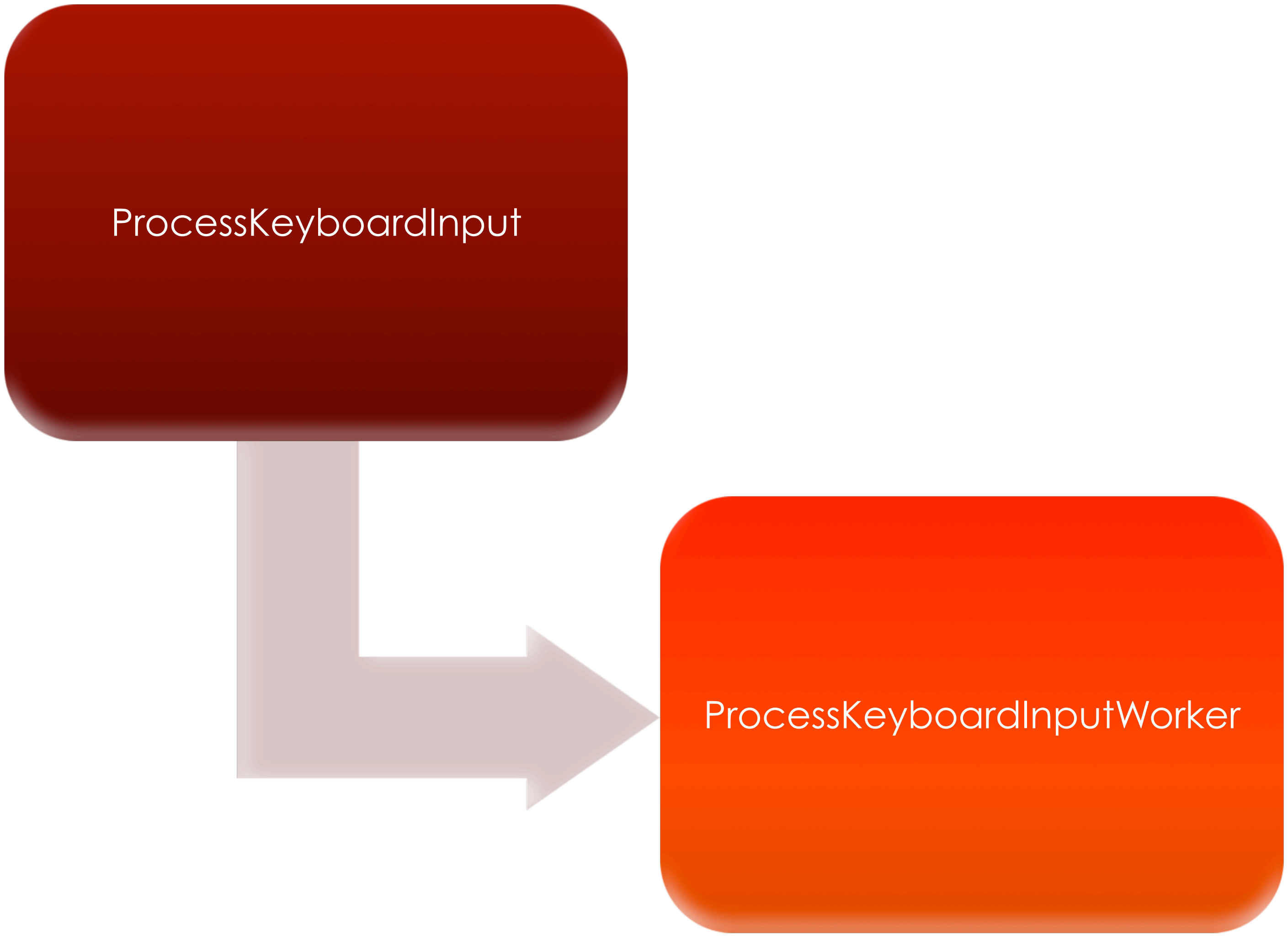
Command

```
kd> dps win32k!aDeviceTemplate
9181c700  0000014c
9181c704  91804784 win32k!GUID_DEVINTERFACE_MOUSE
9181c708  00000024
9181c70c  91808878 win32k!`string'
9181c710  9180884c win32k!`string'
9181c714  91808814 win32k!`string'
9181c718  000f0000
9181c71c  00000050
9181c720  0000000c
9181c724  0000005c
9181c728  000000f0
9181c72c  916317a5 win32k!ProcessMouseInput
9181c730  85f00f40
9181c734  00000000
9181c738  ffffffff
9181c73c  000000ec
9181c740  91804774 win32k!GUID_DEVINTERFACE_KEYBOARD
9181c744  00000025
9181c748  91808800 win32k!`string'
9181c74c  918087d0 win32k!`string'
9181c750  91808794 win32k!`string'
9181c754  000b0000
9181c758  00000050
9181c75c  0000001c
9181c760  00000074
9181c764  00000078
9181c768  917098e2 win32k!ProcessKeyboardInput
9181c76c  86124450
9181c770  00000000
9181c774  00000001
9181c778  00000058
9181c77c  91804824 win32k!GUID_DEVINTERFACE_HID
```

Handlers here

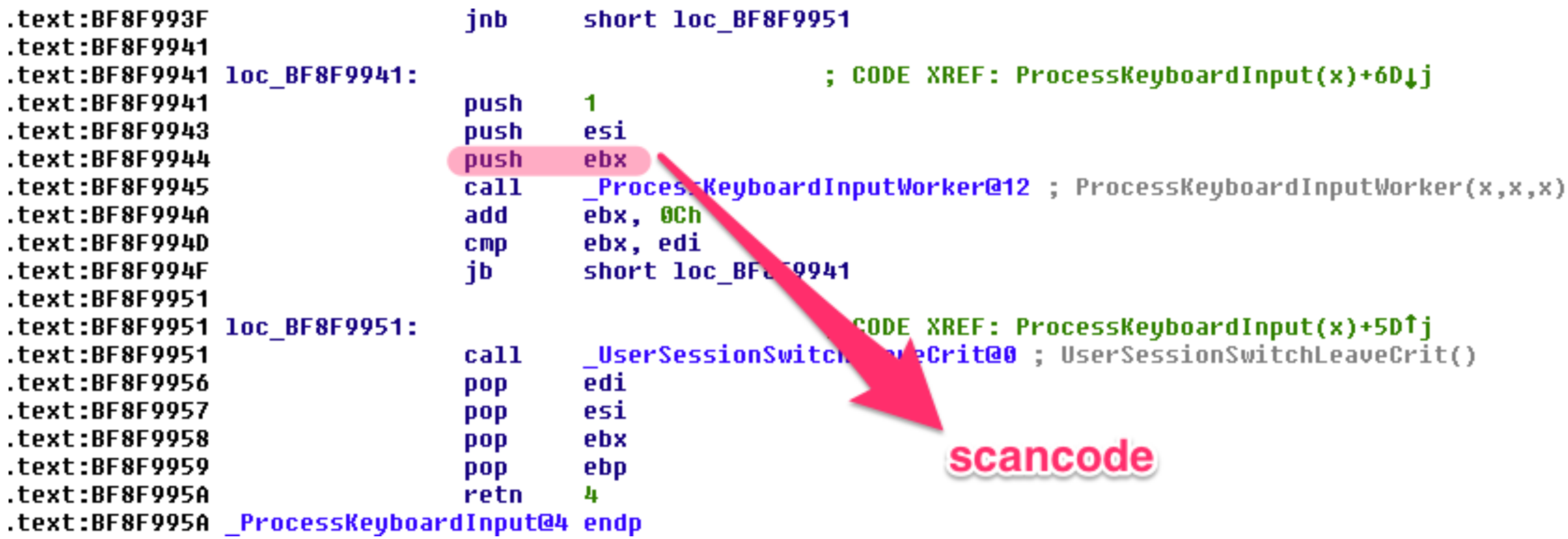
#14 Hook ProcessKeyboardInput

ProcessKeyboardInput



Inside ProcessKeyboardInput

- Find the first call to worker function.
- EBX points to scancode,
- Worker function is also a good target.



#15 Hook ProcessKeyboardInputWorker

Inline Hook ProcessKeyboardInputWorker

- Pretty obvious ☺
- You can easily see that it is a 3 parameter function with the 1st parameter as ScanCode.

#16 Hacking xxxProcessKeyEvent

xxxProcessKeyEvent

- Called by ProcessKeyboardInputWorker until each input event gets consumed.
- Lets take a look at the parameters:
  - Pointer to a Keyboard Event structure,
  - An ULONG_PTR value carrying extra information,
  - A flag indicating if key is from hardware or not.
- Performs some language specific operations.

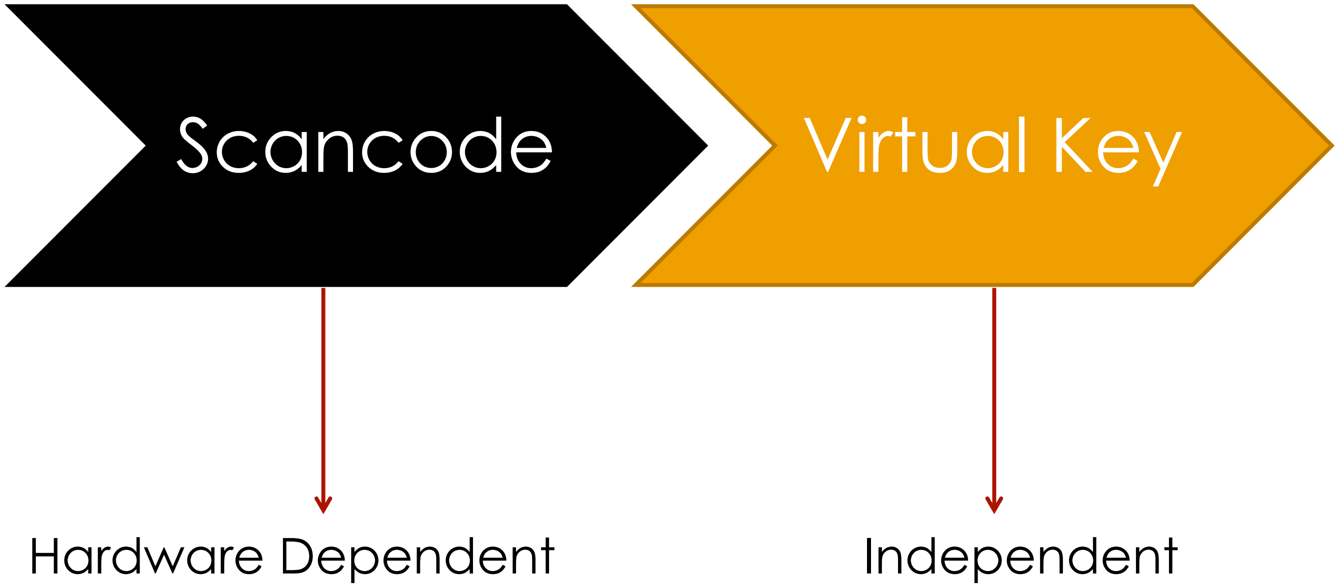
Break on xxxProcessKeyEvent

Command

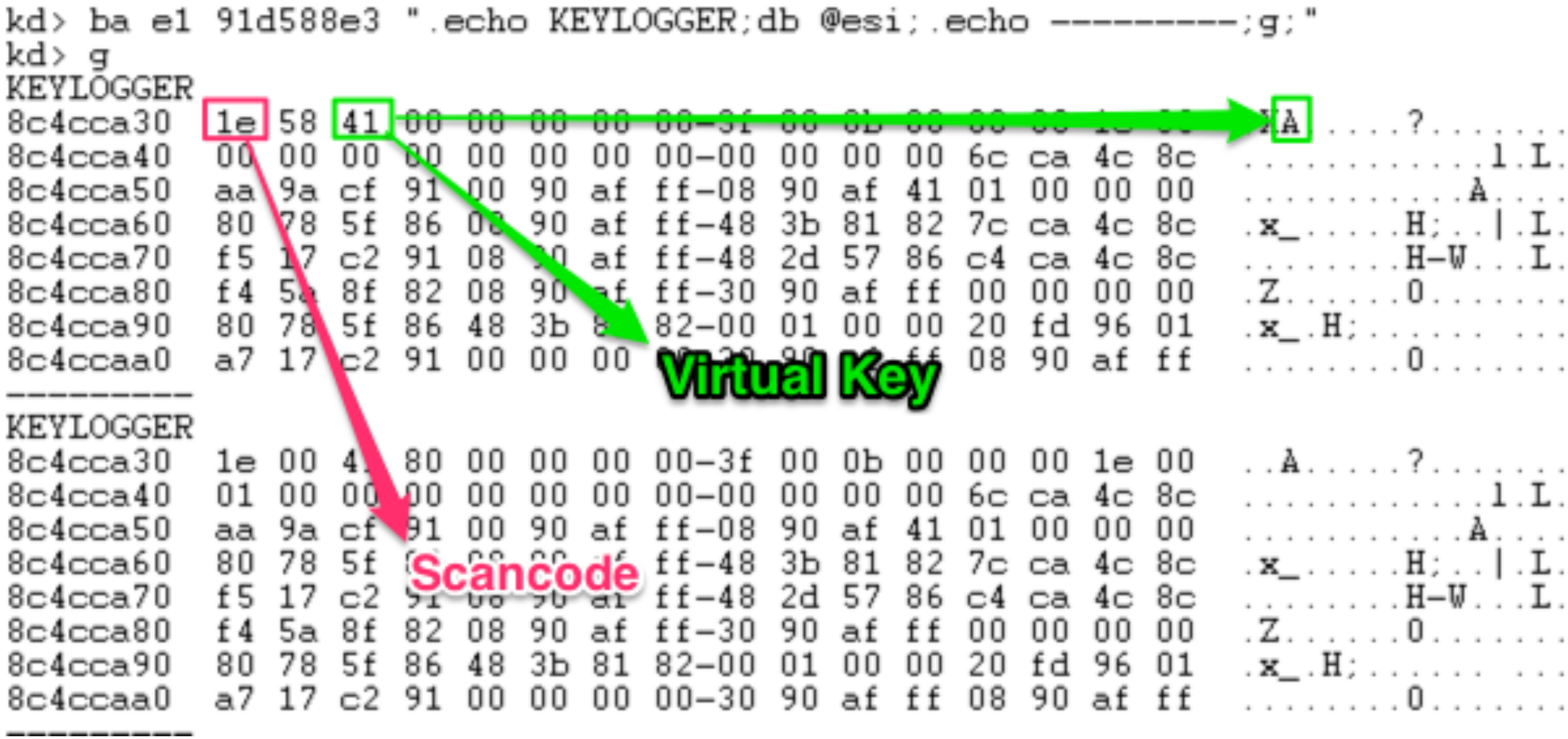
```
kd> uf win32k!xxxProcessKeyEvent
win32k!xxxProcessKeyEvent:
91d588d3 8bff            mov     edi,edi
91d588d5 55              push    ebp
91d588d6 8bec            mov     ebp,esp
91d588d8 51              push    ecx
91d588d9 a1b83de291      mov     eax,dword ptr [win32k!gptiCurrent (91e23db8)]
91d588de 53              push    ebx
91d588df 56              push    esi
91d588e0 8b7508          mov     esi,dword ptr [ebp+8]
91d588e3 8945fc          mov     dword ptr [ebp-4],eax
91d588e6 8a4602          mov     al,byte ptr [esi+2]
91d588e9 57              push    edi
91d588ea 884508          mov     byte ptr [ebp+8],al
91d588ed e85aacffff      call    win32k!GetActiveHKL (91d5354c)
91d588f2 25ff030000      and     eax,3FFh
91d588f7 6683f812        cmp     ax,12h
91d588fb 0fb74602        movzx   eax,word ptr [esi+2]
91d588ff bb00800000      mov     ebx,8000h
91d58904 753f            jne     win32k!xxxProcessKeyEvent+0x72 (91d58945)
```

Key Event

Virtual Key vs. Scan Code

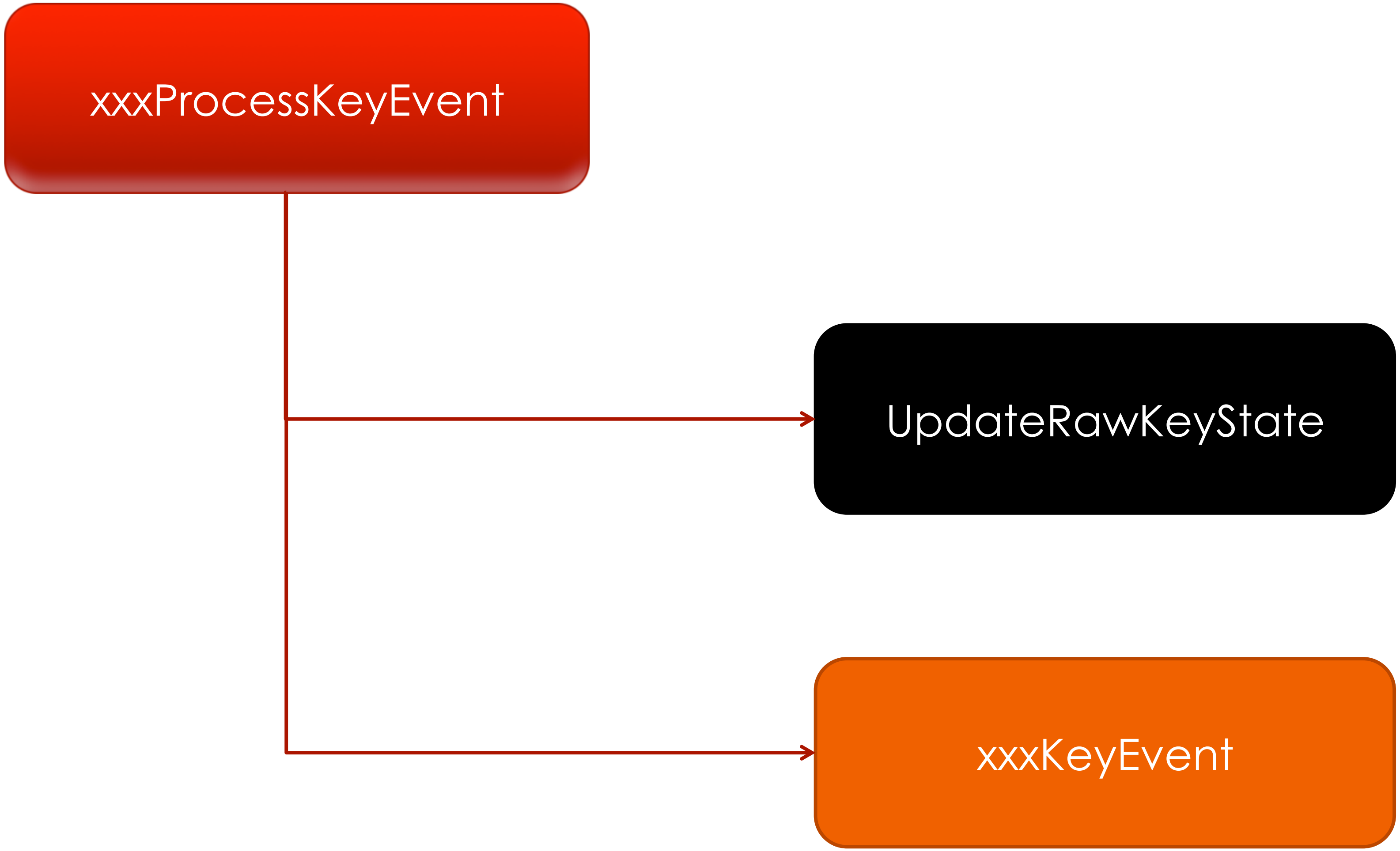


Virtual Key vs. Scan Code



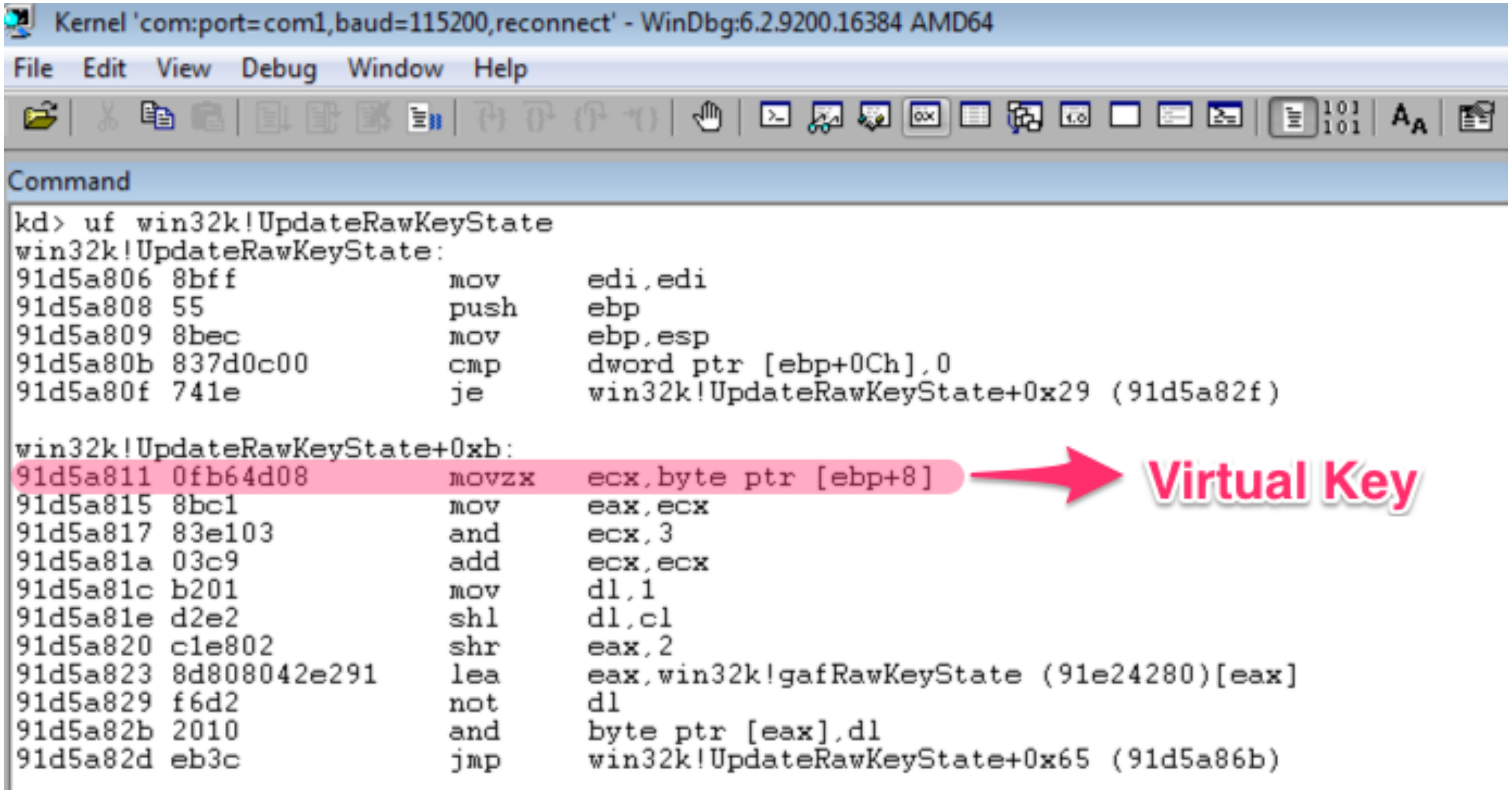
```
kd> ba e1 91d588e3 ".echo KEYLOGGER;db @esi;.echo ---------;g;"
kd> g
KEYLOGGER
8c4cca30  1e 58 41 00 00 00 00 00-3f 00 0b 00 00 00 1e 00  .XA.....?.......
8c4cca40  00 00 00 00 00 00 00 00-00 00 00 00 6c ca 4c 8c  ............l.L.
8c4cca50  aa 9a cf 91 00 90 af ff-08 90 af 41 01 00 00 00  ...........A....
8c4cca60  80 78 5f 86 08 90 af ff-48 3b 81 82 7c ca 4c 8c  .x_.....H;..|.L.
8c4cca70  f5 17 c2 91 08 90 af ff-48 2d 57 86 c4 ca 4c 8c  ........H-W...L.
8c4cca80  f4 5a 8f 82 08 90 af ff-30 90 af ff 00 00 00 00  .Z......0.......
8c4cca90  80 78 5f 86 48 3b 81 82-00 01 00 00 20 fd 96 01  .x_.H;..........
8c4ccaa0  a7 17 c2 91 00 00 00 00-30 90 af ff 08 90 af ff  ........0.......
---------
KEYLOGGER
8c4cca30  1e 00 41 80 00 00 00 00-3f 00 0b 00 00 00 1e 00  ..A.....?.......
8c4cca40  01 00 00 00 00 00 00 00-00 00 00 00 6c ca 4c 8c  ............l.L.
8c4cca50  aa 9a cf 91 00 90 af ff-08 90 af 41 01 00 00 00  ...........A....
8c4cca60  80 78 5f 86 08 90 af ff-48 3b 81 82 7c ca 4c 8c  .x_.....H;..|.L.
8c4cca70  f5 17 c2 91 08 90 af ff-48 2d 57 86 c4 ca 4c 8c  ........H-W...L.
8c4cca80  f4 5a 8f 82 08 90 af ff-30 90 af ff 00 00 00 00  .Z......0.......
8c4cca90  80 78 5f 86 48 3b 81 82-00 01 00 00 20 fd 96 01  .x_.H;..........
8c4ccaa0  a7 17 c2 91 00 00 00 00-30 90 af ff 08 90 af ff  ........0.......
---------
```

xxxProcessKeyEvent



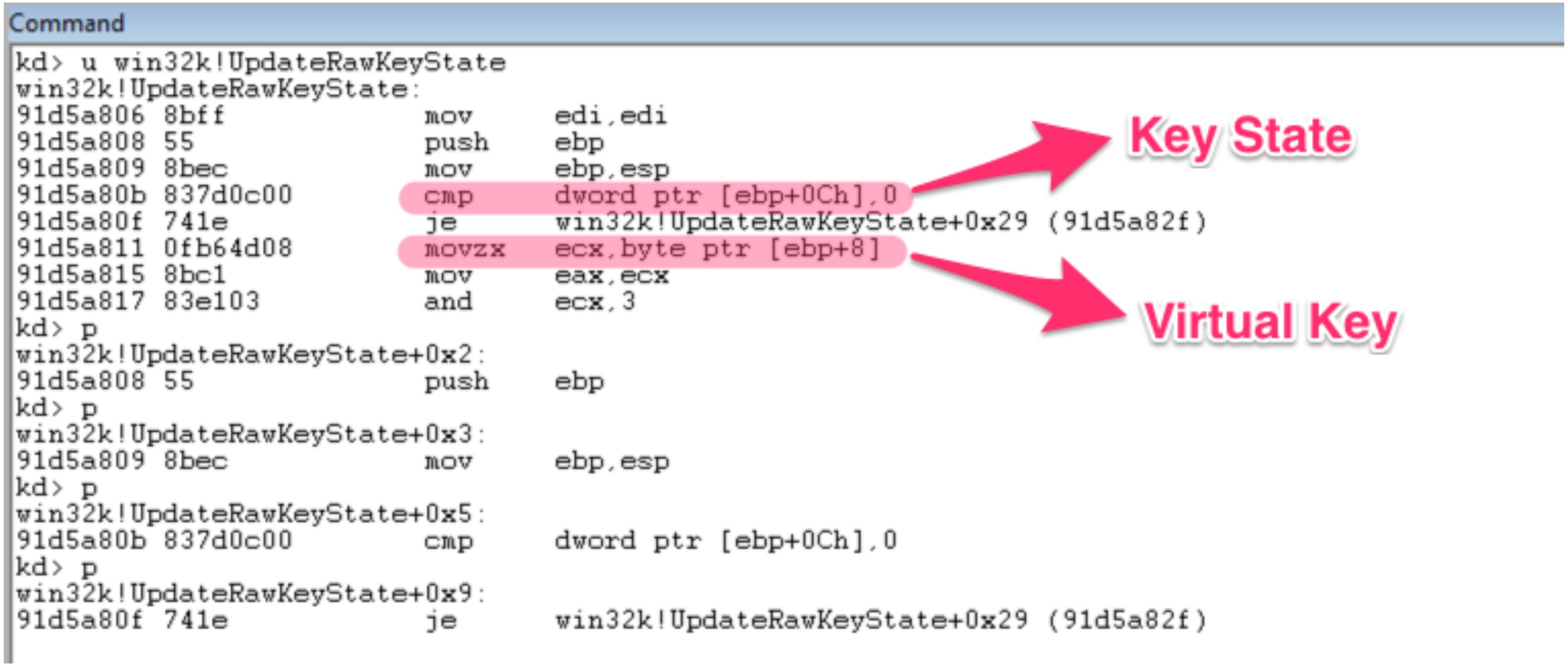
Raw Key State Table

- Just a simple array holding UP / DOWN states of keys.
- Represents the physical state of keyboard.
- Let’s put a BP on it.



Hook UpdateRawKeyState

- Two params:
  - VirtualKey
  - Key State (Make / Break)



```
Command
kd> u win32k!UpdateRawKeyState
win32k!UpdateRawKeyState:
91d5a806 8bff            mov     edi,edi
91d5a808 55              push    ebp
91d5a809 8bec            mov     ebp,esp
91d5a80b 837d0c00        cmp     dword ptr [ebp+0Ch],0
91d5a80f 741e            je      win32k!UpdateRawKeyState+0x29 (91d5a82f)
91d5a811 0fb64d08        movzx   ecx,byte ptr [ebp+8]
91d5a815 8bc1            mov     eax,ecx
91d5a817 83e103          and     ecx,3
kd> p
win32k!UpdateRawKeyState+0x2:
91d5a808 55              push    ebp
kd> p
win32k!UpdateRawKeyState+0x3:
91d5a809 8bec            mov     ebp,esp
kd> p
win32k!UpdateRawKeyState+0x5:
91d5a80b 837d0c00        cmp     dword ptr [ebp+0Ch],0
kd> p
win32k!UpdateRawKeyState+0x9:
91d5a80f 741e            je      win32k!UpdateRawKeyState+0x29 (91d5a82f)
```

#17 RawKeyState Sniffer

Sniffing Raw Key State Table

- Can be easily retrieved by disassembling UpdateRawKeyState.
- First LEA instruction points to it,
- AV buster ☺

Command

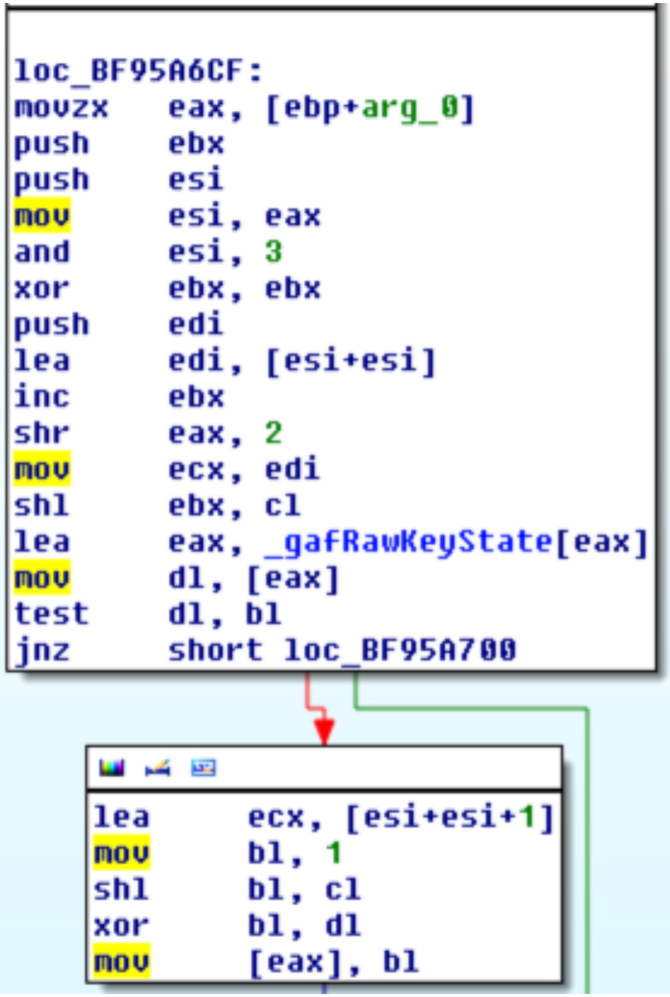
```
kd> uf win32k!UpdateRawKeyState
win32k!UpdateRawKeyState:
91d5a806 8bff            mov     edi,edi
91d5a808 55              push    ebp
91d5a809 8bec            mov     ebp,esp
91d5a80b 837d0c00        cmp     dword ptr [ebp+0Ch],0
91d5a80f 741e            je      win32k!UpdateRawKeyState+0x29 (91d5a82f)

win32k!UpdateRawKeyState+0xb:
91d5a811 0fb64d08        movzx   ecx,byte ptr [ebp+8]
91d5a815 8bc1            mov     eax,ecx
91d5a817 83e103          and     ecx,3
91d5a81a 03c9            add     ecx,ecx
91d5a81c b201            mov     dl,1
91d5a81e d2e2            shl     dl,cl
91d5a820 c1e802          shr     eax,2
91d5a823 8d808042e291    lea     eax,win32k!gafRawKeyState (91e24280)[eax]
91d5a829 f6d2            not     dl
91d5a82b 2010            and     byte ptr [eax],dl
91d5a82d eb3c            jmp     win32k!UpdateRawKeyState+0x65 (91d5a86b)
```

Here we have it!

Raw Key State Sniffer

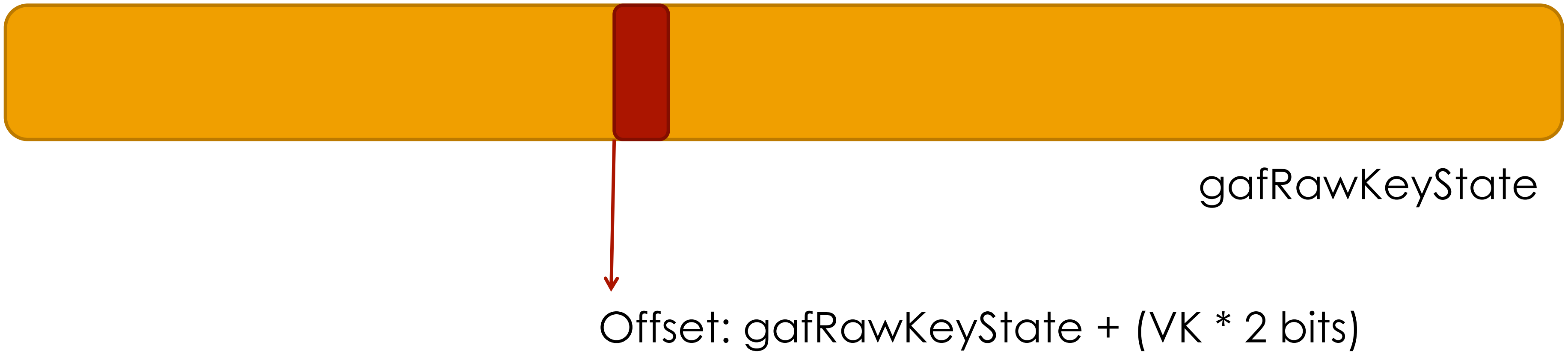
- Put a BP on UpdateRawKeyState
- 2 bits for each VKEY (Down/Up – Toggled)



Bitmap key is updated here

Raw Key State Sniffer Demo

- Put a BP on UpdateRawKeyState end address.

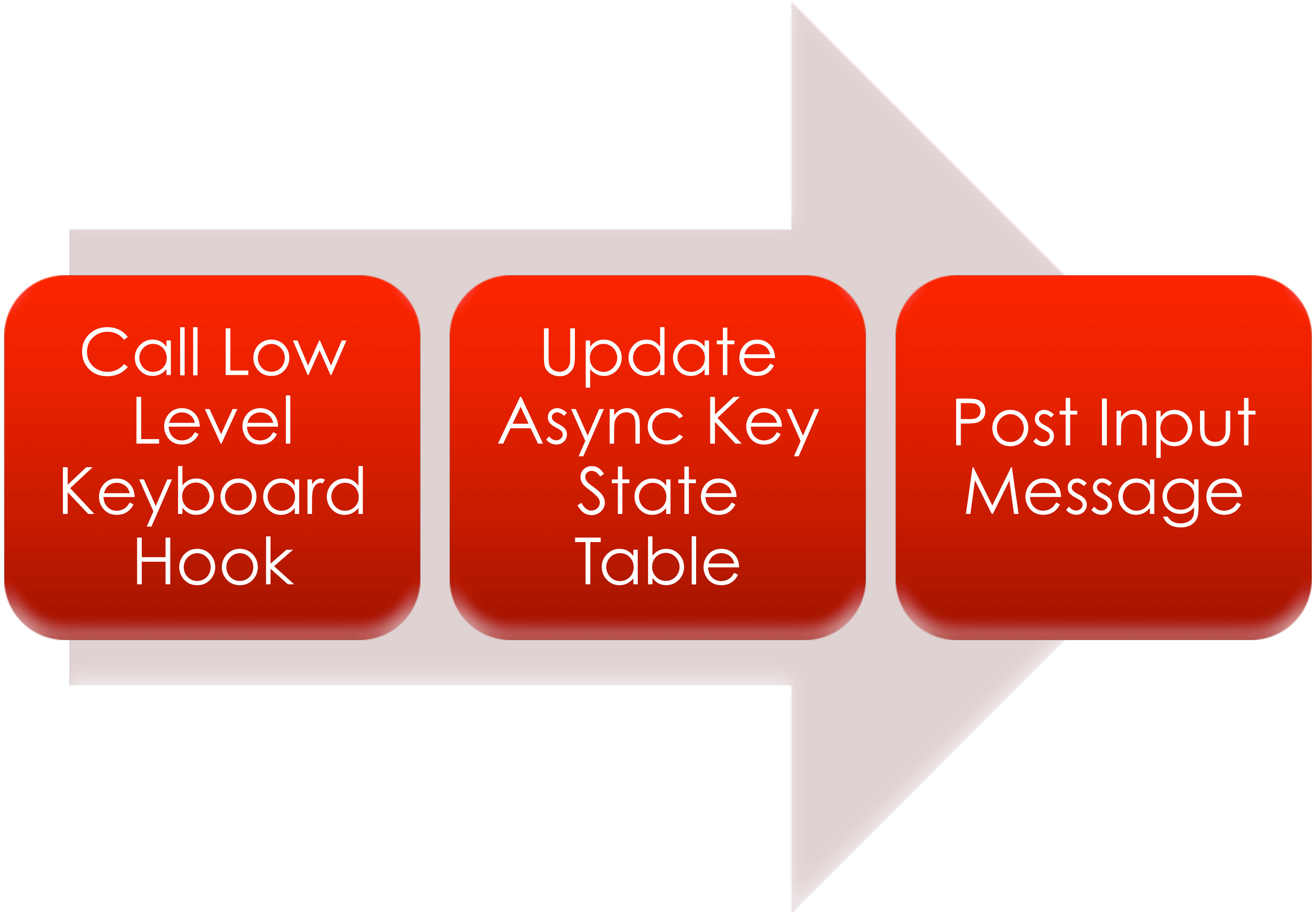


#18 Hacking xxxKeyEvent

xxxKeyEvent

- Very critical function!
- Performs the POST operation of key into input queue.
- Called by xxxProcessKeyEvent for every input event.
- Responsible from calling window hooks (wait for next slides)
- Params:
  - Virtual Key with flags,
  - ScanCode

xxxKeyEvent



xxxKeyEvent

- Very critical function!
- Performs the POST operation of key into input queue.
- Called by xxxProcessKeyEvent for every input event.
- Responsible of calling window hooks (wait for next slides)
- Params:
  - Virtual Key with flags,
  - ScanCode

#19 Hacking UpdateAsyncKeyState

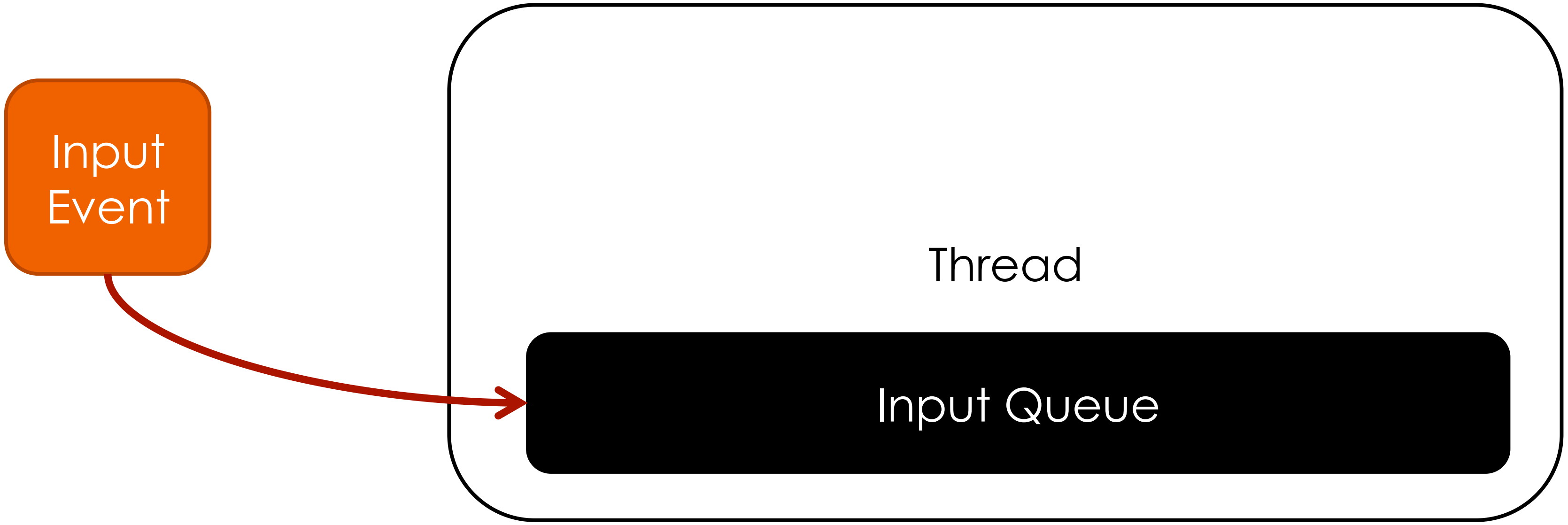
UpdateAsyncKeyState

- Looks same as the method for UpdateRawKeyState
- Async keystate table could also be sniffed.

#20 Hacking PostInputMessage

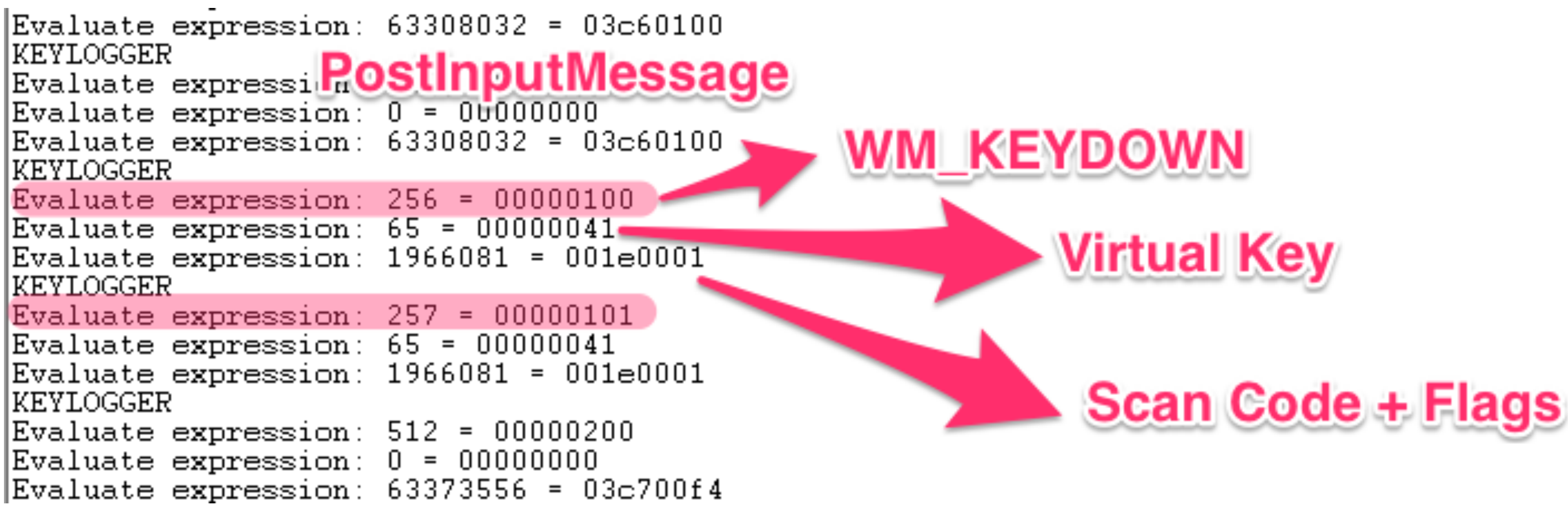
PostInputMessage

- What it does?
- Calls StoreQMessage for saving the message into queue. Another target for hooking ☺
- Foreground thread queue receives the input event.



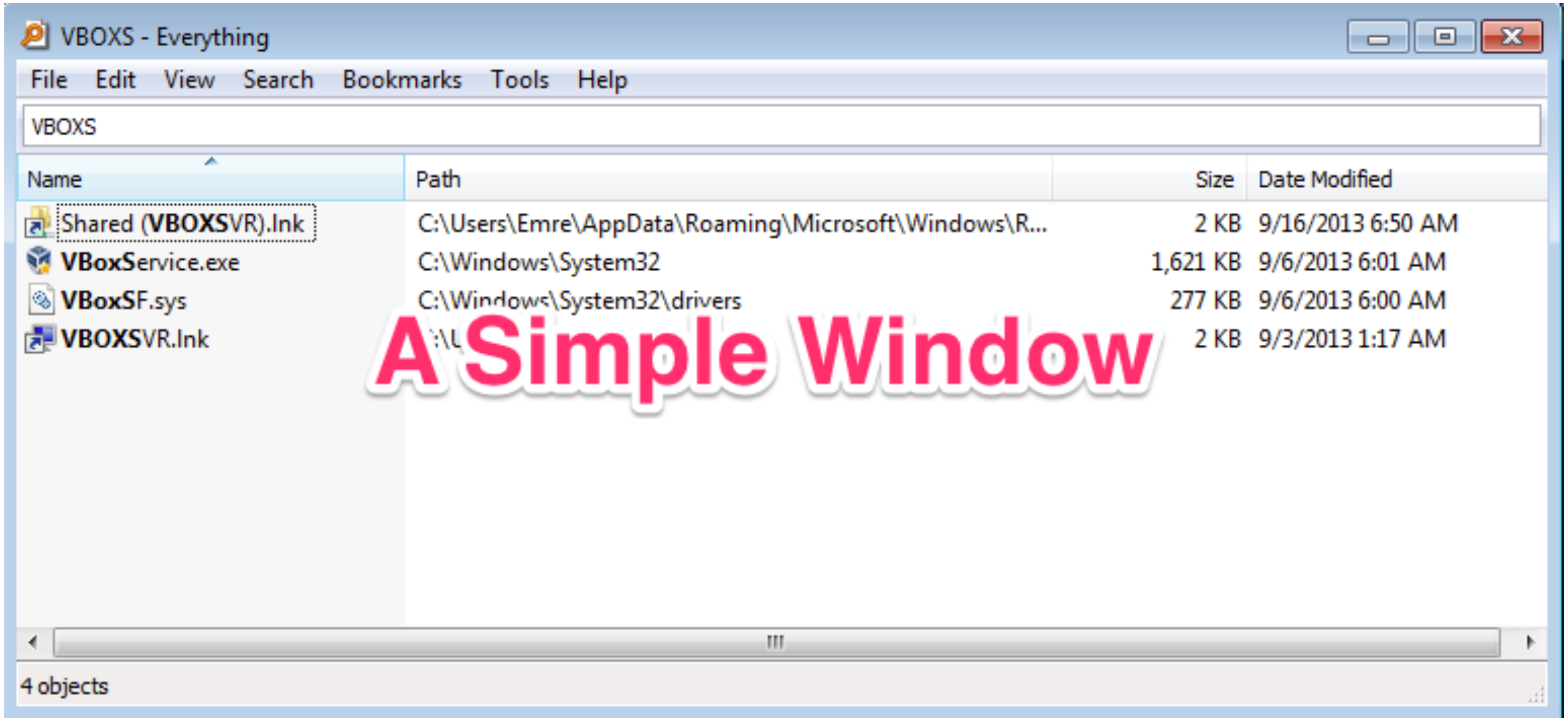
PostInputMessage

- Put a BP on PostInputMessage.



Here comes the second part ☺

- Thread now has an input event in it's queue. Kernel is over!
- What's next?



Create Window API

- Creates a window with a Window Class.
- What is a window class?

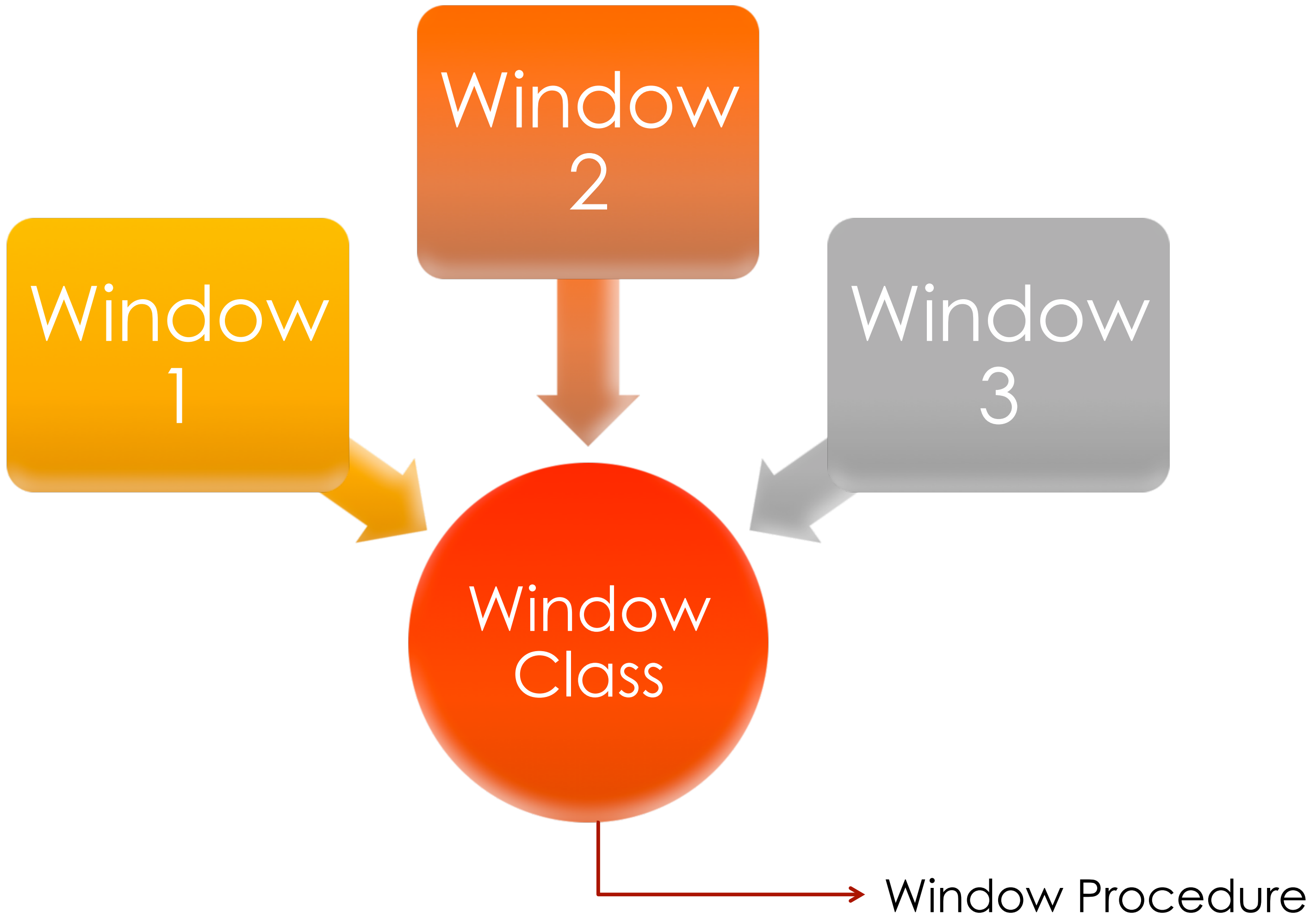
CreateWindowEx function

Creates an overlapped, pop-up, or child window with an extended window style; otherwise, this function is identical to the CreateWindow function. For more information about creating a window and for full descriptions of the other parameters of **CreateWindowEx**, see **CreateWindow**.

```
typedef struct tagWNDCLASS {
  UINT      style;
  WNDPROC   lpfnWndProc;
  int       cbClsExtra;
  int       cbWndExtra;
  HINSTANCE hInstance;
  HICON     hIcon;
  HCURSOR   hCursor;
  HBRUSH    hbrBackground;
  LPCTSTR   lpszMenuName;
  LPCTSTR   lpszClassName;
} WNDCLASS, *PWNDCLASS;
```

WNDPROC ???

Classes vs. Windows



WNDPROC Function

- Function defined as:

```
LRESULT CALLBACK WindowProc(
  _In_  HWND hwnd,
  _In_  UINT uMsg,
  _In_  WPARAM wParam,
  _In_  LPARAM lParam
);
```

- Every window has one WNDPROC. This is the entry point for window messages.

#21 Hacking Window Procedures

WNDPROC Function

- We can either inline hook the WndProc or we can set a new WndProc by using GetWindowLong / SetWindowLong APIs.

C++

```
LONG WINAPI GetWindowLong(
  _In_  HWND hWnd,
  _In_  int nIndex
);
```

GWL_WNDPROC
DWL_DLGPROC

#22 Subclassing a Window

Subclassing

- MSDN Blog: When you subclass a window, you set the window procedure to a function of your choosing, and you remember the original window procedure so you can pass it to the CallWindowProc function when your subclass function wants to pass the message to the original window procedure.

Window Class

→ Window Subclass Procedure (Log The Keystroke)

↓

Call The Original Window Procedure

Subclassing

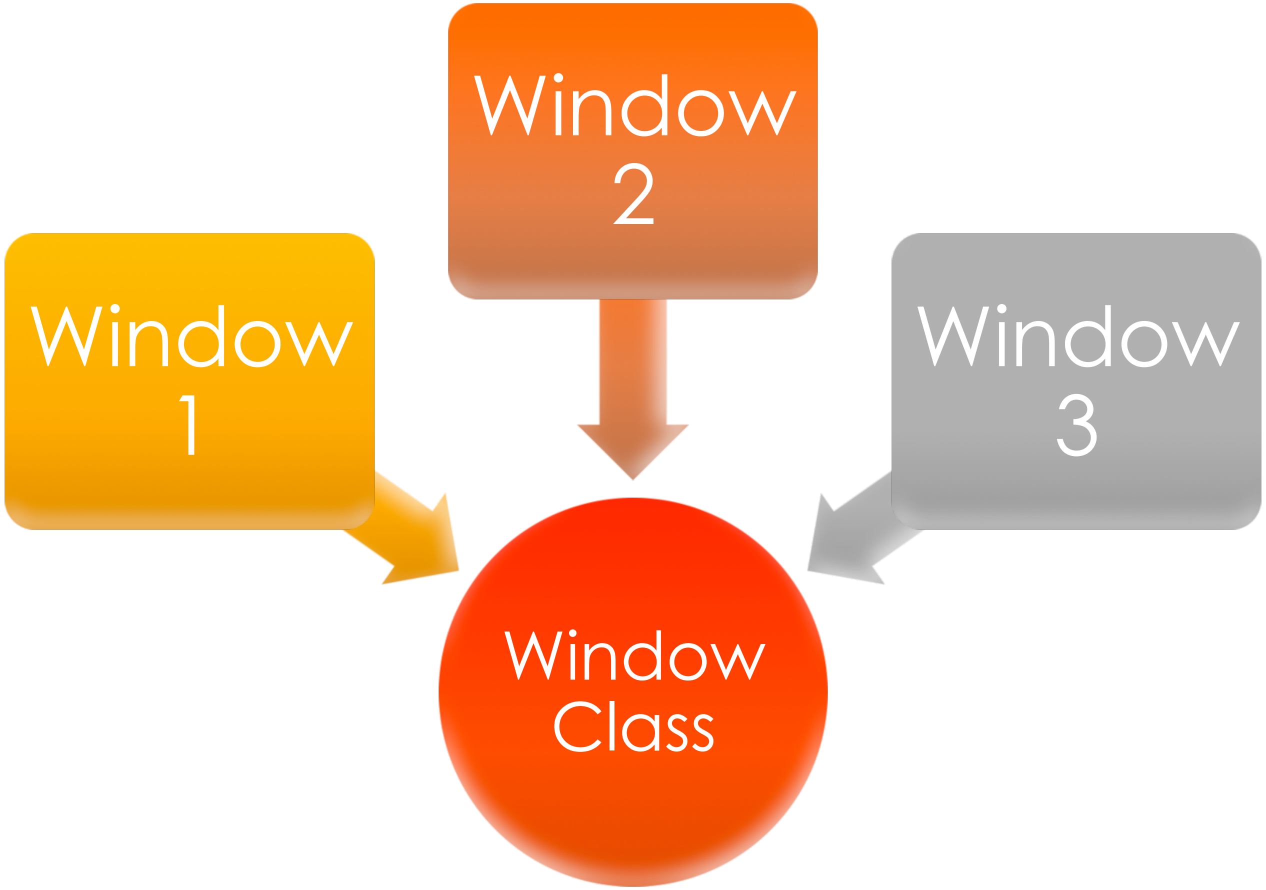
- SetWindowSubclass API is pretty good for that.
- CallWndProc could be used for retrieving keys from subclassed windows.

C++

```
BOOL SetWindowSubclass(
  _In_  HWND hWnd,
  _In_  SUBCLASSPROC pfnSubclass,
  _In_  UINT_PTR uIdSubclass,
  _In_  DWORD_PTR dwRefData
);
```

Keylogger Proc

Classes vs. Windows



Message Loops

- Each UI Thread has one message loop for processing window messages.
- http://msdn.microsoft.com/en-us/library/windows/desktop/ms644928(v=vs.85).aspx

```
while( (bRet = GetMessage( &msg, NULL, 0, 0 )) != 0)   1
{
    if (bRet == -1)
    {
        // handle the error and possibly exit
    }
    else
    {
        TranslateMessage(&msg);   2
        DispatchMessage(&msg);    3
    }
}
```

#23 Hacking GetMessage / PeekMessage

GetMessage / PeekMessage

- Used for getting a message from the thread's message queue.

```
BOOL WINAPI GetMessage(
  _Out_     LPMSG lpMsg,
  _In_opt_  HWND hWnd,
  _In_      UINT wMsgFilterMin,
  _In_      UINT wMsgFilterMax
);
```

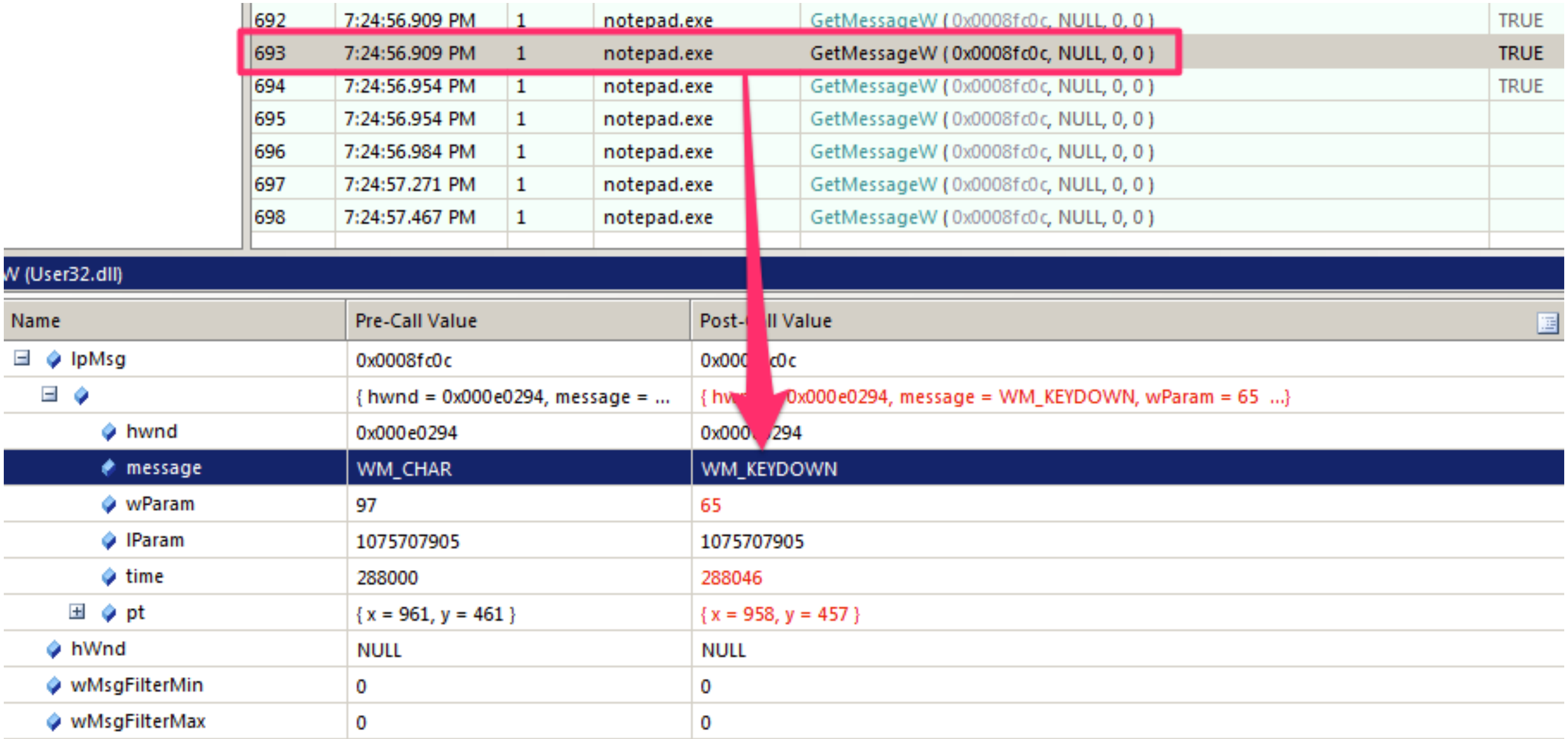
→ Blocking

```
BOOL WINAPI PeekMessage(
  _Out_     LPMSG lpMsg,
  _In_opt_  HWND hWnd,
  _In_      UINT wMsgFilterMin,
  _In_      UINT wMsgFilterMax,
  _In_      UINT wRemoveMsg
);
```

→ Non-Blocking

GetMessage / PeekMessage

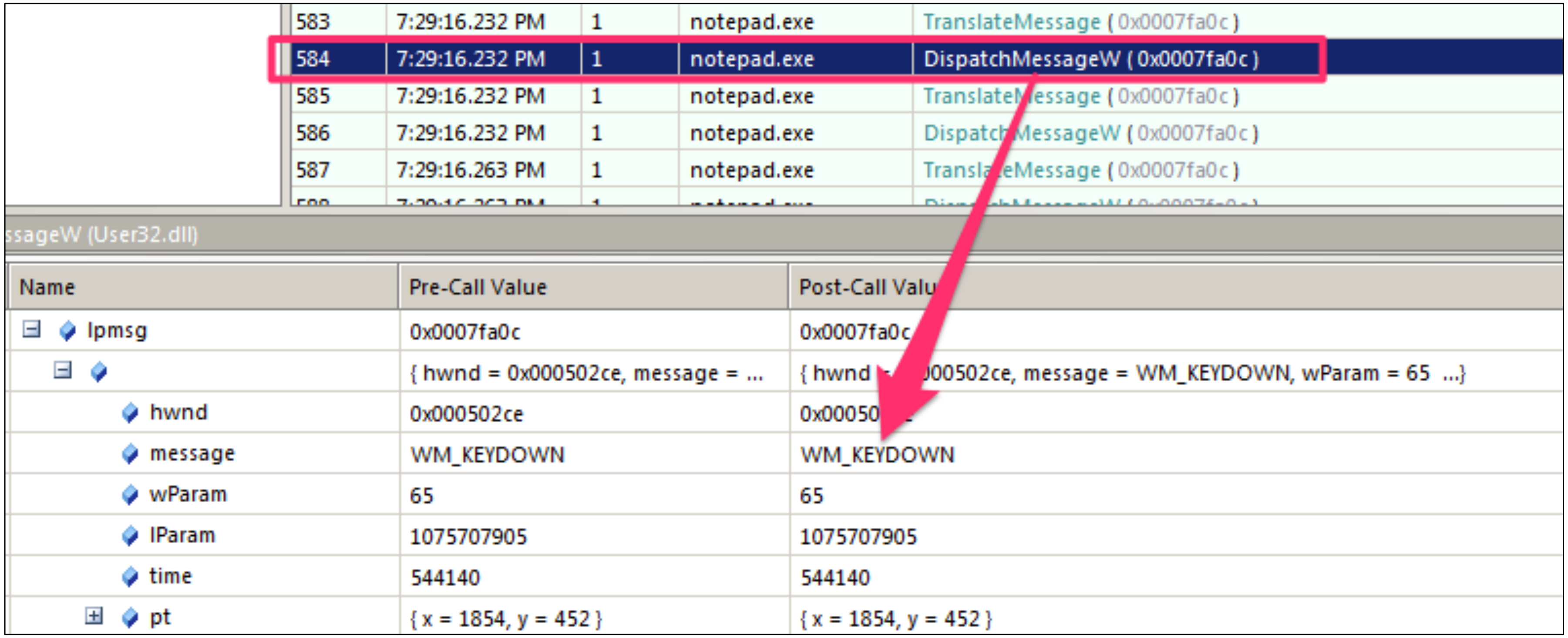
- Sniff GetMessage API call.



#24 Hacking Translate and Dispatch

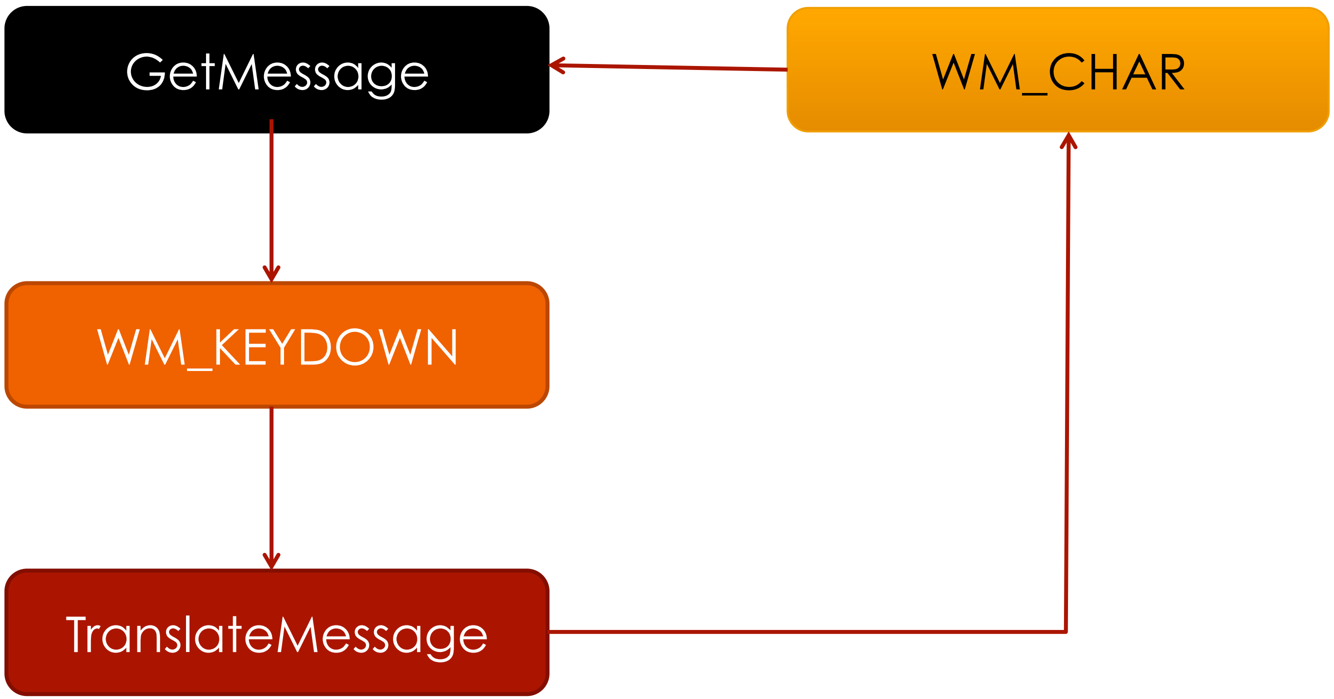
TranslateMessage / DispatchMessage

- Sniff TranslateMessage / DispatchMessage API calls.



TranslateMessage

- Translate to what?



DispatchMessage

- Calls the Window Procedure of a Window Class.
- Hooking it will definitely give you a lot power.

#25 Hacking Counterparts

Kernel Mode Counterparts

- The APIs which are used for message handling and delivering such as DispatchMessage,GetMessage, PeekMessage.
- All of them have their kernel mode counterparts starting with NtUser*. NtUserGetMessage, NtUserPeekMessage, NtUserTranslateMessage.
- These could be inline hooked by kernel mode keyloggers.
- Best example for this is “Elite Keylogger” (newest versions)
- Pretty effective!

Inspecting Kernel Mode Counterparts

- Anti-rootkits such as GMER, KernelDetective or Tuluka could be used for detecting these kind of modifications.

437	NtUserGetListBoxInfo	0x9283DC2D	0x9283DC2D	-	C:\Windows\System32\win32k.sys
438	NtUserGetMenuBarInfo	0x9283B634	0x9283B634	-	C:\Windows\System32\win32k.sys
439	NtUserGetMenuIndex	0x9283E19B	0x9283E19B	-	C:\Windows\System32\win32k.sys
440	NtUserGetMenuItemRect	0x927D191E	0x927D191E	-	C:\Windows\System32\win32k.sys
441	NtUserGetMessage	0x927AB7E7	0x927AB7E7	-	C:\Windows\System32\win32k.sys
442	NtUserGetMouseMovePointsEx	0x9283E8D1	0x9283E8D1	-	C:\Windows\System32\win32k.sys
443	NtUserGetObjectInformation	0x9275E06C	0x9275E06C	-	C:\Windows\System32\win32k.sys
444	NtUserGet[illegible]	[illegible]	[illegible]	[illegible]	[illegible]in32k.sys
445	NtUserGetPriorityClipboardFormat	0x9283E4A8	0x9283E4A8	-	C:\Windows\System32\win32k.sys
446	NtUserGetProcessWindowStation	0x927682E7	0x927682E7	-	C:\Windows\System32\win32k.sys
447	NtUserGetRawInputBuffer	0x928418D3	0x928418D3	-	C:\Windows\System32\win32k.sys
448	NtUserGetRawInputData	0x92841309	0x92841309	-	C:\Windows\System32\win32k.sys

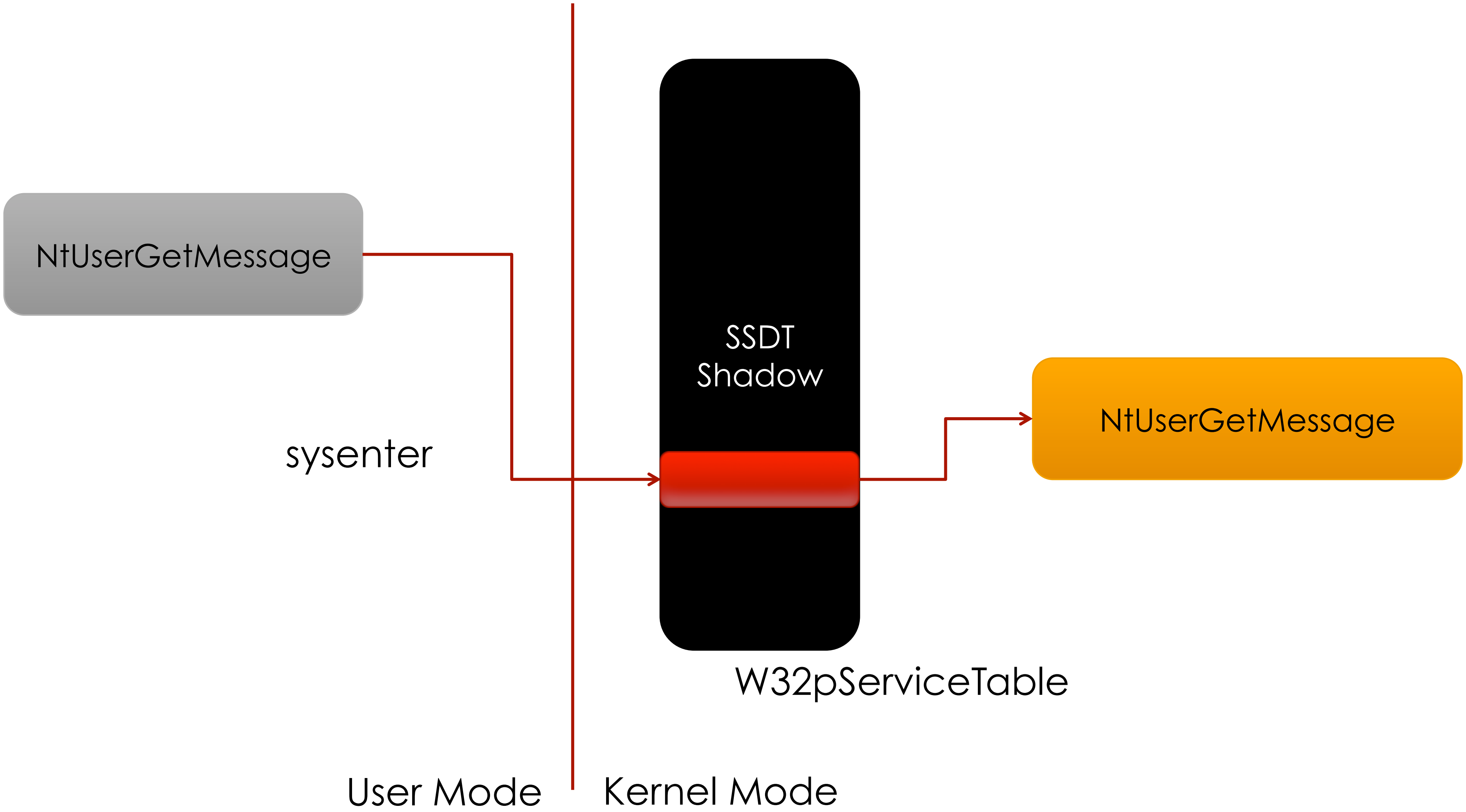
Should be within the limits of win32k.sys address space

#26 SSDT Shadow Hooking

What is SSDT Shadow?

- Just a simple table residing in win32k.sys module.
- Holds the addresses of system services.
- This table is the glue between user mode APIs and the kernel mode counterparts.
- Hooking this table is so easy, and also effective.

How it is used?



How to check?

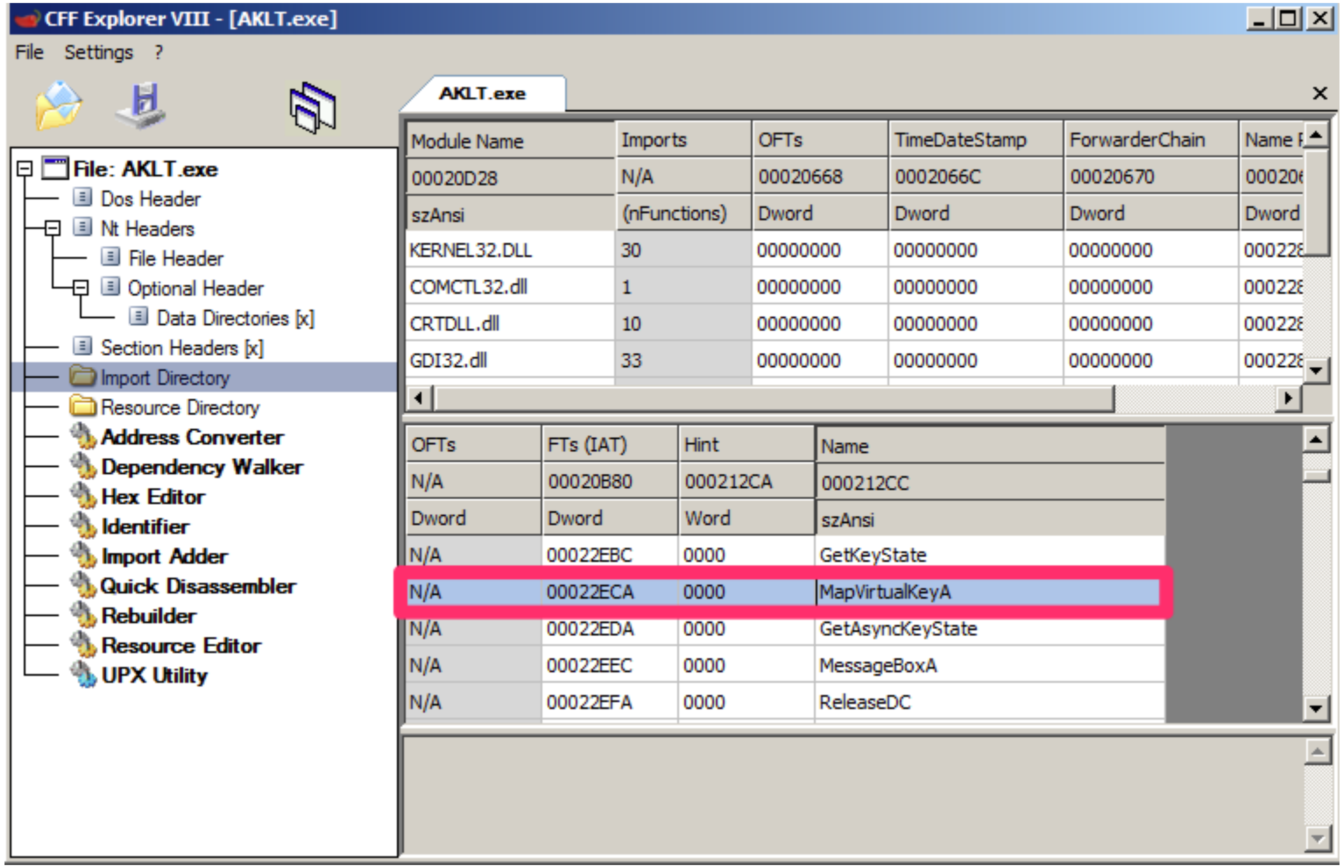
- We can use anti-rootkits
- Windbg can also be used for displaying SSDT Shadow Table.

```
kd> dps win32k!W32pServiceTable L200
9234b000  922d7b91 win32k!NtGdiAbortDoc
9234b004  922efae8 win32k!NtGdiAbortPath
9234b008  92147216 win32k!NtGdiAddFontResourceW
9234b00c  922e6aff win32k!NtGdiAddRemoteFontToDC
9234b010  922f1296 win32k!NtGdiAddFontMemResourceEx
9234b014  922d83ae win32k!NtGdiRemoveMergeFont
9234b018  922d8442 win32k!NtGdiAddRemoteMMInstanceToDC
9234b01c  921ff7fb win32k!NtGdiAlphaBlend
9234b020  922f0ac1 win32k!NtGdiAngleArc
9234b02[illegible]
9234b02[illegible]
9234b02c  922f2d94 win32k!NtGdiArcInternal
9234b030  922f0fb2 win32k!NtGdiBeginGdiRendering
9234b034  922efb5c win32k!NtGdiBeginPath
9234b038  921f45cb win32k!NtGdiBitBlt
9234b03c  922f0f05 win32k!NtGdiCancelDC
9234b040  922f3b38 win32k!NtGdiCheckBitmapBits
9234b044  922efa63 win32k!NtGdiCloseFigure
9234b048  9222686a win32k!NtGdiClearBitmapAttributes
9234b04c  922f103c win32k!NtGdiClearBrushAttributes
9234b050  922f352c win32k!NtGdiColorCorrectPalette
9234b054  921b3ca5 win32k!NtGdiCombineRgn
9234b058  9225ad3d win32k!NtGdiCombineTransform
```

dps win32k!W32pServiceTable

Conversion Functions

- MapVirtualKey / MapVirtualKeyEx
- ToAscii / ToAsciiEx
- VkKeyScan / VkKeyScanEx



#27 GetKeyState / GetAsyncKeyState

GetKeyState / GetAsyncKeyState

- APIs for determining the state of a key at some point time.
- Difference is:
  - GetKeyState is more specific and doesn’t reflect the interrupt-level state information,
  - GetAsyncKeyState reflects the interrupt-level state of keys.
- One of the most widely used technique by keyloggers.

#28 GetKeyboardState

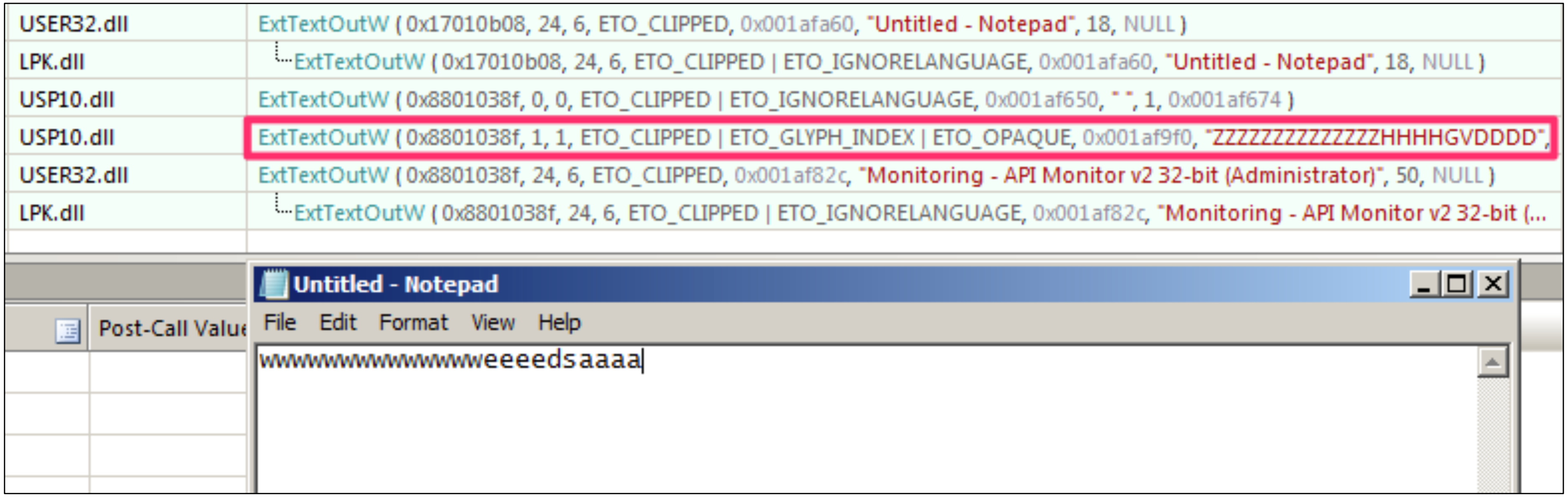
GetKeyboardState

- API for determining the state of a keyboard.
- Fills an array of virtual keys.
- One of the most widely used method used by keyloggers.

#29 Text Output APIs

Text Output APIs

- APIs used by applications to output text.
- Examples:
  - TextOut
  - ExtTextOut
  - DrawText / DrawTextEx



#30 GetWindowText

GetWindowText

- Can be used within an injected thread.
- Copies the text of the specified window's title bar (if it has one) into a buffer. If the specified window is a control, the text of the control is copied. However, **GetWindowText** cannot retrieve the text of a control in **another** application.

#31 WM_GETTEXT Message

WM_GETTEXT Message

- Can be used for retrieving another applications window content.

C++

```
#define WM_GETTEXT                          0x000D
```

Parameters

wParam

The maximum number of characters to be copied, including

ANSI applications may have the string in the buffer reduced i
from ANSI to Unicode.

lParam

A pointer to the buffer that is to receive the text.

#32 SetWindowsHookEx

SetWindowHookEx

- Another term for saying "Keylogger" ☺
- Definitely the MOST WIDELY USED technique for keylogging!!!
- Nearly %95 of keyloggers use it ☺

C++

```
HHOOK WINAPI SetWindowsHookEx(
  _In_  int idHook,
  _In_  HOOKPROC lpfn,
  _In_  HINSTANCE hMod,
  _In_  DWORD dwThreadId
);
```

Why?

- It is a way for providing callbacks to developers but widely used by malware authors.
- Have pretty much variations such as “Low Level Hook”, “Get Message Hook” and etc.

Hook Types

- **WH_CALLWNDPROC :** Installs a hook procedure that monitors messages before the system sends them to the destination window procedure.
- **WH_CALLWNDPROCRET :** Installs a hook procedure that monitors messages after they have been processed by the destination window procedure.
- **WH_CBT :** Installs a hook procedure that receives notifications useful to a Computer Based Training (CBT) application.
- **WH_DEBUG :** Installs a hook procedure useful for debugging other hook procedures.

Hook Types

- **WH_GETMESSAGE :** Installs a hook procedure that monitors messages posted to a message queue.
- **WH_JOURNALRECORD :** Installs a hook procedure that records input messages posted to the system message queue.
- **WH_KEYBOARD :** Installs a hook procedure that monitors keystroke messages.
- **WH_KEYBOARD_LL :** Installs a hook procedure that monitors low-level keyboard input events.

Low Level Hooks

- Starting from this slide
- What is a Hook Function?
- Only low level hooks are allowed in Raw Input Thread.
- Ability to block some input events using these hooks.
- Will be described separately.

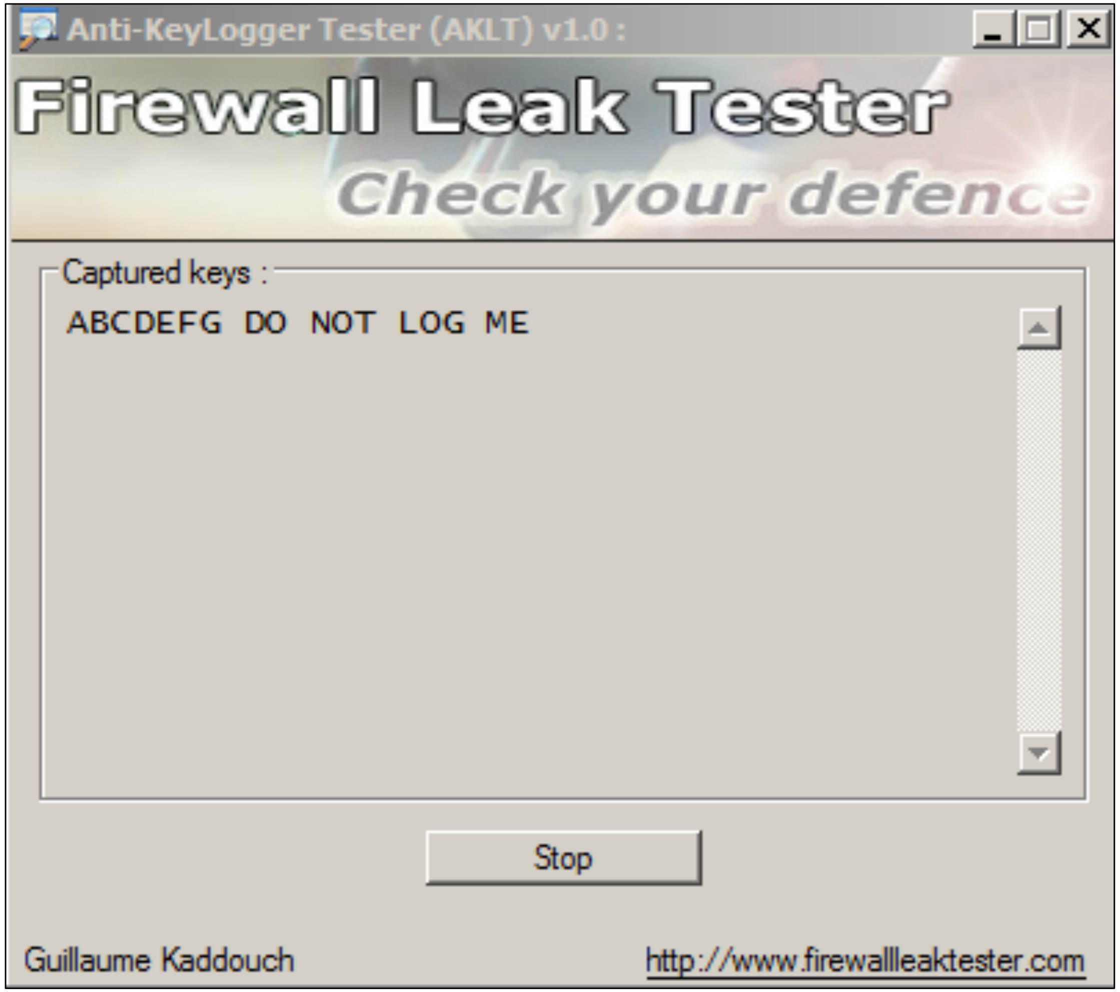
  - WH_CALLWNDPROC and WH_CALLWNDPROCRET
  - WH_CBT
  - WH_DEBUG
  - WH_FOREGROUNDIDLE
  - WH_GETMESSAGE
  - WH_JOURNALPLAYBACK
  - WH_JOURNALRECORD
  - WH_KEYBOARD_LL
  - WH_KEYBOARD
  - WH_MOUSE_LL
  - WH_MOUSE
  - WH_MSGFILTER and WH_SYSMSGFILTER
  - WH_SHELL

Hook Types

#33 DirectX Keylogger

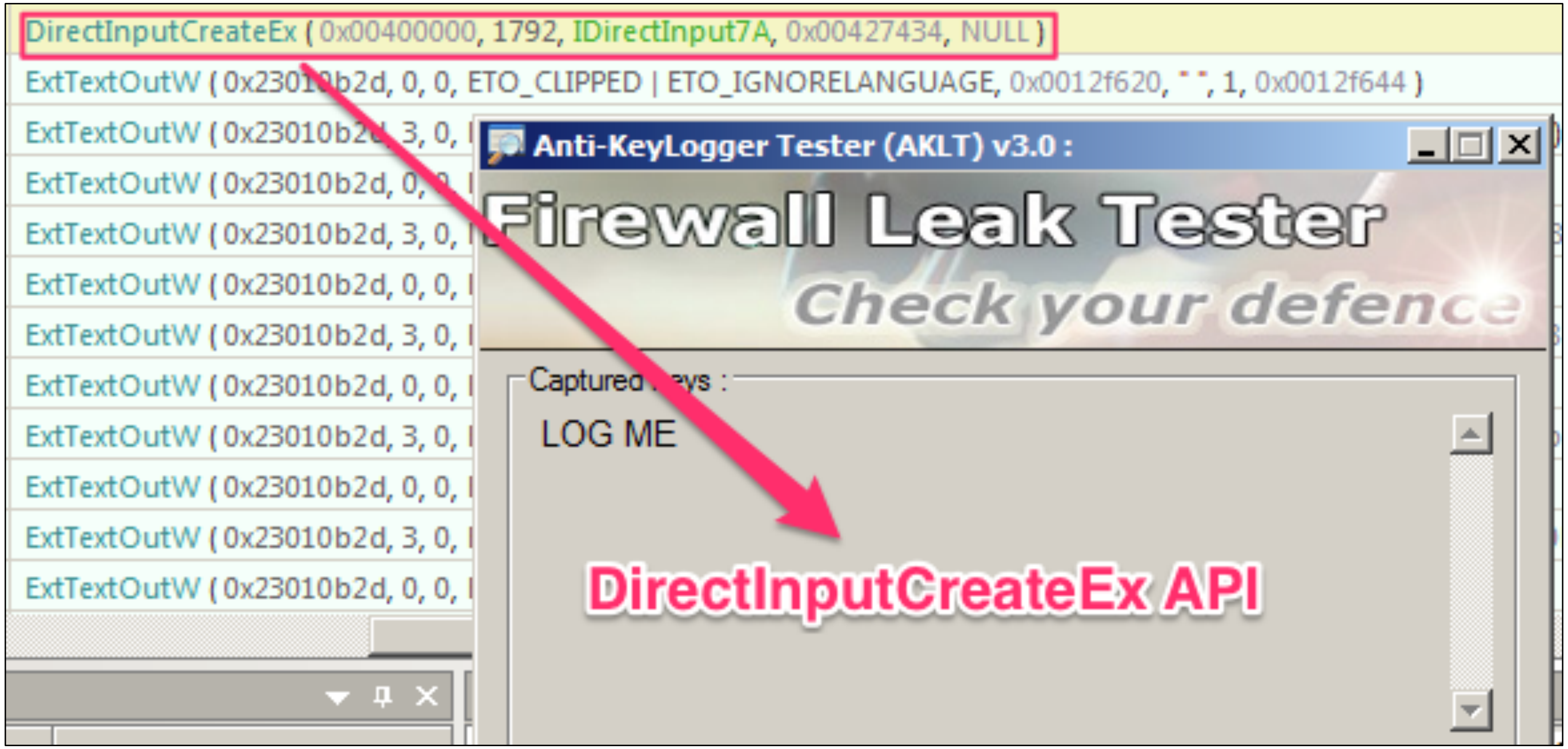
DirectX

- Not widely used but a good way for logging keystrokes.



How?

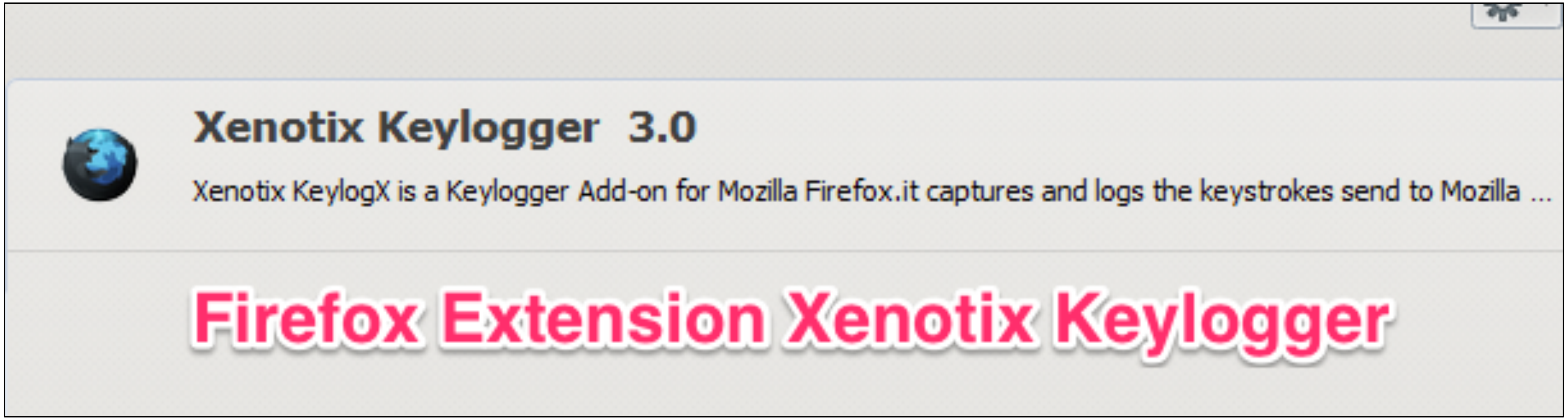
- Pretty easy to implement with DirectInputCreateEx API.
- CreateDevice API is used for keyboard device creation.



#34 Browser Extensions

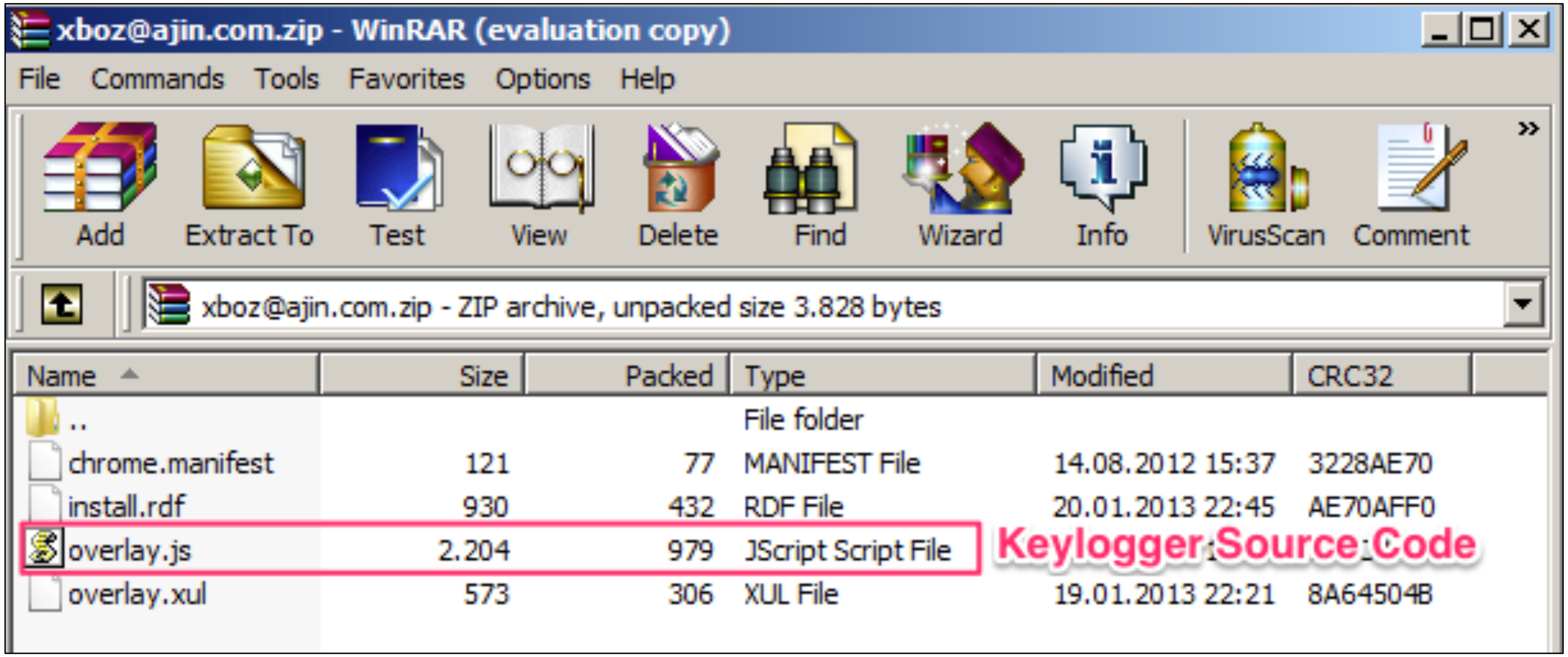
Browser Extensions

- Sneaky creatures!
- Not widely used but a great for bypassing security measures.



Inspecting

- XPI files are just zip files.
- Unzip it and analyze what it does.



Demos

- Go to Materials/Keyloggers folder:
  - Analyze martin.exe
  - Analyze AKLT_3.0.exe
  - Analyze refog_personal_manager.exe”
  - Analyze Elite Keylogger
  - Analyze java keylogger
  - Analyze Free Keylogger

Thanks

Questions?