Microsoft Virtual PC: The Hyper-Hole-Visor bug
&
MS10-048: Win32k Window Creation Vulnerability
(CVE-2010-1897)

Nicolás A. Economou – (**nicolas.economou**@*coresecurity.com*)

MS10-048

Win32k Window Creation Vulnerability

- 06/08/2010 Microsoft patched "win32k.sys"

- **Microsoft Security Bulletin MS10-032**: Vulnerabilities in Windows Kernel-Mode Drivers Could Allow Elevation of Privilege (979559)

- A fix was done in a **wrong way**

Microsoft Security Bulletin MS10-032 - Important

Vulnerabilities in Windows Kernel-Mode Drivers Could Allow Elevation of Privilege (979559)

Published: June 08, 2010

Vulnerability Information

- ⊞ Severity Ratings and Vulnerability Identifiers
- ⊞ Win32k Improper Data Validation Vulnerability - CVE-2010-0484
- ⊞ Win32k Window Creation Vulnerability - CVE-2010-0485
- ⊞ Win32k TrueType Font Parsing Vulnerability - CVE-2010-1255

- **Windows Vulnerable Versions: ALL**

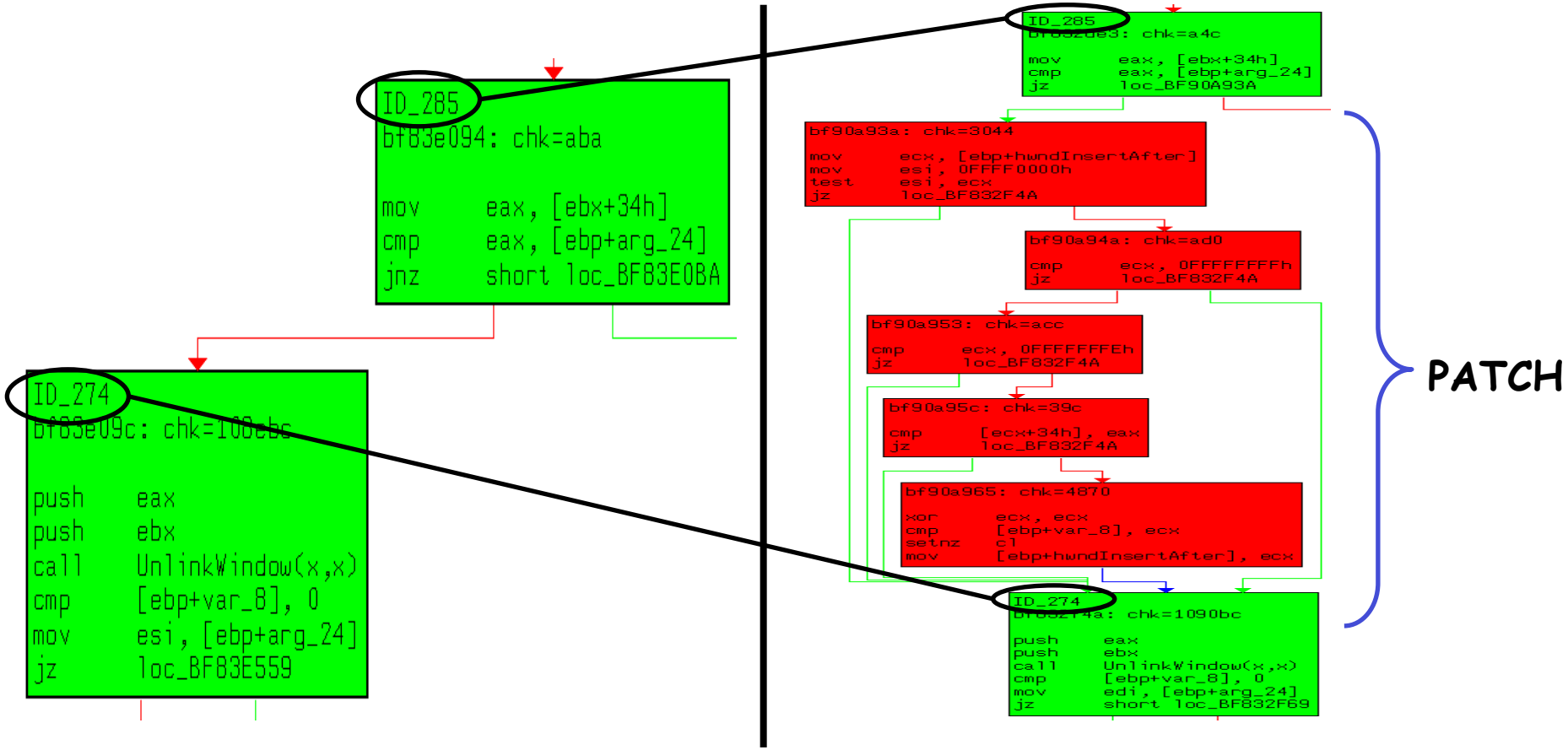
- **Diffing "win32k.sys" "Windows XP SP3"**
  - Using **turbodiff** to compare
  - **"Win32k.sys" v5.1.2600.5863**

VS

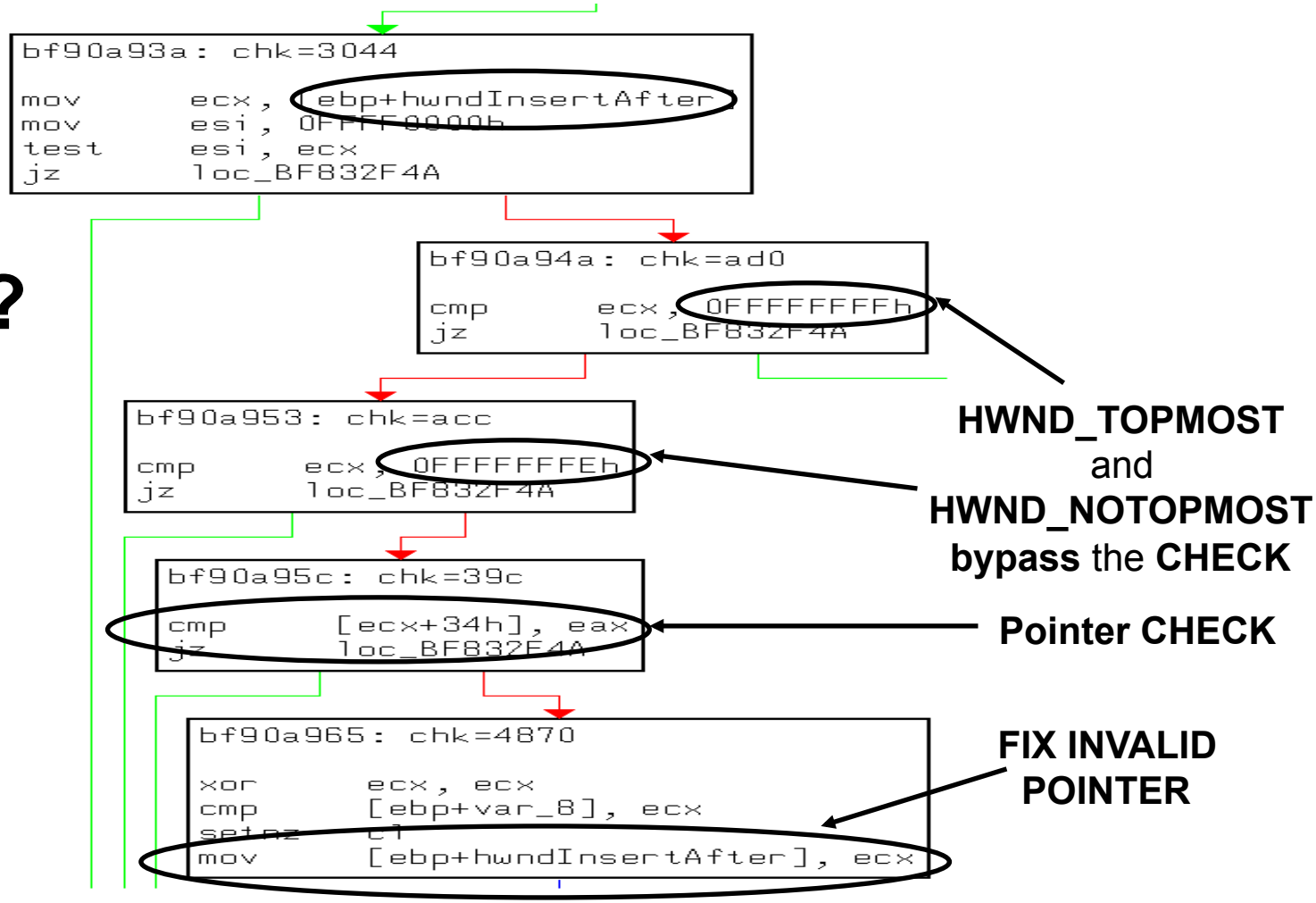
  - **"Win32k.sys" v5.1.2600.5976**

- **Changed Functions: 10**
  - bGeneratePath
  - cjFillPolygon ()
  - ConvertRedirectionDCs ()
  - DestroyCacheDC ()
  - GetDCEx ()
  - RecreateRedirectionBitmap
  - UserGetHwnd
  - xxxCreateWindowEx ()  !!
  - xxxFreeWindow ()
  - xxxSetParent ()

- **"xxxCreateWindowEx" function patch**



What was wrong ?



MS10-032 → MS10-048

Microsoft Security Bulletin MS10-048 - Important

Vulnerabilities in Windows Kernel-Mode Drivers Could Allow Elevation of Privilege (2160329)

Published: August 10, 2010

Vulnerability Information

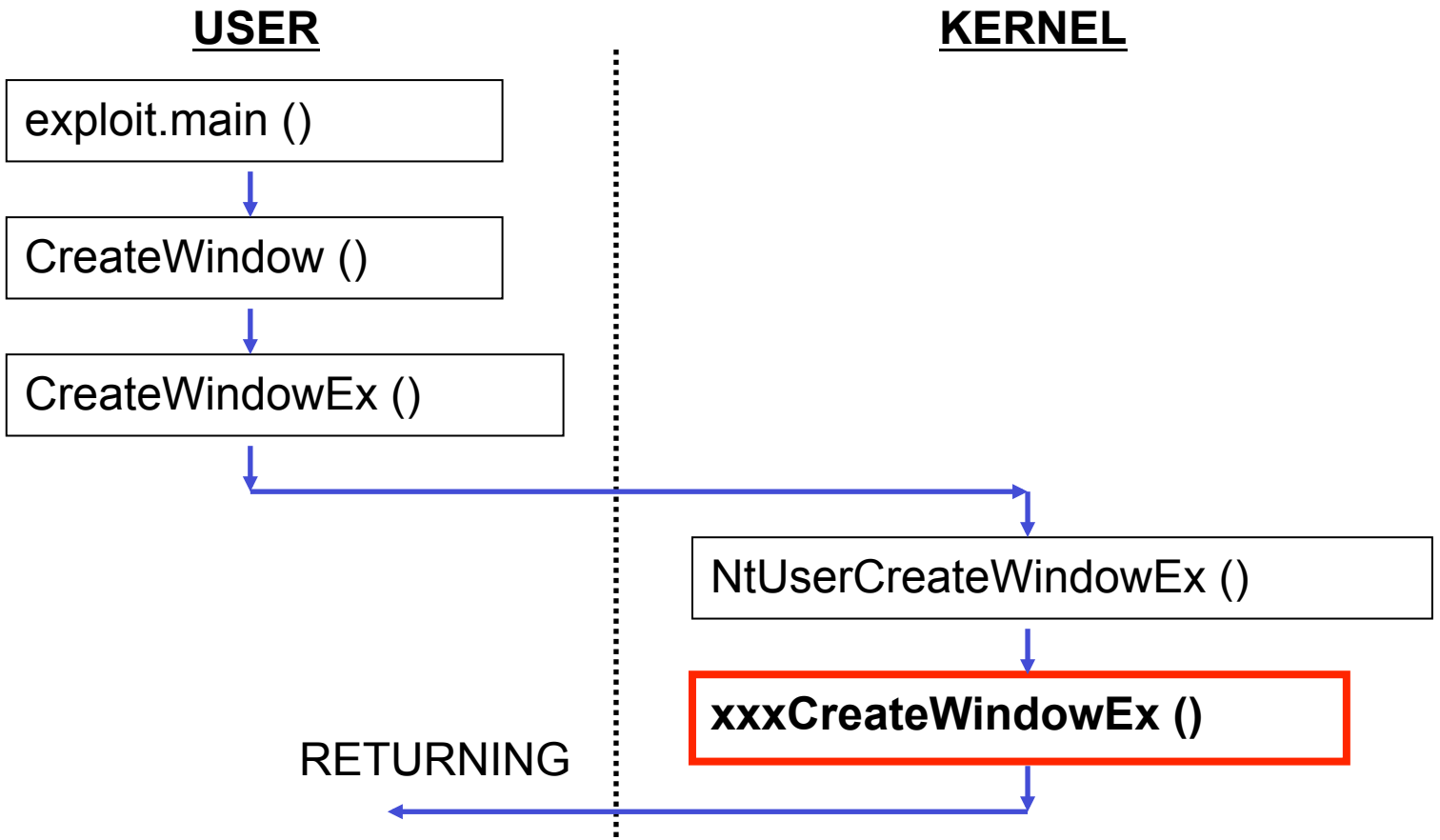
- ⊞ Severity Ratings and Vulnerability Identifiers
- ⊞ Win32k Bounds Checking Vulnerability - CVE-2010-1887
- ⊞ Win32k Exception Handling Vulnerability - CVE-2010-1894
- ⊞ Win32k Pool Overflow Vulnerability - CVE-2010-1895
- ⊞ Win32k User Input Validation Vulnerability - CVE-2010-1896
- ⊞ Win32k Window Creation Vulnerability - CVE-2010-1897

- **Windows Vulnerable Versions: ALL**

- **"CreateWindow" function prototype**

```
HWND WINAPI CreateWindow(
        __in_opt LPCTSTR lpClassName,
        __in_opt LPCTSTR lpWindowName,
        __in DWORD dwStyle,
        __in int x,
        __in int y,
        __in int nWidth,
        __in int nHeight,
        __in_opt HWND hWndParent,
        __in_opt HMENU hMenu,
        __in_opt HINSTANCE hInstance,
        __in_opt LPVOID lpParam
);
```

"CreateWindow" Function Callgraph



Changing the Control Flow …

- Registering a CALLBACK

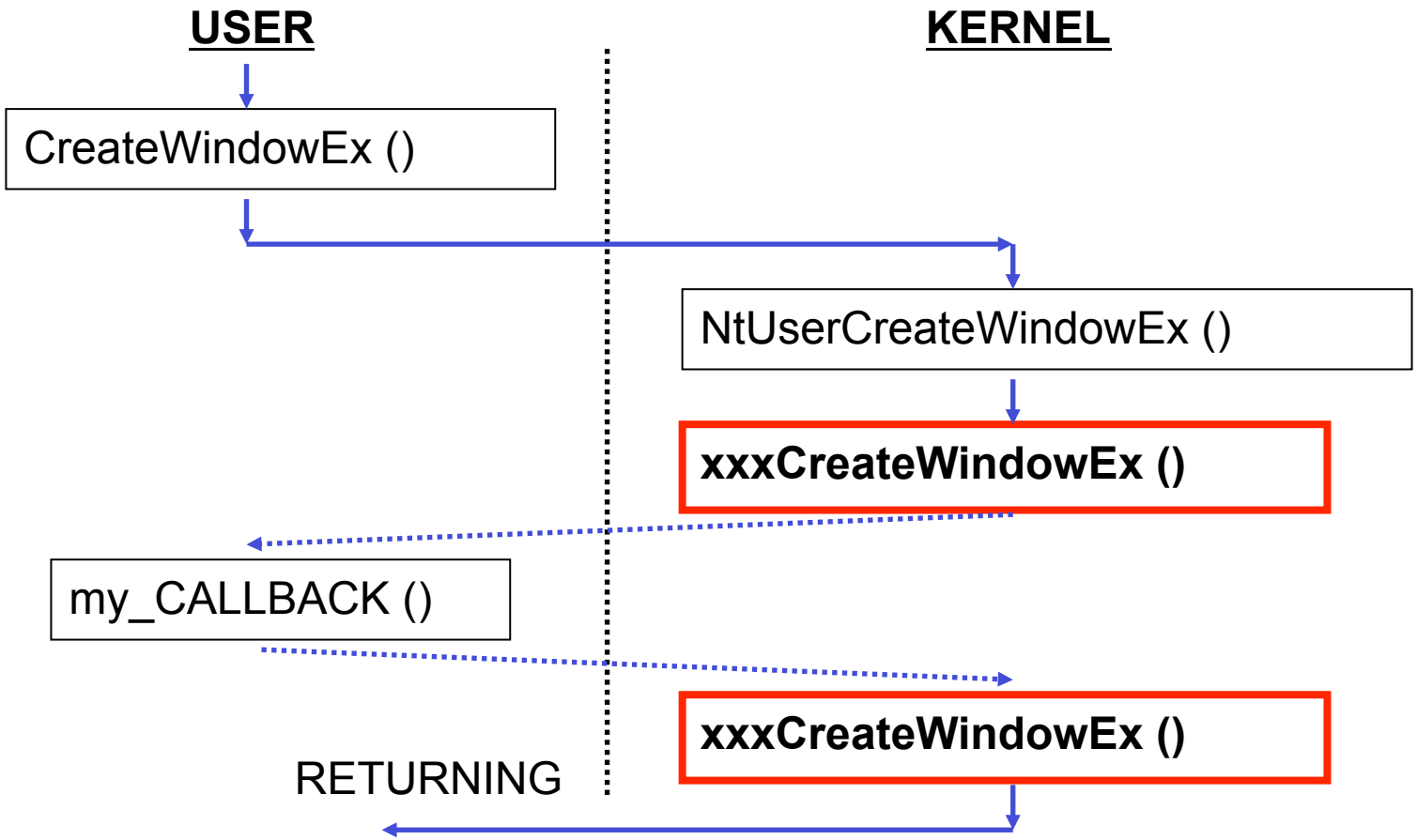
```
HHOOK WINAPI SetWindowsHookEx(
    __in int idHook,
    __in HOOKPROC lpfn,
    __in HINSTANCE hMod,
    __in DWORD dwThreadId);

LRESULT CALLBACK CallWndProc(
    __in int nCode,
    __in WPARAM wParam,

    __in LPARAM lParam );
```

!!

- **"CreateWindow" + registered Callback**

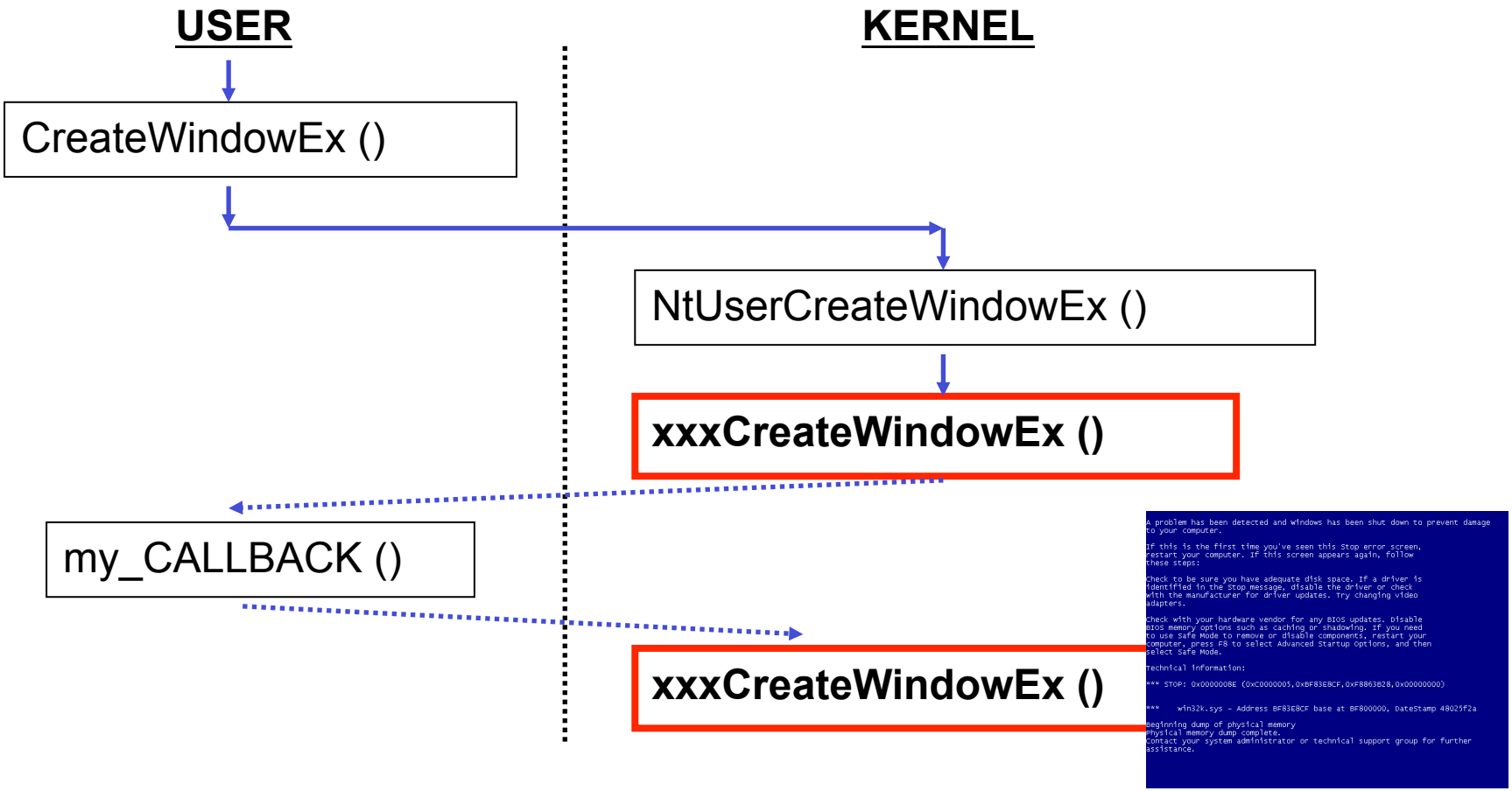


- **Executing my CALLBACK ...**
  - my_CALLBACK(int nCode,WPARAM wParam, LPARAM lParam);

```
typedef struct CBT_CREATEWND
{
    LPCREATESTRUCT lpcs;
    HWND hwndInsertAfter;
} CBT_CREATEWND, *LPCBT_CREATEWND;
```

- ? if (nCode == HCBT_CREATEWND)
  hwndInsertAfter = -1 (0xffffffff) or
  -2 (0xfffffffe)

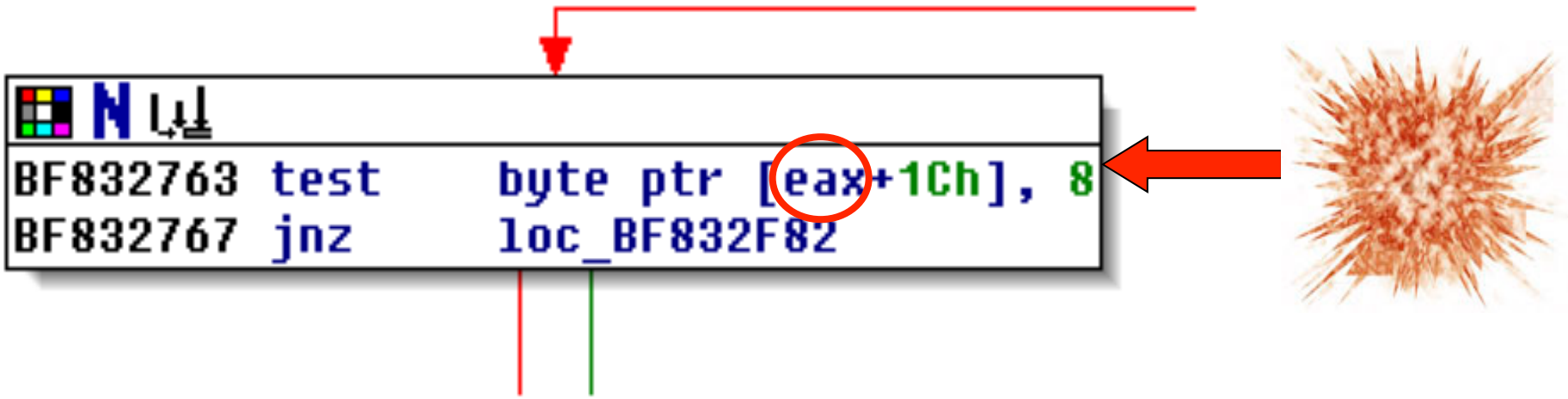
Triggering the Bug !



EXPLOITATION

MISSION:
Injecting code in LSASS.EXE

- **The crash in "xxxCreateWindowEx"**
  - eax = 0xFFFFFFFF (OUR VALUE !)

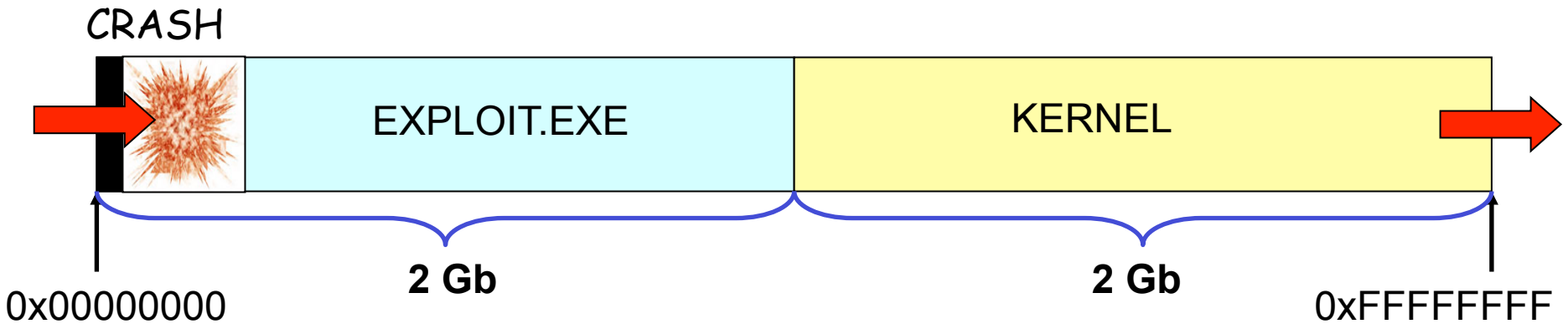


- **The crash in "xxxCreateWindowEx"**

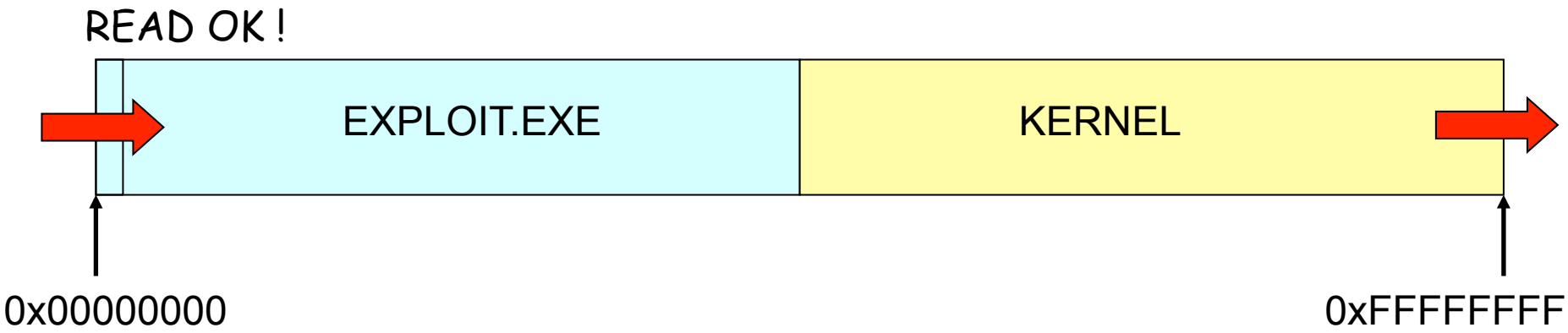
  eax + 0x1C =

  0xFFFFFFFF + 0x1C = **INTEGER OVERFLOW** =

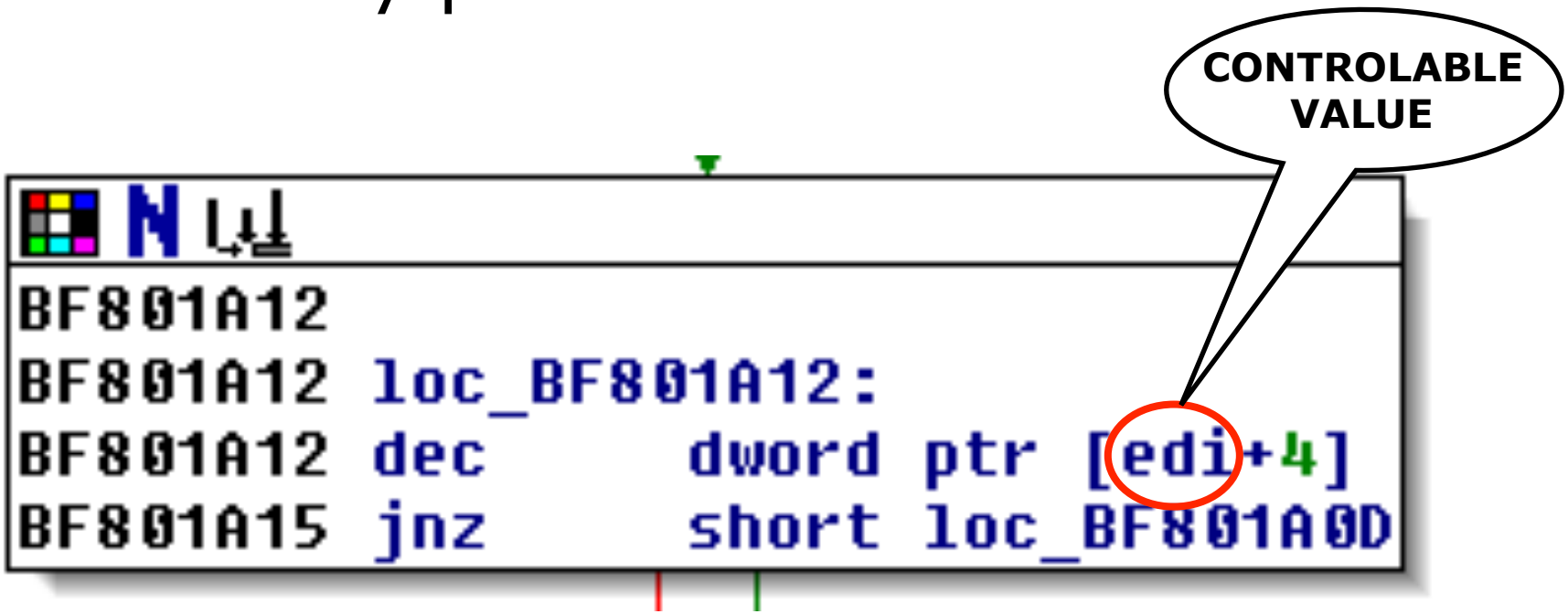
  0x1 0000001B = **0x0000001B**



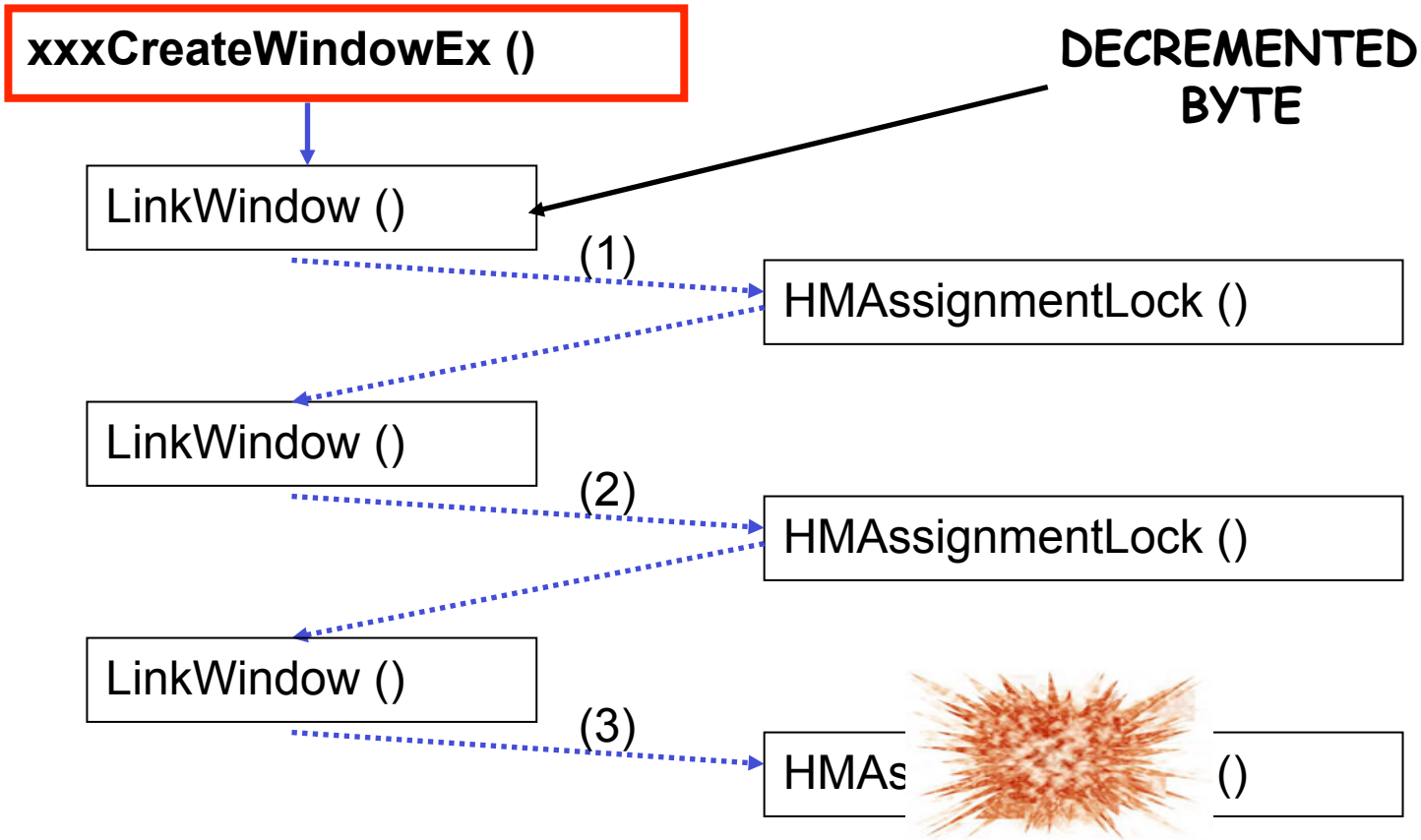
- **Bypassing the crash**
  - Idea
    - » Map at address 0x00000000
    - » Using the function "NtAllocateVirtualMemory"



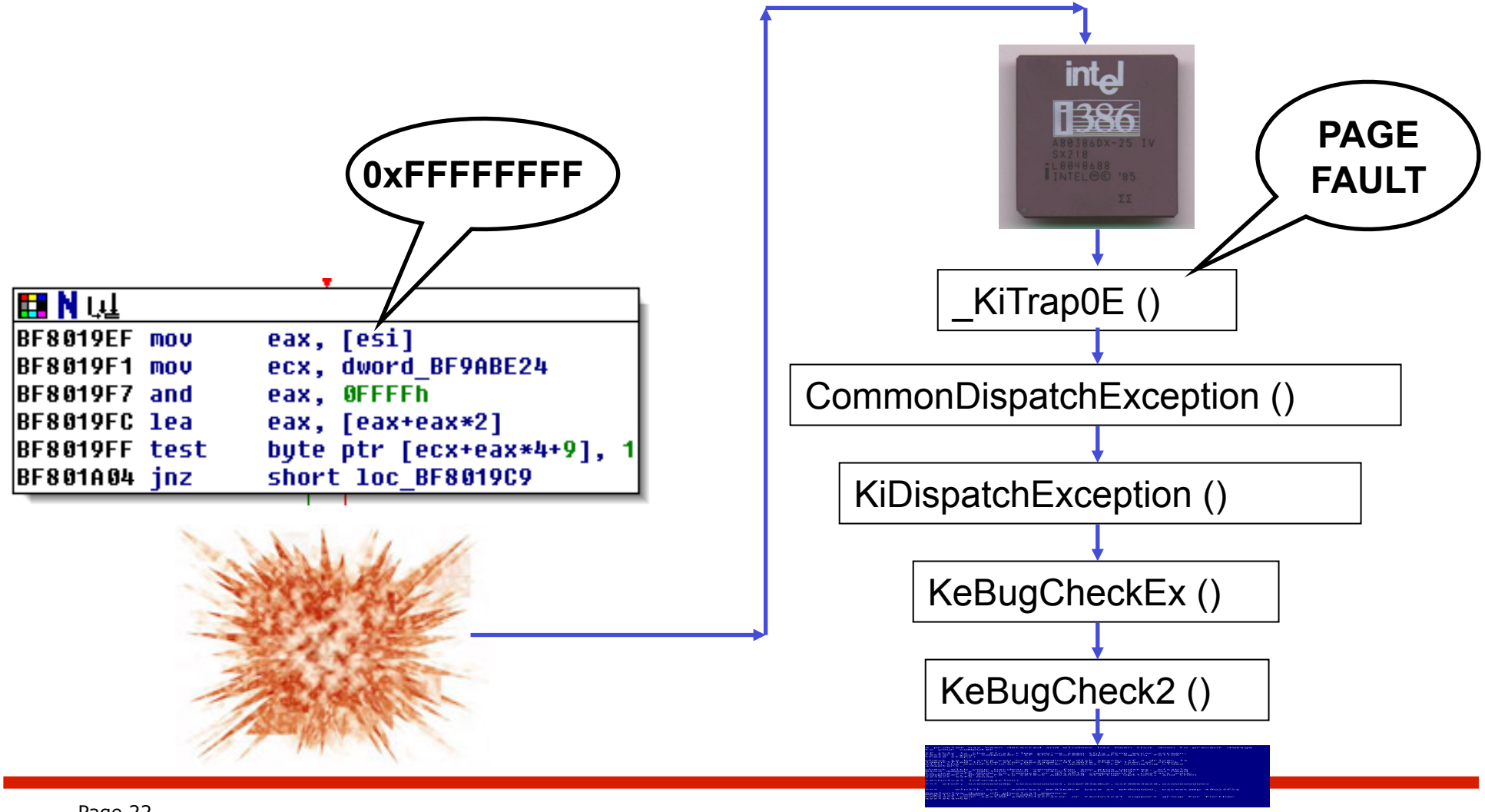
- **What Can I do with this BUG ?**
  - Decrement in **1** a BYTE in any memory position



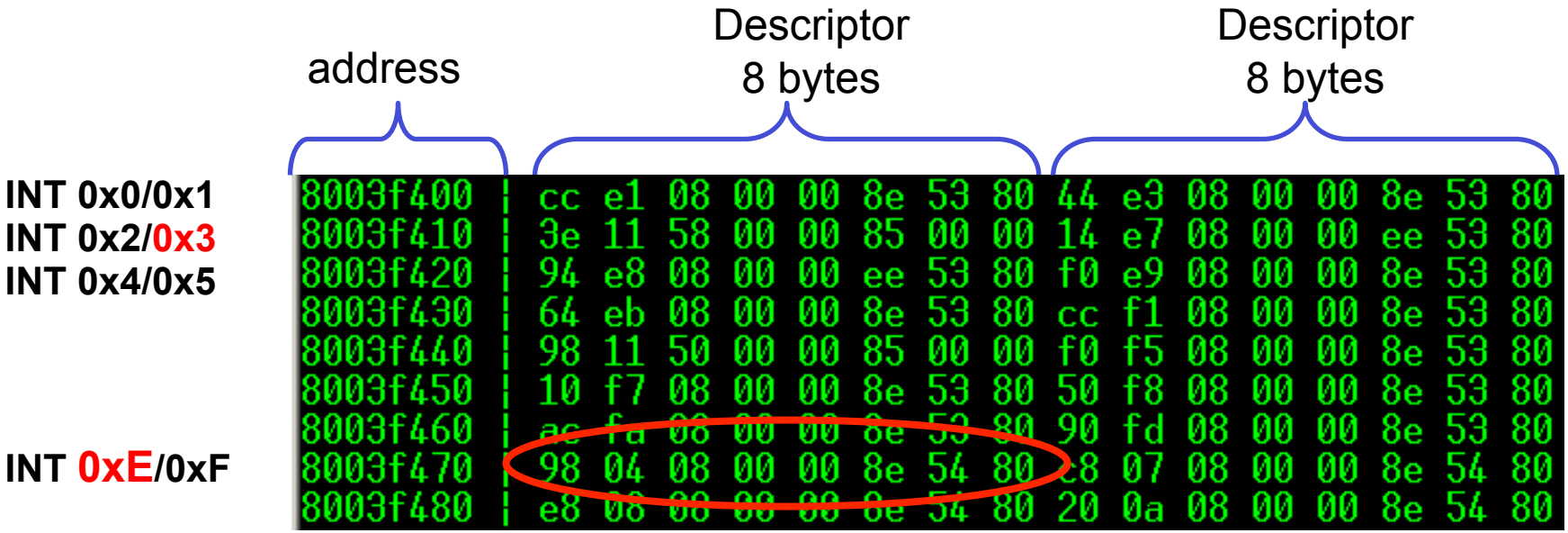
- **The road to the inevitable ...**



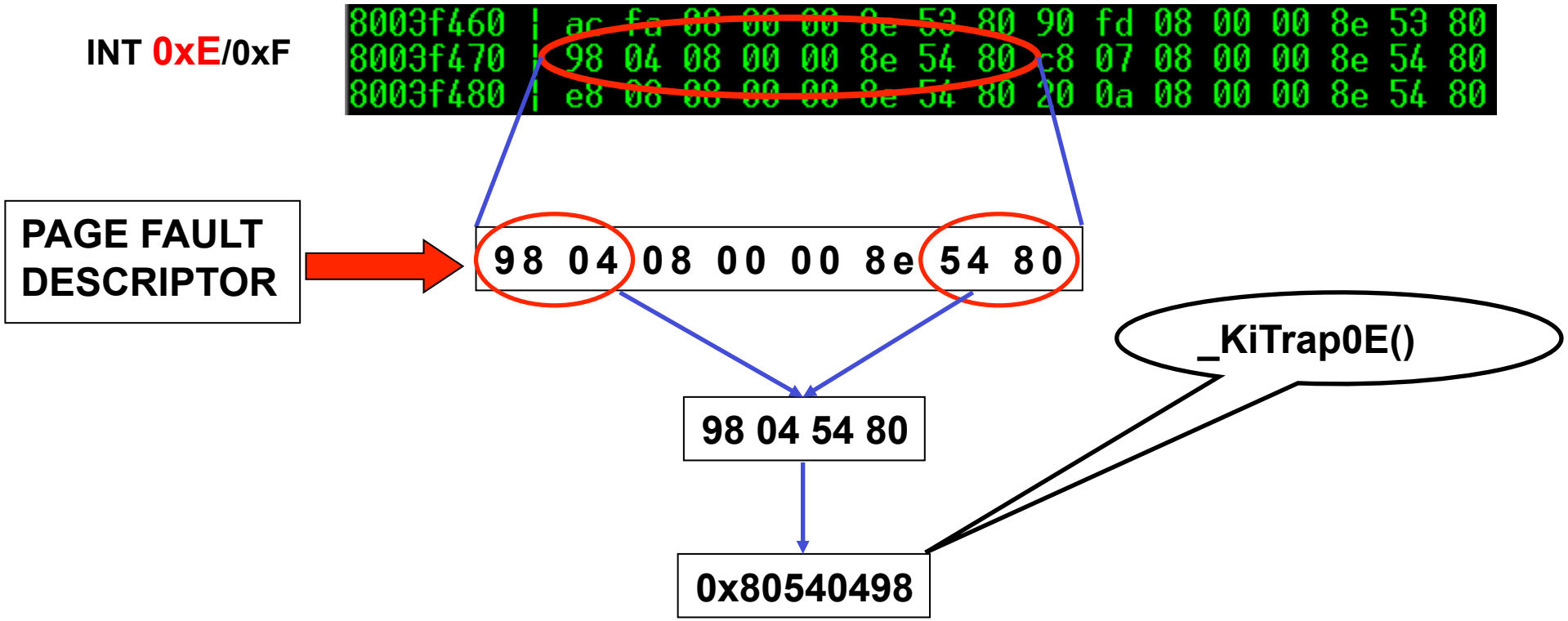
The road to the BSoD



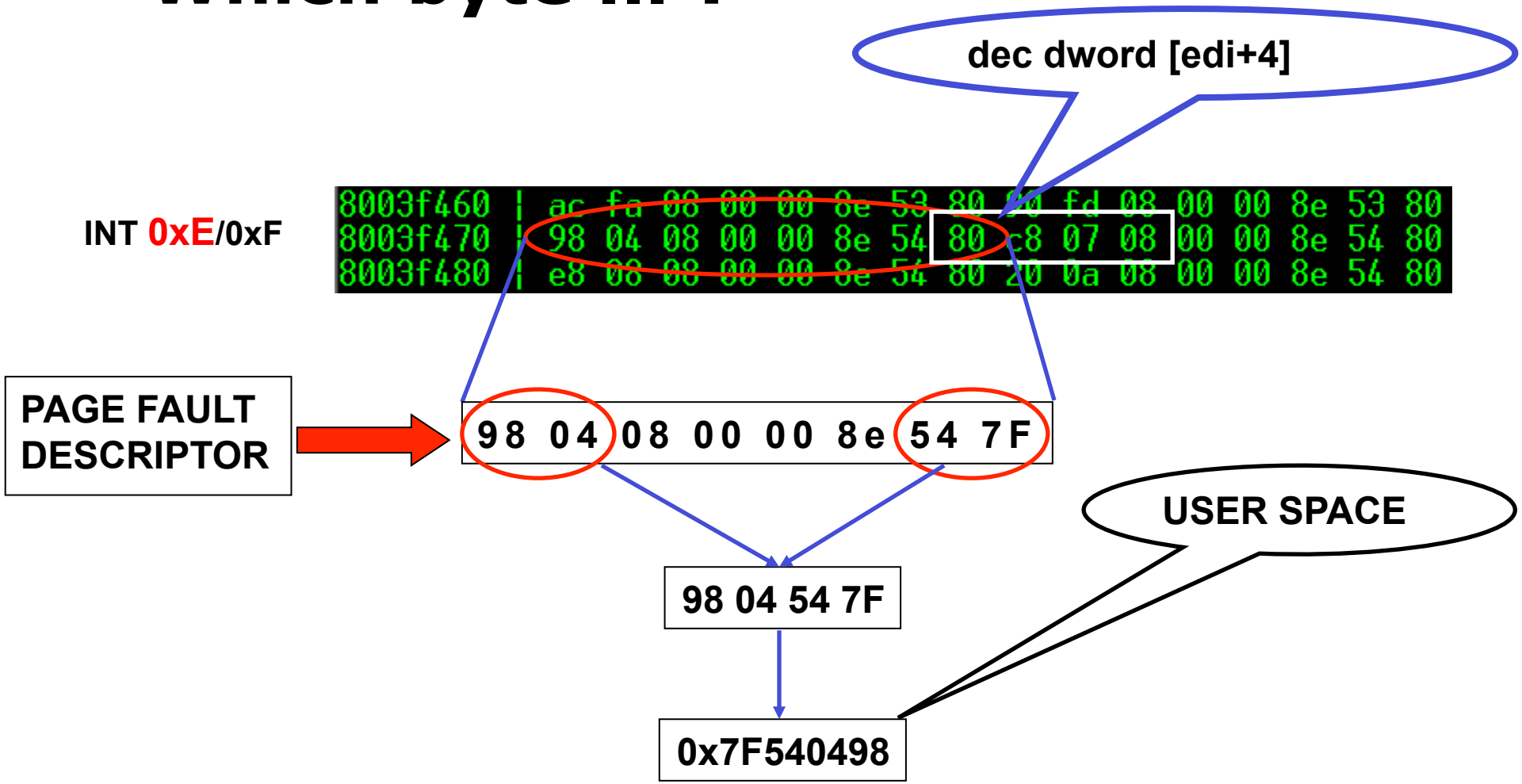
- **Which byte Can I decrement ?**
  - In the IDT (Interrupt Descriptor Table)



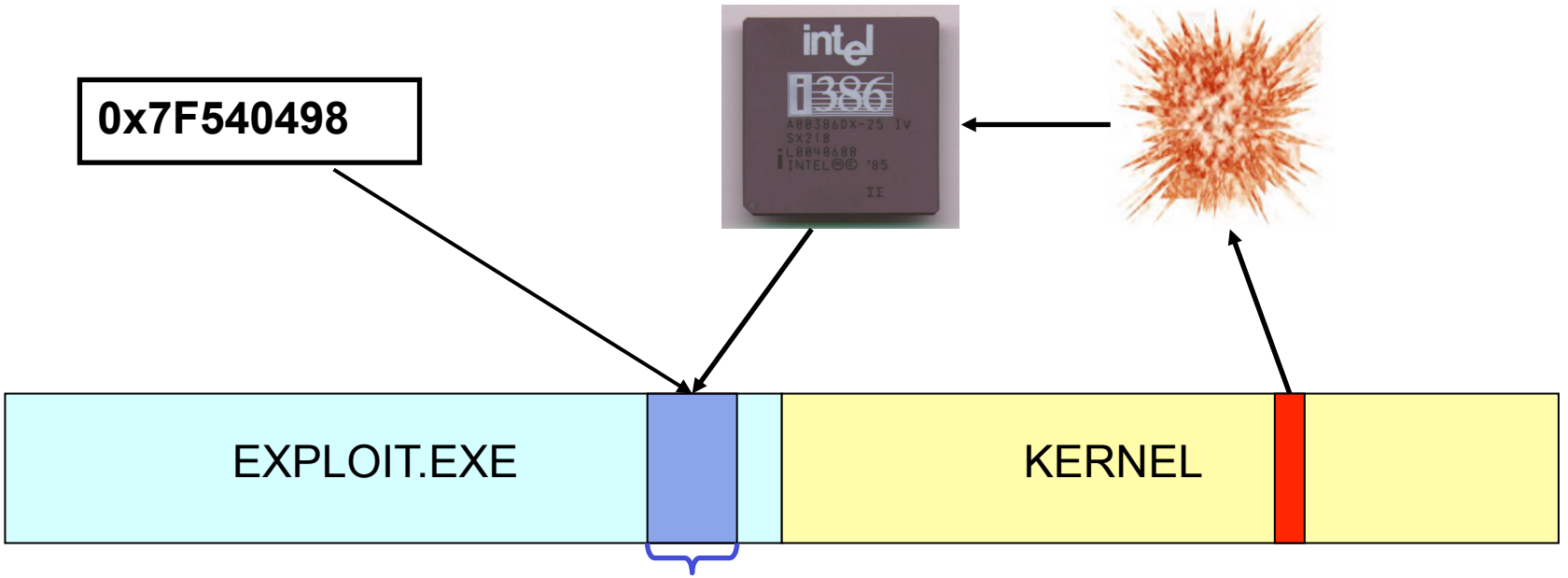
- **Which byte Can I decrement ?**



- **Which byte ... ?**

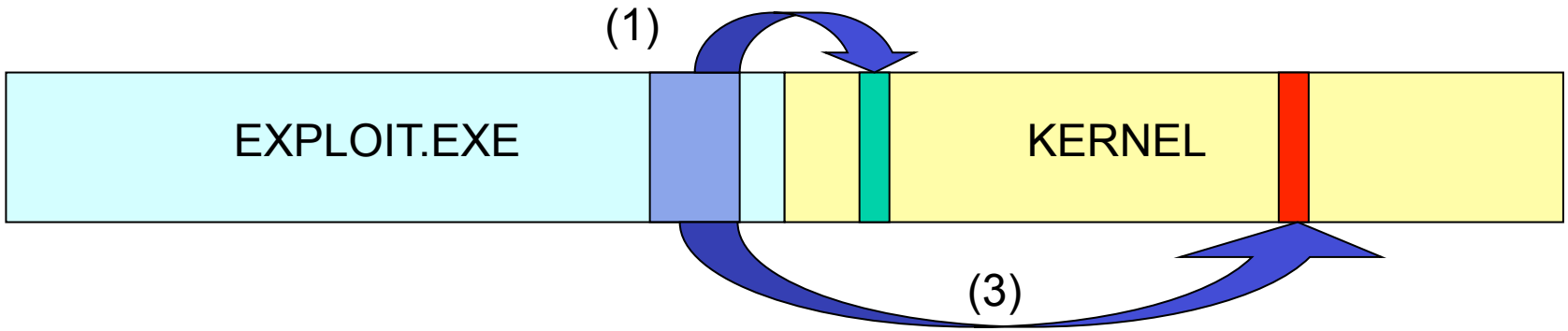

dec dword [edi+4]
INT 0xE/0xF
8003f460 | ac fa 08 00 00 8e 53 80 fd 08 00 00 8e 53 80
8003f470 | 98 04 08 00 00 8e 54 80 c8 07 08 00 00 8e 54 80
8003f480 | e8 08 08 00 00 8e 54 80 20 0a 08 00 00 8e 54 80
PAGE FAULT DESCRIPTOR
98 04 08 00 00 8e 54 7F
USER SPACE
98 04 54 7F
0x7F540498

- **Handling the PAGE FAULT in USER SPACE**

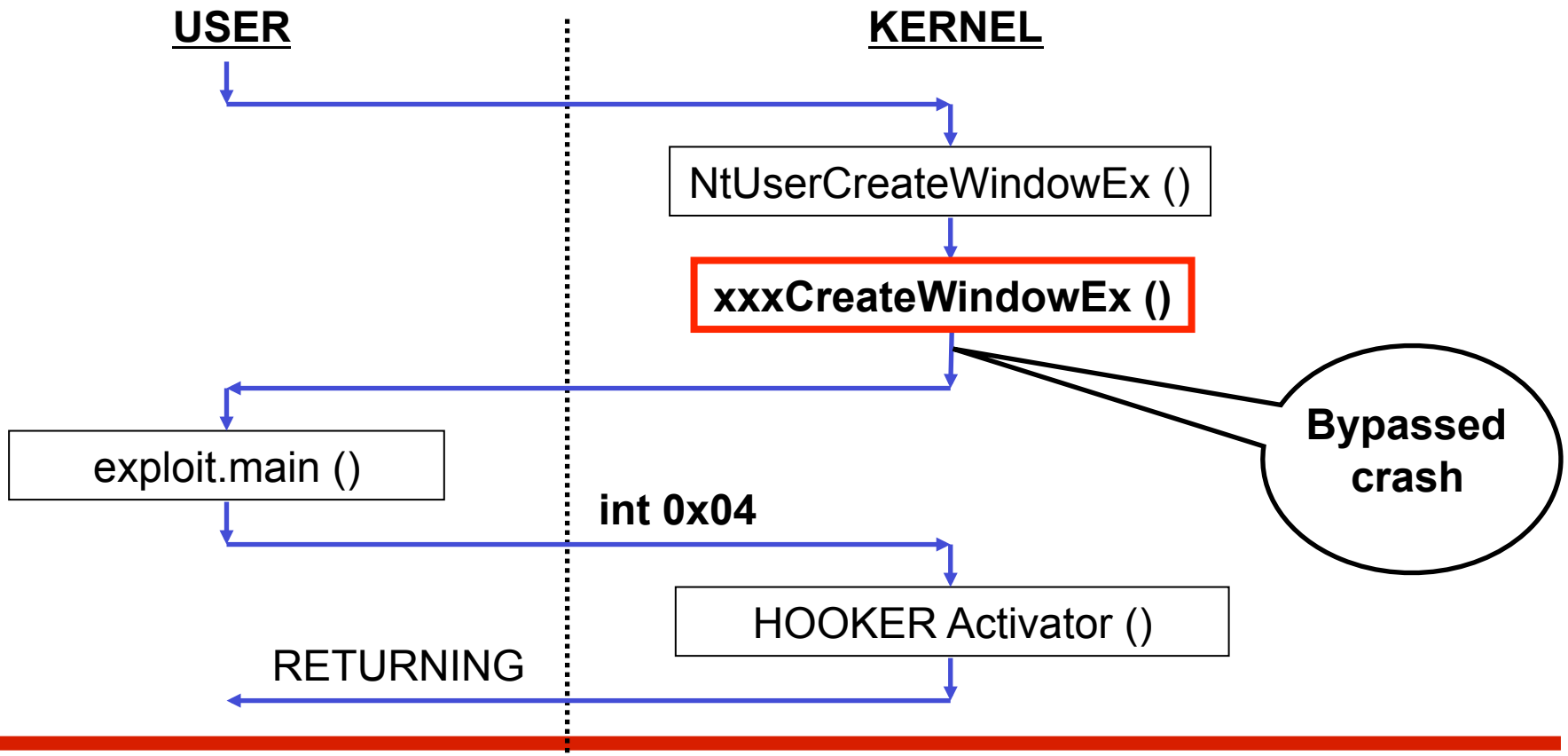


memory MAPPED with NtAllocateVirtualMemory

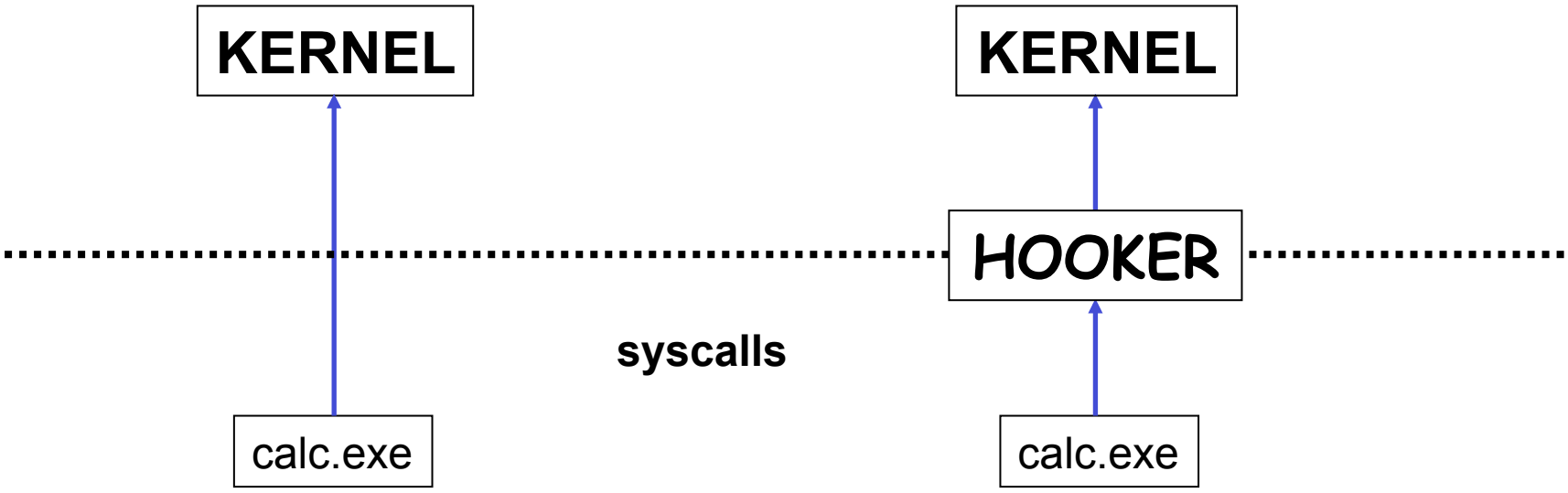
- **Installing a HOOKER in the Kernel**
  - 1. Copy the HOOKER to **unused memory area**
  - 2. Modify the descriptor of the "**int 0x04**" pointing to the HOOKER
  - 3. Restore the CRASH y return from the PAGE FAULT



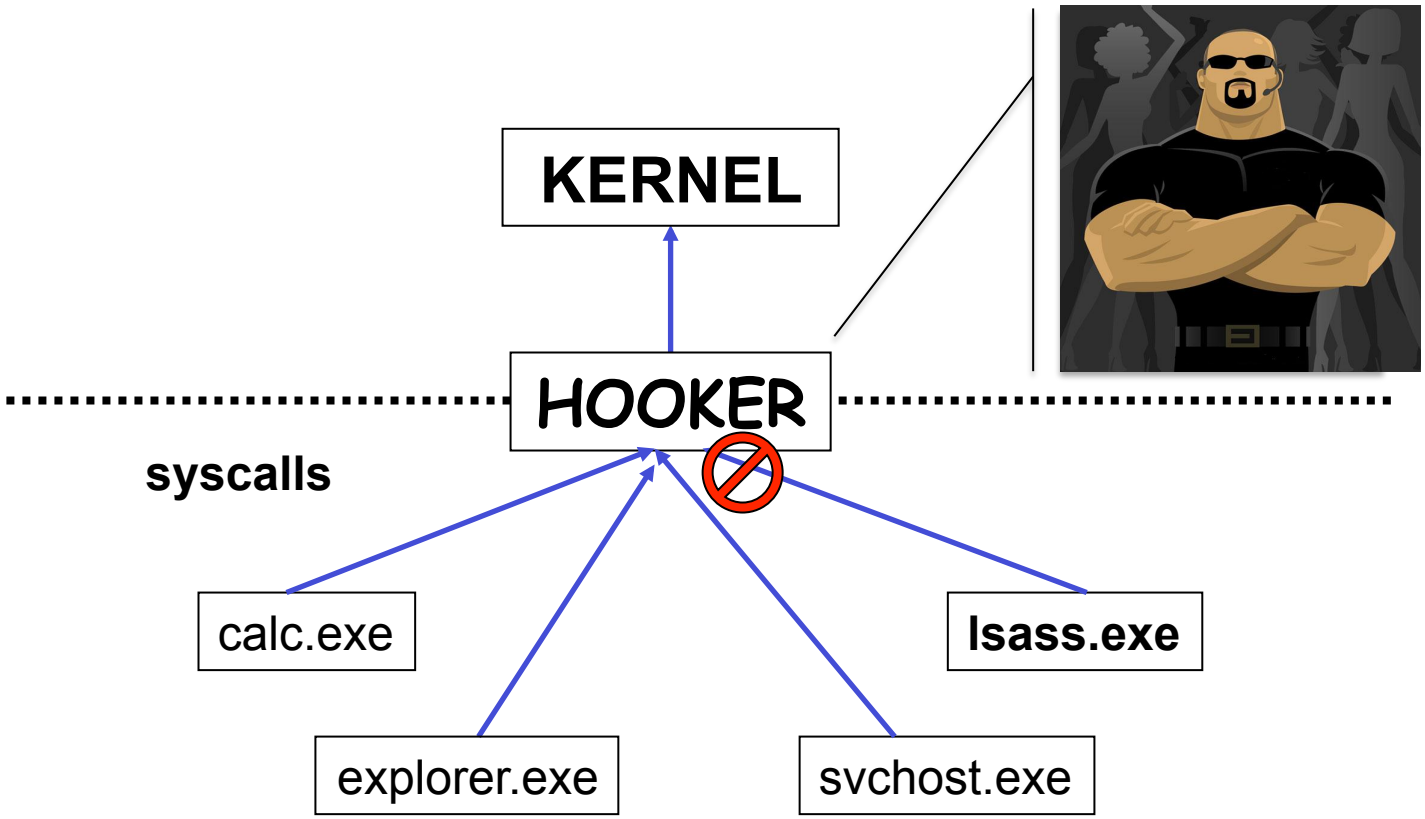
Activating the HOOKER – INT 0x4



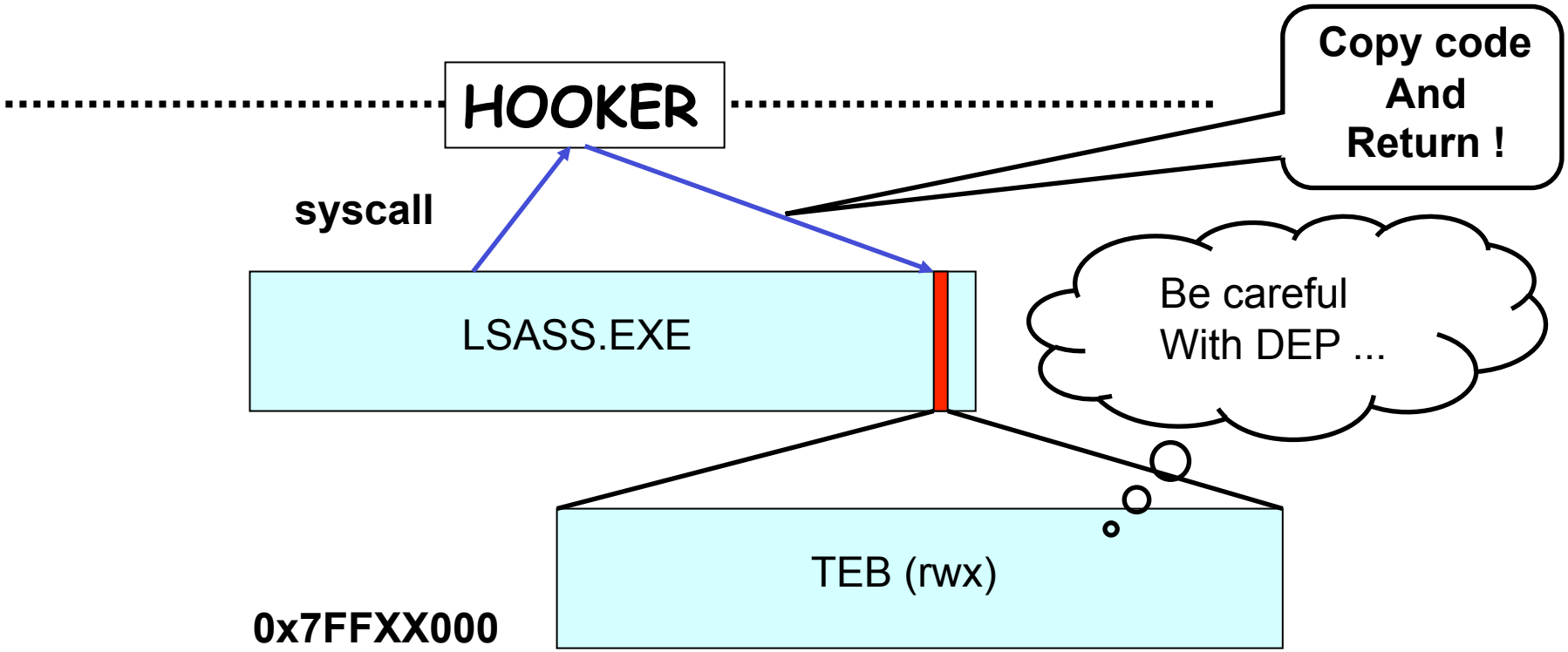
- **The HOOKER Activator**
  - Deflect "SYSENTER" to the HOOKER



- **The HOOKER**
  - You do, you do, you **DONT !**



- **From Ring 0 to Ring 3**
  - Injecting code in LSASS.EXE



- **Conclusions**
  - Microsoft makes MISTAKES when they patch bugs
  - This mistakes can be detecteds through “binary diffing”
  - This bug is very exploitable 

Questions?

Microsoft Virtual PC Hypervisor Memory Protection Vulnerability

(CORE-2009-0803)

Advisory published

03/16/2010 → STILL UNPATCHED!

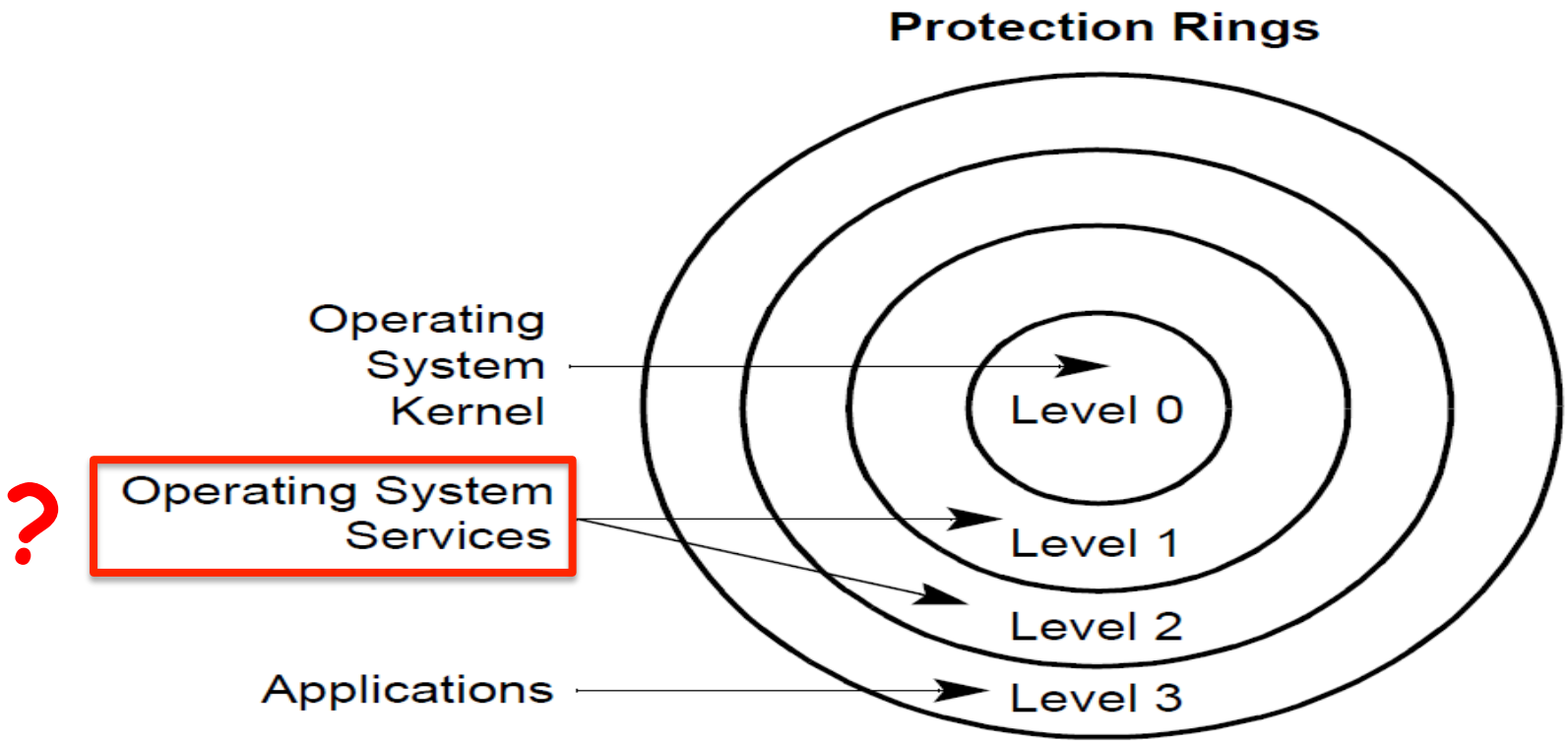
- **What is ?**
  - It’s an application that allows running one or more Virtual Operating Systems **(GUESTs)** inside of a Real Operating System **(HOST)**
  - Originally it was developed by **Connectix**.
  - On February 19th 2003, **Microsoft** acquired **Connectix** Virtual Machine Technology

x86 Feature History

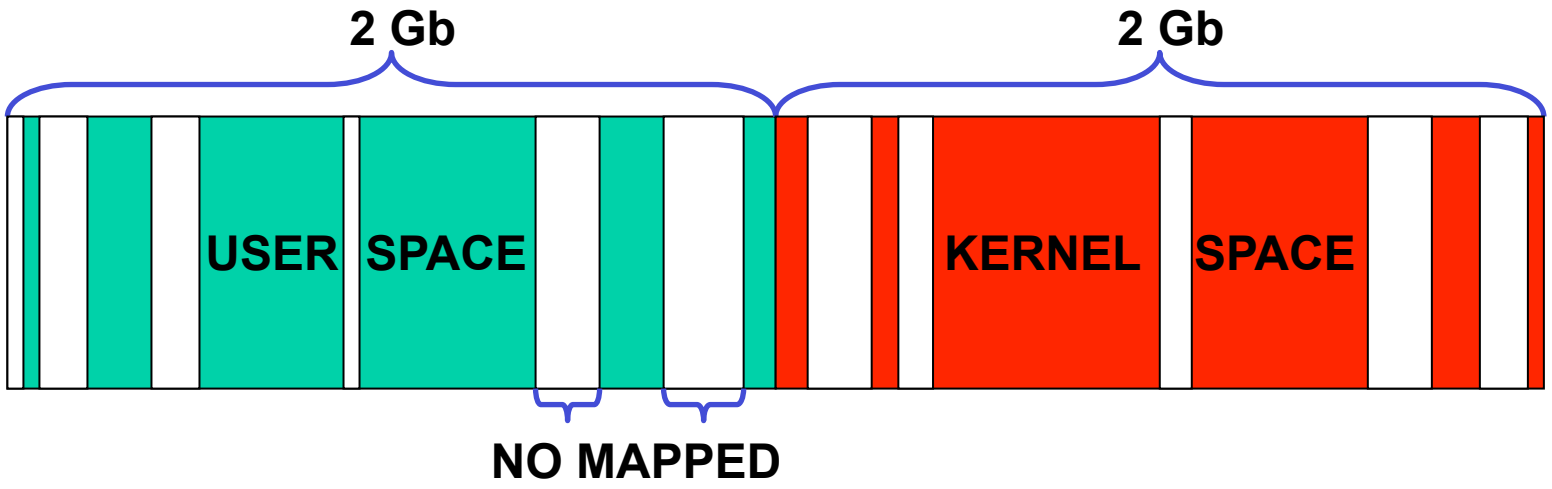
- 80386 (Year 1985)
  - Frecuency: 16 a 40 Mhz
  - Address Mode:
    - 32 bits = 4 GB
    - 8192 segments (selectors) of 4 Gbytes
    - **Pagination (Virtual Memory)**
  - Working Modes:
    - Real mode, **protected mode**, VMM86, SMM
    - **4 Privilege Levels (RINGs)**

- Protected Mode: Segmentation
  - GDT: Array of 8192 entries (Descriptors)
    - E.g Windows → **CS** = 0x1b, **DS**=**ES**=**SS**= 0x23, **FS**=0x3b, **GS**=0
  - IDT: Array of 256 Interruptions
  - Segment Types:
    - » Code ← MEMORY NO WRITABLE
    - » Data ← MEMORY NO EXECUTABLE
  - Memory Models:
    - » Flat **(**← BIG ERROR !!!!!!!!!!!!!!!!!!!!**)**
    - » Segmented

- Protected Mode: Privileges



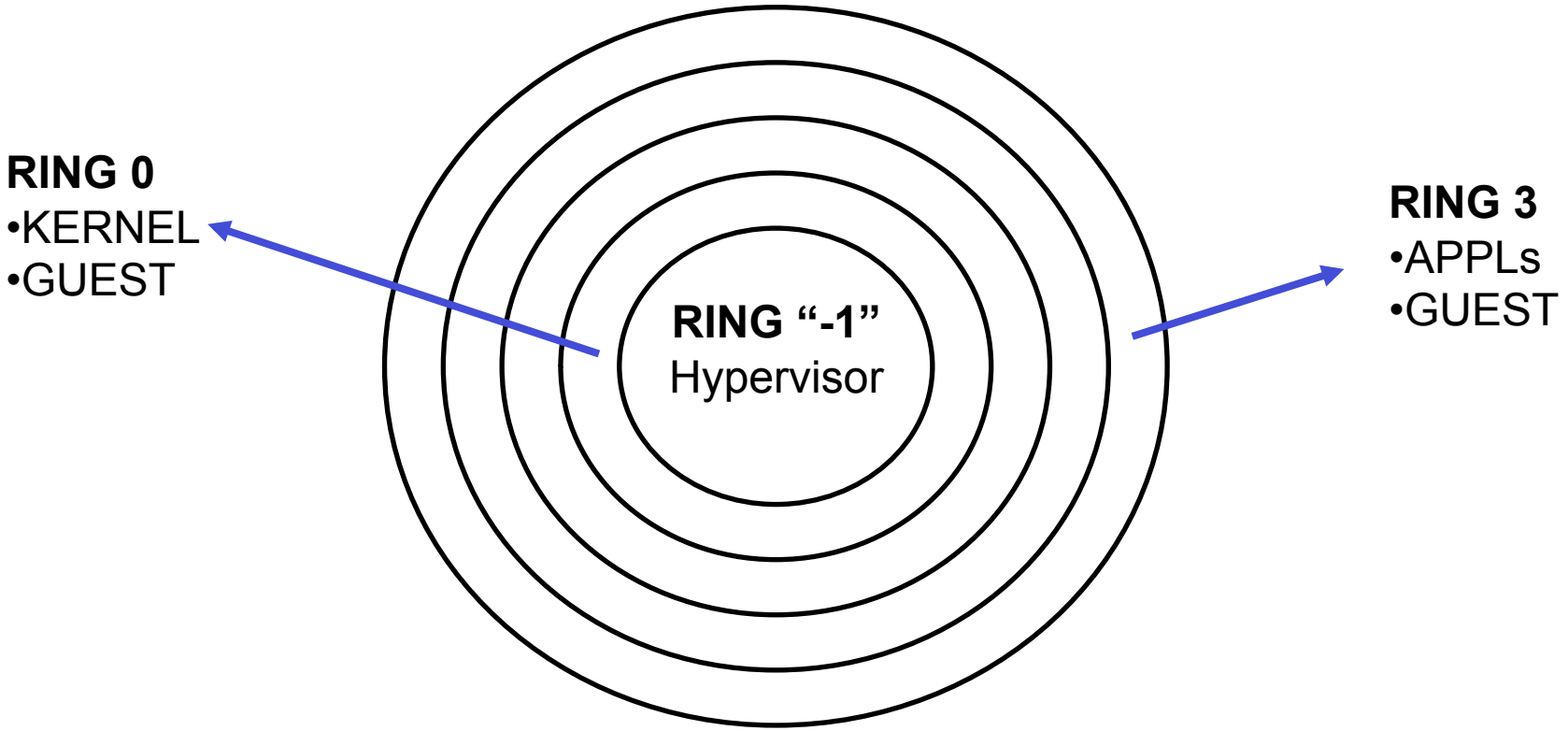
- Pagination
  - Register **CR3** (it uses physical address)
  - Segmentation over Pagination
  - Pages of 4KB with privileges of **USER** or **SUPERVISOR**
  - E.g Typical Windows process



- **Athlon 64 (Year 2006)**

  - Frecuency: ~ 3,8 Ghz
  - Address Mode:
    - 32 bits and 64 bits
    - PAE – Bit NX
    - **Virtualization Instructions (Pacifica – AMD-V)**
  - Working modes:
    - Real mode, protected mode, VMM86, SMM
    - **5 Privilege Levels (RINGs)**

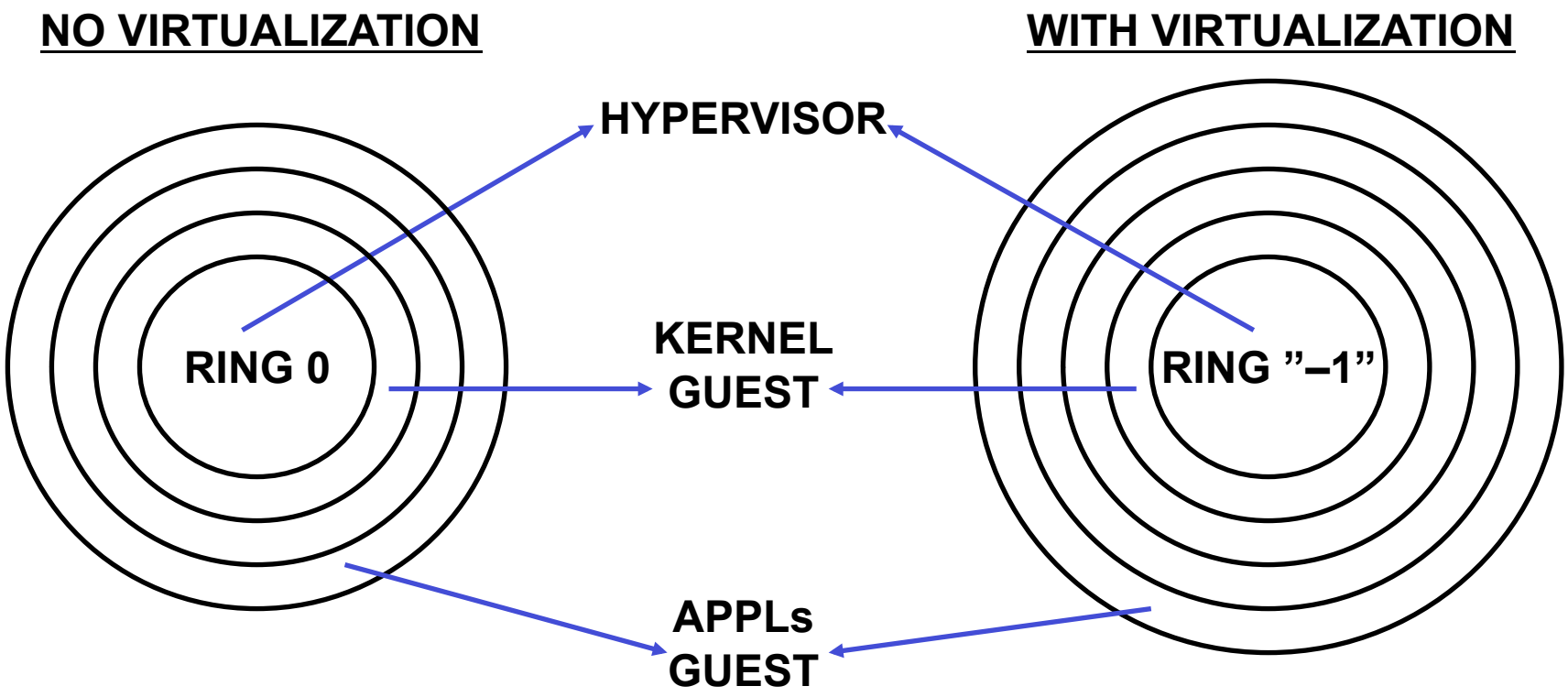
- **Virtualization**
  - It allows to run in ring 0 to the kernel guest



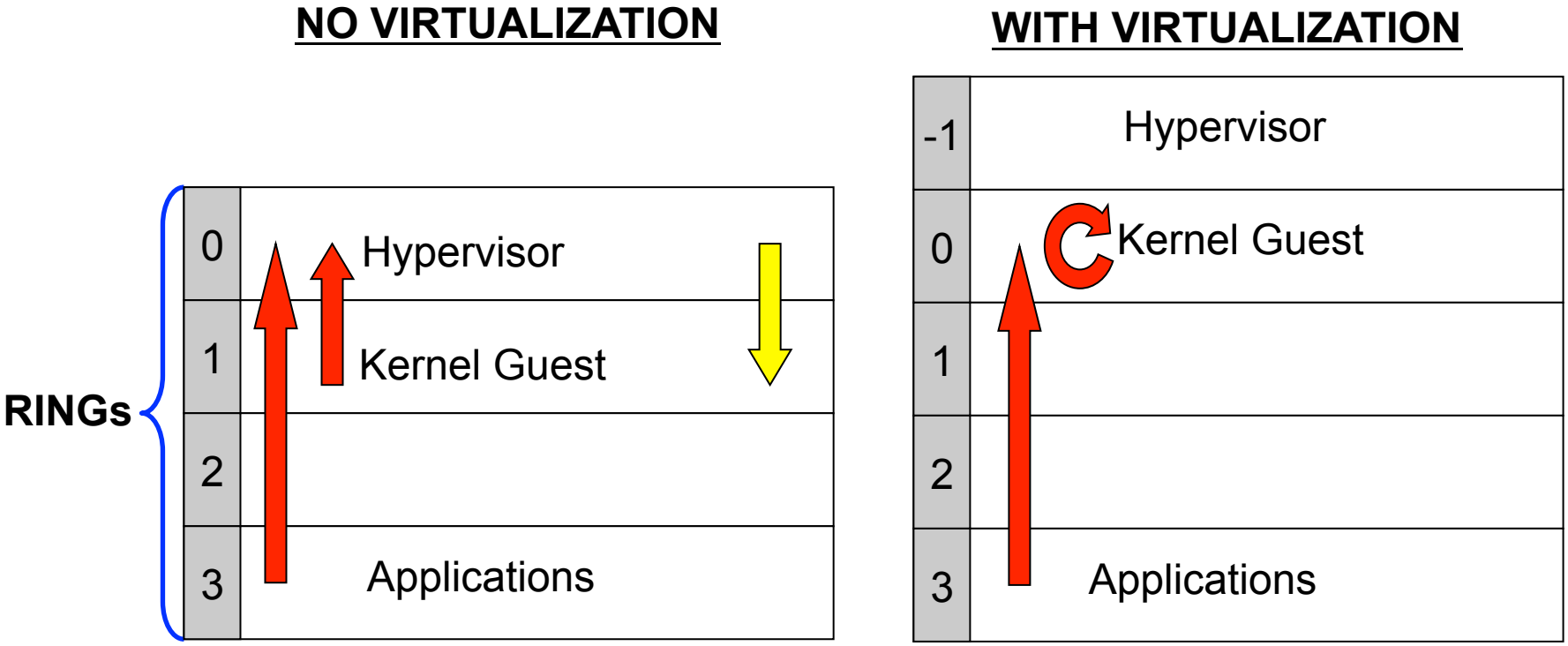
The Hypervisors

- **What is it ?**
  - It’s a program
  - It has the highest privilege level
  - It’s the responsible of devices to the OS Guest that it’s running in a REAL MACHINE
  - **It’s invisible to the OS Guest**

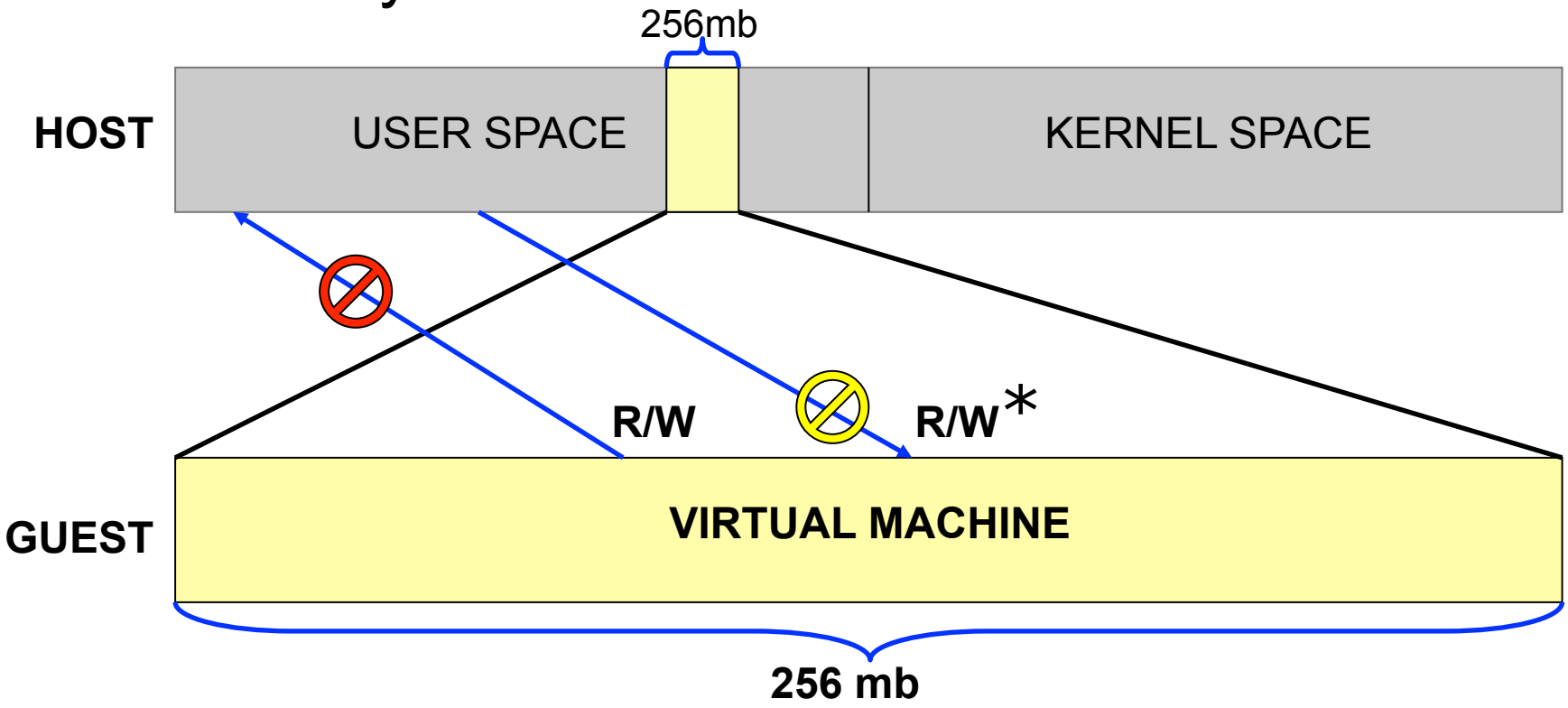
- **Ring Levels**


NO VIRTUALIZATION
WITH VIRTUALIZATION
HYPERVISOR
RING 0
KERNEL
GUEST
RING ”–1”
APPLs
GUEST

- **Interruptions, Exceptions and Traps**



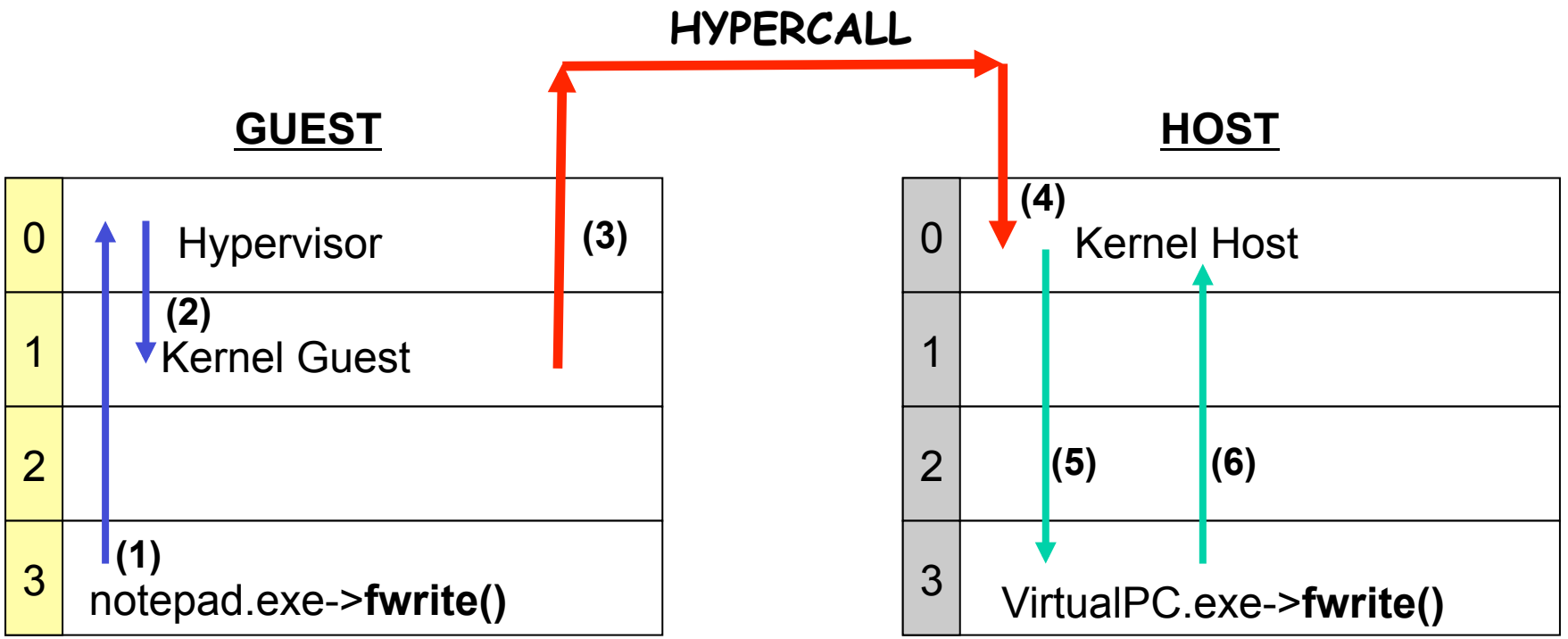
- Memory Managment: E.g “VirtualPc.exe” memory



* See “Code injection on Virtual Machines” (Ekoparty, 2008)

- **Hardware Managment**
  - The hypervisor DOESN'T have access to hardware (except in special occasions)
  - The hardware is VIRTUALIZED (DOESN'T drive real HARDWARE)
  - Use **HYPERCALLs** to comunicate con el Host OS
  - The Host OS processes an executes operations on the REAL HARDWARE

- Hardware Managment



- **Detection from USER without virtualizacion**
  - It can't hide the behavior of few instructions (sensitive instructions).
  - It's easy to detect when an OS is being virtualizated.
  - E.g Sensitive instructions: 'sidt','sgdt','lsl','lar',etc

The Bug

- **Description**
  - This vulnerability is in “VMM.SYS”
  - It allows to read/write few memory areas above to 2GB
  - The GUEST OS kernel **DOESN’T** know this memory area

DEMO

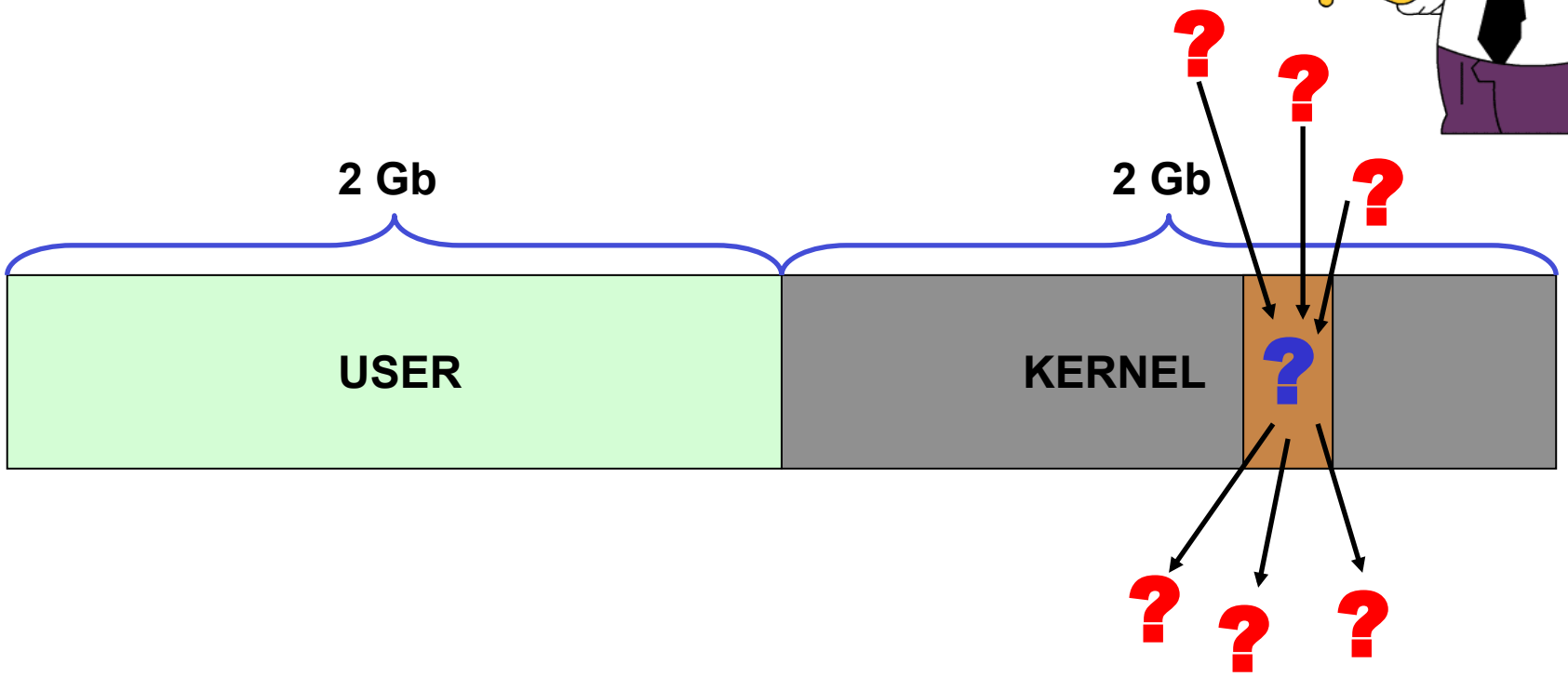
Using the **"vpscanner.exe"**

(or **looking beyond...**)

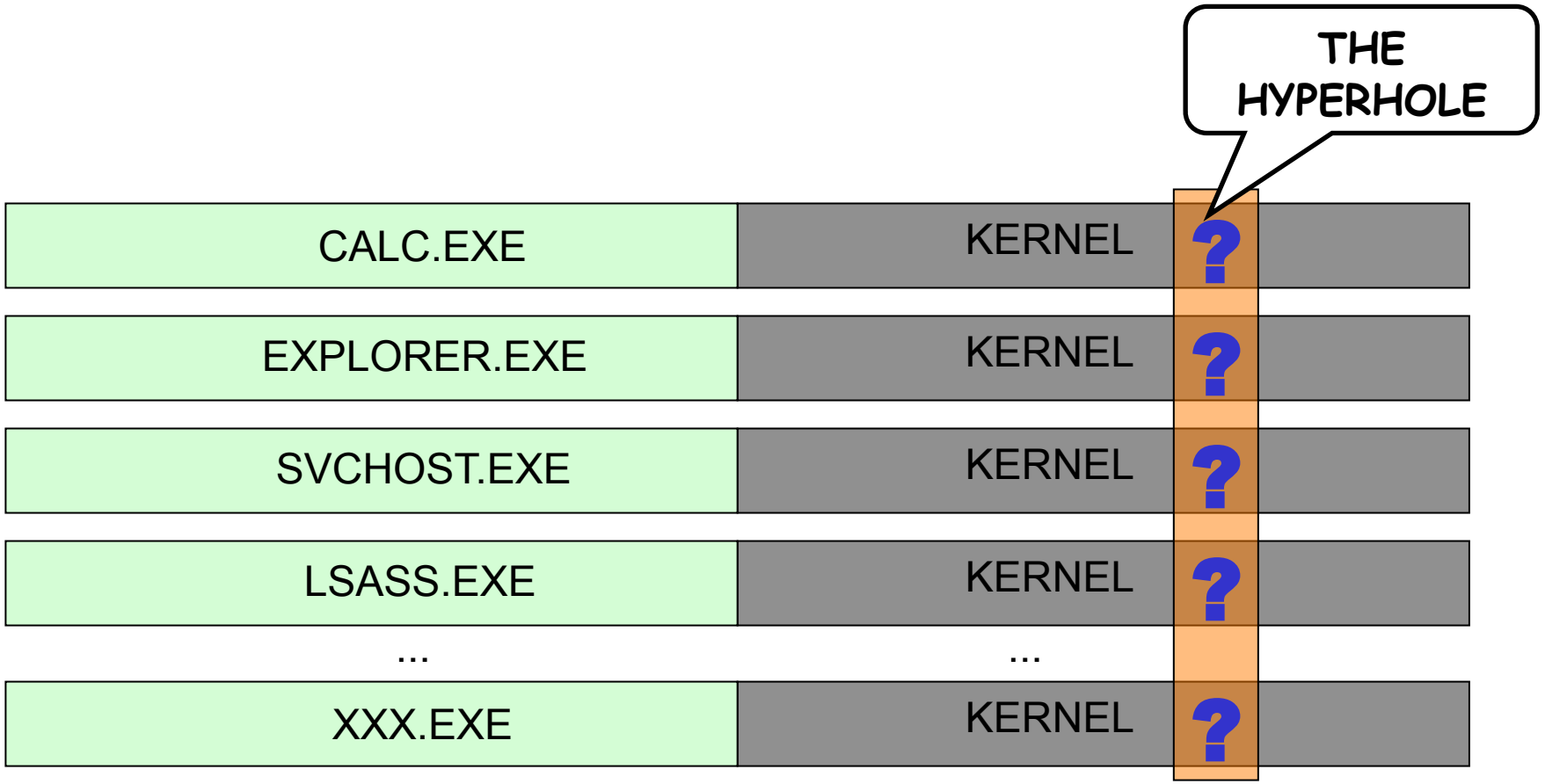
1. It reads/writes above to 2 Gb memory space

2. It uses "IsBadReadPtr" and "IsBadWritePtr"

WTF ?



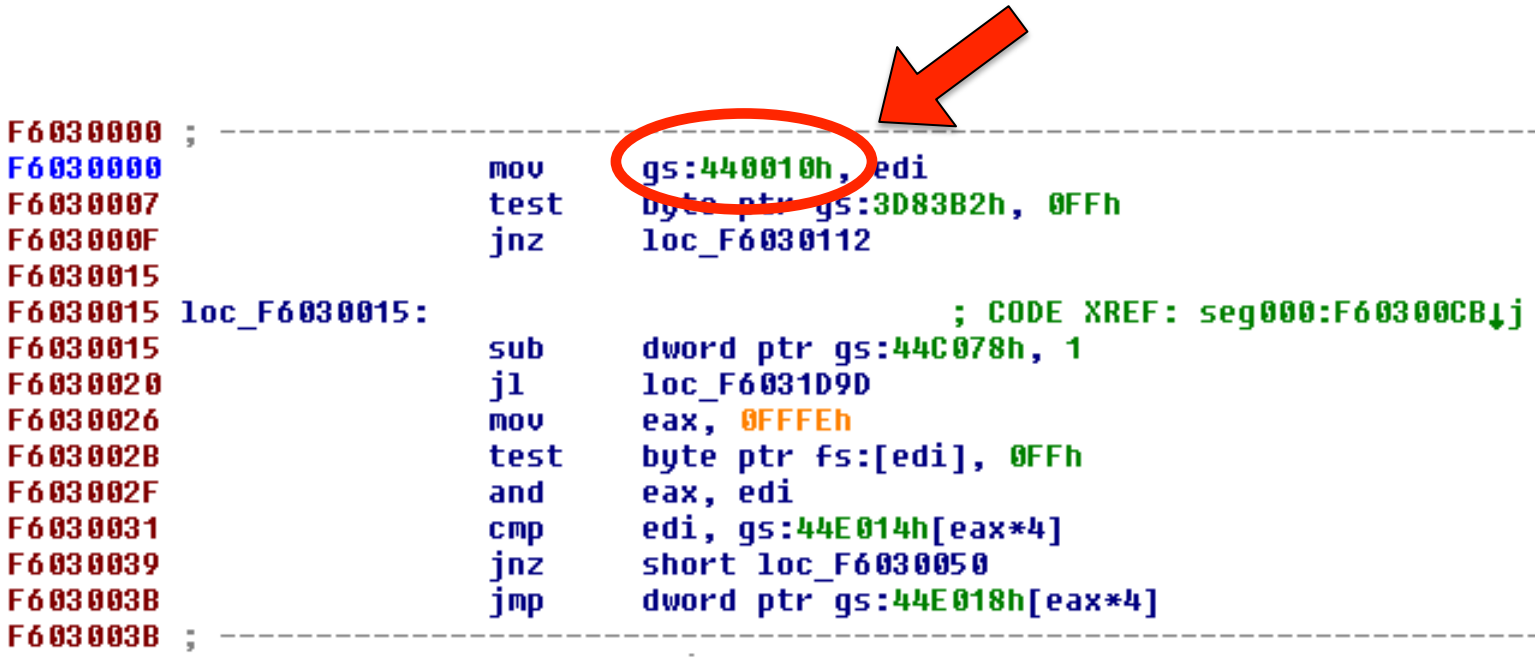
- All affected processes ? ... YES



- **Vulnerable Versions**
  - Virtual PC 2004, 2007, Virtual Server
  - XP Mode en Windows 7

- **Affected guests**
  - ALL

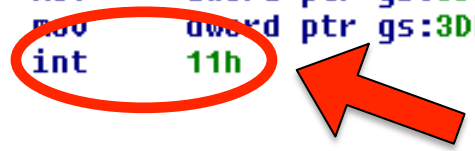
- **Looking at the leaked memory**
  - According to Microsoft,it is "**The work area**"
  - E.g: The first leaked code (offset 0xfc000)



Looking at the leaked memory

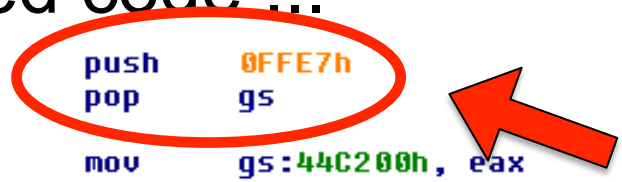
- More leaked code ...

```
F6030337        mov     edi, cs
F6030339        test    edi, 2
F603033F        jz      short loc_F6030398
F6030341        mov     dword ptr gs:3DB004h, 0FFFDh
F603034C        mov     dword ptr gs:3DB008h, 0FFDDh
F6030357        mov     dword ptr gs:3DB000h, 3
F6030362        int     11h             ; EQUIPMENT DETERMINATION
```

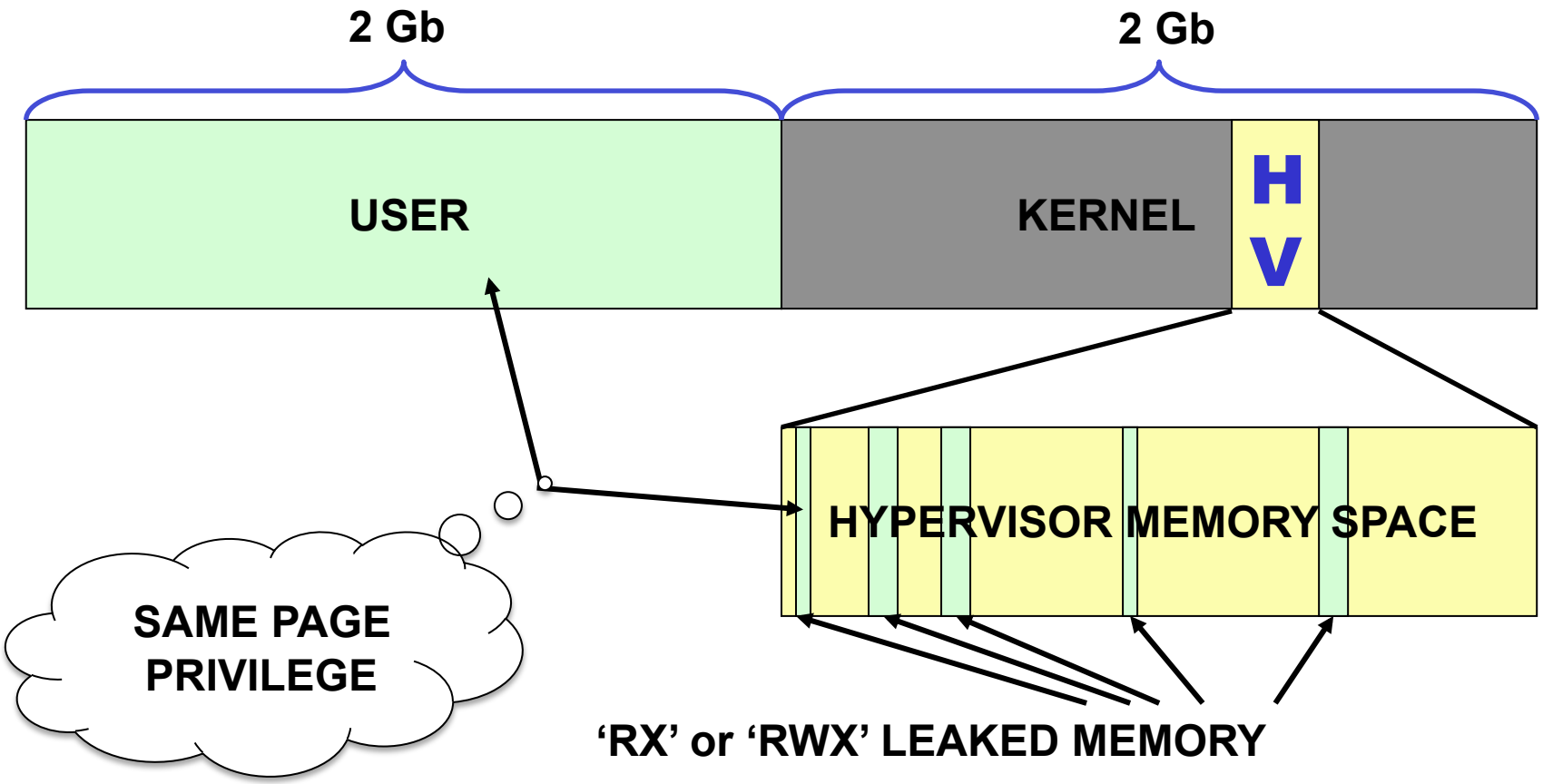


- More leaked code ...

```
F6033661        push    0FFE7h
F6033666        pop     gs
F6033668
F6033668        mov     gs:44C200h, eax
F603366E        mov     gs:44C20Ch, ebx
F6033675        mov     gs:44C204h, ecx
F603367C        mov     gs:44C208h, edx
F6033683        mov     gs:44C218h, esi
F603368A        mov     gs:44C21Ch, edi
F6033691        mov     gs:44C214h, ebp
F6033698        lahf
```



- **Analyzing the leaked memory**



- **A year later…**

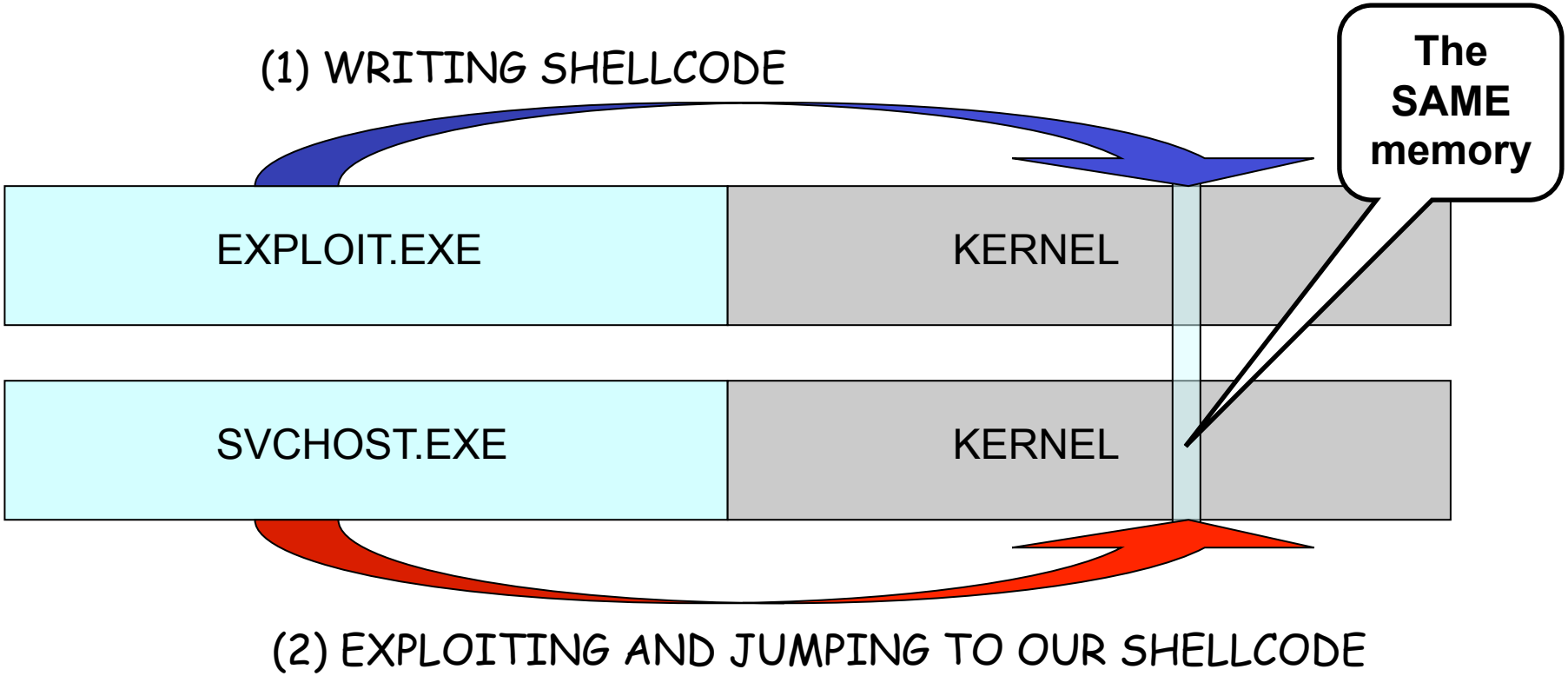
"The functionality that Core calls out is not an actual vulnerability per se …"

Paul Cooke
(Director for Microsoft who manages enterprise security technology in Windows group.)

Breaking Rules

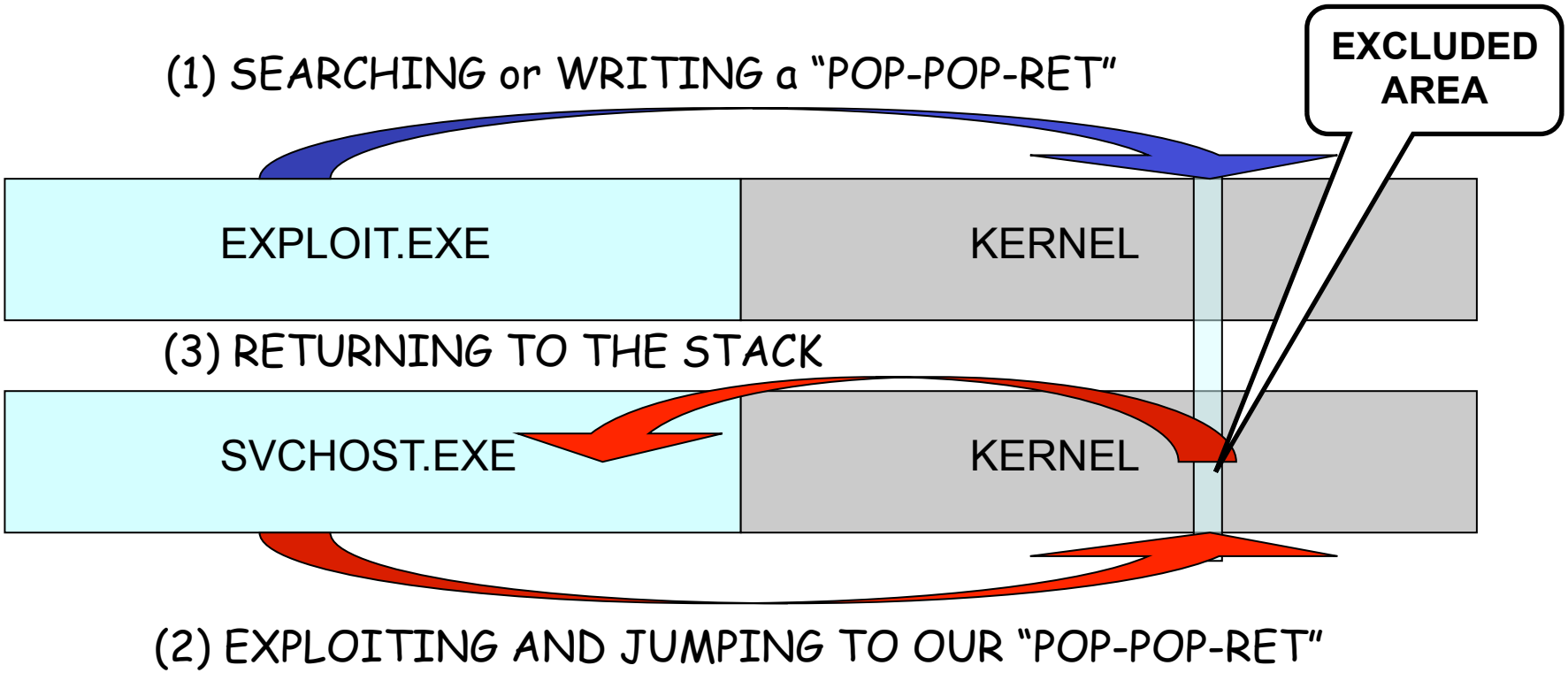
- **Memory protecction**
  - Each process has its own memory space
  - Any process can READS or WRITES memory of other process (except with special APIs)

- **Memory protecction**
  - In local exploits, we know where we can **JUMP** !



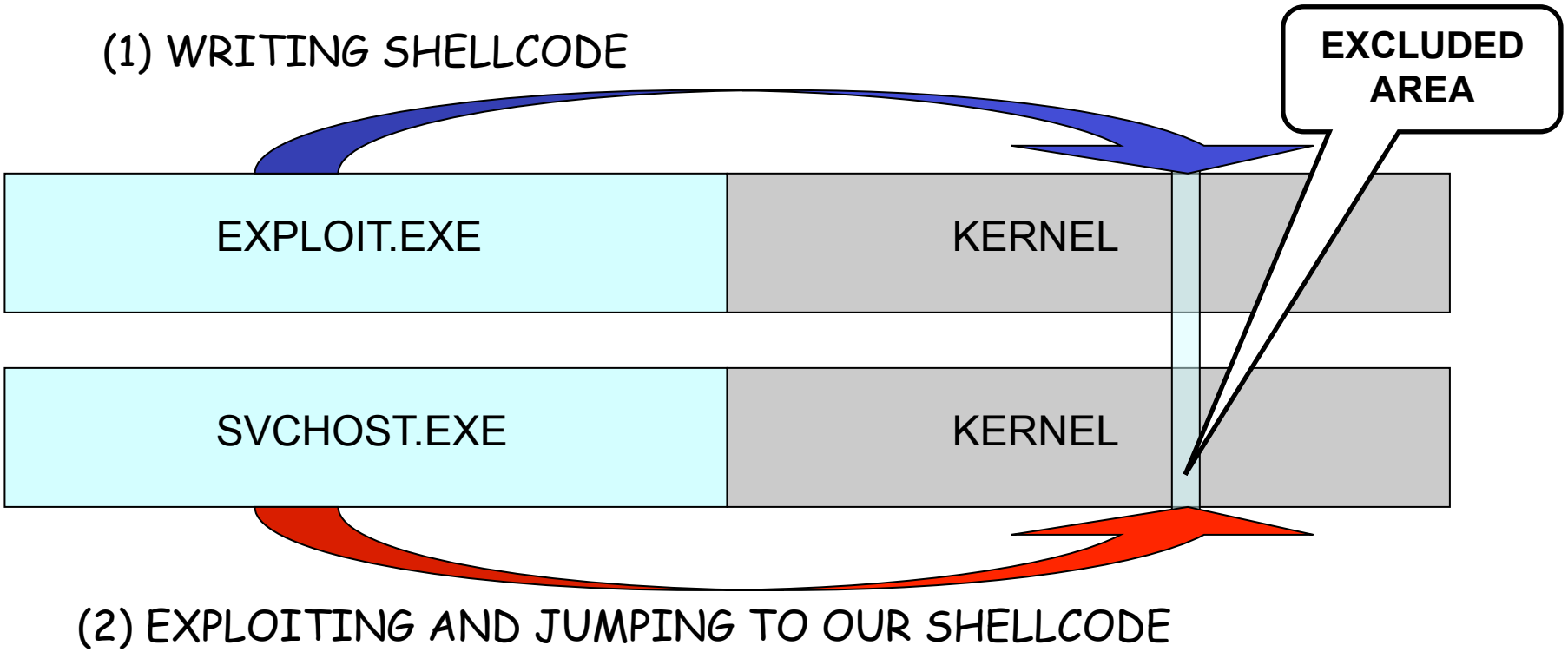
- **SEH (Safe Exception Handling)**
  - We can only use (jump to) addresses in the program or registered DLL addresses (“White List”)
  - We can’t jump to the STACK
  - We can jump to memory addresses that don’t belong to any module (E.g Heap)

- **SEH (Safe Exception Handling)**
  - We can bypass it JUMPING to this memory area.



- **BIT NX (NO EXECUTE)**
  - This bit enables or disables the memory execution
  - It's used for disabling **DATA** execution
  - Used example: STACK and HEAP
  - It's only enabled in "Virtual PC XP Mode"

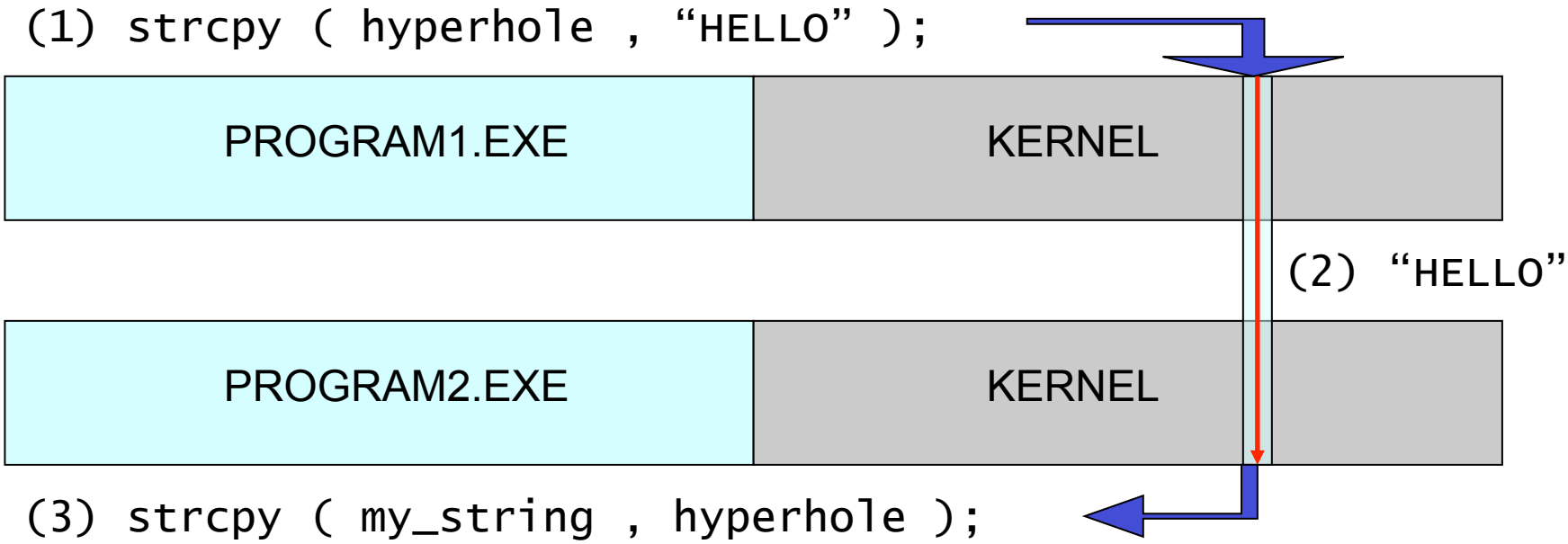
- **BIT NX (NO EXECUTE)**
  - We can bypass it by WRITING and JUMPING to this memory area



- **Fixed Addresses**
  - Addresses with the **SAME** content (e.g "pop-pop-ret")
  - They are used by local, remote and client side **EXPLOITS**
  - They are **FEW** and very **COVETED**

- **ASLR weakening**
  - "Virtual PC XP Mode"
    - Randomize ~4/6bits
  - Virtual PC (depends on the Guest OS)
    - Randomize ~4/8bits

- **Undercover IPC (Inter Process Communication)**
  - Using "My Shared Memory" to do that



Attacking the bug

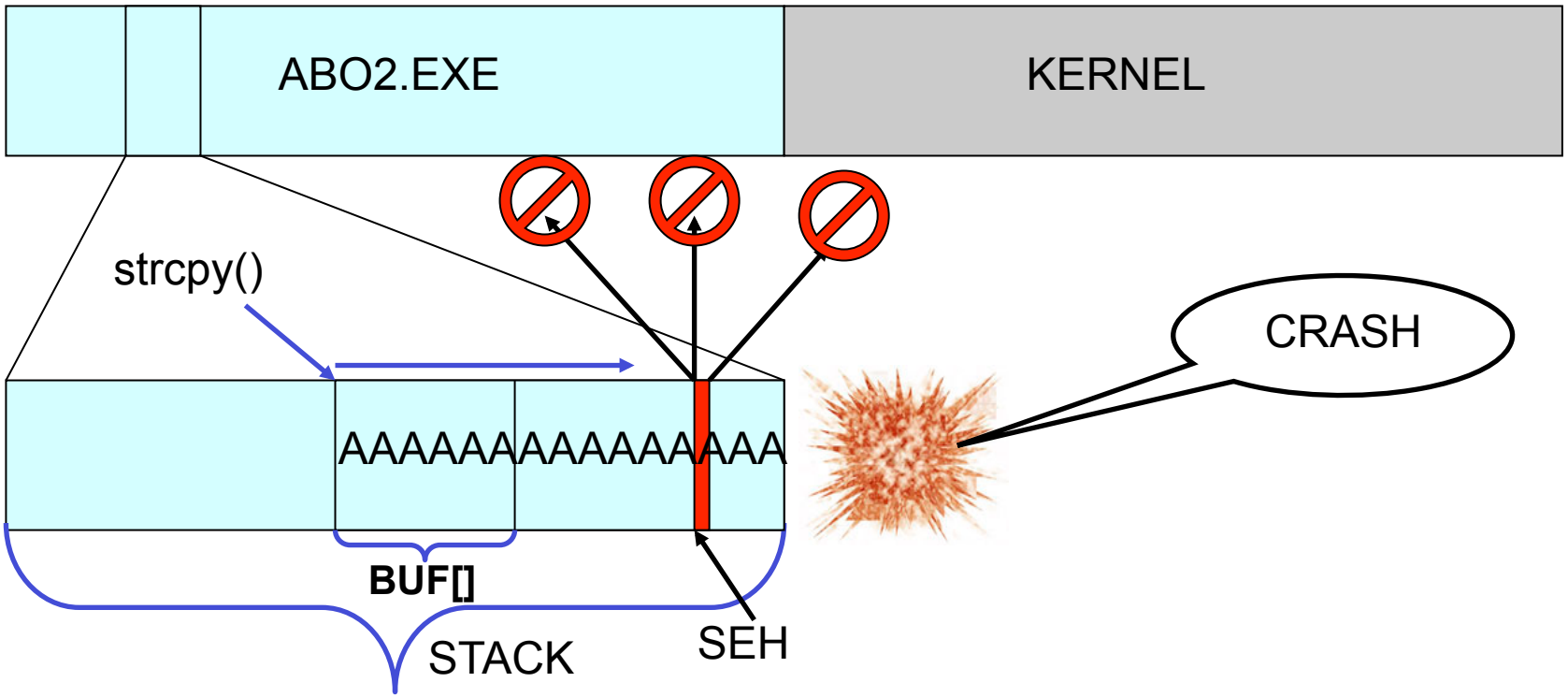
- **“abo2.c” (Advanced Buffer Overflow 2)**
  - It’s not exploitable on “Windows XP SP2 and SP3”

```
int main(int argv,char **argc)
{
    char buf[256];
    strcpy(buf,argc[1]);
    exit(1);
}
```

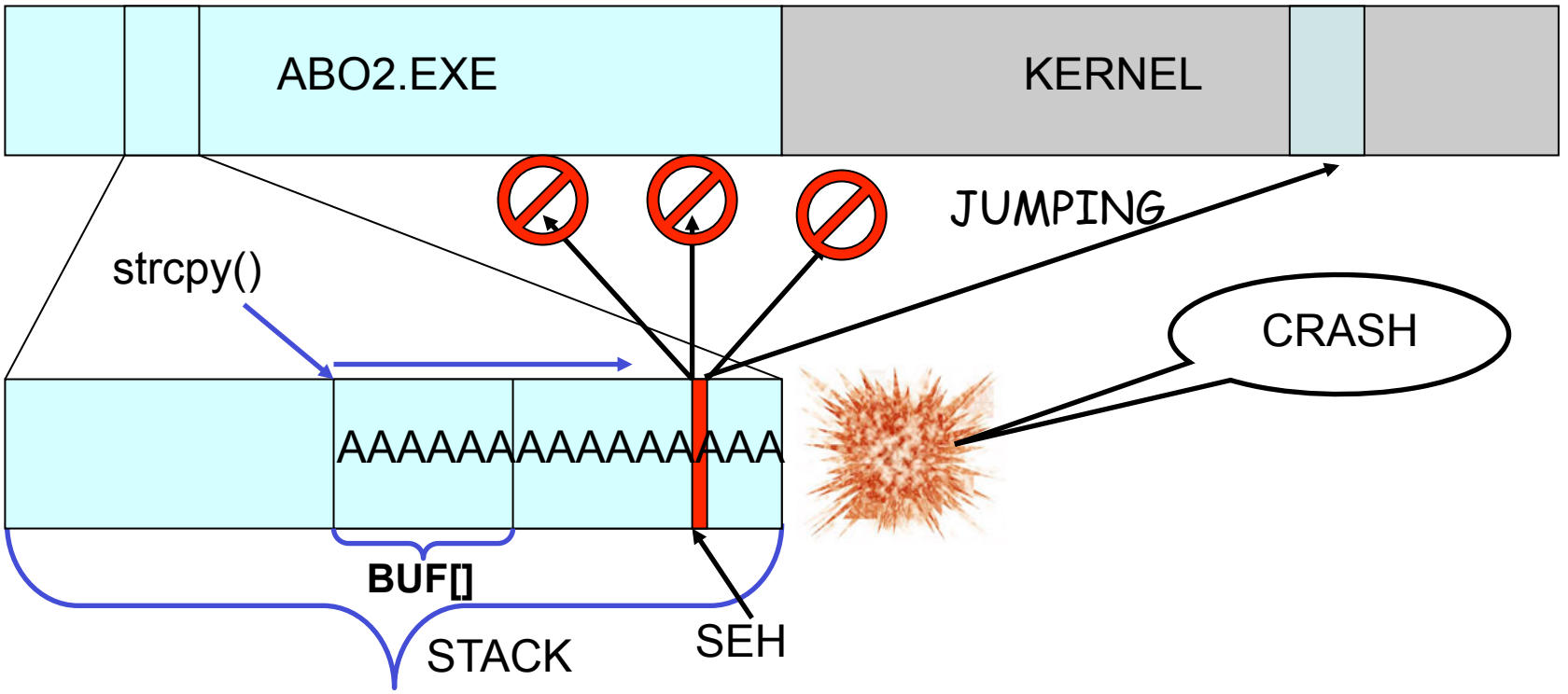
BUG

EXPLOIT USING
EXCEPTION HANDLING

- **“abo2.c” (Advanced Buffer Overflow 2)**
  - In fair conditions



- **“abo2.c” (Advanced Buffer Overflow 2)**
  - In fair conditions



- **DEMO**
  - "vp_abo2_launcher.exe"

1. It searches a writable memory address(xxxx)
2. It writes a "pop-pop-ret"
3. It executes '**abo2.exe** "AAAAAAxxxxBBBB"'

- **Conclusions**
  - Windows weakens if it's runs into Virtual PC
  - Privilege Escalations are TRIVIALS
  - This bug can be used remotely if you can brute force some

This bug is still unpatched, DON'T USE Virtual PC !

Questions?