# HYPERVISOR – NEW BATTLEFIELD FOR MALWARE GAME
# 虛擬機 - 惡意程式攻防的新戰場

王大寶, PK

# 虛擬機 - 惡意程式攻防的新戰場

- **講師簡介**
  王大寶, 小時候大家叫他王小寶,長大後就稱王大寶,目前隸屬一神祕單位. 雖然佯稱興趣在看書與聽音樂, 但是其實晚上都在打Game. 長期於系統最底層打滾,熟悉ASM,C/C++, 對於資安毫無任何興趣,也無經驗,純粹是被某壞人騙上台, 可以說是不可多得的素人講師!!

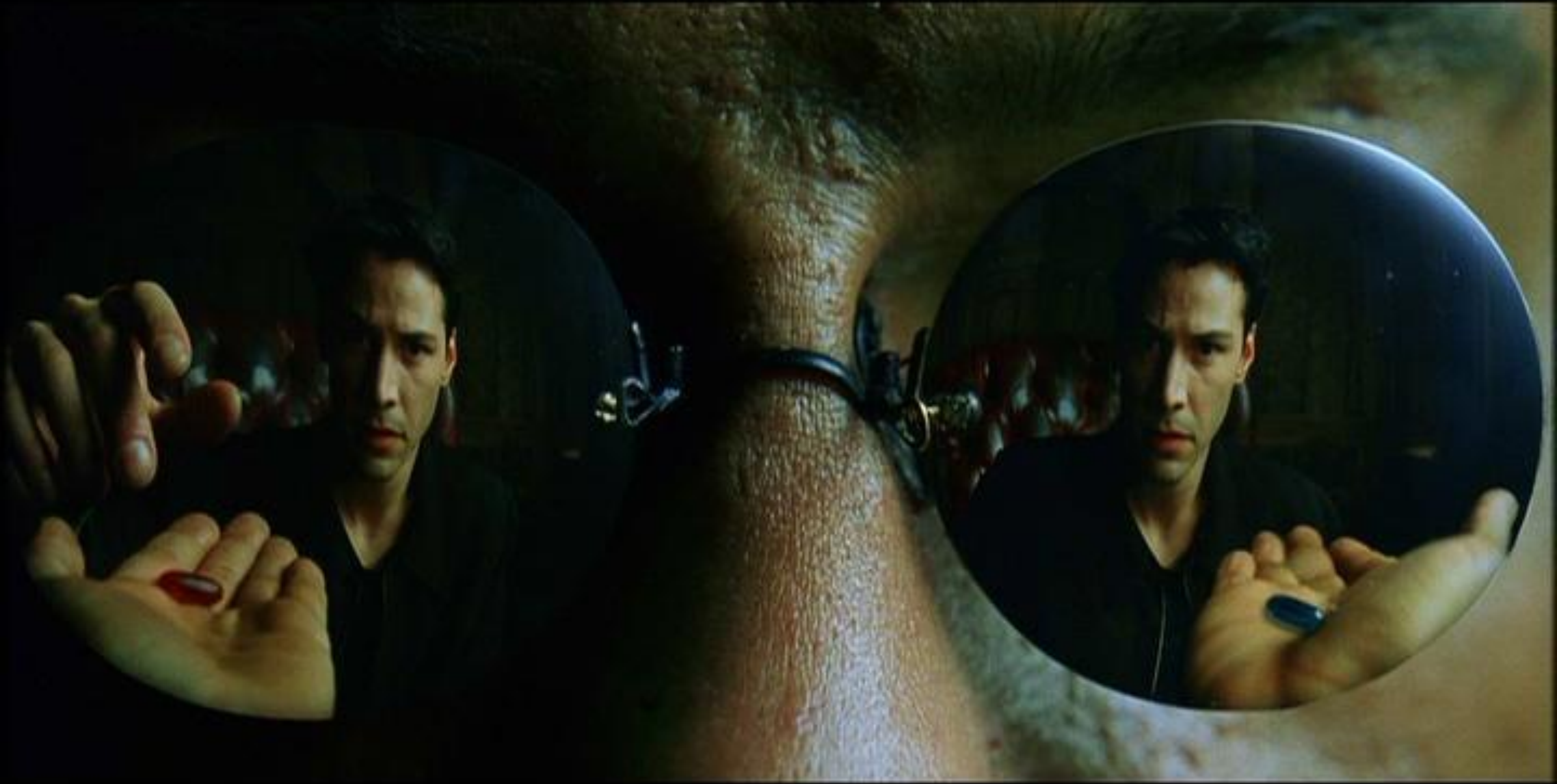
- **議程大綱：**
  現今的 CPU 都支援虛擬化專用指令集, 讓 VM 獲得硬體的支援. 在這個場次中,我們將詳細地介紹 Intel 的 VT指令集與其 Hypervisor 運作的機制. 此外我們將並介紹在惡意軟體研究領域中在 Hypervisor 模式下能有哪些應用,包含惡意程式技術與偵防分析的應用. 最後我們將介紹自行開發能在 Hypervisor 模式下運作的 Malware POC, 而且是無法被目前防毒與防護系統偵測到!

# Agenda

- VMM on x86
- Hardware assisted architecture
- VMM software implementing
- Security & VMM

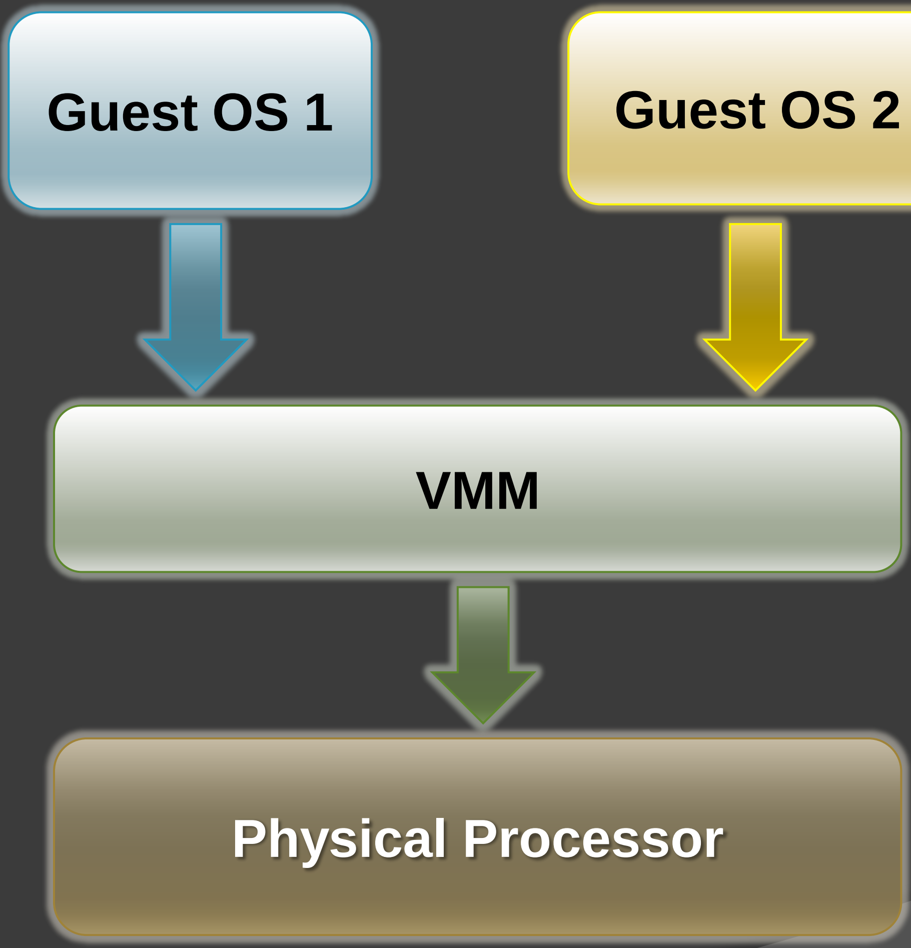
# What is VMM

- Has full control over the platform
- A thin layer between the physical hardware and virtualized environment
- Be able to retain selective control from guest software
- The real world



現實是殘酷的, 從VM中醒過來不一定是好事 ... :P

# What is VMM (conti.)



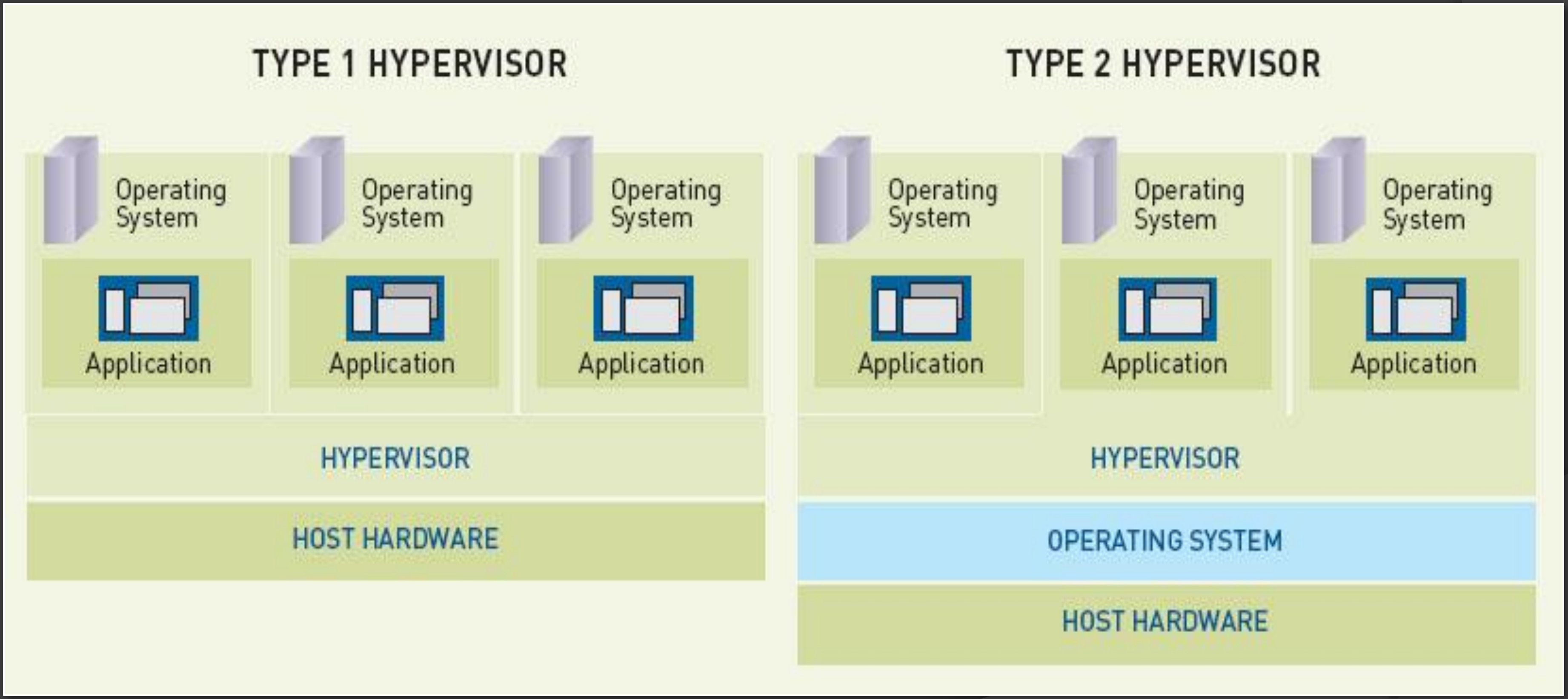
Guest OS 1

Guest OS 2

VMM

Physical Processor

# Types of Hypervisors



# Intel® VT-x

- Introduced by Intel®
- Includes a new set of instructions
- Totally isolated environments for each guest
- Solved many problems which were caused by guest OS executing at the same level of host OS
- Provides better performance than byte code emulation

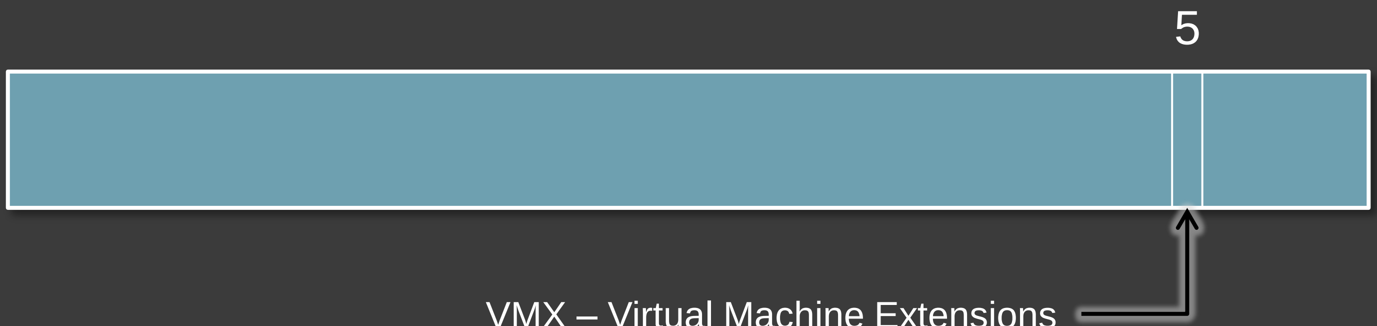
# Keywords

- VMM runs at VMX root operation
- Guest software runs at VMX non-root operation
- Transition from VMM to guest software is called VM entry
- Transition from guest software to VMM is called VM exit

# VMX root operation

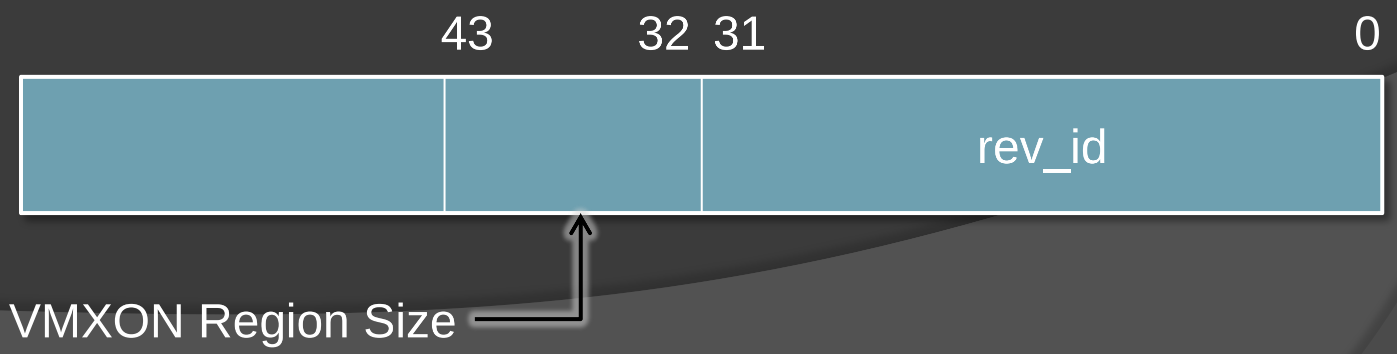
- Check CPU capabilities

```
mov     eax, 1
cpuid
test    ecx, 20h
```



# VMX root operation (conti.)

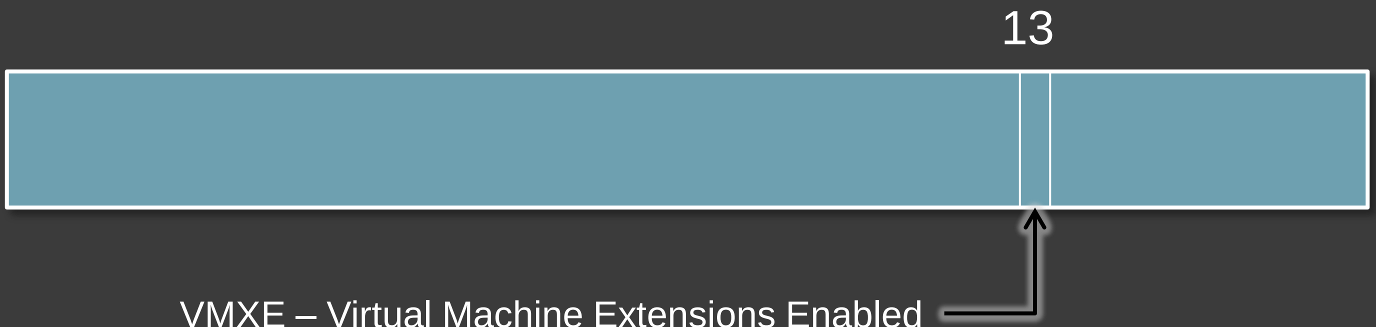
- Prepare a non-pageable memory (VMXON Region)
  - storage of host context
  - aligned to 4KB
  - in MTRR range – Write Back (type 6)
  - size = MSR#480 [43:32]
  - rev_id  = MSR#480 [31:0]



# VMX root operation (conti.)

- Enable VMXE bit (bit13) in CR4

```
mov     eax, cr4
or      eax, Bit13
mov     cr4, eax
```



# VMX root operation (conti.)

- VMXON instruction

```
vmxon  phymem_vmxon_region
```

- Hello, real world…

# VMX non-root operation

- Prepare a non-pageable memory (VMCS)
  - storage of guest software states
  - aligned to 4KB
  - in MTRR range – Write Back (type 6)
  - size = MSR#480 [43:32]
  - rev_id  = MSR#480 [31:0]

# VMX non-root operation (conti.)

- Instructions to initialize VMCS
  - VMCLEAR, VMPTRLD

- VMCLEAR
  - Initialize the new VMCS region in memory
  - Set the launch state to “clear”
  - Invalidates the working VMCS pointer register

- VMPTRLD
  - Initializes the working VMCS pointer with the new VMCS region's physical address.
  - Validates the working VMCS pointer register

# VMX non-root operation (conti.)

- Instructions to access specific field of VMCS
  - VMWRITE, VMREAD

- Each field has its encoding
  - Example:
    - GUEST_RIP = 681eh
    - To set GUEST_RIP into VMCS:

      ```
      mov         eax, 681eh
      vmwrite     eax, dword ptr NEW_GUEST_RIP
      ```

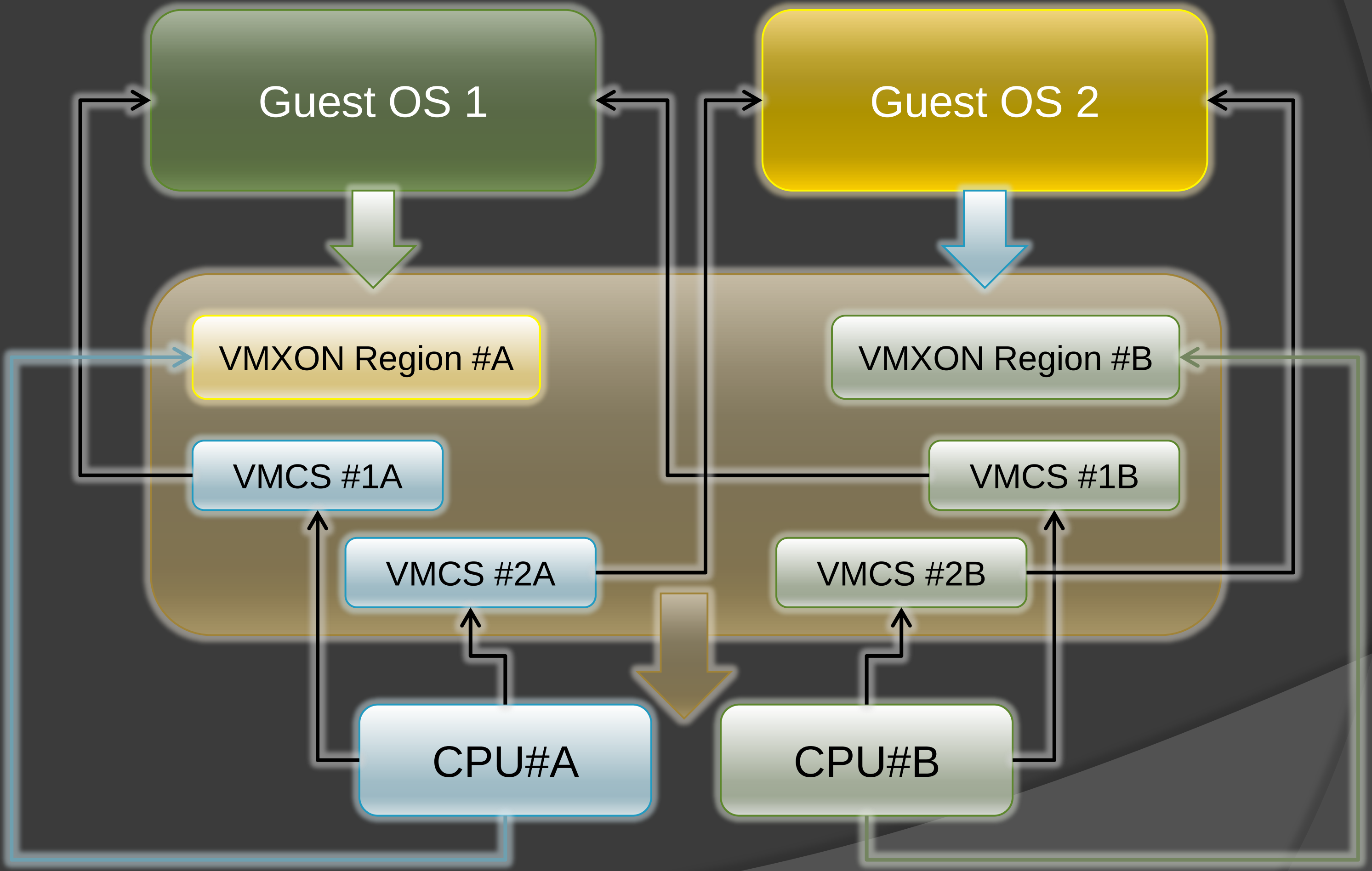
    - To get GUEST_RIP from VMCS:

      ```
      mov         eax, 681eh
      vmread      ebx, eax
      ```

# VMX non-root operation (conti.)

- Now it is time to run guest software
  - VMLAUNCH, VMRESUME
  - Launch state of VMCS will be set to “launched”

# VMM, VMCS, Guest OS



# VM exit handling

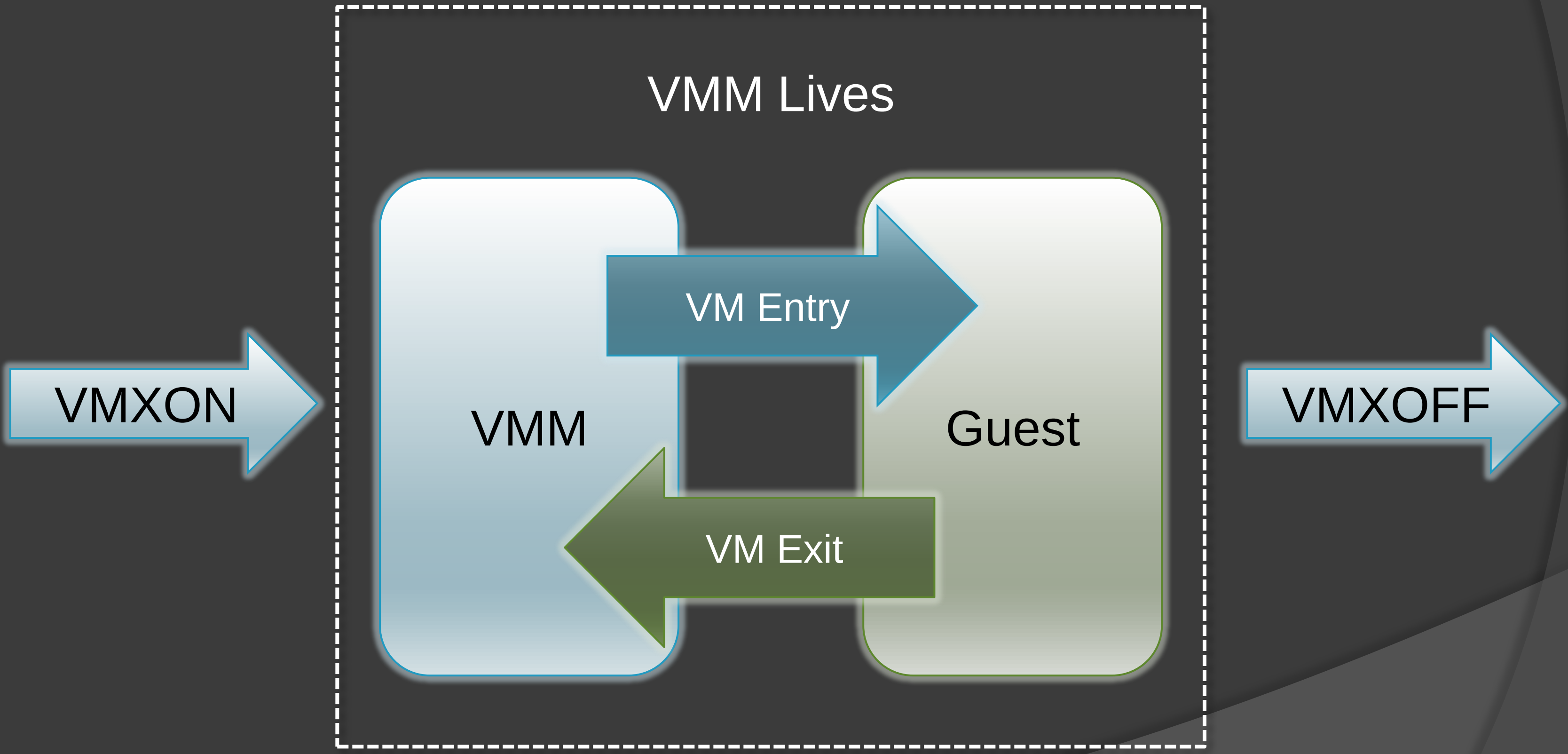
- VMM gets VM exit reason from VMCS, determines handle it or not

| Bit Position(s) | Contents |
|---|---|
| 15:0 | Basic exit reason |
| 27:16 | Reserved (cleared to 0) |
| 28 | Pending MTF VM exit |
| 29 | VM exit from VMX root operation |
| 30 | Reserved (cleared to 0) |
| 31 | VM-entry failure (0 = true VM exit; 1 = VM-entry failure) |

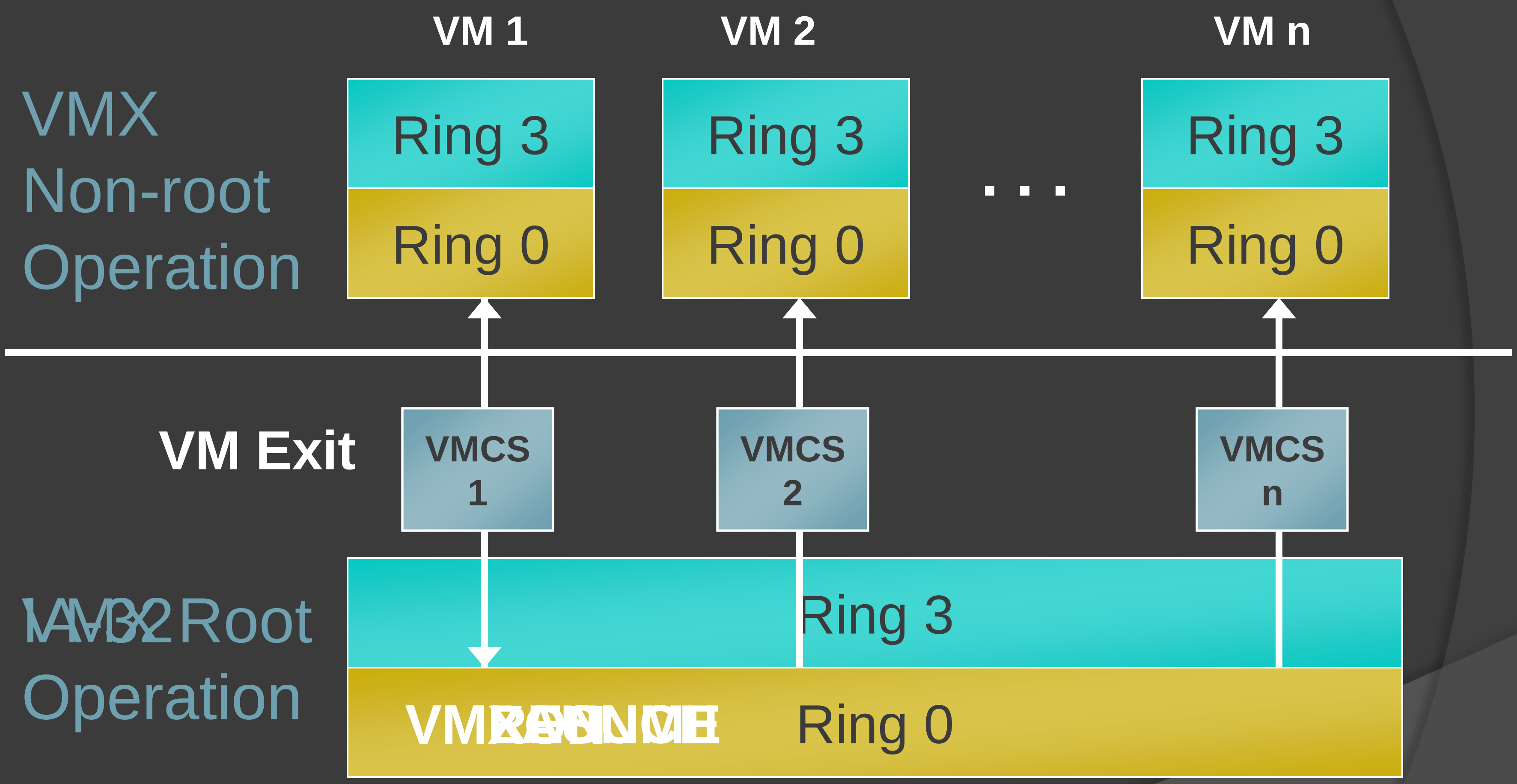
# VM exit handling (conti.)

- VM exit basic reasons
  - > 50
  - Sensitive instructions
  - Privilege registers change
  - Exceptions
  - …
- Exit qualification contains additional information
- Execute VMRESUME after handled VM exit

# Lifecycle of a VMM software



# VT-x Operations



# Security & VMM

- VMM is transparent to its guests
  - A well-implemented VMM is very hard to be detected
  - Almost all VMM-detection technologies in present are based on flaws of VMM itself
  - A positive usage of VMM could be a very powerful weapon against various attacks of malwares
  - So could be in either way…
  - But…

# Security & VMM (conti.)

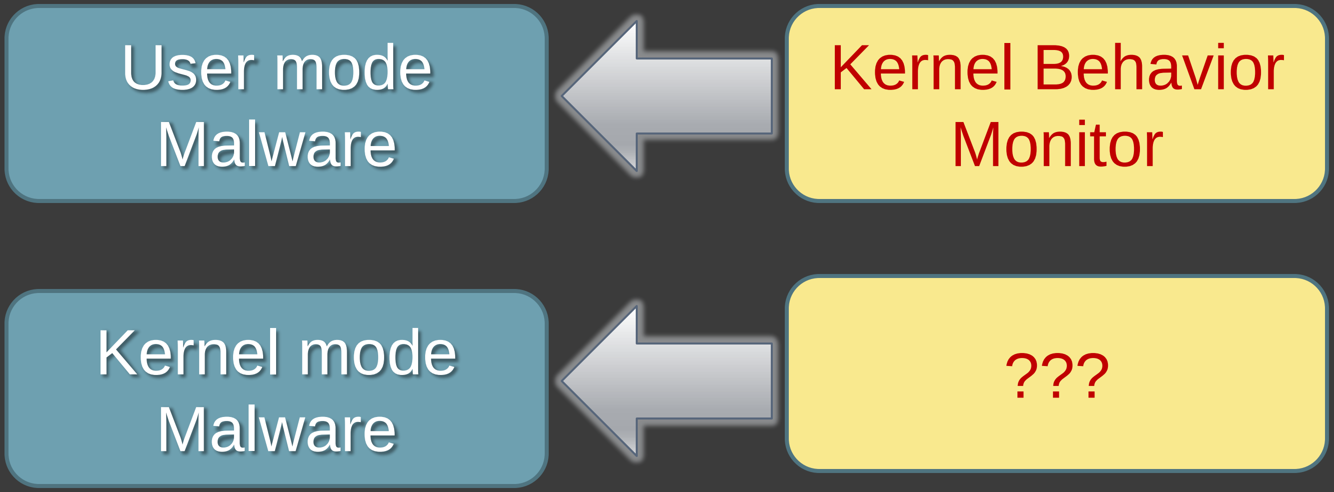
- Difficulties in implementing VMM
  - No OS API
  - No existed input/output
  - No existed drivers
  - Developers implement everything in VMM
    - Disk read/write
    - Keyboard input/output
    - Control video RAM for output
    - Direct manipulation on NIC, USB stack

# VMX vs. SMM

- In a software developer’s aspect, VMX operation is very similar to SMM
  - Transparent to client
  - Has processor context storage
  - Full control over system
  - Isolated environment, DIY everything
- Differences
  - SMM is triggered by hardware
  - SMM has higher priority than VMX
  - SMM is not accessible at runtime

# Malware and VMM

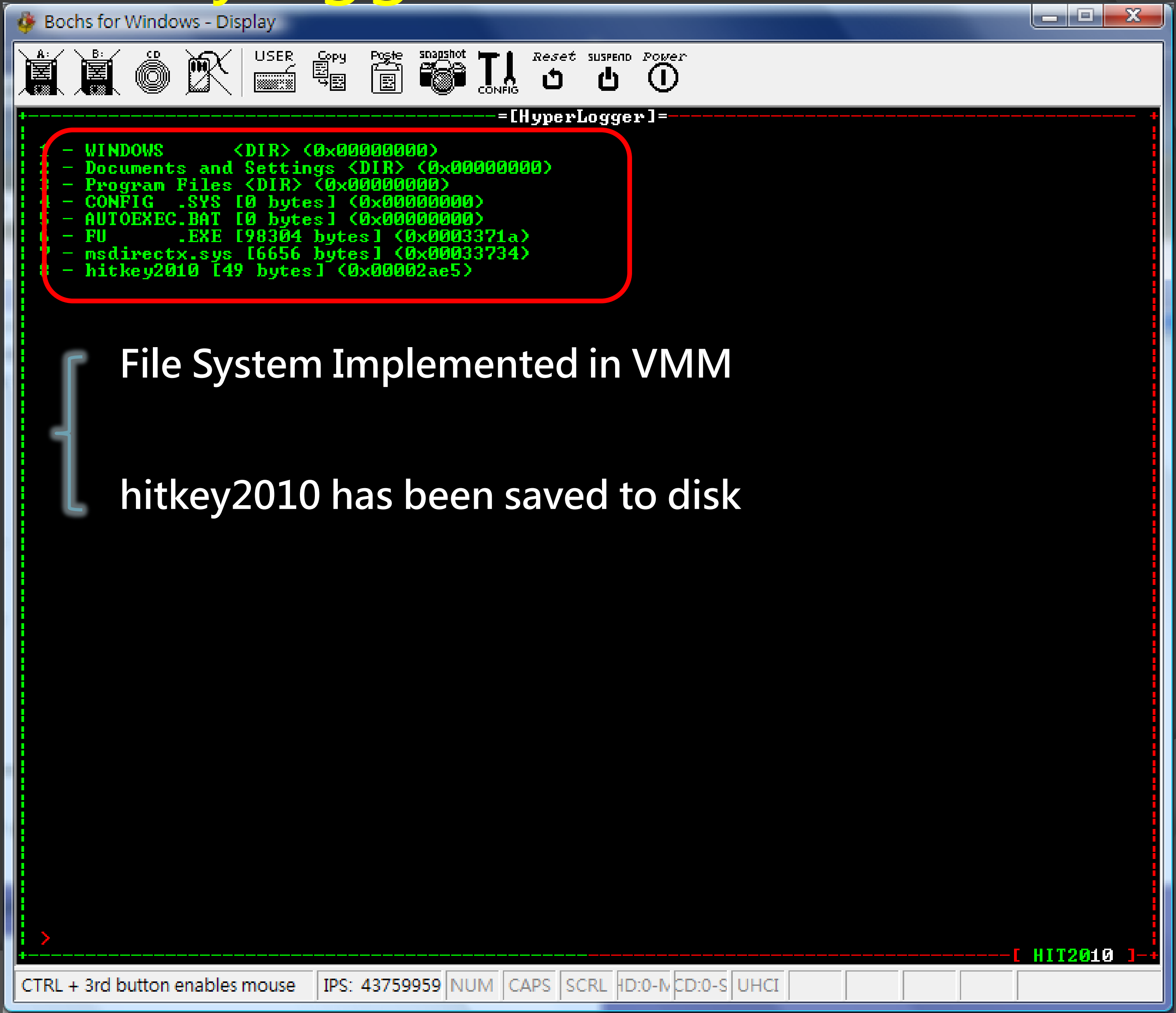
- How to detect or analysis Kernel Malware ??



# Demo 1: Invisible VMM Keylogger

- A handcrafted key logger in VMM
  - Capture KB input from I/O port
  - Hidden File in Guest OS File system !
  - Definitely invisible…Ya ☺
    - Cant be detected by any Anti-Virus or HIPS in the world

# VMM Keylogger



```
=[HyperLogger]=
1 - WINDOWS      <DIR> (0x00000000)
2 - Documents and Settings <DIR> (0x00000000)
3 - Program Files <DIR> (0x00000000)
4 - CONFIG   .SYS [0 bytes] (0x00000000)
5 - AUTOEXEC.BAT [0 bytes] (0x00000000)
6 - FU       .EXE [98304 bytes] (0x0003371a)
7 - msdirectx.sys [6656 bytes] (0x00033734)
8 - hitkey2010 [49 bytes] (0x00002ae5)

>
                                                  [ HIT2010 ]
```

CTRL + 3rd button enables mouse | IPS: 43759959 | NUM | CAPS | SCRL | HD:0-M | CD:0-S | UHCI

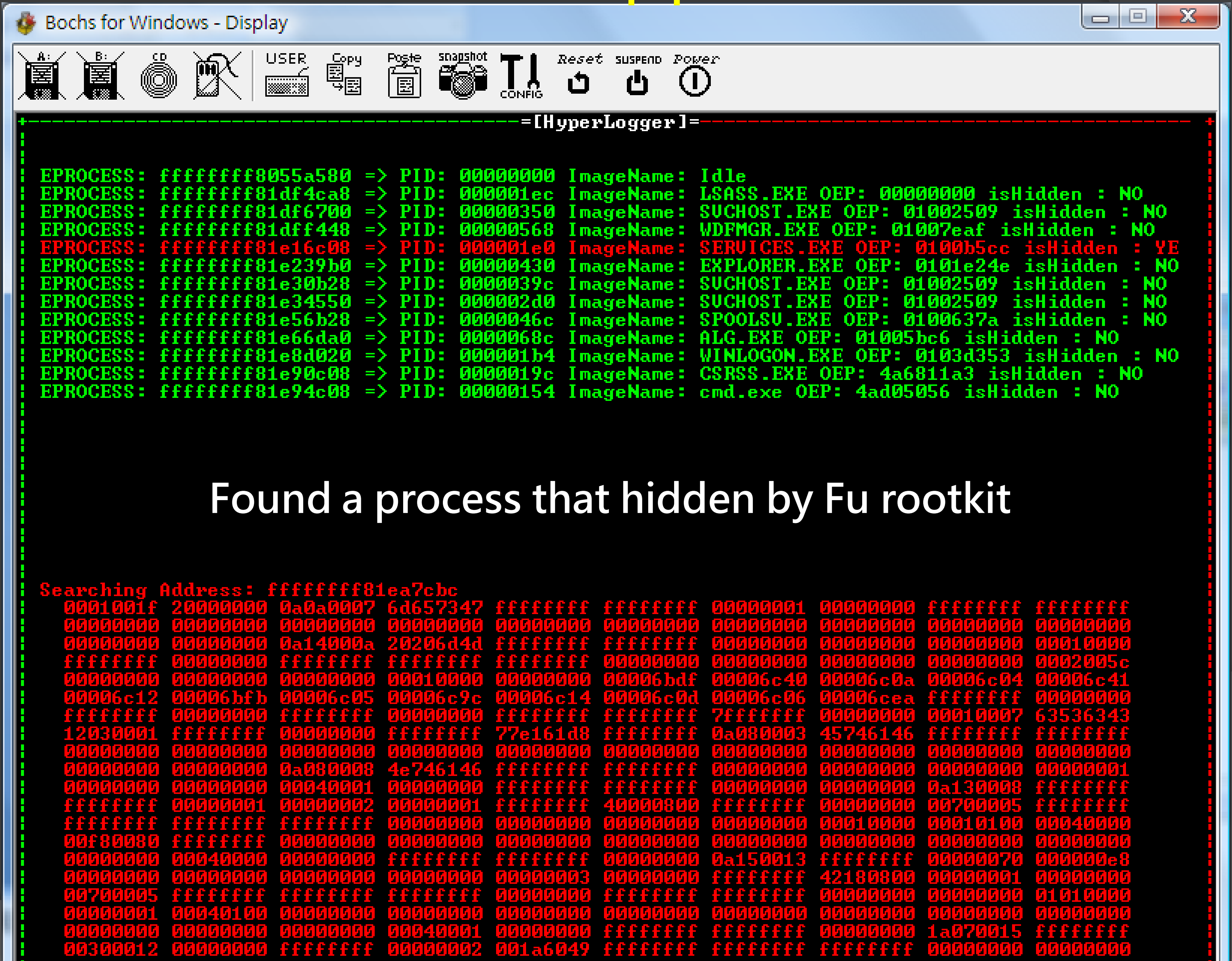
File System Implemented in VMM

hitkey2010 has been saved to disk

# Demo2: Rootkit Detection

- Physical Memory Forensics with VMM !!
  - EPROCESS parsing
  - SSDT parsing
  - Etc.
- Demo our new toy

# VMM on Forensic Approach



```
=[HyperLogger]=

EPROCESS: ffffffff8055a580 => PID: 00000000 ImageName: Idle
EPROCESS: ffffffff81df4ca8 => PID: 000001ec ImageName: LSASS.EXE OEP: 00000000 isHidden : NO
EPROCESS: ffffffff81df6700 => PID: 00000350 ImageName: SVCHOST.EXE OEP: 01002509 isHidden : NO
EPROCESS: ffffffff81dff448 => PID: 00000568 ImageName: WDFMGR.EXE OEP: 01007eaf isHidden : NO
EPROCESS: ffffffff81e16c08 => PID: 000001e0 ImageName: SERVICES.EXE OEP: 0100b5cc isHidden : YE
EPROCESS: ffffffff81e239b0 => PID: 00000430 ImageName: EXPLORER.EXE OEP: 0101e24e isHidden : NO
EPROCESS: ffffffff81e30b28 => PID: 0000039c ImageName: SVCHOST.EXE OEP: 01002509 isHidden : NO
EPROCESS: ffffffff81e34550 => PID: 000002d0 ImageName: SVCHOST.EXE OEP: 01002509 isHidden : NO
EPROCESS: ffffffff81e56b28 => PID: 0000046c ImageName: SPOOLSV.EXE OEP: 0100637a isHidden : NO
EPROCESS: ffffffff81e66da0 => PID: 0000068c ImageName: ALG.EXE OEP: 01005bc6 isHidden : NO
EPROCESS: ffffffff81e8d020 => PID: 000001b4 ImageName: WINLOGON.EXE OEP: 0103d353 isHidden : NO
EPROCESS: ffffffff81e90c08 => PID: 0000019c ImageName: CSRSS.EXE OEP: 4a6811a3 isHidden : NO
EPROCESS: ffffffff81e94c08 => PID: 00000154 ImageName: cmd.exe OEP: 4ad05056 isHidden : NO
```

Found a process that hidden by Fu rootkit

```
Searching Address: ffffffff81ea7cbc
   0001001f 20000000 0a0a0007 6d657347 ffffffff ffffffff 00000001 00000000 ffffffff ffffffff
   00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
   00000000 00000000 0a14000a 20206d4d ffffffff ffffffff 00000000 00000000 00000000 00010000
   ffffffff 00000000 ffffffff ffffffff ffffffff 00000000 00000000 00000000 00000000 0002005c
   00000000 00000000 00000000 00010000 00000000 00006bdf 00006c40 00006c0a 00006c04 00006c41
   00006c12 00006bfb 00006c05 00006c9c 00006c14 00006c0d 00006c06 00006cea ffffffff 00000000
   ffffffff 00000000 ffffffff 00000000 ffffffff ffffffff 7fffffff 00000000 00010007 63536343
   12030001 ffffffff 00000000 ffffffff 77e161d8 ffffffff 0a080003 45746146 ffffffff ffffffff
   00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
   00000000 00000000 0a080008 4e746146 ffffffff ffffffff 00000000 00000000 00000000 00000001
   00000000 00000000 00040001 00000000 ffffffff ffffffff 00000000 00000000 0a130008 ffffffff
   ffffffff 00000001 00000002 00000001 ffffffff 40000800 ffffffff 00000000 00700005 ffffffff
   ffffffff ffffffff ffffffff 00000000 00000000 00000000 00000000 00010000 00010100 00040000
   00f80080 ffffffff 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
   00000000 00040000 00000000 ffffffff ffffffff 00000000 0a150013 ffffffff 00000070 000000e8
   00000000 00000000 00000000 00000000 00000003 00000000 ffffffff 42180800 00000001 00000000
   00700005 ffffffff ffffffff ffffffff 00000000 ffffffff ffffffff 00000000 00000000 01010000
   00000001 00040100 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
   00000000 00000000 00000000 00040001 00000000 ffffffff ffffffff 00000000 1a070015 ffffffff
   00300012 00000000 ffffffff 00000002 001a6049 ffffffff ffffffff ffffffff 00000000 00000000
```

# Q&A

# Reference

- Intel ®64 and IA-32 Architectures Software Developer's Manual Vol.2, Vol.3
- http://code.google.com/p/hyperdbg/
- http://virtualizationtechnologyvt.blogspot.com/
- http://www.ibm.com/developerworks/cn/linux/l-cn-vt/index.html
- http://www.invisiblethingslab.com/