Breaking virtualization by switching to Virtual 8086 mode

Jonathan Brossard
CTO - P1 Code Security

jonathan@p1sec.com
endrazine@gmail.com

Agenda

- Virtualization : big picture
- Attack surface analysis
- The need for new tools
- Introducing Virtual 8086 mode
- Practical fuzzing with vm86()

Virtualization : time to care !

Market shares
Definitions

Virtualization : market shares

Source : Forrester Research 2009

78% of companies have production servers virtualized.

20% only have virtualized servers.

Virtualization : market shares

Source : Forrester Research 2009

VMWare is present in **98%** of the companies.
Microsoft virtualization products are used by 17%.
Citrix/Xen is used by 10%.

Bottom line...

Virtualization software are so widespread that they have become more attractive targets than say web, mail or dns servers !

There is a lower variety too !

Definitions

Virtualization : Definitions

Virtualization

Virtualization is the name given to the simulation with higher level components, of lower level components.

NOTE: Virtualization of applications (as opposed to full Oses) is out of topic.

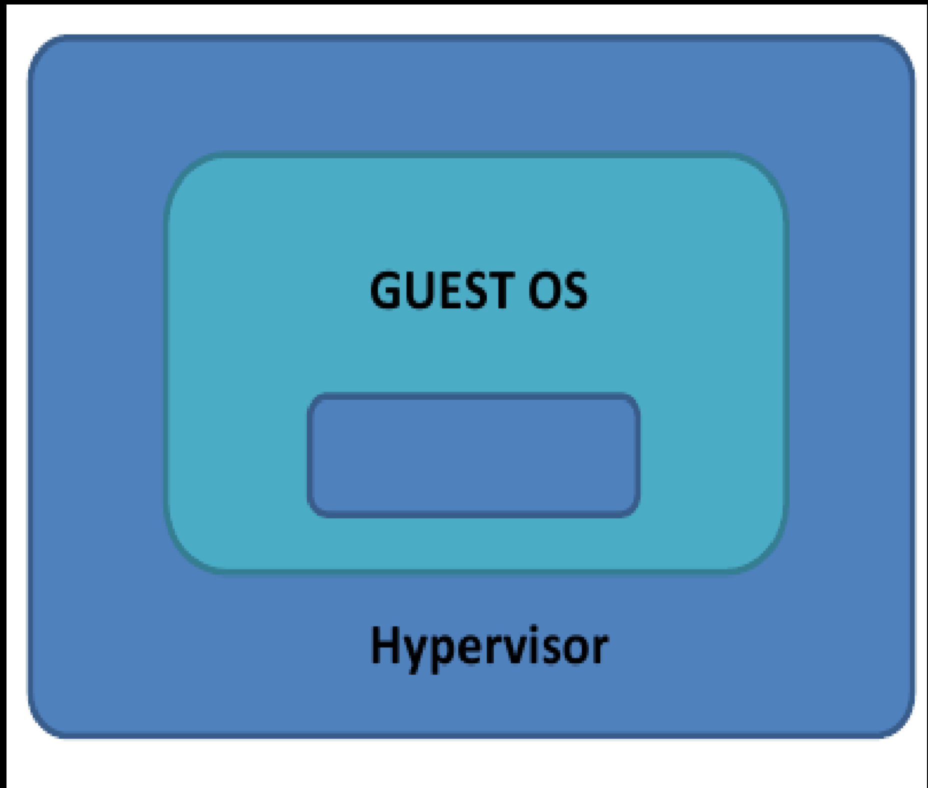
Virtualization : Definitions

Virtual Machine

A **virtual machine** (VM) is : "an efficient, isolated duplicate of a real machine".

-- Gerald J. Popek and Robert P. Goldberg (1974). "Formal Requirements for Virtualizable Third Generation Architectures", Communications of the ACM.

Paravirtualization



Virtualization : Definitions

Paravirtualization

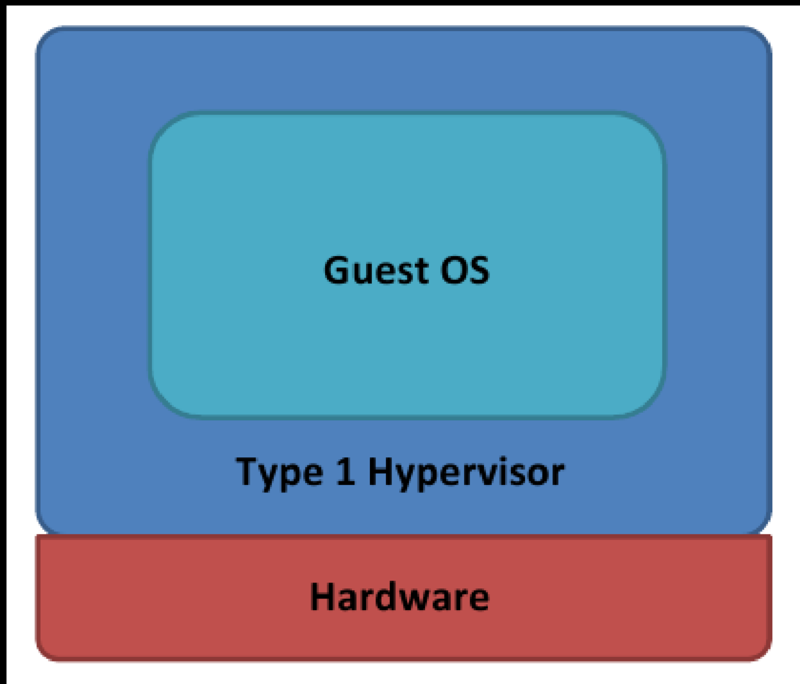
Requires the modification of the guest Oses (eg: Xen, UML, Qemu with kquemu, VMWare Workstation with VMWare Tools).

Opposed to « full virtualization ».

Virtualization : Definitions

There are two types of virtualizations : Virtual Machine Monitors (or **Hypervisors**) of **type I** and **type II**.

Type I Hypervisor

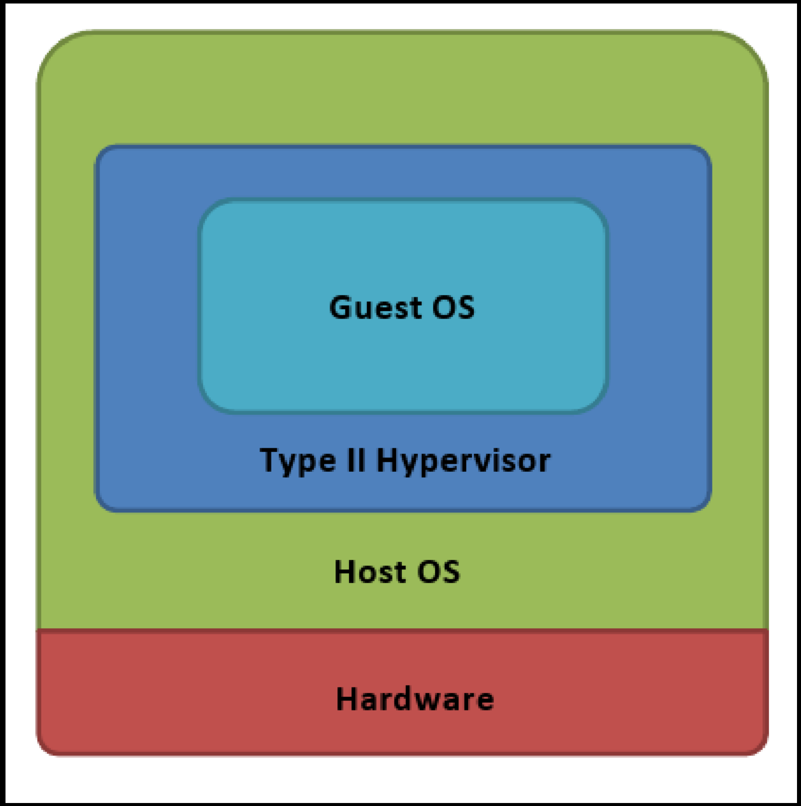


Virtualization : Definitions

Hypervisors of type I

Run on bare metal (eg: Xen, Hyper-V, VMWare ESX).

Type II hypervisor

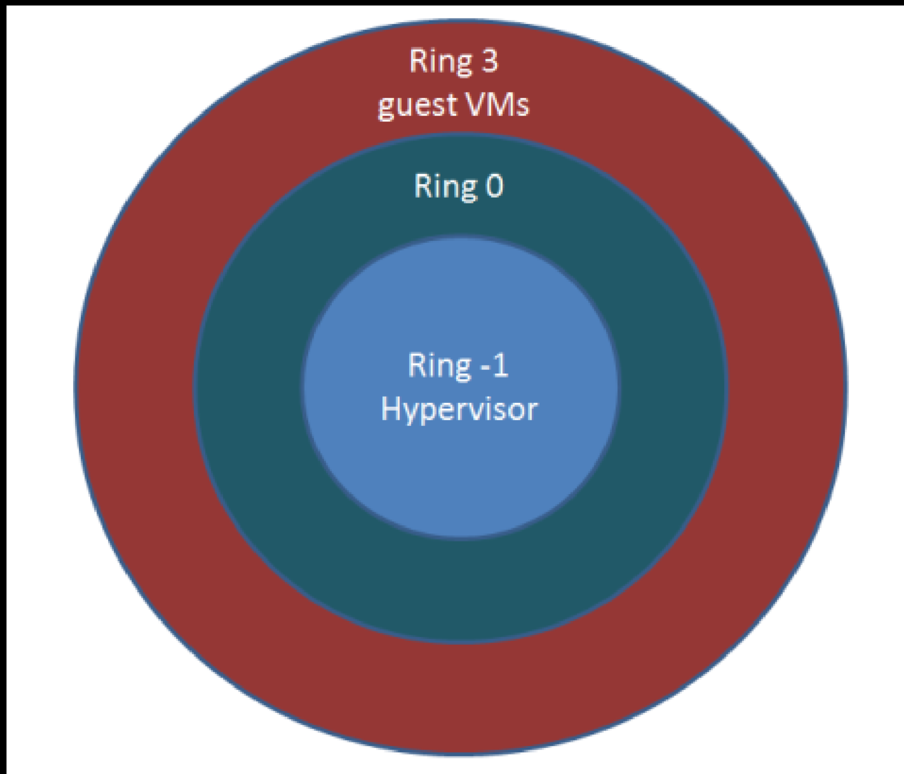


Virtualization : Definitions

Hypervizors of type II

Run as a process inside a host OS to virtualize guests Oses (eg: Qemu, Virtualbox, VMWare Workstation, Parallels).

Hardware assisted virtualization



Hardware assisted virtualization

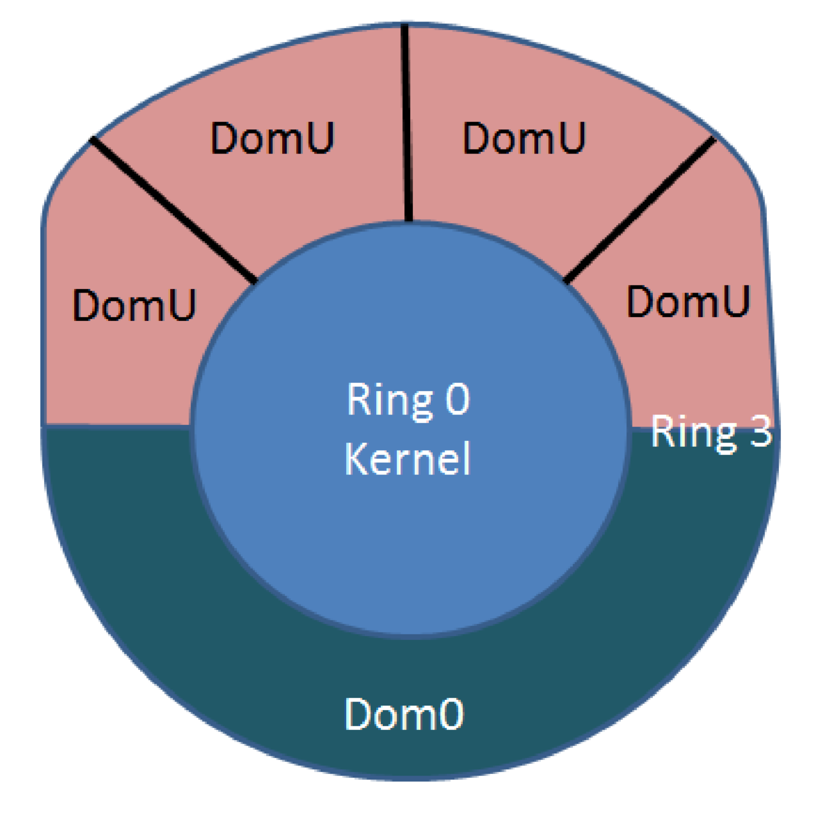
- Takes advantage of AMD-V On Intel VT-x CPU extentions for virtualization.
- x64 Only.
- The hypervizor is running in « ring -1 ».
- Much like the NX bit : requires the motherboard to support it and activation in the BIOS.

Virtualization : Definitions

Isolation

Isolation of the userland part of the OS to simulate independant machines (eg: Linux-Vservers, Solaris « Zones », BSD « jails », OpenVZ under GNU/Linux).

Isolation



Attack surface analysis

Depending on your perspective…

What are the risks ?
Where to attack ?

Privilege escalation on the host

VMware Tools HGFS Local Privilege Escalation Vulnerability

(http://labs.idefense.com/intelligence/vulnerabilities/display.php?id=712)

Privilege escalation on the Guest

CVE-2009-2267 « Mishandled exception on page fault in VMware » Tavis Ormandy and Julien Tinnes

Attacking other guests

Vmare workstation guest isolation weaknesses (clipboard transfer)

http://www.securiteam.com/securitynews/5GP021FKKO.html

DoS (Host + Guests)

CVE-2007-4591 CVE-2007-4593 (bad ioctls crashing the Host+Guests)

Escape to host

Rafal Wojtczuk (Invisible things, BHUS 2008)

IDEFENSE VMware Workstation Shared Folders Directory Traversal Vulnerability (CVE-2007-1744)

Attack surface analysis : usage

Hosting two companies on the same hardware is <u>very common</u> (shared hosting).

Getting a shell on the same machine as a given target may therefor be a matter of paying a few euros a month.

Attack surface : conclusion

Owning the Host OS from the Guest is practical : security through virtualization is a failure.

Seemingly minor bugs (local, DoS) do matter : virtualization amplifies consequences.

The need for dedicated methodologies and tools

The need for new tools : example

How to dynamically test a virtual Hard Drive ?

How to dynamically test a virtual Hard Drive ? Naive approach

Standard API :

```
ssize_t read(int fd, void *buf, size_t count);
ssize_t write(int fd, const void *buf, size_t count);
```

This would mostly fuzz the kernel, not the Virtual Machine :(

We need something (much) lower level.

Standard (low level) attack vectors

Ioports:

outb, outw, outl, outsb, outsw, outsl,
inb, inw, inl, insb, insw, insl, outb_p,
outw_p, outl_p, inb_p, inw_p, inl_p

Problems: sequence, multiple ports

Ioctls:

int ioctl(int d, int request, ...)

Problems : arbitrary input size !

How did we used to do it « back in the days » ?

MS Dos : direct access to the hardware (interrupts : BIOS, HD, Display, …)

Can we get back to this ?

Introducing the Virtual 8086 mode

Introducing the Virtual 8086 mode

Introduced with Intel 386 (1985)

Introducing the Virtual 8086 mode

Intel x86 cpus support 3 modes

- Protected mode
- Real mode
- System Management Mode (SMM)

Introducing the Virtual 8086 mode

Protected mode

This mode is the native state of the processor. Among the capabilities of protected mode is the ability to directly execute “real-address mode” 8086 software in a protected, multi-tasking environment. This feature is called virtual-8086 mode, although it is not actually a processor mode. Virtual-8086 mode is actually a protected mode attribute that can be enabled for any task.

Introducing the Virtual 8086 mode

Real-address mode

This mode implements the programming environment of the Intel 8086 processor with extensions (such as the ability to switch to protected or system management mode). The processor is placed in real-address mode following power-up or a reset.

Introducing the Virtual 8086 mode

System management mode (SMM)

This mode provides an operating system or executive with a transparent mechanism for implementing platform specific functions such as power management and system security. The processor enters SMM when the external SMM interrupt pin (SMI#) is activated or an SMI is received from the advanced programmable interrupt controller (APIC).

Nice things about Real mode / Virtual 8086 mode

Direct access to hardware via interruptions !

example:

```
Mov ah, 0x42 ; read sector from drive
Mov ch, 0x01  ; Track
Mov cl, 0x02   ; Sector
Mov dh, 0x03 ; Head
Mov dl, 0x80  ; Drive (here first HD)
Mov bx, offset buff ; es:bx is destination

Int 0x13            ; hard disk operation
```

Complexity

ax*bx*cx*dx (per interruption)

Id est: [0;65535]^4 ~ 1.8 * 10^19

=> still huge

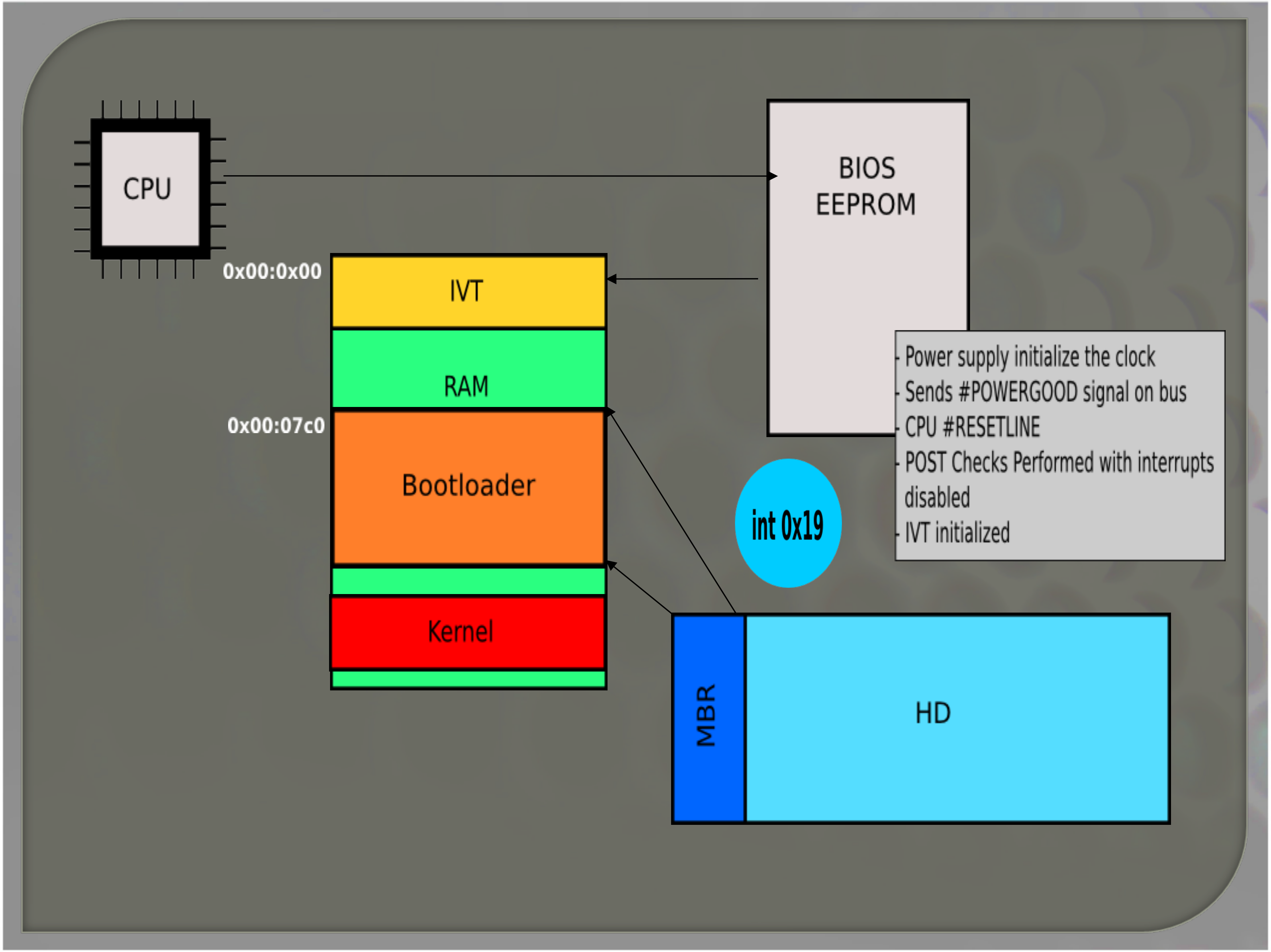
=> much better than ioctl()'s arbitrary input length !

Introducing the Virtual 8086 mode

Problem is… is this even possible inside a virtual machine ?

Introducing the Virtual 8086 mode

A closer look at the boot sequence…


CPU
BIOS
EEPROM
0x00:0x00
IVT
RAM
0x00:07c0
Bootloader
Kernel
- Power supply initialize the clock
- Sends #POWERGOOD signal on bus
- CPU #RESETLINE
- POST Checks Performed with interrupts disabled
- IVT initialized
int 0x19
MBR
HD

Introducing the Virtual 8086 mode

The kernel boots in (16b) real mode, and then switches to protected mode (32b).

The cpu normally doesn't get back to real mode untill next reboot.

Introducing the Virtual 8086 mode

Corollary

The hypervisor could run under any mode. protected mode in practice (being it ring0, ring1 or ring3).

All of the guests run <u>only</u> in protected mode.

Now how to swith to Virtual 8086 mode ? It this even possible ?

Leaving protected mode ?

```
        ------------------                     SMI (interrupt)
     |->|Real Address Mode| -------------------------------------|
     |  ------------------ <--------------------------------|    |
     |  | PE=1     ^ PE=0 (requires ring0) or                |rsm or |
     |  v          | reset                                   |reset  V
     |  ------------------                                   ---------
reset | | Protected Mode  | -------> SMI (interrupt) ------> | SMM Mode |
     |  ------------------ <------- rsm instruction <------  ---------
     |  | VM=1     ^ VM=0                                    |       ^
     |  v          |                                         |rsm    |
     |  ------------------ <--------------------------------|       |
     |- |Virtual 8086 Mode| -------------------------------------|
        ------------------                     SMI (interrupt)
```

(Ascii Art : Courtesy of phrack 65)

Setting the VM flag in CR0 under protected mode would get us to Virtual Mode
Removing the PE flag from CR0 would get us back to real mode

Leaving protected mode ?

linux-2.6.31/arch/x86/kernel/reboot.c:

```
static const unsigned char real_mode_switch [] =
{
    0x66, 0x0f, 0x20, 0xc0,                    /*    movl  %cr0,%eax        */
    0x66, 0x83, 0xe0, 0x11,                    /*    andl  $0x00000011,%eax */
    0x66, 0x0d, 0x00, 0x00, 0x00, 0x60,        /*    orl   $0x60000000,%eax */
    0x66, 0x0f, 0x22, 0xc0,                    /*    movl  %eax,%cr0        */
    0x66, 0x0f, 0x22, 0xd8,                    /*    movl  %eax,%cr3        */
    0x66, 0x0f, 0x20, 0xc3,                    /*    movl  %cr0,%ebx        */
    0x66, 0x81, 0xe3, 0x00, 0x00, 0x00, 0x60,  /*    andl  $0x60000000,%ebx */
    0x74, 0x02,                                /*    jz    f                */
    0x0f, 0x09,                                /*    wbinvd                 */
    0x24, 0x10,                                /* f: andb  $0x10,al         */
    0x66, 0x0f, 0x22, 0xc0                     /*    movl  %eax,%cr0        */
};
```

Trouble is…

This obviously won't work inside a virtual machine !

Because CR[1-4] registers are themselves emulated

IS THIS « GAME OVER » ?

Actually not quite ...

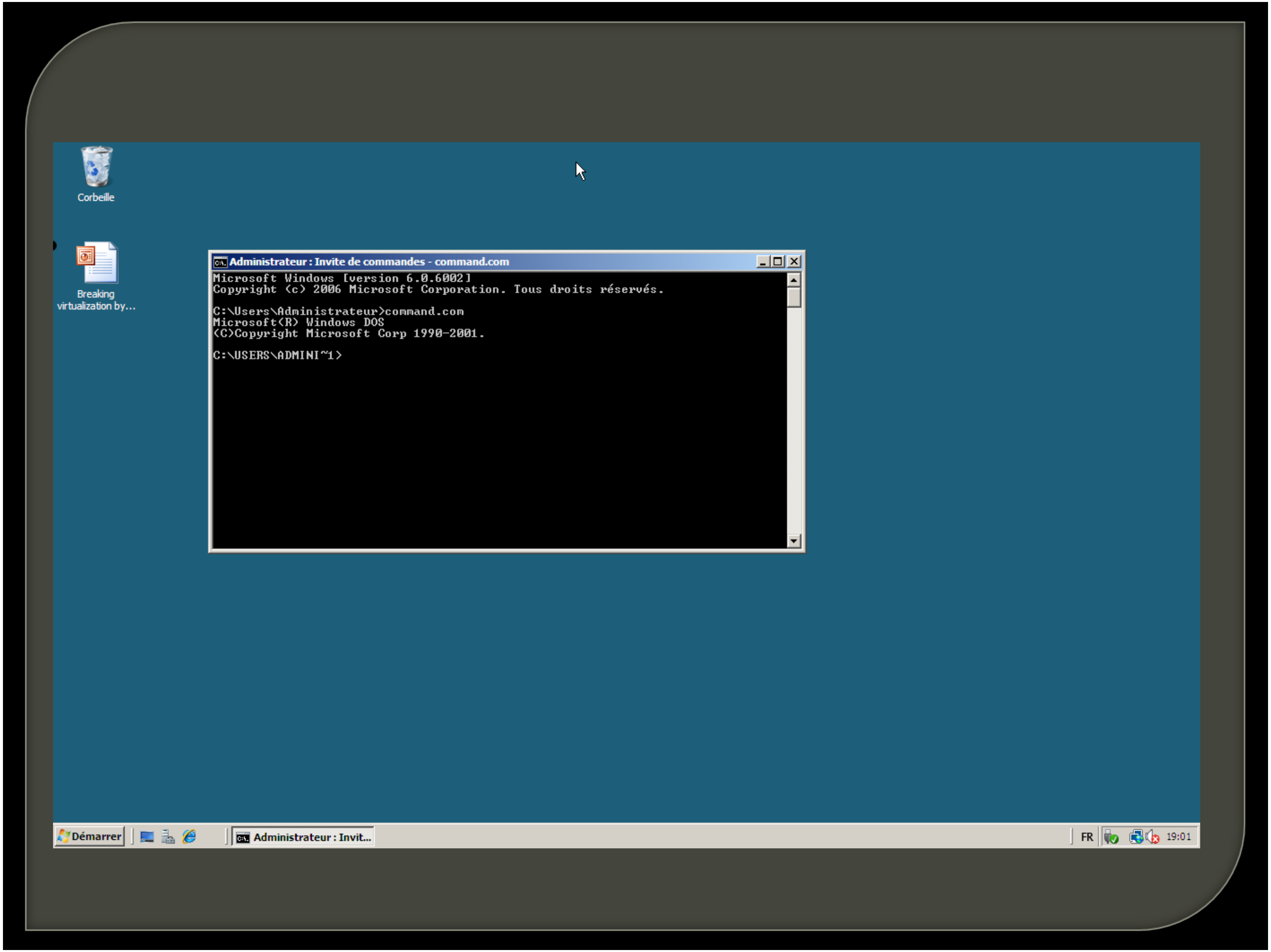
Truth is : we don't need to switch back to real mode/ virtual 8086 mode !

Most Operating systems offer a way to run 16b applications (eg: MS DOS) under protected mode by emulating a switch to Virtual 8086 Mode.

Notably Windows (x86) and Linux (x86).

The Windows case

NTVDM : ntvdm.exe
« Windows 16b Virtual Machine »


Corbeille
Breaking virtualization by...
Administrateur : Invite de commandes - command.com
Microsoft Windows [version 6.0.6002]
Copyright (c) 2006 Microsoft Corporation. Tous droits réservés.
C:\Users\Administrateur>command.com
Microsoft(R) Windows DOS
(C)Copyright Microsoft Corp 1990-2001.
C:\USERS\ADMINI~1>
Démarrer
Administrateur : Invit...
FR
19:01

The Linux case

The linux kernel provides an emulation of real mode in the form of two syscalls:

```
#define __NR_vm86old    113
#define __NR_vm86       166
```

The Linux case

```
#include <sys/vm86.h>

int vm86old(struct vm86_struct *info);

int vm86(unsigned long fn, struct
vm86plus_struct *v86);
```

```
struct vm86_struct {
        struct vm86_regs regs;
        unsigned long flags;
        unsigned long screen_bitmap;
        unsigned long cpu_type;
        struct revectored_struct
                int_revectored;
        struct revectored_struct
        int21_revectored;
};
```

The Linux case

linux-2.6.31/arch/x86/include/asm/vm86.h:

```
struct vm86_regs {
        long ebx;
        long ecx;
        long edx;
        long esi;
        long edi;
        long ebp;
        long eax;
    (…)
        unsigned short es, __esh;
        unsigned short ds, __dsh;
        unsigned short fs, __fsh;
        unsigned short gs, __gsh;

};
```

In a nutshell

- The switch to Virtual mode is entirely emulated by the kernel (this will work inside a VM)
- We can still program using old school interruptions (easy !)
- Those interruptions are delivered to the hardware (id est: either the emulated one, or the real one).

=> We just got a « bare metal (possibly virtualized) hardware interface »

The x64 case…

The x64 case

X64 cpus in 64b long mode can't swith to Virtual mode.

That's too bad : we'd like to fuzz latest Vmware ESX or Microsoft HyperV (necessarily under x64).

But under virtualization, the switch to VM86 mode is being emulated by the kernel...

The x64 case

Using kernel patches, we can add VM86 capabilities to a x64 GNU/Linux kernel.

EG: http://v86-64.sourceforge.net to run Dosemu under x64.

What's not possible in real hardware becomes possible under a virtualized environment !

Practical use : Fuzzing using vm86()

Practical use : Fuzzing using vm86()

Looking at the IVT allows us to fuzz all the hardware know after BIOS Post, efficently (no calls to empty/ dummy interrupts).

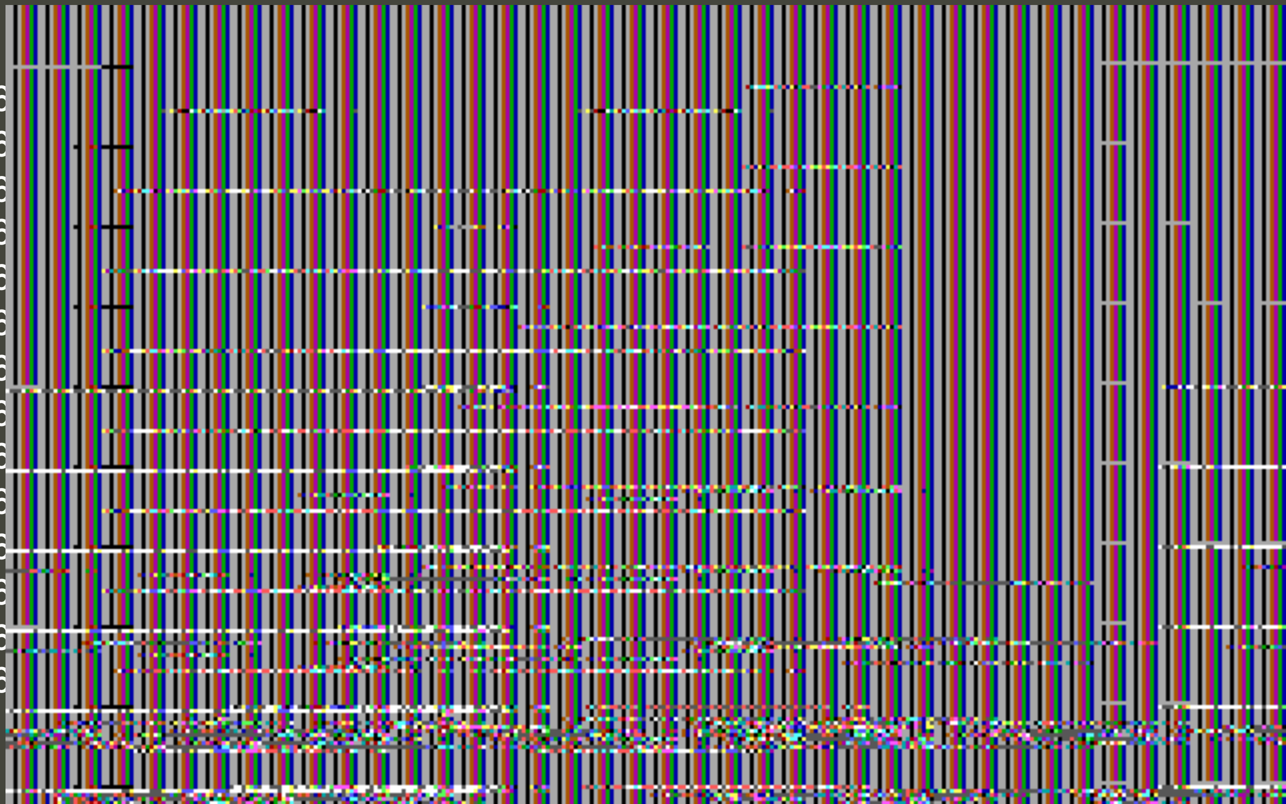
Practical use : Fuzzing using vm86()

Exemple bugs !

Practical use : Fuzzing using vm86()

Bugs in hypervizors…

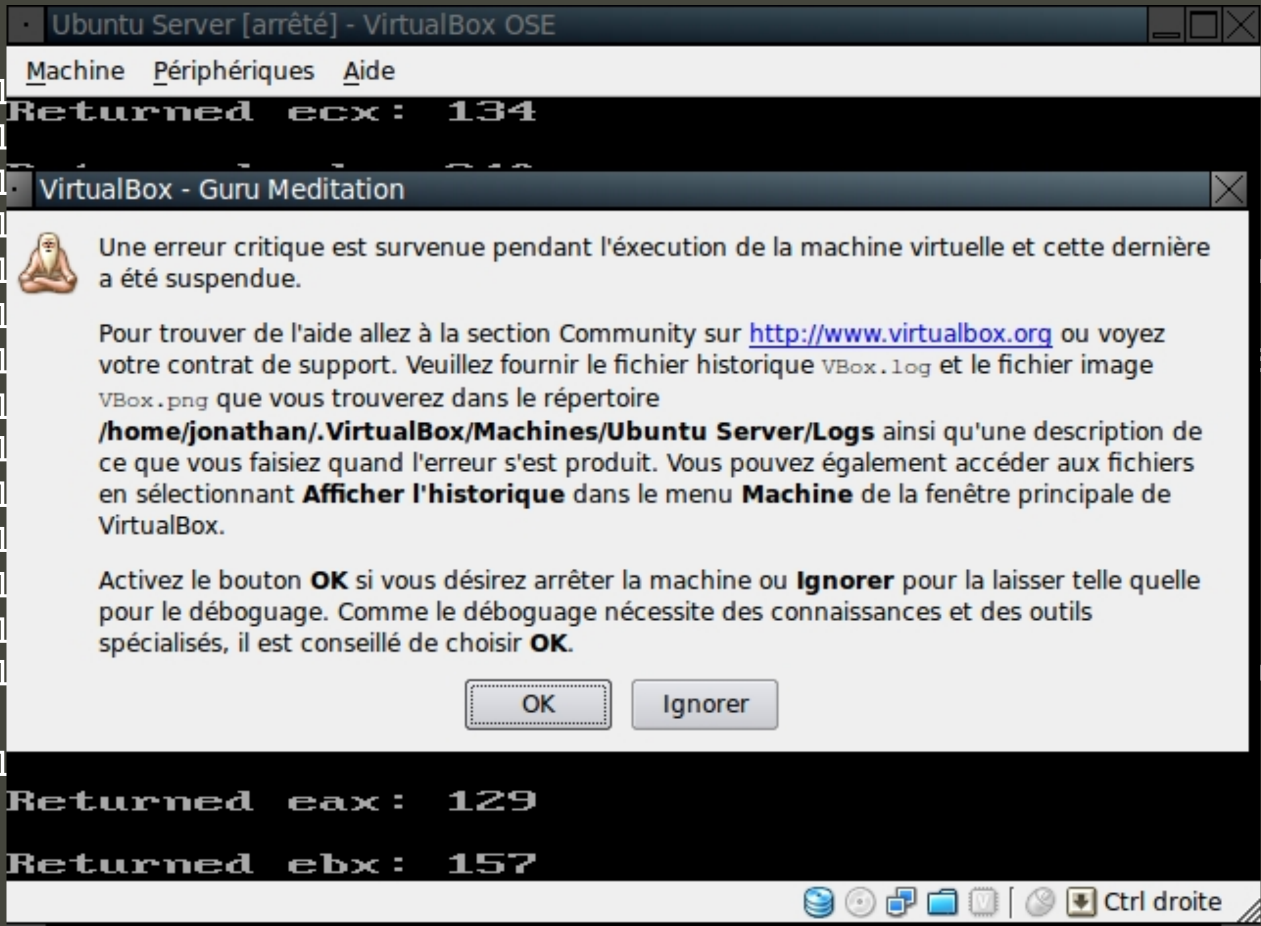
Virtualbox



00:21:13.6
00:21:13.6
00:21:13.6
00:21:13.6
00:21:13.6 215e33 Type=0
00:21:13.6
00:21:13.6 TATEyPj
00:21:13.6
00:21:13.6
00:21:13.6
00:21:13.6
00:21:13.6
00:21:13.6
00:21:13.6 00001 .edi=fec0
1000

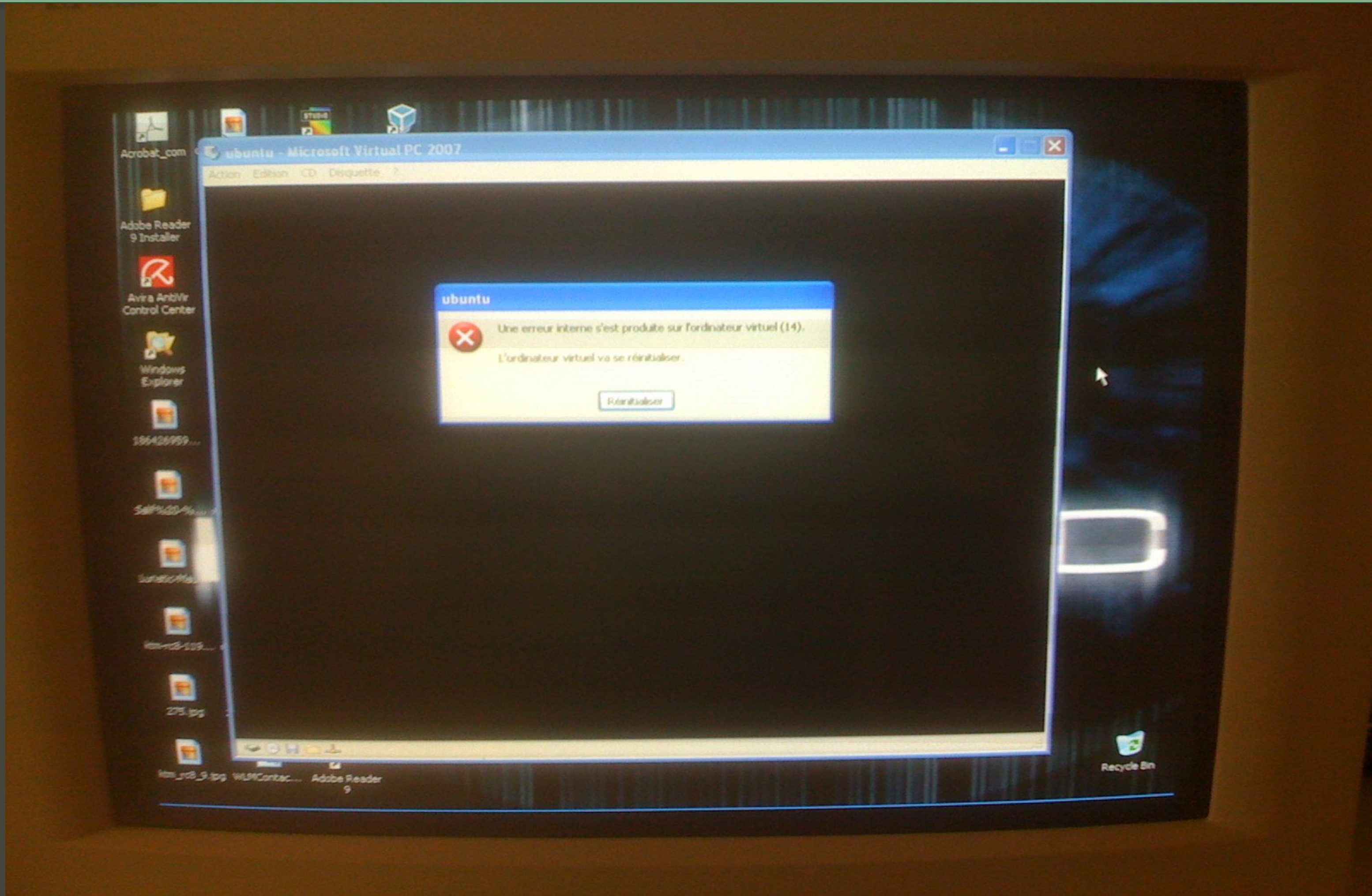
Virtualbox (take 2)

00:02:51.1
00:02:51.1
00:02:51.1
00:02:51.1
00:02:51.1 ff215e33 Type=0
00:02:51.1
00:02:51.1 STATEyPj
00:02:51.1
00:02:51.1
00:02:51.1
00:02:51.1
00:02:51.1
00:02:51.1
00:02:51.1 000001 .edi=fec0
1000
00:02:51.1 zr ac pe cy



More (guest) bugs

Virtual PC


ubuntu - Microsoft Virtual PC 2007
Action Edition CD Disquette ?
ubuntu
Une erreur interne s'est produite sur l'ordinateur virtuel (14).
L'ordinateur virtuel va se réinitialiser.
Réinitialiser
Adobe Reader 9 Installer
Avira AntiVir Control Center
Windows Explorer
Recycle Bin

Parallels (Guest)

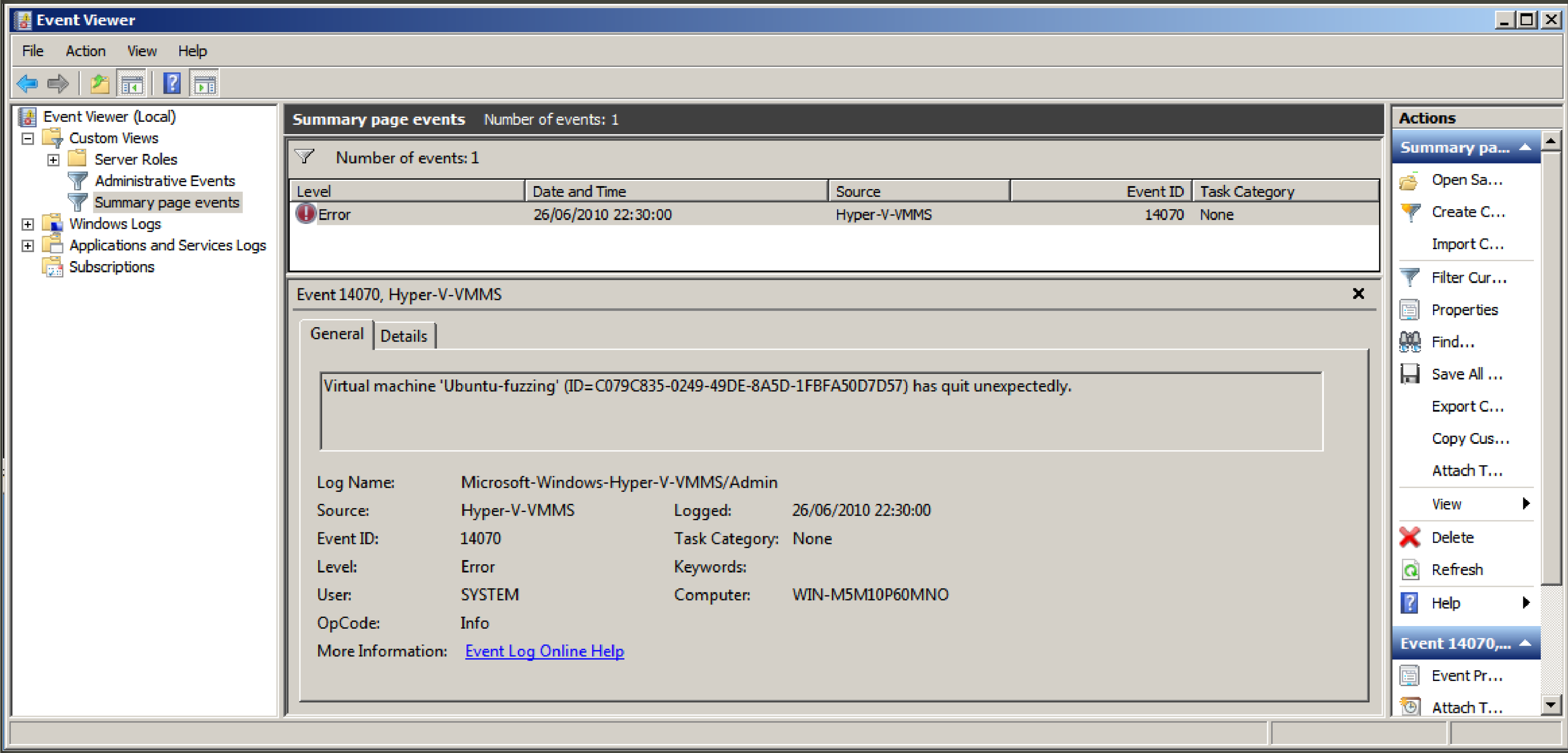
-------------- Guest processor state ---------------
Inhibit Mask=0

CS=FF63 [0000FFFF 0000F30F] V=1
SS=FFD3 [0000FFFF 00CF9300] V=1
DS=0018 [0000FFFF 00CFF300] V=1
ES=0018 [0000FFFF 00CFF300] V=1
FS=FF9B [0000FFFF 00CF9300] V=1
GS=0018 [0000FFFF 00CF9300] V=1

EAX=000000A9 EBX=00005148 ECX=0000F686 EDX=0000000B
ESI=00002D72 EDI=000007E4 EBP=00002E99 ESP=00000FFA
EIP=0000FE96 EFLAGS=00023202

What about x64 ?

Attacking Microsoft HyperV



DEMOS

DEMO

Adding layers of virtualization is actually a <u>bad idea</u> : the only way is to secure the software is to properly test it for security bugs…

Thank you for coming

Questions ?

HITBSECCONF2010

AMSTERDAM

29th June - 2nd July 2010

http://conference.hackinthebox.nl/

P1 Security

Priority One Security