Hypervisor Framework

Edgar Barbosa

H2HC 2008

São Paulo - Brasil

Who am I?

- *Edgar Barbosa*
- *Security Researcher working at* ***COSEINC****, a Singapore based IT company*
- *One of the developers of* ***BluePill****, a hardware virtualization based rootkit.*
- *Experience with kernel programming and reverse engineering.*

edgar@research.coseinc.com

Objectives

- *Explain virtualization theory and technologies.*
- *How Virtual Machine Monitors are implemented in the Intel architecture.*
- *Explain how VMMs uses the Intel Virtual Machine instructions.*
- *Teach how to use the Hypervisor Framework API to create VMMs.*

AGENDA

- *Virtual Machine Monitor*
- *The “efficiency principle”*
- *Creating a VMM with Intel VMX*
- *The Framework*
- *Security aspects*

Hypervisor Framework

VIRTUAL MACHINE MONITOR

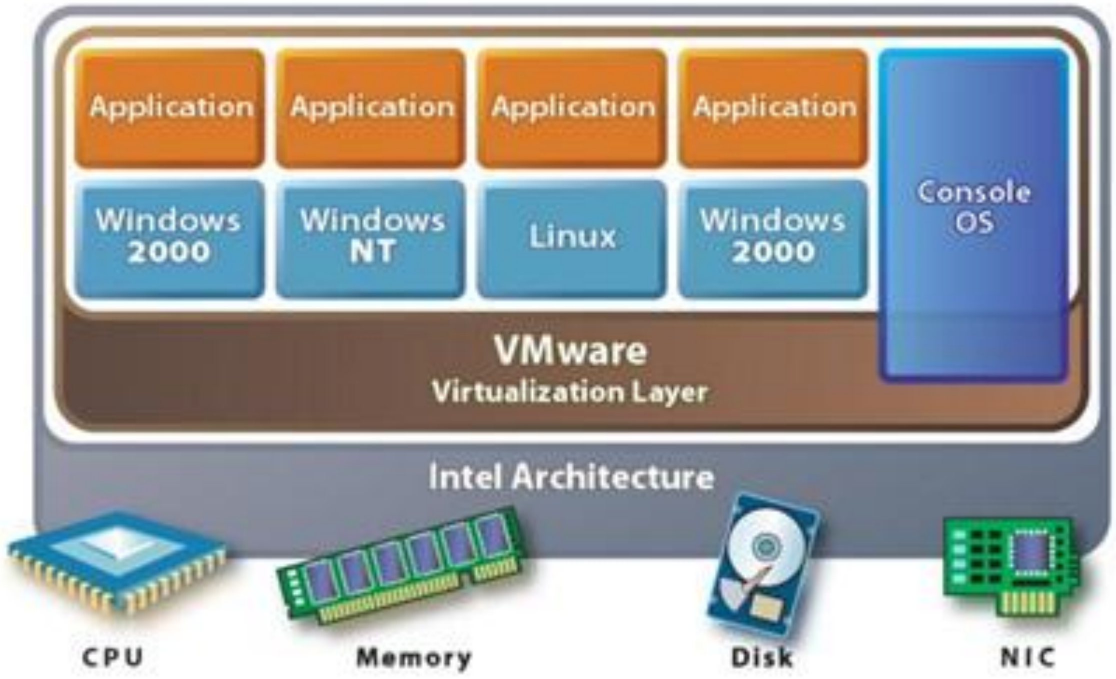
Virtual Machines

- *System Virtual Machines are virtual environments which duplicates a real machine.*
- *A Virtual Machine Monitor (VMM), also called Hypervisor, allows multiple operating systems to run concurrently on virtual machines.*

Types of VMM

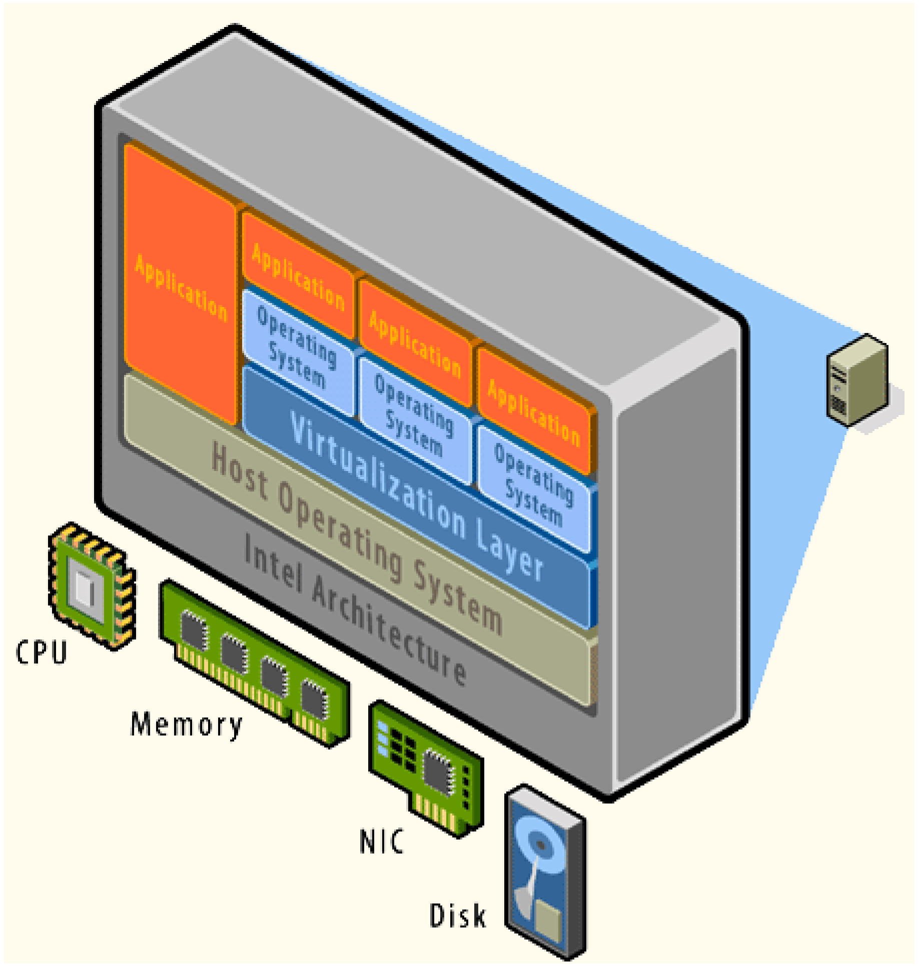
- *There are 2 types of VMM implementation.*
- *Type I*
  - *The VMM runs on a bare machine. It controls all the system resources. It`s basically an operating system with virtualization features.*
  - *Example: VMware ESX*
- *Type II*
  - *The VMM runs as an application. The Operating System controls the hardware resources*
  - *Example: VMware Workstation*

VMware ESX – Type I VMM



VMware ESX

VMware Workstation – Type II VMM



VMM requirements

- *According with Popek and Goldberg's paper, a VMM must satisfy 3 requirements:*

1. ***EQUIVALENCE***
2. ***RESOURCE CONTROL***
3. ***EFFICIENCY***

1 - EQUIVALENCE

- ***“Implies that any program executing on a virtual machine must behave in a manner identical to the way it would have behaved when running directly on the native hardware”*** *[1]*
- *This principle is not fully respected in modern VMMs. We will see more about this soon.*

2 - RESOURCE CONTROL

- ***“Implies that it should not be possible for guest software to directly change the configuration of any system resources”*** *[1]*
- ***Fundamental to guarantee the isolation of the guest virtual machines.***

3 - EFFICIENCY

- ***"Implies that all instructions that are innocuous must be executed natively on the hardware, with no intervention or emulation by the VMM"***

Innocuous Instructions

- *There are* ***innocuous*** *instructions and* ***sensitive*** *instructions.*
- *Innocuous instructions are instructions which doesn’t have access to system resources. Example:*

```
mov eax,ebx
shr eax,8
xchg ecx,edx
```

Sensitive Instructions

- Sensitive instructions are instructions which have access to system resources.
- Example:

```
mov cr3,eax
wrmsr
outb
popf
```

- RULE: The VMM must always **prevent** the direct execution of **sensitive** instructions!

Privileged instructions

- *Some sensitive instructions are privileged, which means that if not executed in the **most privileged level**, it triggers a general protection exception.*
- *Example:*
  The Write MSR instruction (wrmsr)
- *If some code try to execute **wrmsr** with CPL=3, then an exception is triggered.*

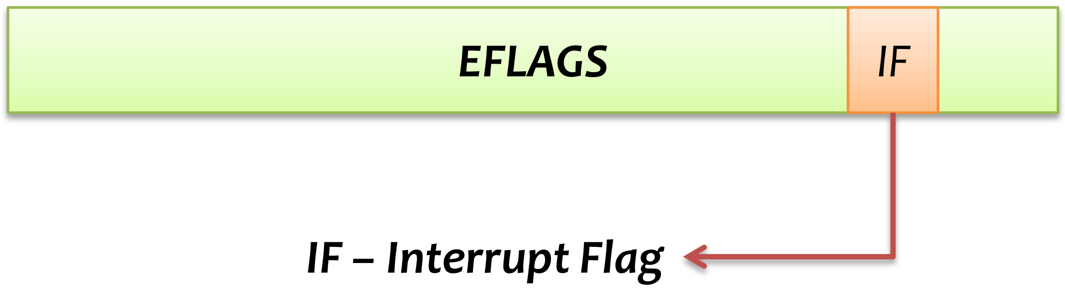
Privileged instructions

- *If all sensitive instructions of some ISA are privileged, the processor is considered to be ‘virtualizable’[3]*
- *This would allow a guest Linux kernel for example to run directly in the processor with CPL=3. When the kernel tries to execute a sensitive instruction, an exception will be intercepted by the VMM and then it will emulate the sensitive instruction.*

The POPF instruction

- *The POPF instruction is an example of a sensitive instruction which is non-privileged.*
- *POPF can write in the EFLAGS registers, which contains a lot of system flags.*
- *If executed with CPL=0, POPF is able to change any system flag.*
- *If executed with CPL=3, the CPU simply* ***ignores*** *the modification on the system flags, instead of generating an exception.*

POPF and the Interrupt Flag



- *The IF flag inside EFLAGS controls the system interrupts. If set, the system accepts interrupts, else it ignores the interrupt signal.*

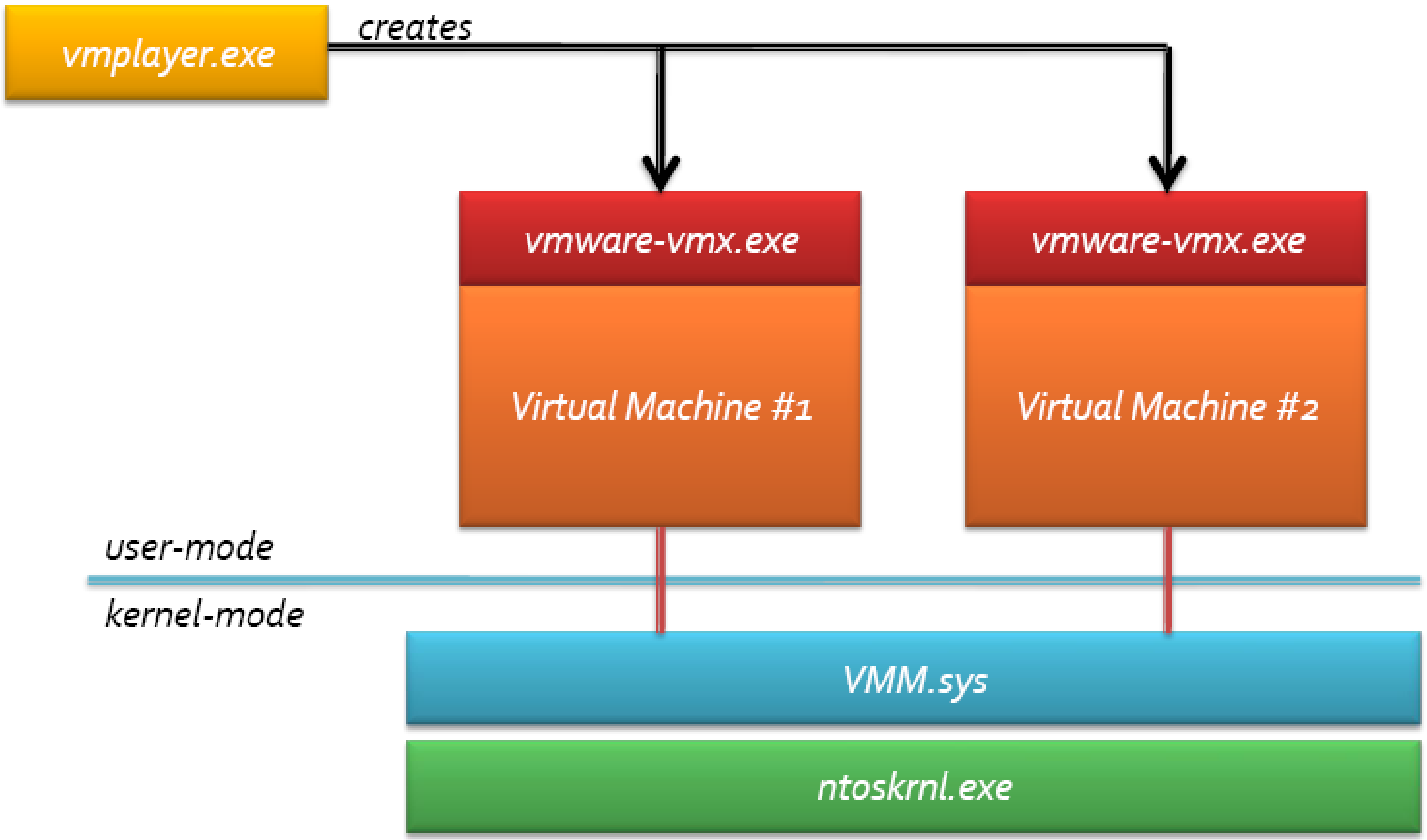
Kernel and POPF

- *What happens if a guest VM kernel tries to execute POPF to clear the IF flag running with CPL=3?*
- *How the VMM can intercept this instruction if POPF is non-privileged?*

VMware Player case

- *How VMware Player VMM is able to prevent direct execution of sensitive instructions?*
- *VMware Player is a* ***Type II VMM*** *implemented as a PE resource inside the binary* ***vmware-vmx.exe*** *executable file.*
- *The resource file is a ELF executable which is loaded directly inside the Windows kernel memory by the vmx86.sys device driver.*

VMware Player architecture



VMware code scanning

1. *The VMware VMM scans the instruction stream being executed in the VM and detects the presence of sensitive and non-privileged instructions. Example: the POPF instruction.*
2. *The VMM then substitutes the sensitive instruction with a privileged instruction and then emulates the action of the original instruction.*

Hypervisor Framework

CREATING A VMM USING INTEL® VMX INSTRUCTIONS

Hardware assisted virtualization

- *Any VMM must prevent the direct execution of sensitive instructions. However, we know there are sensitive and non-privileged instructions.*
- *Intel and AMD created new instructions to help the creation of VMMs. With these new instruction sets, it is possible to change the behaviour of all sensitive instructions to execute as* ***privileged*** *instructions.*

Virtualization technologies

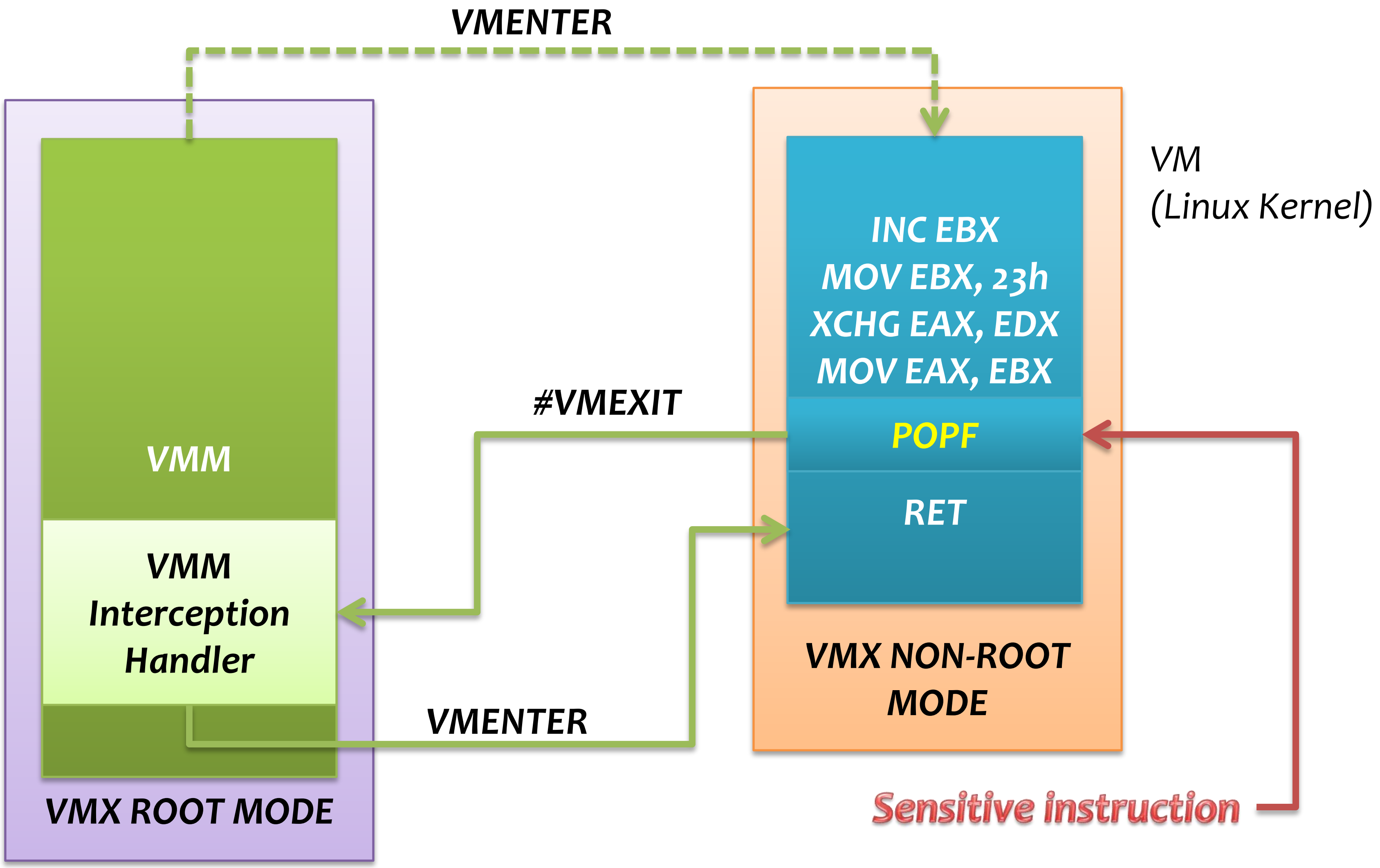
- *AMD created the Secure Virtual Machine (SVM) instruction set.*
- *Intel created the VMX (Virtual Machine eXtensions) instruction set.*
- *This presentation focus on the VMX, but the concepts are very similar in the AMD architecture.*

Intel VMX

- *The main feature of the Intel VMX is a new system operating mode: the VMX mode.*
- *Two sub-modes:*
  - *VMX root mode*
  - *VMX non-root mode*
- *The VMX root mode is for the hypervisor. There is no instruction behaviour modification in this mode.*

VMX

- *When the hypervisor (VMM) transfers control to the VM, it changes the mode to VMX non-root.*
- *In VMX non-root mode, the behaviour of some instructions are changed. It is possible for example to make the POPF instruction to trigger an exception and wake up again the hypervisor.*
- *This transition from the VM to the VMM is called **VMEXIT**.*


VMENTER
VM
(Linux Kernel)
INC EBX
MOV EBX, 23h
XCHG EAX, EDX
MOV EAX, EBX
#VMEXIT
POPF
VMM
RET
VMM
Interception
Handler
VMENTER
VMX NON-ROOT
MODE
VMX ROOT MODE
Sensitive instruction

Steps to create a VMM

- *Fully documented in the Intel manuals.*
- *Detection of VMX instructions support*
  - *Using CPUID instruction*
- *Check MSR lock flag.*
  - *If locked by the BIOS, virtualization will not work!*
- *Enabling VMX*
- *Creation and initialization of the VMCS*
- *The interception handler*

VMCS

- *Virtual Machine Control Structure*
- *Data structure used by the CPU to store all information about the Virtual Machine and the VMM.*
- *Divided in 5 areas:*
  - *Guest-state area*
  - *Host-state area*
  - *VM-execution control fields*
  - *VM-entry control fields*
  - *Vm-exit information fields.*
- *One for each VM and for each CPU*
- *Must not be directly accessed*
  - *Use the VMX instructions to access the VMCS.*

Interception

- *The virtual machine must handle all the interception events (VMEXIT).*
- *Types of events*
  - *CRx/DRx register access*
  - *Interrupts*
  - *I/O instructions*
  - *TSC and MSR registers*
  - *much more….*

Interception event handler

- *The instructions which the hypervisor wants the CPU to intercept must be configured in the VMCS.*
- *The VMCS also must be configured to contain the address of the VMM routine that will handle the interception event (VMEXIT info).*

Hypervisor Framework

THE FRAMEWORK

The framework

- The Hypervisor Framework enables you to easily create a Hosted Virtual Machine Monitor (Type II VMM) using the Windows Operating System.
- The framework exports a simple API which creates an abstraction layer over the different CPU virtualization instruction sets.

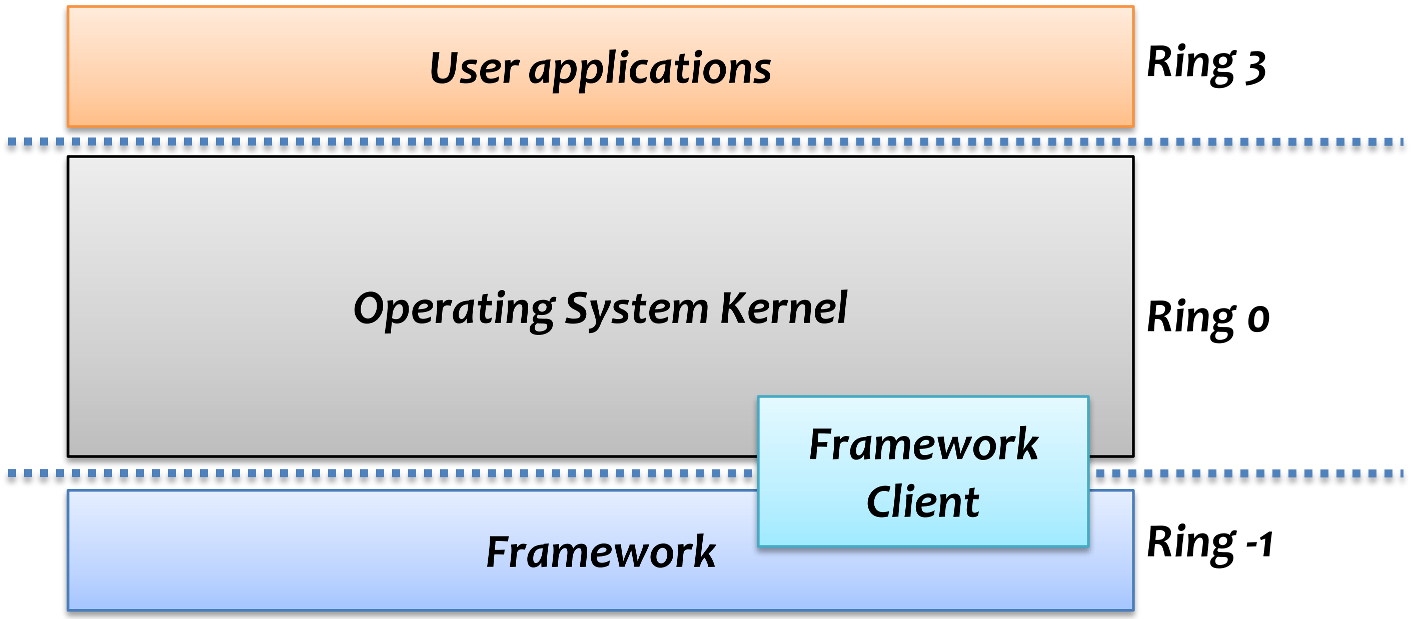
The framework

- *Initially implemented as a Windows Device Driver*
- *OS-dependent functions are implemented in a separate module. Porting to Linux/Mac must be very easy.*
- *2 versions:*
  - *32-bits and 64-bits*
- *First version uses only Intel VT*
  - *AMD version coming soon*
- *Device Drivers can export functions like user-mode DLLs.*
  - *This is the way the driver uses to export the framework functions.*

The framework

- *Initially only binary versions will be released.*
- *The source release is being evaluated.*
- *If released, you will find it at http://code.google.com/p/hypervisor/*
- *Check the COSEINC blog (will open soon).*
- *Full documentation with examples will be released soon.*

Architecture



Features

- *Framework features*
  - *Detection of Intel VMX instructions*
  - *Detection of Intel virtualization status*
  - *Initializes the VMXON area*
  - *Creates and initialize a VMCS area for each Virtual Machine and for each CPU.*
  - *Intercept events and call the clients callback functions.*

API

- *Function categories:*
  - *Virtual Machine management functions*
    - *Creation and deletion of Virtual Machines.*
    - *Executing and resuming a virtual machine.*
  - *Interception Events functions*
    - *The framework call the registered client function callbacks.*

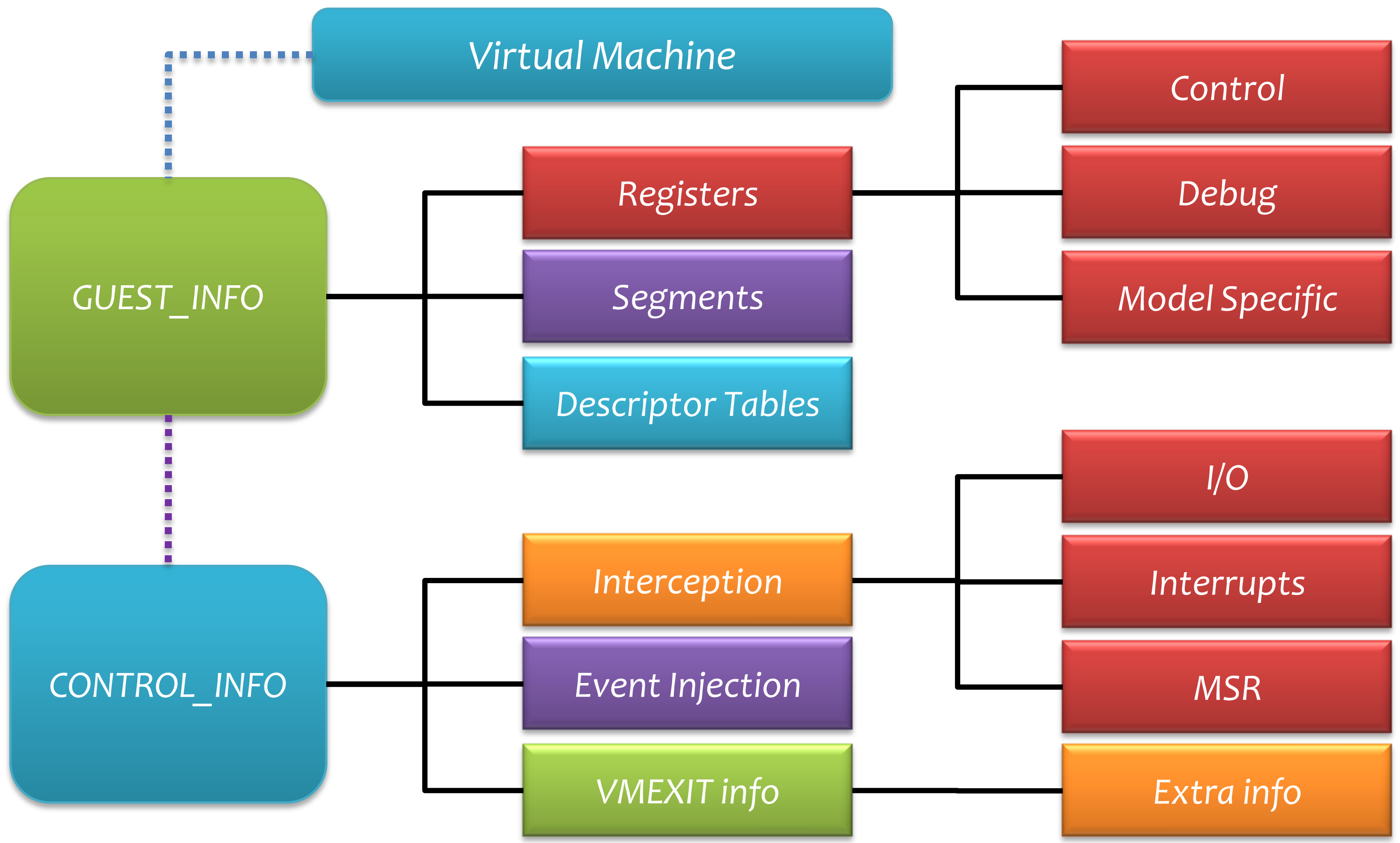
Virtual Machine management

-
  ```
  VMSTATUS
  CreateVirtualMachine (
      IN VMINFO *vminfo
      );
  ```
- *This function creates a new virtual machine in the system.*
- *Fails if virtualization MSR is locked by the BIOS.*

VMINFO data structure

- *The VMINFO structure contains:*
  - *all the GUEST context information*
  - *GDT, LDT, Page Tables, Control Registers, …*
  - *Interception handler function callback address.*
  - *Contains Event Injection information*
  - *VMEXIT information*

VMINFO data structure

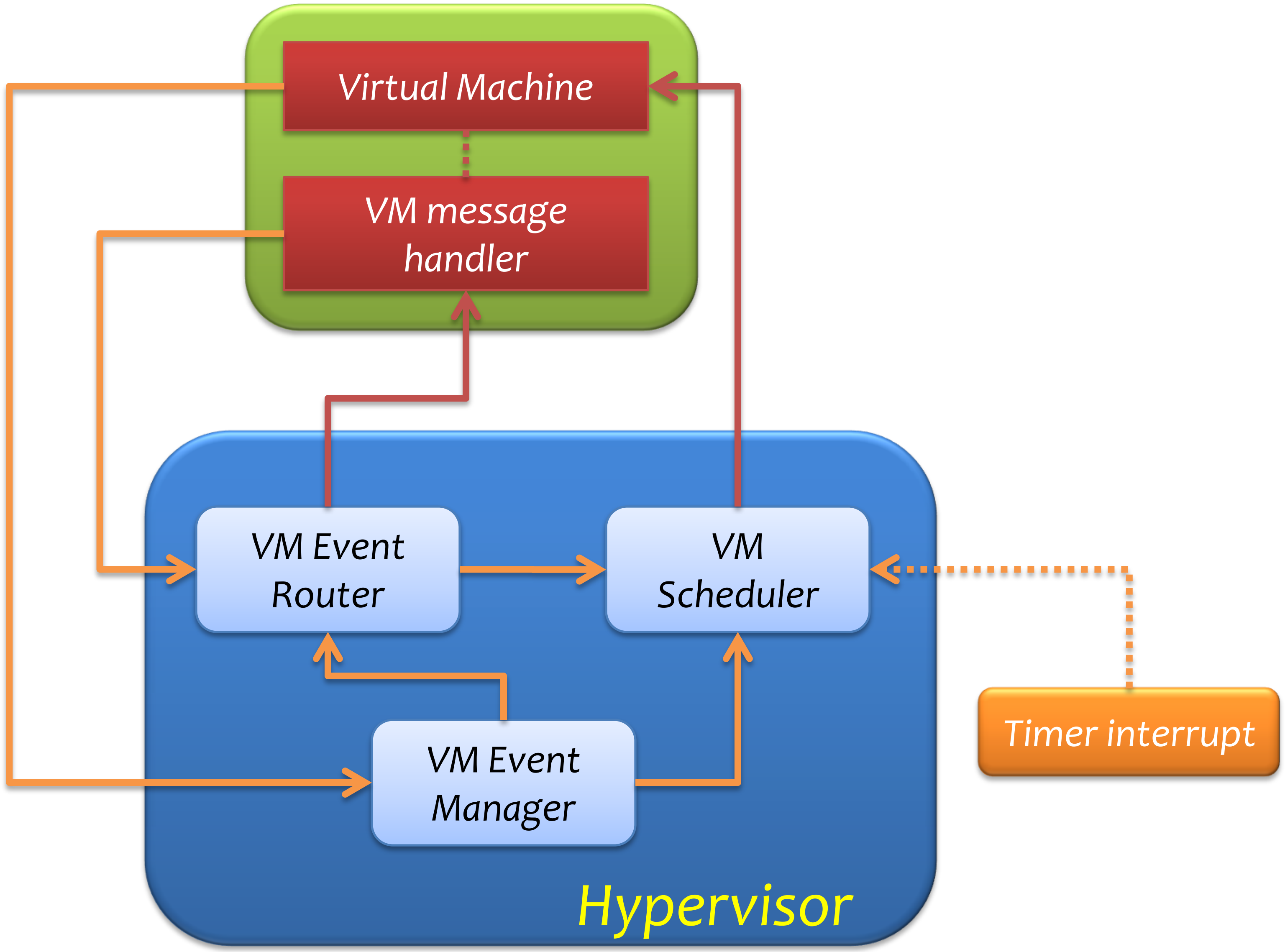


Interception Event management

-
  ```
  VMSTATUS
  VirtualMachineExec (
      IN VMINFO *vminfo
      );
  ```

- *This function controls the execution of the virtual machine. It can be called after the creation of the VM and to resume the execution of the VM after an intercept event.*
- *If the VMM must inject some event in the guest VM, the information is provided in the VMINFO data structure.*

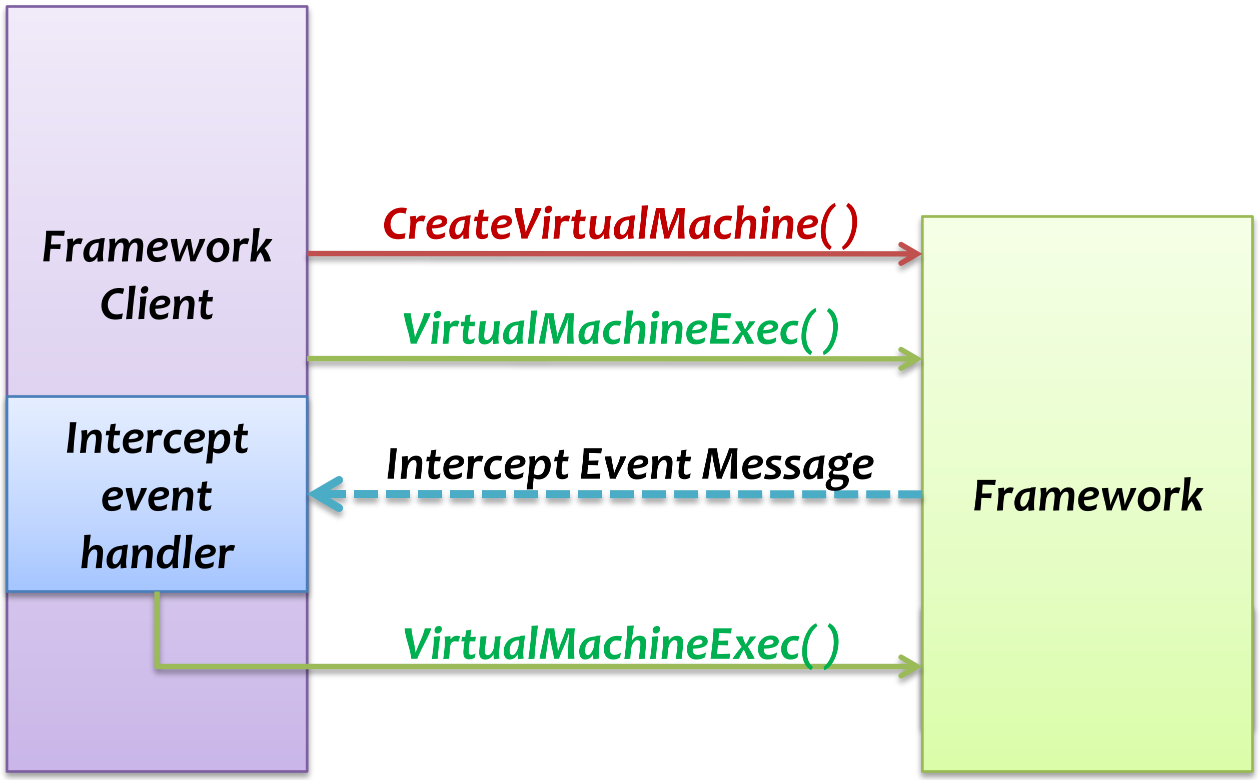
Framework – Client communication



Basic steps

- *Allocate a VMINFO data structure.*
- *Initialize it with the guest context information*
- *Setup the instructions and operations that you want the hypervisor to intercept*
- *Set your interception handler function callback address*
- *Execute your VM.*

Framework


Framework Client
Intercept event handler
CreateVirtualMachine( )
VirtualMachineExec( )
Intercept Event Message
VirtualMachineExec( )
Framework

Additional info

- *You can make the host OS to become a guest VM.*
- *The framework is still a work in progress.*
- *More functions will be added.*
  - *Example: memory protection, Intel VT-d and EPT.*
- *Your feedback is fundamental.*
- *Lots of applications for the framework.*

Hypervisor Framework

SECURITY

Security

- *Modern BIOS which supports hardware virtualization comes with virtualization* ***disabled*** *by default.*
- *MSR protection.*
- *64-bit versions of Windows Vista and XP requires digitally signed device drivers.*

Detection

- *Popek and Goldberg principle of equivalence is not fully respected in the Intel and AMD virtualization instruction sets.*
- *See more information in the "Detecting Virtualized Hardware Rootkits" presentation at the COSEINC website. http://www.coseinc.com/publication.html*

We are hiring

- *Interested in Hypervisor hacking?*
- *Interested to work with kernel and system programming?*

- ***WE ARE HIRING!*** ☺

- *Contact us!*
- *http://www.coseinc.com/recruitment.html*

Questions?

References

- [1] *James E. Smith. Virtual Machines – Versatile Platforms for System and Processes. Morgan Kaufmann.*
- [2] *Intel manuals. http://www.intel.com*
- [3] *J. S. Robin. Analyzing the Intel Pentium's Capability to Support a Secure Virtual Machine Monitor. Master's thesis, Naval Postgraduate School, Monterey, CA, September ,1999.*

THANK YOU FOR YOUR TIME!

edgar@research.coseinc.com