MOCA 2012

Beware of hypervisor: understanding ring -1

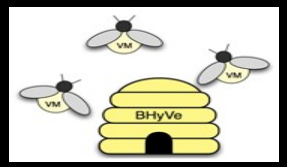
whoami

- Mariano `emdel` Graziano (@emd3l).
- PhD student at EURECOM, Sophia-Antipolis, France.
- Politecnico di Torino, Computer and Communication networks.
- IRC (Azzurra, Efnet, Freenode).
- Ex Playhack Security
- Current interest: Virtualization.

Intro

- Virtualization is everywhere.





- Cloud services (Google, Apple, Amazon, Microsoft etc etc).
- We are security guys @ MOCA 2012, why do we use virtualization?
  - Analyze malware? (Sandboxes)
  - Test our exploits? (Lab)
  - Create new protection systems?
  - Create malware based on virtualization? (Bluepill, Vitriol…).

Virtualization 101

- Virtualization concept is already applied to modern OS:
  - Scheduling time-sharing technique (CPU Virtualization).
  - Virtual memory layout (Memory Virtualization).

- Virtualization of entire Architecture:
  - CPU virtualization.
  - I/O virtualization (Memory and Devices).

Virtualization 101

- Virtual Machine Monitor (VMM aka Hypervisor) definition:
  - Provides an environment which is essentially idential with the original machine.
  - Programs in this environment show at worst only minor decreases in speed.
  - VMM is in complete control of system resources.
- Virtual Machine (VM):
  - It is the environment created by the virtual machine monitor.

Virtualization 101

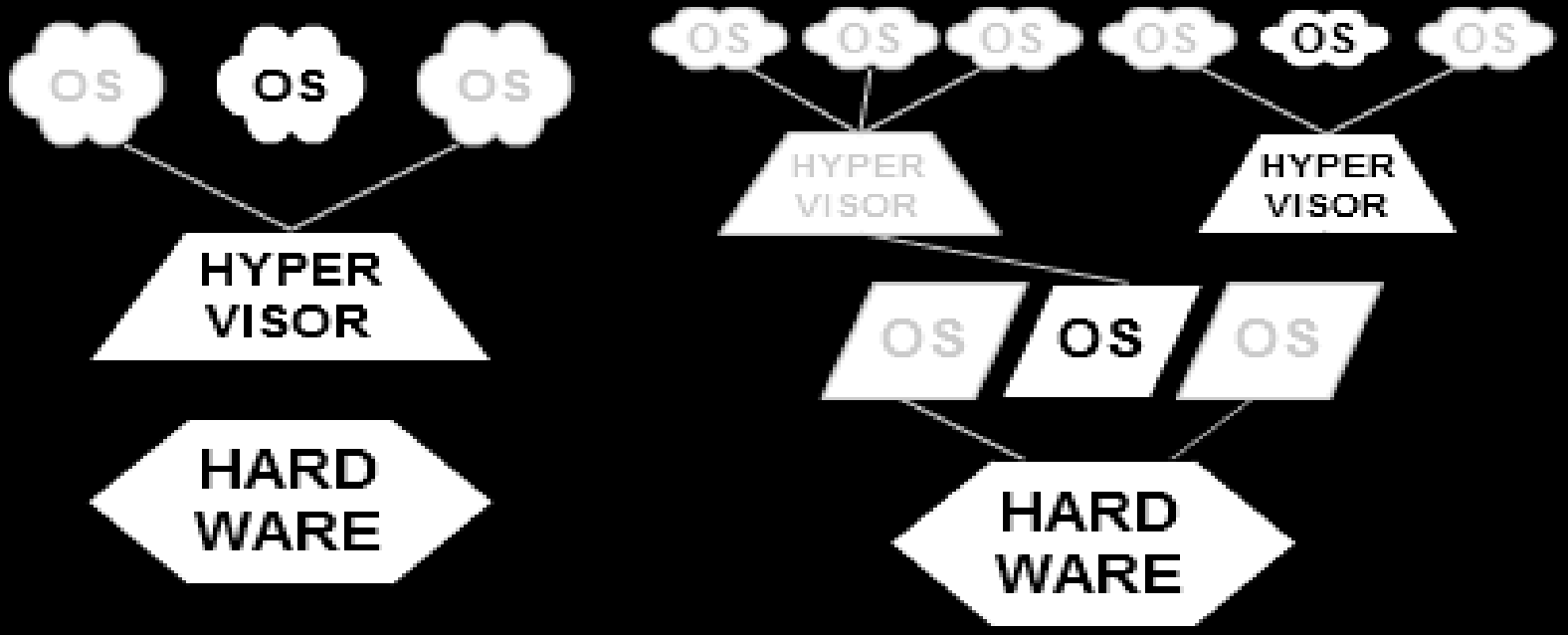
- Virtual Machine Monitor properties:
  - Dispatcher (hypervisor starting point to decide which module to call for the given trap).
  - Allocator (It has to decide what system resources are to be provided)
  - Interpreter (It needs one interpreter routine per privileged instruction, each routine has to simulate the effect of the instruction which trapped).

Virtualization 101

- Virtual Machine properties:
  - Efficiency (All innocuous instructions are executed by the hardware directly).
  - Resource control (Must be impossible for a program running in a VM to affect the system resources).
  - Equivalence (A program running in a VM performs in a manner indistinguishable from the case the program runs in the host).
- VMM is any control program that satisfies the three properties of efficency, resource control and equivalence.
- VM is the environment which any program sees when running with a VMM present (Real machine + VMM).

Virtualization 101

- VMM typology:
  - Type I
  - Type II



*Picture from Wikipedia.

Virtualization 101

- Popek and Goldberg put forward a set of requirements that must be met in their 1974 paper “Formal Requirements for Virtualizability Third Generation Architectures.”
- They divided instructions in 3 categories:
  - Privileged Instructions: are defined as those that may execute in a privileged mode, but will trap if executed outside this mode.
  - Control sensitive instructions: are those that attempt to change the configuration of resources in the system (physical memory assigned to a program or the mode of the system).
  - Behavior sensitive instructions: are those that behave in a different way depending on the configuration of resources, including all load and store operations that act on virtual memory. (mode of the system)

Virtualization 101

- In order for an architecture to be virtualizable, Popek and Goldberg determined that all sensitive instructions must also be privileged instructions.

- This means that a hypervisor must be able to intercept any instructions that change the state of the machine in a way that impacts other processes.

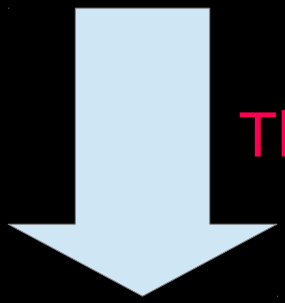
Virtualization 101

- In order to be virtualizable the set of control of sensitive instructions must be a subset of privileged instructions. (Any instructions that modifies the configuration of resources in the system must either be executed in privileged mode or trap if it isn't).
- There is a set of 17 instructions in the x86 instruction set that does not have this property.
- For example, the LAR and LSL instructions load information about a specified segment. Because these cannot be trapped, there is no way for the hypervisor to rearrange the memory layout without a guest OS finding out.

Virtualization 101

- In order to overcome the issue of x86 architecture we can use 3 possible solutions:
  - Binary Rewriting.
  - Paravirtualization.

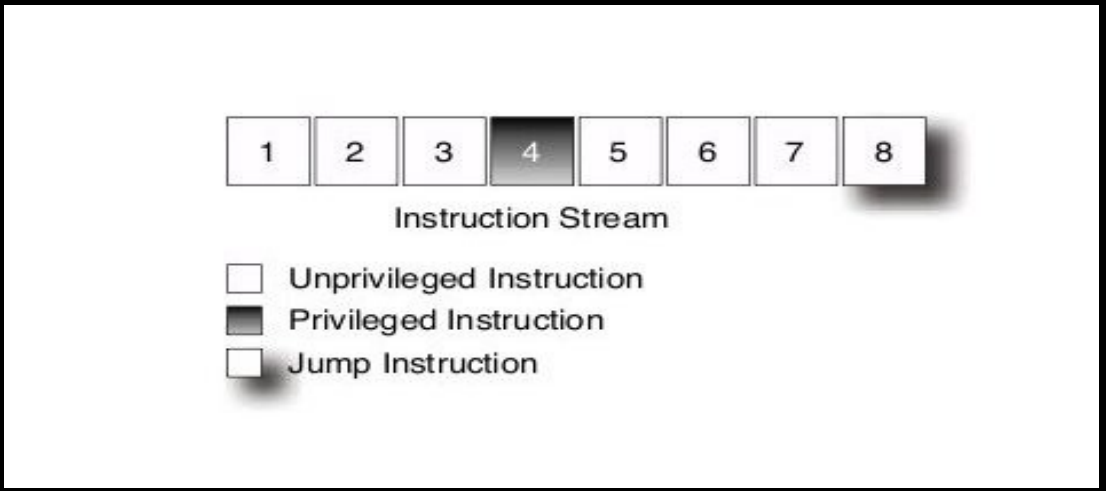
  } Due to lack of hardware support



The normal evolution

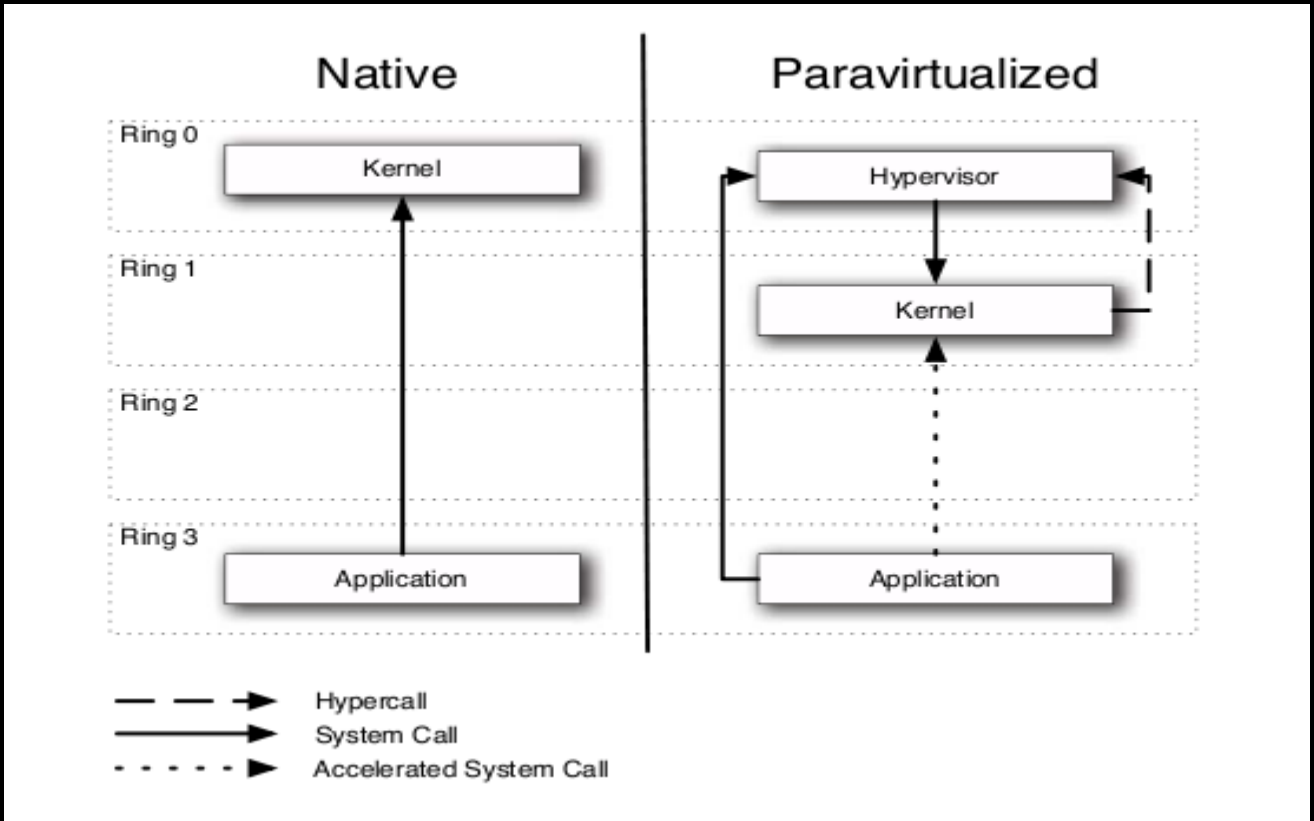
  - Hardware-Assisted Virtualization.
- In the next slides I will give you the main idea of Binary rewriting and Paravirtualization and then we will focus out attention on Hardware Assisted Virtualization.

Binary Rewriting



- The binary rewriting approach requires that the instruction stream be scanned by the virtualization environment and privileged instructions identified.
- It inserts breakpoints on any jump and on any unsafe instruction.
- When it gets to a jump, the instruction stream reader needs to quickly scan the next part for unsafe instructions and mark them.
- When it reaches an unsafe instruction, it has to emulate it.
- Performance problem.

Paravirtualization

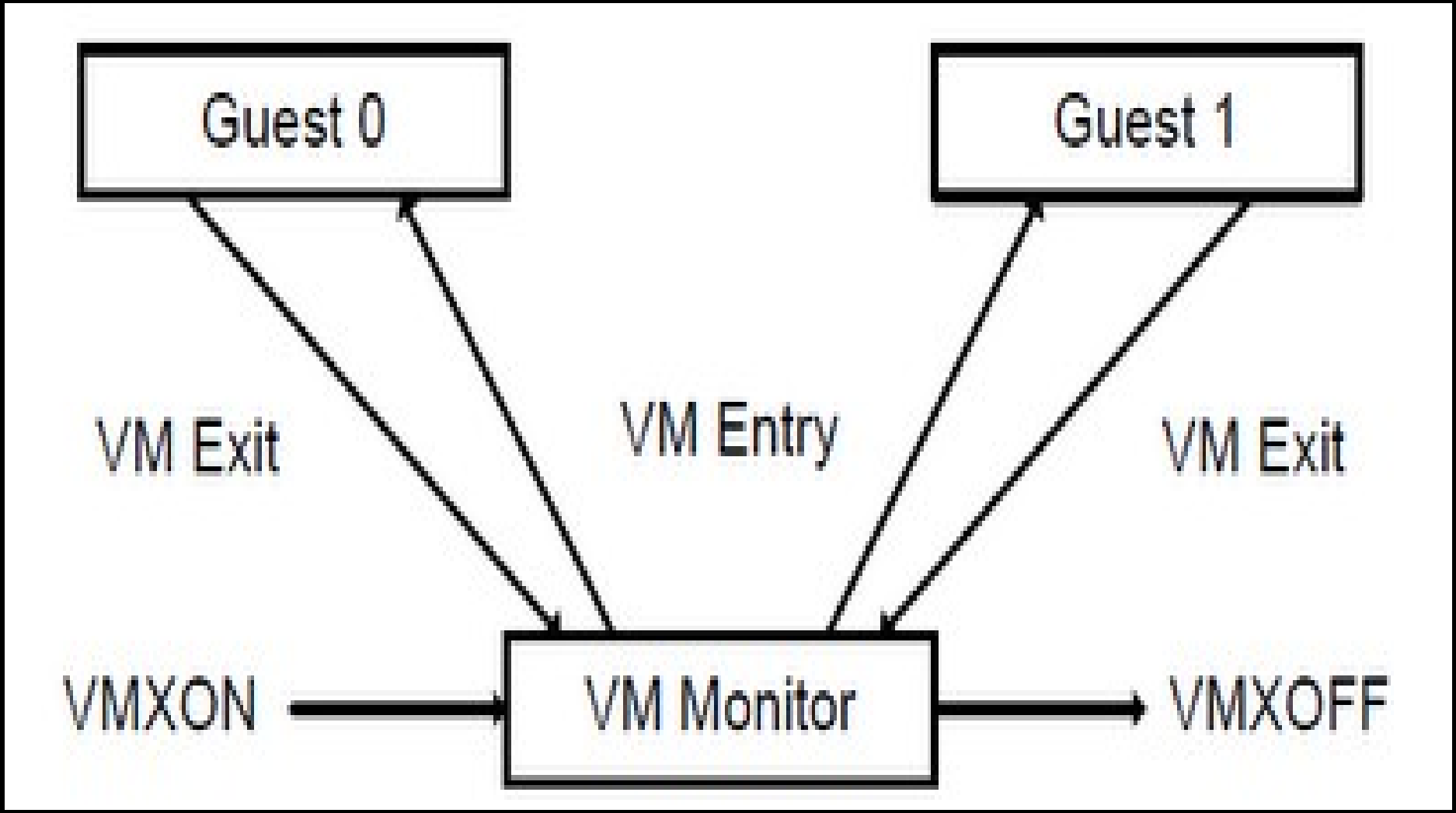


- Conceptually, this is similar to the binary rewriting approach, except that the rewriting happens at compile time (or design time), rather than at runtime.
- In order to simulate a privileged instruction the hypervisor exposes a set of hypercalls (push the values and then raise an interrupt).

Hardware assisted Virtualization

- Intel gives processor level support for virtual machines (support for VMM & VM).
- Processor support by a form of processor operation called Virtual Machine eXtension (VMX) – on AMD we have SVM (in this presentation we will focus on Intel).
- 2 kinds of VMX operations:
  - VMX root operation (VMM)
  - VMX non root operation (VM)
- VMX transitions = transitions between VMX root & VMX non root.
- 2 Kinds of VMX transitions:
  - VM ENTRIES (transitions into VMX non root).
  - VM EXITS (transitions from VMX non root to VMX root).

Hardware assisted Virtualization



*Picture from Intel manuals found on the net.

Hardware assisted Virtualization

- Processor behavior in VMX root operations is like outside VMX but there are a set of new instructions (VMX instructions)
- New instructions:
  - VMXON (enter vmx operation)
  - VMXOFF (leave vmx operation)
  - VMREAD (read from the VMCS)
  - VMWRITE (write to the VMCS)
  - VMPTRLD (load VMCS pointer)
  - VMPTRST (store vmcs pointer)
  - VMLAUNCH/VMRESUME (launch or resume the VM)
  - VMCALL (call to the hypervisor)
- Processor behavior in VMX non root is restricted and modified to facilititate the virtualization.
- It is this limitation that allow the VMM to retain control of processor resources.

Life Cycle

- VMXON to enter VMX operations
- Using VM ENTRIES a VMM can enter guests into VM. VM ENTRY == VMLAUNCH & VMRESUME instructions.
- VM EXITS transfer control to a known entry point. VMM will take the appropriate action.
- The VMM can decide to shut itself down and leave the VMX operation (VMXOFF).

VMCS

- Virtual Machine Control Structure.
- VMX non root operation and VMX transitions are controlled by VMCS.
- VMCS can be accessed through the VMCS pointer (One per logical processor).
- VMCS pointer is read and write using the instructions VMPTRST & VMPTRLD.
- VMM configure the VMCS using VMREAD, VMWRITE & VMCLEAR instructions.
- VMM could use a different VMCS for each VM. For a VM with multiple logical processor VMM can use a VMCS for each virtual processor.

VMCS

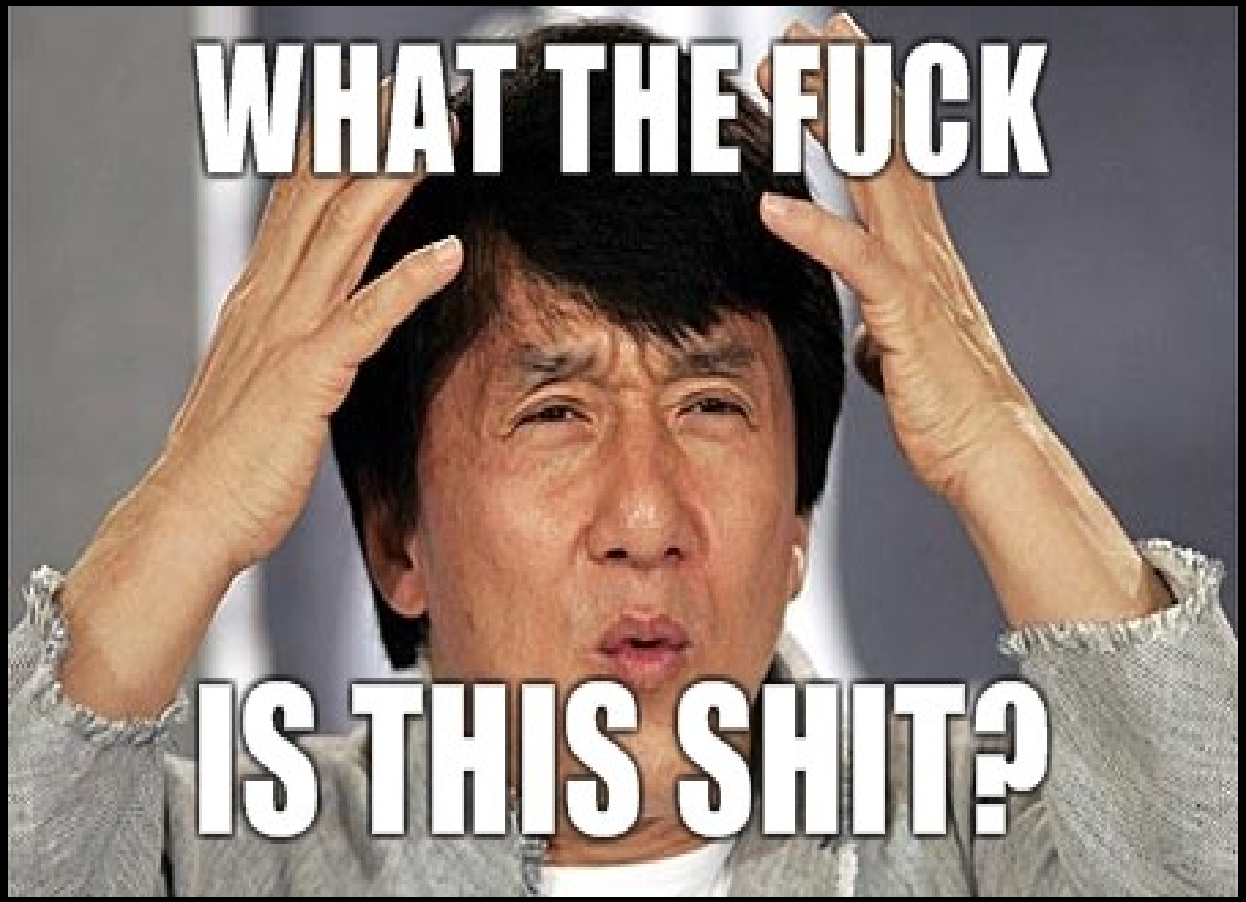
- VMCS manages the transitions in and out of VMX non root operation (VM entry & VM exit).
- VMCS manages the processor behavior in VMX non root operation.
- VMM can use different VMCS for a single VM and can use different VMCS for different virtual processors.
- At any given time only one is the current VMCS.
- A VMCS is active by executing VMPTRLD with the address of that VMCS. Inactive by calling VMCLEAR with the VMCS address.
- A VMCS remains current until the execution of another VMPTRLD with an address of a different VMCS.

VMCS

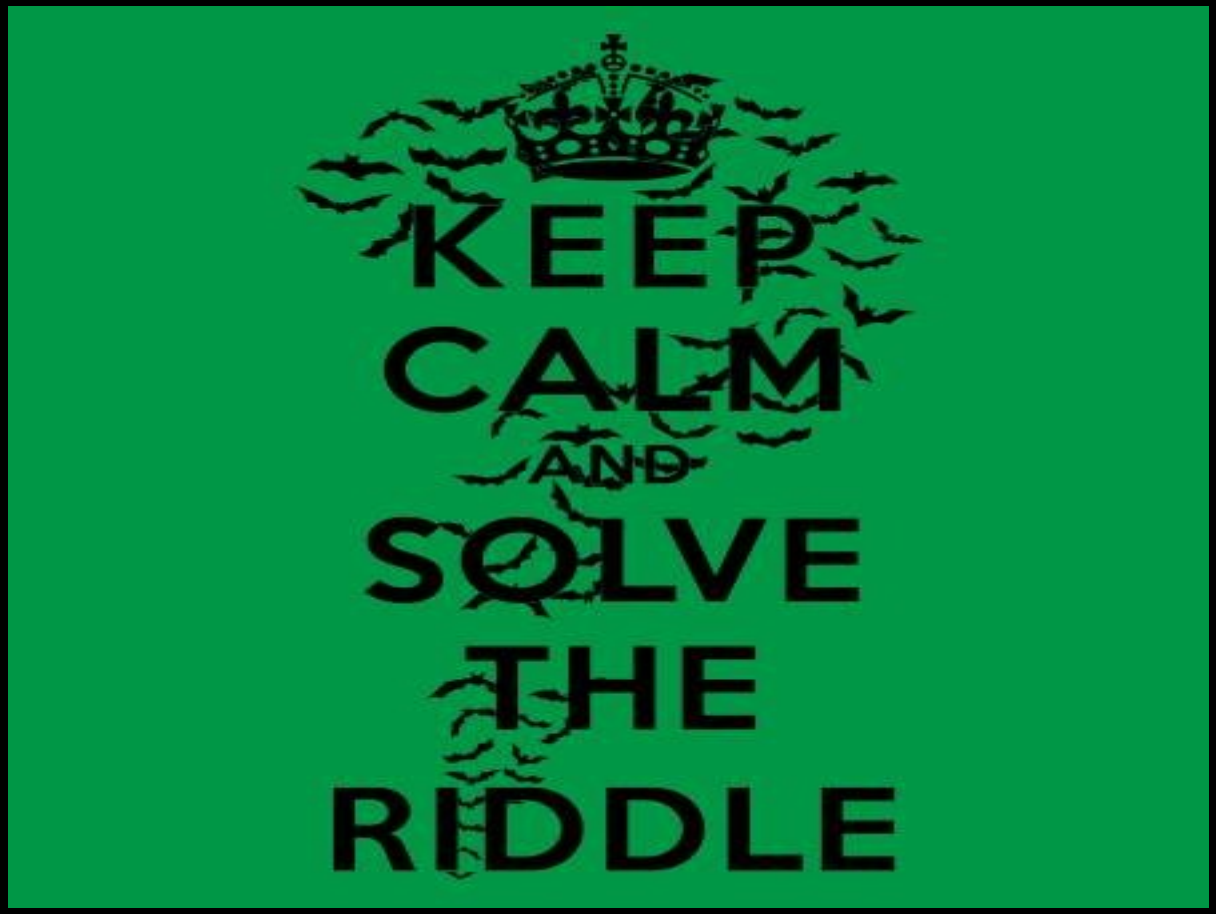
- VMCS region: region in memory with a VMCS associated to a logical processor.
- You can access the VMCS by using its physical address (VMCS pointer).
- VMCS pointer 64 bit address and for architectures that do not support Intel 64 the 63:32 bits are not set.
- Every field in VMCS associated with a 32 bit value (its encoding).
- Encoding is provided to VMREAD/VMWRITE when it is necessary to read/write that field.
- Software should never access or modify the VMCS data using ordinary memory operations.
- Format to store VMCS data is implementation specific and not architecturally defined and in addition some VMCS data can be maintained on the processor and not in the VMCS region.

WTF

IMPLEMENTATION SPECIFIC? WTF?



WTF



Let's dig a bit...

VMCS

- Format of VMCS region:

BYTE OFFSET	CONTENTS
0	VMCS REVISION ID
4	VMX ABORT INDICATOR
8	VMCS DATA

VMCS

- VMCS Revision ID: first 32 bit of the VMCS region.
- Processor that mantains the VMCS data in different formats use different VMCS revision ID.
- Must be set before the VMCS region is used.
- VMPTRLD fails if you use a revision id different from the one used in the processor (MSR_IA32_VMX_BASIC).

VMCS

- VMX_ABORT_INDICATOR: A logical processor writes a non zero value into those bits If a VMX abort occurs.
- VMCS DATA: it's the part that control the VMX non root operation and VMX transitions. The format is implementation specific.

VMCS Data Organization

- VMCS is organized into 6 logical groups:
  - Guest State Area: processor state saved into guest area on VMCS exits and loaded from there on VMCS entries.
  - Host State Area: processor state loaded from host state area on VM exits.
  - VM Execution Control Fields: these fields control the behavior in VMX non root operation. They determine in part the cause of VM exits.
  - VM EXIT Control Fields: these fields control VM exits.
  - VM ENTRY Control Fields: these fields control VM entries.
  - VM EXIT Info Fields: These fields describe the cause and nature of the VM Exits. They are read only.

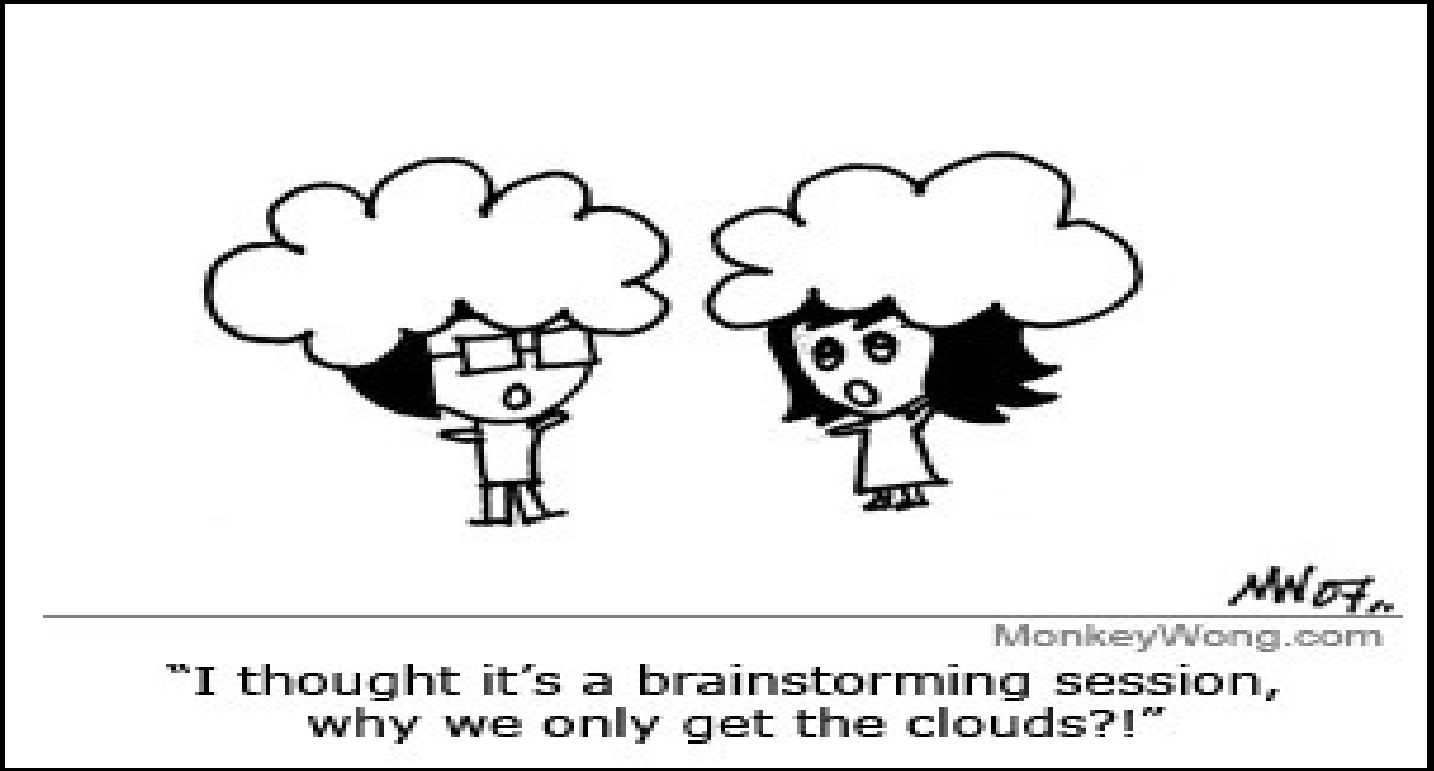
VMCS

- For the sake of brevity I will not list the fields of each group.
- Keep in mind that VMX non root operation can be controlled by data structures that are referenced by pointers in a VMCS (e.g I/O Bitmap, APIC access page etc).
- As already said (do you remember? Wake up :)) encoding is provided to VMREAD/VMWRITE when it is necessary to read/write that field.
- Again the format to store VMCS data is implementation specific and not architecturally defined.
- But we want to know... “Our crime is that of curiosity” (cit.)

Challenge

- We want to know the memory layout of the VMCS.
- We have to keep into account that the offsets of each field will change for the different Intel Microarchitectures.
- But how? We are at ring -1...
- From the OS the memory in which the VMCS is allocated is not visible.
- How?

Brainstorming



We have different approaches:

- The long one: fill the VMCS region with known values and dump the physical memory via Firewire for example.
- The short one: Modify an existing/open source hypervisor.

Approach 1

- For the first approach I suggest you to use Inception is a FireWire physical memory manipulation and hacking tool exploiting IEEE 1394 SBP-2 DMA.
- The tool can unlock (any password accepted) and escalate privileges to Administrator/root on almost any machine you have physical access to.
- The tool now has full read/write access to the lower 4GB of RAM on the victim.
- Thank you Carsten Maartmann-Moe for the tool!

Approach 2

- We have to modify an exisisting hypervisors. Two choices:
  - KVM
  - HyperDBG

  } The changes will be in the vmx.c file.
- General idea:
  - Put in every entry of the VMCS region the number of the entry.
  - Perform the VMREAD for each encoding.
  - The output will be the number of the entry.
  - Print it (you will read it from the debug messages).

Quest

- Remember our goal is to know the memory layout of the VMCS.

- Ok, let's do it in few steps:
  - Test if your system support hardware assisted virtualization (cat /proc/cpuinfo has to contain vmx)
  - Download an opensource hypervisor (KVM or HyperDBG).
  - Modify few lines in the vmx.c file :)

Quest

We Manually write the fields of the VMCS without using the VMREAD/VMWRITE.

- The code is self-explanatory (code for HyperDBG):

```
for(i = 0; i < 1024; i += 1)
    *(vmxInitState.pVMCSRegion + i) =  i;
v = VmxVmcsRead(0x00002004);
GuestLog("ADDRESS_MSR_BITMAPS entry #: %08x" , v);
```

- We will perform a VMREAD for each field, in this way we will obtain the number of the entry.

Quest

- We compile again the driver and we launch again the hypervisor (for sure it will crash but on the debug messages we can see the results and we have to parse it!).
- Now, if we want, we can check if the results are reasonable with the dump of the physial memory...
- Print the physical address of the VMCS and then the revision id for example and check if they match.

VMCS memory layout

- Number of entry for the Intel Core Microarchitecture.
- ADDRESS_MSR_BITMAPS_HIGH : 00000035
- VM_EXIT_MSR_STORE_ADDR_HIGH : 00000037
- VIRTUAL_APIC_PAGE_ADDR : 0000002e
- ….....etc etc.
- The full list is available on my site

Holy Grail

- To have the absolute offset we need just few bash lines:
- pre="0x";for hexnum in `cat offset.log | cut -d ":" -f2`; do val10=`printf "%d\n" $pre$hexnum`; dec=`echo "$val10*4" | bc`;printf "%x\n" $dec;done > mseek_offset.log
- One line of Bash... I know I'm lazy :)
- Until now only the offsets for the Intel Core Microarchitecture are available.

Call for partecipation

- Download the vmx.c file from my site
- Download HyperDBG
- Compile using HyperDBG with the given vmx.c file
- Send me your output from 'dmesg'
- Thank you!
- I will release soon the vmx.c file for KVM

- Known limitation HyperDBG works only on IA32 systems with no PAE.

Links

- Intel 64 and IA-32 Architectures Software Developer’s Manual Volume 3B: System Programming Guide, Part 2
- Gerald J. Popek and Robert P. Goldberg (1974). "Formal Requirements for Virtualizable Third Generation Architectures". Communications of the ACM 17 (7): 412 –421.
- Http://www.ivanlef0u.tuxfamily.org/?p=120#more-120 (Abyss part 1, 2,3)
- http://deroko.phearless.org/cpuid_break.rar
- http://www.breaknenter.org/projects/inception/
- http://www.linux-kvm.org/page/Main_Page (IRC chan too)
- http://code.google.com/p/hyperdbg/

Thanks

- Andrea Lanzi for advices, the beers and some slides :)
- Aristide Fattori for his advices on hyperbdg.
- rookie for his advices on virtualization and his kindness.
- Ivanlef0u for his hints.
- My colleagues at Eurecom, we are kickass :)

Hiring

- Are you interested in security?
- Do you like French Riviera?
- Are you gaining a master of science?
- Are you looking for an internship?
- Are you looking for a PhD?

Let me know and we can drink a beer discussing about that

Questions

QUESTIONS?

emdel@playhack.net
twitter: @emd3l