matasano

Hardware Virtualization Rootkits

Dino A. Dai Zovi

Agenda

- Introductions
- Virtualization (Software and Hardware)
- Intel VT-x (aka “Vanderpool”)
- VM Rootkits
- Implementing a VT-x based Rootkit
- Detecting Hardware-VM Rootkits
- Demonstration

Who We Are

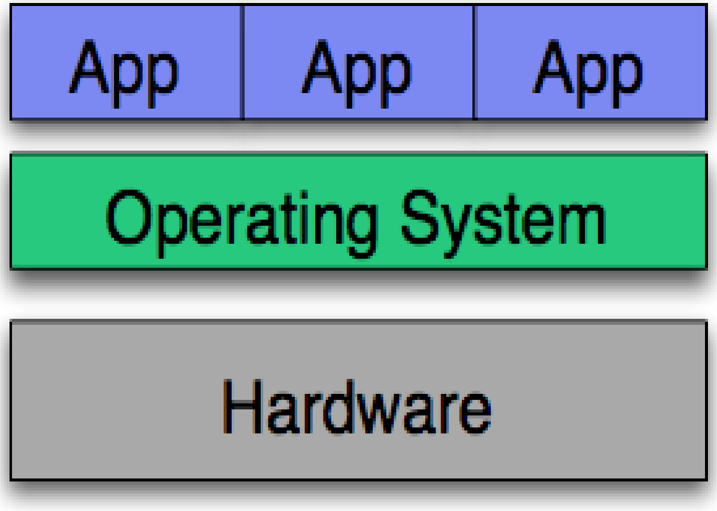
- Dave Goldsmith (@stake cofounder)
- Jeremy Rauch (SecurityFocus cofounder)
- Thomas Ptacek (Arbor)
- Window Snyder (Microsoft XPSP2)
- Dino Dai Zovi (Bloomberg)

What We Do

- DEPLOYSAFE
  Reverse and Pen-Test Products
  for enterprises
- SHIPSAFE
  Audit and Test Products
  for vendors
- CLOCKWORK
  our First Product
  coming July/August 2006

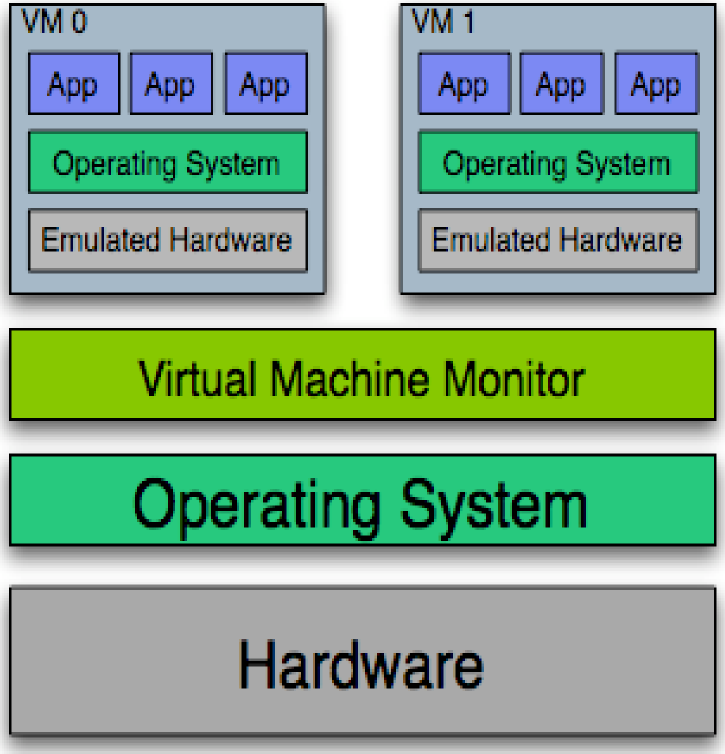
Traditional Operating System

- Modern operating systems perform direct device access in kernel
- “Virtualize” CPU time and devices to applications
  - Pre-emptive multitasking
  - Hardware abstractions



Software-Based Virtualization

- Run multiple operating systems concurrently
- Software Virtual Machine Monitor (VMM) virtualizes hardware
- Approaches:
  - Instruction Interpretation and translation
  - Guest OS de-privileging

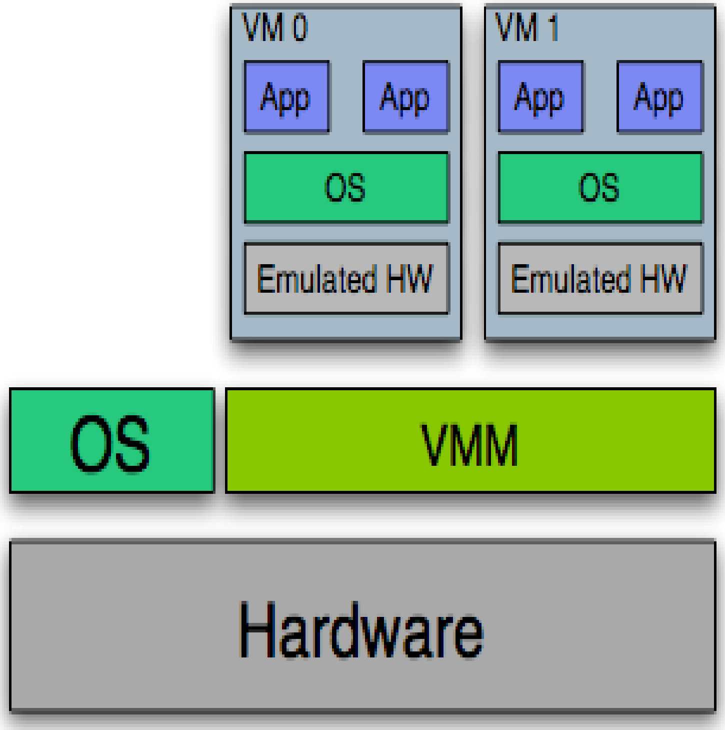


Interpretation and Translation

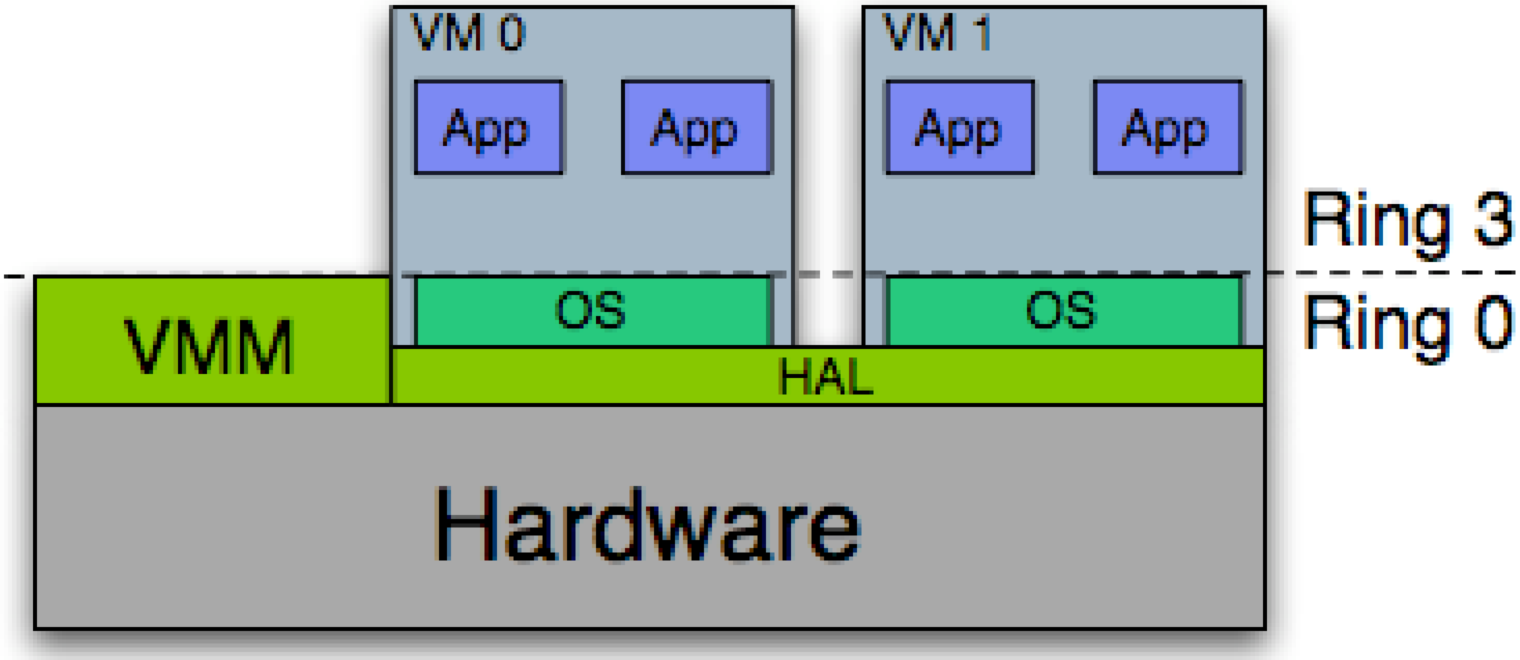
- Interpret processor instructions individually
  - Used if virtual machine may not be the same architecture as the host
- Translate and cache instruction fragments
  - Translate instructions to native instruction set and execute that instead
- Translate privileged instructions
  - Run user mode code natively
  - Translate privileged instructions to emulate expected behavior

Guest OS De-privileging

- VMM occupies Ring 0 along with Host OS
- VMs run in Ring 1
- VMM is essentially a fault handler
  - Privileged operations by guest cause fault
  - Operation is performed or emulated by VMM



Hardware Virtualization



Hardware Virtualization

- Abstracts CPU beyond Ring 0 or Supervisor mode
- New VMM instructions can only be issued in “root” domain
- Events cause transition from guest OS to hypervisor OS.
- Guest/Host state is stored in memory

Hardware Virtualization Implementations

- IBM Logical Partitioning (LPAR)
  - IBM POWER5 processors (1999)
- Intel VT
  - VT-I: Future Itanium processors
  - VT-x: Core Duo and Solo (Jan 2006)
- AMD Pacifica
  - Athlon 64 X2 and FX (June 2006)

Intel VT-x Overview

- Processor operates in two different modes
  - *VMX root* (fully privileged ring 0)
  - *VMX non-root* (less privileged ring 0)
- Virtual Machine Monitor launches Virtual Machines in VMX non-root mode
- Events may cause a *VM exit*
  - Selective exceptions, I/O device access, instructions, special register access
  - VMX non-root state is swapped out
  - VMX root state is swapped in

Intel VT-x in Detail

- Adds 10 new instructions
- Stores host and guest state in Virtual Machine Control Structure (VMCS)
  - Control registers
  - Debug register (DR7)
  - RSP, RIP, RFLAGS
  - Selector, base, ^^^limit, and access rights for segments (CS, SS, DS, ES, FS, GS, LDTR, TR)
  - GDTR, IDTR limit and base
  - MSRs

VMX Instruction Set

VMXON/VMXOFF	Enable/Disable VMX operation
VMCLEAR	Initialize VMCS region
VMPTRLD/VMPTRST	Load/Store Current VMCS pointer
VMREAD/VMWRITE	Read or Write VMCS fields
VMLAUNCH/VMRESUME	Launch or resume virtual machine
VMCALL	Issued from virtual machine to call into VMM

Interesting things about VT-x

- The entire OS-visible state of the processor is swapped in/out of memory
- Virtual Machines can have direct memory and device access
  - Intended to minimize VM exit overhead
  - Direct access to portions of I/O space or memory can be trapped
- Preventing detection was a design goal:
  - “There is no software-visible bit whose setting indicates whether a logical processor is in VMX non-root operation. This fact may allow a VMM to prevent guest software from determining that it is running in a virtual machine” -- Intel VT-x specification

Potential VT-x Hacks

- Run native OS as VM, use VT-x for:
  - Fast sleep and resume
  - Remote kernel debugging
  - “Safe-mode” driver development
    - *Checkpoint OS state before entering development driver*
    - *Resume from checkpoint if there is a fault*
    - *Remote debugging is a pain*
- Really nasty rootkits

Virtual Machine Rootkits

- *SubVirt*, Samuel T. King et al, University of Michigan and Microsoft Research
  - Malicous kernel module modifies boot sequence to load original OS inside Virtual PC
- *BluePill*, Joanna Rutkowska, COSEINC
  - VM rootkit for Windows Vista x64 using AMD Pacifica on AMD Athlon 64
- *Vitriol* (Mine)
  - Proof-of-concept VM rootkit for MacOS X using Intel VT-x on Intel Core Duo/Solo

Hardware VM Rootkits

- Starts running in kernel in ring 0, installs *rootkit hypervisor*.
- Carves out some memory for hypervisor
- Migrates running OS into a VM
- Intercepts access to hypervisor memory and selected hardware devices

Implementing a VT-x Rootkit

- Loadable Kernel Extension installs rootkit and unloads itself
- Three main functions:
  - Vmx_init()
    - *Detects and initialized VT-x capabilities*
  - Vmx_fork()
    - *Migrate OS into VM*
  - On_vm_exit()
    - *Handle VM exit events*

Implementing a VT-x Rootkit

- Vmx_fork()
  - Migrates running operating system into VM
  - Set all VM state to state of running OS
  - Set execution controls to minimize VM exits
  - Copy position independent code into memory
  - Execution in VM continues by unloading kernel module
  - Execution upon VM exits resumes in rootkit code

Implementing a VT-x Rootkit

- on_vm_exit()
  - Handles VM exit events
  - Emulate expected behavior
  - Backdoor functionality
    - *Special instruction sequence for change uid of process to 0 (make me root)*
    - *Filter/monitor/record device access*
    - *Hide blocks on disk by filtering ATAPI packets*
    - *Record keystrokes*

Detecting VT-x Rootkits

- There is no hardware bit or register that indicates that the processor is running in VMX non-root mode
- Attempt to use VMX to create a VM
- Attempt to detect VM exit latency

The VMX Test

- VMX instructions always cause a VM exit
- Create a simple VM to execute a few arithmetic instructions and store result
- If a host should support VMX, but it fails, host may be in a VM
- Is a rootkit going to fully emulate VMX?

VM Exit Latency

- Some instructions always cause VM Exit:
  - CPUID, INVD, MOV from CR3, RDMSR, WRMSR and VMX instructions
- Measure latency of these instructions using RDTSC
  - VT-x supports a TSC offset for guests, so this could theoretically be hampered

matasano

Demonstration

For More Information…

- Rootkit or source code is not available
- Xen 3.0 source code
- “Subverting the Windows Kernel for Fun and Profit”, Joanna Rutkowska

matasano

Questions are your way of proving you listened

ddz@matasano.com