Virtualization

CSE509
Connor Fitzsimons
What is a Virtual Machine?

- Java Virtual Machine (Application Virtualization)
- Software to simulate hardware
- Independent
- ‘Separate’
- Use one piece of hardware to simulate many computers
What are Virtual Machines?

Why are they related to security?

Getting past Virtual Machines
Flavors of Virtualization

- Hardware Virtual Machine
  - Emulation (full system virtualization)
    - Complete hardware virtualization. Unmodified Guest OS for a different CPU can run
  - Native Virtualization (full virtualization)
    - Simulates hardware to run an unmodified OS, but OS has to be for the same type of CPU
- Application Virtual Machine (Paravirtualization)
  - Does not simulate hardware, but offers API that requires OS modifications like JIT compilers or interpreters
- Virtual Environment (Virtual Private Server)
  - Used to run applications, doesn’t simulate a kernel
  - Operating System-Level Virtualization
Machine Aggregation (clustering)

- Use number of different computers to simulate a more powerful single machine
- Parallel Virtual Machine (PVM)
- Message Parsing Interface (MPI)
Why use it?

- Running multiple operating systems
- Physical space
- Mobility (USB Drives)
- Sandboxing
- Honeypots
Hypervisor / VMM

- Platform allowing multiple operating systems to run
- Abstraction layer for a virtual machine
  - Equivalence
  - Resource Control
  - Efficiency
This form of virtualization has the VMM software running directly on a given hardware platform (as an operating system control program). A "guest" operating system thus runs at the second level above the hardware. Sometimes referred to as Type 1 or Bare-metal Virtualization.
Virtualization Benefits

- Cost savings
  - Space, power, cooling
- More efficient use of hardware resources
- Common hardware environments
- Instant provisioning
- Disaster recovery
- Data partitioning
- Distributed resource scheduling
  - Load balancing

Source: Joshua Corman
Research Director
The 451 Group
Usages of Virtualization

- Data Center Consolidation
- Multiplatform Devolvement and Testing
- Redundancy, Failover, Reliability
- New & Evolving Usages
  - SaaS & Application Streaming
  - Cloud Computing
  - Premium Content Delivery
  - Isolation & Sandboxing
The Virtualization Solution

Consolidate for Utilization

Today

App
OS
HW0
App
OS
HWn
App
OS
VMM
HW
The Virtualization Solution

- Consolidate for Utilization
- Develop, Test & Deploy

Today

- App
- OS
- VMM
- HW
The Virtualization Solution

Consolidate for Utilization
Develop, Test & Deploy

Dynamic Load Balancing

Today

Emerging

Note: Usage percentages are illustrative only. Actual loads and balancing will vary.
The Virtualization Solution

- Consolidate for Utilization
- Dynamic Load Balancing
- Develop, Test & Deploy
- Failover

Emerging

Today

App
OS
HW0

App
OS
HWn

App
OS

App
OS

HW

VMM
The Virtualization Solution

- Consolidate for Utilization
- Dynamic Load Balancing
- Maintenance with no Downtime
- Develop, Test & Deploy
- Failover
The Virtualization Solution

- Consolidate for Utilization
- Dynamic Load Balancing
- Maintenance with no Downtime
- Develop, Test & Deploy
- Failover
- Resource Pooling

Today

Emerging

Virtualization & Provisioning Layer

HW1  HWn

OS

App
Dynamic Virtual Clients: The evolution of the client computing stack

**Old**

- Engineered as an IMAGE
- Delivered as an INSTALLATION
- Managed at the END-POINT

**Future**

- Engineered as an VIRTUALIZED LAYERS
- Delivered ON-DEMAND
- Managed in the BACK OFFICE
x86 Virtualization* Overview

From This

x86 Architecture

Operating System

Application

Application

Application

To This

x86 Architecture

Virtualization Layer

Operating System

Operating System

Application

Application

Application

*Represents “Type 1” or Bare Metal “Server” Virtualization
x86 Hierarchical Protection Domains/Rings

Adapted from: http://en.wikipedia.org/wiki/Superior_mode
The Four Horsemen Of the Virtualization Security Apocalypse | Hoff | SecTor 2008

Virtualized: Software Only

- The Guest OS is de-privileged into Ring 1 and the VMM takes its place in Ring 0
- The Guest OS still thinks it is running in Ring 0 with all the privileges thereof
- Can cause issues/contests due to contention for the 17 x86 privileged platform control instructions

Physical/Non-Virtualized

Virtualized: Hardware Assisted

- In this example, Intel VT provides the VMM with an exclusive privileged level where it resides and executes (Ring -1)
- The Guest OS is not de-privileged and is running in Ring 0
- Context switching between VMM and Guest OSs are hardware supported

*There is also para-virtualization, not covered here...
Hypervisors Are a Disruptive Commodity...

*Yes, there are others, but these have pretty logos...*
...and they’re showing up everywhere
KVM architecture

qemu-kvm build-up - http://www.linux-kvm.org/page/Buildup
KVM Codebase

- qemu-kvm (latest: 1.2.0): QEMU + userspace kvm extras
- kvm (latest: 88) deprecated used to have kvm-kmod+libkvm+QEMU
- kvm-kmod (same as linux kernel)
  - kvm.ko: general kernel operations
  - kvm-intel.ko (vmx.ko) / kvm-amd.ko (svm.ko): cpu-specific kernel operations

Look inside KVM: What’s in there?

- VCPU & x86 emulation
- AMD (SVM) & Intel (VMX) support (backup slide)
- MMU: TDP-MMU (nested paging) & SoftMMU (shadow paging) (backup slide)
- Eventfd
- IOMMU
- Emulations (timers, interrupt controllers, APICs)
- debugfs & perf events
KVM API

- `loctl-style`: on device `/dev/kvm`
  - System `loctls`: set/get global attrs, create VMs
    - `KVM_GET_API_VERSION, KVM_CREATE_VM`
  - VM `loctls`: set/get attr for VMs
    - `KVM_CREATE_VCPU, KVM_CREATE_IRQCHIP, KVM_GET/SET_CLOCK`
  - VCPU `loctls`: control VCPUs
    - `KVM_RUN, KVM_GET/SET_REGS, KVM_INTERRUPT, KVM_GET/SET_CPUID`

Linux kernel kvm API documentation -
http://www.mjmwired.net/kernel/Documentation/kvm/api.txt
KVM Control Flow

With preemption or Without preemption

Avi Kivity, “Integrating KVM with Linux”, KVM Forum 2010,
VM Exit Scenario

/* The exit handlers return 1 if the exit was handled fully and guest execution may resume. Otherwise they set the kvm run parameter to indicate what needs to be done to userspace and return 0. */

static int (kvm_vmx_exit_handlers[])(struct kvm_vcpu *vcpu) = {
    [EXIT_REASON_EXCEPTION_NMI] = handle_exception,
    [EXIT_REASON_EXTERNAL_INTERRUPT] = handle_external_interrupt,
    [EXIT_REASON_TRIPLE_FAULT] = handle_triple_fault,
    ...
    [EXIT_REASON_VMCALL] = handle_vmcall,
    [EXIT_REASON_VMCLEAR] = handle_vmclear,
    ...
    [EXIT_REASON_TASK_SWITCH] = handle_task_switch,
    ...
    [EXIT_REASON_MWAIT_INSTRUCTION] = handle_invalid_op,
    [EXIT_REASON_MONITOR_INSTRUCTION] = handle_invalid_op,
};

Shashank Rachamalla, “Kernel Virtual Machine”,
KVM initialization (W/VMX)

- kvm_init
- enable_vpid = 0
- hardwar_setup
- vpid = 0
- ept = 0
- FlexPriority
- LargePage
- kvm_disable_largepage()
- kvm_init
- nested_setup_ctls
- enable_vmxarea
- set-up crs
- csr-config
- create
- kvm-enable
- kvm-enable
KVM initialization (w/SVM)

1. `init_kvm_mmu`
2. TDP enabled?
3. `kvm_enabled_tdp()`
4. Create new kvm_mmu context, assign to vcpu
5. SVM hardware enabled?
6. `kvm_enabled_tdp()`
7. `enable_bits(SVM)`
8. `kvm_mmu()`
9. `setup_vm()`
10. Nested?
11. `create_per_cpu_svm_cpu_data`
12. NPI?
13. Create new kvm_mmu context, assign to vcpu
14. `kvm_mmu()`
15. `setup_vm()`
16. `kvm_init`
Many debates and much ado stems from the inability to distinguish between three fundamental concerns:

- Securing Virtualization
- Virtualizing Security
- Security Via Virtualization

Separate the technical, architectural, and philosophical from the functional, operational and organizational
Threat Models In Review

1. Guest to Guest
2. Guest to Host/VMM/HW
3. Guest to Self
4. External to Host/VMM/HW
5. External to Guest
6. Host/VMM to All...
7. Hardware to VMM
Threat Models In Review

1. Guest to Guest
2. Guest to Host/VMM/HW
3. Guest to Self
4. External to Host/VMM/HW
5. External to Guest
6. Host/VMM to All...
7. Hardware to VMM
Threat Models In Review

1. Guest to Guest
2. Guest to Host/VMM/HW
3. Guest to Self
4. External to Host/VMM/HW
5. External to Guest
6. Host/VMM to All...
7. Hardware to VMM
Threat Models In Review

1. Guest to Guest
2. Guest to Host/VMM/HW
3. Guest to Self
4. External to Host/VMM/HW
5. External to Guest
6. Host/VMM to All...
7. Hardware to VMM
1. Guest to Guest
2. Guest to Host/VMM/HW
3. Guest to Self
4. External to Host/VMM/HW
5. External to Guest
6. Host/VMM to All...
7. Hardware to VMM
Threat Models In Review

1. Guest to Guest
2. Guest to Host/VMM/HW
3. Guest to Self
4. External to Host/VMM/HW
5. External to Guest
6. Host/VMM to All...
7. Hardware to VMM
Threat Models In Review

1. Guest to Guest
2. Guest to Host/VMM/HW
3. Guest to Self
4. External to Host/VMM/HW
5. External to Guest
6. Host/VMM to All...
7. Hardware to VMM
Threat Models In Review

1. Guest to Guest
2. Guest to Host/VMM/HW
3. Guest to Self
4. External to Host/VMM/HW
5. External to Guest
6. Host/VMM to All...
7. Hardware to VMM
But d00d, What About Virtualization Malware?
But d00d, What About Virtualization Malware!?

There are many really interesting topics to discuss here:

- Hypervisor malware, rootkits & hyperjacking
- Exploiting virtualization-enabled chipsets for fun and profit
- Peripheral Hardware/Firmware abuse
- Control channel manipulation

I’m neither qualified or motivated to talk about these topics and we’ve got much more profound and fundamental sets of issues to discuss.

There are lots of other talks featuring this stuff...
Security Challenges to Virtualization

- New attack vector: Hyperjacking
- New attack vector: VM Jumping/Guest Hopping
- VMs and Network Security
- Compliance, Update and Patching

Solutions for Virtualization Security

- Verified Launch and Secure Root of Trust
- Segmentation & Hardening your VMM/VMs
- Virtualization Management, Security and Monitoring solutions
- Integrated Trust Solutions

RSACONFERENCE2010
HyperJacking

- Virtualization allows multiple instances of an operating system to be run on a single box, greatly improving hardware utilization levels.

- Because the hypervisor actually runs underneath the operating system, it makes it a particular juicy target for nefarious types, hell bent on gaining control of computer servers. Get control of the hypervisor and you control everything running on the machine.

- Hyperjacking involves installing a rogue hypervisor that can take complete control of a server. Regular security measures are ineffective because the OS will not even be aware that the machine has been compromised.
Blue Pill/SubVirt use virtualization technology to create an ultra-thin hypervisor that takes complete control of the underlying operating system.

Before Infection

<table>
<thead>
<tr>
<th>host application</th>
<th>host application</th>
<th>guest operating system</th>
</tr>
</thead>
<tbody>
<tr>
<td>host operating system</td>
<td>virtual-machine monitor (VMM)</td>
<td>host hardware</td>
</tr>
</tbody>
</table>

After Infection

<table>
<thead>
<tr>
<th>malicious service</th>
<th>malicious service</th>
<th>target operating system</th>
</tr>
</thead>
<tbody>
<tr>
<td>host operating system</td>
<td>virtual-machine monitor (VMM)</td>
<td>host hardware</td>
</tr>
</tbody>
</table>

SubVirt: Implementing malware with virtual machines
Samuel King & Peter Chen, University of Michigan
Yi-Min Wang, Chad Verbowski, Helen Wang, Jacob Lorch, Microsoft Research
BluePill
Joanna Rutkowska, Invisible Things
Intel Trusted Execution Technology™

A hardware based security foundation to build and maintain a *chain of trust*, to protect the platform from software based attacks

1. **Verified Launch**: Intel TXT hardware-based chain of trust enables launch of MLE into a known, expected state. Changes to MLE can be detected via hash-based measurements.

2. **Protected Configuration**: Intel TXT hardware protects the launched configurations from malicious SW. Maintaining integrity of the measured launched environment identity.

3. **Secret Protection**: Intel TXT hardware removes residual data at improper MLE shut down, protecting data from memory snooping software.

**Intel TXT enhances and complements the capabilities of VT to provide more robust trusted platforms**
VM Jumping/ Guest-hopping threats

- Leverages vulnerabilities in Hypervisors that allow Malware to beat VM protections and gain access to other hosts. The driver for these attacks is that a Hypervisor has to provide at least the illusion of a “ring 0” for a guest operating system to run in.

- The solution here is twofold:
  - 1. Harden the VMs
    - Keep OS and Applications patches updated
  - 2. Segmentation
    - Place applications with like security postures together and isolated from higher/lower level secured applications and systems.

Dark Reading on Virtualization Security
Thomas Ptacek
Host / Hypervisor Level Tools

- **Successes** – Transparent Execution
  - NICKLE – Guest Kernel Code Authenticated Execution
  - SecVisor – Virtualized Kernel Code; W&X
  - TrustVisor – Memory Isolation between PALs

- **Failures** – Semantic Gap
  - Process Implanting – Bridges Gap – Not Fully Transparent
  - Broken Malware Detection
  - Control Flow & Return to Lib-C Attacks
New Attacks

- DKSM Attack
- Loss of cryptographic key data
- Co-Residency can be obtained (Get Off My Cloud)
- Side Channel – Full Risk is Unknown

Trick both Host and Guest level security tools
Hypervisor Level Solutions

- **Virtuoso** – Reduce Semantic Gap
  - Automatically Generate Introspection Tools
  - Training Period

- **HookSafe** – Secure guest kernel hooks
  - Relocate all kernel hooks
  - Training Period

- **HyperSafe** – Secure hypervisor control flow
  - Similar to SecVisor
Guest Level Solutions

- HomeAlone – Determine if any other guests are running on the host
  - Side Channel Cache Analysis
  - Reduces Utilization
References

- DKSM: Subverting Virtual Machine Introspection for Fun and Profit
- Virtuoso: Narrowing the Semantic Gap in Virtual Machine Introspection
- Countering Kernel Rootkits with Lightweight Hook Protection (HookSafe)
- Guest-Transparent Prevention of Kernel Rootkits with VMM-Based Memory Shadowing
- HomeAlone: Co-Residency Detection in the Cloud via Side-Channel Analysis
- HyperSafe: A Lightweight Approach to Provide Lifetime Hypervisor Control-Flow Integrity
- Flicker: An Execution Infrastructure for TCB Minimization
- TrustVisor: Efficient TCB Reduction and Attestation
- Process Implanting: A New Active Introspection Framework for Virtualization
- Secure In-VM Monitoring Using Hardware Virtualization
- SecVisor: A Tiny Hypervisor to Provide Lifetime Kernel Code Integrity for Commodity Oses
- Hey, You, get off of my cloud: Exploring information leakage in third-party compute clouds.
Extra Slides - NICKLE

(a) Kernel code authorization and copying
(b) Guest physical address redirection
Fig. 3. A DKSM attack against loaded modules

(a) An inside view of loaded modules from the first run of lsmod before launching the attack

(b) An inside view of loaded modules from the second run of lsmod after launching the attack

(c) An external view of loaded modules from xenAccess after launching the attack
Figure 5: The implementation of hook indirection
Figure 5. True detection rates for different foe applications (n = 25, $\alpha = 1\%$)
Extra Slides – Process Implanting

Figure 1. Overall design of Process Implanting framework
Extra Slides - Virtuoso

Figure 2. A high-level conceptual view of our system for generating introspection tools. This view corresponds to the training and analysis phases shown in Figure 1. The programmer creates an in-guest program that computes the desired introspection quantity by calling standard OS APIs. This in-guest program is then run repeatedly under various system states, and the instructions executed are logged. These instruction traces are then analyzed to isolate just the instructions that compute the introspection quantity, merged into a unified program, and then translated into an out-of-guest introspection tool.