CSE543 - Computer and Network Security
Module: Virtualization

Professor Trent Jaeger
Q: What is the **primary goal** of system security?

OS enables multiple users/programs to share resources on a physical device

- OS's now have millions of lines of code
- Access control policies of OS become complex
  - E.g., SELinux

**What can we say about security?**
Virtual Machines

• Instead of using system software to control sharing, use system software to enable *isolation*

• Virtualization
  ‣ “a technique for hiding the physical characteristics of computing resources from the way in which others systems, applications, and end users interact with those resources”

• Virtual Machines
  ‣ Single physical resource can appear as multiple logical resources
Virtual Machine

• **Full system simulation**
  ‣ CPU can be simulated

• **Paravirtualization (Xen)**
  ‣ VM has a special API
  ‣ Requires OS changes

• **Native virtualization (VMWare)**
  ‣ Simulate enough HW to run OS
  ‣ OS is for same CPU

• **Application virtualization (JVM)**
  ‣ Application API
Virtual Machine Monitor Approaches

Type 2 VMM
- Host OS
- VMM
- Hardware
- JVM
- CLR
- VMware Workstation

Hybrid VMM
- Host OS
- VMM
- Hardware
- MS Virtual Server
- KVM

Type 1 VMM
- VMM
- Hardware
- VMware ESX
- Xen
- MS Hyper-V
How Can VMs Improve Security?

- **Isolation**
  - Separate two applications to run in two VMs

- **Specialize**
  - Run a hardened, specialized kernel for some applications

- **Isolate groups of VMs**
  - Like a VLAN

- **Better IDS from outside the VM**
  - VM Introspection

- **Control data release to VMs**
  - TCB can decide whether to release data to a new VM

- And more...
VAX VMM Security Kernel

- A1 assured system that enforces MLS
  - Based on an assured virtual machine monitor (VMM)
    - AKA hypervisor
VAX VMM Goals

• Provide isolation boundaries between VMs
  ‣ On real (at the time) hardware
• Keep OS and apps working without modification (much...)
• Enforce MLS access control among VMs
• Enable practical management of security at coarse granularity
  ‣ E.g., VMs, not processes
• Full formal assurance!

• A general-purpose foundation for security!
Provide Isolation

• Why would this be difficult?
  ‣ Real hardware (at that time) was designed for a single OS

• DEC Hardware, VAX/VMS OS
  ‣ UNIX-ish system

• Fundamental problem
  ‣ Some HW instructions are sensitive, but not privileged
    • Sensitive: May impact security - lots on VAX
    • Privileged: Must run in ring 0

• What property should a solution meet?
VAX VMM Access Control

• Want to enforce MLS
  ‣ What kind of protection system do we need?
VAX VMM Access Control

- **Subjects and objects**
  - Coarse-grained access control possible
    - VMs are subjects (as are users)
    - Disk partitions (volumes) are objects
      - Also devices, security kernel files, virtualized resources

- **Lattice policies** for secrecy and integrity
  - Bell-LaPadula for secrecy
  - Biba for integrity

- **Privileges** for special operations
  - E.g., administrative operations

- **Discretionary access controls**
  - Have them, but limited within MAC
Keep OS/Apps Running

- Why would this be difficult?
  - OS thinks it has control of ...
Keep OS/Apps Running

- Why would this be difficult?
  - OS thinks it has control of I/O processing
- But, now the VMM manages hardware resources
  - What happens when OS performs device I/O?
    - Sensitive instruction trapped? Then, no problem
      - But if these are just memory accesses...
- This depends a lot on the hardware design
  - IBM VM/370 used sensitive instructions for I/O
  - VAX and Intel x86 used separate I/O memory
- What can be done?
Paravirtualization

- Modify OS to be aware of virtualization for I/O
  - Mentioned this at start
- Also done for Xen/x86 and VMware initially
VAX VMM Security Kernel

• A1 assured virtual machine system
• Virtualization
  ‣ Provide isolation
    • Sensitive instructions must be virtualized (i.e., require privilege)
    • Access to sensitive data must be virtualized (ditto)
  ‣ MLS
    • Mandatory protection of VMs, volumes
  ‣ I/O Processing
    • Paravirtualization
    • Special driver interface (all in VMM security kernel)
VAX VMM Challenges

- **Q:** Why was the project cancelled?
- Manual formal assurance has impacts
  - New Drivers? In VMM... Need to assure again
  - Limited functionality
    - No Ethernet
  - 48K LOC (bigger than Multics)
    - 11K of assembly
- Does modern virtualization support in hardware solve these problems?
  - Modern hardware
  - Modern hypervisors
  - Modern assurance
IOMMU Role In System

- Application
- Application
- Application
- System Software
- MMU
- RAM
- IOMMU
- Peripheral
- Peripheral
- Peripheral

Control
Future Virtualization

• Modern Hardware
  ‣ Native Virtualization Support
  ‣ IOMMU

• Modern Hypervisors
  ‣ Xen is 300K+ LOC
  ‣ MAC enforcement is becoming complex
    • Xen Security Modules

• Modern Assurance
  ‣ Some advances, but small (seL4)
  ‣ 10K LOC is max that has been assured
Xen

- Paravirtualized Hypervisor
- Privileged VM

Diagram:
- VM: DomU
  - Guest OS'
  - Partitioned Resources
- Dom 0
  - VM Services
  - Host OS'
  - Drivers
- VM: DomU
  - Guest OS'
  - Device Requests

Xen Hypervisor
MAC for Modern VMMs

• Xen, VMware, etc. provide
  ‣ Isolation and I/O: sensitive instructions are made privileged
  ‣ What about enforcing flexible MAC policies?
    • This is something that VAXVMM did...
NetTop

- Isolated networks of VMs
- Alternative to “air gap” security
Intrusion Detection w/ VMs

• Can virtualization help in detecting an intrusion?

• **Network intrusion detection**
  ‣ Can only track packets to and from host
  ‣ Cannot see what is running on the host

• **Host intrusion detection**
  ‣ Can see processes on host
  ‣ **But adversary can see HIDS too!**
  ‣ Stuxnet took advantage of that
Intrusion Detection w/ VMs

- Garfinkel and Rosenblum paper (NDSS 2003)
- **Premise:** Use VMM to enable introspection of one VM from another
  - For antivirus or host intrusion detection
- Leverages 3 properties of VMM
  - **Isolation:** protect from target
  - **Inspection:** can see target’s memory
  - **Interposition:** can intercept privileged instructions
- Can then “checkpoint” target VM
  - What is the checkpoint algorithm in terms of above 3?
Figure 1. A High-Level View of our VMI-Based IDS Architecture: On the right is the virtual machine (VM) that runs the host being monitored. On the left is the VMI-based IDS with its major components: the OS interface library that provides an OS-level view of the VM by interpreting the hardware state exported by the VMM, the policy engine consisting of a common framework for building policies, and policy modules that implement specific intrusion detection policies. The virtual machine monitor provides a substrate that isolates the IDS from the monitored VM and allows the IDS to inspect the state of the VM. The VMM also allows the IDS to interpose on interactions between the guest OS/guest applications and the virtual hardware.
Introspection Challenges

• Can you find what you are looking for?
  ‣ OS’s are complex and have important dynamic data
    • Lots of function pointers (data, but not really dynamic)
  ‣ Semantic gap gets larger when you want to inspect apps

• Can you monitor everything you need to?
  ‣ Need to create at critical times
  ‣ Use privileged commands, hardware watchpoints, debuggers, or voluntary hooks (like paravirtualization)
  ‣ Too many interrupts impedes performance

• Can you protect yourself from adversary?
  ‣ Adversary could try to compromise IDS from VM
  ‣ Adversary could try to compromise VMM from VM or IDS
Virtual Machine Threats

- How does the insertion of a virtual machine layer change the threats against the system?
Virtual Machine Rootkit

• Rootkit
  ‣ Malicious software installed by an attacker on a system
  ‣ Enable it to run on each boot

• OS Rootkits
  ‣ Kernel module, signal handler, ...
  ‣ When the kernel is booted, the module is installed and intercepts user process requests, interrupts, etc.
  ‣ E.g., keylogger

• VM Rootkit
  ‣ Research project from Michigan and Microsoft
  ‣ If security service runs in VM, then a rootkit in VMM can evade security
Java Virtual Machine

• Interpret Java bytecodes
  ‣ Machine specification defined by bytecode
  ‣ On all architectures, run same bytecodes
    • Write once, run anywhere

• Can run multiple programs w/i JVM simultaneously
  ‣ Different ‘classloaders’ can result in different protection domains

• How do we enforce access control?
Java Security Architecture

- Java 1.0: Applets and Applications
- Java 1.1: Signed code (trusted remote -- think Authenticode)
Java Security Architecture

- Java 1.2: Flexible access control, included in Java 2
Stack Inspection

- Authorize based on protection domains on the stack
  - Union of all sources
    - All must have permission

<table>
<thead>
<tr>
<th>class</th>
<th>method</th>
<th>protection domain</th>
</tr>
</thead>
<tbody>
<tr>
<td>Example2b</td>
<td>main()</td>
<td>CDROM</td>
</tr>
<tr>
<td>com.artima.security.stranger.Stranger</td>
<td>doYourThing()</td>
<td>STRANGER</td>
</tr>
<tr>
<td>com.artima.security.friend.Friend</td>
<td>doYourThing()</td>
<td>FRIEND</td>
</tr>
<tr>
<td>java.security.AccessController</td>
<td>dPrivileged()</td>
<td>BOOTSTRAP</td>
</tr>
<tr>
<td>com.artima.security.friend.Friend$1</td>
<td>run()</td>
<td>FRIEND</td>
</tr>
<tr>
<td>TextFileDisplayer</td>
<td>doYourThing()</td>
<td>CDROM</td>
</tr>
<tr>
<td>java.io.FileReader</td>
<td>&lt;init&gt;()</td>
<td>BOOTSTRAP</td>
</tr>
<tr>
<td>java.io.FileInputStream</td>
<td>&lt;init&gt;()</td>
<td>BOOTSTRAP</td>
</tr>
<tr>
<td>java.lang.SecurityManager</td>
<td>checkRead()</td>
<td>BOOTSTRAP</td>
</tr>
<tr>
<td>java.lang.SecurityManager</td>
<td>checkPermission()</td>
<td>BOOTSTRAP</td>
</tr>
<tr>
<td>java.security.AccessController</td>
<td>checkPermission()</td>
<td>BOOTSTRAP</td>
</tr>
<tr>
<td>java.security.AccessControlContext</td>
<td>checkPermission()</td>
<td>BOOTSTRAP</td>
</tr>
</tbody>
</table>
Do Privileged

- `doPrivileged` terminates backtrace
- Like setuid, with similar risks

<table>
<thead>
<tr>
<th>class</th>
<th>method</th>
<th>protection domain</th>
</tr>
</thead>
<tbody>
<tr>
<td>com.artima.security.stranger.Stranger</td>
<td>main()</td>
<td>CDROM</td>
</tr>
<tr>
<td>com.artima.security.friend.Friend</td>
<td>doYourThing()</td>
<td>STRANGER</td>
</tr>
<tr>
<td>java.security.AccessController</td>
<td>doYourThing()</td>
<td>FRIEND</td>
</tr>
<tr>
<td>com.artima.security.friend.Friend$1</td>
<td>dPrivileged()</td>
<td>BOOTSTRAP</td>
</tr>
<tr>
<td>TextFileDisplay</td>
<td>run()</td>
<td>FRIEND</td>
</tr>
<tr>
<td>java.io.FileReader</td>
<td>doYourThing()</td>
<td>CDROM</td>
</tr>
<tr>
<td>java.io.FileInputStream</td>
<td>&lt;init&gt;()</td>
<td>BOOTSTRAP</td>
</tr>
<tr>
<td>java.lang.SecurityManager</td>
<td>checkRead()</td>
<td>BOOTSTRAP</td>
</tr>
<tr>
<td>java.lang.SecurityManager</td>
<td>checkPermission()</td>
<td></td>
</tr>
<tr>
<td>java.security.AccessController</td>
<td>checkPermission()</td>
<td></td>
</tr>
<tr>
<td>java.security.AccessControlContext</td>
<td>checkPermission()</td>
<td></td>
</tr>
</tbody>
</table>
Take Away

• VM systems focus on isolation between VMs
  ‣ Useful for coarse-grained security

• VM systems sometimes provide MAC to allow controlled interaction
  ‣ Same kind of policies as for OS, coarse-grained objects (VMs)

• Can use for VM introspection
  ‣ Watch out for VMM rootkits...