CSE543 - Computer and Network Security

Module: Virtualization

Professor Trent Jaeger
Operating System Quandary

• Q: What is the primary goal of OS in 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
Virtualization Architectures

- **Full system simulation (QEMU)**
  - 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

Virtual Machine Types

- **Type I**
  - Lowest layer of software is VMM
  - E.g., Xen, VAX VMM, etc.

- **Type II**
  - Runs on a host operating system
  - E.g., VMWare, JVM, etc.

Q: What are the trust model issues with Type II compared to Type I?
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...
What is Virtualized

- What do you need to do to virtualize a system?
- All sensitive instructions must be privileged
  - Sensitive: May impact security of VMs
  - Privileged: Must run in privileged domain (ring 0)
- VMs must still be able to use devices
  - Must be able to use host’s hardware devices despite not being the “host’s” operating system
  - Without compromising the VMM
- Must control access to virtualized resources
  - Different granularity than typical OS access control
- **Subjects**: VMs; **Objects**: Disk volumes
• A1 assured system that enforces MLS (circa 1991)
  ‣ Based on an assured virtual machine monitor (VMM)
    • AKA hypervisor
VAX VMM Security Kernel

• A1 assured virtual machine system

• Virtualization
  ▸ Provide isolation
    • Sensitive instructions must be virtualized (i.e., require privilege)
  ▸ MLS
    • Mandatory protection of VMs, volumes
  ▸ I/O Processing
    • Paravirtualization - modify OSes to call VMM
    • Special driver interface (all in VMM security kernel)
Modern Virtualization

• Modern **Hardware**
  ‣ Native Virtualization Support
  ‣ IOMMU

• Modern **Hypervisors**
  ‣ Xen is 300K+ LOC
  ‣ MAC enforcement in VMMs
    • NetTop, sHype, Xen Security Modules

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

• What does virtualization hardware do?
  • Self-virtualization
    ‣ All sensitive instructions are now privileged
  • Device I/O
    ‣ Paravirtualization improvements
    ‣ Direct device assignment (using IOMMU for protection)
IOMMU

• Memory Management Unit for I/O
• What does a tradition MMU do?
• What does an IOMMU do?
MAC for Modern VMMs

• Xen, VMware, etc. provide
  ‣ Isolation and I/O: sensitive instructions are made privileged
  ‣ What about enforcing flexible MAC policies?
    • VAXVMM could do that…
Xen

- Originally, Paravirtualized Hypervisor
- Privileged VM

Xen Hypervisor

VM: DomU

Guest OS’

Partitioned Resources

VM Services

Dom 0

Host OS’

Drivers

VM: DomU

Guest OS’

Device Requests
NetTop

- Isolated networks of VMs
- Alternative to “air gap” security

```
VM: Secret
VM: Public
VM: Secret
VM: Public
```

```
Guest OS'
Guest OS'
Guest OS'
Guest OS'
```

```
VMWare MLS
SELinux Host OS
VMWare MLS
SELinux Host OS
```
Xen sHype

- Controlled information flows among VMs
  - Subjects (VMs) and Objects (VMs - via network)
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
Problem: Kernel Rootkits

- Given a root exploit
  - Make the exploit stealthy and persistent
- What’s the best way to do that?
  - Compromise the OS and hide the process
  - Originally, by modifying the boot record - now lots of ways
- E.g., Have OS hide a malicious process by not including it on the task list, but still schedule the process
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?
Intrusion Detection w/ VMs

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 mediate 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
  - Intersection of permissions 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 1</td>
</tr>
<tr>
<td>com.artima.security.stranger.Stranger</td>
<td>doYourThing()</td>
<td>STRANGER 2</td>
</tr>
<tr>
<td>com.artima.security.friend.Friend</td>
<td>doYourThing()</td>
<td>FRIEND 3</td>
</tr>
<tr>
<td>java.security.AccessController</td>
<td>dPrivileged()</td>
<td>BOOTSTRAP 4</td>
</tr>
<tr>
<td>com.artima.security.friend.Friend$1</td>
<td>run()</td>
<td>FRIEND 5</td>
</tr>
<tr>
<td>java.io.FileDisplay</td>
<td>doYourThing()</td>
<td>CDROM 6</td>
</tr>
<tr>
<td>java.io.FileReader</td>
<td>&lt;init&gt;()</td>
<td>BOOTSTRAP 7</td>
</tr>
<tr>
<td>java.lang.SecurityManager</td>
<td>checkRead()</td>
<td>BOOTSTRAP 8</td>
</tr>
<tr>
<td>java.lang.SecurityManager</td>
<td>checkPermission()</td>
<td>BOOTSTRAP 9</td>
</tr>
<tr>
<td>java.security.AccessController</td>
<td>checkPermission()</td>
<td>BOOTSTRAP 10</td>
</tr>
<tr>
<td>java.security.AccessControlContext</td>
<td>checkPermission()</td>
<td>BOOTSTRAP 11</td>
</tr>
<tr>
<td></td>
<td></td>
<td>BOOTSTRAP 12</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>Example2b</td>
<td><code>main()</code></td>
<td>CDROM</td>
</tr>
<tr>
<td><code>com.artima.security.stranger.Stranger</code></td>
<td><code>doYourThing()</code></td>
<td>STRANGER</td>
</tr>
<tr>
<td><code>com.artima.security.friend.Friend</code></td>
<td><code>doYourThing()</code></td>
<td>FRIEND</td>
</tr>
<tr>
<td><code>java.security.AccessController</code></td>
<td><code>dPrivileged()</code></td>
<td>BOOTSTRAP</td>
</tr>
<tr>
<td><code>com.artima.security.friend.Friend$1</code></td>
<td><code>run()</code></td>
<td>FRIEND</td>
</tr>
<tr>
<td><code>java.io.FileReader</code></td>
<td><code>doYourThing()</code></td>
<td>CDROM</td>
</tr>
<tr>
<td><code>java.io.FileInputStream</code></td>
<td><code>&lt;init&gt;()</code></td>
<td>BOOTSTRAP</td>
</tr>
<tr>
<td><code>java.lang.SecurityManager</code></td>
<td><code>checkRead()</code></td>
<td>BOOTSTRAP</td>
</tr>
<tr>
<td><code>java.lang.SecurityManager</code></td>
<td><code>checkPermission()</code></td>
<td>BOOTSTRAP</td>
</tr>
<tr>
<td><code>java.security.AccessController</code></td>
<td><code>checkPermission()</code></td>
<td>BOOTSTRAP</td>
</tr>
<tr>
<td><code>java.security.AccessControlContext</code></td>
<td><code>checkPermission()</code></td>
<td>BOOTSTRAP</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...
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...