CSE543 - Computer and Network Security

Module: Virtualization

Professor Trent Jaeger
Operating System

• Recall Saltzer-Schroeder
  ‣ Q: What is the primary goal of system security?

• OS enables multiple users/programs to share resources on a physical device
  ‣ Access control policies of OS become complex
  ‣ E.g., SELinux

• What are we to do?
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 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?
VM Security

- Isolation of VM computing
- Like a separate machine

Virtual Machine Monitor

Partitioned Resources

Device Requests

Physical Device Controls
VAX VMM Security Kernel

• A1 assured virtual machine system

• Virtualization
  ‣ Protect sensitive state
    • Sensitive instructions must be virtualized (i.e., require privilege)
    • Access to sensitive data must be virtualized (ditto)
  ‣ Need to hide virtualization
    • Systems cannot see that they are being virtualized
  ‣ I/O Processing
    • Need to share access to devices correctly
    • Special driver interface (all in VMM security kernel)
  ‣ Self-virtualization: Run VMM as VM
VM Security

- Do VMs need to communicate or share resources?
- How do they do it?
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
Aside

- **Simple security property**
  - Read-down only (BLP); Read-up only (Biba)
  - S can read O if and only if S's access class dominates (or equal) O

- ***(star)-security property***
  - Write-up only (BLP); Write-down only (Biba)
  - S can write to O if and only if O's access class dominates (or equal) S

- **Basic Security Theorem**
  - Every protection state satisfies simple and *(star)-security properties
  - Bell-LaPadula and Biba meet this trivially
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
NetTop

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

- Paravirtualized Hypervisor
- Privileged VM

![Diagram of Xen Hypervisor]

**VM: DomU**

- Guest OS'

**Partitioned Resources**

**VM: DomU**

- Guest OS'

**Device Requests**

**Dom 0**

- Host OS'
- Drivers

**Xen Hypervisor**
Xen sHype

- Controlled information flows among VMs

![Diagram showing Xen Hypervisor and VMs](image-url)
Future Virtualization

• Modern Hardware
  ‣ Native Virtualization Support
  ‣ IOMMU
  ‣ *Hardware guaranteed isolation is not there*

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

• Modern Assurance
  ‣ Basically the same
  ‣ 10K LOC is max that has been assured
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>CDROM</td>
</tr>
<tr>
<td>TextFileDisplayer</td>
<td>doYourThing()</td>
<td>FRIEND</td>
</tr>
<tr>
<td>java.i.o.FileReader</td>
<td>&lt;init&gt;()</td>
<td>BOOTSTRAP</td>
</tr>
<tr>
<td>java.i.o.FileInputStream</td>
<td>checkRead()</td>
<td>BOOTSTRAP</td>
</tr>
<tr>
<td>java.lang.SecurityManager</td>
<td>checkPermission()</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>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>java.lang.SecurityManager</code></td>
<td><code>run()</code></td>
<td>FRIEND</td>
</tr>
<tr>
<td><code>java.io.FileInput</code></td>
<td><code>doYourThing()</code></td>
<td>CDROM</td>
</tr>
<tr>
<td><code>java.lang.FileReader</code></td>
<td><code>&lt;init&gt;()</code></td>
<td>BOOTSTRAP</td>
</tr>
<tr>
<td><code>java.io.FileInput.Stream</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>
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
Take Away

• VM systems focus on isolation
  ‣ Enable reuse, but limited by security requirements

• Enable limited communication
  ‣ The policies are not trivial