Virtual Machine Security

CSE497b - Spring 2007
Introduction Computer and Network Security
Professor Jaeger

www.cse.psu.edu/~tjaeger/cse497b-s07/
Operating System Quandary

• Q: What is the primary goal of system security?
  • OS enables multiple users/programs to share resources on a physical device

• Q: What happens when we try to enforce Mandatory Access Control policies on UNIX systems
  • Think SELinux policies

• What can we do to simplify?
Virtual Machines

• Instead of using system software to enable 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 Architectures

- **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
Ensure Protection of VMM

• Processor Instructions
  • Each processor supports an instruction set
  • Some can only be run privileged mode
    • i.e., a more privileged ring (ring 0)

• Privileged versus Sensitive Instructions
  • Privileged: only run in ring 0
  • Sensitive: read or write privileged state
  • All sensitive instructions must be privileged

• Examples
  • Page Table Entries: memory accesses
  • Code Segment Selector read: this register indicates level
A Proper VMM

• Virtualization Requirements
  • 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
  • Self-virtualization: Run VMM as VM
    • Can’t do this on traditional x86, but now we have VT architecture
NetTop

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

- Paravirtualized Hypervisor
- Privileged VM
Xen sHype

- Controlled information flows among VMs

VM: DomU

Partitioned Resources

VM: DomU

Guest OS'

Device Requests

Xen Hypervisor

VM Services

Host OS'

Drivers

Dom 0

Ref Mon

Guest OS'
Xen sHype Policies

• Type Enforcement over VM communications

• VM labels are subjects
• VM labels are objects

• How do VMs communicate in Xen?

• **Grant tables**: pass pages between VMs
• **Event channels**: notifications (e.g., when to pass pages)

• sHype controls these

• Q: What about VM communication across systems?
Xen Security Modules

- Comprehensive Reference Monitor interface for Xen
  - Based on LSM ideas
  - Includes about 57 “hooks” (more expected)
  - Supports sHype hooks
  - Plus, hooks for VM management, resource partitioning
- Another aim: Decompose domain 0
  - Specialize kernel for privileged operations
  - E.g., Remove drivers
VM Security Status

- Aim is simplicity
- Are we achieving this?
- Do we care what happens in the VMs?
  - When might we care?
- Trusted computing base
  - How does this compare to traditional OS?
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 Security Architecture

• Java 1.1: Signed code (trusted remote -- think Authenticode)
• 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><code>Example2b</code></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>&lt;init&gt;()</code></td>
<td>CDROM</td>
</tr>
<tr>
<td><code>java.io.FileInputStream</code></td>
<td><code>&lt;init&gt;()</code></td>
<td>STRANGER</td>
</tr>
<tr>
<td><code>java.lang.SecurityManager</code></td>
<td><code>checkPermission()</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>
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>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>run()</td>
<td>BOOTSTRAP</td>
</tr>
<tr>
<td>com.artima.security.friend.Friend$1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>java.io.FileDisplay</td>
<td></td>
<td></td>
</tr>
<tr>
<td>java.lang.SecurityManager</td>
<td>init&gt;</td>
<td></td>
</tr>
<tr>
<td>java.lang.SecurityManager</td>
<td></td>
<td></td>
</tr>
<tr>
<td>java.security.AccessController</td>
<td>checkRead()</td>
<td></td>
</tr>
<tr>
<td>java.security.AccessController$1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>java.security.AccessController</td>
<td>checkPermission()</td>
<td></td>
</tr>
<tr>
<td>java.security.AccessController</td>
<td></td>
<td></td>
</tr>
<tr>
<td>java.security.AccessController$1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The diagram shows the class, method, and protection domain relationships.
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
  – E.g., Can continue to run even if the system appears to be off
Take Away

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

• Enable limited communication
• The policies are not trivial, but refer to coarser-grained objects