Advanced Systems Security: Virtual Machine Systems

Trent Jaeger
Systems and Internet Infrastructure Security (SIIS) Lab
Computer Science and Engineering Department
Pennsylvania State University
Where are we?

- OS Security from Reference Monitor perspective
  - Mediation
    - LSM
  - Tamperproof
    - Linux and SELinux
  - Simple enough to verify
    - Correct code
    - Correct policy
Basis for OS Security

- Isolation
  - A protection domain defines a boundary of isolation

- Based on
  - Rings
  - Address spaces
  - Access control policy

- Do these work for modern OSes?
Virtual Machine Systems

• Protection domain is extended to operating systems on one physical platform
  ‣ Invented for resource utilization

• But, also provide a potential security benefit due to default
  ‣ ISOLATION

• How does VM isolation differ from OS isolation?
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 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?
Hardware Virtualization

- CPU virtualization
  - Instructions (still there)
  - Sensitive instructions must be privileged

- Memory virtualization
  - MMU (still there)
  - Nested/extended page tables

- I/O virtualization
  - IOMMU (new)
  - Chipset support for configuration and address translation
VM Systems and Ref Monitor

- How does a VM System improve ability to achieve reference monitor guarantees?
VM Systems and Ref Monitor

• How does a VM System improve ability to achieve reference monitor guarantees?

• Mediation
  ‣ Mediation between VM interactions

• Tamperproof
  ‣ Protection boundaries between VMs (OS)

• Simple Enough to Verify
  ‣ Code that needs to be correct?
  ‣ Policy
VAX VMM

- A1-assured (formally assured) VMM system
- Carefully crafted VMM
- Mediation
  - VM (subject) and volume (object)
- Tamperproof
  - “Minimal” TCB – VMM only
- Simple enough to verify
  - Code assurance
  - Policy assurance: MLS policy, Biba policy, privileges
VAX VMM Architecture

Applications (Top Secret) | Applications (Secret) | Applications (Unclassified)
--------------------------|-----------------------|--------------------------
Ultrix OS                | VMS OS                | VMS OS

VMM Security Kernel

Memory Device | Disk Device | Print Device | Display Device
--------------|-------------|--------------|---------------
VAX VMM Reference Monitor

• Key design tasks
  ‣ Virtualize processor
    • Make all sensitive instructions privileged
  ‣ More rings
    • Need a new ring for the VMM
  ‣ I/O emulation
  ‣ Self-virtualizable

• What components constitute the VAX VMM reference monitor?
VAX VMM Policy

- MLS
  - Control secrecy
- Biba
  - Control integrity
- Privileges
  - Exceptional accesses
  - Audited
  - There are more of these than meets the eye!
- How is the protection state modified?
VAX VMM Evaluation

- **Mediation**: ensure all security-sensitive operations are mediated?
  - Virtualizing instructions, I/O emulation
  - VM-level operations? Privileges

- **Mediation**: mediate all resources?
  - VMM level

- **Mediation**: verify complete mediation?
  - A1-assured at VMM level
VAX VMM Evaluation

- **Tamperproof**: protect VMM?
  - Similar to Multics (no gatekeepers, but some kind of filters); authentication in VMM; protection system ops in VMM

- **Tamperproof**: protect TCB?
  - All trusted code at ring 0; trusted path from VMs for admin;

- **Verification**: verify code?
  - A1-assured at VMM level

- **Verification**: verify policy?
  - MLS and Biba express goals and policy; Privileges are ad hoc
VAX VMM Challenges

• Despite A1 assurance still several challenges in VAX VMM system
  ‣ Device driver management; no network
  ‣ Amount of assembler code
  ‣ Covert channel countermeasures
  ‣ Implications of ‘privileges’

• Nonetheless, interesting mechanisms
  ‣ Virtualization for security
  ‣ Architecture of VMM system
  ‣ Trusted path administration
Modern VM Systems

- The development of a virtual machine monitor for x86 systems unleashed VMs on the masses
  - Why did this take so long?
- VMware, Xen, KVM, NetTop, …
  - Everyone is a virtual machine monitor now
- How do we implement a reference validation mechanism for these systems?
  - What granularity of control?
Isolation and Network

- Type 1 VM Systems assume that the VMM (and privileged VM) will **isolate guest VMs**
- Then, the problem is to control inter-VM communication
  - VMs talk to VMM (hypercalls, like system calls)
  - All other communication is **via the network**
- **sHype** adds reference monitor for controlling network access between VMs
- **NetTop** is built on VMware where only VMs of the same label may communicate via network
Control of VMM Resources

• There are many virtual machine monitor resources that may be used to communicate
  ‣ Memory, devices, IPC, VMs themselves, …
  ‣ E.g., VMware permits VMCI – like IPC between VMs

• **Xen Security Modules (XSM)** adds reference validation on the Xen hypervisor’s distribution of these resources
  ‣ Less trust in privileged VMs, so finer-grained policy results

• Minimizing TCB versus simplicity
Xen as a Reference Monitor?

- **Reference Monitor**
  - XSM in Xen
  - Scope includes “dom0” VM
- **Mediation**
  - XSM to control VMM operations
  - SELinux in dom0; use network to communicate
- **Tamperproof**
  - Xen and Linux
- **Verification (Xen)**
  - Xen Code – 200K+ LOC – and Dom 0 Linux
  - Policy – SELinux style
Container Systems

- A hybrid approach is developed in container systems
- Linux containers run multiple Linux systems (process hierarchies) on one Linux host operating system
  - Cgroups enables resource control without starting VMs
  - Also, each container gets its own namespaces for processes, mounts (filesystem), userids, and networks
    - Idea is to give each container an isolated view
- How do we configure access control for containers?
Container Systems

- How do we configure access control for containers?
  - E.g., SELinux across and within containers…
Container Systems

• How do we configure access control for containers?
  ‣ Currently, the host system defines mandatory access control policies that govern every container
  ‣ What are issues with that approach?
Dune

- **Goal**: Safe access to hardware features from processes

- Normally, only the operating system can configure hardware features, such as page tables, ring protection, and TLBs

- However, applications may benefit from direct access to such hardware features
  - Modifying the kernel to provide such access in a sufficiently flexible way while maintaining security is a problem
Dune

• **Approach:** Dune uses virtualization hardware to provide a “process” rather than a “machine” abstraction

• **Alternative:** Instead of modifying the host kernel to achieve application-specific use of hardware features, an alternative is to deploy processes in a VM with a custom OS to do so

• However, launching a process in a VM can be complex because of sharing of OS abstractions, such as file descriptors between parent and child
  ‣ Won’t work if they are in different VMs
Dune – Process Abstraction

- **Process**: Can enter “Dune mode” to access hardware features
  - Including privilege modes, virtual memory registers, page tables, and interrupt, exception, and system call vectors
  - Through use of virtualization hardware – Intel VT-x
    - VMX root and VMX non-root modes
      - VMX root – for VMM
      - VMM non-root – for virtualized operating systems, governed by VMM
    - Dune processes use VMCALL to invoke system calls – with help of library provided
Dune – System Architecture

- **System**: Dune mode is VMX non-root mode
  - Kernel is in VMX root mode like a VMM
  - Dune processes are in VMX non-root mode
  - Dune module intercepts VM exits, which are the only way to access the kernel – for syscalls and traps
- Other processes are unaffected

Figure 1: The Dune system architecture.
Dune – Memory Management

• **Goal:** manage page tables from user processes
  ‣ But, just what programs want to manage – not all
  ‣ Without allowing arbitrary access to memory
• Dune processes reference guest-virtual memory, so protected by extended page table – like process is a VM
  ‣ Sync EPT to kernel PT

![Figure 2: Virtual memory in Dune.](image)
Container Security

- Better or worse than VMs?
- Worse: Containers share the same OS
- Better: Containers only have one application
- Better: Containers can have limited attack surface by running it in a “jail”
- Worse?: Hypervisor can provide stronger isolation than an OS
  ‣ However, Dune shows that such isolation is implemented by VT-x hardware, so same in OS and hypervisor
Conventional OS vs VM System

- **Conventional OS**
  - Broken easily and often
- **VM system**
  - Coarser control based on isolation
- If we trust the VM system and don’t trust the OS, what can we do?
Deploy Critical Applications

- Don’t trust OS, but need its services
- Run programs directly on VMM
  - No services
- Run programs on a specialized, trusted system
  - Custom services must be written (yuk!)
- Reuse untrusted system services
  - Trusted system (custom, but potentially smaller) must enable secure use of such services
Splitting Interfaces

Solution

• Separate application from other apps/kernel
  – Use separate VM for app with a Private OS separate from Commodity OS
• Provide interaction between apps/kernel in a secure way
  – Application developer decides what is sensitive and what is not
    • Separate sensitive part into VM on Private OS
    • Public part remains on Commodity OS
    • Interaction between apps also passes through kernel (eg. pipe(), mkfifo())
    • Sensitive part communicates through system calls with other apps
  – Use policy to decide if system calls are to be performed on commodity OS or private OS
Proxos Guarantees

• Assumption
  – VMM enforces separation
  – Application developer correctly specifies routing rules

• Guarantee
  – Confidentiality and integrity of sensitive private application data inspite of malicious commodity OS
    • VMM => No direct interference possible
    • Commodity OS can interfere with system calls routed to it, which are not security-sensitive
  – Availability not guaranteed
Proxos Routing Language

• Needs to specify which system calls go where (arguments need be considered)
• Solution: Partition system calls by resources they access
  – Disk, Network, UI, Randomness, System Time, Memory
  – Randomness, System Time -> Always routed to VMM
  – Memory -> Always routed to private OS

```
# Rules Section
# route accesses to /etc/secrets to private OS
DISK:("/etc/secrets", priv_fs)
# route accesses to UNIX domain socket bound
# to /tmp/socket and TCP socket bound to peer
# 192.100.0.4 port 1337 to private OS
NETWORK:("unix:/tmp/socket", priv_unix),
       ("tcp:192.100.0.4:1337", priv_tcp)
# route all accesses to stdin, stdout
# and stderr to private OS
UI: (*,priv_ui)

# Methods Section
# individual methods in the private OS
# that are bound to system calls
priv_fs = {
    .open = priv_open,
    .close = priv_close,
    .read = priv_read,
    .write = priv_write,
    .lseek = priv_lseek
 }```

Proxos Implementation

Host Process
pr_execve(app_name)

Linux Kernel
Allocate shared mem
Pass (app_name, addr of shared mem) to VMM
Start private VM

VMM/dom0
VMM creates private VM, gives it shared mem addr
Associate VMID with addr
Return VMID

Private VM
Maps syscall shared mem into addr space

When private VM gives map request, checks if VMID corresponds to addr

Associates VMID with PID of host for identifying future syscalls
Proxos SSH Server
Compare to Privilege Separation

Partitioning Interfaces to Resources

Private SSH Server
- Linux OS
- SSH listening parent
- Command Shell
- Encrypted Traffic
- Set up pipe & start shell
- Private SSH Server
- Password File & Host keys
- Private Application VM

Partitioning Code (Provos et al)

- Network connection
- privilged OpenSSH
  - Listen *:22
  - fork unpriviledged child
- Monitor
  - Request Auth
  - Auth Result
  - State Export
  - fork user child
  - Request PTY
  - Pass PTY
- unprivilledged OpenSSH
  - Network Processing
- user unprivilledged OpenSSH
  - Monitor
  - Network Processing
- Key Exchange
- Authentication
- User Network Data

Timeline
### Implementation Effort

<table>
<thead>
<tr>
<th>Application</th>
<th>Rules</th>
<th>LOC Modified</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dillo</td>
<td>53</td>
<td>22</td>
</tr>
<tr>
<td>SSH Server</td>
<td>35</td>
<td>108</td>
</tr>
<tr>
<td>Apache &amp; OpenSSL</td>
<td>28</td>
<td>667</td>
</tr>
<tr>
<td>Glibc</td>
<td></td>
<td>218</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Total LOC</th>
<th>% Modification</th>
</tr>
</thead>
<tbody>
<tr>
<td>20,528</td>
<td>0.1%</td>
</tr>
<tr>
<td>27,000</td>
<td>0.4%</td>
</tr>
<tr>
<td>135,916</td>
<td>0.5%</td>
</tr>
<tr>
<td>1,775,440</td>
<td>0.01%</td>
</tr>
</tbody>
</table>
Performance

- System call forwarding overhead
  - Context Switch Cost: 14us.

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Linux</th>
<th>Proxos</th>
<th>Overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>NULL system call</td>
<td>0.37</td>
<td>12.88</td>
<td>12.51</td>
</tr>
<tr>
<td>fstat</td>
<td>0.57</td>
<td>14.28</td>
<td>13.71</td>
</tr>
<tr>
<td>stat</td>
<td>8.76</td>
<td>25.98</td>
<td>17.22</td>
</tr>
<tr>
<td>open &amp; close</td>
<td>14.57</td>
<td>47.18</td>
<td>32.61</td>
</tr>
<tr>
<td>read</td>
<td>0.45</td>
<td>13.51</td>
<td>13.06</td>
</tr>
<tr>
<td>write</td>
<td>0.42</td>
<td>13.24</td>
<td>12.82</td>
</tr>
</tbody>
</table>
TrustShadow

- The TrustShadow system employs the Proxos approach to deploy isolated applications that do not trust the Linux kernel
  - An application of Proxos to approximate SGX guarantees (next time)

- Isolated, unmodified applications are launched on the TrustShadow runtime system using the ARM TrustZone “Secure World”

- Runtime intercepts most system calls and forwards to them to the Linux kernel in the “Normal World”
Take Away

• VM Systems provide isolation
  ‣ At OS granularity: some can be untrusted

• Moving towards container systems
  ‣ Dune enables flexible use of hardware by “containers”

• Can we use VM isolation to prevent compromise of applications by malicious OS?
  ‣ Proxos: use a “trusted” OS and redirect service requests
    • Applied in TrustShadow to isolate domains