CIS 610: Advanced Topics in Systems Security
Virtual Machine Systems

Prof. Kevin Butler

Spring 2014
Where are we?

- **OS Security from Reference Monitor perspective**
  - Mediation
  - LSM
  - Tamperproof
    - Linux and TCB
    - 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 in 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 Machine Security

• Rather than using system software to facilitate sharing between VMs, enforce *isolation*.

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

• Virtual machine: single physical resource can appear to be multiple logical resources.
VM Architectures

• Full system simulation
  ‣ simulate operation of the CPU (e.g., Bochs, QEMU)

• Paravirtualization
  ‣ VM has special API, requires OS changes (e.g., Xen)

• Full/Native virtualization
  ‣ support from hardware not requiring OS changes (e.g., VMWare, Xen running on processor with Intel VT extensions)

• Application Virtualization
  ‣ application API (e.g., JVM)
VM Types

- **Type 1**
  - lowest-level, “bare metal” VMM
  - e.g., Xen, VAX WMM

- **Type 2**
  - Runs on host operating system
  - e.g., VirtualBox, KVM, JVM

- What are the trust issues with Type 2 VMMs compared to Type 1?
• 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 OS
- Simple Enough to Verify
  - Code that needs to be correct?
  - Policy
VAX VMM

- A I-assured VMM system
- Carefully crafted VMM
- Mediation
  - VM interaction
- Tamperproof
  - Minimal TCB
- Simple enough to verify
  - Code assurance
  - Policy assurance: MLS policy, Biba policy, privileges
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 Design

- Applications (Top Secret)
  - Ultrix OS

- Applications (Secret)
  - VMS OS

- Applications (Unclassified)
  - VMS OS

VMM Security Kernel

- Memory Device
- Disk Device
- Print Device
- Display Device

...
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?
  ‣ A I-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; fixed system?

• **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 AI 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
  ‣ Trusted path administration
  ‣ Architecture of VMM
  ‣ Virtualization for security
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

- VMware and NetTop assume that the VMM (and privileged VM) will isolate guest VMs.

- Then, the problem is to control inter-VM communication.
  - Only other communication is via the network.

- VMware uses network layer.
  - Also supports VMCI for direct communication between VMs.

- NetTop is built on VMware where only VMs of the same label may communicate.
VMs as Processes

- Type II VM systems can treat VMs as processes
- **KVM** uses SELinux to control access of VMs as if they are a process
  - VMs are processes to the host OS
  - VMs can access host OS resources (files)
- Uses SELinux to control VM access
Control of VMM Resources

- There are many virtual machine monitor resources that may be used to communicate
  - Memory, devices, IPC, …

- sHype adds reference monitor for some objects (IPC) and the privileged VM uses for networking

- 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 Linux and user-level

- Mediation
  - XSM to control VMM operations
  - SELinux in dom0; use network to communicate

- Tamperproof
  - Xen has a much larger TCB, and more flexible

- Verification
  - Code – lots
  - Policy – SELinux style
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?
Untrusted OS

• Don’t trust OS, but need its services

• Run programs on a specialized, trusted system
  ‣ But use conventional OS like an untrusted network

• Run programs directly on VMM
  ‣ But use conventional OS like an untrusted network
    (must use more OS)

• How do we accomplish these options?
Options

• Microkernels
  ‣ Reduce code running in kernel mode
  ‣ But, need the same services
  ‣ These are just as “trusted” running in user-space

• SELinux/AppArmor/Trusted Solaris
  ‣ What do you think?

• Isolate in VM systems (e.g., Terra)
  ‣ Can deploy an application on a custom OS
  ‣ Still have to trust all services used though
A solution should...

- **Ease Adoption**
  - It is usable...

- **Support Diverse Applications**
  - ...to a lot of people...

- **Have an Incremental Path to Higher Assurance**
  - ...also.
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 Architecture
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 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
Proxos Implementation
Proxos SSH Server

Private VM
- Private OS
  - Passwords
  - Host Keys
- SSH Server
  - Encrypt
- Proxos

Commodity OS VM
- Command Shell
- Host Process
- Linux Kernel
- Pipe

Network
Compare to Privilege Separation
Remaining Problem

- Deploying a custom OS is painful
  - Building a special kernel is non-trivial
- And it may not be secure itself
  - Still need a methodology to determine code correctness and tamperproofing
- What if you want to eliminate trust in the OS altogether?
Insight: Shadowing Memory

- VMMs need to manage physical to virtual mapping of memory
- This is done with a shadow page table
- Multi-shadowing give context aware views of this memory
  - Use encryption instead
Memory Cloaking

- Not new idea
  - XOM, LT
- Leverage power of VMMs
- Encrypt the pages in memory
  - (IV, H) meta data
- This is used for writes to disk too
  - How do we store the metadata?
Overshadow

- Mediate all application interaction with OS to ensure correct cloaking of memory
  - Context Identification
  - Secure Control Transfer
  - System Call Adaptation
  - Mapping Cloaked Resources
  - Managing Protection Metadata
• The key to overshadow is the **Shim**
  ‣ Manages transitions to and from VMM via a hypercall

• **Shim Memory** protects application
  ‣ CTC protects control registers

• **Uncloaked Shim**
  ‣ Neutral ground
  ‣ Trampoline!
Loading Applications

- The Shim uses a **Loader** program
- Sets up the cloaked memory with a hypercall
- The loader / shim must be trusted
  - Metadata on the CTC checks for compromise
  - Here is the **meat** of the problem
    - Is it even used?
- Propagate shims to spawned applications
Its not that easy...

- Lot of OS interfaces that must be handled
- Faults / Interrupts
- System Calls
  - Pass control to the VMM
  - The shim catches this and stores registers
  - Clear the registers to prevent side channels
Complex Syscalls

- Some syscalls are easy
  - No *side effects*
  - Nice, getpid, sync

- Others, less so...
  - Pipe, r/w (Zero data)
  - Clone
  - Fork
  - Signal Handling
Performance

• Microbenchmarks
  ‣ Not so hot 15-60%
  ‣ Although a lot better than Proxos

• Application Benchmarks
  ‣ SPEC isn’t so bad
  ‣ High bandwidth hits some bottlenecks
  ‣ Why?
Take Away

• VM Systems provide isolation
  ‣ Between OSes/apps that may be untrusted

• VM Systems enable a small TCB
  ‣ Type I VMMs
  ‣ A1-Assured, like VAX VMM

• VM Systems can mediate inter-VM actions
  ‣ Virtualized operations
  ‣ Inter-VM operations
Take Away

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

• OS provides services used by applications
  ‣ Access to devices demultiplexed among VMs

• Can we use VM isolation to prevent compromise of applications by OS compromise?
  ‣ Proxos: use a “trusted” OS and redirect service requests
  ‣ Overshadow: use OS as untrusted communication media
Trusted VMs

- VMware and NetTop assume that the privileged VM (there is only one in these systems) prevents information flow (like a kernel)

- Thus, the only information flows between VMs are via networking
  - Privileged VM controls inter-VM communication via networking

- sHype controls IPC and networking at hypervisor level
  - Privileged VM uses hypervisor as policy store