Advanced Systems Security: VMs and Untrusted OSes

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

March 23, 2010
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
Options

- Hardware
  - XOM: Still need to configure software services

- Secure co-processors
  - IBM 4758: Extremely specialized and limited devices
Splitting Interfaces

Key Insight

• Intuition:
  — The TCB of an application = Code that it executes +
    everything that this code depends on (library
    functions, system calls)

• Not necessarily the case

• Eg. Concept of “trusted wrappers”
  — Wrap untrusted calls with some functionality that
    detects any malicious attempt
  — Eg: Wrap HTTP by SSL
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

Private VM
- Private OS Methods
- Private Application

Commodity OS VM
- Other Applications
- Host Process
- Commodity OS Kernel

Proxos
- Trusted system calls are routed to private OS methods
- Untrusted system calls are routed to the commodity OS kernel

VMM
- Private apps can interact with other apps via the host process
- All apps can access commodity OS resources
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

```plaintext
# 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

- **Associates VMID with PID of host for identifying future syscalls**
Proxos Implementation
Proxos SSH Server

Diagram:

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

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

- Network
Compare to Privilege Separation

Partitioning Interfaces to Resources

Partitioning Code (Provos et al)

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>
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?
A solution should...

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

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

- **Have an Incremental Path to Higher Assurance**
  - ...also.
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 the awesome 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?
Tasks of the Overshadow

- Context Identification
- Secure Control Transfer
- System Call Adaptation
- Mapping Cloaked Resources
- Managing Protection Metadata
Shim baby Shim

• 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

- Application Benchmarks
  - SPEC isn’t so bad
  - High bandwidth hits some bottlenecks
  - Why?
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