Keeping an Eye on Virtualization: VM Monitoring Techniques & Applications

Hardware Invariant-Based Techniques
Plan

• Passive VM Monitoring Example
• Literature
  – Antfarm/Lycosid
• Hypertap
Traditional VM Monitoring

Out-of-VM monitor manipulated by an in-VM attacker!

😊 Places trust on guest Operating System Properties
Antfarm

• HW-based Address Space ID (ASID) for each process

• Use ASID for out-of-VM process tracking

## Antfarm

<table>
<thead>
<tr>
<th></th>
<th>x86</th>
<th>SPARC</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASID</td>
<td>Page directory PA</td>
<td>Context ID</td>
</tr>
<tr>
<td>Creation</td>
<td>New ASID</td>
<td>New ASID</td>
</tr>
<tr>
<td>Exit</td>
<td>No user mappings and TLB flushed</td>
<td>Context demap</td>
</tr>
<tr>
<td>Context switch</td>
<td>CR3 change</td>
<td>Context ID change</td>
</tr>
</tbody>
</table>
Antfarm - Uses

- ASID implies process state
- Used in anticipatory disk scheduler
  - Disk seek time >> computation time
  - Add delay to r/w to avoid long seeks
- Lycosid - Can find hidden rootkits

Hardware Invariant Approach

Virtual Machine

Process list

1 → 2 → 3

Context switch to process 1

(Rootkit)

load cr3, p1 → Force VM Exit

Hardware

Hypervisor

Monitor

Inspect running processes

Inspect process 1

Notify VM Exit
Hardware Invariant Approach

Guaranteed to expose every hidden process

Context switch to process 2

(Rootkit)

load cr3, p2 → Force VM Exit

Notify VM Exit

Inspect running processes

Inspect process 2

✓ Fundamental OS operations ↔ hardware state

x86 with Hardware Assisted Virtualization (HAV) enabled. CR3 holds the Page Directory Base Pointer (PDBP) of running process.
VM Monitoring via HW Invariants

<table>
<thead>
<tr>
<th>Event</th>
<th>Hardware* Invariants (x86)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Context switch</td>
<td>MMU, CR3 access</td>
</tr>
<tr>
<td>Thread/task switch</td>
<td>Page protection, TSS</td>
</tr>
<tr>
<td>System call</td>
<td>MSR, Exception</td>
</tr>
<tr>
<td>IO access</td>
<td>IO instructions, Interrupts</td>
</tr>
<tr>
<td>Memory access</td>
<td>Page protection, Exception</td>
</tr>
</tbody>
</table>

Basis to support a wide range of failure & attack detections
Hidden Rootkit Detection

- Real world rootkits on Windows and Linux
- All rootkits detected

<table>
<thead>
<tr>
<th>Rootkit</th>
<th>Target OS</th>
<th>Hiding techniques</th>
</tr>
</thead>
<tbody>
<tr>
<td>FU</td>
<td>Windows XP, Vista</td>
<td>DKOM</td>
</tr>
<tr>
<td>HideProc</td>
<td>Windows XP, Vista</td>
<td>Hijack system calls</td>
</tr>
<tr>
<td>AFX</td>
<td>Windows XP</td>
<td>Hijack system calls</td>
</tr>
<tr>
<td>HideToolz</td>
<td>Windows Vista, 7</td>
<td>Hijack system calls</td>
</tr>
<tr>
<td>HE4Hook</td>
<td>Windows XP</td>
<td>Hijack system calls</td>
</tr>
<tr>
<td>BH</td>
<td>Windows XP</td>
<td>...</td>
</tr>
<tr>
<td>Enyelkm 1.2</td>
<td>Linux kernel 2.6</td>
<td>Kmem, dkom</td>
</tr>
<tr>
<td>SucKIT</td>
<td>Linux kernel 2.6</td>
<td></td>
</tr>
<tr>
<td>PhalanX</td>
<td>Linux kernel 2.6</td>
<td>DKOM</td>
</tr>
</tbody>
</table>

- Independent of hiding mechanism
HRKD Demo

• Windows XP Guest
• Keylogger hidden by rootkit
Privilege Escalation Detection (PED)

• Privilege Escalation Attack

• Detection
Ninja

- Escalation if root process has a non-root parent
- Whitelist for setuid programs (e.g., sudo)
- http://forkbomb.org/ninja/
# The Three Ninjas

Privilege Escalation Detection (PED)

<table>
<thead>
<tr>
<th>Ninja</th>
<th>Location</th>
<th>Description</th>
<th>Monitoring</th>
</tr>
</thead>
<tbody>
<tr>
<td>O-Ninja</td>
<td>In-VM</td>
<td>Original Ninja</td>
<td>Polling</td>
</tr>
<tr>
<td>H-Ninja</td>
<td>Out-of-VM</td>
<td>Uses OS invariants</td>
<td>Polling</td>
</tr>
<tr>
<td>HT-Ninja</td>
<td>Out-of-VM</td>
<td>Rooted in HW invariants (HyperTap)</td>
<td>Event-driven</td>
</tr>
</tbody>
</table>

**HT-Ninja** checks at *process switches and IO system calls*
Ninjas vs. transient attacks

O-Ninja and H-Ninja: vulnerable to transient attacks

Side channel:
O-Ninja’s interval from /proc

Spamming:
Dummy processes

<table>
<thead>
<tr>
<th>Ninja’s Monitoring Interval (seconds)</th>
<th>Predicted Interval (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.00039</td>
</tr>
<tr>
<td>2</td>
<td>2.00023</td>
</tr>
<tr>
<td>4</td>
<td>4.00025</td>
</tr>
<tr>
<td>8</td>
<td>8.00025</td>
</tr>
</tbody>
</table>

HT-Ninja uses event-driven monitoring and is not vulnerable to transient attacks
Evaluation of GOSHID

• Guest OS hang detection

• Injected spinlock bugs* into Linux kernel
  – Four types of spinlock-related bugs
  – ~18,000 injections to 2-CPU VMs

Evaluation of GOSHD

- Overall hang detection coverage is 99.8%
- **Partial OS hangs are significant:**
  - 18% to 26% on non-pre-emptible and pre-emptible
  - Heartbeats are not effective for partial hangs
HyperTap Summary

• **Key design principles**
  – Use Hardware Architectural Invariants as root-of-trust
  – Support event-driven logging via VM exits
  – Unify reliability and security monitoring

• **Results**
  – Efficient continuous monitoring of VMs
  – Detected new failure mode & new type of attacks
VM-based Monitoring Comparison

<table>
<thead>
<tr>
<th>VMWare</th>
<th>LiveWire</th>
<th>HyperTap</th>
<th>Auto-generate Invariants</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>
Summary

• OS Invariants
• Antfarm/Lycosid
• Hidden Rootkit Detection
• HyperTap
• HyperTap Detectors – HRKD, GOSHID, PED

• Libvmi now supports events! (xen only)
  – https://github.com/libvmi/libvmi/blob/master/notes/events.txt
Case Studies & Evaluation

✓ Reliability & Security monitors
  ✓ Hidden Rootkit Detection (HRKD)
  ✓ Guest OS Hang Detection (GOSHD)
  ✓ Privilege Escalation Detection (PED)

✓ Performance Evaluation
  ✓ <2% overhead for CPU workloads
  ✓ <5% overhead for IO workloads
HyperTap-based VM Auditing

Only use guest OS state that can be derived directly from HW State
Evaluation of GOSHD: Detection Latency

- Detection of partial hangs results in shorter detection latency
Performance Overhead

• Combined overhead < sum of individual overheads

• <2% overhead for CPU workloads

• <5% overhead for IO workloads

• Micro-benchmark:
  – Highest performance loss for NOOP system call (~19%)
Performance Overhead:
CPU-intensive Workloads

<2% overhead for CPU-intensive workloads

(95% Confidence Interval)
Performance Overhead:
IO-intensive Workloads

- HRKD
- HT-Ninja
- GOSHD
- HRKD+HT-Ninja+GOSHD

~5% overhead for IO-intensive workloads

(95% Confidence Interval)
Performance Overhead: Micro-benchmark

Combined overhead < sum of individual overheads

~10% context switch overhead

Highest performance loss for NOOP system call (~19%)
Three Ninjas against transient attacks

- Transient attacks against polling monitoring

- Propose three transient attacks against Ninja
  - Side channel attack to determine monitoring intervals
  - Spamming attack to increase vulnerable intervals
  - Rootkit combined attack to become persistent
Event Driven Monitoring via VM Exits

VM Exit Events

Auditors

Privilege Escalation Detection
Rootkit Detection
OS Hang Detection

HyperTap

Thread switch events ✓ ✓ ✓
Process switch events ✓ ✓ ✓
IO syscall events ✓ ✓ ✓