HyperSafe:
A Lightweight Approach to Provide Lifetime Hypervisor Control-Flow Integrity

Zhi Wang, Xuxian Jiang
North Carolina State University
Virtualization Adoption

- Rapidly growing in industry
  - 16% server workloads on virtual machines now
  - 50% by 2012

- Widely applied to security problems
  - Guest integrity monitoring
    - ReVirt (Dunlap et al, OSDI ‘02), Livewire (Garfinkel et al, NDSS ‘03), VMwatcher (Jiang et al, CCS ‘07), Lares (Payne et al, Oakland ‘08), SIM (Sharif et al, CCS ‘09)...
  - Guest integrity protection
    - SecVisor (Seshadri et al, SOSP ‘07), NICKLE (Riley et al, RAID ‘08), HookSafe (Wang et al, CCS ‘09)...
  - System software analysis
    - AfterSight (Chow et al, USENIX ATC ’08), K-Tracer (Lanzi et al, NDSS ‘09), PoKeR (Riley et al, EuroSys ‘09) ...
  - ...

1: Gartner Symposium/ITxpo 2009
Common Assumption

A Trustworthy Hypervisor!
Bloated TCB of Type I Hypervisors

<table>
<thead>
<tr>
<th>Hypervisor</th>
<th>Hypervisor SLOC</th>
<th>TCB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Xen-4.0</td>
<td>194K</td>
<td>Xen, Dom0</td>
</tr>
<tr>
<td>VMware ESXi(^1)</td>
<td>200K</td>
<td>VM Kernel</td>
</tr>
<tr>
<td>Hyper-V(^1)</td>
<td>100K</td>
<td>Hyper-V, Windows 2008 Server</td>
</tr>
<tr>
<td>BitVisor</td>
<td>194K</td>
<td>BitVisor</td>
</tr>
</tbody>
</table>

1. NOVA: A Microhypervisor-Based Secure Virtualization Architecture (Udo Steinberg et al, EuroSys ‘10)
Vulnerabilities & Attacks

- Common Vulnerabilities and Exposures (CVE)
  - Xen - 26, VMware ESX - 18 (til 11/2009)

- VM escape attacks
  - Xen Owning Trilogy (Invisible Things Lab, Blackhat ‘08)
  - Cloudburst: A VMware Guest to Host Escape (Kostya Kortchinsky, Blackhat ‘09)

- Hypervisor based rootkits
  - SubVirt (King et al, Oakland ‘06), Blue Pill (Invisible Things Lab, Blackhat ‘06), Virtiol (Dino A. Dai Zovi, Blackhat ‘06)
Existing Solutions

- Reduce TCB
  - TrustVisor (McCune et al, Oakland ‘10), NOVA (Steinberg et al, EuroSys ‘10), Improving Xen Security through Disaggregation (Murray et al, VEE ‘08), ...

- Formal verification
  - seL4 (Klein et al, SOSP ‘09), ...

Our goal is to enable **self-protection** of commodity type-I (bare-metal) hypervisors!
Outline

- Motivation
- Design
- Implementation & Evaluation
- Related Work
- Summary
Assumptions

- Trustworthy (x86) hardware
  - IOMMU to prevent malicious DMA transactions
  - Trusted System Management Mode (SMM)
- Software bugs in the hypervisor
Our Approach: HyperSafe

- lifetime hypervisor CFI
- load-time integrity
- runtime CFI
- code integrity
- control data integrity

trusted booting (e.g. tboot)
non-bypassable memory lockdown
restricted pointer indexing
Non-bypassable Memory Lockdown
**x86 Paging Mode**

- Page tables determine memory properties
  - Permissions in a page table entry:
    - NX  – Non-executable
    - R/W – Read-only or Writable
    - U/S  – User or supervisor page
  - \( W \oplus X \): a page can be either writable or executable, but not both

- All memory accesses by software are translated and controlled by page tables
  - Including reads/writes of page tables

<table>
<thead>
<tr>
<th></th>
<th>N</th>
<th>R</th>
<th>W</th>
<th>U</th>
</tr>
</thead>
<tbody>
<tr>
<td>NX</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R/W</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>U/S</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
HyperSafe’s Memory Lockdown

- Pitfalls in existing $W \oplus X$
  - Mixed code and data
    - Mixed code and data are prohibited
  - Double mapping with conflicting attributes
    - Double mapping must have conforming attributes
  - Writable page tables
    - Read-only page tables

**No code** can modify the write-protected hypervisor code and data!
Challenge

How to safely allow benign page table updates???
Write-protect (WP) bit in CR0 controls interaction of supervisor and read-only pages
- WP = 1: Read-only pages are protected even from supervisor
- WP = 0: Supervisor can write into read-only pages
Benign Page Table Updates

- WP = 1 by default to lock down memory
- Update page table **atomically**
  1. Disable interrupt
  2. WP = 0
  3. Verify proposed change
  4. Update **read-only** page table
  5. WP = 1
  6. Enable interrupt

![Diagram of Read-only Page Tables]
Restricted Pointer Indexing (RPI)
Control Flow Integrity (CFI)

- CFI: runtime execution paths must follow control flow graph (CFG)
- CFG may have different granularities

**Indirect call may go to:**

- All indirectly called functions
- Functions with same type
- Points-to set

**Granularities:**

- Coarse-grained
- Fine-grained
Points-to analysis required

Manual analysis to handle domain knowledge / assembly code in prototype

- e.g. assembly code to access per-cpu data (function pointers) in gs segments
Enforce Control Flow Integrity

- Restricted Pointer Indexing
  - Collect control data into tables (protected by memory lockdown)
  - Replace control data with the indexes to the table
  - Convert the index back to transfer control

Only legitimate control data in the table can be used for control flow transfer!
Outline

- Motivation
- Design
- Implementation & Evaluation
- Related Work
- Summary
Implementation

- Implementing techniques:
  - Memory lockdown: modify hypervisor’s memory management code
  - Restricted Pointer Indexing: extend LLVM compiler to instrument related instructions

- Prototypes of HyperSafe:
  - Full support for BitVisor
  - Partial support for Xen, additional engineering needed
Security Analysis

- Disable WP bit
  - Misuse page table update function ← RPI

- Subvert page table
  - Misuse page table update function ← RPI
  - Map hypervisor memory to a compromised guest VM ← Memory lockdown

- Return-oriented programming ← Memory lockdown, RPI
HS-2 implements coarse-grained RPI with two target tables (return instructions and indirect calls)

HS-m implements fine-grained RPI with one target table per function and indirect call
Performance: LMbench

Normalized LMbench Overhead Compared to Original BitVisor

<table>
<thead>
<tr>
<th>Operation</th>
<th>HS-2</th>
<th>HS-m</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctx</td>
<td>5.5%</td>
<td>0.6%</td>
</tr>
<tr>
<td>stat</td>
<td>1.2%</td>
<td>0.0%</td>
</tr>
<tr>
<td>mmap</td>
<td>-2.4%</td>
<td>0.6%</td>
</tr>
<tr>
<td>sh proc</td>
<td>-2.2%</td>
<td>3.4%</td>
</tr>
<tr>
<td>10K file</td>
<td>4.5%</td>
<td>3.5%</td>
</tr>
<tr>
<td>bcopy</td>
<td>5.4%</td>
<td>2.8%</td>
</tr>
</tbody>
</table>
Related Work

- **Program Analysis and Formal Proof**
  - seL4 (Klein et al, SOSP ‘09), WIT (Akritidis et al, SOSP ‘08), KLEE (Cadar et al, OSDI ‘08), ...

- **Guest Integrity Monitoring or Protection**
  - SIM (Sharif et al, CCS ‘09), SecVisor (Seshadri et al, SOSP ‘07), SBCFI (Petroni et al, CCS ‘07), ...

- **Trusted Computing**
  - TrustVisor (McCune et al, Oakland ‘10), Flicker (McCune et al, EuroSys ‘08), Pioneer (Seshadri et al, SOSP ’05), ...
HyperSafe is a lightweight approach to provide lifetime control-flow integrity for commodity Type-I hypervisors.

- lifetime hypervisor CFI
- runtime CFI
- load-time integrity
- code integrity
- control data integrity
- trusted booting (e.g. tboot)
- non-bypassable memory lockdown
- restricted pointer indexing
Thanks, Questions?