SecPod: A Framework for Virtualization-based Security Systems

Xiaoguang Wang[†♮], Yue Chen[†], Zhi Wang[†], Yong Qi[♮], Yajin Zhou[‡]

Florida State University[†] Xi'an Jiaotong University[♮] Qihoo 360[‡]

Outline

1. Motivation
2. SecPod Design
3. Implementation
4. Evaluation
5. Related Work

Page Table Integrity

Kernel protection requires page table integrity

- Page tables decide address translation (from VA to PA)

Page Table Integrity

Kernel protection requires page table integrity

- Page tables decide address translation (from VA to PA)
- Page tables control memory protection

Page Table Integrity

Kernel protection requires page table integrity

- Page tables decide address translation (from VA to PA)
- Page tables control memory protection
- e.g. Data Execution Prevention (Write $\oplus$ eXecute)

Page Table Integrity

Kernel protection requires page table integrity

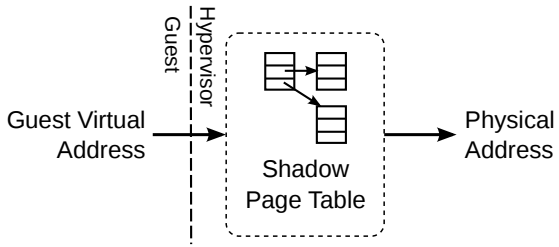
- Page tables decide address translation (from VA to PA)
- Page tables control memory protection
- e.g. Data Execution Prevention (Write $\oplus$ eXecute)

However, page tables are always writable in the kernel

- Kernel needs to frequently change memory mapping
- Kernel protection can be subverted by manipulating page tables

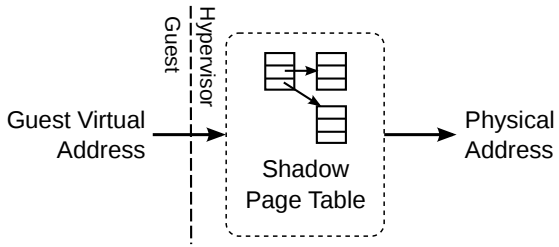
Virtualization-based Kernel Protection

- Security tools are isolated "out-of-the-box", but need to intercept key guest events
  - e.g., guest page table updates, control-register updates



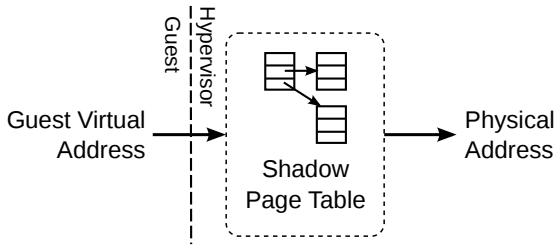
Virtualization-based Kernel Protection

- Security tools are isolated "out-of-the-box", but need to intercept key guest events
  - e.g., guest page table updates, control-register updates
- **Shadow paging** enables reliable kernel memory protection
  - Hypervisor uses shadow paging to virtualize memory



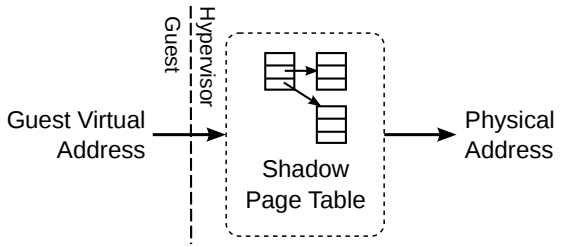
Virtualization-based Kernel Protection

- Security tools are isolated "out-of-the-box", but need to intercept key guest events
  - e.g., guest page table updates, control-register updates
- **Shadow paging** enables reliable kernel memory protection
  - Hypervisor uses shadow paging to virtualize memory
  - SPTs are synchronized with GPTs by the hypervisor
    →security tools can intercept guest page table updates



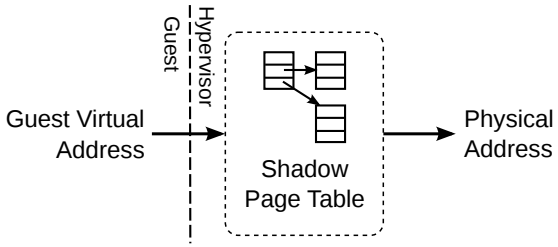
Virtualization-based Kernel Protection

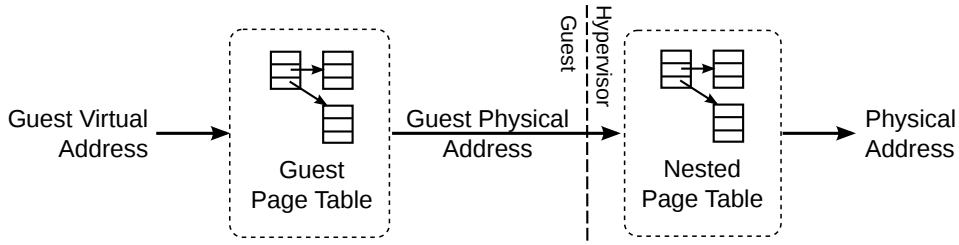
- Security tools are isolated "out-of-the-box", but need to intercept key guest events
  - e.g., guest page table updates, control-register updates
- **Shadow paging** enables reliable kernel memory protection
  - Hypervisor uses shadow paging to virtualize memory
  - SPTs are synchronized with GPTs by the hypervisor
    →security tools can intercept guest page table updates
  - SPTs supersede GPTs for address translation



Virtualization Hardware Obsoletes Shadow Paging

- **Nested paging** introduces two-level address translation for VMs
  - Both GPT and NPT are used by CPU for address translation

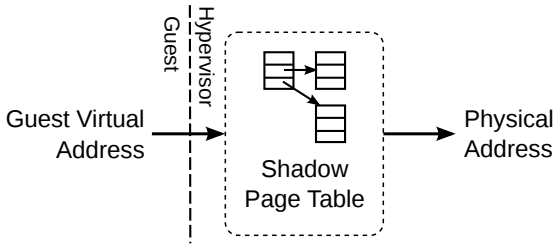


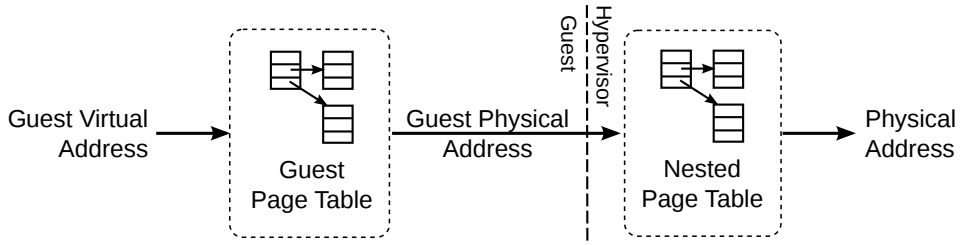


[1] VMware: performance evaluation of Intel EPT hardware assist.

Virtualization Hardware Obsoletes Shadow Paging

- **Nested paging** introduces two-level address translation for VMs
  - ▷ Both GPT and NPT are used by CPU for address translation
- Nested paging has big performance advantage over SPT
  - ▷ An acceleration of up to 48% for MMU-intensive tasks [1]

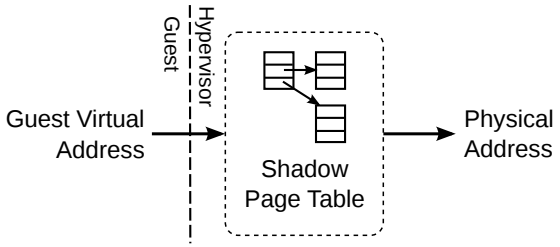


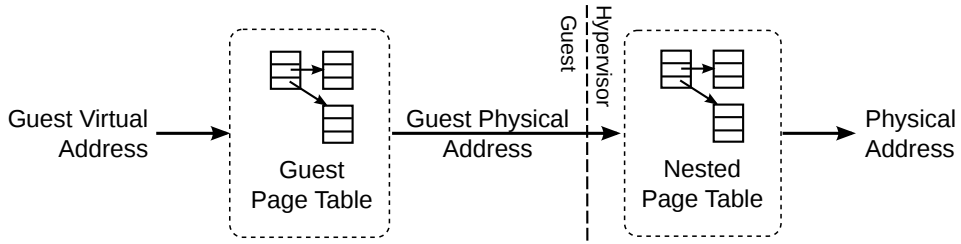


[1] VMware: performance evaluation of Intel EPT hardware assist.

Virtualization Hardware Obsoletes Shadow Paging

- **Nested paging** introduces two-level address translation for VMs
  - Both GPT and NPT are used by CPU for address translation
- Nested paging has big performance advantage over SPT
  - An acceleration of up to 48% for MMU-intensive tasks [1]
- Security tools cannot intercept guest memory updates with NPT
  - Guest is free to change its GPTs, without notifying hypervisor





[1] VMware: performance evaluation of Intel EPT hardware assist.

Why SecPod

Our Goal:

A framework for virtualization-based security tools on the modern virtualization hardware with **nested paging**

Why SecPod

Our Goal:

A framework for virtualization-based security tools on the modern virtualization hardware with **nested paging**

Our Solution:

SecPod: A Framework for Virtualization-based Security Systems

Threat Model and Assumption

- Trustworthy hardware and trusted booting
  - Load-time integrity is protected by trusted booting
  - IOMMU is properly configured to prevent DMA attacks

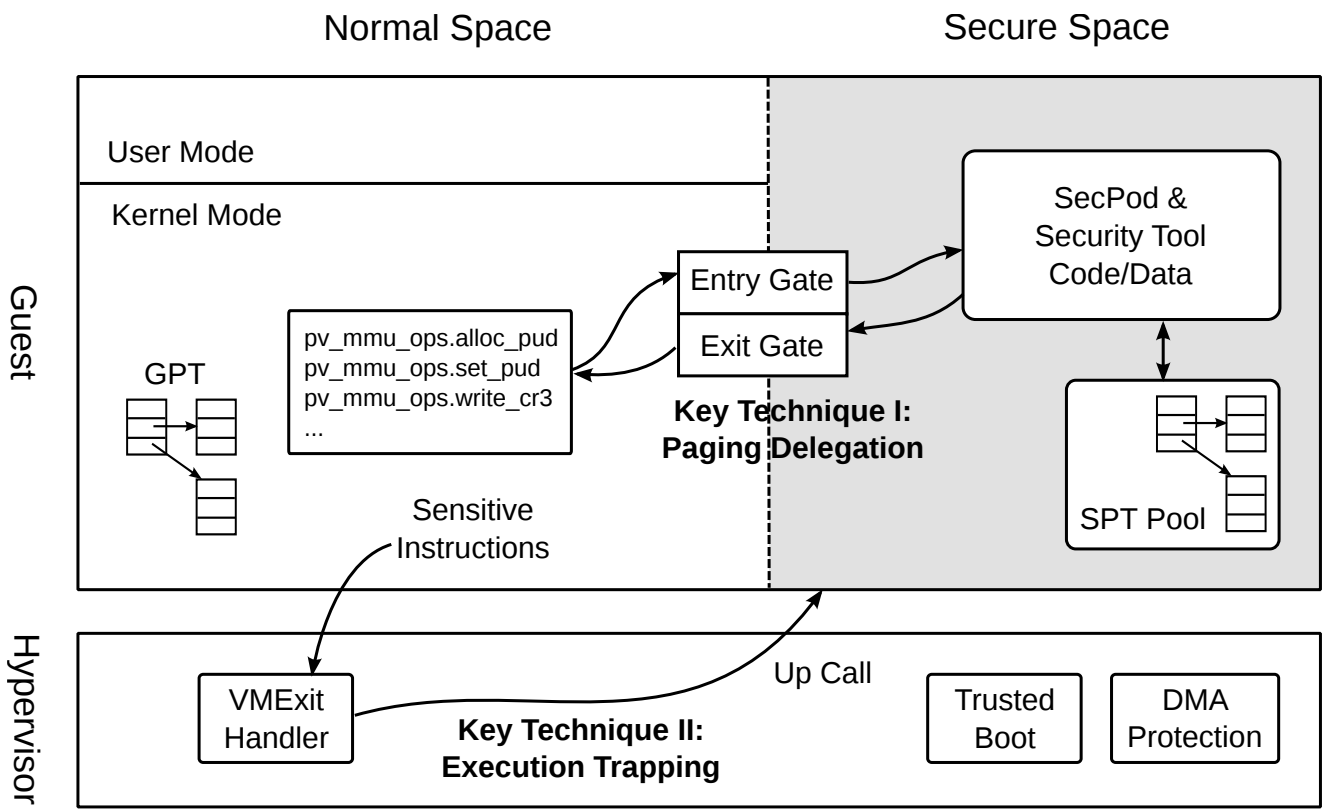
Threat Model and Assumption

- Trustworthy hardware and trusted booting
  - ▷ Load-time integrity is protected by trusted booting
  - ▷ IOMMU is properly configured to prevent DMA attacks
- Hypervisor is trusted
  - ▷ Formal verification [seL4, SOSP'09], integrity protection and monitoring [HyperSafe, S&P'10]

Threat Model and Assumption

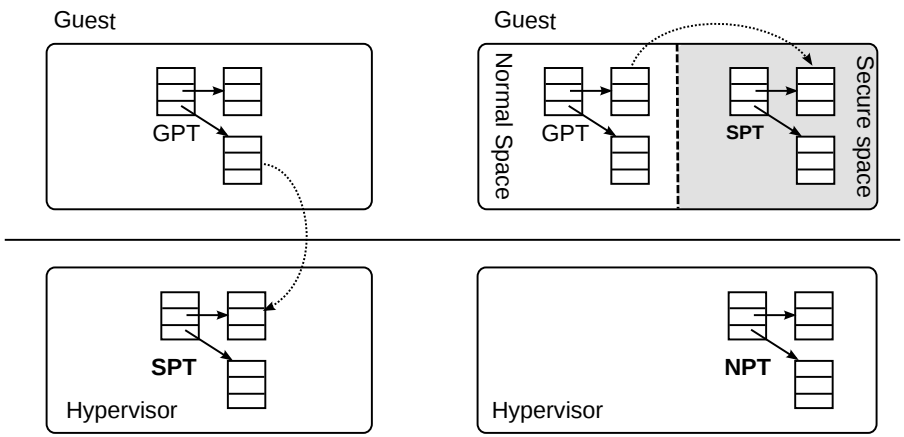
- Trustworthy hardware and trusted booting
  - ▷ Load-time integrity is protected by trusted booting
  - ▷ IOMMU is properly configured to prevent DMA attacks
- Hypervisor is trusted
  - ▷ Formal verification [seL4, SOSP'09], integrity protection and monitoring [HyperSafe, S&P'10]
- Kernel is benign but contains vulnerabilities
  - ▷ Powerful attackers can change arbitrary memory of the kernel

SecPod Architecture



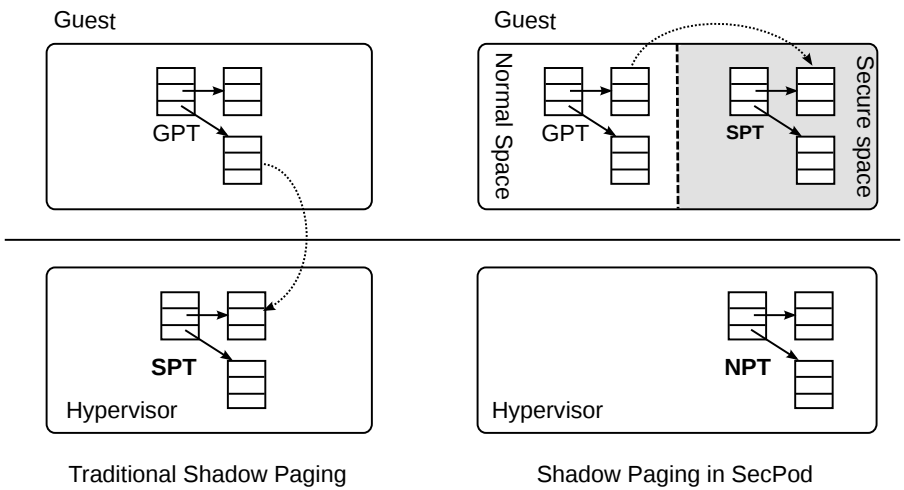
Key Technique I: Paging Delegation

- SecPod creates an isolated address space from the kernel



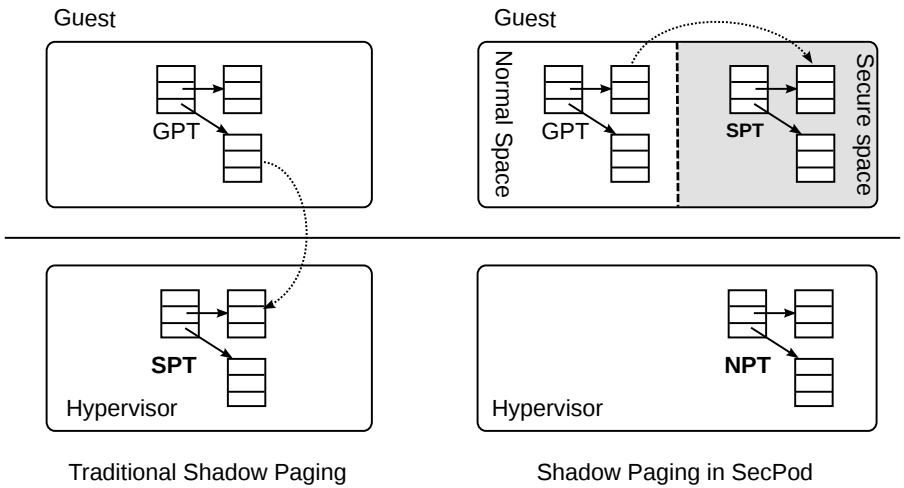
Key Technique I: Paging Delegation

- SecPod creates an isolated address space from the kernel
- Secure space maintains SPTs for the guest
  - SPTs are the only effective page tables for the guest
  - SPTs mirror GPTs (if no memory protection violation)



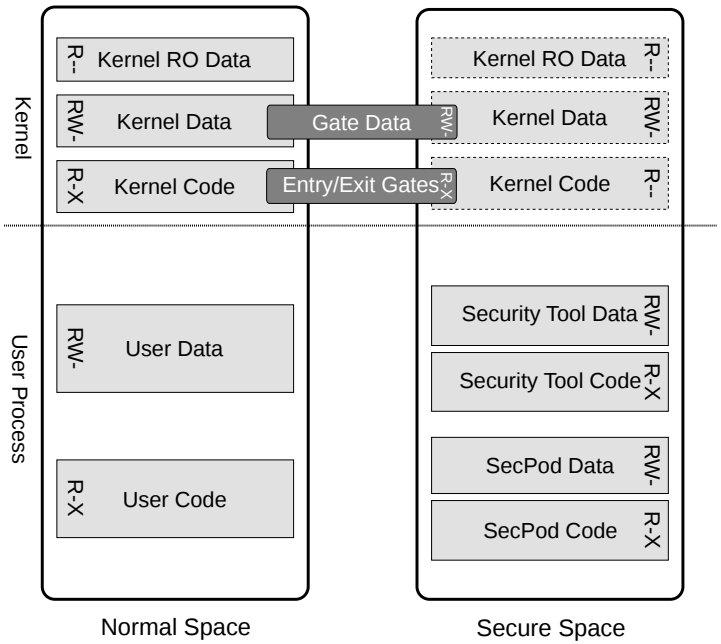
Key Technique I: Paging Delegation

- SecPod creates an isolated address space from the kernel
- Secure space maintains SPTs for the guest
  - SPTs are the only effective page tables for the guest
  - SPTs mirror GPTs (if no memory protection violation)
- SecPod forwards guest page table updates to secure space



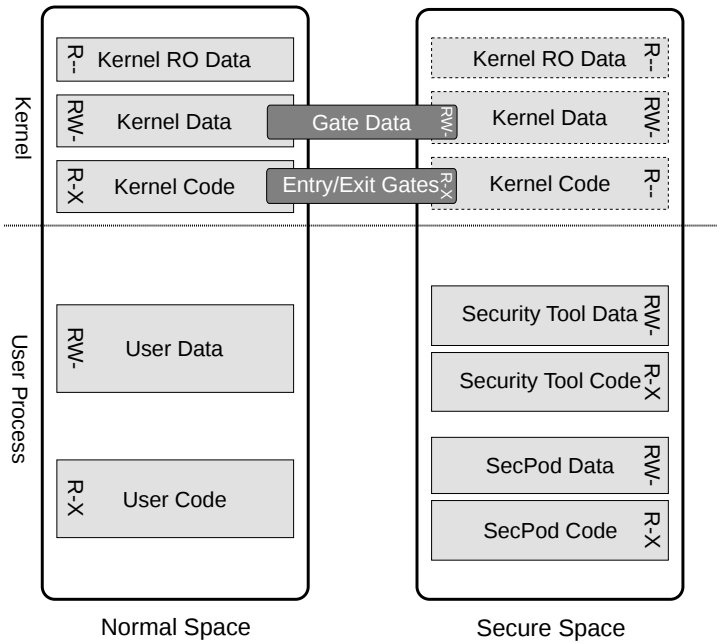
SecPod Address Space Layout

- Normal/secure spaces use page-table based isolation
  - Entry/exit gates are the only passage



SecPod Address Space Layout

- Normal/secure spaces use page-table based isolation
  - ▷ Entry/exit gates are the only passage
- Guest kernel is mapped in secure space
  - ▷ Security tools can access guest memory, but not execute it



Protecting Secure Space

Attacker might try to:

- Enter secure space without security checks
- Request malicious page table updates
  - e.g., to map secure memory in guest
- Misuse privileged instructions
  - e.g., to load a malicious page table, to disable paging...

Protecting Secure Space

Attacker might try to:

- Enter secure space without security checks
- Request malicious page table updates
  - e.g., to map secure memory in guest
- Misuse privileged instructions
  - e.g., to load a malicious page table, to disable paging...

Our countermeasures:

- Secure and efficient context switch
- Page table update validation
- Execution trapping of privileged instructions

Secure and Efficient Context Switch

- Entry/exit gates are only passage between secure/normal spaces
  - Each gate switches the page table, the stack...
  - Entry gate runs atomically by disabling interrupts (SIM [CCS '09])

Secure and Efficient Context Switch

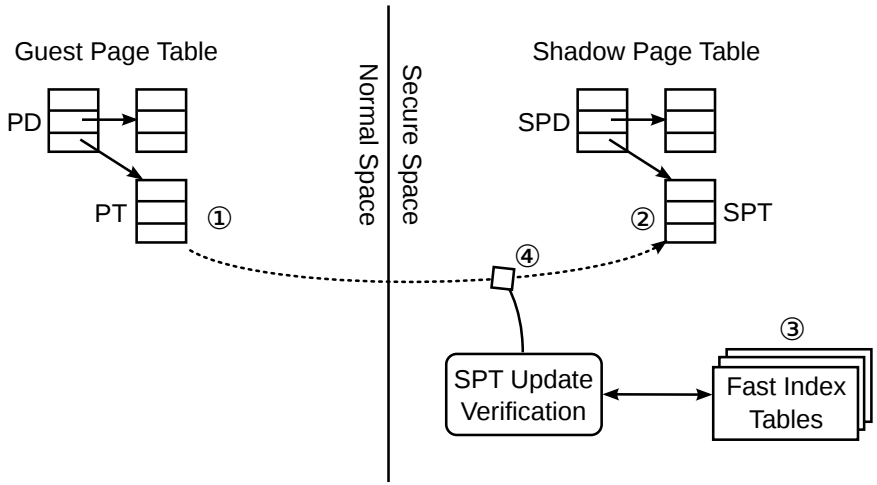
- Entry/exit gates are only passage between secure/normal spaces
  - Each gate switches the page table, the stack...
  - Entry gate runs atomically by disabling interrupts (SIM [CCS '09])
- Loading `CR3` is a privileged instruction trapped by SecPod
  - Performance overhead is high if each context switch is trapped

Secure and Efficient Context Switch

- Entry/exit gates are only passage between secure/normal spaces
  - Each gate switches the page table, the stack...
  - Entry gate runs atomically by disabling interrupts (SIM [CCS '09])
- Loading `CR3` is a privileged instruction trapped by SecPod
  - Performance overhead is high if each context switch is trapped
- Intel CR3 target list to the rescue:
  - Four page tables can be loaded without being trapped by CPU
  - There are many SPTs for the guest
    → use a fixed top-Level page table for all SPTs

Page Table Update Validation

- SecPod enforces basic kernel memory integrity for guest
  - No mapping is allowed to the secure space code/data
  - Enforce kernel W⊕X



Key Technique II: Execute Trapping

- SecPod traps malicious privileged instructions executed by guest
  - ▷ It can trap intended and unintended[2] privileged instructions
- Hypervisor notifies secure space trapped instructions via upcalls
  - ▷ Similar to signal delivery in the traditional OS

[2] X86 has variable-length instructions, unintended instructions can be "created" by jumping to the middle of an instruction.

Trapped Sensitive Instructions

Instruction	Semantics
`LGDT`	Load global descriptor table
`LLDT`	load local descriptor table
`LIDT`	load interrupt descriptor table
`LMSW`	load machine status word
`MOV to CR0`	write to `CR0`
`MOV to CR4`	write to `CR4`
`MOV to CR8`	write to `CR8`
`MOV to CR3`	load a new page table
`WRMSR`	write machine-specific registers

Implementation

- Paging delegation
  - Leverage Linux paravirtualization interface: `pv_mmu_ops`
- Execution trapping implemented in the Hypervisor (KVM)

Implementation

- Paging delegation
  - ▷ Leverage Linux paravirtualization interface: `pv_mmu_ops`
- Execution trapping implemented in the Hypervisor (KVM)
- Security tools:
  - ▷ Compiled as ELF libraries and loaded into secure space
  - ▷ Implemented an example tool to prevent unauthorized kernel code from execution (Patagonix [USENIX Sec '08], NICKLE [RAID '08])

Security Analysis

- Maliciously modify secure space memory
  - Secure space memory is not mapped in the normal space
    $\rightarrow$ try to map the secure space memory in the guest

Security Analysis

- Maliciously modify secure space memory
  - Secure space memory is not mapped in the normal space
    $\longrightarrow$ try to map the secure space memory in the guest
  - Directly change the page mapping
  - Ask SecPod to map secure memory

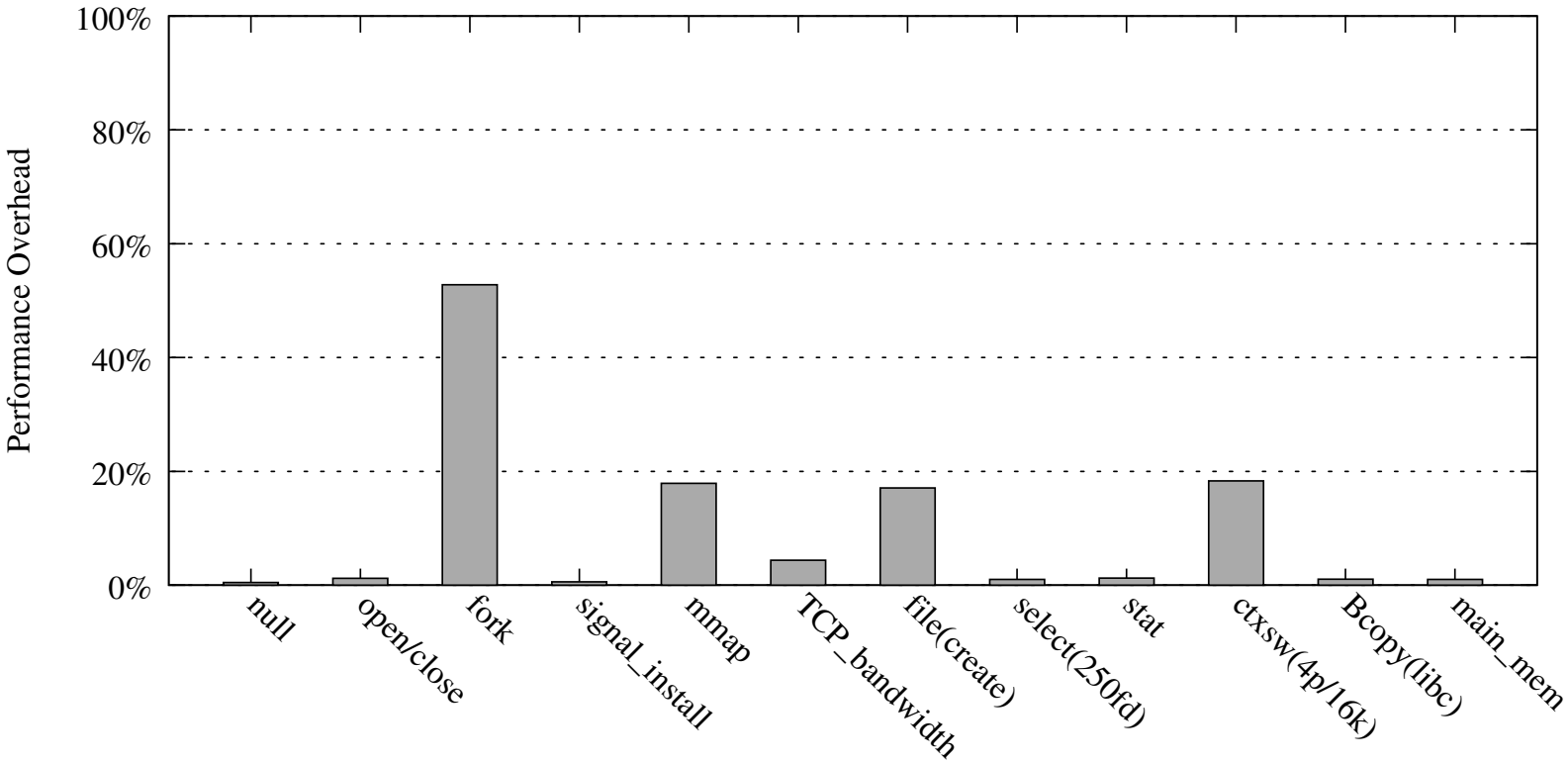
Security Analysis

- Maliciously modify secure space memory
  - Secure space memory is not mapped in the normal space
    → try to map the secure space memory in the guest
  - Directly change the page mapping ← SPT is isolated
  - Ask SecPod to map secure memory ← SPT update validation

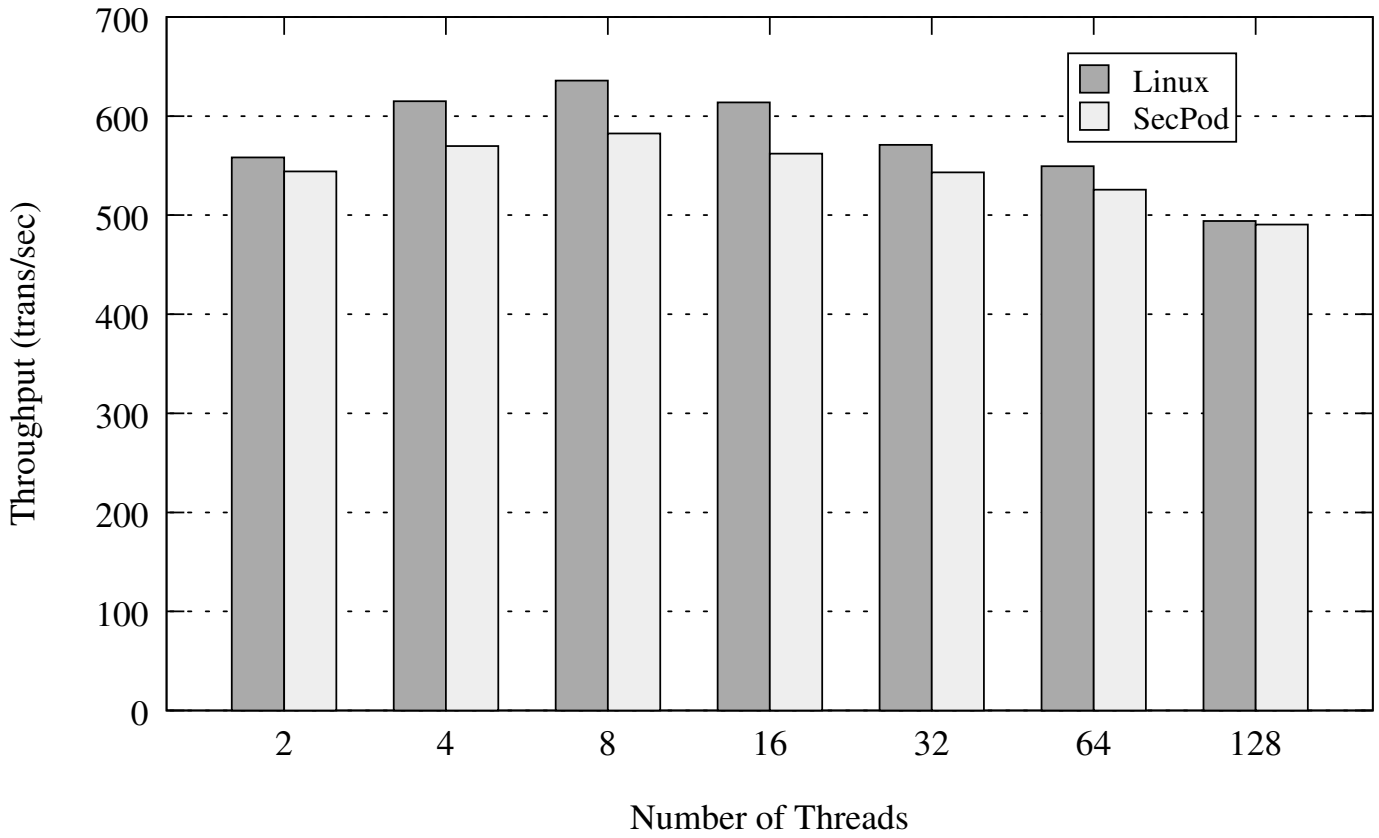
Security Analysis

- Maliciously modify secure space memory
  - Secure space memory is not mapped in the normal space
    → try to map the secure space memory in the guest
  - Directly change the page mapping ← SPT is isolated
  - Ask SecPod to map secure memory ← SPT update validation
- Misuse privileged instructions
  - Privileged instructions by guest are trapped and verified

Performance Evaluation: LMBench



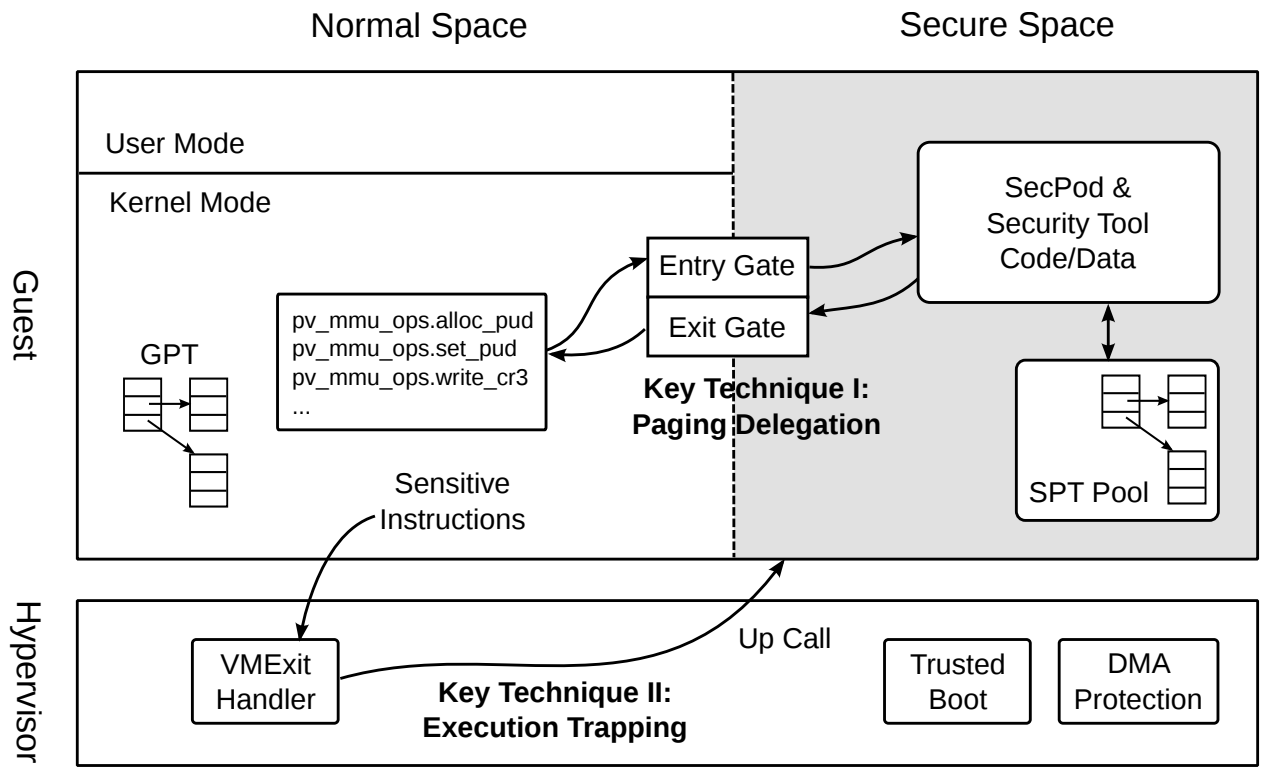
Performance Evaluation: SysBench OLTP



Related Work

- Virtualization-based security
  - Malware analysis: Ether[CCS'08]
  - Rootkit detection and prevention: PoKeR[EuroSys'09]
  - Virtual machine introspection: Virtuoso[S&P'11], SIM[CCS'09]
- Kernel/user application security
  - Exploit mitigation techniques: ASLR, DEP, CFI[CCS'07]
  - Kernel/hypervisor memory integrity: TZ-RKP[CCS'14], HyperSafe[S&P'10], Nested Kernel[ASPLOS'15]

Summary



- SecPod: A Framework for Virtualization-based Security Systems

Thank you & Questions?