SecVisor: A Tiny Hypervisor for Lifetime Kernel Code Integrity

Arvind Seshadri, Mark Luk, Ning Qu, Adrian Perrig
Carnegie Mellon University
**Motivation**

- **Kernel rootkits**
  - Malware inserted into OS kernels
Motivation

- Kernels increasingly vulnerable
  - Increasing code sizes
  - New attack methods
    - DMA-based attacks

- Current security tools insufficient
  - Assume kernel integrity
  - Detection-based
    - Cannot find all attacks
Objective

- Security hypervisor that
  - Prevents attacker injected code from executing at kernel privilege
  - Permits only user-approved code to execute at kernel privilege
    - User can specify approval policy

- Design goals
  - Security
  - Ease of porting commodity OS kernels
SecVisor

- Tiny (~1100 line runtime) hypervisor
- Enforce approved code execution in kernel mode
- Property holds over system lifetime
- Amenable to formal verification or manual audit
Attacker Model

- Attacker can perform all attacks except HW attacks against CPU and memory subsystem

- Examples
  - Employ malicious code to modify memory contents
  - Employ malicious peripherals to perform DMA writes
  - Modify system firmware (BIOS)

- Attacker can have knowledge of zero-day kernel exploits
Assumptions

- Single CPU
- CPU has hardware virtualization support
  - AMD SVM and Intel TXT (LT)
- OS kernel
  - Executes in 32-bit mode
  - Does not use self-modifying code
- SecVisor does not have any vulnerabilities
  - Amenable to formal verification or manual audit
Outline

- Introduction
- Conceptual Design
- Implementation
- Experiments and Results
- Related Work and Conclusion
Required Properties

- **Constrained Instruction Pointer (IP)**
  - IP should point within approved code regions as long as CPU executes in kernel mode

- **Approved code regions immutable**
  - Approved code regions cannot be modified by attacker
Constraining IP

- Each kernel mode entry sets IP within approved code regions.
- IP is within approved code regions as long as CPU is in kernel mode.
- Each kernel mode exit sets CPU privilege level to user mode.
Constraining IP

- Each kernel mode entry sets IP within approved code regions
- IP is within approved code regions as long as CPU is in kernel mode
- Each kernel mode exit sets CPU privilege level to user mode
Kernel Mode Entry

Check: All CPU entry pointers point to approved code
Constraining IP

- Each kernel mode entry sets IP within approved code regions
- IP is within approved code regions as long as CPU is in kernel mode
- Each kernel mode exit sets CPU privilege level to user mode
Kernel Mode Execution

- $W \oplus X$ protection over kernel memory
- Ensures that kernel data is not executable
- Additional steps needed…
Problem: Shared Address Space

- **Attack:** Attacker can execute application code with kernel privilege!
- **Solution:** Mark all app memory non-executable on kernel entry
- **Requires:** Intercept all user-to-kernel mode switches
Intercepting User-to-Kernel Switch

- All CPU entry pointers point to approved code
- Mark approved code regions non-executable during user mode execution
- All user-to-kernel switches throw exceptions
Constraining IP

- Each kernel mode entry sets IP within approved code regions
- IP is within approved code regions as long as CPU is in kernel mode
- Each kernel mode exit sets CPU privilege level to user mode
Kernel Mode Exit

- Requires: Intercept all kernel-to-user mode switch
- App memory non-executable in kernel mode
- Exception on mode switch from kernel to user
- Set privilege level of CPU to user mode by intercepting exception
Summary: Control Flow

- **Application (RWX)**
  - Approved Code (R)
  - Kernel Data (RW)

- **Application (RW)**
  - Approved Code (RX)
  - Kernel Data (RW)

- **User Mode**
  - Exception

- **Kernel Mode**
  - Modify Perm.

- **SecVisor**

- **Kernel mode entry**
Summary: Control Flow

- **Kernel Mode**
  - Kernel Data (RW)
  - Approved Code (RX)

- **User Mode**
  - Application (RWX)
  - Approved Code (R)
  - Kernel Data (RW)

- **SecVisor**
  - Exception
  - Modify Perm.
  - Kernel mode exit

- **Application**
  - (RW)
  - Approved Code (R)
  - Kernel Data (RW)
Required Properties

- **Constrained Instruction Pointer (IP)**
  - IP should point within approved code regions as long as CPU executes in kernel mode

- **Approved code regions immutable**
  - Approved code regions cannot be modified by attacker
Immutable Approved Code

- Memory regions can be written by:
  - SW executing on CPU
  - DMA writes by peripherals
- Memory protections mark approved code regions read-only
- IOMMU protection against DMA writes to approved code regions
Outline

- Introduction
- Conceptual Design
- Implementation
  - Setting memory protections
    - Intercept user↔kernel switches
    - Protect approved code from modification
  - Checking and protecting entry pointers
    - Constrains IP on kernel mode entry
- Experiments and Results
- Related Work and Conclusion
Setting Memory Protections

- Set memory permissions independent of OS
  - Virtualization is a convenient mechanism
- Virtualize physical memory to set permissions
  - SW virtualization: Shadow page tables
  - HW virtualization: Nested page tables
- AMD SVM-based implementation platform
  - Intel TXT can also be used
- DMA exclusion vector (DEV) for DMA-write protection
Setting Memory Protections

- Set memory permissions independent of OS
  - Virtualization is a convenient mechanism
- Virtualize physical memory to set permissions
  - SW virtualization: Shadow page tables
  - HW virtualization: Nested page tables
- AMD SVM-based implementation platform
  - Intel TXT can also be used
- DMA exclusion vector (DEV) for DMA-write protection
Memory Virtualization

 Requires CPU to support three kinds of address spaces
Shadow Page Tables (SPT)
Shadow Page Tables (SPT)
Shadow Page Tables (SPT)

- SecVisor uses SPT to set memory protections
  - Intercept user↔kernel switches
  - Protect approved code from modification
Protecting Approved Code

- Set approved code regions read-only in SPT
- Use DEV to prevent DMA writes to approved code regions
- Prevent aliasing of approved code physical pages (not mentioned in the paper)
Problem: Physical Address Aliasing

Check: No aliasing of approved code physical pages
Outline

- Introduction
- Conceptual Design
- Implementation
  - Setting memory protections
    - Intercept user→kernel switches
    - Protect approved code from modification
  - Checking and protecting entry pointers
    - Constrains IP on kernel mode entry
- Experiments and Results
- Related Work and Conclusion
Checking Entry Pointers

- On the x86, entry pointers can exist in GDT, LDT, IDT, and MSRs
- Entry pointers are all virtual addresses
- Two checks are needed:
  1. Entry pointers contain virtual addresses of approved code
  2. Entry pointer virtual pages must translate to physical pages containing approved code (not mentioned in paper)
Remapping Attack Using KPT

Check KPT mappings for approved code
Protecting Entry Pointers

- Attacker could modify entry pointers in memory during user mode execution
  - Could use DMA writes, for example
- Protect in-memory entry pointers by shadowing GDT, LDT, and IDT
- Details in paper
Outline

- Introduction
- Conceptual Design
- Implementation
- Experiments and Results
- Related Work and Conclusion
Experimental Setup

- HP Compaq dc5750 Microtower PC
- 2.2 GHz AMD Athlon64 X2 (dualcore CPU)
- 2 GB RAM
- Two sources of overhead:
  1. Intercepting user↔kernel mode switches
  2. SPT synchronization and KPT checks
- I/O intensive workloads with rapidly changing working sets will be most affected
Results – Specint 2006

- Runtimes normalized to native Linux (lower is better).
- The chart compares the performance of SecVisor, SecVisor New, and Xen.

### Runtimes

- **perlbench**: 1.08, 1.02, 0.99
- **bzip2**: 1.09, 1.03, 1.00
- **gcc**: 1.93, 1.57
- **mcf**: 1.16, 1.10
- **gobmk**: 1.03, 0.93
- **hmmr**: 1.02, 1.00
- **sjeng**: 1.06
- **libquantum**: 1.06
- **h264ref**: 1.07
Results – Applications

The chart shows the runtime (normalized to native Linux, lower is better) for different benchmarks:

- **Kernel Build**
  - SecVisor: 2.19
  - SecVisor New: 1.66
  - Xen: 1.12

- **Kernel Unzip**
  - SecVisor: 1.40
  - SecVisor New: 1.31
  - Xen: 1.09

- **Postmark**
  - SecVisor: 1.86
  - SecVisor New: 1.51
  - Xen: 1.13
Related Work

- **Kernel integrity protection**
  - IBM 4758, Program Shepherding, Livewire, SVA

- **Small VMMs**
  - Terra, TVMM, lguest

- **Kernel rootkit detection**
  - Software-based: AskStrider, Pioneer…
  - Hardware-based: Copilot…
Cool Things Not Mentioned

- Secure startup
- Dealing with BIOS
- Whitelist-based approval policy
- Implementation using nested page tables
- Identifying entry pointers on x86
- Protecting GDT, LDT, and IDT on x86
- Allocating and protecting SecVisor memory
- Application to code attestation
Future Work

- Release source code
- Update paper to describe new defenses
- Finish up formal verification of SecVisor code
Conclusions

- **SecVisor prevents** code injection attacks against commodity kernels
  - All other techniques are detection-based
- Defends against powerful attackers
- Amenable to formal verification and manual audit
Acknowledgements

- Shepherd - Richard Draves
- Anonymous reviewers
- Bernhard Kauer, Benjamin Serebrin, Leendert van Doorn, Elsie Wahlig, Daniel Wendlandt
- ARO, NSF, AMD, KDDI for research grants
- NSF for SOSP student travel grant
Thank you!