

# Coprocessor-based Integrity Monitoring

WATSH November 9, 2004

Nick L. Petroni, Jr.

*npetroni@cs.umd.edu*

Timothy Fraser

*tfraser@umiacs.umd.edu*

Jesus Molina

*chus@cs.umd.edu*

William A. Arbaugh

*waa@cs.umd.edu*



# State of Commodity OS Security

- Complexity abounds
  - Commodity OSs are already complex (and growing)
  - Placing assurance on the many parts is difficult
  - Hard to make claims about runtime OS integrity
- Existing security tools rely on system correctness
  - All host software relies on some aspect of kernel integrity
  - The kernel itself is large and is often a point of attack

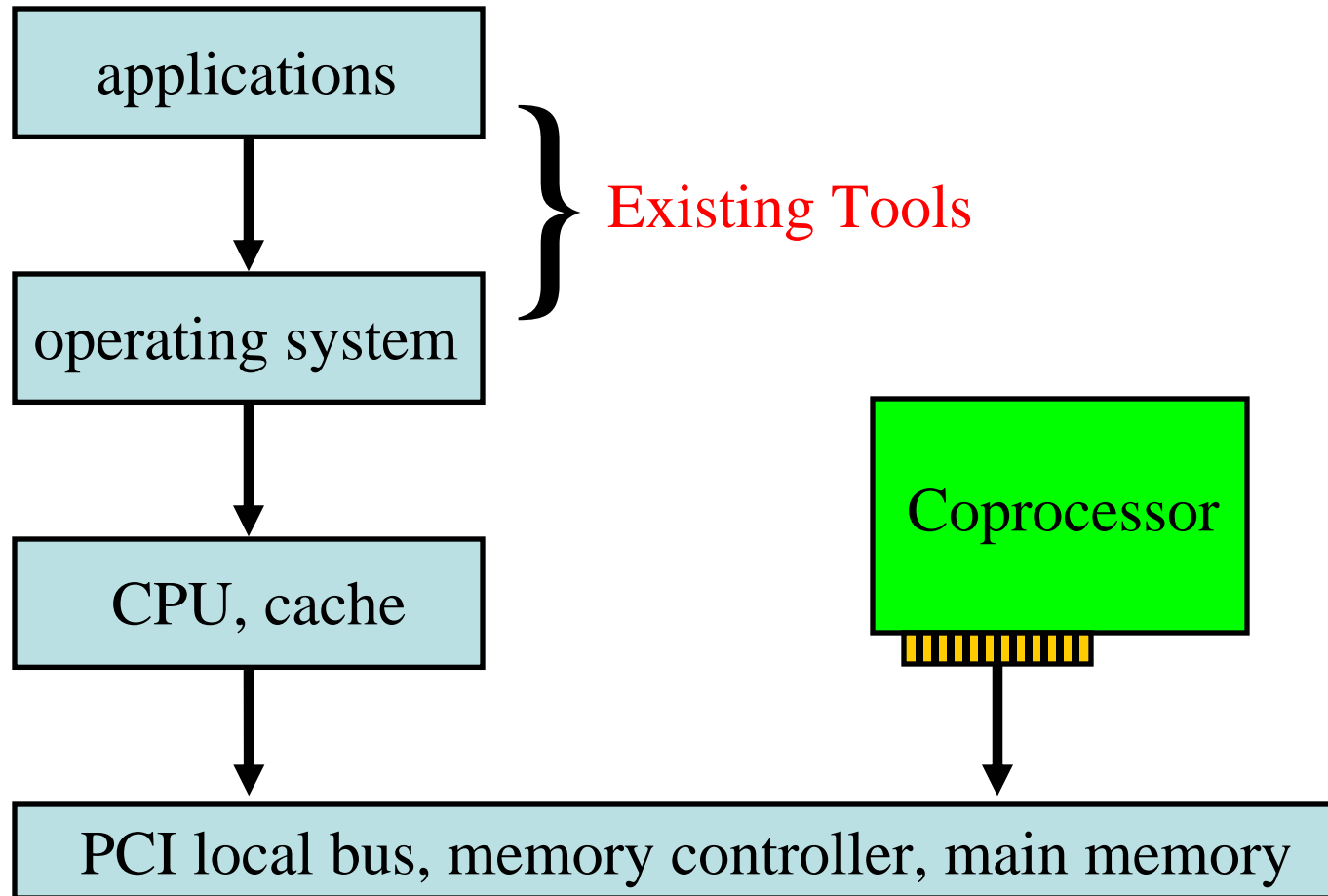


# Coprocessor Integrity Monitoring

- Remove reliance on system software correctness
  - Use hardware access to resources (e.g., memory)
  - Run protection code on a coprocessor (NOT the host)
  - Provide a secure reporting mechanism
- Basic model:
  - Collect data based on monitor's observables
  - Determine if data violates monitor policy
  - Take some action (e.g., report, recover, etc.)



# Correctness Dependencies



# Are Kernel Integrity Threats Real?

- A large body of information exists on kernel rootkits
  - Hundreds of articles in online content such as Phrack
  - Dozens of tools dedicated to detecting thousands of examples
  - Training sessions at popular conferences (e.g. BlackHat 2004)
- Incidents involving kernel modifications do occur
  - <http://securecomputing.stanford.edu/alerts/multiple-unix-6apr2004.html>  
“Multiple UNIX compromises on campus”
- Vulnerabilities do arise in OS kernels
  - OSVDB 3315 (CVE-2003-0985): overflow in the Linux Kernel
  - OSVDB 7330 (CVE-2004-0602): FreeBSD kernel code injection



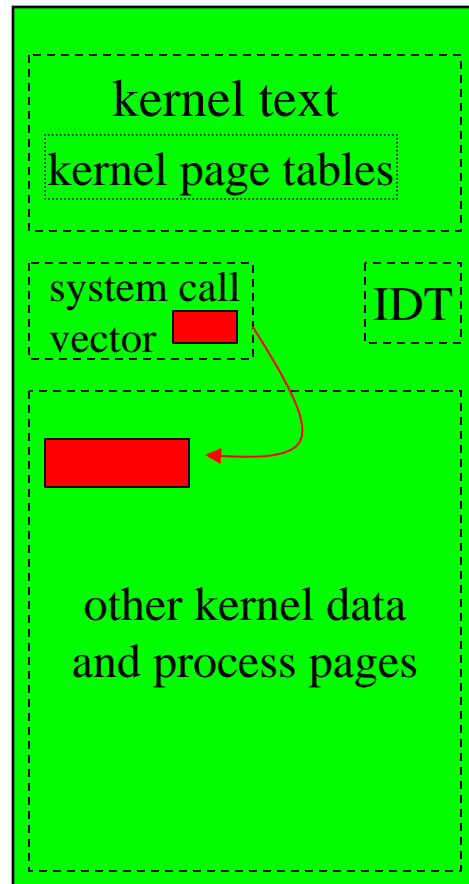
# Copilot Kernel Integrity Monitor

- Compatible
  - works on commodity GNU/Linux x86 PCs
  - works on commodity Windows XP/2K x86 PCs
- Effective
  - detected tampering by 12 real-world rootkits
  - check every 30 seconds = 1% performance penalty
- Isolated
  - implemented on its own PCI add-in card
- Independent
  - operates even if host kernel is compromised



# Integrity Threat Example

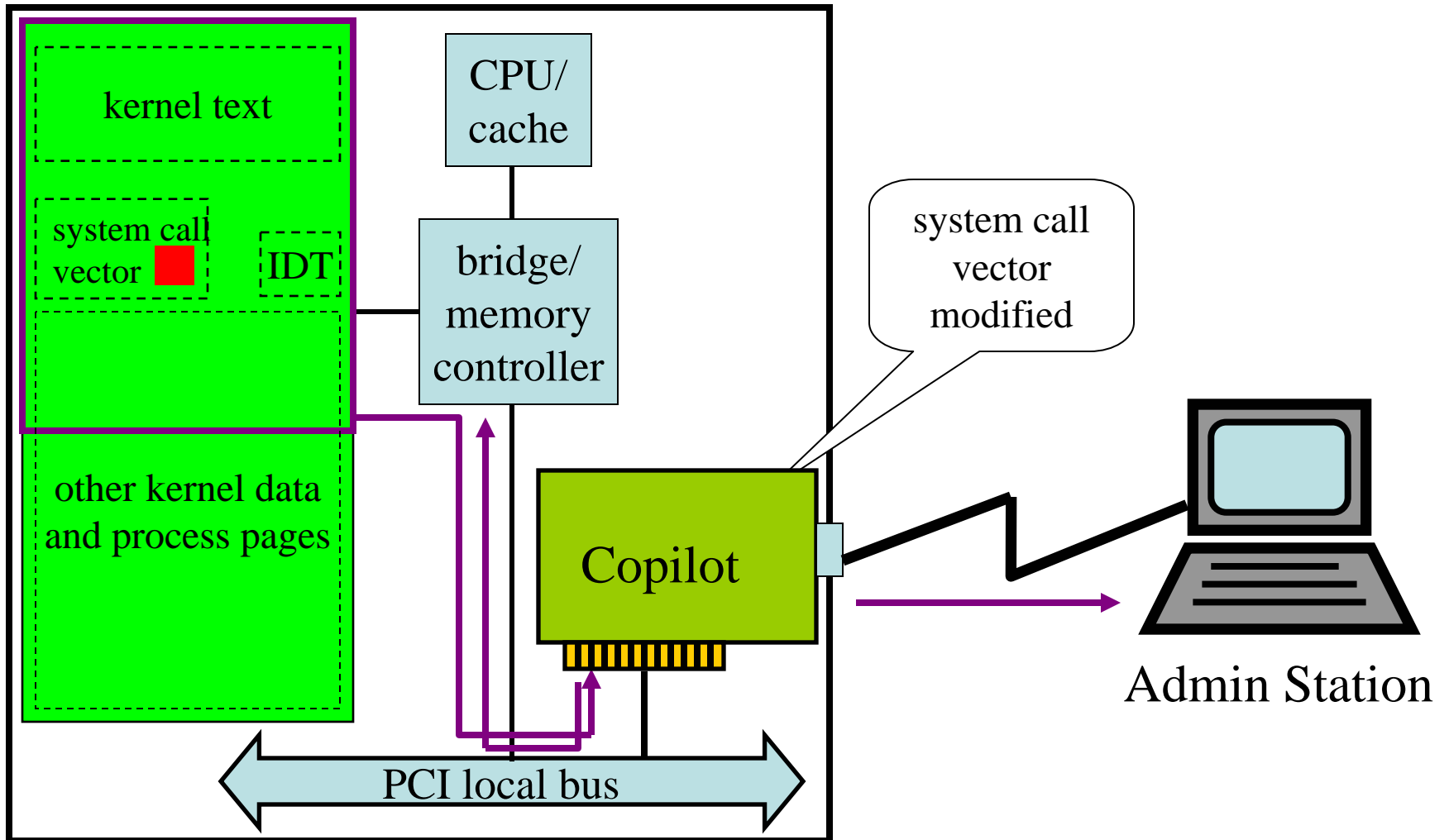
Host RAM



1. Attacker gains entry
2. Attacker inserts/modifies code
3. Attacker gets his code executed



# Copilot Integrity Protection



# Copilot Protection Strategy

- Copilot currently uses the following traditional methods:
  - Hash of Linux kernel text
  - Linux system call vector
  - Linux interrupt descriptor table
  - Linux module list
  - Hash of Linux module text
- Compare the above with a “known-good” state
- Adding hashing/jump table targets is simple
- Copilot improves these methods by providing an isolated and independent platform for kernel monitoring



# PCI Add-In Card Requirements

- Unrestricted access to bus
  - EBSA-285 has bus mastering capability
- Independence from host
  - EBSA-285 has a mode that ignores host commands
- Sufficient processing power, memory
  - StrongARM SA-110 CPU, 16MB RAM
- Independent communication channel for reporting
  - RS-232 serial port



# Architecture/OS Requirements

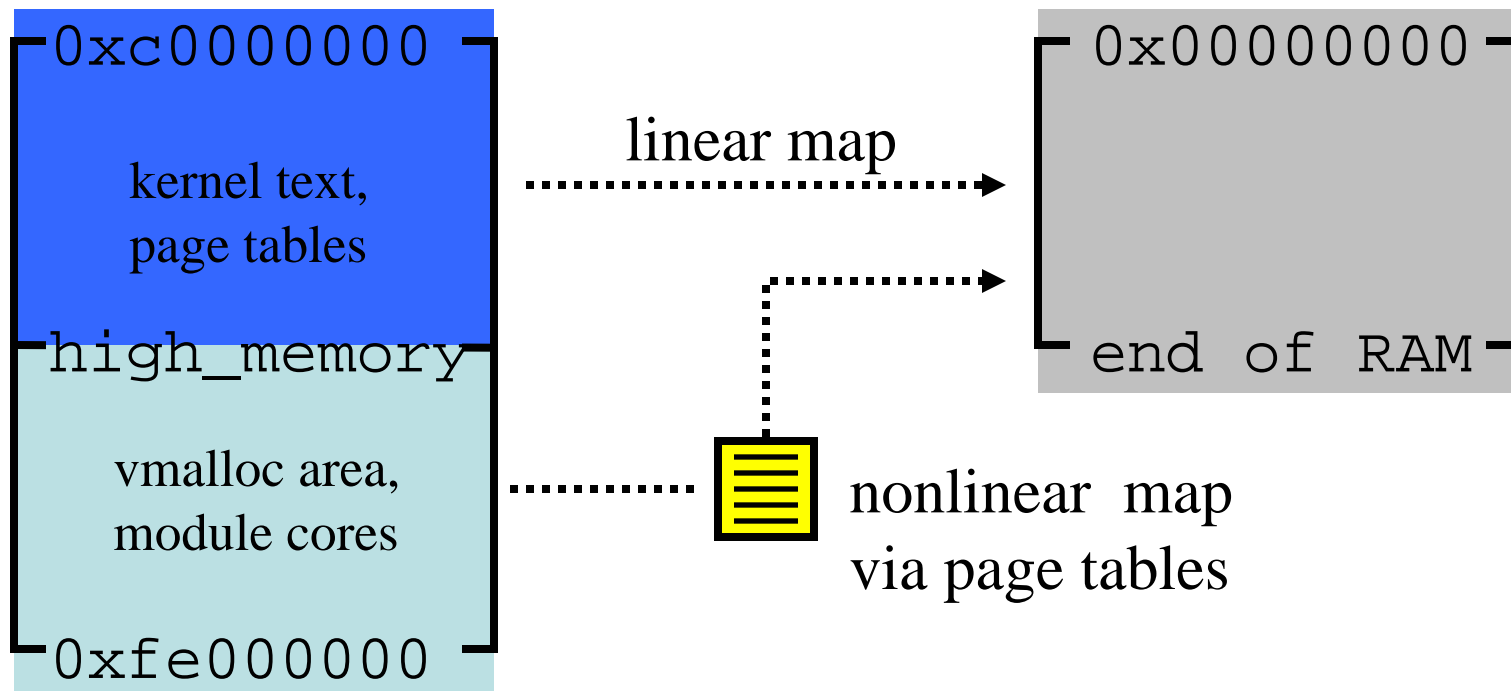
- All kernel data structures must be available to monitor
  - Linux provides virtual addresses for data structures
  - Linux x86 virtual address translation easy to replicate
  - Linux kernel memory is never paged out of RAM
  - PC PCI bus addresses, physical addresses identical
  - PC PCI bus can address all of physical memory



# Virtual Memory Translation

virtual addresses  
used by kernel:

physical addresses  
used during DMA:



# What is a Rootkit?

Software used after system compromise to:

- Hide the attacker's presence
- Provide backdoors for easy reentry

Simple rootkits:

- Modify user programs (ls, ps)
- Detectable by tools like Tripwire

Sophisticated rootkits:

- Modify the kernel itself
- Hard to detect from userland



# Rootkit Features

Typical rootkit implementation:

- An LKM that interposes on the system call vector:
- Adore, rial, rkit, synopsis, modhide1, phide, kbd, linspy...

More sophisticated, more stealthy:

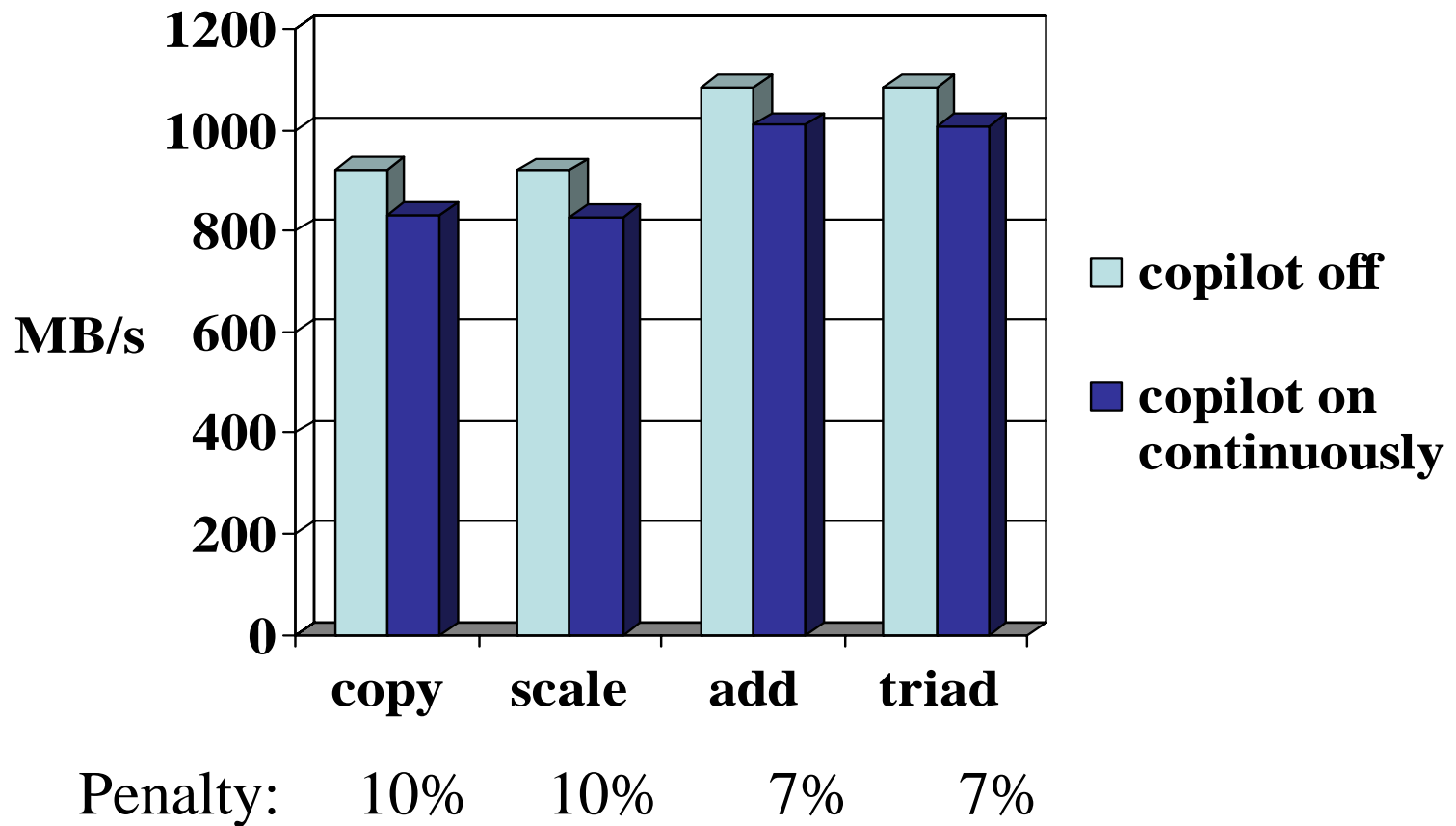
- SucKIT - loads via /dev/kmem instead of LKM
- Phantasmagoria - modifies kernel text, not syscall vector

Insecurity by Obscurity:

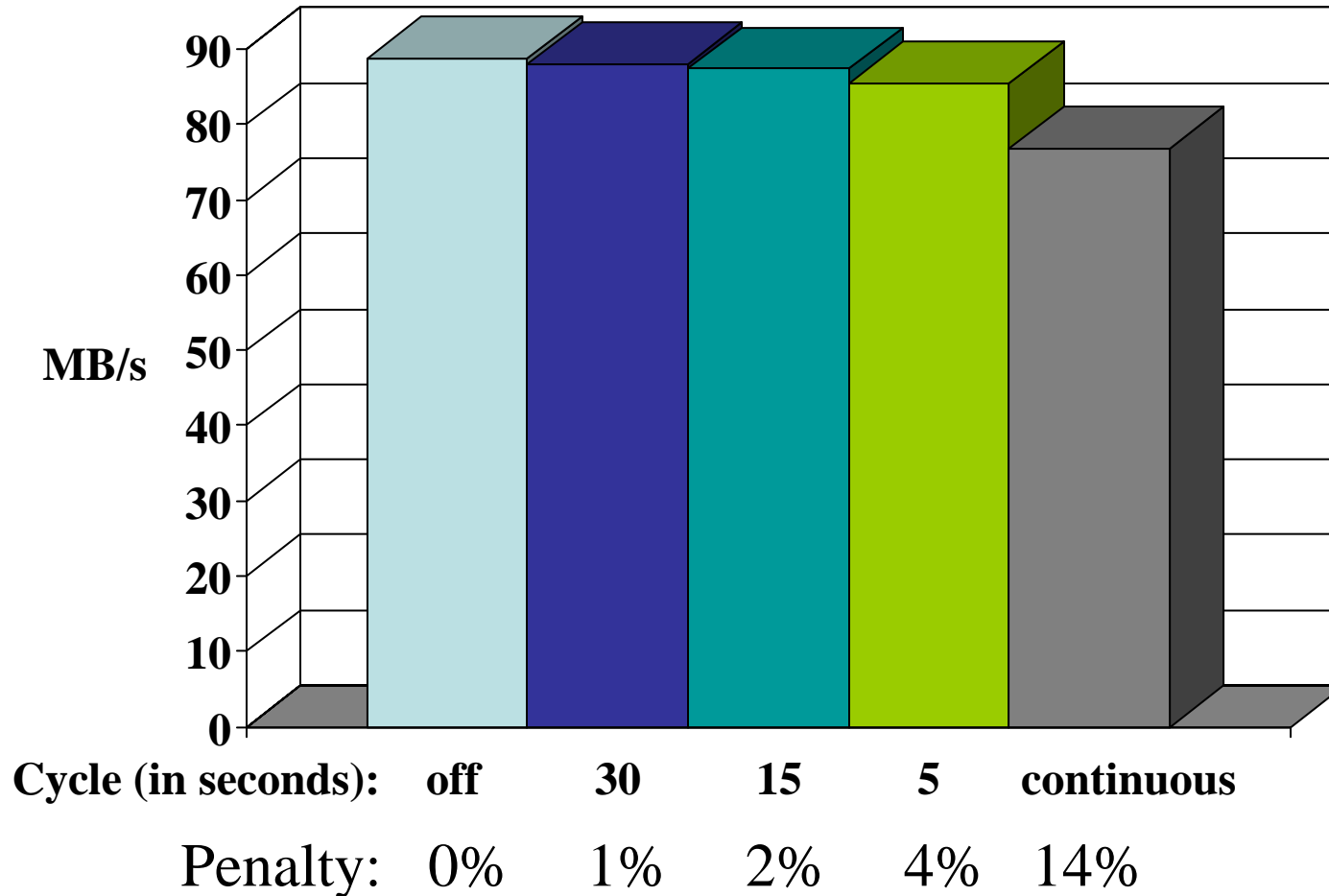
- Taskigt - adds a hook to /proc filesystem
- Knark - adds inet protocol handler



# STREAM memory throughput benchmarks



# WebStone HTTP throughput benchmark



# Limitations of Current Copilot Monitor

- Problems with a PCI-based approach
  - Inability to monitor execution
    - Monitor can only see effects of running processes on main memory, not the order of execution or operations
  - Relocation/cache attacks
    - Attacker can modify registers to point to a different place in memory. The monitor will continue checking the old locations
  - Memory race conditions
    - While interpreting kernel data structures, the host OS will also be modifying data



# Related Approaches

- Software-based attestation
- Hardware-based attestation
- Virtual Machine-based approaches
- Multi-processor (SMP)-based monitoring





# Software-based Attestation (cont.)

- Advantage: No need for new hardware
- Disadvantages
  - SWATT
    - Will not work for more advanced architectures that have VM
    - For Von Neumann machines, the verifier needs to predict dynamic data
    - Test requires device isolation since there is no work binding to a machine
  - Genuinity
    - Correctness is disputed: Shankar et al., USENIX Security 2004
    - Requires changes to host OS and cannot be used for slower machines
    - Relies on intimate knowledge of host architecture
  - Both
    - Checks occur before/after actual work on the system – an attacker has a large window of opportunity



# Hardware-based Attestation

- Based on use of Trusted Computing Group's Trust Platform Module
- Hardware components provide a set of primitives that can cryptographically prove the computation took place on a particular device
- Tamper-resistance ensures that the device remains unaltered
- A set of entities take measurements of the system and store them in TPM
- Measurements cannot be reversed and are reset only on reboot
- How to provide runtime integrity?
  - Unalterable piece of BIOS measures Bootloader
  - Bootloader measures kernel
  - Kernel measures changes to itself and loaded programs, static data, etc.
  - Third party can verify the signed measurements are what they expect

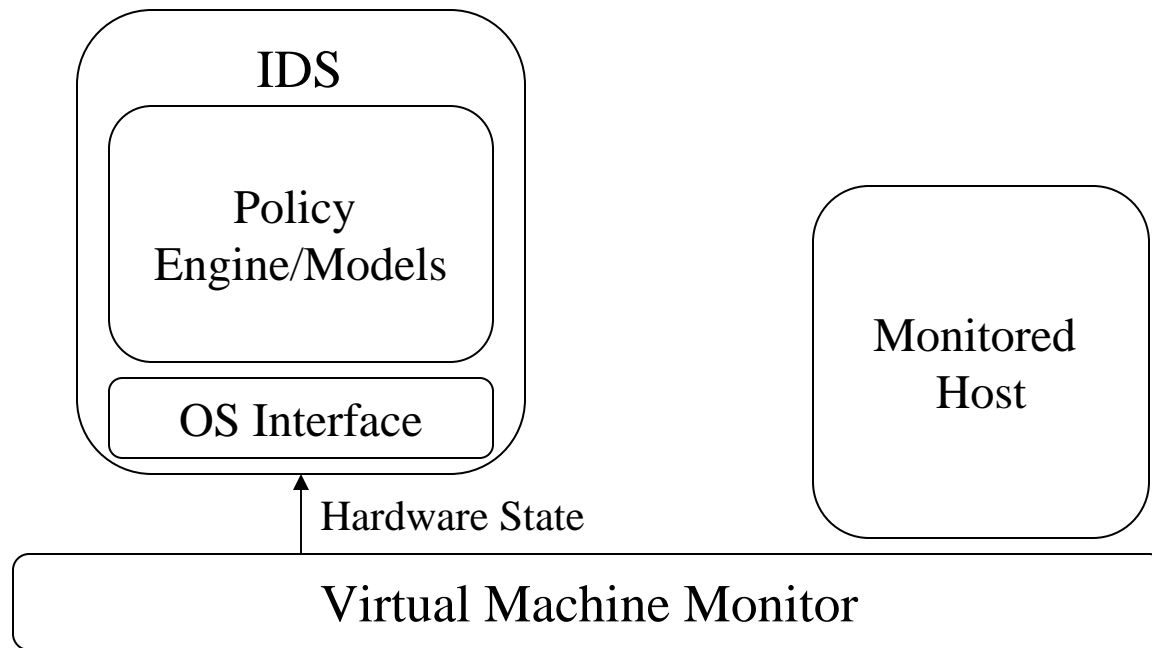


# Hardware-based Attestation (cont.)

- Advantages:
  - Strong root of trust
  - Provides measurements from the beginning of the boot process
  - Will be available on many new systems
- Disadvantages:
  - Measurements take place only at load time, flaws in “valid” software may allow runtime changes
  - Requires special hardware not available to legacy systems



# Virtual Machine-based Monitors



- NDSS 2003: Virtual Machine Introspection
- VMM has access to ALL of the monitored host's virtual hardware
- VMM can pause guest OS to see a consistent state

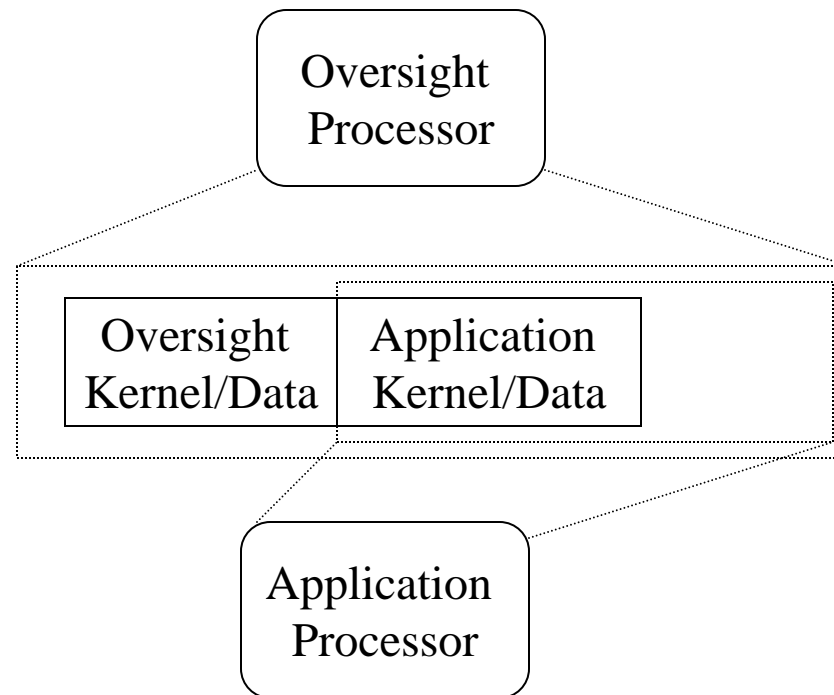


# VM-based Monitors (cont.)

- Advantages:
  - Has visibility of all virtual hardware
  - Has greater control over the guest OS
  - Can be integrated with TCG to provide stronger assurance of the virtualization layer
- Disadvantages:
  - Includes host OS and VMM in Trusted Computing base
  - More difficult to add to an existing (non-virtualized) host
  - Performance penalty of virtualization may be high



# SMP-based Monitoring



- MILCOM 2000: Hollingworth, Redmond
- Use one processor of SMP machine as oversight monitor
- Oversight kernel has full access to system memory
- Application processor configured to use a subset



# SMP-based Monitoring (cont.)

- Advantages:
  - Takes advantage of architectural access to main memory
  - Provides a more ‘in line’ approach – processor can trap on certain kernel calls
- Disadvantages:
  - Without hardware changes, difficult to prevent advanced attacker from modifying visible memory window
  - Requires SMP hardware without the added performance of using the second processor for real work
  - As with copilot, cannot see registers of second processor
  - Contention with Application Processor resources (e.g. NIC)



# Future Work

- Research as to the most effective policies and models for performing memory monitoring
- Explore methods for integrating processor register data into the monitor's set of observables
- Develop methods for automating the protection strategy based on code/binary
- Expand the response capabilities of monitor



# Conclusion

- Proven effective in lab tests:
  - Detected 12 real rootkits
  - Less than 30-second detection window
  - Low host performance penalty
- Advantages over existing technologies:
  - Applicable to commodity x86 hosts
  - No reliance on host software for correctness
  - Plugs into unmodified commodity host



# Questions?

Links to this presentation and the paper can be found at:

<http://www.cs.umd.edu/~npetroni/research.html>

