TOCTOU, Traps, & Trusted Computing

Sergey Bratus
Nihal D'Cunha
Evan Sparks
Sean Smith
Dartmouth College

The TOCTOU threat to the TCG architecture

- TCG provides only load-time guarantees
  - measurements are taken when software is loaded into memory
  - loaded software & data assumed unchanged
- TPM holds only static measurements:
  - code or data changes after measurements will not be reflected in TPM's state
- Run-time vulnerabilities that results in changes of code & data are a significant threat.

What can we do about it?

"Trusted" != "Trustworthy"

Write safer (*more trustworthy*) programs :-)

- (1) Provide the programmers with the means to do so, and/or
- (2) Mitigate insecurity from programming complexity with policy enforcement.

This talk explores an approach to (1) and (2).

Try an “outside-the-box” look at the TCG architecture (1)

“Next generation trusted platforms should be able to enforce policies.”

-- Proudler, 'Trusted Computing', 2005

- What policies can we enforce with TPM/TCG-based architectures?
- How do they compare with, e.g., SELinux policies?
- What new secure programming primitives can we provide for interested programmers?

Secure programming primitives

A TPM is used to provide a range of hardware-based security features to *programs that know how to use them.*

Classic UNIX examples:

- read-only/*const* data in programs
- privilege drop syscalls & privilege separation
- SELinux API, e.g., *setcon()* to give up access rights

It's up to the programmer to design the software to take advantage of the new features.

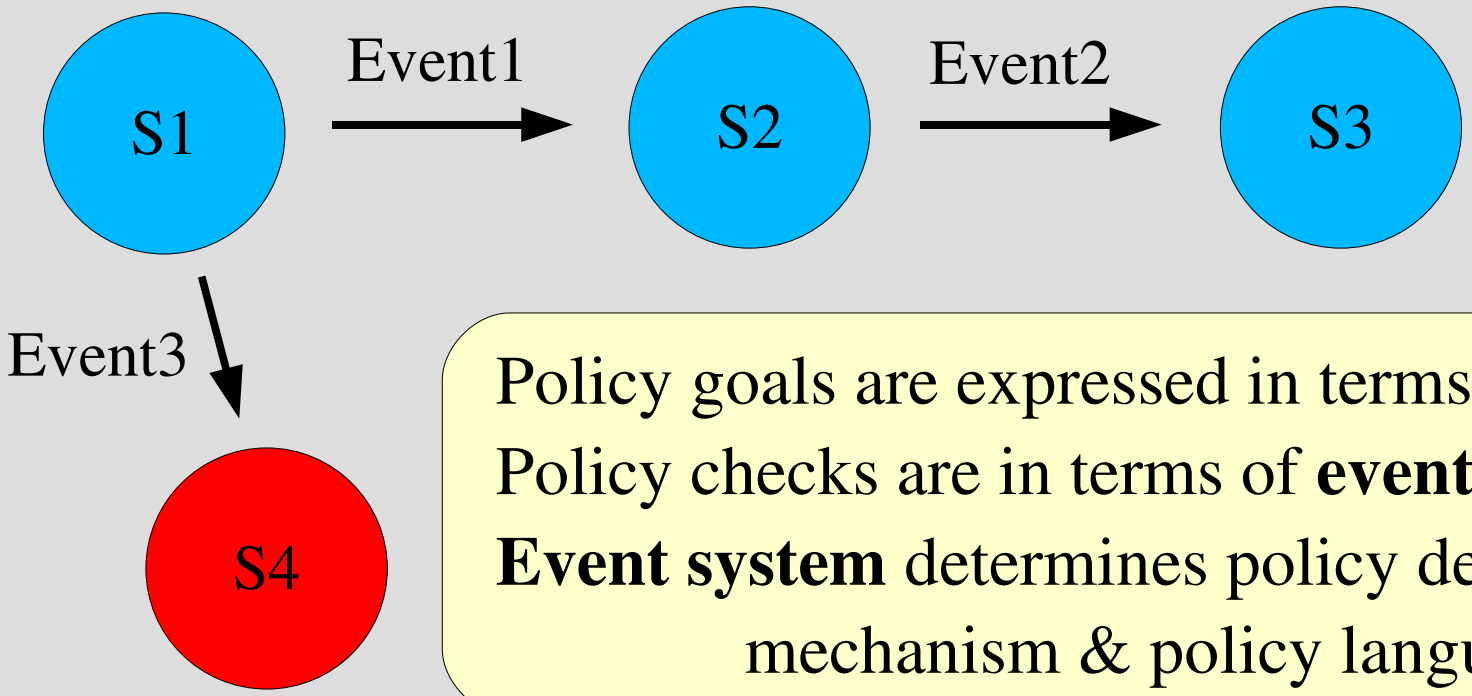
How can we help TC programmers to mitigate TOCTOU?

- Provide TPM-based secure programming primitives.
- Common TOCTOU vulnerabilities involve changing code or data between TOC & TOU.
- Allow programmers to express extra semantics of when and how data and code objects are (& are not) allowed to change.
  - They know it best, as with const-designated data, privilege separation, or SELinux MAC
- Make a policy of it!

A look at policy mechanisms

A policy should prevent the system transitions to "untrusted" states from "trusted states"

- agrees with the TCG *chain of trust* concept



Policy goals are expressed in terms of **states.**

Policy checks are in terms of **events/**transitions.

Event system determines policy design, mechanism & policy language.

SELinux policies

- Monitored events are privileged operations
  == selected system calls
- The syscall hooking system, LSM, provides an implicit definition of the event system
  - affects design and scope of SELinux policies
  - too many events => hard to check & manage
- Practitioners looked for other ways to express their policies:
  - Solaris Zones, AppArmor
  - Virtualization (Linux Vservers, BSD jails, etc.)

Measured states for TCG-based policies? (1)

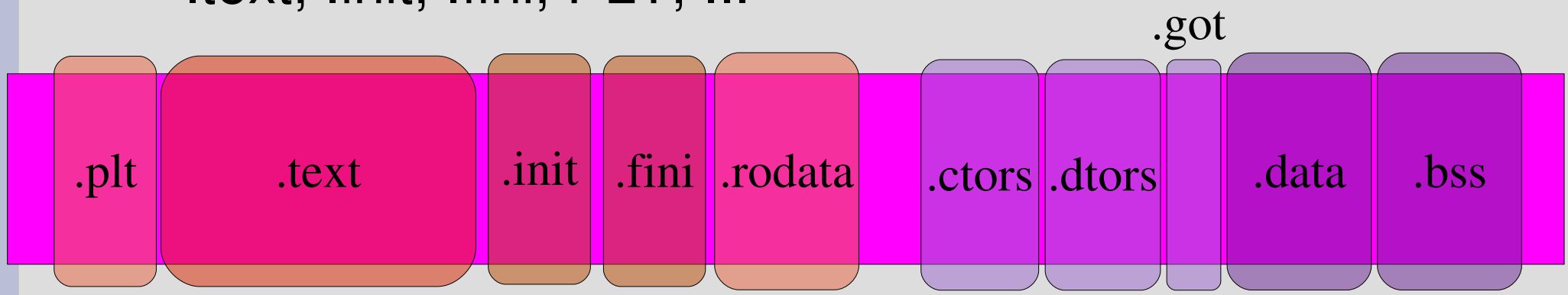
"We know of no practical way for a machine to distinguish arbitrary software than to measure it (create a digest ... using a hash algorithm) and hence we associate secrets and private data with software measurement."

-- Proudler, 'Trusted Computing', 2005

- Defining *public* vs. *private* vs. *secret* trusted data is inevitably left to the programmer
- This data must then be annotated in the program
- Policy states & goals are in terms of this annotation

Measured states for TCG-based policies? (2)

- ELF defines various data & code objects:
  - .bss, .rodata, .ctors, .dtors, GOT, .dynsym, ...
  - .text, .init, .fini, PLT, ...



- Can accommodate custom section types:
  E.g.., ELF format + gcc:
  __attribute__((section("PRIVATE_DATA")))

Selective immutability as a matter of policy

Addresses many kinds of TOCTOU:

- .text, .dtors, .rodata, .got, .plt attacks
  (run-time attacks that change the code or linking-related objects, e.g., destructor list .dtors)
- programmer-defined private or secret data sections with special trust semantics
- protected sections part of attested software state

"Make the TPM notice when TPM-measured protected memory is changed"

Measured states for TCG-based policies? (2)

- Loadable segments are page-aligned
  - protection enforced by PDE, PTE bits (x86) that cause page faults to occur on access
  - protection bits imply a rudimentary kind of access policy to data & code objects
- Why not push for HW design for finer trap granularity?
  - could enforce finer policy for custom sections
  - must change the MMU...
  - other MMU changes are desirable, too

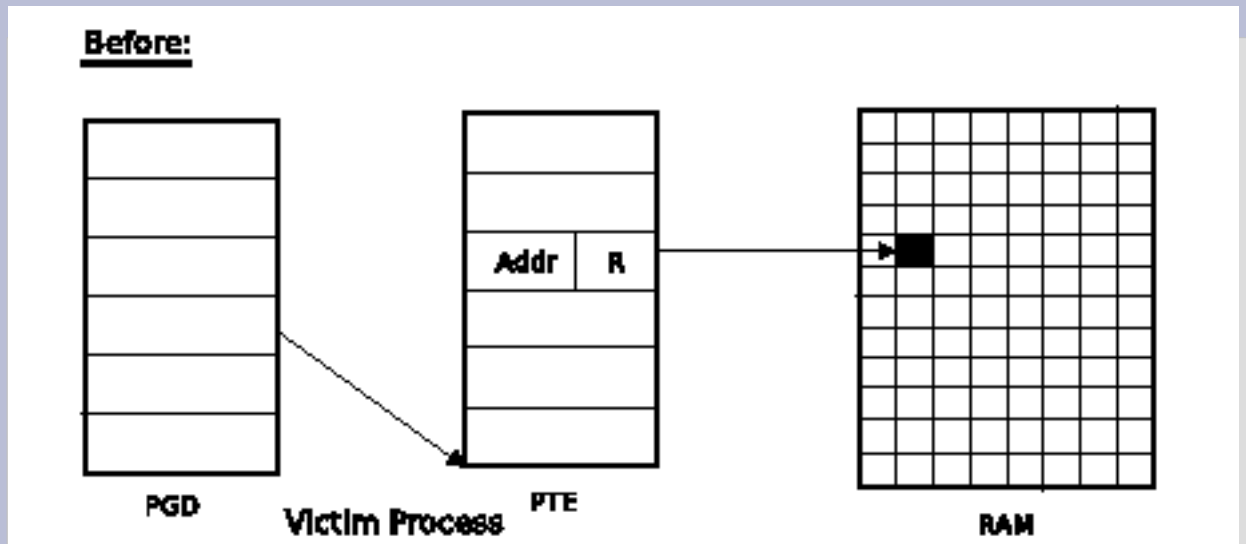
Attack scenario example

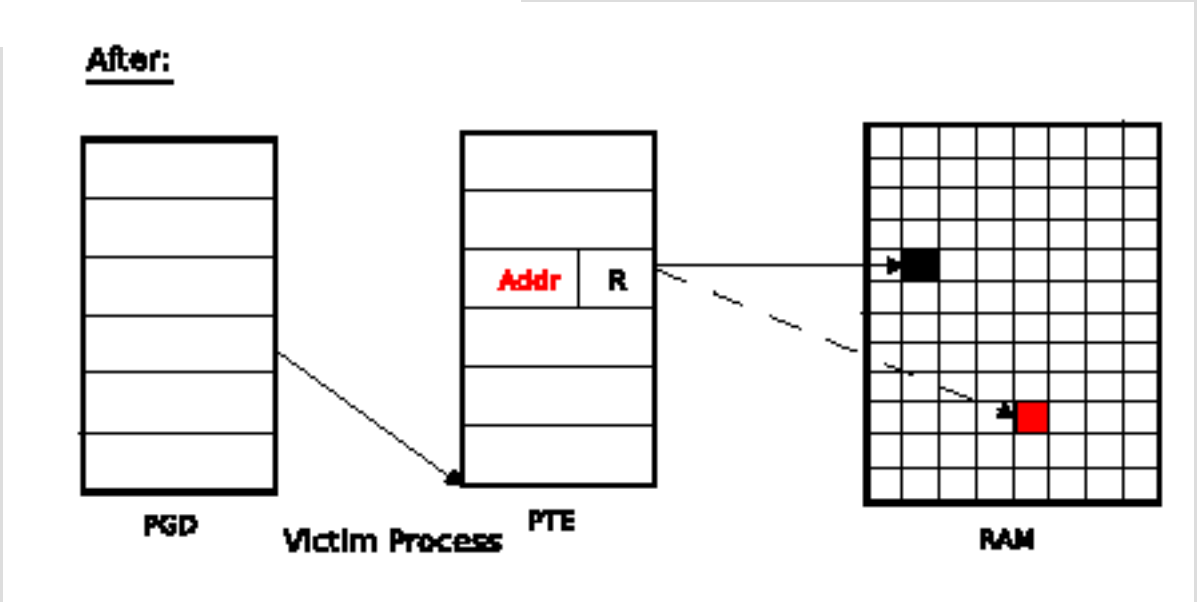
- *Certification Authority* bootable liveCD package
  - uses TPM to store CA's private key
  - private key wrapped to specific values of PCRs
- A vulnerability that allows memory modification (CA process or kernel) breaks the trust
- E.g.: attacker exploits
  - remote code execution vuln in X.509 parsing
  - local privilege elevation vuln to get root
  - loads an LKM that changes some “je” to “jne”
  - the TPM is none the wiser

Proposed solution (1)

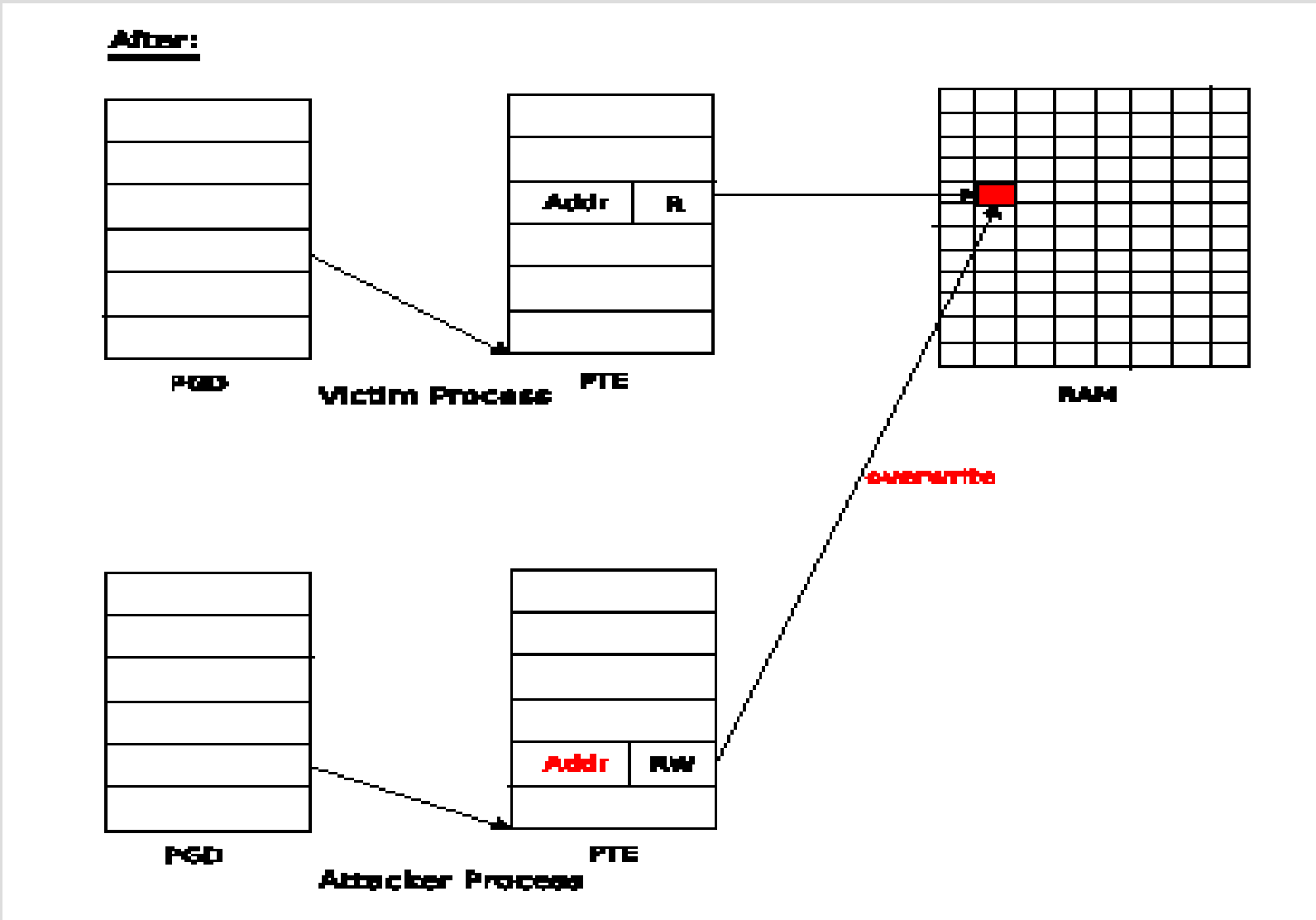
- Trap when protected memory is written to
  - TPM-aware handler: invalidates TPM PCRs
- Must watch not only writes to virtual addresses within protected section, but also to any physical pages that get mapped there.
  - needs careful integration with memory management, best at the MMU level
  - change the MMU!
  - also, MMU changes can give higher granularity
- Preserves passive nature of the TPM.

Attacker changes victim's PTE to point to another frame





Attacker points PTE of owned process to point to victim's page

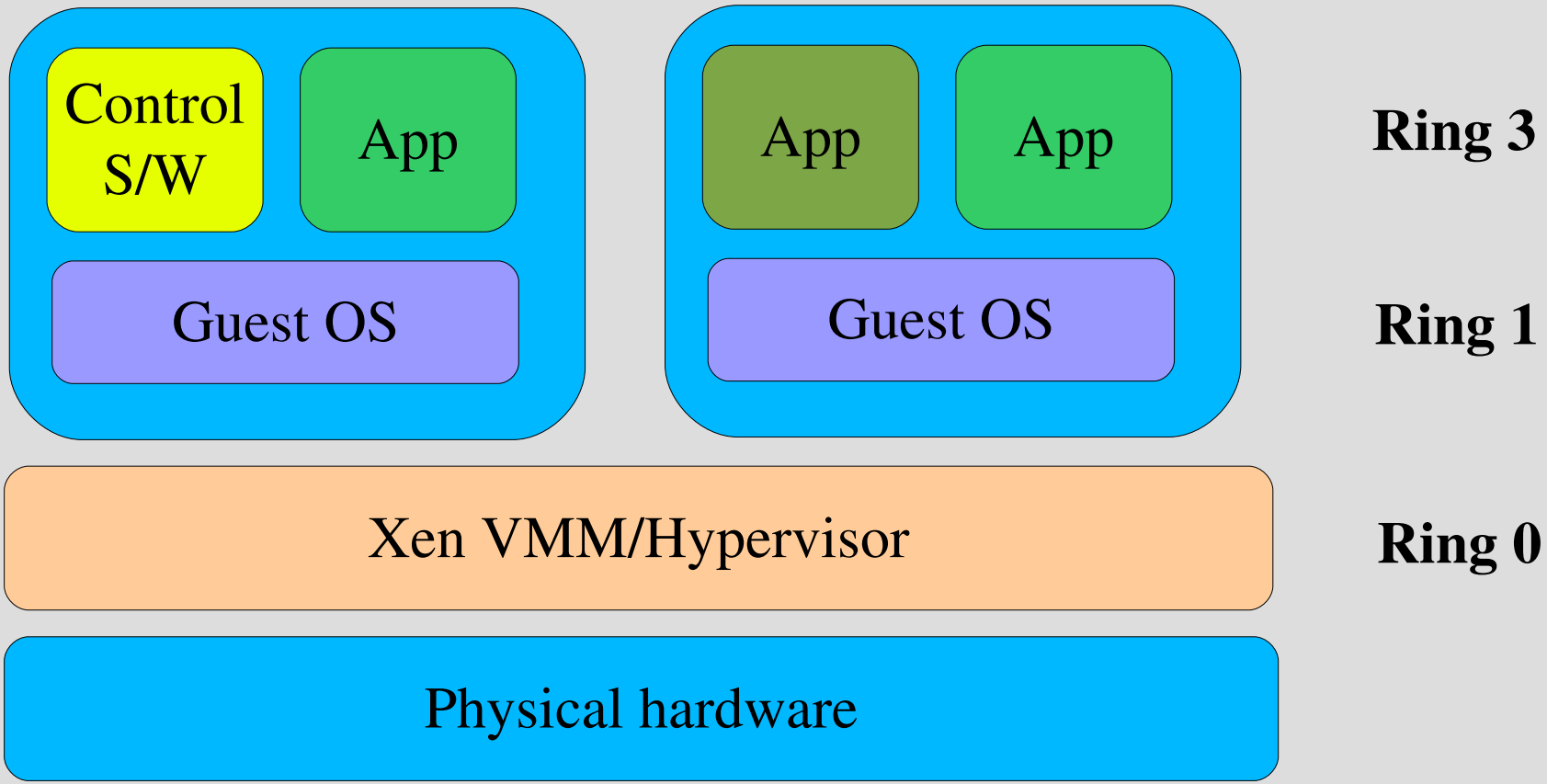


Proposed solution (2)

- Modify the MMU to generate traps on
  - writes to marked memory regions
  - mapping of marked memory regions
  - higher memory granularity is desirable
- Programmers get a new secure programming primitive: "TPM-sealed" memory region
  - compiler and ELF format support is almost there
  - programmer gets to specify access semantics of different data and code objects
    - when sealed (e.g., after linking) or unsealed

Software prototype for feasibility study

- ... because hardware hacking is hard...
- We use Xen as a memory trapping layer

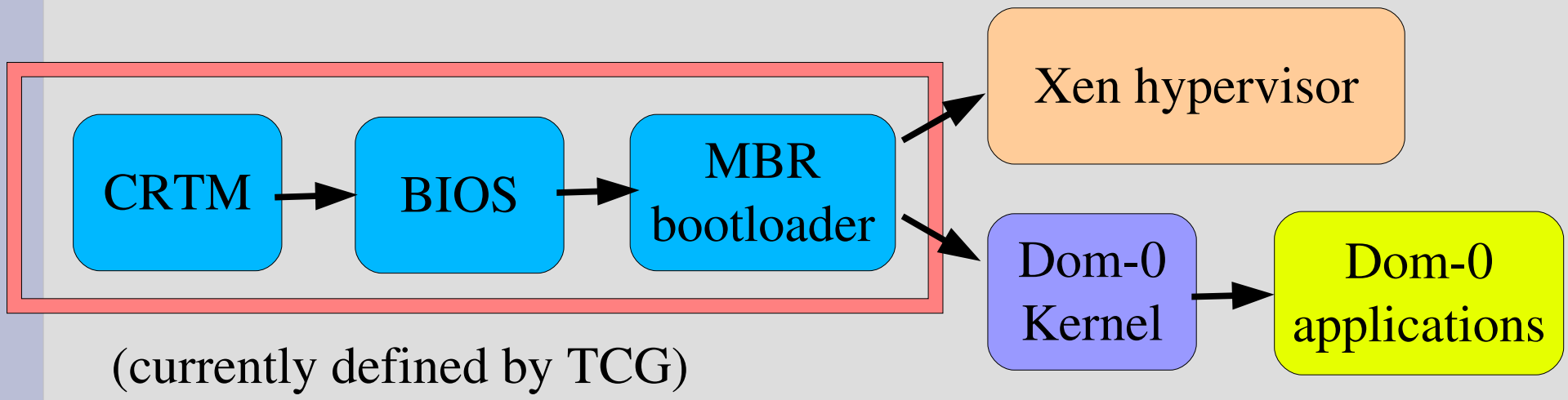


Why Xen?

- *Xen already does a lot of MMU-type work!*
- We use Xen to *emulate a hardware trap framework* for intercepting memory events of interest to our policy
  - thin hypervisor, traps all memory updates
- Bochs & Qemu did not support emulating a TPM, do not integrate with software-based emulated TPM (Strasser, 2004)
- vTPM (Berger et al., 2006) integrates with Xen
  - provides TPM access in test Domain-1, ...

Can Xen be a part of the TCB?

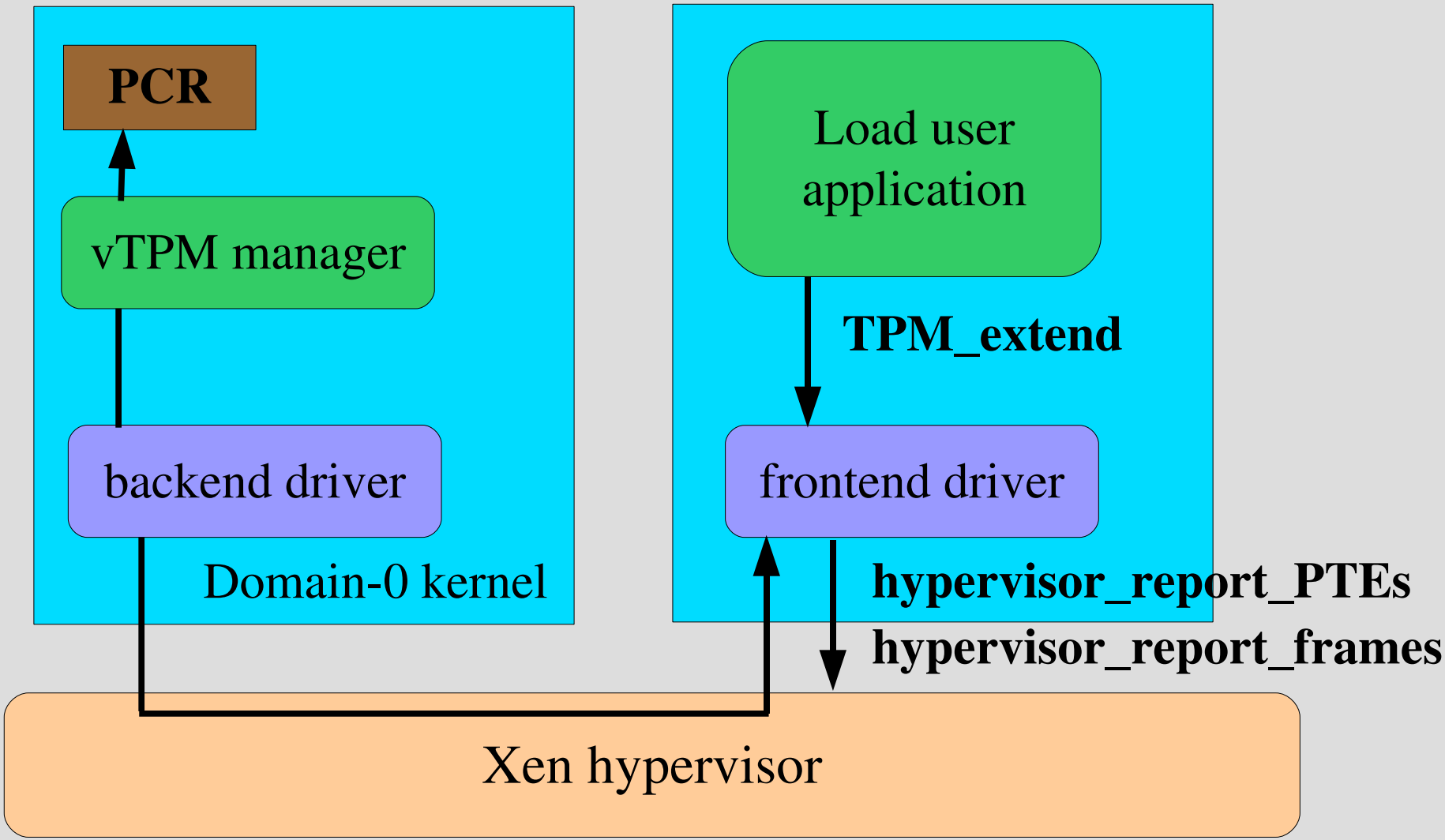
- Alternatively, one could imagine Xen as part of the Trusted Software Stack (TSS)
  - trusted boot loader measures the Xen hyperviser, Dom-0 kernel
  - chain of trust then extended to Domain-0 control and management applications



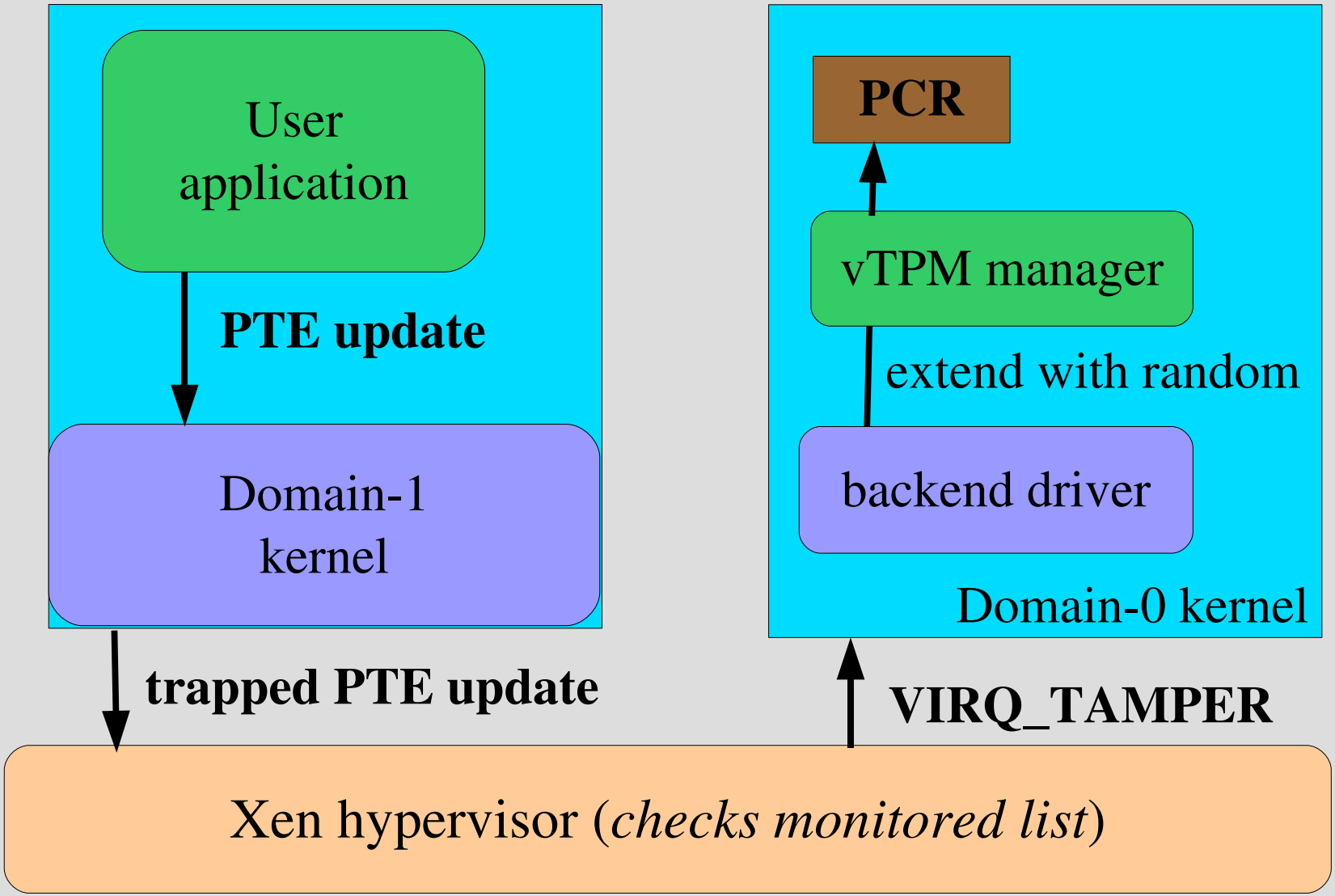
Two possible ways to hook memory updates

- "Dynamic TPM"
  - update TPM state every time measure memory region is changed
  - re-measure region, extend the TPM with the new measurement
  - TPM state reflects memory state at all times
- "Tamper-indicating TPM" (<-- our choice)
  - updated only the first time measured memory region is changed
  - no re-measurement, extends TPM with random value
  - less performance hit

Implementation: reporting frames & PTEs to monitor



Implementation: monitoring for updates



Gory details

- Xen's Writable Page Table mode as default
  - writes to page tables are all trapped & emulated by the hypervisor (via *update_l1e()* function)
  - we instrument *update_l1e()*
- New hypercalls to manage monitored lists:
  - *HYPERVISOR_report_PTEs*
  - *HYPERVISOR_report_frames*
  - *HYPERVISOR_report_exit*
- New virtual IRQ: VIRQ_TAMPER for guest OS
  - some hacks required to deliver *TPM_Extend* to the actual TPM from Domain-0 with vTPM

Related work: Attestation

- IBM's Integrity Measurement Architecture (IMA) for Linux, (Sailer et al., 2004)
  - extends trust from BIOS to application layer, measurements taken on load, stored in kernel
- Dartmouth's LSM-based Bear/Enforcer (Marchesini et al., 2004)
  - checks selected files' integrity on open()
- BIND Fine-grained attestation (Shi et al., 2005)
  - attests only critical pieces of code about to execute
  - requires annotation, narrows “TOATOU” gap
  - runs on LaGrande-style CPUs

Related work: High granularity RAM traps

- Page-granularity causes performance grief
  - not only for us: e.g., memory architecture research (studies of access patterns, caches, ...)
- Creative use of ECC bits to get finer memory traps on non-x86:
  - Wisconsin Wind Tunnel (Reinhardt et al., 1993)
  - Blizzard-E (Schoinas et al., 1994)
  - ...?
- Emulated prototypes:
  - Mondrix (Witchel et al., 2005)

Open problems & issues

- Non-page table memory operations: DMA?
- Sealing/binding secrets to owner process?
  - TCG specifies using (optional) password stored in wrappedKey data, but:
  - without password, data could potentially be unsealed by any running application
  - **Idea**: a resettable PCR extended with the measurement of the currently running process?
    - then owner must be active on the CPU to unseal
    - the special PCR reset on each context switch
- Dynamic TPM?

Thank you!

- Questions?