Faculty of Computer Science Institute for System Architecture, Operating Systems Group

Hardware Enforcement of Application Security Policies Using Tagged Memory

Nickolai Zeldovich, Hari Kannan, Michael Dalton and Christos Kozyrakis

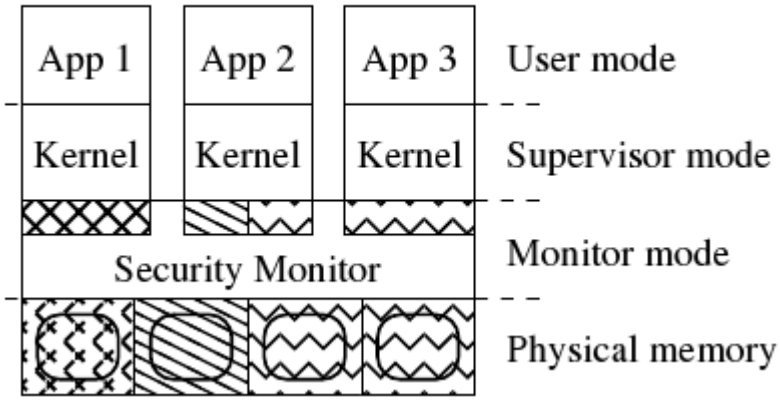
Dresden, 2008-12-17

Motivation

- Gap between security mechanisms and different abstraction layers
  - Facebook example: only friends can see profile
- Reflect upon data vs. individual operations
  - Take primitive mechanism that fits all
- Need for small TCB, that ensures certain security policies at runtime, as verification is hard (example: Anti-Virus scanner)

Solution: tagged memory

- Additional memory protection mechanism in hardware
- Word-granular security tags for memory, enforced by the CPU
- Hardware extensions are used in new mode



Starting Point: HiStar

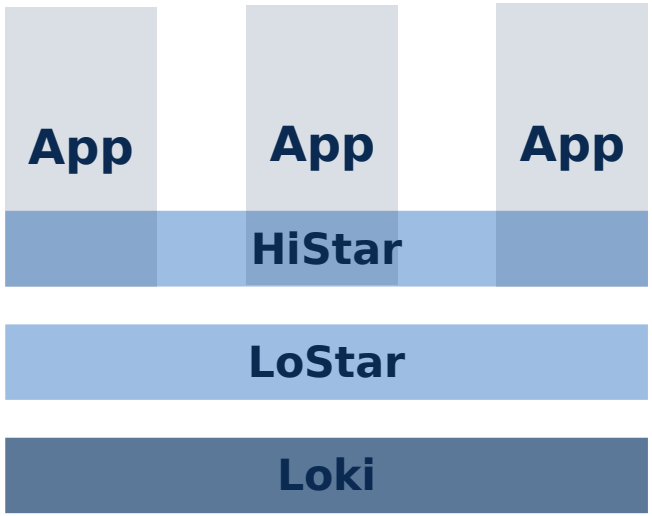
- HiStar is a kind of micro-kernel
  - Inspired by AsbestOS's labels
  - Decentraliced information flow control
  - In contrast to AsbestOS labels are altered explicitly
  - Provides Unix environment that translates Unix implicit policies to HiStar's labeling mechanism (remember Flume: Linux enhanced by DIFC)

HiStar labeling mechanism

- Each kernel object has a *label*:
  - threads, address spaces, segments, gates, containers, devices
- Label is a set of *categories* (61-Bit ID) and taint levels
- Thread's label:
  - Shows degree of information contamination, regarding specific categories
  - Can be extended explicitly up to specified *clearance*
- Gates have an associated label and clearance

Tagged memory and LoStar

- Using hardware memory tags to divide kernel into thread domains
- New super-supervisor mode running small kernel subset, that drives tagging hardware



LoStar's protection

- Twofold memory tags page-wide or word-wide
- Integrity protection of all kernel object's labels and reference counters and the global object hash table by word-tags
- Monitor call API:
  - Thread switch (changes permissions)
  - Allocate or free kernel object (set or free label)
  - IPC by invoking a Gate (label check)
  - Increment or decrement object's refcounter
  - Adjusting the thread's label
  - ...

Cruel thing: Devices

- Interrupts don't change (non-)monitor mode
- Interrupt handling code and most device handling code resides in security monitor
- Interrupts received by non-monitor world need to be delivered to monitor!
- Device driver code that deals with various tags (like disk driver) will remain in trusted code part

Loki

- Associates 32-bit tag for each 32-bit word in physical memory
- Page-tag array indexed by frame address contains tags, is cached by TLB like tag-cache
- Page-tag array entries have a special indirection-bit to indicate word-granular tags for that page
- Additional P-cache contains tag-values and permissions of running thread
- P-cache misses are handled by monitor

Loki prototype

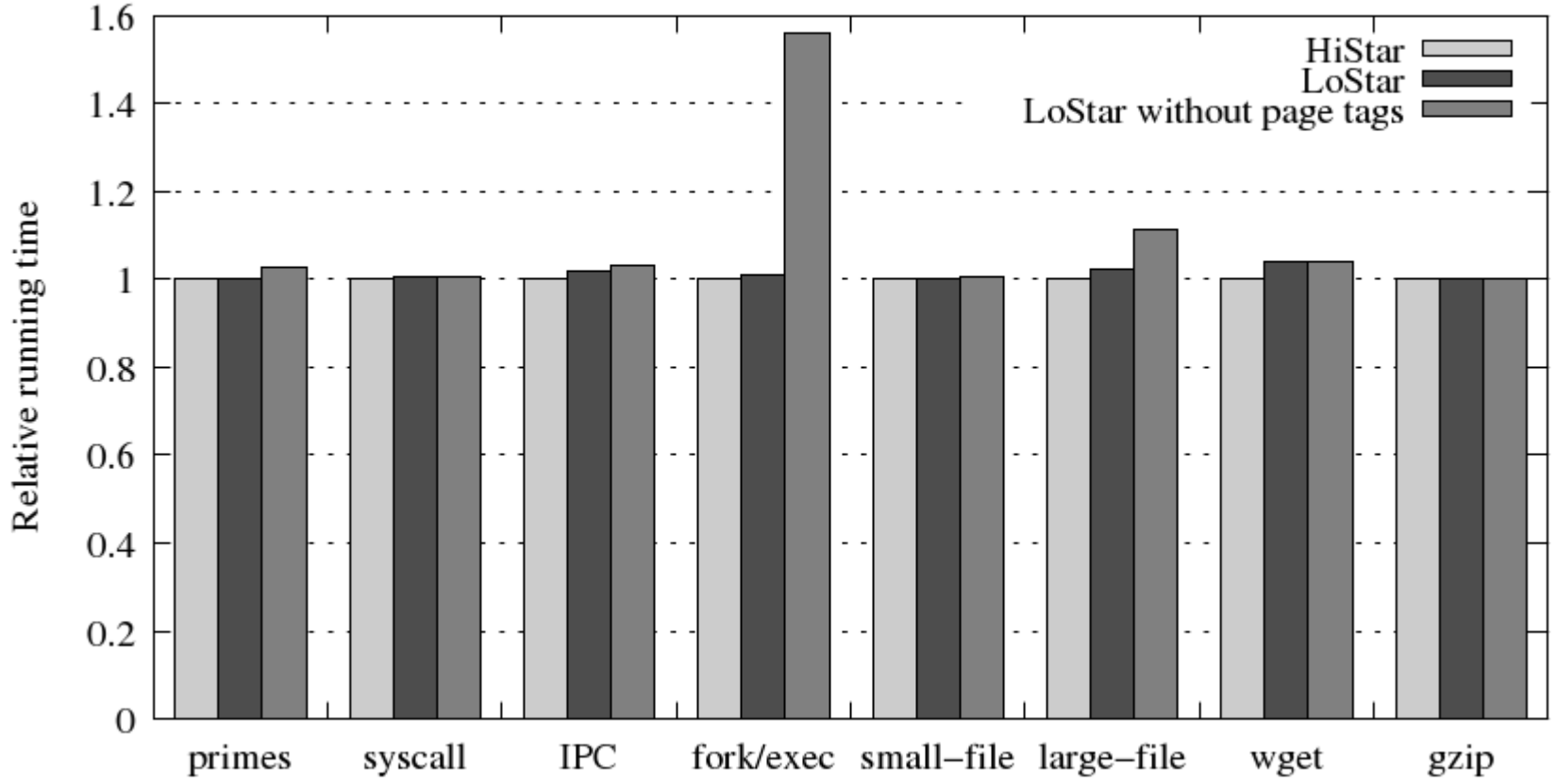
- Leon SPARC V8 (FPGA core) 65 MHz
- Added:
  - 6 instructions (e.g.: P-cache handling)
  - 7 registers (e.g.: faulting tag value)
  - 32 entry 2-way associative P-cache
  - 8 entry full-associative tag-TLB
- P-cache gets accessed at least once by each instruction !!!

Evaluation: TCB

- Trusted code base shrinks?
- They didn't state exactly what is meant by trusted
- To be sure that my applictions policy is enforced by the monitor + Loki I need to trust the whole kernel

Lines of code	HiStar	LoStar
Kernel code	11,600 (trusted)	12,700 (untrusted)
Bootstrapping code	1,300	1,300
Security monitor code	N/A	5,200 (trusted)
TCB size: *trusted code*	**11,600**	**5,200**

Performance



Conclusion / Discussion

- What do they won? TCB size reduction?
- What about 'more realistic hardware and the supposable greater overhead?
- Singularity went in the opposite direction and left out hardware memory protection at all, who is right?
- Is there one security mechanism (like labels) that fits all, or do we need the different layers (example: timing issues, resource accounting ...)

Loki pipeline