Threats

Speculation

An “unknown unknown” until recently

Microarchitectural Timing Channel

A “known unknown” for decades
What Are Timing Channels?
Timing Channels

Information leakage through timing of events

- Typically by observing response latencies or own execution speed

**Covert channel:** Information flow that bypasses the security policy

**Side channel:** Covert channel exploitable without insider help

Victim executes normally

Attacker observes
Cause: Competition for Shared HW Resources

- Inter-process interference
- Competing access to micro-architectural features
- Hidden by the HW-SW contract!

Affect execution speed
Preventing Timing Channels
Confidentiality Needs **Time Protection**

Traditionally OSes enforce security by *memory protection*, i.e. enforcing spatial isolation.

**Time protection**: A collection of *OS mechanisms* which collectively *prevent interference* between security domains that make execution speed in one domain dependent on the activities of another.

[Ge et al. EuroSys’19]
**Time Protection: Partition Hardware**

- **Temporally partition**
  - Cannot spatially partition on-core caches (L1, TLB, branch predictor, pre-fetchers)
    - virtually-indexed
    - OS cannot control

- **Flush**
  - Need both!
    - Flushing useless for concurrent access
      - HW threads
      - cores

- **Spatially partition**
  - Virtually indexed
  - OS cannot control
Requirements for Time Protection

Timing channels can be closed *iff* the OS can
- (spatially) partition or
- reset
all shared hardware
Sharing 1: Stateless Interconnect

H/W is bandwidth-limited
• Interference during concurrent access
• Generally reveals no data or addresses
• Must encode info into access patterns
• Only usable as covert channel, not side channel

No effective defence with present hardware!
Sharing 2: Stateful Hardware

HW is capacity-limited
- Interference during
  - concurrent access
  - time-shared access
- Collisions reveal addresses
  - Usable as side channel

Any state-holding microarchitectural feature:
- cache, branch predictor, pre-fetcher state machine

Solvable problem – focus of this work
Implementing Time Protection on Stateful Hardware
Spatial Partitioning: Cache Colouring

- Partitions get frames of disjoint colours
- seL4: userland supplies kernel memory ⇒ colouring userland colours dynamic kernel memory
- Per-partition kernel image to colour kernel [Ge et al. EuroSys’19]
1. \( T_0 = \text{current\_time()} \)
2. Switch user context
3. Flush on-core state
4. Touch all shared data needed for return
5. \( \text{while} \ (T_0 + \text{WCET} < \text{current\_time}()) \);
6. Reprogram timer
7. return

Latency depends on prior execution!

Must remove any history dependence!

Ensure deterministic execution

Time padding to Remove dependency
Reality Check: Flushing On-Core State
Mitigation on Intel and Arm processors:
• Disable data prefetcher (just to be sure)
• On context switch, perform all architected flush operations:
  • Intel: wbinvd + invpcid (no targeted L1-cache flush supported!)
  • Arm: DCCISW + ICIALLU + TLBIALL + BPIALL
Methodology: Prime and Probe

1. Fill cache with own data
2. Touch $n$ cache lines
3. Traverse cache, measure execution time
Methodology: Channel Matrix

Channel Matrix:
- Conditional probability of observing time, $t$, given input, $n$.
- Represented as heat map:
  - bright = high probability

Horizontal variation indicates channel
I-Cache Channel With Full State Flush

CHANNEL!

CHANNEL!

No evidence of channel

SMALL CHANNEL!

Intel Sandy Bridge

Intel Haswell

Intel Skylake

HiSilicon A53
HiSilicon A53 Branch History Buffer

Branch history buffer (BHB)
- One-bit channel
- All reset operations applied

Channel!
Found residual channels in all recent Intel and ARM processors examined!

Branch target buffer
• All reset operations applied
Intel added *indirect branch control* (IBC) feature, which closes most channels, but…

Intel Skylake Branch history buffer

Requirements on Hardware
New HW/SW Contract: aISA
Augmented ISA supporting time protection

For all shared microarchitectural resources:

1. Resource must be spatially partitionable or flushable
2. Concurrently shared resources must be spatially partitioned
3. Resource accessed solely by virtual address must be flushed and not concurrently accessed
   • Implies cannot share HW threads across security domains!
4. Mechanisms must be sufficiently specified for OS to partition or reset
5. Mechanisms must be constant time, or of specified, bounded latency
6. Desirable: OS should know if resettable state is derived from data, instructions, data addresses or instruction addresses
THANK YOU

Gernot Heiser | gernot@unsw.edu.au | @GernotHeiser

https://trustworthy.systems