Side-Channel Attacks & Transient Execution Vulnerabilities

Allison Randal
Rivos Inc.

Overview

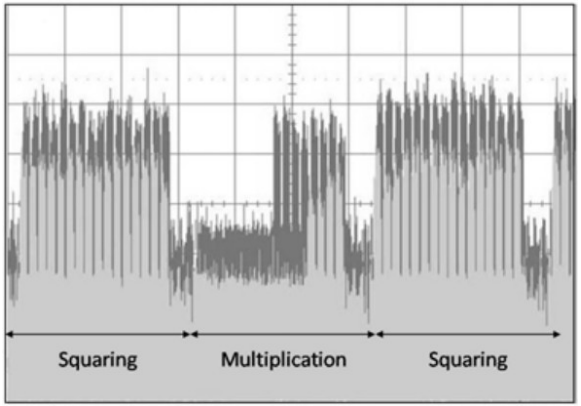
- Physical side-channel attacks (background)
- Microarchitecture side-channel attacks
- RISC-V Microarchitecture Side-Channels groups
- Aims:
  - General understanding of the classes of vulnerabilities, the hardware design features that make them possible, and options for countermeasures
  - Explain terms and concepts along the way

Terminology: “Covert” vs “Side” Channels

- Covert-channel attacks: use a hardware resource not designed to transfer information as a communication channel, to bypass isolation mechanisms (first published in the mid-1970s)
  - Communication is **intentional**
  - Sender and receiver are **both malicious** (sometimes called “Trojan” and “Spy”)
- Side-channel attacks: attacker observes hardware resources used by victim to access or infer victim’s private data
  - Communication is **unintentional**
  - Sender is **victim**, receiver is **malicious**
- They use some of the same channels, but they aren’t the same thing

Physical side-channel attacks (background)

- First published in the mid-1990s
- Exploit indirect physical information to extract secrets, such as cryptographic keys
- Power Analysis
  - Technique: measure power usage related to encryption/decryption for different inputs/outputs, infer secret information from variations in power consumption
  - Countermeasures: change logic to equalize power consumption (hiding) or to add random noise (masking)



Power traces as part of RSA decryption, from “Simple Power Analysis on Exponentiation Revisited” (2010) https://doi.org/10.1007/978-3-642-12510-2_6

Physical side-channel attacks (background)

- Electromagnetic Analysis
  - Technique: measure electromagnetic waves produced by current flow over device, infer secret information from variations in EM signals
  - Countermeasures: EM shielding, add EM noise, or change logic to reduce electric/electromagnetic coupling

Physical side-channel attacks (background)

- Fault Analysis
  - Physical fault-injection attack (integrity), used as a source of information for physical side-channel attacks (confidentiality)
  - Technique: physically tamper with voltage levels, clock signal, etc. to trigger a fault in the device (e.g. disturb a few memory or register bits), infer secret information based on variations in the output of faulty operations
  - Countermeasures: replication of critical operations, error-detection mechanisms, anti-tamper protection modules

Physical side-channel attacks (background)

- Timing Analysis
  - Technique: measure execution time of operations for different inputs, infer secret information from variations in timing (usually combined with other physical side-channel attacks)
  - Countermeasures: change logic to equalize execution time (“constant-time”) or add random delays
- More obscure physical side-channel analysis: sound, temperature, vibration

Terminology: “Physical” vs “Microarchitectural” Channels

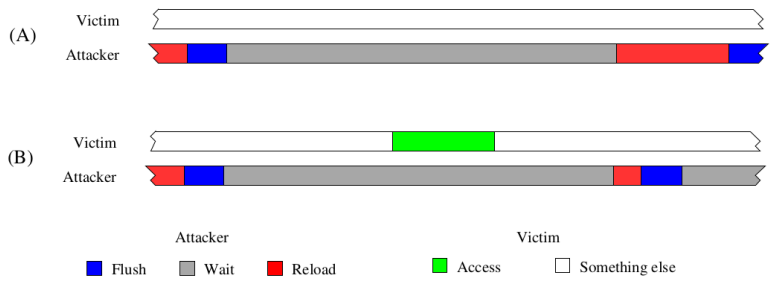
- Physical side-channel attacks:
  - Exploit indirect physical information to extract secrets
  - Require physical access or proximity to the device (harder)
- Microarchitecture side-channel attacks:
  - Exploit indirect microarchitectural information to extract secrets
  - No physical access required, may be software-induced (easier)
- Information sources are different, analysis and objectives are similar
- Countermeasures for either have costs in performance, power, or die area

Microarchitecture side-channel attacks

- Microarchitecture resources used as channels: cache timing, TLB, page tables, DRAM, prefetchers, branch predictors, FPU timing, SMT port contention, CPU frequency, etc.
- A few representative examples, to walk you through the concepts:
  - Cache-Timing Analysis
  - RAMBleed
  - Meltdown
  - Spectre
  - SpecHammer

Cache-timing side-channel attacks

- Active area of work since the mid-2000s
- Technique:
  - Attacker establishes a pre-defined cache state
  - Allow the victim to perform an operation
  - Infer information about the victim based on cache state changes
- Variants:
  - By attacker action: Prime+Probe, Evict+Time, Flush+Reload, Flush+Flush, etc.
  - By cache: L1 requires victim/attacker on the same core, but not LLC
- Countermeasures: partition caches (statically or dynamically), flush caches, randomize cache access time (eviction or add noise), detection (signature or anomaly)



Flush+Reload, from “FLUSH+RELOAD: a High Resolution, Low Noise, L3 Cache Side-Channel Attack” (2014) https://doi.org/10.5555/2671225.2671271

RAMBleed

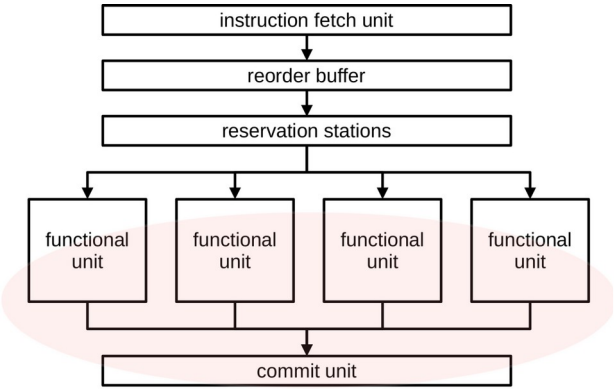
- Published in 2020
- Technique:
  - Use Rowhammer to flip bits in memory (software-induced fault attack)
  - Infer secret information (one bit at a time) based on success of bit flip
- Countermeasures: memory encryption, flushing cryptographic keys from memory, probabilistic memory allocator

Terminology: “Out-of-Order” vs “Speculative” Execution

- Out-of-order execution is a microarchitecture optimization that parallelizes fetch, decode, execute, and write-back stages of the pipeline, allowing instructions to execute when their operands are available
  - Executes the exact same instructions as in-order, just in a flexible order
- Speculative execution goes one step further, and makes predictions about control flow and memory access
  - May execute instructions/operands that an in-order microarchitecture never would
- Both retire/commit instructions in-order, supposedly cleaning up after themselves, so optimizations should be architecturally invisible … but ...

Terminology: “Transient” Execution

- Transient execution describes the “limbo” for both out-of-order and speculative execution:
  - Before instructions retire/commit (and clean up)
  - Before exceptions are raised
  - Before it’s known whether predictions were accurate



Transient execution vulnerabilities – zone of risk, from “Ghosting the Spectre: fine-grained control over speculative execution” (2021)

- False assumption: out-of-order/speculative execution can safely execute transient instructions and create transient microarchitectural state, as long as it’s cleaned up on commit, not architecturally visible
- Reality: transient instructions and microarchitectural state can leak secret information and run arbitrary code gadgets

Meltdown

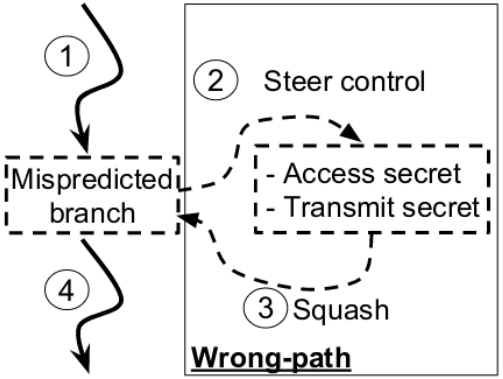
- Discovered in 2017, published in 2018
- One of new class of transient execution vulnerabilities
- Affects both out-of-order and speculative microarchitectures
- Technique:
  - Attempt to access an unauthorized value
  - Transient exception is not raised immediately, delayed until retirement
  - During the delay, if transient microarchitectural state (e.g. cache) has been created for unauthorized value, it can be leaked using existing side-channel attacks
  - On retirement, exception is raised, and microarchitectural state (may be) cleaned up, but too late, the damage is done

Meltdown

- Variants:
  - By exception: page fault, general protection fault, device-not-available, bound range exceeded
  - By target: out-of-bounds access, kernel memory, read-only memory, privileged system registers, floating point/SIMD registers, memory-protection keys, TEE or VM memory
- Countermeasures: many proposed, but best is to delay updating microarchitectural state until permission checks are complete

Spectre

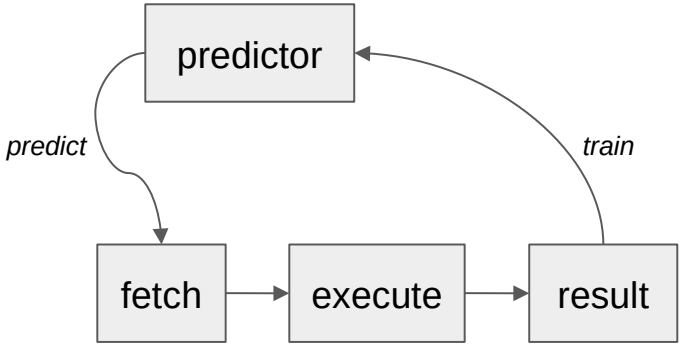
- Discovered and published at the same time as Meltdown
- Also in the class of transient execution vulnerabilities
- Affects only speculative microarchitectures
- Technique:
  - Mistrain a predictor
  - Allow victim to run with bad predictions, creating transient microarchitectural state
  - Infer or access secret information from microarchitectural state, using existing side-channel attacks (like Flush+Reload or Prime+Probe)



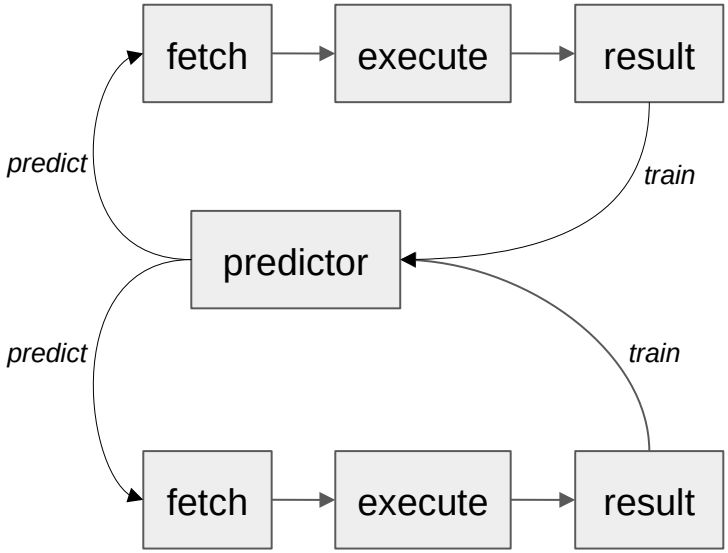
Transient control steering, from “NDA: Preventing Speculative Execution Attacks at Their Source” (2019) https://doi.org/10.1145/3352460.3358306

Mistraining a predictor

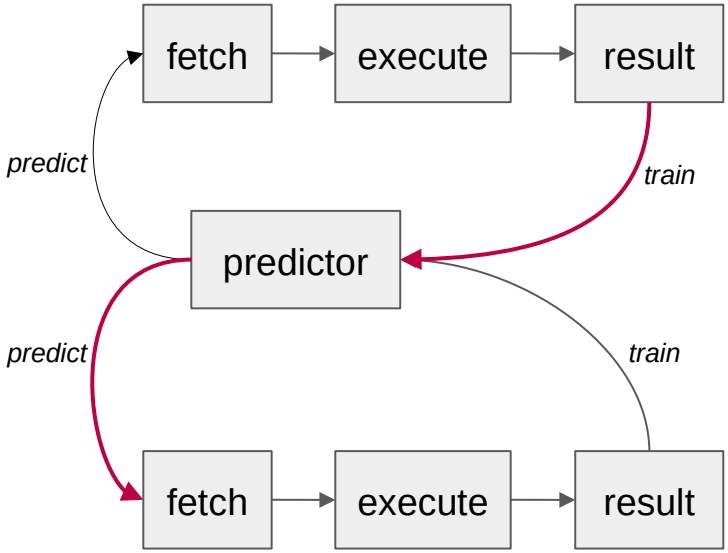
Mistraining a predictor



Mistraining a predictor



Mistraining a predictor



Spectre

- Variants:
  - By predictor: conditional branch, direct and indirect branch, memory load/store, return address, cache way
  - By secondary attack vector: redirect transient control flow to arbitrary destination, trigger a transient unauthorized/invalid load or store, divert return control flow to code gadget, etc.
  - By side channel: mostly cache, but also TLB, DRAM, vector instruction timing, SMT port contention, and Branch Target Buffer (BTB)
  - By target: reading or writing out-of-bounds, reading stale cache values, steal secrets from protected TEEs, sandbox escape, breaking type and memory safety guarantees, reading entire user memory or entire kernel memory, even remotely over the network, etc.

Spectre

- Countermeasures:
  - Isolate or flush predictions, speculation-barrier instructions, speculative taint tracking, delayed execution
  - Many proposed mitigations only address some variants and most have prohibitive performance penalties
  - Promising new approach, SpecTerminator[1], combines taint tracking, instruction masking, and delayed execution, only 2.6% performance penalty for cache/TLB/DRAM variants, 6% for all variants

[1]"SpecTerminator: Blocking Speculative Side Channels Based on Instruction Classes on RISC-V" (2022) https://doi.org/10.1145/3566053

SpecHammer

- Published in 2022
- Technique:
  - Use Rowhammer to flip bits in the victim gadget of a Spectre-PHT attack
  - Modifies victim code that wouldn’t work as a gadget (no attacker-controlled offset variable) to make it a viable as a Spectre-PHT gadget
- Countermeasures: some existing mitigations for Spectre-PHT and Rowhammer apply, but SpecHammer defeats taint tracking mitigations
- Compare: SpecHammer uses fault-injection attack to amplify side-channel attack (confidentiality) vs GhostKnight uses speculation to amplify the Rowhammer fault-injection attack (integrity) across privilege boundaries

Security verification tools

- Side-channel vulnerabilities are complex, and non-functional behaviors are difficult to reason about
- Mistakes have been made, including mitigations (shipped in production hardware) that didn’t actually deliver the protection promised
- Don’t trust that your design is right, verify
- Tools are available to help
  - Some commercial (easier to use)
  - Some academic (generally proof-of-concept, but over time techniques are integrated into commercial tools)

RISC-V Microarchitecture Side-Channels (uSC) SIG

- Meetings: alternate Tuesdays (on hold for holiday season)
- Mailing list: https://lists.riscv.org/g/sig-uarch-side-channels/
- Notes, Slides, and Docs: https://github.com/riscv-admin/uarch-side-channels
- New task group Microarchitecture Side-Channel Resistant Instruction Spans (uSCR-IS) TG
  - Builds on concepts from *fence.t*[1] and *dome*[2] approaches to microarchitecture side-channels
- Also interested in exploring verification tools

[1]"Microarchitectural Timing Channels and their Prevention on an Open-Source 64-bit RISC-V Core" (2021) https://doi.org/10.23919/DATE51398.2021.9474214
[2]"Under the dome: preventing hardware timing information leakage" (2021) https://doi.org/10.1007/978-3-030-97348-3_13

Further Reading

- “Opening Pandora's Box: A Systematic Study of New Ways Microarchitecture Can Leak Private Data” (2021) by Jose Rodrigo Sanchez Vicarte, Pradyumna Shome, Nandeeka Nayak, Caroline Trippel, Adam Morrison, David Kohlbrenner, Christopher W. Fletcher, https://doi.org/10.1109/ISCA52012.2021.00035
- “A Survey of Microarchitectural Side-channel Vulnerabilities, Attacks, and Defenses in Cryptography” (2021) by Xiaoxuan Lou, Tianwei Zhang, Jun Jiang, and Yinqian Zhang, https://doi.org/10.1145/3456629
- “Hardware Security: A Hands-On Learning Approach” (2019) by Swarup Bhunia and Mark Tehranipoor, https://doi.org/10.1016/C2016-0-03251-5