RISC-V Summit China 2024 (RISC-V 中国峰会 2024)

A Study on Transient Execution Vulnerabilities of RISC-V Implementations (RISC-V 实现的瞬态执行漏洞研究)

Tuo Chen (陳 拓) (potato008@gmail.com)
Suzaki Lab (須崎研究室)
Institute of Information Security (IISEC), Japan

2024-08-23

Self introduction

INSTITUTE of INFORMATION SECURITY

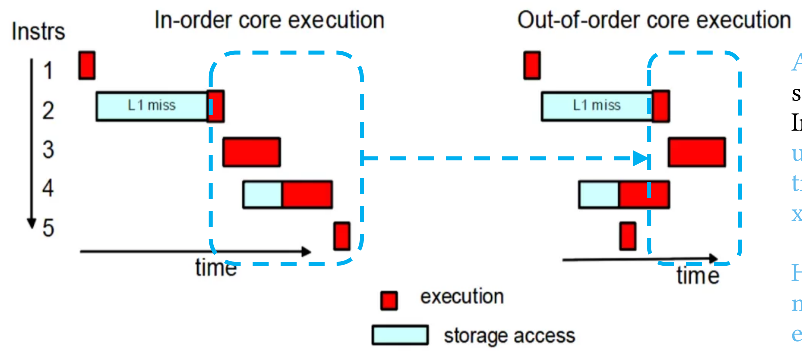
- About me
  - Master student Tuo Chen (陳 拓)
  - IISEC (2023~): information security of open hardware
  - Renesas Electronics group (2017~2023): prototype evaluation, device test, mass production setup for semiconductor tests, export procedures, etc.
  - NUAA, CEIE (2013~2017): electronic circuit, microwave systems and devices
- About IISEC Suzaki Lab: https://lab.iisec.ac.jp/~suzaki_lab/index-e.html
  - Prof. Kuniyasu Suzaki （須崎有康教授）
  - Research topics: RISC-V, TEE, virtualization, confidential computing, etc.
  - Currently other members are researching on: FPGA cryptography applications, malicious activity statistics, fuzzing for security purposes, vehicle cyber security, infosec of IoT devices, confidential computing + TEE

Contents

INSTITUTE of INFORMATION SECURITY

Background (1)

- Out-of-order (OoO) execution
  - Paradigm that allows subsequent instructions in the pipeline to be executed ahead of or concurrently with preceding ones, rather than strictly adhering to program order (=> *in-order execution*).
  - A OoO CPU temporarily stores executed instructions in the reorder buffer, and later adjusts the order in which they are reflected in the registers during the retire stage, thereby achieving the same results as an in-order processor.
  - Mainstream x86 CPUs and some ARM processors have adopted OoO execution. Commercial RISC-V OoO core designs are still relatively few, but their numbers are growing rapidly.



Source: https://www.techspot.com/article/2000-anatomy-cpu/

Ref: [1] S. Hily and A. Seznec, 1999

A OoO CPU can execute Instrs No.4 and No.5 ahead of Instrs No.2 and No.3, without waiting for their completion, reducing the overall execution time.

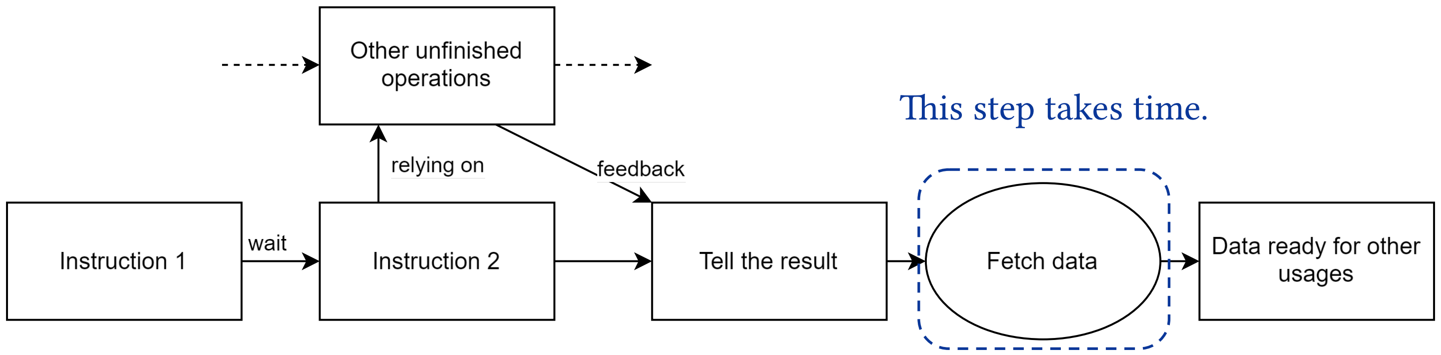
However, in terms of implementation cost-performance, OoO processors do not necessarily have the advantage. (See Ref)

Some known RISC-V OoO cores

Name	Maintainer	Category
AX65	Andes	Commercial
BOOM	UC Berkeley	Academic
C910, C920	XuanTie	Commercial
P550, P670	SiFive	Commercial
RSD	U Tokyo	Academic
Tooba	U Cambridge	Academic
Xiangshan	BOSC	Academic

Background (2)

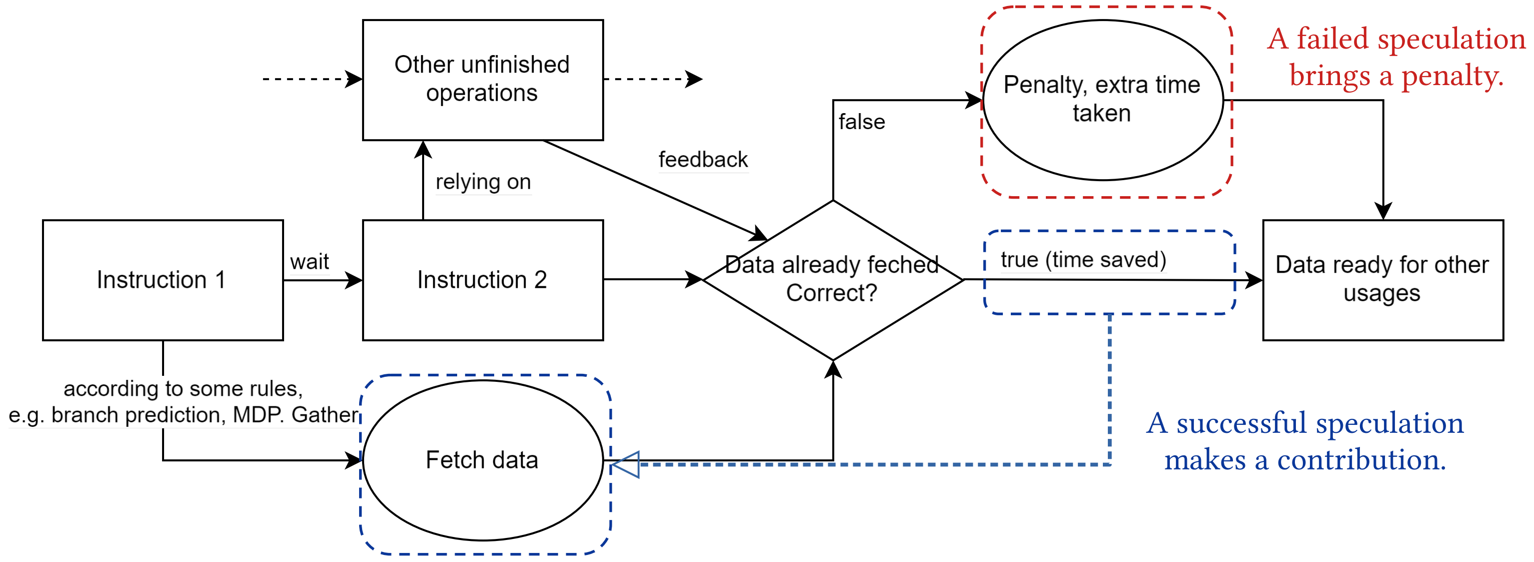
- Speculative execution
  - The premise of OoO execution is the ability to recognize the program's flow. Obviously, waiting, or selecting the program's direction randomly/evenly, is inefficient. In contrast, determining it as early as possible is beneficial.
  - Retrieving and processing information that "might be needed" based on certain grounds (e. g. information from *predictors*), before the definite results come, is called *speculative execution.*
  - Speculative execution requires both *parallel processing* and OoO. Specific implementation methods are diverse. Examples include *branch* prediction, *memory dependence* prediction, *value* prediction, *Gather* instruction, etc.



Flow of a *non-speculative* execution

Background (3)

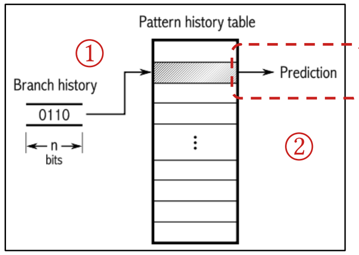
- Speculative execution (cont.)
  - In speculative execution, if a program instruction depends on an incomplete operation, the CPU first saves the current register state as a checkpoint. Then, it tentatively executes the state-undetermined instruction using the various methods mentioned before.
    - If the trial result is correct, it saves execution time, and the instruction is retired as usual.
    - If the trial result is incorrect, a rollback is necessary. The incorrect trial results or in-progress instructions are discarded, the state saved at the checkpoint is restored, and the correct results are obtained anew.

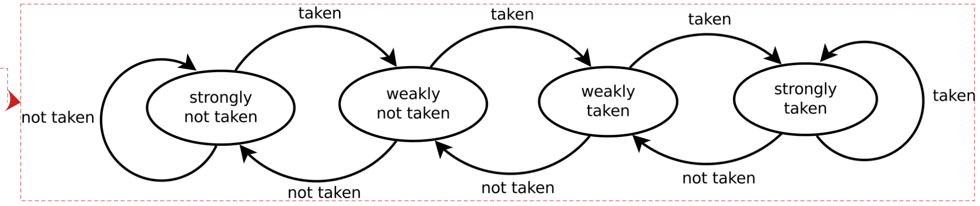

Other unfinished operations
relying on
feedback
false
Penalty, extra time taken
A failed speculation brings a penalty.
Instruction 1
wait
Instruction 2
Data already feched Correct?
true (time saved)
Data ready for other usages
according to some rules, e.g. branch prediction, MDP. Gather
Fetch data
A successful speculation makes a contribution.

Flow of the *speculative* execution

Background (4)

- Branch prediction
  - One method of improving speculative execution. By predicting the most likely outputs and increasing the number of correct instructions executed, performance is improved. Almost all RISC-V OoO CPUs have adopted this.
  - BPU (Branch Prediction Unit) utilizes various CPU components:
    - **PHT (Pattern History Table), BHT (Branch History Table), CBP (Conditional Branch Predictor):** Separate *local* history buffers for handling *conditional* branches with two possible directions. 2-level branch prediction is often used. The 2 levels are: ① remembering the patterns that occurred in the last n times, and ② selecting a prediction value from these records using a 2-bit counter (a state machine with 4 possible outcomes).
    - **BTB (Branch Target Buffer):** A *global* history buffer for *direct* and *indirect* branches. It is not limited to two choices and can remember frequently used destination addresses of *jump*.
    - **RSB (Return Stack Buffer), RAS (Return Address Stack):** The most frequently used part of the *software call stack* (*return* addresses) is copied to and stored in this *hardware buffer*.
    - **LP (Loop Predictor**, found in C910, C920), **FTB (Fetch Target Buffer**, an alternative of the BTB found in Xiangshan Nanhu), and **hybrid branch predictors** of ones above.





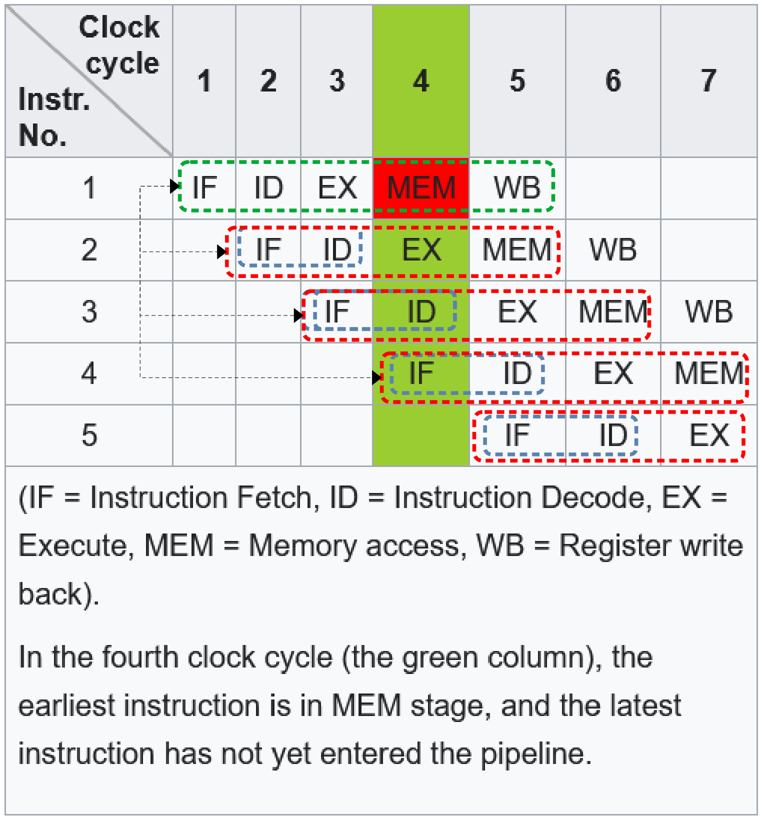
In 2-level prediction, each branch has 4 states. Why not simpler 1-level 2 states (taken vs. not taken)? A 4-state machine is stabler, less likely to change its prediction value after just one change in the output, thus increasing the accuracy.

Source: https://en.wikipedia.org/wiki/Branch_predictor

Background (5)

- Branch prediction (cont.)
  - BP is not unique to OoO CPUs; it can also be implemented in in-order processors. Although the latter do not perform speculative execution, it is still effective for improving data retrieval speed (speculative *prefetching*).

Basic five-stage pipeline



Instr. No. \ Clock cycle	1	2	3	4	5	6	7
1	IF	ID	EX	MEM	WB		
2		IF	ID	EX	MEM	WB	
3			IF	ID	EX	MEM	WB
4				IF	ID	EX	MEM
5					IF	ID	EX

(IF = Instruction Fetch, ID = Instruction Decode, EX = Execute, MEM = Memory access, WB = Register write back).

In the fourth clock cycle (the green column), the earliest instruction is in MEM stage, and the latest instruction has not yet entered the pipeline.

Source: https://en.wikipedia.org/wiki/Instruction_pipelining#Branches

- Speculative prefetching of in-order processors:
  - The CPU fetches (IF) the next instruction to the Icache, and decodes (ID) it, but does not proceed to execution (EX). Naturally, it also does not reach memory access (MEM).

- Speculative execution of out-of-order processors:
  - Complex hardware mechanisms determine the dependencies between instructions and executes (EX) them based on CPU resource availability. In the MEM step, information is loaded from main memory into the Dcache.

Background (6)

- Memory dependence prediction
  - Another method of improving speculative execution. An OoO processor with a *memory dependence predictor* (MDP) predicts whether two memory operations that access the same memory location (such as a preceding *Store* op and a subsequent *Load* op) interact with each other.
  - By far, Xiangshan and RSD have been observed to integrate MDPs with dynamic speculation and synchronization such as store sets, store barrier, etc.
  - Speculative memory operations require the CPU's load/store unit (LSU) can accomplish *memory disambiguation* (addressing ordering violations). For that purpose, a specific MDP is an early assistance, but *not* a precondition. There are RISC-V OoO CPUs, such as BOOMv3, that do not possess an MDP but still allow naive memory dependence speculations, i.e. a load may bypass any preceding store.

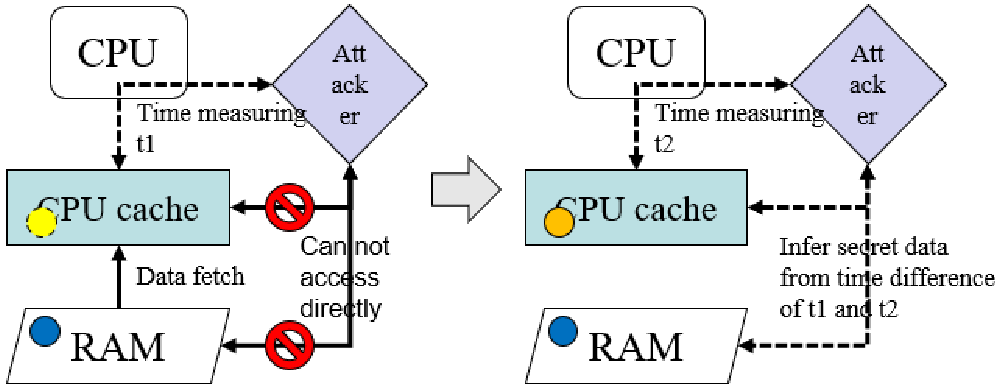
Note: memory dependencies:

Read-After-Write (RAW), "true dependencies":	Write-After-Read (WAR), "anti dependencies":	Write-After-Write (WAW), "output dependencies":
X2:=X1+X3	Y4:=Y1+Y3	Z2:=Z1+Z3
X4:=X2+X3	Y3:=Y1+Y2	Z2:=Z4+Z5

Cache timing side-channel attack (SCA) (1)

- Cause
  - Registers and main memory transfer data through the CPU's cache. The cache holds a portion of the main memory's contents, allowing for a quick response when data transfer requests are made again.
  - For a failed speculative execution, the CPU performs a rollback. The purpose is to ensure that, from the perspective of the *architecture*, the overall state of the device is as if “the erroneous execution never occurred”.
  - However, the *micro-architectural* physical behavior of the cache is not canceled. Even if the secret data can not be directly read out, it may still be loaded into the cache, resulting in much shorter cache access time *t2*, compared to the relatively longer RAM access time *t1*.

- If the trace is not concealed and can be observed, an attacker may infer the secret data by measuring difference *t1 – t2*, creating a vulnerability.
- There are many specific cache timing SCA techniques.

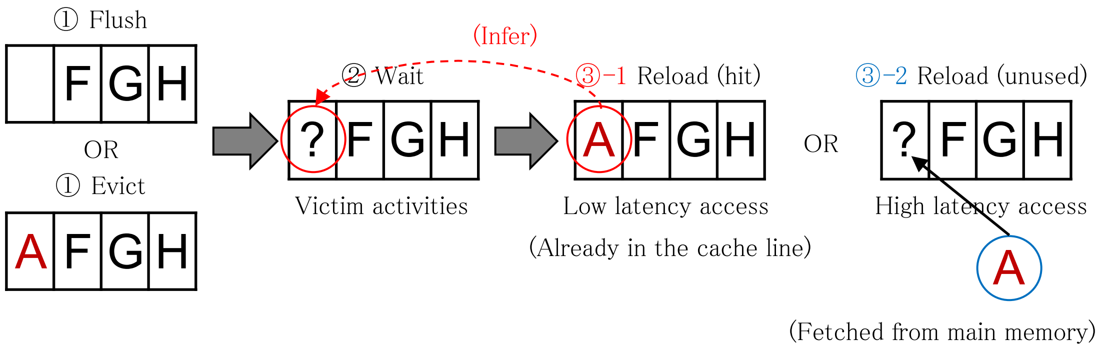


Process of the cache timing SCA

Cache timing side-channel attack (SCA) (2)

INSTITUTE of INFORMATION SECURITY

- Techniques
  - Flush[2]/Evict[3]+Reload, Flush+Flush[4], Flush/Evict+Fault[6]
  - Evict+Time[5], Cache+Time[6]
  - Prime+Probe[5], Prime+Count[6]
  - Simple branch prediction analysis[6]
  - Branch shadowing[6]
  - CycleDrift[6] => a RISC-V-specific attack technique using *rdinstret()*
  - …



Example: Flush/Evict+Reload

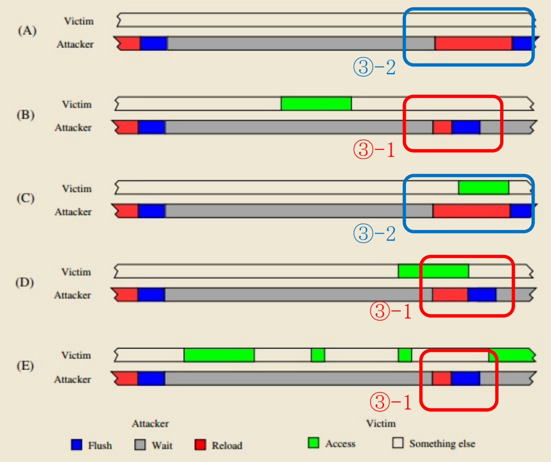
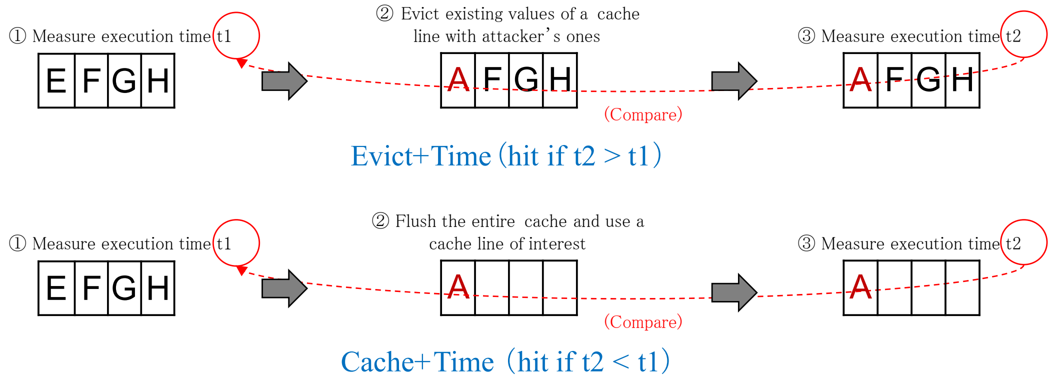


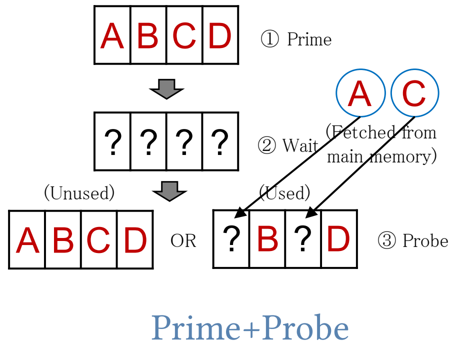
Figure 3: Timing of FLUSH+RELOAD. (A) No Victim Access (B) With Victim Access (C) Victim Access Overlap (D) Partial Overlap (E) Multiple Victim Accesses

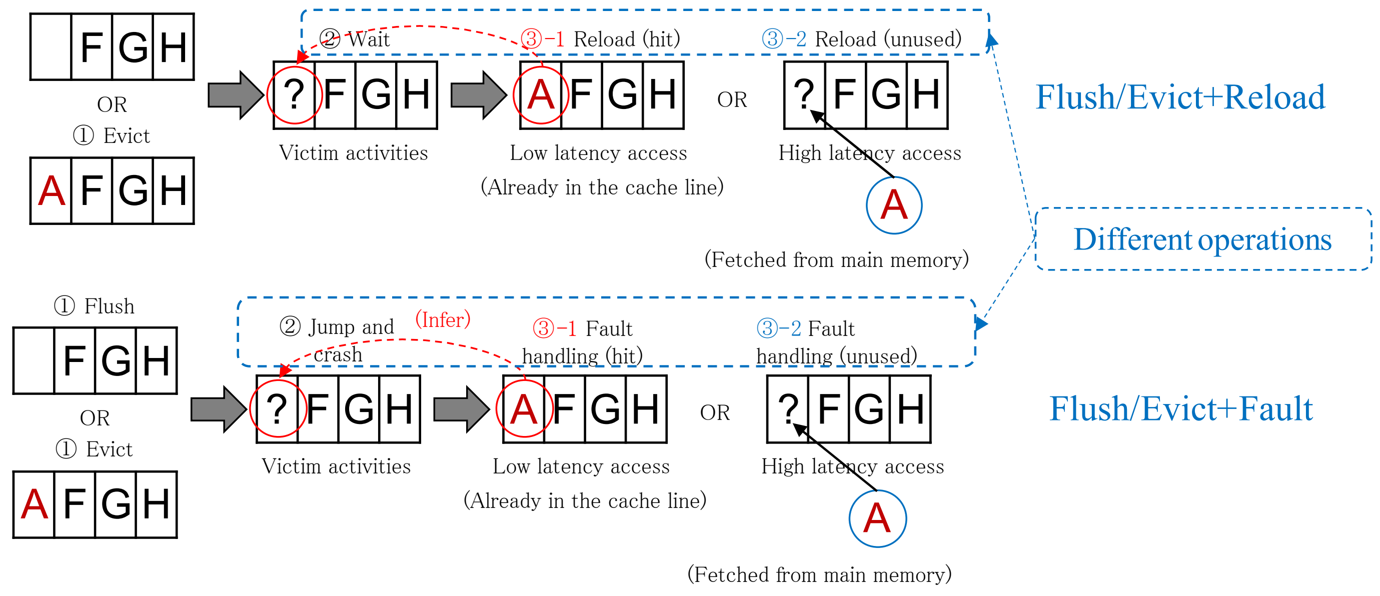
[2] Y. Yarom and K. Falkner, USENIX Security 14, 2014, pp. 719–732.
[3] P. Kocher et al., IEEE S&P 2019, San Francisco, CA, USA: May 2019, pp. 1–19.
[4] D. Gruss, C. Maurice, K. Wagner, and S. Mangard, DIMVA 2016. Jul. 2016, pp. 279–299.
[5] D. Gruss, R. Spreitzer, and S. Mangard, USENIX Security 15, 2015, pp. 897–912.
[6] L. Gerlach, D. Weber, R. Zhang, and M. Schwarz, IEEE S&P 2023, May 2023, pp. 2321–2338.

Cache timing side-channel attack (SCA) (3)

- Techniques (cont.)


① Measure execution time t1
E F G H
② Evict existing values of a cache line with attacker's ones
A F G H
(Compare)
③ Measure execution time t2
A F G H
Evict+Time (hit if t2 > t1)
① Measure execution time t1
E F G H
② Flush the entire cache and use a cache line of interest
A
(Compare)
③ Measure execution time t2
A
Cache+Time (hit if t2 < t1)





Transient execution vulnerabilities (1)

- Definition
  - A security issue where a device's *transient execution* state is exploited to indirectly infer secret data within access-restricted memory areas. It includes three successive phases[7]: an initial *preparation* phase, a later *access* phase, plus a final *transmission* phase.

- Categories
  - *Spectre*-type (*speculation*-based fault injection + SCA): BCB, BTI, ret2spec, SSB, Retbleed, Inception, BHI, GDS, SCO, ZDI, etc.
  - *Meltdown*-type (*exception*-based fault injection + SCA): RDCL, LazyFP, Foreshadow, Fallout, etc.
  - Others (different combinations and purposes, such as double fault injections): LVI, GVI, etc.

Spectre

Meltdown

MDS

LVI

Downfall (GDS, GVI)

iLeakage

CROSSTalk, Retbleed, INCEPTION

... and more

[7] Allison Randal, "This is How You Lose the Transient Execution War," Sep. 06, 2023, arXiv: arXiv:2309.03376.

Transient execution vulnerabilities (2)

- Summary of review papers
  - From 2018 when Spectre and Meltdown were revealed, until 2024-08, more than *28 general surveys* (not counting individual technical proposal papers) regarding *transient execution attacks* (TEAs) and their countermeasures have been published. Since RISC-V is still an emerging ISA, it is not included in the scope of discussion in most reviews.
  - Among them, I recommend some milestone RISC-V+TEA-related survey papers for a careful reading.

Allison Randal, "This is How You Lose the Transient Execution War," Sep. 06, 2023, arXiv:2309.03376.
This is the latest review and a novel inspection into mechanisms of TEAs. Thanks to the author Ms. Randal as a leader in the OSS world, RISC-V also receives her attention in this paper.

L. Gerlach, D. Weber, R. Zhang, and M. Schwarz, "A Security RISC: Microarchitectural Attacks on H ardware RISC-V CPUs," in IEEE S&P 2023, San Francisco, CA, USA: May 2023, pp. 2321–2338.
This paper demonstrates many cache timing SCAs (including 3 newly proposed techniques) against RISC-V *in-order* processors. It also provides 6 case studies of high practicalities.

A. Gonzalez, B. Korpan, E. Younis, and J. Zhao, "Spectrum: Classifying, replicating and mitigating Sp ectre attacks on a speculating RISC-V microarchitecture," University of California at Berkeley, 2019.
The first paper focsuing on taxamony of Spectre attacks and defenses. The authors also replicate Spe ctre v1 and v2 on BOOMv2, an academic RISC-V processor.

Transient execution vulnerabilities (3)

- Summary of review papers (cont.)
  - There are other important reviews about TEAs on x86 instances.

W. Kosasih, Y. Feng, C. Chuengsatiansup, Y. Yarom, and Z. Zhu, "SoK: Can We Really Detect Cache Side-Channel Attacks by Monitoring Performance Counters?," at the AsiaCCS 2024, Jan. 2024.

The authors were skeptical of numerous papers published in the past ten years, which propose "detecting cache SCAs by monitoring the high performance counter (HPC) of processors", so they tested the effectiveness of these prior studies in real-world scenarios using a gadget made up of Spectre v1 and Flush+Reload. They reached a *negative* conclusion, indicating that using HPC to detect side channels is not yet feasible even for mature ISAs like x86. It is a good lesson for new ISAs like RISC-V.

L. Fiolhais and L. Sousa, "Transient-Execution Attacks: A Computer Architect Perspective," ACM Comput. Surv., vol. 56, no. 3, p. 74:1-74:38, Oct. 2023

T. Ghasempouri, J. Raik, C. Reinbrecht, S. Hamdioui, and M. Taouil, "Survey on Architectural Attacks: A Unified Classification and Attack Model," ACM Comput. Surv., vol. 56, no. 2, p. 42:1-42:32, Sep. 2023

For various cache side-channel vulnerabilities, the principles are explained, and refreshing taxonomies are proposed.

S. Cauligi, C. Disselkoen, D. Moghimi, G. Barthe, and D. Stefan, "SoK: Practical Foundations for Software Spectre Defenses," IEEE S&P 2022, May 2022, pp. 666–680.

G. Hu, Z. He, and R. Lee, "SoK: Hardware Defenses Against Speculative Execution Attacks," SEED 2021, Sep. 2021, pp. 108–120.

Systematic explanations of software and hardware defenses against transient execution attacks like Spectre.

Spectre attacks (1)

INSTITUTE of INFORMATION SECURITY

- Definition and variants
  - According to the aforementioned “Spectre white paper” from P. Kocher et al. [3], Spectre attacks involve **inducing a victim** to **speculatively perform operations** that would not occur during correct program execution and which leak the victim’s confidential information **via a side channel** to the adversary. That is, a category of TEA that combines speculation-based fault injection and SCA.
  - Variants: BCB (v1), BTI (v2), retspec (v5), SSB (v4), BHI, Retbleed, etc.

Predictors	Mechanisms	Variants
PHT/BHT/CBP	Conditional branch	BCB, v1.1, etc
BTB	Direct or indirect branch	BTI, Retbleed, etc
RSB/RAS	Return address prediction	ret2spec, INCEPTION, etc.
LSU (w/ or w/o MDP)	Store-load forwarding	SSB, CTL, etc
Others (BHI, SCO, ZDI)		

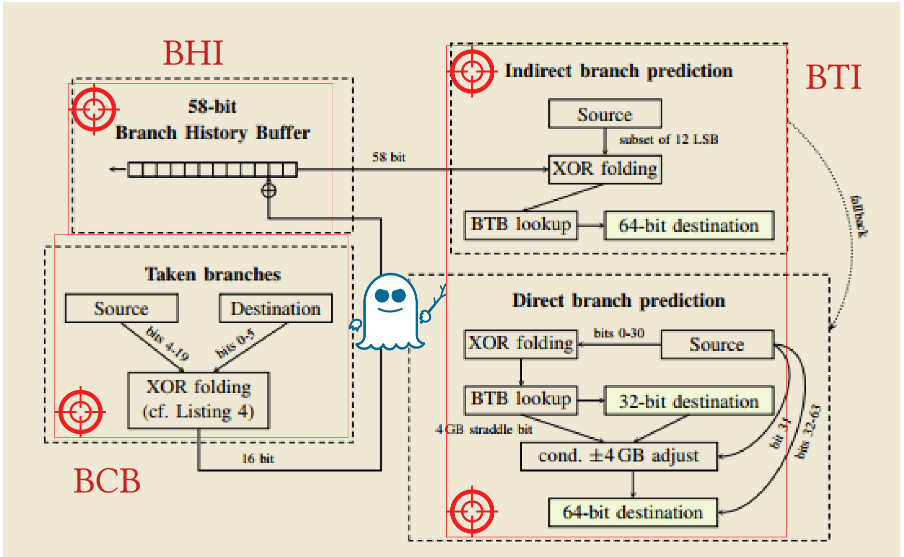


Fig. 3: Multiple mechanisms influence the prediction of direct, indirect, and conditional branches.

Spectre attacks (2)

- Example 1: flow of the BCB (Bound Check Bypass)
  1. *Preparation* phase: Within the boundary of *len(array1)*, repeatedly feed a valid *x* value into the conditional branch, making the processor's branch predictor trust this path and predicts that its subsequent outcomes will also be "true".
  2. *Access* phase: Provide a malicious value *x=k* that exceeds the memory array boundary *len(array1)*, i.e. *k>=len(array1)*. The CPU, still predicting the path as "true" from previous phase 1, **transiently** performs a memory access **beyond the boundary** and loads the normally inaccessible memory address *array1[k]* into the L1 data cache.
  3. *Transmission* phase: The CPU will correct states to "false" later, but before that, the address *array1[k]* remains in the cache for a short time. The attacker then conducts a cache timing SCA to indirectly infer the secret data.

Example codes:

- Reading out-of-bounds:
*if (x < len(array1)) { y = array2[array1[x] * 4096];*

- Writing out-of-bounds:
if (x < len(array)) { array[x] = value; }

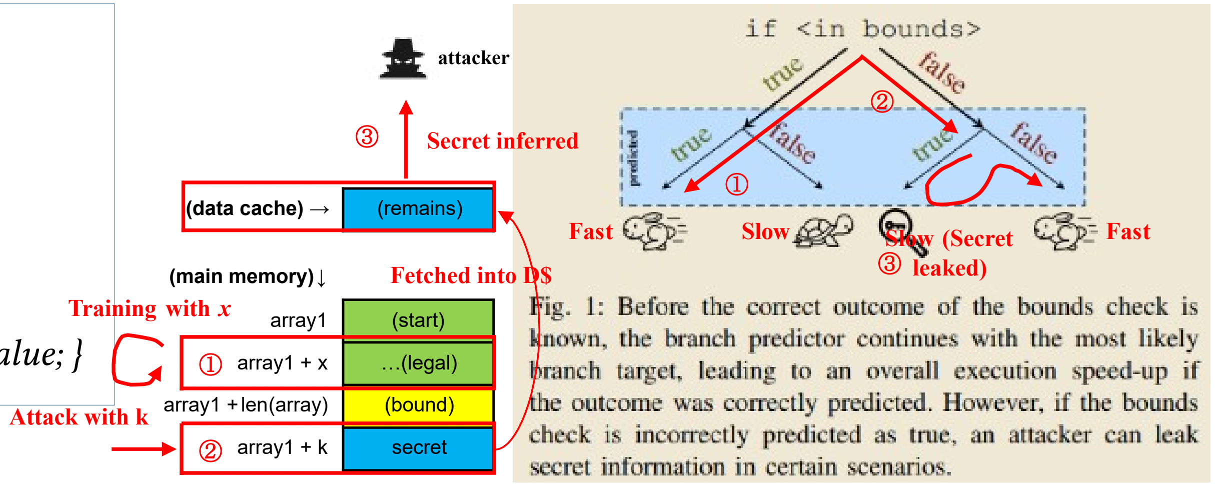


Fig. 1: Before the correct outcome of the bounds check is known, the branch predictor continues with the most likely branch target, leading to an overall execution speed-up if the outcome was correctly predicted. However, if the bounds check is incorrectly predicted as true, an attacker can leak secret information in certain scenarios.

Spectre attacks (3)

- Example 2: flow of the BTI (Branch Target Injection)
  1. *Preparation* phase: Assume the attacker controls certain addressing elements such as registers R1 and R2. In Context A, use R1 and R2 (e.g., R2 = R2 + R1) to repeatedly jump to valid memory addresses, gaining the BTB predictor's trust. The CPU predicts that "if the form of the jump requests does not change, the next one will also be valid."
  2. *Access* phase: In Context B, submit a request to access a **malicious** memory address, using a similar **"gadget" in the same form** that can trigger BTB branch prediction. The CPU immediately fetches the normally inaccessible protected information into the data cache.
  3. *Transmission* phase: As in the final stage of BCB, although a rollback will be performed later by the processor, the attacker can still use the data remained in the cache to conduct cache SCA, indirectly inferring the secret data.

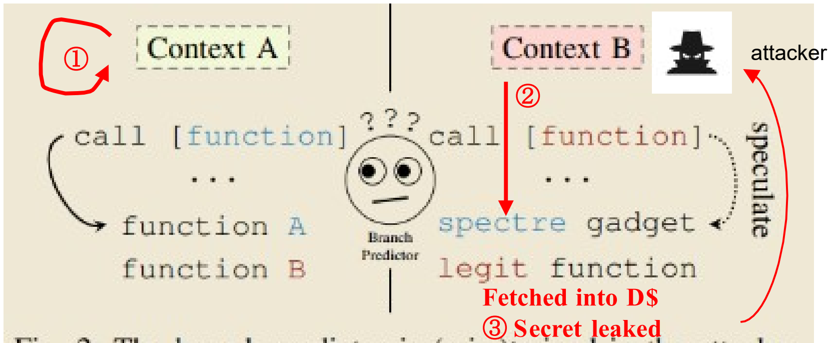


Fig. 2: The branch predictor is (mis-)trained in the attacker-controlled context A. In context B, the branch predictor makes its prediction on the basis of training data from context A, leading to speculative execution at an attacker-chosen address which corresponds to the location of the Spectre gadget in the victim's address space.

BTI is more dangerous than BCB:

BCB: Conducted along *a restricted mispre diction path*. Access is limited to the addr esses specified by conditional branch inst ructions.

BTI: Takes over the entire control flow a nd *may access any desired memory addres s* without relying on conditional branch i nstructions.

Spectre attacks (4)

- Feasibility on RISC-V implementations
  - When encompassing the transient execution attacks discovered in the industry and academia so far, more than **70** variants have been identified, across multiple ISAs.
  - From the perspective of RISC-V applicability, all those published TEAs can be generally classified into three categories:
    1) Can be reliably reproduced on RISC-V implementations
    2) Depend on proprietary designs of other ISAs
    3) Have not yet been explored
  - Currently the 1) part is concentrated on Spectre-type attacks.

CVE-	Names (alias)	RISC-V references (implementations)
2017-5753	BCB (v1)	Gonzalez et al., UCB report, 2019 (BOOMv2, Tooba, commercial chips)
2017-5715	BTI (v2)	
2017-5754	RDCL (Meltdown)	Lin et al, IEEE MWSCAS 2022 (BOOMv3)
2018-3639	SSB (v4)	Jin et al., ACM Trans. Archit. Code Optim. 2023 (BOOM, SSB also on Tooba, ret2spec also on Tooba and commercial chips)
2018-15 572	ret2spec (v5)	
Unindexed	SpectreRewind	
	Spectre-TLB	
	Bombard	Hur et al., ACM CCS 2022 (BOOM, Nutshell)
	Birgus	
	Register port contention	Hu et al. (BOOMv3)

Spectre attacks (5)

- Mitigation
  - From our observation of the progress, Spectre countermeasure proposals in RISC-V implementations can be roughly attributed with 4 tags:
    1. Theoretical check (mainly formal verification)
    2. Software: SLH derivations (mainly against BCB), Retpoline derivations (mainly against BTI or ret2spec), etc.
    3. Hardware redesign
    4. Utilizing machine learning (ML)
- Previous work[7] also classifies methods of verification for TEAs on all ISAs into 4 types according to stages of design:
  - Formal model
  - Pre-silicon
  - Post-silicon
  - Software-only

Proposal	RISC-V reference	Tags
SpecLFB	Cheng et al, USENIX Security 2024	3
Indirect jumps and calls	R. Bălucea and P. Irofti, SecITC 2023	2
SpecTerminator	Jin et al, ACM Trans. Archit. Code Optim. 20, no. 1	1
ProSpeCT	Daniel et al, USENIX Security 2023	1, 3
SpecDoctor	Hur et al, ACM CCS 2022	1, 2
IntroSpectre	Ghaniyoun et al, ISCA 2021	1, 2
SSE-RV	M. Sabbagh et al, CARRV 2021	3
ML-based real time detection	A.-T. Le et al, IEEE Access, vol. 9, pp. 164 597–164 612, 2021	4
UPEC	Fadiheh et al, DATE 2019	1, 3
SpecBuf	Gonzalez et al, U. C. Berkeley report, 2018	3

Conclusion

- Points explained in this speech
  - Optimizations of high-performance processors: OoO execution, speculative execution, branch prediction, memory prediction
  - Techniques of cache timing SCA
  - Review papers of TEAs across different ISAs including RISC-V
  - Variants and mitigation of Spectre attacks on RISC-V implementations

- Work in progress
  - Evaluate Spectre-SSB on some academic RISC-V OoO processors.
  - Test new TEAs on contemporary commercial RISC-V cores.

(For various reasons, we can not share details about these projects right now.)

- Plan
  - Sort and systematize outcomes of the above efforts to form a survey+experiment investigation for future RISC-V security researchers.