Open-Source Computer Architecture Research (OSCAR)
2025-06-21 Tokyo, Japan

Speculative Store Bypass Vulnerability in a Memory Dependence Predictor-Equipped RISC-V Processor

Tuo Chen[1], Reoma Matsuo[2], Ryota Shioya[2], and Kuniyasu Suzaki[1]

[1] *Institute of Information Security*
mgs234502@iisec.ac.jp, suzaki@iisec.ac.jp
[2] *Department of Creative Informatics, Graduate School of Information Science and Technology, The University of Tokyo*
matsuo@rsg.ci.i.u-tokyo.ac.jp, shioya@ci.i.u-tokyo.ac.jp

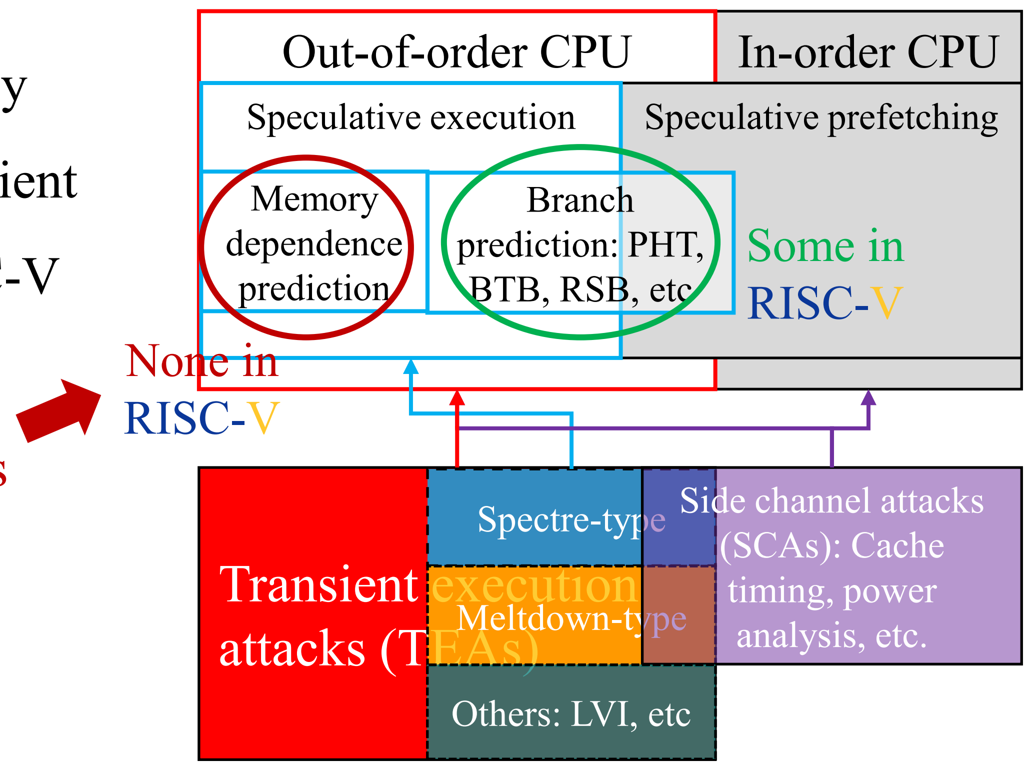
Contents

Overview

INSTITUTE of INFORMATION SECURITY

- Transient execution vulnerabilities (TEVs) identified in CPUs
- Growing attention on situation of RISC-V implementations
- Existing gap
  - Current TEV research is heavily concentrated on BOOM. Transient execution attacks against RISC-V implementations under more aggressive prediction strategies remain unexamined.

Processors of various ISAs (including RISC-V)



Microarchitectural attacks on CPUs

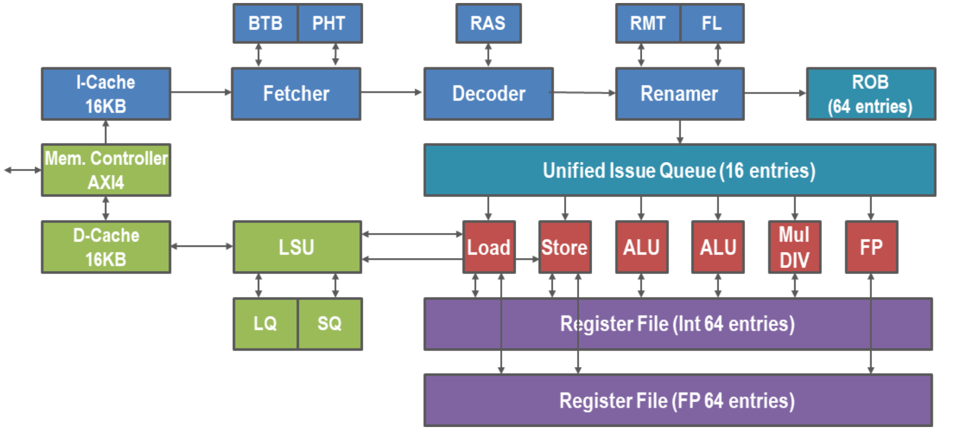
- Research objectives
  - Explore the feasibility of the Spectre-type SSB attack against a memory dependence predictor (MDP)-equipped RISC-V CPU, "RSD".
  - Confirm the results using "Konata", a pipeline visualization tool.
  - Investigate mitigations if the SSB is verified.

CVE-	Name (Alias)	Transient execution attacks => RISC-V CPU
2017-5753	BCB (v1)	Gonzalez et al., UCB report, 2019 => BOOMv2 F. A. Fuchs, KTH, 2021 => Tooba Jin et al., ACM Trans. Archit. Code Optim. 2023 => BOOMv3 Cheng et al., USENIX Security 24 => BOOMv3
2017-5715	BTI (v2)	
2017-5754	RDCL (v3)	Lin et al, IEEE MWSCAS 2022 => BOOMv3
2018-15572	Ret2spec (v5)	F. A. Fuchs, KTH, 2021 => Tooba Jin et al., ACM Trans. Archit. Code Optim. 2023 => BOOMv3 Cheng et al., USENIX Security 24 => BOOMv3
2018-3639	SSB (v4)	F. A. Fuchs, KTH, 2021 => Tooba Jin et al., ACM Trans. Archit. Code Optim. 2023 => BOOMv3 Cheng et al., USENIX Security 24 => BOOMv3 **Our work => RSD**
Unindexed	SpectreRewind	Jin et al., ACM Trans. Archit. Code Optim. 2023 => BOOMv3
	Spectre-TLB	
	Bombard	Hur et al., ACM CCS 2022 => BOOM & Nutshell
	Birgus	

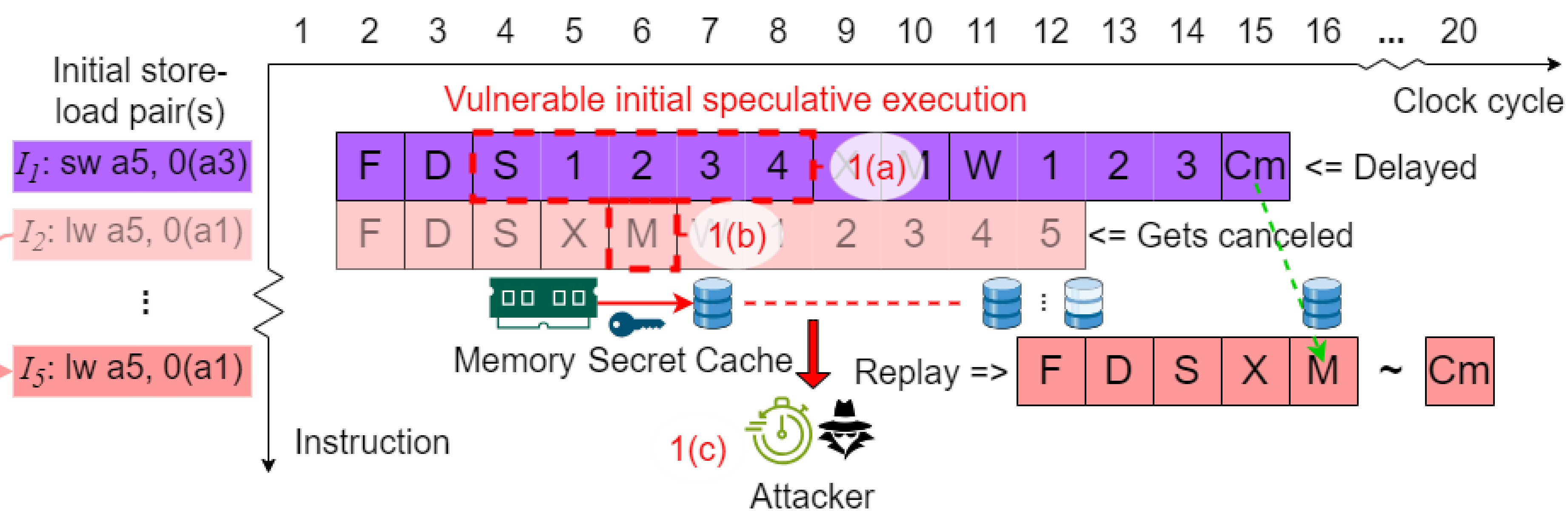
Open-Source RISC-V processor RSD

INSTITUTE of INFORMATION SECURITY

- RSD: an RV32IMF out-of-order superscalar processor core
  - Advantages: **compact**, can be synthesized for small FPGAs; and **efficient**, featuring a memory dependence prediction mechanism.
  - Conference paper: S. Mashimo et al., "An Open Source FPGA-Optimized Out-of-Order RISC-V Soft Processor," in 2019 International Conference on Field-Programmable Technology (ICFPT), Dec. 2019, pp. 63–71.
  - Main RSD repository: https://github.com/rsd-devel/rsd
  - Forked and modified RSD repo: https://github.com/cctsirjin/rsd-mod

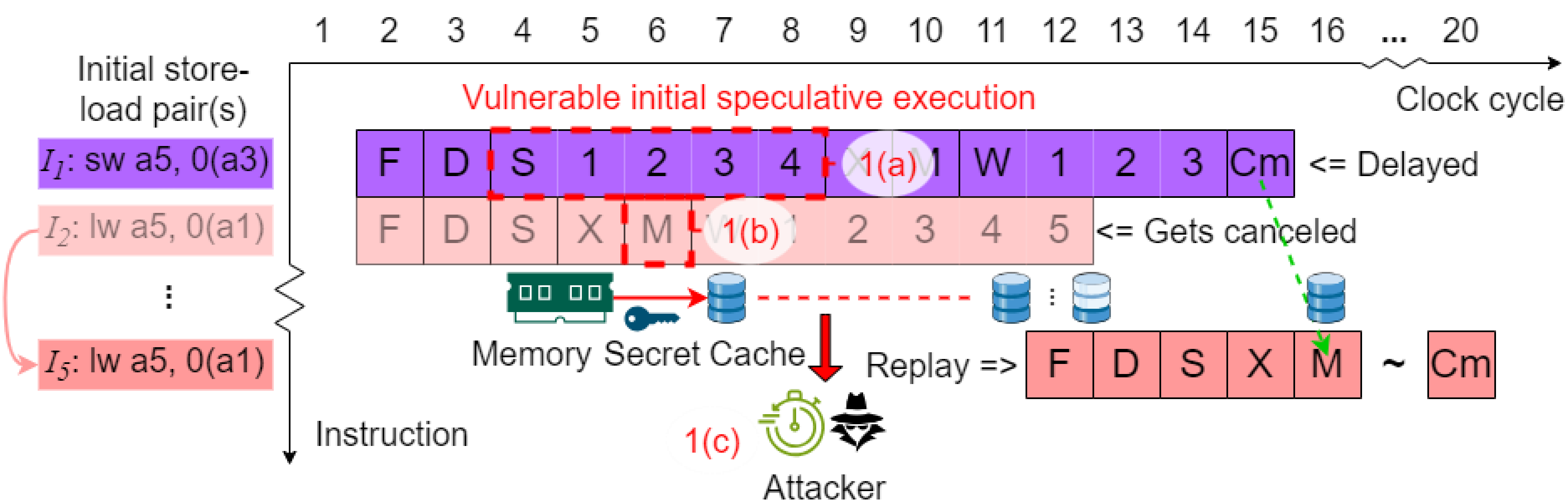


- Exploiting speculative load/store execution
  1. The first n (temporarily let $n=1$) store-load instruction pair $I_1 + I_2$ enters the pipeline and accesses the same memory address.
  2. In the absence of prior execution, the CPU cannot determine whether load I_2 is dependent on store I_1. To accelerate execution, typically it speculatively assumes they are independent.



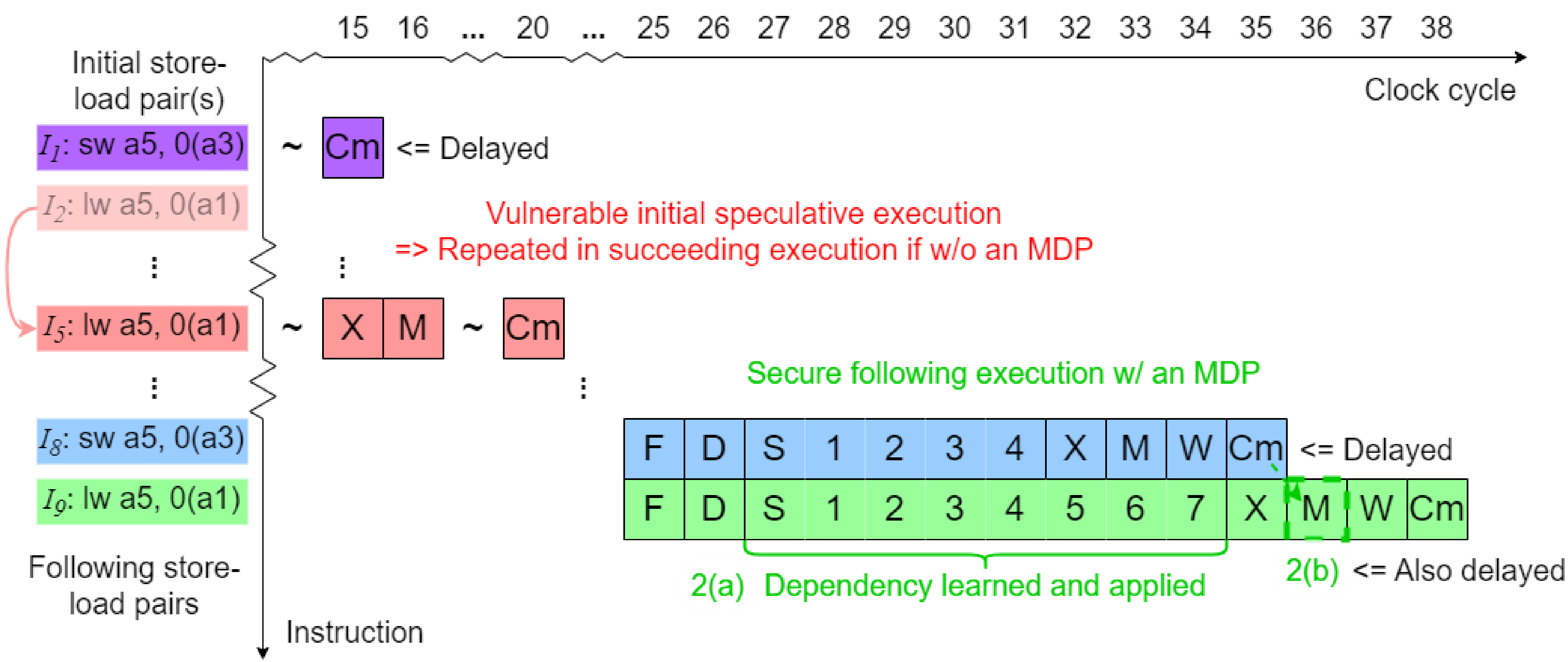
Speculative Store Bypass (SSB) vulnerability (2)

- Exploiting speculative load/store execution *(cont'd)*
  3. Owing to that assumption, secret data are loaded from the memory into the cache in 1(b), simultaneously with the store operation during 1(a).
  4. The attacker then conducts a side-channel attack on the cache to extract the secret data, as depicted in 1(c). The detection of memory ordering violation and rollback later at I_5 cannot undo this damage.



Speculative Store Bypass (SSB) vulnerability (3)

- In subsequent executions after the initial one(s) …
  - Processors w/o an MDP, such as BOOM, can be constantly exploited.
  - An MDP is anticipated to form a *partial* defense, as depicted in 2(a) and 2(b) of store-load pair $I_8 + I_9$. However, the initial *n* round(s) remain vulnerable.



Attack verification (1)

INSTITUTE of INFORMATION SECURITY

- Identifying the MDP trigger value *n* of RSD
  - Since RSD is open source, it is possible to determine the *n* by analyzing its source codes. However, compared to this theoretical approach …
  - A more empirical method involves executing a script that is prone to inducing memory ordering violations and subsequently observing the pipeline's behavior through a visualization tool, "Konata".

```c
__attribute__((noinline)) int test(volatile int* a, volatile int* b, int n)
{
    int j = 0;
    for (int i = 0; i < n; i++) {
        *a = i/2+i+1;
        j += *b;
    }
    return j;
}

int x = 0;
int y = 0;

int main(){
    test(&x, &x, 1000);
    return 0;
}
```

C language code of the test script

```
srai    a5,a4,1
add     a5,a5,a4
addi    a5,a5,1
sw      a5,0(a3)
lw      a5,0(a1)
addi    a4,a4,1
add     a0,a0,a5
bne     a2,a4,.L3
ret
```

Prone to inducing a store-load ordering violation

RISC-V assembly code of the store-load pair

Experiment platform: Verilator and ZedBoard:

VERILATOR

Attack verification (2)

INSTITUTE of INFORMATION SECURITY

- Identifying the MDP trigger value *n* of RSD *(cont'd)*
  - From Fig. 1, it can be confirmed that our early assumption of $n = 1$ is correct. Also from Fig. 2, it is evident that the learned dependency was applied.

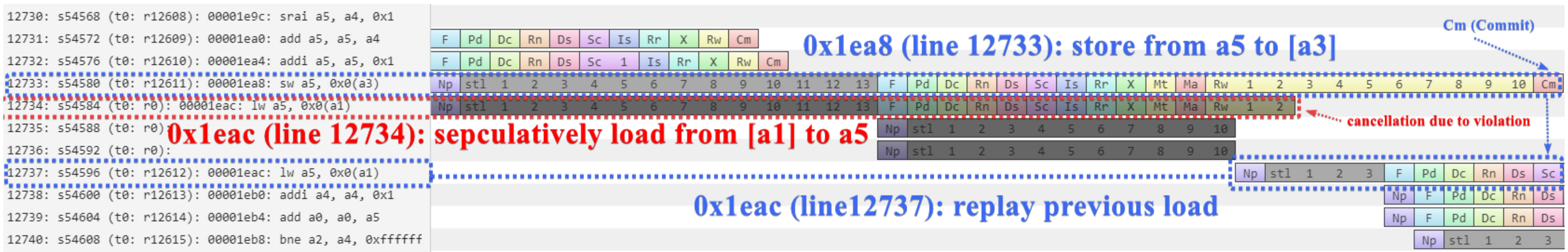
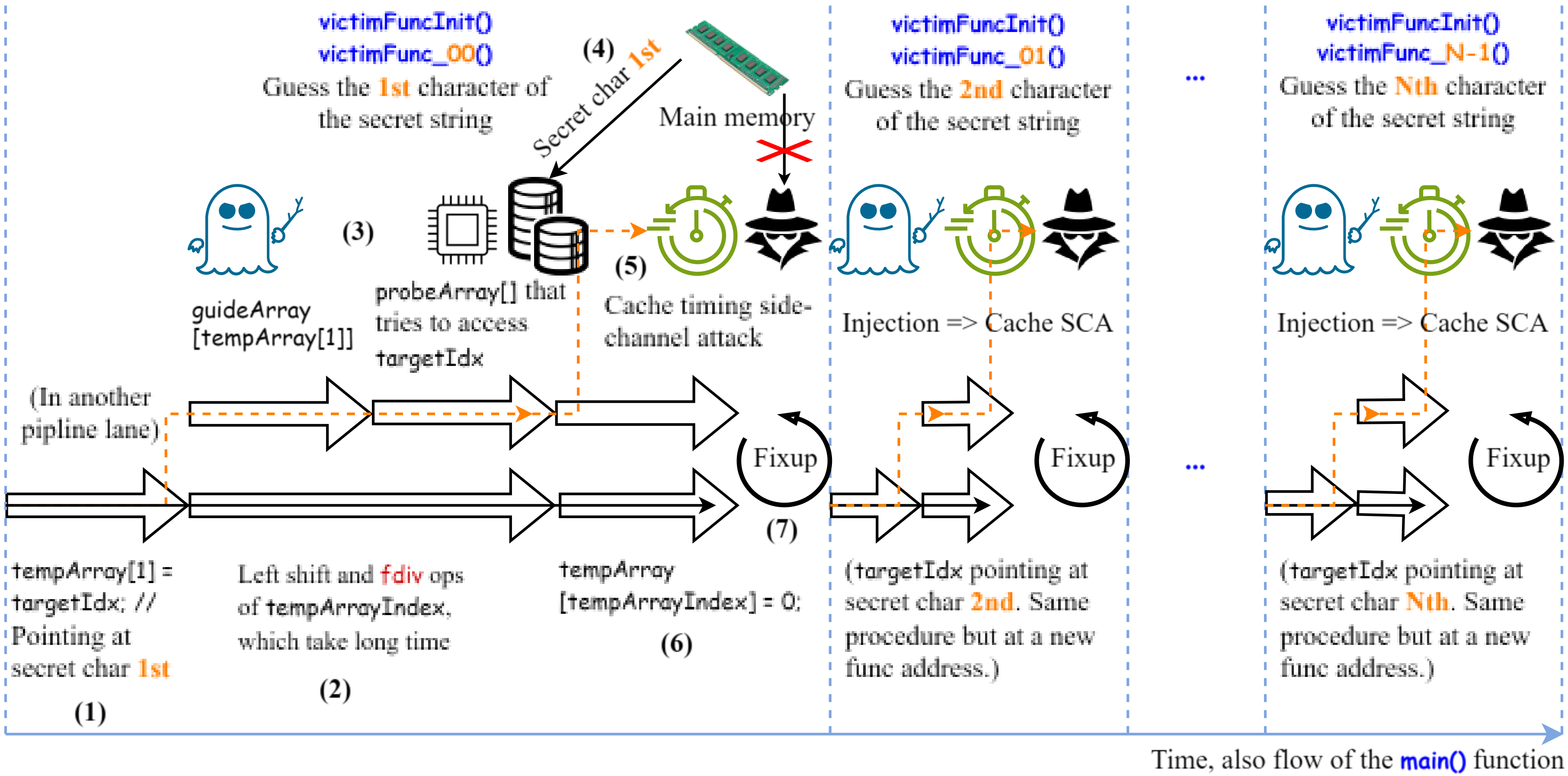


Fig. 1: Pipeline behavior during the initial round of MDP test

Fig. 2: Pipeline behavior in subsequent loops of the MDP test (from the 2nd execution onward)

INSTITUTE of INFORMATION SECURITY

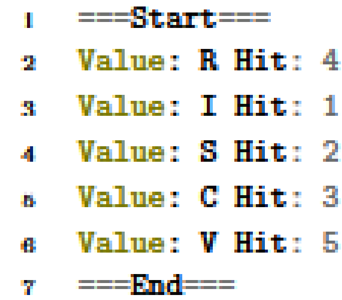
- SSB attack process and result
  - Switching among addresses victimFunc_00(), … , _N-1() to keep exploiting the property $n = 1$ and extracting secret characters successively.



Attack verification (4)

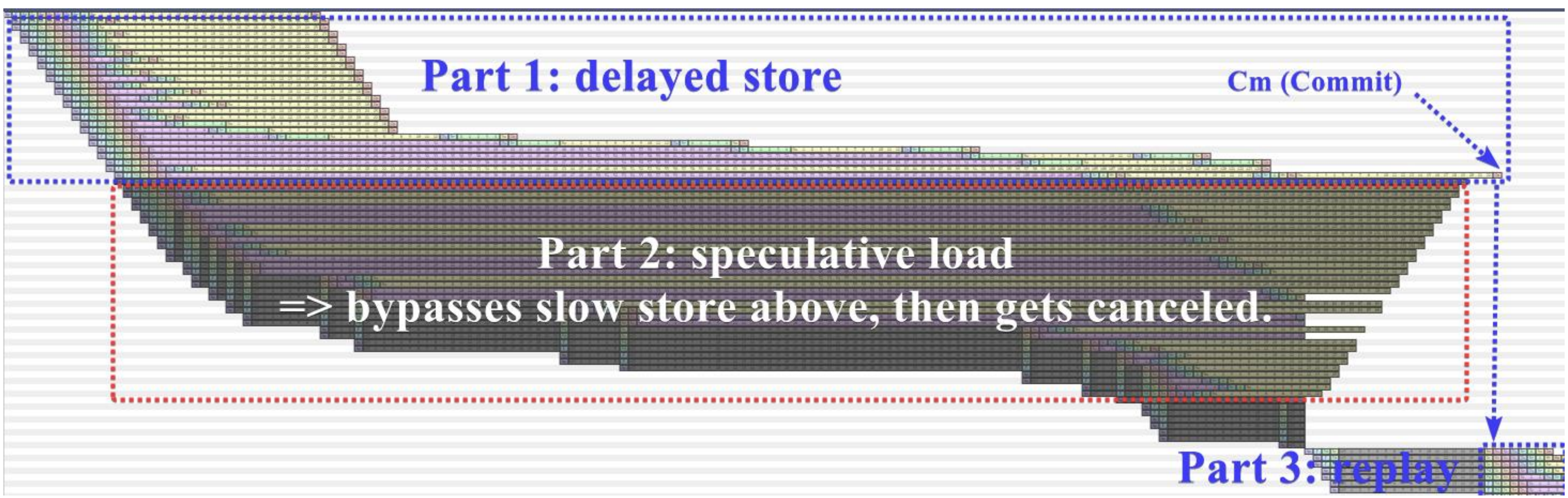
INSTITUTE of INFORMATION SECURITY

- SSB attack process and result *(cont'd)*
  - The secret string "RISCV" was correctly inferred. The execution log was also analyzed using the Konata tool.



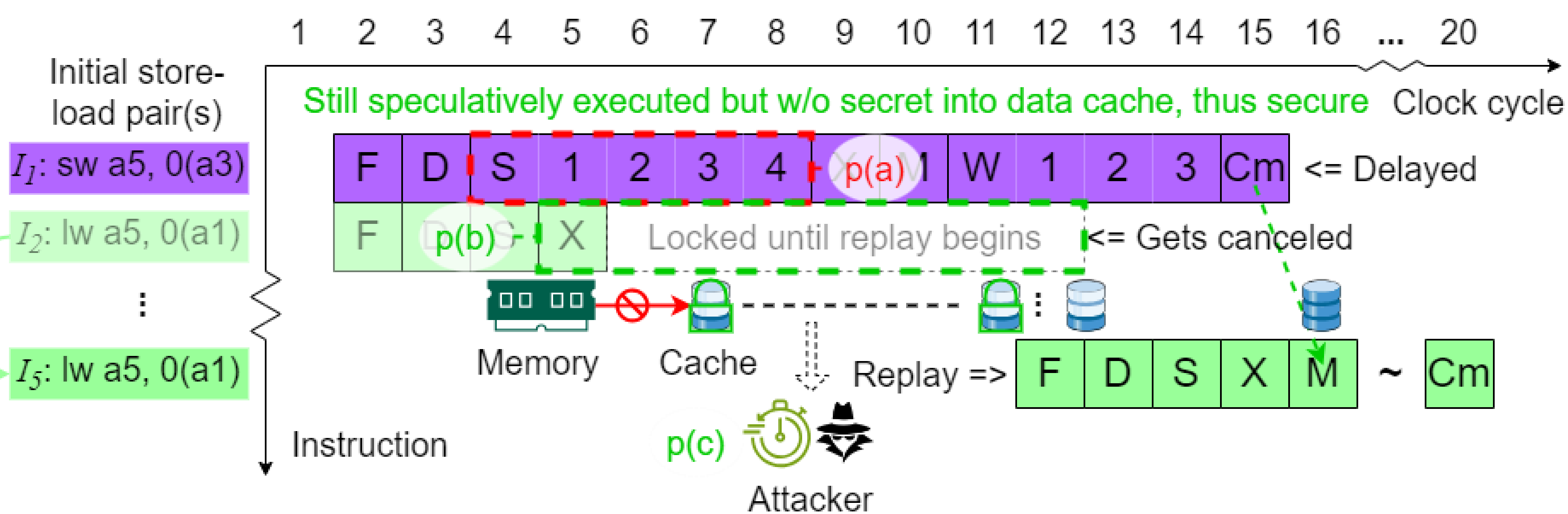
```
===Start===
Value: R Hit: 4
Value: I Hit: 1
Value: S Hit: 2
Value: C Hit: 3
Value: V Hit: 5
===End===
```

  - **Part 1** represents an intentionally delayed store operation. RSD issues a speculative load operation in **Part 2**, entering a transient execution state and causing one secret character into the dcache. It's later rolled back in **Part 3**.



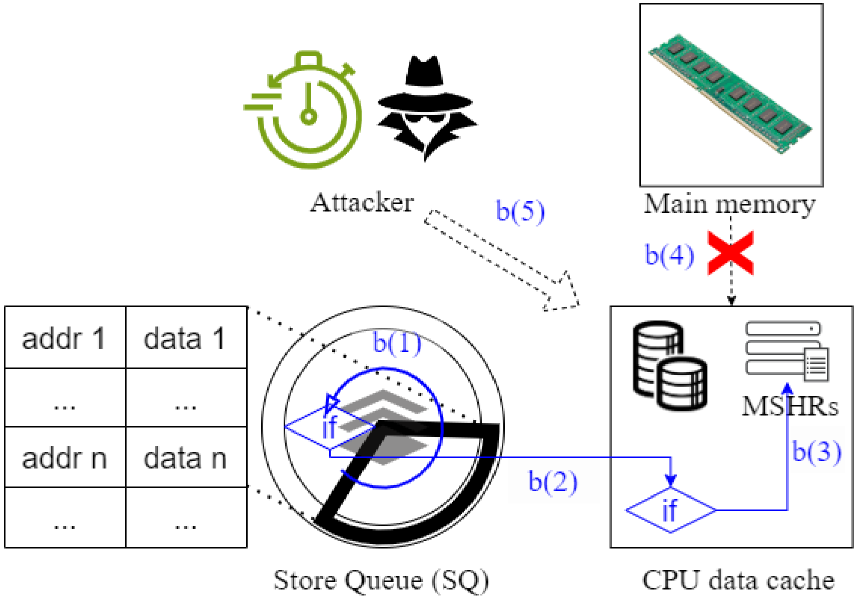
Hardware mitigation (1)

- PseudoConflict: minor modifications to the RSD μ-arch
  - Idea: When a preceding store has an unresolved address, a subsequent load will be prevented from memory accesses, even in the event of a cache miss.
  - The proposed method is illustrated in p(b). If the address of the preceding store I_1 remains unresolved, a locking mechanism can be introduced starting from the eXecution stage of I_2 in place of the former Memory stage.



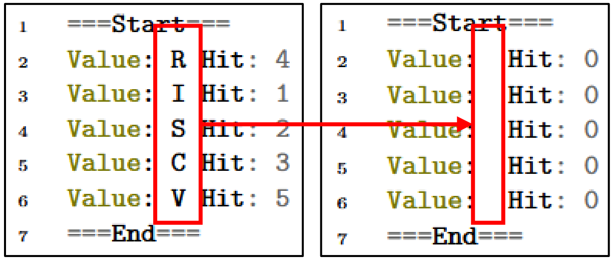
INSTITUTE of INFORMATION SECURITY

- Implementation of PseudoConflict
  - Store Queue (SQ): if a preceding store has been issued, its address and data are recorded. During the execution of a load, the system checks whether any preceding stores contain unresolved addresses.
  - Miss Status Handling Registers (MSHRs): if a preceding store operation with an unresolved address exists, MSHR allocation will be suppressed.



Hardware mitigation (3)

- Results after application of PseudoConflict
  - On the same Verilator and ZedBoard platform, the effectiveness of mitigation was confirmed.



- Evaluation of the mitigation
  - The CoreMark score / MHz (CM / MHz) and the Dhrystone MIPS (DMIPS): The baseline and the proposal are identical or nearly identical.
  - FPGA resource utilization: The mitigation leads to only a slight increase that is insignificant in the demand for LUTs and registers.
  - The operation frequency of the RSD remains unchanged, as the proposed method does not affect the critical path.

	CM/MHz	DMIPS	LUT	Register
Baseline	2.675 (100%)	201.0 (100%)	25956 (100%)	11901 (100%)
Proposal	2.675 (100%)	200.6 (99.8%)	26028 (100.28%)	11904 (100.03%)

Hardware mitigation (4)

- Benefits of PseudoConflict
  - Since the modified RSD still performs speculative execution of loads, it does not interfere with the normal operation of the MDP and **preserves the initial memory dependency learning process**.
  - **Low-cost and highly efficient**. Using precisely the characteristic of an SSB attack as a prerequisite to trigger the defense, the impact on program executions is minimal, resulting in low overhead. Hardware-based approach also offers greater cost advantages.
  - **Versatile**. Not dependent on the specific design of RSD and may be ported to other OoO CPUs.

Hardware mitigation (5)

- Current limitations of PseudoConflict
  - In implementing this mitigation, it is crucial to examine the **compatibility with other CPU components** beyond the SQ and data cache, such as the Replay Queue (RQ) of RSD in this paper, necessitating more granular hardware adjustments.
  - We have not yet conducted a statistical analysis on the proportion of normal, non-malicious programs exhibiting "preceding store with an unresolved address" behavior, similar to SSB attacks, across various real-world application scenarios. Therefore, **we cannot accurately estimate the extent of the impact** that widespread adoption of this mitigation across many CPUs would cause.

Conclusion

- Findings
  - For an OoO CPU like RSD, even if an MDP is present and only the first loop of execution is susceptible to SSB, it is still sufficient for exploitation.
  - On the other hand, this vulnerability can also be remedied with minimal effort at the hardware level, and the mitigation is generic.

- Future work
  - Adversary: Enhancing the existing SSB algorithm using new methodologies to achieve similar or improved results and efficiency.
  - Defense: Conduct additional assessments of performance impact to support large-scale adoption of PseudoConflict's framework.

For more details

- Analyzing and Mitigating the SSB Vulnerability in an MDP-Equipped RISC-V Processor
  - International Workshop on Security (IWSEC) 2025 @ Fukuoka, Japan. Nov. 25-27

  https://www.iwsec.org/2025/index.html

- IISEC Suzaki Lab

  https://lab.iisec.ac.jp/~suzaki_lab/index.html

- U Tokyo Shioya Lab

  https://www.rsg.ci.i.u-tokyo.ac.jp/lab/