Speculative Data-Oblivious Execution: Mobilizing Safe Prediction For Safe and Efficient Speculative Execution

Jiyong Yu, Namrata Mantri, Josep Torrellas, Adam Morrison*, Christopher W. Fletcher

University of Illinois at Urbana-Champaign, *Tel Aviv University

Appears in ISCA'20

ILLINOIS

The Blavatnik School of Computer Science
The Raymond and Beverly Sackler Faculty of Exact Sciences
Tel Aviv University

INTRODUCTION

Speculative Execution Attacks

- *Access* instructions speculatively read sensitive data into architectural state (e.g., registers)
- *Transmit* instructions transmit sensitive data via shared hardware states
- Goal: leak secret (speculatively-accessed data)

```
if (addr < N) { // speculation
    // access instruction
    uint8_t val = A[addr];
    // transmit instruction
    uint8_t tmp = B[64 * val];
}
```

Existing Mitigations

Defense Strategy	Invisible Loads	Delayed Execution
Examples	InvisiSpec [MICRO'18] SafeSpec [DAC'19] CleanupSpec [MICRO'19]	SpecShield [PACT'19] Conditional Spec. [HPCA'19] NDA [MICRO'19] STT [MICRO'19]
Pros	**High-performance** Never block execution	**High-security** Guarantee security properties (e.g., non-interference)
Cons	**Low-security** Do not deliver rigid and comprehensive security	**Low-performance** Block execution of transmit instructions

Our goal

FOUNDATION: SPECULATIVE TAINT TRACKING

Key feature: Blocking implicit channels

- For prediction: secret data cannot update predictors/be used for prediction
- For resolution: delay resolution (squashes) until condition is no longer secret

Observation: STT makes prediction SAFE

- Once applying the implicit channel protection, we can use prediction for performance optimization without worrying about any speculation leakage!

KEY IDEAS

Idea 1: Execute transmit instructions in a data-oblivious fashion → worst-case execution

Idea 2: Avoid worst-case execution by predicting how the execution should be performed

Idea 3: Protect the prediction with STT's implicit channel protection

Key capability: execute unsafe transmitters early and safely

SPECULATIVE DATA OBLIVIOUS EXECUTION (SDO)

SDO Framework

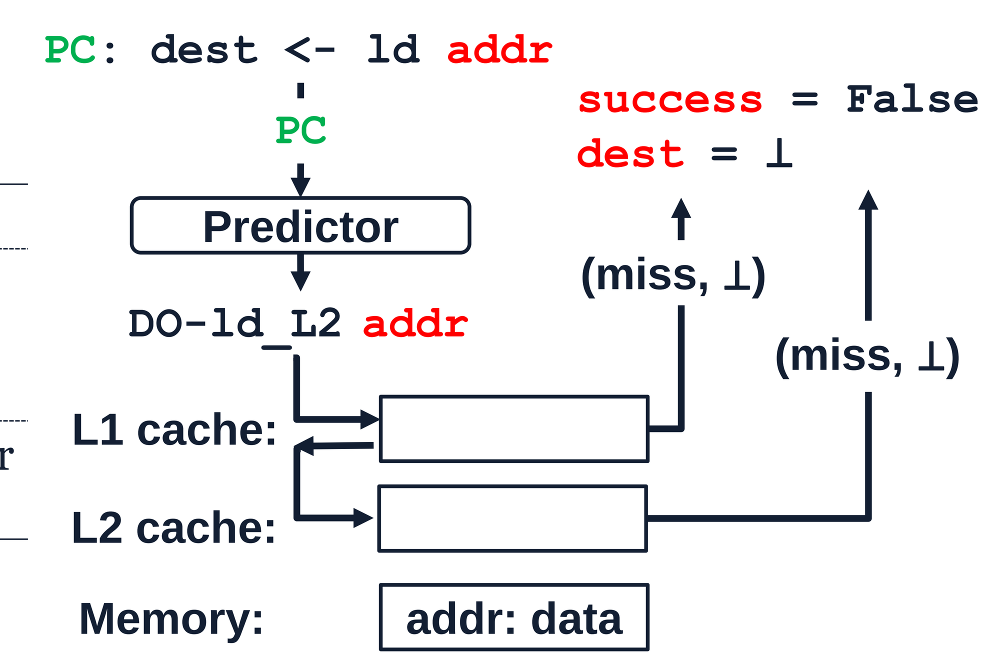
- Define Data-Oblivious (DO) variants for a given transmit instruction
  - Each DO variant must be data-oblivious
  - Each DO variant may produce invalid result unless inputs satisfies certain condition
- Create dedicated DO predictor to predict DO variant at runtime
- [Follow STT's protection] At runtime:
  - Secret data cannot update DO predictor/be used for predicting DO variant
  - Delay resolution (squash) until condition is no longer secret

Transmit instruction signature	**dest <- op args**
DO variant signatures	**($dest_1$, $success_1$) <- $DO\text{-}op_1$ args** … **($dest_N$, $success_N$) <- $DO\text{-}op_N$ args**
DO variant predictor	**i <- Predictor.predict (public_input)** **($dest_i$, $success_i$) <- $DO\text{-}op_i$ args**
Resolving when safe (condition is no longer secret)	**Predictor.update(…)** **if (!$success_i$)** **squash from "dest <- op args"**

SDO Design for Loads

- Define DO variants:

dest <- ld args
DO-ld_L1 : access L1 DO-ld_L2 : access L1, L2 DO-ld_Mem : access L1, L2, Mem
(dest_X, success_X) ← DO-ld_X addr dest_X = ⊥ if success_X == FALSE



- DO variant must be data-oblivious

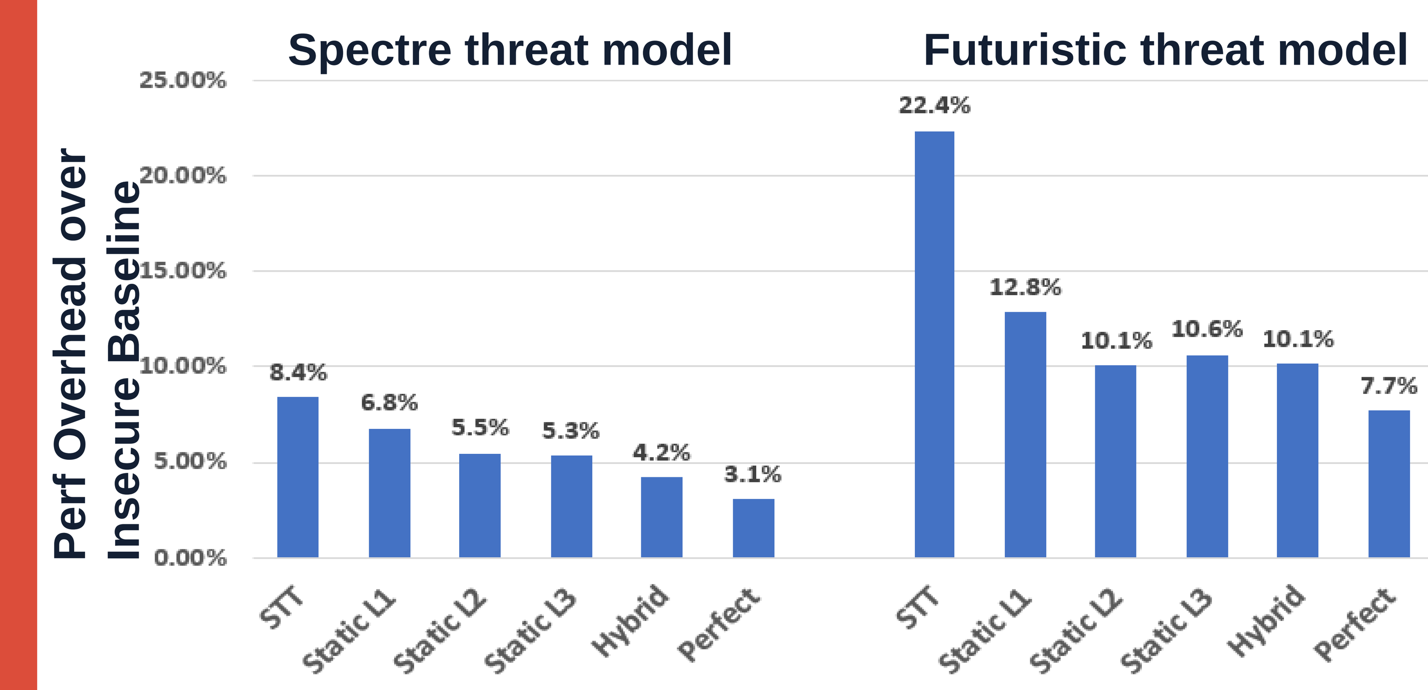
Attack Vectors	Reason	Mitigation Strategy
MSHR coalescing	Requests share MSHR if addresses match	Disallow MSHR coalescing for **DO-ld_X** requests
Bank conflict	Banks cover different addresses	Serialize **DO-ld_X** access to banks
Way prediction	Use incoming address to predict cache way	Disable way prediction / Apply STT's prediction mechanism
……	……	

- Customize DO predictor for loads (cache level predictor). General metrics:
  - 😀 Accurate and precise: predicted cache level equal to actual cache level
  - 😐 Accurate but imprecise: predicted cache level lower than actual cache level
  - ☹ Inaccurate: predicated cache level higher than actual cache level
- Resolve DO prediction when safe
  - Update predictor; squash pipeline if success == FALSE
  - For multi-processor:
    - DO-ld_X must not modify cache state
      - → Data fetched by DO-ld_X may not be cached in L1
      - → May miss cache invalidation
    - Solution: Apply *invalidation* infrastructure from Invisispec [MICRO'18]

PERFORMANCE EVALUATION

Evaluation Settings

- Gem5 simulator, w/ 3-layer cache with MESI protocol
- Transmitter covered by SDO:
  - Floating-point multiply/divide: always predict non-subnormal
  - Load: evaluating multiple DO predictors
    - Static L1: always predict DO-ld_L1
    - Static L2: always predict DO-ld_L2
    - Static L3: always predict DO-ld_L3
    - Hybrid: our customized tournament cache-level predictor
    - Perfect: a theoretically-best DO predictor (oracle)



	Spectre model		Futuristic model	
Config	**Precision**	**Accuracy**	**Precision**	**Accuracy**
Static L1	71.87%	71.87%	75.48%	75.48%
Static L2	7.01%	78.74%	6.58%	83.39%
Static L3	4.60%	85.04%	3.71%	89.25%
Hybrid	84.30%	86.49%	84.34%	87.18%

CONCLUSIONS

- SDO is a new speculative execution attack mitigation framework that enables strong security (equivalent to STT) and high performance
- Key ideas
  - STT provides principles for safe prediction and resolution
  - SDO uses safe prediction/resolution to execute transmit instruction **early** and **safely** by combining prediction with data-obliviousness

ACKNOWLEDGEMENTS

This work was funded in part by NSF under grant CNS-1816226, Blavatnik ICRC at TAU, ISF under grant 2005/17,and by an Intel Strategic Research Alliance (ISRA) grant.