Data Oblivious ISA Extensions for Side Channel-Resistant and High-Performance Computing

Jiyong Yu, Lucas Hsiung, Mohamad El Hajj, Christopher W. Fletcher
Department of Computer Science, College of Engineering, University of Illinois at Urbana-Champaign

Appears in Network and Distributed System Security Symposium (NDSS), 2019
Distinguished Paper Award Honorable Mentions

Introduction

Microarchitecture side channel attacks
- Huge privacy threat
- Fundamental problem:
  Secret data impacts HW resource usage
- Various attacks proposed for different HW resources

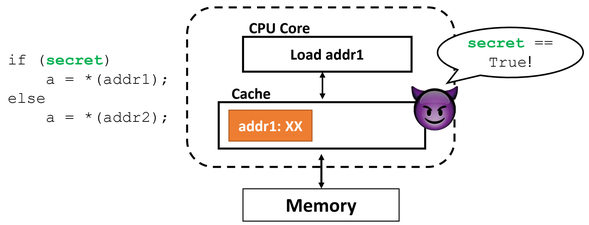


Figure 1: Attacker monitors cache pressure to learn `secret`

Difficulty in solving this type of attacks:
No contract between hardware and software
- Software doesn't know what hardware leaks
- Hardware doesn't know what is secret in software

Data Oblivious Programming

Definition
- Software solution to block microarchitectural side channel attacks
- Rewrite programs in a data oblivious form (as a static data-flow graph)

```
/* Source program */
if (secret)
    a = *(addr1);
else
    a = *(addr2);

/* machine code */
a ← load (addr1);
b ← load (addr2);
cmov secret, a, b;
    // a = secret? b : a
```

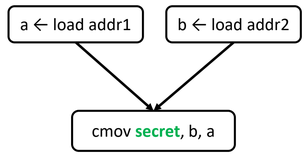


Figure 2: Data oblivious programming example

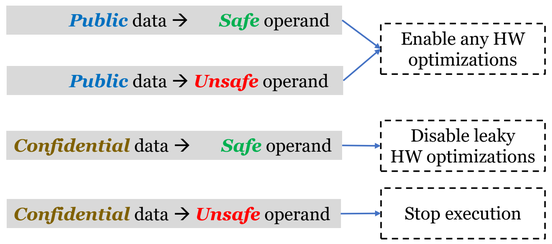
Security assumptions
- Instructions are evaluated in a data-independent manner
- Data is transferred in a data-independent manner
- Instruction sequence is not a function of data

But SW-only Data Oblivious Programming fails on modern processors!
All security assumptions can be undermined by hardware optimizations

Solution: Data Oblivious ISA (OISA)

ISA Design Methodology
- New instructions marking data as ***Confidential***/***Public***
- New instructions featuring ***Safe***/***Unsafe*** operands

Runtime Checking:



Component 1: New Dynamic Information Flow Tracking
- Tracking ***Confidential***/***Public*** in hardware
  - Software defines secret data as ***Confidential***
  - Hardware tracks and taints data using DIFT

Component 2: New Instructions with Safe operands
- Each input operand is defined as ***Unsafe*** or ***Safe***
  - ***Safe*** operand blocks side channels from that operand
  - ***Unsafe*** operand provides no protection

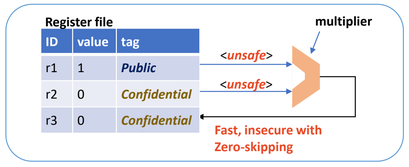


Figure 3: Multiply with ***Unsafe*** operand. ***Confidential*** R2 is **leaked** due to optimization zero-skipping.

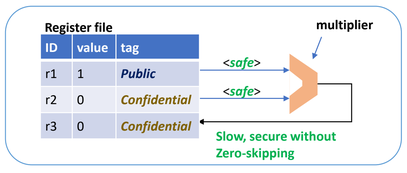
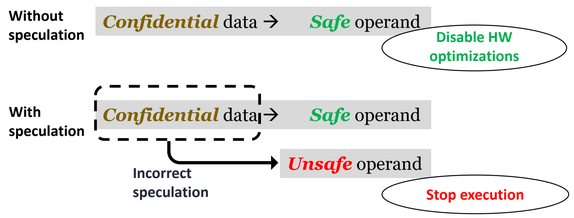


Figure 4: Multiply with ***Safe*** operand. ***Confidential*** R2 is **not leaked** with optimization disabled.

Design Features

Security: Defense against non-speculative & speculative side-channel attacks



Efficiency: Design space for safe optimizations
- High-performance instructions with safe operand
- Case 1: Oblivious load (with *Safe* address) from an object of size N
  - Baseline: linear scan – O(N)
  - Optimization 1: Oblivious RAM – O(logN)
  - Optimization 2: Hardware partitioning – O(1)
- Case 2: Oblivious sort
  - Baseline: bitonic sort – $O(N\log^2 N)$
  - Optimization 1: constant time merge sort – O(NlogN)

Portability: Consistent security guarantee across hardware instances

Putting it All Together

ISA design time
ISA designers decide instructions with *Safe* /*Unsafe* operands

Hardware design time
Hardware designers augment processors with logic to enable/disable hardware optimizations

Programming time
Programmers annotate data as *Confidential* /*Public*

Compilation time
Compilers generate executables with correct security semantics

Runtime
Processors enforce runtime checking

Evaluation

Hardware prototyping on RISC-V BOOM
- Proposing a Data Oblivious ISA Extension for RISC-V instruction set
- Implementing new instructions with *Safe* operand
- Implementing new hardware DIFT logic

- Int/FP arithmetic with *Safe* operands
- Branches/Jumps with *Unsafe* operands
- Two flavors of loads/stores
  - *Safe* data, *Unsafe* address
  - *Safe* data, *Safe* address
- Instructions to set data as *Confidential*/*Public*

Figure 5: RISC-V OISA Extension

Performance Evaluation
- Achieving speedup of up to 8.8x over baseline data oblivious programming
- Case studies:
  - AES: 4.4x speedup over bitslice AES
  - Memory oblivious library: more than 4.6x speedup over ZeroTrace [SGF'18]

Security Evaluation
- Proving non-interference property for the trace of observable processor states
- Challenges:
  - Formalizing attacker's observability
  - Modeling complicated modern processors

Long-Term Impact

OISA is a HW-SW security abstraction
- It closes ALL side-channel leakages
- It incorporates different side-channel mitigations

OISA is a bridge between secure hardware and applied cryptography
- OISA is a preferred backend for data oblivious programming frameworks
- OISA supports high-complexity Safe instructions

OISA motivates future (speculative) side-channel defenses
- Speculative Taint Tracking [MICRO'19, best paper award] is inspired by OISA

ILLINOIS