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

Jiyong Yu, Lucas Hsiung, Mohamad El Hajj, Christopher W. Fletcher

Presented by Brandon John
Motivation

- Rise of Side Channel attacks
- Current SW workarounds for data-oblivious operation:
  - Constant time: `cmov` instead of branches
  - Cache side channels: Loop over entire memory to retrieve one value
  - Both are slow and in many cases not actually guaranteed to be safe!
- Solution: Need contract in ISA as to how microarchitecture acts when handling secure data
Threat Model

- Adversary is software running as supervisor or user.
- Assume shielding such as SGX enclave
- Non goals:
  - Power/EM side channels
  - Integrity
  - Availability
Overview

- Dynamic sensitive data tracking
  - Labels all registers and memory as Confidential or Public
- All instructions: Operands are either Safe or Unsafe
  - Confidential + Safe:
    - Operate in constant time
    - Side effects must be hidden
    - Output marked confidential
  - Confidential + Unsafe:
    - Label violation / Label fault
- Public + Safe/Unsafe
  - Performance optimizations allowed

- Oblivious Memory Extension
  - Scratchpad area for fast memory access
  - Invisible to everything but the secure process
Thoughts?
My Thoughts

**Strengths**
- Side channel resistance achieved without significant slowdown for the general case
  - Still allows: Super scalar, speculative, out of order execution
- Secrecy propagation inspired by GLIFT
- Minor area cost (<5%)
- 1.7x to 8.8x speedup compared to current data oblivious software

**Weaknesses**
- All DRAM accesses are doubled due to need to store the 1 bit label separately from the data
- “Our slowdown [2.17x] relative to insecure is caused by the compiler not optimizing code around ocld instructions”
- Unclear how much of slowdown is due to optimization issues
- That said, still clearly faster than bitslice
- The proofs were entirely beyond my comprehension
Current Data-Oblivious Code Issues

- Branch, Jump, Memory speculation
  - Can reveal aliased resources
- Sub-address optimizations
  - Can't rely on whole cache line to be treated uniformly
  - Is there even a guarantee in the ISA that cache lines are 64 bytes?
- Input-dependent math
  - Multiply/Divide non constant-time
- Microcode
  - No guarantee that any operations (cmov, etc.) are constant-time
- Other data-dependent optimizations
  - Compression + speculation are allowed by the ISA
## ISA Adjustments

<table>
<thead>
<tr>
<th>Classification</th>
<th>Safe Alternatives</th>
<th>Oblivious Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>oseal</code></td>
<td><code>oadd</code>, <code>oaddi</code>, etc.</td>
<td><code>ocld</code>, <code>ocst</code>, etc.</td>
</tr>
<tr>
<td>- Marks register as confidential</td>
<td>- Functions like <code>add</code>, <code>addi</code></td>
<td>- Load/Store at a confidence address</td>
</tr>
<tr>
<td><code>ounseal</code></td>
<td>- Must be implemented data-obliviously</td>
<td>- Requires extra local memory</td>
</tr>
<tr>
<td>- Marks register as public.</td>
<td>- Output label: confidential if either input is confidential</td>
<td>- Implemented here as a portion of cache</td>
</tr>
<tr>
<td>- Serializing!</td>
<td>- Can operate on confidential data</td>
<td>- Must load/store entire range on context switch</td>
</tr>
</tbody>
</table>
Implementation: Oblivious Memory Partition

**Software**
- OSZ: Oblivious memory partition size
  - Not defined in ISA, can vary between implementations
- Software interface to place memory in OMP
  - `obl_alloc()`, `obl_free()`, `obl_read()`, `obl_write()`
- Allows for arbitrary size storage,
  - will automatically use memory scan to read values if `obl_alloc()` doesn’t fit in the OMP

**Hardware**
- New instructions
  - `ocpuid`: Gives size of OMP (OSZ)
  - `oc1d/ocst`: load/store from OMP
- Needs OSZ bytes of “fast” memory
- Guarded with SGX/other enclave mechanisms
- Can implement by partitioning the cache
  - Allocate N ways to the OMP when in appropriate modes
Implementation: Labels

Label Storage
- Each word needs 1 bit for public vs confidential.
- In processor: Can expand size of storage
  - Pipelines
  - Register File
  - Cache
- DRAM: Must somehow store this extra bit
  - Current implementation takes 2x DRAM accesses to retrieve a single cache line

Label Checks
- All execution units wrapped in a label station.
  - Checks for operand public/confidential bit
  - Raises label violation/label fault as needed
  - Disables hardware optimizations on confidential inputs
  - Propagates label based on operatnds
- Disabling data-dependent optimizations:
  - Arithmetic: Counter for known max duration, don’t release data until then
Results

<table>
<thead>
<tr>
<th>Area (um²)</th>
<th>BOOM</th>
<th>BOOM+ OISA</th>
<th>Overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logic</td>
<td>363,900</td>
<td>388,658</td>
<td>6.80%</td>
</tr>
<tr>
<td>SRAM</td>
<td>384,232</td>
<td>391,291</td>
<td>1.84%</td>
</tr>
<tr>
<td>Total</td>
<td>748,132</td>
<td>779,949</td>
<td>4.25%</td>
</tr>
</tbody>
</table>

- Note that their PRNG is 30% of the total overhead. A TRNG would be much smaller in actual hardware.

Performance

- OISA + Oblivious Memory Partition:
  - Small data size: 8.8x faster than OISA
  - Large data size: 1.7x faster than OISA

- OISA vs insecure
  - Small data size: 3.2x slower than insecure
  - Large data size: 40.4x slower than insecure
Discussion Questions

✧ Could an ISA contract similar to this OISA help with power side channel prevention?

✧ Can we take advantage of persistent state to reduce required taint tracking?
  ✧ (operation X always uses private data) -> (don’t dynamically track taint here, assume its always tainted)
  ✧ Would only enabling dynamic tracking while in SGX mode be valid?
Discussion Questions

❖ What can be done to improve the labeling system?
  ❖ Specifically, can we avoid the double RAM read requirement?

❖ What parts of the inclusion of the OISA slow down normal (not-secret) workloads? How much of a problem is this?
Discussion Questions

✧ Can label faults be exploited to cause denial of service attacks?

✧ How does a microarchitecture implementing an OISA prove it meets the required functionality? Is there an expectation for OISA chips to be formally verified?
Discussion Questions

Would the Oblivious Memory Partition be useful on its own for performance, without the rest of the OISA being implemented?