USING INFORMATION FLOW TO DESIGN AN ISA THAT CONTROLS TIMING CHANNELS
WE NEED A NEW
HW-SW CONTRACT (ISA) TO
CONTROL MICROARCHITECTURAL
TIMING CHANNELS
MICROARCHITECTURE

- Unspecified CPU behavior + state
- Primarily affect performance

### Caches

<table>
<thead>
<tr>
<th>Valid</th>
<th>Address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0xfffff</td>
<td>0x1234</td>
</tr>
<tr>
<td>0</td>
<td>0xabcd</td>
<td>0xaaaaa</td>
</tr>
<tr>
<td>1</td>
<td>0x10af</td>
<td>0xde12</td>
</tr>
</tbody>
</table>

### Branch Predictors

<table>
<thead>
<tr>
<th>PC</th>
<th>Branch?</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x1234</td>
<td>Y</td>
</tr>
<tr>
<td>0xa3a4</td>
<td>N</td>
</tr>
<tr>
<td>0xddddd</td>
<td>Y</td>
</tr>
</tbody>
</table>

- Arithmetic Units, Prefetchers, Address Translators, Fill buffers, Memory Arbiters, Pipeline State, etc.
MICROARCHITECTURAL TIMING CHANNEL

- Essence of the *Meltdown ('18)* attack

```
s1 = p0[s0]
p1 = p0[0]
```

<table>
<thead>
<tr>
<th>Valid</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0xfffff</td>
</tr>
<tr>
<td>0</td>
<td>0xabcd</td>
</tr>
<tr>
<td>0</td>
<td>0x10af</td>
</tr>
</tbody>
</table>
MICROARCHITECTURAL TIMING CHANNEL

- Essence of the *Meltdown ('18)* attack

\[
s_1 = p_0[s_0] \\
p_1 = p_0[0]
\]

<table>
<thead>
<tr>
<th>CPU Cache</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Valid</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0xfffff</td>
</tr>
<tr>
<td>0</td>
<td>0xabcd</td>
</tr>
<tr>
<td>0</td>
<td>0x10af</td>
</tr>
</tbody>
</table>
**MICROARCHITECTURAL TIMING CHANNEL**

- Essence of the *Meltdown* ('18) attack

```
s1 = p0[s0]
p1 = p0[0]
```

<table>
<thead>
<tr>
<th>Valid</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0xffffffff</td>
</tr>
<tr>
<td>1</td>
<td>p0 + s0</td>
</tr>
<tr>
<td>0</td>
<td>0x10af</td>
</tr>
</tbody>
</table>

**CPU Cache**
MICROARCHITECTURAL TIMING CHANNEL

- Essence of the *Meltdown (’18)* attack

\[
s_1 = p_0[s_0] \\
p_1 = p_0[0]
\]

<table>
<thead>
<tr>
<th>Valid</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0xffffffff</td>
</tr>
<tr>
<td>1</td>
<td>(p_0 + s_0)</td>
</tr>
<tr>
<td>0</td>
<td>0x10af</td>
</tr>
</tbody>
</table>

\(s_0 == 0\)  Cache HIT!
INTRODUCTION

MICROARCHITECTURAL SIDE CHANNELS

Spectre ('18)
Foreshadow ('18)
Meltdown ('18)
RIDL ('19)
Zombieload ('19)
INTRODUCTION

MITIGATING MICROARCHITECTURAL SIDE CHANNELS
MITIGATING MICROARCHITECTURAL SIDE CHANNELS

- Microarchitectural side channels bypass current contracts
- Need a HW-SW co-design mitigation mechanism
INTRODUCTION

THIS PAPER: INFORMATION FLOW CONTROL ISA

1. Sound and portable contract for secure SW design
   - Extension of RISC-V
   - Timing-Sensitive Noninterference

2. Guide to HW designers
   - Microarchitectural Noninterference
   - No implementation details

3. Practical ISA Features
   - Constrained Downgrading
   - Control Transfer Primitives
DYNAMIC IFC CONTRACT

- Software controls IFC labels
  - Labels are *mutable state*
- SW must explicitly change architectural labels
- Hardware enforces policies at runtime

FLOATING LABELS
ENFORCING NONINTERFERENCE

- Change existing semantics of RISC-V

\[ L(pc) \sqcup L(x2) \sqcup L(x3) \sqsubseteq L(x1) \]

\[ \text{add x1, x2, x3} \]

\[ L(pc) \sqcup L(x2) \sqcup L(x3) \not\sqsubseteq L(x1) \]

\[ X1 := X2 + X3 \]

NO-OP
ENFORCING NONINTERFERENCE

- Change existing semantics of RISC-V

\[ L(x_1) \cup L(x_2) \subseteq L(pc) \]

beq x1, x2, LOC

\[ L(x_1) \cup L(x_2) \nsubseteq L(pc) \]

if (x1 == s2) then PC := LOC

NO-OP

SIMILAR RESTRICTIONS ON OTHER INSTRUCTIONS
TIMING SENSITIVITY

- How does software control timing?
- State that influences timing \( \subseteq t \)
- Language-based Timing Mitigation [Zhang et al. ’12]
TIMING SENSITIVITY

(SECRET CAN INFLUENCE TIMING)

\[ s1 = p0[s0] \]
\[ p1 = p0[0] \]
HARDWARE REQUIREMENTS

TIMING SENSITIVITY

(SECRETS CAN INFLUENCE TIMING)

\[ s_1 = p_0[s_0] \]
\[ p_1 = p_0[0] \]

<table>
<thead>
<tr>
<th>LBL</th>
<th>ADDR</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>??</td>
</tr>
<tr>
<td>X</td>
<td>??</td>
</tr>
<tr>
<td>X</td>
<td>??</td>
</tr>
</tbody>
</table>
TIMING SENSITIVITY

\[ s_1 = p_0[s_0] \]
\[ p_1 = p_0[0] \]

(SECRETS CAN INFLUENCE TIMING)

HARDWARE REQUIREMENTS

CPU Cache

<table>
<thead>
<tr>
<th>LBL</th>
<th>ADDR</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEC</td>
<td>p_0 + s_0</td>
</tr>
<tr>
<td>X</td>
<td>??</td>
</tr>
<tr>
<td>X</td>
<td>??</td>
</tr>
</tbody>
</table>
HARDWARE REQUIREMENTS

TIMING SENSITIVITY

(SECRETS CAN INFLUENCE TIMING)

CPU Cache

s0 == 0
Cache HIT!

s1 = p0[s0]
p1 = p0[0]
TIMING SENSITIVITY

(SECURIT SANNOT INFLUENCE TIMING)

\[ s_1 = p_0[s_0] \]
\[ p_1 = p_0[0] \]

<table>
<thead>
<tr>
<th>LBL</th>
<th>ADDR</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>??</td>
</tr>
<tr>
<td>X</td>
<td>??</td>
</tr>
<tr>
<td>X</td>
<td>??</td>
</tr>
</tbody>
</table>
HARDWARE REQUIREMENTS

TIMING SENSITIVITY

(SECRETS CANNOT INFLUENCE TIMING)

PUBLIC

\[ s_1 = p_0[s_0] \]
\[ p_1 = p_0[0] \]

<table>
<thead>
<tr>
<th>LBL</th>
<th>ADDR</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEC</td>
<td>p_0 + s_0</td>
</tr>
<tr>
<td>X</td>
<td>??</td>
</tr>
<tr>
<td>X</td>
<td>??</td>
</tr>
</tbody>
</table>

CPU Cache
TIMING SENSITIVITY

(SECURTS CANNOT INFLUENCE TIMING)

CPU Cache

<table>
<thead>
<tr>
<th>LBL</th>
<th>ADDR</th>
</tr>
</thead>
<tbody>
<tr>
<td>PUB</td>
<td>XX</td>
</tr>
</tbody>
</table>

Can’t use Secret Entries! Cache Miss!

s1 = p0[s0]
p1 = p0[0]
MICROARCHITECTURAL NONINTERFERENCE

- Software controls data and timing labels
- Hardware ensures timing and microarchitecture observe noninterference

\[ \forall C_1, C_2 . (C_1 =_L C_2) \land (C_i \rightarrow C'_i) \]

\[ \implies (C'_1[\mu] =_L C'_2[\mu]) \land (C'_1[t] =_L C'_2[t]) \]
**MICROARCHITECTURAL NONINTERFERENCE**

\[ \forall C_1, C_2. (C_1 \equiv_L C_2) \land (C_i \rightarrow C'_i) \]

\[ \implies (C'_1[\mu] \equiv_L C'_2[\mu]) \land (C'_1[t] \equiv_L C'_2[t]) \]

- Implementation independent
  - Allows many performance optimizations (e.g. speculation)
- HW designer must prove implementation safe
## IFC Security Guarantees

<table>
<thead>
<tr>
<th>Security Guarantee</th>
<th>Requires Static Enforcement</th>
<th>Requires Compiler Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>TS-Noninterference</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>TS-Nonmalleability</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>Legacy Isolation</td>
<td>N</td>
<td>N</td>
</tr>
</tbody>
</table>
CONSTRaining DOWNGRADING

- Downgrades violate noninterference
  - *Declassification, Endorsement*
  - *Nonmalleable Information Flow* [Cecchetti et al. ’17]
  - *Robust Declassification + Transparent Endorsement*
CONSTRAINING DOWNGRADING

- Key Idea: *Compromised* data may not be downgraded
- Compromised = More secret than trustworthy
CONSTRaining DOWNGRADING

- Compromised = More secret than trustworthy
- Novel contribution:
  - More general re-label restrictions in dynamic label setting
  - Must keep pc uncompromised
CONTROL TRANSFER

- Required for sharing HW across security domains
- System Calls and other HW primitives provide support
- **Protection Rings** are an instance of a Lattice
- **Call Gates**
  - Generalized control transfer for a lattice policy
CALL GATES

- Upcalls
  - Establish return conditions a priori
  - Primitive for black-box timing mitigation [Zhang et al. ’12]
- Primitive for sandboxing
CALL GATES

- Downcalls
  - Analogous to system calls
  - Pre-register entry points

WE PROVIDE OTHER CONSTRAINTS ON CALL GATES TO PROVE NONMALLEABILITY
MORE TECHNICAL DETAILS IN PAPER / TECHNICAL REPORT

- Attacker model, low equivalence relation, more explicit restrictions and full proofs of *Timing-Sensitive Noninterference* (TS-Nonmalleable Information Flow)
- Discussion on Exception Handling + Asynchrony
- Example Programs + Call Gate Usage
FUTURE WORK

- Model multicore and concurrency ISA operations

- How do we verify that HW satisfies Microarchitectural NI?

- Existing tools not expressive enough (SecVerilog ’12, ’19)
CONCLUSION

- General and portable contract for secure SW design
  - **Microarchitectural NI** Security Condition Guides Secure HW Development

- Proof of **Timing-Sensitive Noninterference** and **Timing-Sensitive Nonmalleability**

- Practical primitives for downgrading and control transfer
  - Nonmalleability w/ Dynamic Labels

- Call Gates provide Generalized Control Transfer
THANK YOU!