CONTEXT-SENSITIVE FENCING:
SECURING SPECULATIVE EXECUTION VIA MICROCODE CUSTOMIZATION

Mohammadkazem Taram, Ashish Venkat, Dean Tullsen
University of California San Diego, University of Virginia
PERFORMANCE V.S. SECURITY
PERFORMANCE V.S. SECURITY
PERFORMANCE V.S. SECURITY

FORESHADOW

Security

Performance

MELTDOWN
- Leak secrets via side-channels + speculative execution
- Any modern processor with a Branch Predictor is vulnerable
int Kernel_api_( int x ){
    y = array2[array1[x] * 64];
}

SPECTRE V1 — BOUNDS CHECK BYPASS

4
```c
int Kernel_api_( 
int x )
{
    if ( x < array1_size ) //bounds check
        y = array2[array1[x] * 64];
}
```
```c
int Kernel_api_( int x ){
    Mispredicted if ( x < array1_size ) //bounds check
    y = array2[array1[x] * 64]; //not taken/fallthrough code
}
```
```c
int Kernel_api_( int x ){
    if ( x < array1_size ) //bounds check
        y = array2[array1[x] * 64]; //not taken/fallthrough code
}
```

Too late to recover — data is exposed via side-channels
CURRENT SPECTRE V1 MITIGATIONS

➤ Restricting Speculation Using Fences and Barriers:
CURRENT SPECTRE V1 MITIGATIONS

➤ Restricting Speculation Using Fences and Barriers:

```c
if ( x < array1_size)
    y = array2[array1[x] * 64];
```
CURRENT SPECTRE V1 MITIGATIONS

➤ Restricting Speculation Using Fences and Barriers:

```c
if ( x < array1_size)
    speculative_fence;

y = array2[array1[x] * 64];
```
Restricting Speculation Using Fences and Barriers:

```c
if ( x < array1_size )
    speculative_fence;

y = array2[array1[x] * 64];
```

Up to 10x Performance Overhead!

THIS WORK: CONTEXT SENSITIVE FENCING

➤ Surgically injects fence micro-ops
THIS WORK: CONTEXT SENSITIVE FENCING

➤ Surgically injects fence micro-ops

Only When Necessary
THIS WORK: CONTEXT SENSITIVE FENCING

➤ Surgically injects fence micro-ops

Only When Necessary

Right Type of Fence
THIS WORK: CONTEXT SENSITIVE FENCING

➤ Surgically injects fence micro-ops

- Only When Necessary
- Right Type of Fence
- No Recompilation
MICRO-OP STREAM CUSTOMIZATION BY CONTEXT-SENSITIVE DECODING

Native Instructions (e.g., inc [0x803ac])

Fetch → Instruction Decoder → Rename → Execute → WB

ld t0, [0x803ac]
add, t0, t0, 1
st [0x803ac], t0

"Context-Sensitive Decoding: On-Demand Microcode Customization for Security and Energy Management"
ISCA 2018, IEEE Micro Top Picks 2019
ISCA 2018, IEEE Micro Top Picks 2019
CONTEXT-SENSITIVE FENCING: AN EXAMPLE

- Fence Enforcement Configurations
- OS
- X86 Instructions
- Macro-op Dispatcher
  - Model Specific Registers
- Decoders
  - Regular Decoders
  - Context-Sensitive
  - MSROM
- Micro-op Fusion
- Micro-op Queue
  - beq
  - add
CONTEXT-SENSITIVE FENCING: AN EXAMPLE

Fence Enforcement Configurations

OS

X86 Instructions

inc [0x803ac]
(*ptr)++

Model Specific Registers

Macro-op Dispatcher

Decoders

Regular Decoders

Context-Sensitive Decoders

MSROM

Micro-op Fusion

Micro-op Queue

beq

add
CONTEXT-SENSITIVE FENCING: AN EXAMPLE

OS

Micro-op Queue

- beq
- add

Micro-op Fusion

MSROM

Context-Sensitive

Regular Decoders

Decoders

Model Specific Registers

Macro-op Dispatcher

Taint Tracking

X86 Instructions

inc [0x803ac]
(*ptr)++

Fence Enforcement Configurations

OS

Micro-op Instructions
CONTEXT-SENSITIVE FENCING: AN EXAMPLE

- **Fence Enforcement Configurations**
- **OS**
- **X86 Instructions**
- **Model Specific Registers**
- **Macro-op Dispatcher**
- **Taint Tracking**
- **Decoders**
- **Regular Decoders**
- **Context-Sensitive Decoders**
- **MSROM**
- **Micro-op Fusion**
- **Micro-op Queue**

X86 Instructions:
- `inc [0x803ac]`
- `ld t0,[0x803ac]`
- `add t0, t0, 1`
- `st [0x803ac], t0`

Micro-op Queue:
- `beq`
- `add`
CONTEXT-SENSITIVE FENCING: AN EXAMPLE

Fence Enforcement Configurations

Model Specific Registers

Macro-op Dispatcher

Taint Tracking

OS

X86 Instructions

Decoders

Regular Decoders

MSROM

Context-Sensitive Model

Specific Registers

Micro-op Dispatcher

Micro-op Fusion

Micro-op Queue

st
add
ld
FENCE
beq
add

inc [0x803ac]

ld t0,[0x803ac]

add t0, t0, 1

st [0x803ac], t0

FENCE

add
CONTEXT SENSITIVE FENCING

➤ Surgically injects fence micro-ops

No Recompilation

Right Type of Fence

Only When Necessary
## Existing Intel Fences

<table>
<thead>
<tr>
<th>Type of Fence</th>
<th>Instruction Opcode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Privileged Serializing Instructions</td>
<td>INVD</td>
<td>Invalidate Internal Caches</td>
</tr>
<tr>
<td></td>
<td>INVEPT</td>
<td>Invalidate Translations from EPT</td>
</tr>
<tr>
<td></td>
<td>INVLPG</td>
<td>Invalidate TLB Entries</td>
</tr>
<tr>
<td></td>
<td>INVVPID</td>
<td>Invalidate Translations Based on VPID</td>
</tr>
<tr>
<td></td>
<td>LiDT</td>
<td>Load Interrupt Descriptor Table Register</td>
</tr>
<tr>
<td></td>
<td>LGDT</td>
<td>Load Global Descriptor Table Register</td>
</tr>
<tr>
<td></td>
<td>LLDT</td>
<td>Load Local Descriptor Table Register</td>
</tr>
<tr>
<td></td>
<td>LTR</td>
<td>Load Task Register</td>
</tr>
<tr>
<td></td>
<td>MOV</td>
<td>Move to Control Register</td>
</tr>
<tr>
<td></td>
<td>MOV</td>
<td>Move to Debug Register</td>
</tr>
<tr>
<td></td>
<td>WBINVD</td>
<td>Write Back and Invalidate Cache</td>
</tr>
<tr>
<td></td>
<td>WRMSR</td>
<td>Write to Model Specific Register</td>
</tr>
<tr>
<td>Non-Privileged Serializing Instructions</td>
<td>CPUID</td>
<td>CPU Identification</td>
</tr>
<tr>
<td></td>
<td>IRET</td>
<td>Interrupt Return</td>
</tr>
<tr>
<td></td>
<td>RSM</td>
<td>Resume from System Management Mode</td>
</tr>
<tr>
<td>Memory Ordering Instructions</td>
<td>SFENCE</td>
<td>Store Fence</td>
</tr>
<tr>
<td></td>
<td>LFENCE</td>
<td>Load Fence</td>
</tr>
<tr>
<td></td>
<td>MFENCE</td>
<td>Memory Fence</td>
</tr>
</tbody>
</table>
**BUT WHAT FENCE SHOULD WE USE?**

### Existing Intel Fences

<table>
<thead>
<tr>
<th>Type of Fence</th>
<th>Instruction Opcode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Privileged Serializing Instructions</td>
<td>INVD</td>
<td>Invalidate Internal Caches</td>
</tr>
<tr>
<td></td>
<td>INVEPT</td>
<td>Invalidate Translations from EPT</td>
</tr>
<tr>
<td></td>
<td>INVLPG</td>
<td>Invalidate TLB Entries</td>
</tr>
<tr>
<td></td>
<td>LIDT</td>
<td>Inhibit Address Translation Based on VPID</td>
</tr>
<tr>
<td></td>
<td>LGDT</td>
<td>Load Interrupt Descriptor Table Register</td>
</tr>
<tr>
<td></td>
<td>LLDT</td>
<td>Load Global Descriptor Table Register</td>
</tr>
<tr>
<td></td>
<td>LDR</td>
<td>Load Local Descriptor Table Register</td>
</tr>
<tr>
<td></td>
<td>MOV</td>
<td>Move to Debug Register</td>
</tr>
<tr>
<td></td>
<td>WBINVD</td>
<td>Write Back and Invalidate Cache</td>
</tr>
<tr>
<td></td>
<td>WRISR</td>
<td>Write to Model Specific Register</td>
</tr>
<tr>
<td></td>
<td>IRET</td>
<td>Interrupt Return</td>
</tr>
<tr>
<td></td>
<td>RSM</td>
<td>Resume from System Management Mode</td>
</tr>
<tr>
<td>Memory Ordering Instructions</td>
<td>SFENCE</td>
<td>Store Fence</td>
</tr>
<tr>
<td></td>
<td>LFENCE</td>
<td>Load Fence</td>
</tr>
<tr>
<td></td>
<td>MFENCE</td>
<td>Memory Fence</td>
</tr>
</tbody>
</table>

- **Require Privileged Access**
- **Clobber Architectural Registers**
- **Enforced Early in the Pipeline**
EXISTING FENCES: SERIALIZING INSTRUCTIONS (SI)

- Enforced early in the pipeline
- Examples:
  - All Serializing Instructions
  - Intel’s MFENCE
  - Intel’s SFENCE
EXISTING FENCES: INTEL LFENCE

- Enforced early in the pipeline
- Example:
  - Intel’s LFENCE
LATE ENFORCEMENT FENCES

➤ Shifts fence enforcement towards the leaking structure
➤ Reduces the impact on other instructions
LATE ENFORCEMENT FENCES

➤ Shifts fence enforcement towards the leaking structure
➤ Reduces the impact on other instructions
LATE ENFORCEMENT FENCES

➤ Shifts fence enforcement towards the leaking structure
➤ Reduces the impact on other instructions
NEWLY PROPOSED FENCES

- Load-Store Queue LFENCE (LSQ-LFENCE)
- Load-Store Queue MFENCE (LSQ-MFENCE)
- Reservation Station Fence (RSFENCE)
- Cache Fence (CFENCE)
CACHE FENCE (CFENCE)

- Allows all the load and stores to pass
- *CFENCE* labels any subsequent load as a *non-modifying load*
- *allows* non-modifying loads to pass through the *CFENCE*
- Non-modifying loads are restricted from modifying the cache state.
CACHE FENCE (CFENCE)

- If a cache hit, read the contents of the cache, doesn't change meta-data (e.g., LRU age bits).
- If a cache miss, mark as uncacheable, allow mem read without altering the cache state.

Normal Load

**Cache Controller**

Hit?

- Hit: Update Metadata (LRU bits) → Serve the request
- Miss: Fetch Cache Block from memory → Update Cache

Diagram:

1. Hit? (Diamond)
   - If Hit: Update Metadata (LRU bits) → Serve the request
   - If Miss: Fetch Cache Block from memory → Update Cache
CACHE FENCE (CFENCE)

For a non-modifying load:

- If a cache hit, read the contents of the cache, doesn't change meta-data (e.g., LRU age bits).
- If a cache miss, mark as uncacheable and allow mem read without altering the cache state.

Flowchart:

1. **Hit?**
   - **Hit**:
     - Serve the request
   - **Miss**:
     - Fetch Cache Block from memory
Our CFENCE reduces the incurred performance overhead by 2.3X, bringing down the execution time overhead from 48% to 21%.
CONTEXT SENSITIVE FENCING

➤ Surgically injects fence micro-ops

No Recompilation

Right Type of Fence

Only When Necessary
FENCE FREQUENCY OPTIMIZATIONS

➤ Liberal Injection

➤ Injects fences before all the loads of a program

➤ completely stops speculation
FENCE FREQUENCY OPTIMIZATIONS

➤ Liberal Injection

➤ Injects fences before all the loads of a program

➤ completely stops speculation

```java
jeq
Fence
ld
add
Fence
ld
Fence
ld
```
FENCE FREQUENCY OPTIMIZATIONS

➤ Basic Block-Level Fence Insertion*

➤ Speculation begins with a branch prediction
➤ A fence between branch and subsequent loads

```
jeq
Fence
ld
add
Fence
ld
```

* Targeted Optimization — Only protects against variant 1
FENCE FREQUENCY OPTIMIZATIONS

➤ Basic Block-Level Fence Insertion

➤ Speculation begins with a branch prediction
➤ We want a fence between each branch and subsequent loads

```
jeq
Fence
ld
add
ld
ld
```

* Targeted Optimization — Only protects against variant 1
FENCE FREQUENCY OPTIMIZATIONS

➤ Taint-Based Fence Insertion

➤ Even one fence per basic block is too conservative

➤ Attacker performs operations based on untrusted data
  (e.g., attacker controlled out of bound index)

➤ Insert fences for only vulnerable loads that operate on untrusted data

➤ Dynamic Information Flow Tracker (DIFT)
DLIFT- AN INFORMATION FLOW TRACKER FOR SPECTRE ERA

➤ Classic Information Flow Trackers
  ➤ Maintain and Evaluate Taints at Late Stages of the Pipeline
  ➤ Not so useful for Spectre!
DLIFT—AN INFORMATION FLOW TRACKER FOR SPECTRE ERA

➤ Classic Information Flow Trackers
  ➤ Maintain and Evaluate Taints at Late Stages of the Pipeline
  ➤ Not so useful for Spectre!
DLIFT—AN INFORMATION FLOW TRACKER FOR SPECTRE ERA

➤ Classic Information Flow Trackers
  ➤ Maintain and Evaluate Taints at Late Stages of the Pipeline
  ➤ Not so useful for Spectre!

Detect The Threat before it’s too late.
DECODER-LEVEL (SPECULATIVE) INFORMATION FLOW TRACKER

**Fetch/Decode**

- Fetch
- Macro-to-Micro

**Execute**

- Taint Evaluator
- Reg. File
- TLB

**Commit**

- Commit Logic

### Speculative Taint Map

<table>
<thead>
<tr>
<th>Register</th>
<th>Tainted</th>
</tr>
</thead>
<tbody>
<tr>
<td>rax</td>
<td>No</td>
</tr>
<tr>
<td>rbx</td>
<td>No</td>
</tr>
</tbody>
</table>

### Arch. Taint Map/RF

<table>
<thead>
<tr>
<th>Register</th>
<th>Tainted</th>
</tr>
</thead>
<tbody>
<tr>
<td>rax</td>
<td>No</td>
</tr>
<tr>
<td>rbx</td>
<td>Yes</td>
</tr>
</tbody>
</table>
DECODER-LEVEL (SPECULATIVE) INFORMATION FLOW TRACKER

Fetch/Decode

Macro-to-Micro

Fetch

add (%rbx), %rax

Taint Evaluator

Reg. File

Taint

TLB

Taint

Execute

Commit

Commit Logic

Speculative Taint Map

<table>
<thead>
<tr>
<th>Register</th>
<th>Tainted</th>
</tr>
</thead>
<tbody>
<tr>
<td>rax</td>
<td>No</td>
</tr>
<tr>
<td>rbx</td>
<td>No</td>
</tr>
</tbody>
</table>

Arch. Taint Map/RF

<table>
<thead>
<tr>
<th>Register</th>
<th>Tainted</th>
</tr>
</thead>
<tbody>
<tr>
<td>rax</td>
<td>No</td>
</tr>
<tr>
<td>rbx</td>
<td>Yes</td>
</tr>
</tbody>
</table>
DECODER-LEVEL (SPECULATIVE) INFORMATION FLOW TRACKER

Fetch/Decode
- Fetch
- Macro-to-Micro
- Add (%rbx), %rax
- Not Tainted

Speculative Taint Map

<table>
<thead>
<tr>
<th>Register</th>
<th>Tainted</th>
</tr>
</thead>
<tbody>
<tr>
<td>rax</td>
<td>No</td>
</tr>
<tr>
<td>rbx</td>
<td>No</td>
</tr>
</tbody>
</table>

Execute
- Taint Evaluator
- Reg. File
- TLB
- Taint

Arch. Taint Map/RF

<table>
<thead>
<tr>
<th>Register</th>
<th>Tainted</th>
</tr>
</thead>
<tbody>
<tr>
<td>rax</td>
<td>No</td>
</tr>
<tr>
<td>rbx</td>
<td>Yes</td>
</tr>
</tbody>
</table>
DECODER-LEVEL (SPECULATIVE) INFORMATION FLOW TRACKER

Fetch/Decode

Fetch → Macro-to-Micro → Fence-Free Translation

Not Tainted

Fetch

Execute

Taint Evaluator

ld t1, (%rbx)
add t1, %rax, %rax

Reg. File

Taint

TLB

Taint

Commit

Commit Logic

Speculative Taint Map

<table>
<thead>
<tr>
<th>Register</th>
<th>Tainted</th>
</tr>
</thead>
<tbody>
<tr>
<td>rax</td>
<td>No</td>
</tr>
<tr>
<td>rbx</td>
<td>No</td>
</tr>
</tbody>
</table>

Arch. Taint Map/RF

<table>
<thead>
<tr>
<th>Register</th>
<th>Tainted</th>
</tr>
</thead>
<tbody>
<tr>
<td>rax</td>
<td>No</td>
</tr>
<tr>
<td>rbx</td>
<td>Yes</td>
</tr>
</tbody>
</table>
DECODER-LEVEL (SPECULATIVE) INFORMATION FLOW TRACKER

Fetch/Decode

Fetch

Macro-to-Micro

Taint Evaluator

Execute

ld t1, (%rbx)
add t1, %rax, %rax

Commit Logic

Commit

Arch. Taint Map/RF

<table>
<thead>
<tr>
<th>Register</th>
<th>Tainted</th>
</tr>
</thead>
<tbody>
<tr>
<td>rax</td>
<td>No</td>
</tr>
<tr>
<td>rbx</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Speculative Taint Map

<table>
<thead>
<tr>
<th>Register</th>
<th>Tainted</th>
</tr>
</thead>
<tbody>
<tr>
<td>rax</td>
<td>No</td>
</tr>
<tr>
<td>rbx</td>
<td>No</td>
</tr>
</tbody>
</table>

Fence-Free Translation

Reg. File

TLB

Taint

Taint
DECODER-LEVEL (SPECULATIVE) INFORMATION FLOW TRACKER

Fetch/Decode

Macro-to-Micro Translation

Fetch

Not Tainted

Speculative Taint Map

<table>
<thead>
<tr>
<th>Register</th>
<th>Tainted</th>
</tr>
</thead>
<tbody>
<tr>
<td>rax</td>
<td>No</td>
</tr>
<tr>
<td>rbx</td>
<td>No</td>
</tr>
</tbody>
</table>

Execute

Taint Evaluator

ld t1, (%rbx)
add t1, %rax, %rax

Under-Tainted?

Reg. File

Taint

TLB

Taint

Commit

Commit Logic

<table>
<thead>
<tr>
<th>Register</th>
<th>Tainted</th>
</tr>
</thead>
<tbody>
<tr>
<td>rax</td>
<td>No</td>
</tr>
<tr>
<td>rbx</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Arch. Taint Map/RF
DECODER-LEVEL (SPECULATIVE) INFORMATION FLOW TRACKER

Fetch/Decode
- Fetch
- Macro-to-Micro Translation
- Not Tainted

Execute
- Taint Evaluator
  - Reg. File
  - TLB
  - Under-Tainted?

Commit
- Logic
- Commit

Speculative Taint Map

<table>
<thead>
<tr>
<th>Register</th>
<th>Tainted</th>
</tr>
</thead>
<tbody>
<tr>
<td>rax</td>
<td>No</td>
</tr>
<tr>
<td>rbx</td>
<td>No</td>
</tr>
</tbody>
</table>

Arch. Taint Map/RF

<table>
<thead>
<tr>
<th>Register</th>
<th>Tainted</th>
</tr>
</thead>
<tbody>
<tr>
<td>rax</td>
<td>No</td>
</tr>
<tr>
<td>rbx</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Example code:
```
ld t1, (%rbx)
add t1, %rax, %rax
```
DECODER-LEVEL (SPECULATIVE) INFORMATION FLOW TRACKER

Fetch/Decode

Fetch

Macro-to-Micro

Fence-Free Translation

Not Tainted

Execute

Taint Evaluator

Under-Tainted?

Reg. File

Taint

TLB

Taint

Commit

Commit Logic

Register Tainted

<table>
<thead>
<tr>
<th>Register</th>
<th>Tainted</th>
</tr>
</thead>
<tbody>
<tr>
<td>rax</td>
<td>No</td>
</tr>
<tr>
<td>rbx</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Speculative Taint Map

Arch. Taint Map/RF

<table>
<thead>
<tr>
<th>Register</th>
<th>Tainted</th>
</tr>
</thead>
<tbody>
<tr>
<td>rax</td>
<td>No</td>
</tr>
<tr>
<td>rbx</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Taint Recovery Copy

ld t1, (%rbx)
add t1, %rax, %rax

ld t1, (%rbx)
add t1, %rax, %rax
DECODER-LEVEL (SPECULATIVE) INFORMATION FLOW TRACKER

Fetch/Decode

Fetch

Macro-to-Micro

Fence-Free Translation

Not Tainted

Execute

Taint Evaluator

Under-Tainted?

Reg. File

Taint

TLB

Taint

Commit

Commit Logic

Speculative Taint Map

Register | Tainted |
---------|---------|
rax      | No      |
rbx      | Yes     |

Arch. Taint Map/RF

Register | Tainted |
---------|---------|
rax      | No      |
rbx      | Yes     |
DECODER-LEVEL (SPECULATIVE) INFORMATION FLOW TRACKER

### Fetch/Decode

- **Fetch**
- Macro-to-Micro

### Execute

- **Taint Evaluator**
- Under-Tainted?

### Commit

- **Commit Logic**

#### Speculative Taint Map

<table>
<thead>
<tr>
<th>Register</th>
<th>Tainted</th>
</tr>
</thead>
<tbody>
<tr>
<td>rax</td>
<td>No</td>
</tr>
<tr>
<td>rbx</td>
<td>Yes</td>
</tr>
</tbody>
</table>

#### Fence-Free Translation

- `add (%rbx), %rax`

#### Taint Recovery Copy

- `ld Flush bx
  add t1, %rax, %rax`
DECODER-LEVEL (SPECULATIVE) INFORMATION FLOW TRACKER

Fetch/Decode

Fetch

Macro-to-Micro

Fence Injection

FENCE
ld t1, (%rbx)
add t1, %rax, %rax

Taint Evaluator

Under-Tainted?

Reg. File

Taint

TLB

Taint

Execute

Commit

Commit Logic

Speculative Taint Map

Register | Tainted
---|---
rax | No
rbx | Yes

Arch. Taint Map/RF

Register | Tainted
---|---
rax | No
rbx | Yes
➤ Taint-Based CFENCE injection reduces the performance overhead to just 7.7%
CONTEXT-SENSITIVE FENCING

Low Performance Overhead
CONTEXT-SENSITIVE FENCING

- Low Performance Overhead
- No Recompilation
CONTEXT-SENSITIVE FENCING

- Low Performance Overhead
- No Recompilation
- Minimal Changes to Processor
THANKS!
QUESTIONS?