Speculative Interference Attacks: Breaking Invisible Speculation Schemes

Mohammad Behnia, \textsuperscript{1} Prateek Sahu, Riccardo Paccagnella, Jiyong Yu, Zirui Zhao, \textsuperscript{2} Xiang Zou, \textsuperscript{1} Thomas Unterluggauer, Josep Torrellas, \textsuperscript{2} Carlos Rozas, \textsuperscript{3} Adam Morrison, \textsuperscript{2} Frank McKeen, \textsuperscript{2} Fangfei Liu, \textsuperscript{4} Ron Gabor, Christopher W. Fletcher, \textsuperscript{2} Abhishek Basak, \textsuperscript{5} Alaa Alameldeen

University of Illinois at Urbana-Champaign, \textsuperscript{1} University of Texas at Austin, \textsuperscript{2} Intel Corporation, \textsuperscript{3} Tel Aviv University, \textsuperscript{4} Toga Networks, \textsuperscript{5} Simon Fraser University
Introduction

• Microarchitectural Side Channels
  • Cache-based

• Spectre Attack
  • Variant 1

• Advantages to Attacker
  • Persistent State Change
  • Shared Cache Hierarchy

\[
\text{if}(i < N) \\
\text{secret} = A[i] \\
k = B[\text{secret}]
\]
Invisible Speculation Schemes

• Invisible Speculation Schemes
  • Mechanisms to thwart speculative, persistent cache state changes

• Example: Delay-On-Miss
  • Any cache state change is deferred until load becomes non-speculative
  • Loads that hit in the L1 forward results to dependent instructions
Speculative Interference Attacks

• **Observation:** Secret-Dependent timing effects can be monitored indirectly by how they interact with **older non-speculative instructions**

• **Idea:** By creating a “ripple effect” we can transform transient interactions into persistent state changes in the cache even with invisible speculation enabled
Speculative Interference Attacks

- Can induce contention on a large number of microarchitectural resources using different instructions.
- If this “ripple effect” targets non-speculative memory accesses it can affect their ordering.
Attack Framework

```c
interference_target;
if (i < N) { // mispredict
    secret = A[i]; // access
    interference_gadget(secret);
}
```
Story of this Paper

• Speculative Interference Attacks undermine the security of a prominent family of Hardware Spectre Defenses

• 1. Mis-speculated younger instructions can affect the timing of older bound-to-retire instructions including memory operations

• 2. Altering timing of memory operations can change the order of one memory operation relative to others and expose secrets via persistent changes to cache state
Outline

• Attack Variants
• D-Cache PoC
• Defenses
Interference Gadgets

- **Type 2**: Secret-dependent interference time

```
secret == 1

secret = load(...)  
x = load(&S[secret])  
f(x)

secret == 0

interference target;
```

(a) Attack framework
Interference Gadgets

• **Type 1**: Operand-dependent resource usage patterns
  
  ```
  secret = load(...) 
  f(secret)
  ```

• **Type 3**: Interference existence is secret-dependent
  
  ```
  secret = load(...) 
  if(secret) 
    f()
  ```

```
interference_target;
if (i < N) { // mispredict
  secret = A[i]; // access
  interference_gadget(secret);
}
```

(a) Attack framework
Gadget + Target

\[ z = \ldots \]
\[ A = f(z) \]
\[ y = \text{load}(A) \]
\[ \text{if } (i < N): \]
  \[ \text{secret} = \text{load}(&T\text{Array}[i]) \]
  \[ x = \text{load}(&S[\text{secret*64}]) \]
  \[ f'(x) \]
Gadget + Target

\[
\begin{align*}
  z &= \ldots \\
  A &= f(z) \\
  y &= \text{load}(A) \\
  \text{if } (i < N): \\
    \text{secret} &= \text{load}(&\text{ToArray}[i]) \\
    x &= \text{load}(&S[\text{secret}*64]) \\
    f'(x)
\end{align*}
\]
Gadget + Target

\[ z = \ldots \]
\[ A = f(z) \]
\[ y = \text{load}(A) \]
\[ \text{if } (i < N): \]
\[ \text{secret} = \text{load}(&\text{Array}[i]) \]
\[ x = \text{load}(&S[\text{secret} \times 64]) \]
\[ f'(x) \]
$z = \ldots$
$A = f(z)$
$y = \text{load}(A)$
\text{if } (i<N):$
  \text{secret} = \text{load}(&\text{Array}[i])$
  \text{x} = \text{load}(&S[\text{secret}*64])$
  f'(x)$
Interference Targets

- Victim L1 D-cache and L1 I-cache access streams
- Can also manifest in permutations of D-cache and I-cache memory access patterns

![Diagram showing cache levels and traffic patterns]
Vulnerability Matrix

<table>
<thead>
<tr>
<th>Target Variant</th>
<th>Reference Load</th>
</tr>
</thead>
<tbody>
<tr>
<td>$V\uparrow D - V\uparrow D$</td>
<td>$V\uparrow D$</td>
</tr>
<tr>
<td>$V\uparrow D - V\uparrow I$</td>
<td>$V\uparrow D$</td>
</tr>
<tr>
<td>$V\uparrow D - A\uparrow D$</td>
<td>$A\uparrow D$</td>
</tr>
<tr>
<td>$V\uparrow I - A\uparrow D$</td>
<td>$A\uparrow D$</td>
</tr>
</tbody>
</table>

**$V\uparrow D$**: Victim Data Access

**$V\uparrow I$**: Victim Instruction Fetch

**$A\uparrow D$**: Attacker Data Access

### Vulnerability Matrix

<table>
<thead>
<tr>
<th>Gadget</th>
<th>Target Variant</th>
<th>Reference Load</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type 2 (NPEU)</td>
<td>InvisiSpec (Spectre, DoM (non-TSO), SafeSpec (WFB))</td>
<td>All</td>
</tr>
<tr>
<td>Type 1 (MSHR)</td>
<td>InvisiSpec (Spectre), SafeSpec (WFB)</td>
<td>InvisiSpec, SafeSpec, MuonTrap</td>
</tr>
<tr>
<td>Type 1 (RS)</td>
<td>–</td>
<td>InvisiSpec, SafeSpec, MuonTrap</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Gadget</th>
<th>$V\uparrow D - V\uparrow D$</th>
<th>$V\uparrow I - V\uparrow D$</th>
<th>$V\uparrow D - A\uparrow D$</th>
<th>$V\uparrow I - A\uparrow D$</th>
</tr>
</thead>
</table>
| InvisiSpec (Spectre) | All | All | – | }
Victim and Attacker Threads on Separate Cores

Shared memory addresses A and B that map to same LLC set and slice

Victim issues A-B or B-A using secret dependent load ordering

Attacker primes and probes replacement policy state of LLC set to identify issue order

\[
\begin{align*}
\text{z} &= \ldots \\
\text{A} &= f(z) \\
\text{y} &= \text{load}(A) \\
\text{if } (i < N): \\
\quad & \text{secret} = \text{load}(&\text{TArray}[i]) \\
\quad & \text{x} = \text{load}(&\text{S}[\text{secret} \times 64]) \\
\quad & f'(x)
\end{align*}
\]
D-Cache PoC Interference Gadget

```c
+- A = load(interference_target())
 |    // VSQRTPD dependency chain
 |    if (...) //miss-speculation
 |    secret = load(...);
 |    x = load(&S[secret]); //hit-miss
+- interference_gadget(x);
    // VSQRTPD ready to execute
```

**VSQRTPD**
- 1 micro-op execution port 0
- Latency of 15–16 cycles
- Reciprocal throughput of 9–12 cycles
D-Cache PoC Receiver Protocol

- Quad Age LRU Replacement Policy
  - QLRU_H11_M1_R0_U0

<table>
<thead>
<tr>
<th></th>
<th>EV0</th>
<th>EV1</th>
<th>EV2</th>
<th>EV3...EV11</th>
<th>EV12</th>
<th>EV13</th>
<th>EV14</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a) After Prime Sequence</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>(b) Victim Access A-B</td>
<td>B</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>(c) Probe with EV15-EV29</td>
<td>B</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>2</td>
</tr>
</tbody>
</table>
D-Cache PoC End-to-End

Victim (Core 1)

\[
A = \text{contention}\_\text{target}() \\
y = \text{load}(A) \\
B = \text{fixed}\_\text{latency}() \\
z = \text{load}(B) \\
N = \text{pointer}\_\text{chase}() \\
\text{if} (i < N): //\text{mis-spec} \\
\quad \text{secret} = \text{load}(\text{tgt}[i]) \\
\quad x = \text{load}(&S[\text{secret}\times64]) \\
\quad //\text{miss(secret=0)} \Rightarrow A-B \\
\quad //\text{hit(secret=1)} \Rightarrow B-A \\
\quad \text{interference}\_\text{gadget}()
\]

Attacker (Core 2)

1. \text{find}\_\text{eviction}\_\text{set}(A, B)
2. \text{train}\_\text{branch}\_\text{predictor}()
3. \text{prime}\_\text{llc}\_\text{set}()
4. \text{probe}\_\text{llc}\_\text{set}()
5. \text{load}(A), \text{load}(B)

\text{if}(A\ \text{cache}\_\text{hit} & B\ \text{cache}\_\text{miss}) \\
\quad \text{secret} = 1 \\
\text{if}(A\ \text{cache}\_\text{miss} & B\ \text{cache}\_\text{hit}) \\
\quad \text{secret} = 0
D-Cache PoC Bitrate

- Intel Core i7-7700 Kaby Lake CPU with 4 physical cores @ 3.6GHz
- Unified Reservation Station, 8 execution ports
- POC Attacker and Victim Threads run in multi-core configuration

![Figure 10: D-Cache PoC channel error vs. bit rate.](image)
Discussion of Defenses

- Ideal Invisible Speculation: LLC access pattern being invariant of speculation
- Basic Defense: Fences to prevent issue of ROB instructions until window becomes non-speculative
- More advanced Defense:
  - Not Delaying Older Instructions:
    - Priority Tagging based on speculative window in RS
    - Scheduler to predict speculative interference
  - Not Releasing Resources Early:
    - Operand independent executions times
Thank You