InvisiSpec: Making Speculative Execution Invisible in the Cache Hierarchy
Mengia Yan, Jiho Choi, Dimitrios Skarlatos, Adam Morrison, Christopher W. Fletcher, Josep Torrellas
University of Illinois at Urbana-Champaign • Tel Aviv University

1. INTRODUCTION
Speculative execution attacks
Hardware speculation offers a major surface for micro-architectural covert and side channel attacks.

Goal
Efficiently defend against hardware speculation attacks

Highlights
- Generalization of speculative execution attacks
- InvisiSpec
  - Make unsafe loads invisible in cache hierarchy
  - Novel mechanisms to ensure invisible loads do not violate memory consistency
  - Optimizations to reduce the overhead of InvisiSpec

2. BACKGROUND
An example of Spectre attack
Victim code
```
if (x < array1_size)
    y = array2[array1[x] * 4096];
```

Attack procedure
1) Mistrain: train the branch predictor to be taken
2) Exploit: invoke the branch with a malicious value of x out of bounds of array1
3) Side channel: measure access latency to array2 to determine the value of array1[x]

3. ATTACK ANALYSIS
Generalization of attacks
- transient instructions: speculatively-executed instructions that are destined to be squashed
- Speculative execution attack exploits side effects of transient instructions

Source of transient instructions

<table>
<thead>
<tr>
<th>Attack</th>
<th>What Creates the Transient Instructions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spectre</td>
<td>Control-flow misprediction</td>
</tr>
<tr>
<td>Mislabeled</td>
<td>Li Terminal Fault</td>
</tr>
<tr>
<td>Lazy Floating</td>
<td>Virtual memory exception</td>
</tr>
<tr>
<td>Rogue System</td>
<td>Exception reading a disabled or privileged register</td>
</tr>
<tr>
<td>Register Read</td>
<td>Address alias between a load and an earlier store</td>
</tr>
<tr>
<td>Speculative</td>
<td>Any event that can cause a squash</td>
</tr>
<tr>
<td>Futuristic</td>
<td></td>
</tr>
</tbody>
</table>

Futurist attack model: an attacker can exploit any speculative load. It includes all existing attacks and future speculative execution attacks.

4. INVISISPEC DESIGN
Load is issued to memory

- Visibility Point
- Window of Suppressed Visibility
- Load is safe

1) visibility point: a point in time when an unsafe load can transition to a safe load
- In Spectre attack model, it is when all prior control-flow instructions resolve
- In Futuristic attack model, it is when the instruction cannot be squashed anymore

Challenge 1: A load is unsafe before this point. Need to be invisible in the cache hierarchy

2) window of suppressed visibility: the time period before a load makes itself visible

Challenge 2: A load can not receive invalidations in this window. Risk of memory consistency violation

Making unsafe loads invisible

- An unsafe load issues Spec-GetS
  - No modification to cache states
  - Data is stored in Speculative Buffer (SB)

Maintaining memory consistency
- Making loads visible using Validations or Exposures
  - Exposure:
    - Issue a normal coherence transaction at visibility point
  - Validation:
    - Issue a normal coherence transaction at visibility point
    - Compare the up-to-date data and the one in the SB
    - If mismatch, squash the load as it violates memory consistency model

5. OPTIMIZATIONS
Transform validations to exposures
- Opportunity: Under TSO, we do not need validation if a load will not violate memory consistency model
- Which loads? The load that when it is issued, all the previous loads have received the data they requested

Overlap validations and exposures
An exposure can overlap with all subsequent exposure transactions up to, and including, the next validation transaction

Reduce Main-Memory Accesses
- Add a per-core LLC SB
- Validations/Exposures get data from LLC-SB, skipping the extra DRAM accesses

6. EVALUATION

<table>
<thead>
<tr>
<th>Names</th>
<th>Configurations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base</td>
<td>Conventional, insecure baseline processor</td>
</tr>
<tr>
<td>Fe-Sp</td>
<td>Insert a fence after every indirect/conditional branch</td>
</tr>
<tr>
<td>IS-Sp</td>
<td>USL modifies only SB, and is made visible after all the preceding branches are resolved</td>
</tr>
<tr>
<td>Fe-Fu</td>
<td>Insert a fence before every load instruction</td>
</tr>
<tr>
<td>IS-Fu</td>
<td>USL modifies only SB, and is made visible when it is either non-speculative or spec non-squashable</td>
</tr>
</tbody>
</table>

Normalized Execution Time

Normalized Networks Traffic

- Average slowdown under TSO

Using Fences InvisiSpec

<table>
<thead>
<tr>
<th>Spectre attacks</th>
<th>74%</th>
<th>21%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Futuristic attacks</td>
<td>208%</td>
<td>72%</td>
</tr>
</tbody>
</table>

7. FUTURE WORK
Improve InvisiSpec to reduce its execution overhead. Potential directions:
- Reduce the usage of InvisiSpec’s mechanisms on the loads that can be proven safe in advance
- Redesign InvisiSpec’s mechanisms to be more aggressive