WiP: Isolating Speculative Data from Microarchitectural Covert Channels

Kristin Barber, Anys Bacha*, Li Zhou, Yinqian Zhang, Radu Teodorescu
Department of Computer Science and Engineering
The Ohio State University
http://arch.cse.ohio-state.edu
*University of Michigan
Motivation

- Speculative execution is pervasive in OOO processors
- Fundamental to performance

- Transient execution leave traces in the micro architectural state
- Covert channel used to leak information to attacker
- We now know it is vulnerable to a wide range of attacks

- We have to re-invent speculation with security in mind and little performance impact
Outline

- Threat Model
- Design
- Evaluation
- Conclusion
Outline

• Threat Model
• Design
• Evaluation
• Conclusion
• Attack vectors:
  — Illegal memory access (Meltdown)
  — Illegal control flow (Spectre)

• Sensitive data resides anywhere in the memory hierarchy
  — Accessed through a transient/mis-speculated instruction

• Any covert channel can be used to exfiltrate secret data:
  — Caches, ALUs, SIMD units, TLBs, etc.

• Out-of-scope: leakage of data retrieved through non-transient instructions
Existing Solutions

- Software-only mitigation solutions
  - Generally very high overhead for good coverage
  - Ad-hoc and specific to exploits, rely on manual insertion, static analysis shown to miss corner cases.

- Existing hardware solutions
  - Lower-overhead, better coverage
  - Mostly focused on closing specific covert channels
Outline

- Threat Model
- Design
- Evaluation
- Conclusion
Defense Strategy: Main Idea

A more general solution that prevents covert channel formation

Key Observation:

- All covert channels have dependences on secret data
- Restrict speculative data use by dependent instructions
Defense Strategy: Main Idea

- Preventing covert channel formation:
  - Monitor speculative status of Load instructions
  - Forward data to dependents only when “safe”
- What we consider “safe” is implementation-dependent
  - Two schemes, different performance/security tradeoffs
Outline

- Threat Model
- Design
  - Conservative Scheme
  - Optimized Scheme
- Evaluation
- Conclusion
Outline

- Threat Model
- Design
  - Conservative Scheme
  - Optimized Scheme
- Evaluation
- Conclusion
Conservative Scheme

- Speculative Loads are issued and executed normally
- When data returns from memory (cache)
  - Register file is updated
  - Delay forwarding data to dependent instructions
- When Load commits, forward to dependents
- All data guaranteed to be non-speculative before use
- Downside: relatively large performance impact
Conservative Scheme

- Impact on Wakeup/Select Logic
- Baseline: LD-dependents speculatively woken up on Select
  — Assuming that LD will be hit
- Wakeup delayed until retirement
Outline

- Threat Model
- Design
  - Conservative Scheme
  - Optimized Scheme
- Evaluation
- Conclusion
• **Observation:** Most Loads are safe earlier than retirement

• Define Early Resolution Point (ERP)
  — All older branches have resolved
  — All older loads and stores have had addresses computed
  — No branch mispredictions or memory-access exceptions

• Forwarding of speculative data allowed after ERP

• Much lower performance impact, equivalent security
SpecShield Hardware Support

No changes to the memory hierarchy, coherence protocol, consistency guarantees, etc.

SpecShield Changes/Additions
Outline

• Threat Model
• Design
• Evaluation
• Conclusion
Evaluation Methodology

Experimental Platform

- Simulator: gem5, full-system mode, Ubuntu 14.04 OS
- Benchmarks: spec2006, reference input set

<table>
<thead>
<tr>
<th>CPU Architecture</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU Clock</td>
<td>2GHz</td>
</tr>
<tr>
<td>L1 ICache</td>
<td>32KB (4-way)</td>
</tr>
<tr>
<td>L1 DCache</td>
<td>32KB (8-way)</td>
</tr>
<tr>
<td>L2 Cache</td>
<td>2MB (16-way)</td>
</tr>
<tr>
<td>Issue Width</td>
<td>8</td>
</tr>
<tr>
<td>ROB Entries</td>
<td>192</td>
</tr>
<tr>
<td>Branch Predictor</td>
<td>LTAGE</td>
</tr>
<tr>
<td>LSQ Entries</td>
<td>32</td>
</tr>
<tr>
<td>IQ Entries</td>
<td>64</td>
</tr>
<tr>
<td>BTB Entries</td>
<td>4096</td>
</tr>
<tr>
<td>dTLB Entries</td>
<td>64</td>
</tr>
<tr>
<td>iTLB Entries</td>
<td>64</td>
</tr>
<tr>
<td>FP Registers</td>
<td>256</td>
</tr>
<tr>
<td>Int Registers</td>
<td>256</td>
</tr>
</tbody>
</table>
**Performance**

![Graph showing normalized execution time for various computer architecture benchmarks.]

- **LFENCE > 200%**
- **SpecShieldSTL 55%**
- **SpecShieldERP 18%**

- Benchmarks with low miss rates most impacted
Cache Latency for Spectre Attack

- Spectre-attack, using cache as covert channel
- Exfiltrated value visible in access latency
- Secret value no longer appears in the cache channel
Conclusions

- Microarchitectural framework for preventing transient execution attacks on arbitrary memory
- SpecShield is more general
  - Unlike prior work that has focused on closing specific covert channels, SpecShield controls all speculative data-flow within the pipeline, preventing channel formation.
- SpecShield is easier to implement
  - No changes to the memory hierarchy, coherence protocol, consistency guarantees, etc.
- Possibility for flexibility and security policies