Replicating and Mitigating Spectre Attacks on an Open Source RISC-V Microarchitecture

CARRV 2019 – June 22nd, 2019 - Phoenix, Arizona
Abraham Gonzalez, Ben Korpan, Jerry Zhao, Ed Younis
Krste Asanović
University of California, Berkeley
Outline

• Motivation
• Open-source Approach to Hardware
  • BOOM: Berkeley Out-of-Order Machine
• Replicating Spectre Attacks on BOOM
• Implementing a Speculation Buffer
  • Comparisons
  • Implementation
• Conclusion
Motivation
Researchers discover seven new Meltdown and Spectre attacks
Experiments showed that processors from AMD, ARM, and Intel are affected.

By Catalin Cimpanu for Zero Day | November 14, 2018 — 14:44 GMT (06:44 PST) | Topic: Security

New Spectre attack enables secrets to be leaked over a network
It’s no longer necessary to run attacker code on the victim system.

Intel LazyFP vulnerability: Exploiting lazy FPU state switching

June 6 2018

Beyond Spectre: Foreshadow, a new Intel security problem
Researchers have broken Intel’s Software Guard Extensions, System Management Mode, and x86-based virtual machines.

Speculative Store Bypass explained: what it is, how it works

May 21, 2018 | Jon Masters, chief ARM architect, Red Hat

Researchers discover SplitSpectre, a new Spectre-like CPU attack
Why are Spectre-style attacks hard?

**Attack Scenarios**
- User process attacks kernel
- User process attacks user space
- Intra-process sandbox escape
- User process attacks enclaves
- Remote timing attacks
- ...

**Covert Channels**
- Changes in cache state
- Power consumption
- Resource contention (FPUs, buffers)
- ...

**Leakage Mechanisms**
- Conditional branch
- Indirect jump
- Return instructions
- Speculative store bypass
- Data speculation
- ...

**Target CPUs**
- ARM
- Intel
- AMD
- RISC-V
- ...

**Spectre Variations**

Taken from “Panel On the Implications of the Meltdown & Spectre Design Flaws”, ISCA 2018
Mitigation Approaches

InvisiSpec/SafeSpec: Blocking unsafe loads from altering the data cache
DAWG: Partition data cache between security domains
StealthMem/CATalyst: Hide visibility of a secure memory region
Context-based fencing: Dynamically stop speculation in secure code
Compiler-inserted fencing: Statically analyze program for Spectre-vulnerable snippets

Lots of interesting approaches, but how to compare them?
Use them together?

V. Kiriansky, et. al. 2018. DAWG: A Defense Against Cache Timing Attacks in Speculative Execution Processors. In MICRO.
T. Kim, et. al. 2012. STEALTHMEM: System-Level Protection Against Cache-Based Side Channel Attacks in the Cloud. In USENIX.
F. Liu, et. al. 2016. CATalyst: Defeating last-level cache side channel attacks in cloud computing. In HPCA.
Microsoft. 2018. Microsoft’s compiler-level Spectre fix shows how hard this problem will be to solve. In Ars Technica.
Open-source Approach to Hardware
Open-source HW + Agile Design Tools + Fast Simulation/Emulation = Security?

Large proliferation of open-source software stacks, cores, and simulation/design infrastructure
The Open-source RISC-V Approach

Security benefits from open-source work

1. Think of new security mitigation/exploit
2. Use open-source RTL to start implementation
3. Quickly iterate through design development with easy, fast, and free tooling
4. Open-source work and have others scrutinize or use your work
Modern Microarchitectures

Commercial Spectre-vulnerable cores are complex, out-of-order, and closed-source.

Need to do speculation-security research on an equivalent open-source academic core.
BOOM: The Berkeley Out-of-Order Machine
BOOM Overview

- Open-source, out-of-order, superscalar RISC-V core
- Runs RISC-V ISA RV64GC
- Linux-capable - boots Fedora + Buildroot
- Silicon-proven - taped out
- ~18K LoC of open-source Chisel RTL
- Highly parameterizable and configurable
- Full integration with Rocket Chip, FireSim, HAMMER

BOOM Microarchitecture

Fetch (4 cycles) → Decode and Rename → Rename and Dispatch → Issue → Register Read → Execute → Writeback

Fetch Buffer → Decode Unit and Rename Logic → Rename Logic and Dispatch → Mem. Issue Queue → Physical Integer RF: 6R3W 100x64b → DS Shim → To Int/FP RF

ALU Issue Queue → ALU → BR/ALU → To Int RF

FP Issue Queue → FP Issue Queue → FP2Int → FMA → To Int/FP RF

ROB → Commit → Physical FP RF: 3R2W 64x65b → Int2FP → To Int/FP RF

Backed Predictor → BTB → PC/Fetch/Width
Replicating Spectre Attacks
Speculation:
• Performance-seeking behavior of modern processors
• Execute instructions before we know they will commit

Side-channel:
• Microarchitectural state which holds interacts with program execution
• Caches, TLBs, power…

Typical Spectre attack:
1. Setup processor to misspeculate in victim code (e.g. train branch predictors)
2. Misspeculation leaks secret into a side channel
3. Attacker recovers secret from side channel

Spectre v1 Example

Steps:
1. Access *if* statement multiple times correctly (predict *if* to fall-through)
2. Give \( x > array1\_sz \)
3. Predict the *if* to be true and bring in *secret* and *array2* value
4. Use the time difference between cached and uncached lines to determine *secret*
5. Repeat!

\[
\text{if} \ (x < array1\_sz): \\
\quad \text{secret} = array1[x] \\
\quad \text{out} = array2[secret * amount]
\]

<table>
<thead>
<tr>
<th>before array2 addresses</th>
<th>after array2 addresses</th>
</tr>
</thead>
<tbody>
<tr>
<td>0*amount</td>
<td>0*amount</td>
</tr>
<tr>
<td>1*amount</td>
<td>1*amount</td>
</tr>
<tr>
<td>2*amount</td>
<td>2*amount</td>
</tr>
<tr>
<td>3*amount</td>
<td>3*amount</td>
</tr>
<tr>
<td>4*amount</td>
<td>4*amount</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Components Needed – With BOOM?

• Branch Prediction
  • Set associative BTB and GShare branch predictors

• Speculative Execution
  • Out-of-order execution and branch kill masks for speculative execution

• Caching
  • L1 data cache and a outer memory set to the latency of an L2 cache

• Cache Manipulation
  • Custom-made L1 data cache `clflush`

BOOM provides all the elements to replicate Spectre!
Spectre v1 Running on FireSim

Easy-to-use, FPGA-accelerated Cycle-accurate Hardware Simulation of RISC-V Systems in the Cloud

Get Started!

What is FireSim?

FireSim is an open-source cycle-accurate, FPGA-accelerated scale-out computer system simulation platform developed in the Berkeley Architecture Research Group in the Electrical Engineering and Computer Sciences Department at the University of California, Berkeley.

FireSim is capable of cycle-exactly simulating from one to thousands of multi-core compute nodes, derived directly from

Implementing a Speculation Buffer
Problem: Load refills are not subject to architectural guarantees
• Misspeculated loads leave side-effects, creating a side-channel

Solution: Treat the data cache as an architectural structure
• Only alter the cache state when instructions commit
• Implement a working prototype in BOOM RTL

```
ld t0, 0(s0)
blt t0, a0, end
sll t1, t0, 2
add t2, a1, t1
ld t3, 0(t2)
end:
```

Misspeculated region

Block speculative cache refills

New cache line

Data Cache
Prior Work

InvisiSpec
- Per load-queue-entry speculation buffer
- Speculation-aware cache-coherence policy

Safespec
- Speculation-depth sized “shadow structures”
- Protect DCache, ICache, TLBs

BOOM Speculation Buffer:
- Hold speculated loads in line-fill-buffers

Life of a Misspeculated Load

Data/tag arrays modified by unsafe instructions/Side-channel

Tag Array
0x1
0x3
0x5
0x7

Data Array
0xabbccdde

Load Queue

check tags

Get(0x200)

To core

Miss, allocate MSHR

MSHR N

MSHR 1

MSHR 0

Replay Queue
idq[5]
idq[4]
0x200

Outer Memory

ld 0x200
check tags

ldq[4]
Refill(0x200)

ldq[5]
Refill(0x200)

ld 0x202
Blocking Misspeculated Loads

Load Queue

Tag Array
- 0x1
- 0x3
- 0x5
- 0x7

Data Array

Data/tag arrays protected from misspeculation

Miss, allocate MSHR

Replay Queue
- ldq[5]
- ldq[4]

Speculation Buffer
- 0xabbccdde

To core

Get(0x200)

Refill(0x200)

Outer Memory
Blocking Misspeculated Loads

Load Queue

Tag Array

Data Array

MSHR 0

Replay Queue

Speculation Buffer

Outer Memory

Miss, allocate MSHR

Get(0x200)

Refill(0x200)

To core

check tags

ld 0x200

ld 0x202

ld 0x200

ld 0x202

ld 0x2

ld 0x2

ld 0x2

ld 0x2

0x1

0x3

0x5

0x2

0xabbccdde

0xabbccdde

0x200

0xabbccdde

0xabbccdde
Blocking Misspeculated Loads

• Load refills wait in the buffer until one of their misses has committed
• Stall writeback until one of the following occurs
  • A load-miss to that line has committed OR
  • A store-miss hits that line (stores are non-speculative)
• If all load misses to that line were misspeculated, discard it
• Bypass loads out of the load-fill-buffer
  • Subsequent loads “see” the data in the DCache
  • Minimizes performance penalty
When to commit load refills to the DCache?

- When the ROB commits the load?
  - Most secure.
  - Huge performance penalty for load misses
- When the load is free from branches?
  - Does not consider exceptions/interrupts
  - Minimal performance penalty
- When the load reaches the **point-of-no-return**
  - New ROB pointer, tracks instructions which are guaranteed to commit
Speculation Buffer Results

1 month implementation time
Microbenchmarks
• Set of assembly routines to test edge cases
Dhrystone results
• Original: 2176 dps
• W. Speculation buffer: 2216 dps
• Impact: ~2% better IPC
Preliminary physical results in TSMC 45nm
• ~3% larger area

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Normal</th>
<th>With Speculation Buffer</th>
<th>% Difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-speculative LD misses to same sets</td>
<td>540 cycles</td>
<td>640 cycles</td>
<td>-19%</td>
</tr>
<tr>
<td>Non-speculative LD misses to different sets</td>
<td>264 cycles</td>
<td>297 cycles</td>
<td>-11%</td>
</tr>
<tr>
<td>MSHR evicted speculative LD misses</td>
<td>48 cycles</td>
<td>67 cycles</td>
<td>-40%</td>
</tr>
<tr>
<td>Dhrystone</td>
<td>2176 dps</td>
<td>2216 dps</td>
<td>+2%</td>
</tr>
</tbody>
</table>
## Comparison

<table>
<thead>
<tr>
<th></th>
<th>InvisiSpec</th>
<th>SafeSpec</th>
<th>BOOM Speculation Buffer</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Implementation Platform</strong></td>
<td>Custom GEM5</td>
<td>Marssx86</td>
<td>BOOM RTL</td>
</tr>
<tr>
<td><strong>Buffer size</strong></td>
<td>Additional cacheline * load-queue-size</td>
<td>Additional cacheline * speculation depth</td>
<td>Repurposed line-fill-buffers</td>
</tr>
<tr>
<td><strong>Commit condition</strong></td>
<td>Wait for branch OR Wait for non-speculative</td>
<td>Wait for branch OR Wait for commit</td>
<td>Wait for point-of-no-return</td>
</tr>
<tr>
<td><strong>Physical design feedback</strong></td>
<td>CACTI estimates</td>
<td>CACTI estimates</td>
<td>Trial TSMC 45nm implementation</td>
</tr>
<tr>
<td><strong>Protected components</strong></td>
<td>L1D, LLC, multicores</td>
<td>L1D, L1I, TLBs</td>
<td>L1D</td>
</tr>
<tr>
<td><strong>Performance impact</strong></td>
<td>-22% performance</td>
<td>+3% performance</td>
<td>+2% performance</td>
</tr>
</tbody>
</table>
Conclusion

Demonstrated application of RISC-V ecosystem towards secure hardware

- Working demonstrations of Spectre attacks on a RISC-V core
- RTL of Spectre mitigation available in an open-source core

Continue improving BOOM security

- Secure other structures: TLBs, ICache, LLC, branch predictors
- Enable secure enclave execution

BOOMv3 Tapeout + More Attacks

- Planning to add Speculation Buffer and CSRs to enable/disable it
- More attacks with different predictors/structures (TAGE, RAS, etc)
Questions?

Thanks CARRV19!

Contact: {abe.gonzalez,bkorpan,jzh,edyounis,krste}@berkeley.edu

Links:
- Core: boom-core.org
- Github: github.com/riscv-boom
- FireSim: fires.im
- HAMMER: github.com/ucb-bar/hammer

Thanks:
- Chris Celio, David Kohlbrenner
- UCB ADEPT Lab