POSITION PAPER:
A CASE FOR EXPOSING EXTRA-ARCHITECTURAL STATE IN THE ISA

Jason Lowe-Power, Venkatesh Akella, Matthew K. Farrens, Samuel T. King, Christopher J. Nitta

@JasonLowePower
Specify speculation in the ISA?

“Invisible” behavior hides security vulnerabilities

Need to include all state
Not only “architectural” state

We want to reason about security of processors

ISA 2.0?

Architectural state
- registers
- memory data
- interrupts

Extra-architectural state
- cached addresses
- branch predictor
- phys. register mapping?
Deep dive into Spectre

Details on how speculation works

- Prevent speculative state changes
- Undo speculative state changes
- Specify speculative state changes

Applying traditional speculation recovery to extra-arch. state

Rethinking the whole system
void victim_function(size_t x) {
  if (x < array1_size) {
    temp &= array2[array1[x] * 512];
  }
}
```c
void victim_function(size_t x) {
    if (x < array1_size) {
        temp &= array2[array1[x] * 512];
    }
}
```

```
000000000040105e <victim_function>:
40105e:    push       %rbp
40105f:    mov        %rsp,%rbp
401062:    mov        %rdi,-0x8(%rbp)
401066:    mov        0x2bf014(%rip),%eax
40106c:    mov        %eax,%eax
40106e:    cmp        -0x8(%rbp),%rax
401072:    jbe        40109f <victim_function+0x41>
401074:    mov        -0x8(%rbp),%rax
401078:    add        $0x6c00a0,%rax
40107e:    movzbl     (%rax),%eax
401081:    movzbl     %al,%eax
401084:    shl        $0x9,%eax
401087:    cltq
401089:    movzbl     0x6c1d80(%rax),%edx
401090:    movzbl     0x2e0ce9(%rip),%eax
401097:    and        %edx,%eax
401099:    mov        %al,0x2e0ce1(%rip)
40109f:    pop        %rbp
4010a0:    retq
```

https://gist.github.com/ErikAugust/724d4a969fb2c6ae1bbd7b2a9e3d4bb6
void victim_function(size_t x) {
    if (x < array1_size) {
        temp &= array2[array1[x] * 512];
    }
}
if (x < array1_size)

load array1_size

load array1[x]

load array2[array1[x] * 512]
if (x < array1_size)
    load array1_size
    load array1[x]
    load array2[array1[x] * 512]
Back to basics

How to keep architectural state consistent

Prevent speculative state changes

Specify speculative state changes

Undo speculative state changes

SPECMATTER.COM

HOW DO THEY WORK?
<table>
<thead>
<tr>
<th><strong>Prevent</strong></th>
<th><strong>Undo</strong></th>
<th><strong>Specify</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Prevent speculative state changes</td>
<td>Undo speculative state changes</td>
<td>Specify speculative state changes</td>
</tr>
</tbody>
</table>

**Ex: Store buffer**

“Undo” a store?

Wait until commit to send to memory

**Ex: Register writes**

Checkpoint the RF

Physical register file & rename tables

**Ex: Relaxed consistency**

Description of allowed ld/st interleavings

Formal specifications
Spectre

Architectural state is unaffected but... the *cache state* changes

Not part of the architectural state
Part of the *extra-architectural state*
Extra-architectural state

Any *state* that is *not specified* in the ISA but *perceivable*

- Cached addresses
- Branch predictor state
- Values in unmapped physical registers???
- Physical to logical register mappings???
  
  . . .

Need to apply same three techniques:  *Prevent  Undo  Specify*
Spectre: **Prevent** EA-state change

Obvious strawman
   - Prevent all speculation
   - 2.4x-24x slowdown

Slightly better
   - Only prevent speculative loads
   - Closes the cache and memory side channel
   - 1.7x-9.8x slowdown
Prevent *cache* changes

Only on cache misses will the state change

Buffer all missed loads until commit

Only up to 1.9x slowdown
Spectre: **Undo** EA-state change

“Undo” the cache change
Checkpoint the cache?

Squash the insert: Insert-side SLB

Limited performance impact

Doesn’t mitigate SpectrePrime
Spectre: **Specify** EA-State change

**MITIGATION G-2**
**Description:** Set an MSR in the processor so that LFENCE is a dispatch serializing instruction and then use LFENCE in code streams to serialize dispatch (LFENCE is faster than RDTSCP which is also dispatch serializing). This mode of LFENCE may be enabled by setting MSR C001_1029[1]=1.

**MITIGATION V2-3**
**Description:** Execute a series of CALL instructions upon entering more privileged code to fill up the return address predictor.
**Effect:** The processor will only predict RET targets to the RIP values in the return address predictor, thus preventing attacker controlled RIP values from being predicted.
**Applicability:** All AMD processors. The size of the return address predictor varies by processor, all current AMD processors have a [return address predictor with 32 entries or less](https://developer.amd.com/wp-content/resources/Managing-Speculation-on-AMD-Processors.pdf). Future processors that have more than 32 RSB entries are planned to be architected to not require software intervention.
Return-address prediction stacks are a common feature of high-performance instruction-fetch units, but require accurate detection of instructions used for procedure calls and returns to be effective. For RISC-V, hints as to the instructions’ usage are encoded implicitly via the register numbers used. A JAL instruction should push the return address onto a return-address stack (RAS) only when \( rd=x_1/x_5 \). JALR instructions should push/pop a RAS as shown in the Table 2.1.

<table>
<thead>
<tr>
<th>( rd )</th>
<th>( rs1 )</th>
<th>( rs1=rd )</th>
<th>RAS action</th>
</tr>
</thead>
<tbody>
<tr>
<td>!( link )</td>
<td>!( link )</td>
<td>-</td>
<td>none</td>
</tr>
<tr>
<td>!( link )</td>
<td>!( link )</td>
<td>-</td>
<td>none</td>
</tr>
<tr>
<td>( link )</td>
<td>!( link )</td>
<td>-</td>
<td>push</td>
</tr>
<tr>
<td>( link )</td>
<td>( link )</td>
<td>0</td>
<td>push and pop</td>
</tr>
<tr>
<td>( link )</td>
<td>( link )</td>
<td>1</td>
<td>push</td>
</tr>
</tbody>
</table>

Table 2.1: Return-address stack prediction hints encoded in register specifiers used in the instruction. In the above, \( link \) is true when the register is either \( x_1 \) or \( x_5 \).
ISA: Contract between hardware and software

*Our* job is to create this contract

Allow designers flexibility. If it’s imperceivably, no need to specify.

Rethink the interface for security
   the μarch, the operating system, the compiler, etc.

Give security researchers formal specifications
Conclusions

“ Invisible” performance optimizations are great

Need to rigorously document potential side-effects
(extra-architectural state changes)

Find the right balance between truly invisible and documented effects
ISA 2.0?

Need a new formalism for speculation
More details on Spectre+gem5
Spectre-v4

Load/store disambiguation

(I think) Current gem5 doesn’t suffer from this
When there’s a possible alias, gem5’s OOO CPU stalls

SLB still works
When speculation recovers, no changes to cache state
Potential formalism for caches

From CCI-Check: Value in cache lifetime (ViCL)
   ViCL create: Time when something is inserted
   ViCL expire: Time when evicted or data changes

Need to add a new notion of “speculation order” that includes non-program order instructions

Loads can be issued in speculation order unless preceded by a speculation fence
Spectre: **Prevent** EA-state change

![Graph showing relative performance of different benchmarks with two conditions: No speculation and No speculative loads. The graph indicates an average slowdown of 4.4x-14x for SPECffloat and 2.8x-7.7x for SPECint.](graph.png)

- **Average 4.4x-14x slowdown for SPECffloat**
- **Average 2.8x-7.7x slowdown for SPECint**
Spectre: Prevent EA-state change

Average 1.3x slowdown for SPECfloat

Average 1.1x slowdown for SPECint
if (x < array1_size)

load array1_size

load array2[array1[x] * 512]

load array1[x]