Speculator
A Tool to Analyze Speculative Execution Attacks and Mitigations

Andrea Mambretti
(mbr@ccs.neu.edu)

Matthias Neugschwandtner, Alessandro Sorniotti and Anil Kurmus - IBM Research
William Robertson and Engin Kirda - Northeastern University
Transient Execution Attacks

- Transient Execution
  - Speculation-based: Spectre v1, Spectre v2
  - Fault-based: Meltdown, Foreshadow

Long term problem... **meant to stay**

Implementation problem “Easily” fixable in new CPUs!
Transient Execution Attacks

- **Speculation-based**
  - For instance: Spectre v1, Spectre v2

- **Fault-based**
  - For instance: Meltdown, Foreshadow

Long term problem... meant to stay

Implementation problem “Easily” fixable in new CPUs!
Spectre v1 - Bounds Check Bypass

if (x < array1_size) {
    y = array2[array1[x]];
}

Cached

Not Cached
Spectre v1 - Bounds Check Bypass

```c
if (x < array1_size) {
    y = array2[array1[x]];
}
```

Example:
- `array1_size = 8`
- `x = 15` (attacker controlled)
Spectre v1 - Bounds Check Bypass

```c
if (x < array1_size) {
    y = array2[array1[x]];
}
```

Example:
- array1_size = 8
- x = 15 (attacker controlled)
Spectre v1 - Bounds Check Bypass

if \( x < array1\_size \) {
    Speculative Execution Trigger

    \[ y = array2[array1[x]]; \]
}

Example:
- array1_size = 8
- \( x = 15 \) (attacker controlled)
Spectre v1 - Bounds Check Bypass

```c
if (x < array1_size) {
    y = array2[array1[x]];  
}
```

**Example:**
- array1_size = 8
- \( x = 15 \) (attacker controlled)

```
array1: 0, 1, 2, 3, 4, 5, 6, 7
array2: 0, 0, 0, 0, 0, 0, 0, 0
```
Spectre v1 - Bounds Check Bypass

```c
if (x < array1_size) {
    y = array2[array1[x]];
}
```

Example:
- array1_size = 8
- x = 15 (attacker controlled)
Spectre v1 - Bounds Check Bypass

```c
if (x < array1_size) {
    y = array2[array1[x]]
}
```

Example:
- `array1_size = 8`
- `x = 15` (attacker controlled)
Spectre v1 - Bounds Check Bypass

```cpp
if (x < array1_size) {
    y = array2[array1[x]];
}
```

Example:
- array1_size = 8
- x = 15 (attacker controlled)
How can we study this type of attacks?

In memory corruption?

GDB

In speculative execution attacks (SEA)?

???

To understand a SEA, we should be able to observe it
Observe Speculation - Practice

Side channels

Problem:
- Costly to setup
- Noisy to read
- Long execution time for each run

Performance counters (NEW)
- Model-specific
- Architecture-specific
- Plenty of counters available
- Implemented in all modern CPU
Speculative Execution Markers

Special *instructions* or sequences *detectable* by performance counters *even* when they *do not retire*
State of the art

Perf_events (Tool or Syscall)

- **Sampling** mode: impossible to get quantitate info counting
- **Counting** mode: high overhead due to in-kernel design

Likwid

Lack of flexibility, only system-wide measures

Others (e.g. Oprofile, Perfmon2, Perfctl, PAPI)

Outdated, inaccurate, not flexible or unmaintained
Speculator
Speculator

Based on CPU program counters

Direct configuration through MSR register

Creation of incremental snippets

Two modes of execution
  Test mode
  Attacker/Victim mode
Speculator

1. Snippet Generation
   - JSON
   - ASM

2. PM Init
3. Test Execution
4. PMC Read

CPU-0

Speculator Monitor

CPU-N

PRE-PROCESSING

RUNTIME

POST-PROCESSING

5. Aggregate Results

6. Results
Speculative Execution Markers (Intel)

**UOPS_EXECUTED.CORE/THREAD**

Count μ-ops executed by a CPU

**UOPS_ISSUED.SINGLE_MUL**

(e.g. `mulps xmm2, xmm1`)

Count single-precision floating-point instructions that operates on xmm register is issued

**UOPS_ISSUED.SLOW_LEA**

(e.g. `lea rax, [array+rax*2]`)

Count lea instruction with 3 operands

Drawback: clflush are counted as slow lea

**LD_BLOCK.STORE_FORWARD**

Count failed store forward

Example:

```
mov DWORD[array], eax
mov DWORD[array+4], edx
movq xmm0, QWORD[array]
```
Findings
Findings - RSB Size
Findings - RSB Size

```c
entry:
  start_counter
  call victim
  ;marker
  lfence

victim:
  call filler
  push myexit
  clflush [rsp]
  lfence
  ret

filler:
  ;##### SNIPPET STARTS HERE #####
  ;growing nested call ret sequence
  ;##### SNIPPET ENDS HERE #####

myexit:
  stop_counter
  msr_close
  exit 0
```
Findings - RSB Size

entry:
  start_counter
  call victim
  ;marker
  lfence

victim:
  call filler
  push myexit
  cflush [rsp]
  lfence
  ret

filler:
  ;##### SNIPPET STARTS HERE #####
  ;growing nested call ret sequence
  ;##### SNIPPET ENDS HERE #####

myexit:
  stop_counter
  msr_close
  exit 0

Return Stack Buffer
Findings - RSB Size

entry:
  start_counter
  call victim
  ;marker
  lfence
  
  victim:
  call filler
  push myexit
  clflush [rsp]
  lfence
  ret
  
  filler:
  ;#### SNIPPET STARTS HERE ####
  ;growing nested call ret sequence
  ;#### SNIPPET ENDS HERE ####
  
  myexit:
  stop_counter
  msr_close
  exit 0

Return Stack Buffer
Findings - RSB Size

```assembly
entry:
    start_counter
    call victim
    ;marker
    lfence

victim:
    call filler
    push myexit
    clflush [rsp]
    lfence
    ret

filler:
    ;; SNIPPET STARTS HERE ######
    ;growing nested call ret sequence
    ;; SNIPPET ENDS HERE ######

myexit:
    stop_counter
    msr_close
    exit 0
```

Return Stack Buffer
Findings - RSB Size

```
entry:
    start_counter
    call victim
    ;marker
    lfence

victim:
    call filler
    push myexit
    clflush [rsp]
    lfence
    ret

filler:
    ;#### SNIPPET STARTS HERE ####
    ;growing nested call ret sequence
    ;#### SNIPPET ENDS HERE ####

myexit:
    stop_counter
    msr_close
    exit 0
```

Return Stack Buffer
Findings - RSB Size

entry:
    start_counter
call victim
;marker
lfence

victim:
call filler
push myexit
clflush [rsp]
lfence
ret

filler:
;####### SNIPPET STARTS HERE #######
;growing nested call ret sequence
;####### SNIPPET ENDS HERE #######

myexit:
    stop_counter
msr_close
exit 0

<table>
<thead>
<tr>
<th>VICTIM</th>
<th>FILLER</th>
</tr>
</thead>
</table>

Return Stack Buffer
Findings - RSB Size

```
entry:
  start_counter
  call victim
  ;marker
  lfence

victim:
  call filler
  push myexit
  clflush [rsp]
  lfence
  ret

filler:
  ;##### SNIPPET STARTS HERE #####
  ;growing nested call ret sequence
  ;##### SNIPPET ENDS HERE #####

myexit:
  stop_counter
  msr_close
  exit 0
```

<table>
<thead>
<tr>
<th>VICTIM</th>
<th>FILLER</th>
<th>PHONY</th>
</tr>
</thead>
</table>

Return Stack Buffer
Findings - RSB Size

entry:
  start_counter
  call victim
  ;marker
  lfence

victim:
  call filler
  push myexit
  clflush [rsp]
  lfence
  ret

filler:
  ;####### SNIPPET STARTS HERE #######
  ;growing nested call ret sequence
  ;####### SNIPPET ENDS HERE #######

myexit:
  stop_counter
  msr_close
  exit 0

<table>
<thead>
<tr>
<th>VICTIM</th>
<th>FILLER</th>
<th>PHONY</th>
<th>PHONY</th>
</tr>
</thead>
</table>

Return Stack Buffer
Findings - RSB Size

entry:
  start_counter
  call victim
  ;marker
  lfence

victim:
  call filler
  push myexit
  clflush [rsp]
  lfence
  ret

filler:
  ;###### SNIPPET STARTS HERE ######
  ;growing nested call ret sequence
  ;###### SNIPPET ENDS HERE ######

myexit:
  stop_counter
  msr_close
  exit 0

Return Stack Buffer
Findings - RSB Size

entry:
  start_counter
call victim
;marker
lfence

victim:
call filler
push myexit
clflush [rsp]
lfence
ret

filler:
;####### SNIPPET STARTS HERE #######
;growing nested call ret sequence
;####### SNIPPET ENDS HERE #######

myexit:
  stop_counter
  msr_close
  exit 0

Return Stack Buffer
Findings - RSB Size

entry:
    start_counter
call victim
;marker
lfence

victim:
call filler
push myexit
clflush [rsp]
lfence
ret

caller:

;;;; SNIPPET STARTS HERE │★★★★★
; growing nested call ret sequence
;;;; SNIPPET ENDS HERE  │★★★★★

myexit:
    stop_counter
    msr_close
    exit 0

<table>
<thead>
<tr>
<th>VICTIM</th>
<th>FILLER</th>
<th>PHONY</th>
<th>PHONY</th>
<th>PHONY</th>
<th>PHONY</th>
<th>PHONY</th>
</tr>
</thead>
</table>

Return Stack Buffer
Findings - RSB Size

entry:
  start_counter
call victim
  ;marker
  lfence

victim:
call filler
push myexit
clflush [rsp]
lfence
ret

filler:
  ;###### SNIPPET STARTS HERE ######
  ;growing nested call ret sequence
  ;###### SNIPPET ENDS HERE ######

myexit:
  stop_counter
  msr_close
  exit 0

<table>
<thead>
<tr>
<th>VICTIM</th>
<th>FILLER</th>
<th>PHONY</th>
<th>PHONY</th>
<th>PHONY</th>
<th>PHONY</th>
<th>PHONY</th>
<th>PHONY</th>
<th>PHONY</th>
</tr>
</thead>
</table>

Return Stack Buffer
Findings - RSB Size

```
entry:
  start_counter
  call victim
  ;marker
  lfence

victim:
  call filler
  push myexit
  clflush [rsp]
  lfence
  ret

filler:
  ;##### SNIPPET STARTS HERE ######
  ;growing nested call ret sequence
  ;##### SNIPPET ENDS HERE ######

myexit:
  stop_counter
  msr_close
  exit 0
```

<table>
<thead>
<tr>
<th>VICTIM</th>
<th>FILLER</th>
<th>PHONY</th>
<th>PHONY</th>
<th>PHONY</th>
<th>PHONY</th>
<th>PHONY</th>
<th>PHONY</th>
<th>PHONY</th>
</tr>
</thead>
</table>

Return Stack Buffer
Findings - RSB Size

entry:
start_counter
call victim
;marker
lfence

victim:
call filler
push myexit
clflush [rsp]
lfence
ret

filler:
;;;; SNIPPET STARTS HERE #####
;growing nested call ret sequence
;;;; SNIPPET ENDS HERE #####

myexit:
stop_counter
msr_close
exit 0

Return Stack Buffer
Findings - RSB Size

entry:
   start_counter
   call victim
   ;marker
   l_fence

victim:
   call filler
   push myexit
   clflush [rsp]
   l_fence
   ret

filler:
   ;##### SNIPPET STARTS HERE #####
   ;growing nested call ret sequence
   ;##### SNIPPET ENDS HERE #####

myexit:
   stop_counter
   msr_close
   exit 0

Return Stack Buffer
Findings - RSB Size

```
entry:
   start_counter
call victim
;marker
lfence

victim:
call filler
push myexit
clflush [rsp]
lfence

ret

filler:
;#### SNIPPET STARTS HERE ####
;growing nested call ret sequence
;#### SNIPPET ENDS HERE ####

myexit:
   stop_counter
msr_close
exit 0
```

Return Stack Buffer
Findings - RSB Size

entry:
    start_counter
    call victim
    ; marker
    lfence

victim:
    call filler
    push myexit
    clflush [rsp]
    lfence

ret

filler:
    ;######## SNIPPET STARTS HERE ########
    ; growing nested call ret sequence
    ;######## SNIPPET ENDS HERE ########

myexit:
    stop_counter
    msr_close
    exit 0

Return Stack Buffer
Findings - RSB Size

```c
entry:
    start_counter
    call Victim

;marker
    lfence

victim:
    call filler
    push myexit
    clflush [rsp]
    lfence

ret

filler:
    ;#### SNIPPET STARTS HERE ####
    ;growing nested call ret sequence
    ;#### SNIPPET ENDS HERE ####

myexit:
    stop_counter
    msr_close
    exit 0
```

Return Stack Buffer
Findings - RSB Size

entry:
    start_counter
    call victim

;marker
    lfence

victim:
    call filler
    push myexit
    clflush [rsp]
    lfence

ret

filler:
    ;#### SNIPPET STARTS HERE #####
    ;growing nested call ret sequence
    ;#### SNIPPET ENDS HERE #####

myexit:
    stop_counter
    msr_close
    exit 0

Marker Hit

Return Stack Buffer
Findings - RSB Size

```plaintext
entry:
    start_counter
    call victim
    ;marker
    lfence

victim:
    call filler
    push myexit
    clflush [rsp]
    lfence

ret

filler:
    ;#### SNIPPET STARTS HERE #####
    ;growing nested call ret sequence
    ;#### SNIPPET ENDS HERE #####

myexit:
    stop_counter
    msr_close
    exit 0
```

Return Stack Buffer
Findings - RSB Size

entry:
  start_counter
call victim
;marker
lfence

victim:
call filler
push myexit
clflush [rsp]
lfence
ret

caller:
  ; SNIPPET STARTS HERE ######
  ; growing nested call ret sequence
  ; SNIPPET ENDS HERE ######

myexit:
  stop_counter
  msr_close
  exit 0

Return Stack Buffer
Findings - RSB Size

tenry:
  start_counter
  call victim
  ;marker
  lfence

victim:
  call filler
  push myexit
  clflush [rsp]
  lfence
  ret

filler:
  ;####### SNIPPET STARTS HERE #######
  ;growing nested call ret sequence
  ;####### SNIPPET ENDS HERE #######

myexit:
  stop_counter
  msr_close
  exit 0
Findings - RSB Size

```
entry:
  start_counter
  call victim
  ;marker
  lfence

victim:
  call filler
  push myexit
  clflush [rsp]
  lfence
  ret

filler:
  ;##### SNIPPET STARTS HERE ######
  ;growing nested call ret sequence
  ;##### SNIPPET ENDS HERE ######

myexit:
  stop_counter
  msr_close
  exit 0
```

Return Stack Buffer
Findings - RSB Size

entry:
  start_counter
  call victim
  ;marker
  lfence

victim:
  call filler
  push myexit
  clflush [rsp]
  lfence
  ret

filler:
  ;##### SNIPPET STARTS HERE #####
  ;growing nested call ret sequence
  ;##### SNIPPET ENDS HERE #####

myexit:
  stop_counter
  msr_close
  exit 0

<table>
<thead>
<tr>
<th>VICTIM</th>
<th>FILLER</th>
<th>PHONY</th>
<th>PHONY</th>
<th>PHONY</th>
<th>PHONY</th>
<th>PHONY</th>
<th>PHONY</th>
<th>PHONY</th>
<th>PHONY</th>
</tr>
</thead>
</table>

Return Stack Buffer
Findings - RSB Size

entry:
    start_counter
    call victim
    ;marker
    lfence

victim:
    call filler
    push myexit
    clflush [rsp]
    lfence
    ret

filler:
    ;##### SNIPPET STARTS HERE #####
    ;growing nested call ret sequence
    ;##### SNIPPET ENDS HERE #####

myexit:
    stop_counter
    mrs_close
    exit 0

PHONY  FILLER  PHONY  PHONY  PHONY  PHONY  PHONY  PHONY  PHONY  PHONY

Return Stack Buffer
### Findings - RSB Size

<table>
<thead>
<tr>
<th>entry:</th>
</tr>
</thead>
<tbody>
<tr>
<td>start_counter</td>
</tr>
<tr>
<td>call victim</td>
</tr>
<tr>
<td>;marker</td>
</tr>
<tr>
<td>lfence</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>victim:</th>
</tr>
</thead>
<tbody>
<tr>
<td>call filler</td>
</tr>
<tr>
<td>push myexit</td>
</tr>
<tr>
<td>clflush [rsp]</td>
</tr>
<tr>
<td>lfence</td>
</tr>
<tr>
<td>ret</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>filler:</th>
</tr>
</thead>
<tbody>
<tr>
<td>;;;;; SNIPPET STARTS HERE ;;;;;</td>
</tr>
<tr>
<td>;growing nested call ret sequence</td>
</tr>
<tr>
<td>;;;;; SNIPPET ENDS HERE ;;;;;</td>
</tr>
<tr>
<td>myexit:</td>
</tr>
<tr>
<td>stop_counter</td>
</tr>
<tr>
<td>msr_close</td>
</tr>
<tr>
<td>exit 0</td>
</tr>
</tbody>
</table>

---

Return Stack Buffer
Findings - RSB Size

```
entry:
  start_counter
  call victim
  ;marker
  lfence

victim:
  call filler
  push myexit
  clflush [rsp]
  lfence
  ret

filler:
  ;####### SNIPPET STARTS HERE #######
  ;growing nested call ret sequence
  ;####### SNIPPET ENDS HERE #######

myexit:
  stop_counter
  msr_close
  exit 0
```

Return Stack Buffer
Findings - RSB Size

entry:
  start_counter
  call victim
  ;marker
  lfence

victim:
  call filler
  push myexit
  clflush [rsp]
  lfence

ret

filler:
  ;#### SNIPPET STARTS HERE ####
  ;growing nested call ret sequence
  ;#### SNIPPET ENDS HERE ####

myexit:
  stop_counter
  msr_close
  exit 0

Return Stack Buffer
Findings - RSB Size

entry:
  start_counter
  call victim
  ;marker
  lfence

victim:
  call filler
  push myexit
  clflush [rsp]
  lfence

ret

filler:
  ;###### SNIPPET STARTS HERE #######
  ;growing nested call ret sequence
  ;###### SNIPPET ENDS HERE #######

myexit:
  stop_counter
  msr_close
  exit 0
Findings - RSB Size

```
entry:
    start_counter
    call victim
    ;marker
    lfence

victim:
    call filler
    push myexit
    clflush [rsp]
    lfence

ret

filler:
    ;####### SNIPPET STARTS HERE #######
    ;growing nested call ret sequence
    ;####### SNIPPET ENDS HERE #######

myexit:
    stop_counter
    msr_close
    exit 0
```

NO Marker Hit

Return Stack Buffer
Findings - RSB Size

entry:
  start_counter
  call victim
  ;marker
  lfence

victim:
  call filler
  push myexit
  clflush [rsp]
  lfence
  ret

filler:
  ;###### SNIPPET STARTS HERE ######
  ;growing nested call ret sequence
  ;###### SNIPPET ENDS HERE ######

myexit:
  stop_counter
  msr_close
  exit 0

Return Stack Buffer
Findings - RSB Size

entry:
  start_counter
  call victim
  ;marker
  lfence

victim:
  call filler
  push myexit
  clflush [rsp]
  lfence
  ret

filler:
  ;##### SNIPPET STARTS HERE #####
  ;growing nested call ret sequence
  ;##### SNIPPET ENDS HERE #####

myexit:
  stop_counter
  msr_close
  exit 0

Return Stack Buffer

Therefore:

\[ RSB\ size = VICTIM + FILLER + X \text{ nested call/ret} \]

When X+1 nested calls cause no marker hit
Findings - RSB Size Results

Intel

RSB_{size} = 1 + 1 + 14 Nested call/ret = 16

AMD

RSB_{size} = 1 + 1 + 29 Nested call/ret = 31
Conclusions

**New methodology** to observe speculative execution **based on markers**

New low-overhead tool, **Speculator**, tailored to study new attacks and mitigations

Several **new insights** on speculative execution behavior on different CPU (e.g. Broadwell, Skylake, AMD Ryzen)

Speculator and the markers **easier** the **study** of old and new **attacks techniques**
Speculator is open source:

https://github.com/ibm-research/speculator

Questions?
Findings - Nesting Speculative Execution
Findings - Nesting Speculative Execution

if (slowest condition) {
    // marker
    if (slow condition) {
        // marker
        if (fast condition) {
            // marker
        }
        else { //marker
    }
    else { //marker
}
else { //marker
}
Findings - Nesting Speculative Execution

```java
if (slowest condition) {
    // marker
    if (slow condition) {
        // marker
        if (fast condition) {
            // marker
        }
        else { //marker
    }
    else { //marker
}
else { //marker
}
```
Findings - Nesting Speculative Execution

```java
if (slowest condition) {
    // marker
    if (slow condition) {
        // marker
        if (fast condition) {
            //marker
        }
        else { //marker}
    }
    else { //marker}
}
else { //marker}
else { //marker}
```
Findings - Nesting Speculative Execution

```java
if (slowest condition) {
    // marker
    if (slow condition) {
        // marker
        if (fast condition) {
            // marker
        }
        else { // marker}
    }
    else { // marker}
}
else { // marker}
else { // marker}
```
Findings - Nesting Speculative Execution

```java
if (slowest condition) {
    // marker
    if (slow condition) {
        // marker
        if (fast condition) {
            //marker
        } else { //marker}
    } else { //marker}
} else { //marker}
else { //marker}
```
Findings - Nesting Speculative Execution

```java
if (slowest condition) {
    // marker
    if (slow condition) {
        // marker
        if (fast condition) {
            //marker
        }
    } else {
        //marker
    }
} else {
    //marker
}
else {
    //marker
}
```
Findings - Nesting Speculative Execution

```java
if (slowest condition) {
    // marker
    if (slow condition) {
        // marker
        if (fast condition) {
            //marker
        }
        else { //marker}
    }
    else { //marker}
}
else { //marker}
else { //marker}
```
Findings - Nesting Speculative Execution

```java
if (slowest condition) {
    // marker
    if (slow condition) {
        // marker
        if (fast condition) {
            //marker
        }
        else { //marker
    }
    } else { //marker
}
else { //marker
}
else { //marker
}
Findings - Nesting Speculative Execution

```java
if (slowest condition) {
    // marker
    if (slow condition) {
        // marker
        if (fast condition) {
            //marker
        }
        else { //marker
        }
    }
    else { //marker
    }
}
else { //marker
}
else { //marker
}
```
Findings - Nesting Speculative Execution

```java
if (slowest condition) {
    // marker
    if (slow condition) {
        // marker
        if (fast condition) {
            //marker
        }
    } else { //marker
    }
else { //marker
}
else { //marker
}
else { //marker
}
```
Findings - Nesting Speculative Execution

if (slowest condition) {
    // marker
    if (slow condition) {
        // marker
        if (fast condition) {
            // marker
        }
    } else { //marker}
} else { //marker}
else { //marker}
Findings - Nesting Speculative Execution

```c
if (slowest condition) {
    // marker
    if (slow condition) {
        // marker
        if (fast condition) {
            // marker
        }
        else { //marker}
    }
    else { //marker}
} else { //marker}
```

SPECULATIVE EXECUTION
Findings - Nesting Speculative Execution

```c
if (slowest condition) {
    // marker
    if (slow condition) {
        // marker
        if (fast condition) {
            // marker
        }
        else { //marker}
    }
    else { //marker}
} else { //marker}
else { //marker}
```
Findings - Nesting Speculative Execution

```java
if (slowest condition) {
  // marker
  if (slow condition) {
    // marker
    if (fast condition) {
      // marker
    }
    else { //marker}
  }
  else { //marker}
}
else { //marker}
```
Findings - Nesting Speculative Execution

```java
if (slowest condition) {
    // marker
    if (slow condition) {
        // marker
        if (fast condition) {
            // marker
        }
        else { //marker}
    }
    else { //marker}
}
else { //marker}
else { //marker}
```
Findings - Nesting Speculative Execution

```c
if (slowest condition) {
    // marker
    if (slow condition) {
        // marker
        if (fast condition) {
            // marker
        }
    } else { //marker}
} else { //marker}
else { //marker}
```
Findings - Nesting Speculative Execution

```java
if (slowest condition) {
    // marker
    if (slow condition) {
        // marker
        if (fast condition) {
            //marker
        }
    }
    else { //marker}
}
else { //marker
}
else { //marker
}
```
Findings - Nesting Speculative Execution

```java
if (slowest condition) {
    // marker
    if (slow condition) {
        // marker
        if (fast condition) {
            //marker
        }
    }
    else { //marker}
}
else { //marker}
}
else { //marker}
else { //marker}
```
Findings - Nesting Speculative Execution

```java
if (slowest condition) {
    // marker
    if (slow condition) {
        // marker
        if (fast condition) {
            //marker
        }
    } else { //marker}
} else { //marker}
else { //marker}
```
Findings - Nesting Speculative Execution

```java
if (slowest condition) {
    // marker
    if (slow condition) {
        // marker
        if (fast condition) {
            // marker
        }
    } else { //marker
    }
} else { //marker
}
else { //marker
}
```
Findings - Nesting Speculative Execution

```java
if (slowest condition) {
    // marker
    if (slow condition) {
        // marker
        if (fast condition) {
            // marker
        }
    } else { //marker
    }
} else { //marker
}
else { //marker
}```
Findings - Nesting Speculative Execution

```c
if (slowest condition) {
    // marker
    if (slow condition) {
        // marker
        if (fast condition) {
            // marker
        }
        else { //marker
    }
    else { //marker
}
else { //marker
}
else { //marker
```
Findings - Nesting Speculative Execution

```c
if (slowest condition) {
    // marker
    if (slow condition) {
        // marker
        if (fast condition) {
            // marker
        }
        else { //marker
        }
    } else { //marker
    } else { //marker
}
else { //marker
}
else { //marker
}
```
Findings - Nesting Speculative Execution

```java
if (slowest condition) {
    // marker
    if (slow condition) {
        // marker
        if (fast condition) {
            // marker
        } else { //marker
        } else { //marker
    } else { //marker
} else { //marker
}
```
Findings - Nesting Speculative Execution

```java
if (slowest condition) {
    // marker
    if (slow condition) {
        // marker
        if (fast condition) {
            // marker
        }
        else { //marker}
    }
    else { //marker}
}
else { //marker}
```
Findings - Nesting Speculative Execution

```java
if (slowest condition) {
    // marker
    if (slow condition) {
        // marker
        if (fast condition) {
            // /marker
        }
    } else { // /marker
    } else { // /marker
} else { // /marker
```

SPECULATIVE EXECUTION
Findings - Nesting Speculative Execution

```java
if (slowest condition) {
    // marker
    if (slow condition) {
        // marker
        if (fast condition) {
            // marker
        }
    } else { //marker
    } else { //marker
} else { //marker
}
```
Findings - Speculation Window Size
Findings - Speculation Window Size

Conditional Branches

<table>
<thead>
<tr>
<th>Conditional branch</th>
<th>Broadwell</th>
<th>Skylake</th>
<th>Zen</th>
</tr>
</thead>
<tbody>
<tr>
<td>Register access</td>
<td>14</td>
<td>16</td>
<td>7</td>
</tr>
<tr>
<td>Access to cached memory</td>
<td>19</td>
<td>17</td>
<td>9</td>
</tr>
<tr>
<td>Access to uncached memory</td>
<td>144</td>
<td>280</td>
<td>321</td>
</tr>
<tr>
<td>Mul with register</td>
<td>19</td>
<td>19</td>
<td>2</td>
</tr>
<tr>
<td>Mul with cached memory</td>
<td>33</td>
<td>33</td>
<td>8</td>
</tr>
<tr>
<td>Mul with uncached memory</td>
<td>154</td>
<td>290</td>
<td>362</td>
</tr>
<tr>
<td>Div with register</td>
<td>35</td>
<td>41</td>
<td>17</td>
</tr>
<tr>
<td>Div with cached memory</td>
<td>34</td>
<td>39</td>
<td>30</td>
</tr>
<tr>
<td>Div with uncached memory</td>
<td>164</td>
<td>306</td>
<td>353</td>
</tr>
</tbody>
</table>
## Findings - Speculation Window Size

### Conditional Branches

<table>
<thead>
<tr>
<th>Conditional branch</th>
<th>Broadwell</th>
<th>Skylake</th>
<th>Zen</th>
</tr>
</thead>
<tbody>
<tr>
<td>Register access</td>
<td>14</td>
<td>16</td>
<td>7</td>
</tr>
<tr>
<td>Access to cached memory</td>
<td>19</td>
<td>17</td>
<td>9</td>
</tr>
<tr>
<td>Access to uncached memory</td>
<td>144</td>
<td>280</td>
<td>321</td>
</tr>
<tr>
<td>Mul with register</td>
<td>19</td>
<td>19</td>
<td>2</td>
</tr>
<tr>
<td>Mul with cached memory</td>
<td>33</td>
<td>33</td>
<td>8</td>
</tr>
<tr>
<td>Mul with uncached memory</td>
<td>154</td>
<td>290</td>
<td>362</td>
</tr>
<tr>
<td>Div with register</td>
<td>35</td>
<td>41</td>
<td>17</td>
</tr>
<tr>
<td>Div with cached memory</td>
<td>34</td>
<td>39</td>
<td>30</td>
</tr>
<tr>
<td>Div with uncached memory</td>
<td>164</td>
<td>306</td>
<td>353</td>
</tr>
</tbody>
</table>
## Findings - Speculation Window Size

### Conditional Branches

<table>
<thead>
<tr>
<th>Conditional branch</th>
<th>Broadwell</th>
<th>Skylake</th>
<th>Zen</th>
</tr>
</thead>
<tbody>
<tr>
<td>Register access</td>
<td>14</td>
<td>16</td>
<td>7</td>
</tr>
<tr>
<td>Access to cached memory</td>
<td>19</td>
<td>17</td>
<td>9</td>
</tr>
<tr>
<td>Access to uncached memory</td>
<td>144</td>
<td>280</td>
<td>321</td>
</tr>
<tr>
<td>Mul with register</td>
<td>19</td>
<td>19</td>
<td>2</td>
</tr>
<tr>
<td>Mul with cached memory</td>
<td>33</td>
<td>33</td>
<td>8</td>
</tr>
<tr>
<td>Mul with uncached memory</td>
<td>154</td>
<td>290</td>
<td>362</td>
</tr>
<tr>
<td>Div with register</td>
<td>35</td>
<td>41</td>
<td>17</td>
</tr>
<tr>
<td>Div with cached memory</td>
<td>34</td>
<td>39</td>
<td>30</td>
</tr>
<tr>
<td>Div with uncached memory</td>
<td>164</td>
<td>306</td>
<td>353</td>
</tr>
</tbody>
</table>
### Findings - Speculation Window Size

#### Indirect Control Flow Transfer

<table>
<thead>
<tr>
<th>Indirect branch target location</th>
<th>Broadwell</th>
<th>Skylake</th>
<th>Zen</th>
</tr>
</thead>
<tbody>
<tr>
<td>Register</td>
<td>28</td>
<td>22</td>
<td>24</td>
</tr>
<tr>
<td>Cached memory</td>
<td>41</td>
<td>34</td>
<td>35</td>
</tr>
<tr>
<td>Uncached memory</td>
<td>154</td>
<td>303</td>
<td>301</td>
</tr>
</tbody>
</table>
Findings - Speculation Window Size

Store to Load Forward

Avg: 15 μ-ops
Max: 23 μ-ops
55 CPU cycles
Findings - MPX

Setup:

10 iterations with correct bound check

Then fail on $bn	ext{\textsubscript{dcu}}$ instruction

Using NOP sled we can speculative execute 122 instructions after bound check violation
Findings - MPX

Result: 22 FNOP instructions speculation window
Findings - Executable Page Permission

Is the NX bit lazily evaluated as access permission in Meltdown?

We load a memory area we control inside TLB and cache

We speculatively execute a control transfer to such area

Result: The execute page table permission bit is honored
Findings - Issued vs Executed

Are issued $\mu$-ops measured by the markers really executed?
Findings - Speculation across system calls

Traced very small syscall (sys_getppid, ~47 instructions)

User-mode μ-ops count does not variate with more instructions after call

Kernel-mode μ-ops count does not variate between speculated and non speculated execution

Conclusion: system calls stop speculation
Findings - Flushing the cache
Findings - Flushing the cache

```assembly
setup
.loop:
  clflush[counter]
  clflush[var]
  lfence

  mov eax, DWORD[var] ;cached version
  lfence ;only

start_counter

  cmp 12, DWORD[counter]
  je .else

  clflush[var]
  lfence

.else:
  mov eax, DWORD[var] ;final load
  lfence

stop_counter

  inc DWORD[counter]
  cmp DWORD[counter], 13
  jl loop
```
Findings - Flushing the cache

Conclusions:
CLFLUSH does not affect the cache until it retires

CLFLUSH must be paired with speculation blocker (e.g. LFENCE) to be sure it has the intended effect
Performance counters (INTEL)

3 fixed counters

4 programmable counters with SMT

8 programmable counters without SMT

Plenty of different counters available for front-end or back-end of the CPU
Performance counters (INTEL)

3 fixed counters

4 programmable counters with SMT

8 programmable counters without SMT

Plenty of different counters available for front-end or back-end of the CPU