Exploiting the Microarchitecture: Transient Execution Attacks

Michael Schwarz (@misc0110)
April 11, 2019
Graz University of Technology
Michael Schwarz
PhD candidate @ Graz University of Technology

@misc0110

michael.schwarz@iaik.tugraz.at
INTEL REVEALS DESIGN FLAW THAT COULD ALLOW HACKERS TO ACCESS DATA
DEVELOPING STORY

COMPUTER CHIP FLAWS IMPACT BILLIONS OF DEVICES
GLOBAL

COMPUTER CHIP SCARE
The bugs are known as 'Spectre' and 'Meltdown'
• Bug-free software does not mean safe execution
- Bug-free software does not mean safe execution
- Information leaks due to underlying hardware
• Bug-free software does not mean safe execution
• Information leaks due to underlying hardware
• Exploit leakage through side-effects
• Bug-free software does not mean safe execution
• Information leaks due to underlying hardware
• Exploit leakage through side-effects
• Instruction Set Architecture (ISA) is an abstract model of a computer (x86, ARMv8, SPARC, ...)
• Instruction Set Architecture (ISA) is an abstract model of a computer (x86, ARMv8, SPARC, …)
• Interface between hardware and software
• Instruction Set Architecture (ISA) is an abstract model of a computer (x86, ARMv8, SPARC, …)
• Interface between hardware and software
• Microarchitecture is an ISA implementation
• Instruction Set Architecture (ISA) is an abstract model of a computer (x86, ARMv8, SPARC, …)
• Interface between hardware and software
• Microarchitecture is an ISA implementation
• Modern CPUs contain multiple microarchitectural elements
Microarchitectural Components

- Modern CPUs contain multiple **microarchitectural elements**

Caches and buffers

Predictors
Microarchitectural Components

- Modern CPUs contain multiple microarchitectural elements

Caches and buffers
Predictors

- Transparent for the programmer
Modern CPUs contain multiple microarchitectural elements

- Caches and buffers
- Predictors
- Transparent for the programmer
- Timing optimizations → side-channel leakage
```c
printf("%d", i);
printf("%d", i);
```
printf("%d", i);
printf("%d", i);
printf("%d", i);
printf("%d", i);

Cache miss

Request
printf("%d", i);
printf("%d", i);

Cache miss

Request
Response
printf("%d", i);
printf("%d", i);
```c
printf("%d", i);  // Cache miss
printf("%d", i);  // Cache hit
```
CPU Cache

DRAM access, slow

printf("%d", i);

printf("%d", i);

Cache miss

Request

Response

Cache hit
CPU Cache

DRAM access, slow

printf("%d", i);

Cache miss

printf("%d", i);

Cache hit

No DRAM access, much faster

Request

Response

5

Michael Schwarz (@misc0110) — Graz University of Technology
Caching speeds up Memory Accesses

![Graph showing cache hits vs. access time in CPU cycles]

- **Access time [CPU cycles]**
  - 60 to 340
- **Number of accesses**
  - 0 to 10^5
- **Cache Hits**

Michael Schwarz (@misc0110) — Graz University of Technology
Caching speeds up Memory Accesses

Access time [CPU cycles]

Number of accesses

Cache Hits

Cache Misses

Michael Schwarz (@misc0110) — Graz University of Technology
Flush+Reload

Attacker
flush
access

Shared Memory

Victim
access

Michael Schwarz (@misc0110) — Graz University of Technology
Flush+Reload

Attacker
flush
access

Shared Memory

Victim
access

Michael Schwarz (@misc0110) — Graz University of Technology
Flush+Reload

Attacker
 flush
 access

Shared Memory

Victim
 access
Flush+Reload

Attacker

flush
access

Shared Memory

Victim

access
Flush + Reload

Attacker
flush
access

Shared Memory

Victim
access
Flush+Reload

Attacker
- flush
- access

Shared Memory

Victim
- access

Michael Schwarz (@misc0110) — Graz University of Technology
Flush+Reload

Attacker

flush

access

Shared Memory

Victim

access

7

Michael Schwarz (@misc0110) — Graz University of Technology
Flush+Reload

Attacker

flush

access

Shared Memory

Victim

access

Victim accessed (fast) vs Victim did not access (slow)

Michael Schwarz (@misc0110) — Graz University of Technology
char array[256 * 4096]; // 256 pages of memory
Toy example

```c
char array[256 * 4096]; // 256 pages of memory

*(volatile char*) 0; // raise_exception();
array[84 * 4096] = 0;
```
• Flush+Reload over all pages of the array
• Flush+Reload over all pages of the array

• “Unreachable” code line was actually executed
- Flush+Reload over all pages of the array

- “Unreachable” code line was actually executed
- Exception was only thrown afterwards
• Flush+Reload over all pages of the array

![Access time graph](image)

- “Unreachable” code line was actually executed
- Exception was only thrown afterwards
- Out-of-order instructions leave microarchitectural traces
Building the Code

- Flush+Reload over all pages of the array

- “Unreachable” code line was actually executed
- Exception was only thrown afterwards
- Out-of-order instructions leave microarchitectural traces
- Give such instructions a name: transient instructions
• Add another **layer of indirection** to test

```c
char array[256 * 4096]; // 256 pages of memory
```
• Add another layer of indirection to test

```c
char array[256 * 4096]; // 256 pages of memory

// read kernel address (raises exception)
char data = *(char*) 0xffffffff81a000e0;
array[data * 4096] = 0;
```
• Add another layer of indirection to test

```c
char array[256 * 4096]; // 256 pages of memory

// read kernel address (raises exception)
char data = *(char*) 0xffffffff81a000e0;
array[data * 4096] = 0;
```

• Then check whether any part of array is cached
• Flush+Reload over all pages of the array

• Index of cache hit reveals data
- Flush+Reload over all pages of the array

- Index of cache hit reveals data
- Permission check is in some cases too late
• CPU uses data in out-of-order execution before permission check
Meltdown

- CPU uses data in out-of-order execution before permission check
- Meltdown can read any kernel address
Meltdown

- CPU uses data in out-of-order execution before permission check
- Meltdown can read any kernel address
- Physical memory is usually mapped in kernel
- CPU uses data in out-of-order execution before permission check
- Meltdown can read any kernel address
- Physical memory is usually mapped in kernel
  → Read arbitrary memory
• Assumed Meltdown can one only read data from the L1
• Assumed Meltdown can one only read data from the L1
• Leakage from L3 or memory is possible, just slower
• Assumed Meltdown can one only read data from the L1
• Leakage from L3 or memory is possible, just slower
• Even leakage of UC (uncachable) memory regions...
Uncached and uncachable memory

• Assumed Meltdown can one only read data from the L1
• Leakage from L3 or memory is possible, just slower
• Even leakage of UC (uncachable) memory regions...
  • ...if other hyperthread (legally) accesses the data
Uncached and uncachable memory

- Assumed Meltdown can one only read data from the L1
- Leakage from L3 or memory is possible, just slower
- Even leakage of UC (uncachable) memory regions...
  - ...if other hyperthread (legally) accesses the data
    → ...leaks from line fill buffer
• Kernel addresses in user space are a problem
• Kernel addresses in user space are a problem
• Why don’t we take the kernel addresses…
...and remove them

- ...and remove them if not needed?
...and remove them if not needed?
- User accessible check in hardware is not reliable
Meltdown Mitigation: KAISER

- Userspace
- Kernelspace
- Applications
- Operating System
- Memory
Kernel View

User View

context switch

Michael Schwarz (@misc0110) — Graz University of Technology
Mitigations

- **Linux**: Kernel Page-table Isolation (KPTI)

Apple: Released updates

Windows: Kernel Virtual Address (KVA) Shadow
Mitigations

- **Linux**: Kernel Page-table Isolation (KPTI)
- **Apple**: Released updates
Mitigations

- **Linux**: Kernel Page-table Isolation (KPTI)
- **Apple**: Released updates
- **Windows**: Kernel Virtual Address (KVA) Shadow
Problem Solved?

• Meltdown fully mitigated in software
Problem Solved?

- Meltdown **fully mitigated** in software
- Problem **seemed** to be solved
Problem Solved?

• Meltdown fully mitigated in software
• Problem seemed to be solved
• No attack surface left
Meltdown fully mitigated in software
Problem seemed to be solved
No attack surface left
That is what everyone thought
There are no bugs, just happy little accidents
• Meltdown is a whole category of vulnerabilities
Meltdown is a whole category of vulnerabilities
Not only the user-accessible check
• Meltdown is a whole category of vulnerabilities
• Not only the user-accessible check
• Looking closer at the check...
• CPU uses **virtual address spaces** to isolate processes
• CPU uses **virtual address spaces** to isolate processes
• Physical memory is organized in **page frames**
• CPU uses *virtual address spaces* to isolate processes
• Physical memory is organized in *page frames*
• Virtual memory pages are *mapped* to page frames using *page tables*
Address Translation on x86-64

- CR3
- PML4
  - PML4E 0
  - PML4E 1
  - ... (PML4E 511)
- PDPT
  - PDPTI 0
  - PDPTI 1
  - ... (PDPTI 511)
- Page Directory
  - PDE 0
  - PDE 1
  - ... (PDE 511)
  - PDE #PDI
  - PDE 511
- Page Table
  - PTE 0
  - PTE 1
  - ... (PTE 511)
  - PTE #PTI
  - PTE 511
- 4 KiB Page
  - Byte 0
  - Byte 1
  - ... (Byte 4095)

48-bit virtual address

Michael Schwarz (@misc0110) — Graz University of Technology
### Page Table Entry

<table>
<thead>
<tr>
<th>P</th>
<th>RW</th>
<th>US</th>
<th>WT</th>
<th>UC</th>
<th>R</th>
<th>D</th>
<th>S</th>
<th>G</th>
<th>Ignored</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Physical Page Number**

<table>
<thead>
<tr>
<th>Ignored</th>
<th>X</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- User/Supervisor bit defines in which **privilege level** the page can be accessed
<table>
<thead>
<tr>
<th>P</th>
<th>RW</th>
<th>US</th>
<th>WT</th>
<th>UC</th>
<th>R</th>
<th>D</th>
<th>S</th>
<th>G</th>
<th>Ignored</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Physical Page Number

<table>
<thead>
<tr>
<th>Ignored</th>
<th>X</th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
<td>RW</td>
</tr>
<tr>
<td>---</td>
<td>----</td>
</tr>
</tbody>
</table>

**Physical Page Number**

<table>
<thead>
<tr>
<th></th>
<th>Ignored</th>
<th>X</th>
</tr>
</thead>
</table>

- **Present** bit is the next obvious bit
• An even worse bug → Foreshadow-NG/L1TF
• An even worse bug $\rightarrow$ Foreshadow-NG/L1TF
• Exploitable from VMs
• An even worse bug → Foreshadow-NG/L1TF
• Exploitable from VMs
• Allows leaking data from the L1 cache
Foreshadow-NG

- An even worse bug → Foreshadow-NG/L1TF
- Exploitable from VMs
- Allows leaking data from the L1 cache
- Same mechanism as Meltdown
• An even **worse** bug $\rightarrow$ Foreshadow-NG/L1TF
• Exploitable from **VMs**
• Allows **leaking** data from the **L1** cache
• Same mechanism as Meltdown
• Just a **different bit** in the PTE
<table>
<thead>
<tr>
<th>Page Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>PTE 0</td>
</tr>
<tr>
<td>PTE 1</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>PTE #PTI</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>PTE 511</td>
</tr>
</tbody>
</table>

L1 Cache
| PTE 0 | PTE 1 | ... | PTE #PTI | ... | PTE 511 |

Page Table

![Diagram showing Page Table and L1 Cache]

L1 Cache

present
Page Table

<table>
<thead>
<tr>
<th>PTE 0</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>PTE 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PTE #PTI</td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PTE 511</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

present

Guest Physical to Host Physical

L1 Cache
L1 lookup with physical address
### Page Table

<table>
<thead>
<tr>
<th>PTE 0</th>
<th>PTE 1</th>
<th>\ldots</th>
<th>PTE #PTI</th>
<th>PTE 511</th>
</tr>
</thead>
</table>

*not present*
Page Table

| PTE 0 |
| PTE 1 |
|   ... |
| PTE \#PTI |
|   ... |
| PTE 511 |

not present

L1 lookup with virtual address

L1 Cache
• KAISER/KPTI/KVA does not help
• KAISER/KPTI/KVA does not help
• Only software workarounds
• KAI$	ext{SER/KPTI/KVA}$ does not help
• Only software workarounds
  → Flush L1 on VM entry
- KAI$\text{SER}/KPTI/KVA does not help
- Only software workarounds
  - Flush L1 on VM entry
  - Disable HyperThreading
• KAISER/KPTI/KVA does not help
• Only software workarounds
  → Flush L1 on VM entry
  → Disable HyperThreading
• Workarounds might not be complete
Meltdown Variants

Pagefault
Meltdown Variants

Pagefault → Meltdown-US

Meltdown-US → Meltdown-US-L1
Meltdown-US → Meltdown-US-LFB
Meltdown Variants

- Pagefault
- Meltdown-US
- Meltdown-P
- Meltdown-US-L1
- Meltdown-US-L3
- Meltdown-US-LFB
Meltdown Variants

- Pagefault
  - Meltdown-US
  - Meltdown-P
  - Meltdown-RW
  - Meltdown-PK
  - Meltdown-US-L1
  - Meltdown-US-L3
  - Meltdown-US-LFB

Michael Schwarz (@misc0110) — Graz University of Technology
Meltdown Root Cause

operation \#n

time
Meltdown Root Cause

operation #n

data

time
Meltdown Root Cause

operation \#n

data
data dependency

operation \#n+2

time
Meltdown Root Cause

operation \#n

data
data dependency

operation \#n+2

transient execution

possibly architectural

time
Meltdown Root Cause

operation \#n

exception

data
data dependency

operation \#n+2

possibly architectural

transient execution

time

Michael Schwarz (@misc0110) — Graz University of Technology
Meltdown Root Cause

operation \#n

Retire

exception

Data

Meltdown

data dependency

operation \#n+2

possibly
architectural

transient execution

time

Michael Schwarz (@misc0110) — Graz University of Technology
Meltdown Root Cause

operation \#n

data

exception

raise

Meltdown

data dependency

operation \#n+2

possibly architectural

time

transient execution
Transient cause?
Meltdown Tree

Transient cause?

fault

Meltdown-type
Meltdown Tree

Transient cause?

Meltdown-type

- Meltdown-NM
  - Meltdown-AC
  - Meltdown-DE
  - Meltdown-PF
  - Meltdown-UD
  - Meltdown-SS
  - Meltdown-BR
  - Meltdown-GP
Transient cause?

Meltdown-type

Meltdown-NM
- Meltdown-AC
- Meltdown-DE
- Meltdown-PF
- Meltdown-UD
- Meltdown-SS
- Meltdown-BR
- Meltdown-GP

Meltdown-US
- Meltdown-US-L1
- Meltdown-US-L3
- Meltdown-US-LFB
- Meltdown-P
- Meltdown-RW
- Meltdown-PK
- Meltdown-XD
- Meltdown-SM
Meltdown Tree

Meltdown-Type

Transient cause?

Fault

Meltdown-NM

Meltdown-AC

Meltdown-DE

Meltdown-PF

Meltdown-UD

Meltdown-SS

Meltdown-BR

Meltdown-GP

Meltdown-US

Meltdown-US-L1

Meltdown-US-L3

Meltdown-US-LFB

Meltdown-P

Meltdown-RW

Meltdown-PK

Meltdown-XD

Meltdown-SM

Meltdown-MPX

Meltdown-BND

Michael Schwarz (@misc0110) — Graz University of Technology
• Meltdown is not a fully solved issue
There are two important points to note about the outlook for the Meltdown vulnerability:

1. **Meltdown is not a fully solved issue**
2. **The tree is extensible**
• Meltdown is not a fully solved issue
• The tree is extensible
• More Meltdown-type issues to come
• Meltdown is **not** a fully solved issue
• The tree is extensible
• **More** Meltdown-type issues to come
• Silicon fixes might not be complete
Transient Execution Attacks

- Meltdown not the only transient execution attacks
• Meltdown not the only transient execution attacks
• Spectre is a second class of transient execution attacks
Transient Execution Attacks

- Meltdown not the only transient execution attacks
- Spectre is a second class of transient execution attacks
- Instead of faults, exploit control (or data) flow predictions
• CPU tries to predict the future (branch predictor), ...
Speculative Execution

• CPU tries to predict the future (branch predictor), . . .
  • . . . based on events learned in the past
Speculative Execution

- CPU tries to predict the future (branch predictor), ...
  - ... based on events learned in the past
- Speculative execution of instructions
Speculative Execution

- CPU tries to predict the future (branch predictor), …
  - …based on events learned in the past
- Speculative execution of instructions
- If the prediction was correct, …
Speculative Execution

- CPU tries to predict the future (branch predictor), ...
  - ...based on events learned in the past
- Speculative execution of instructions
- If the prediction was correct, ...
  - ...very fast
Speculative Execution

- CPU tries to predict the future (branch predictor), …
  - …based on events learned in the past
- Speculative execution of instructions
- If the prediction was correct, …
  - …very fast
  - otherwise: Discard results
index = 0

if (index < 4) then
  glyph[data[index]]
else
  {}

Shared Memory

Memory

data[0]
data[1]
data[2]
data[3]
Spectre-PHT (aka Spectre Variant 1)

index = 0

if (index < 4)
then
  glyph[data[index]]
else
  {}

Michael Schwarz (@misc0110) — Graz University of Technology
index = 0

if (index < 4)
  glyph[data[index]]
else
  {}

Shared Memory

A B
C D E
F G H
I J K
L M N
O P Q
R S T
U V W
X Y Z

Memory

D
DATA
TAKE
KEY
...

data[0]
data[1]
data[2]
data[3]
Spectre-PHT (aka Spectre Variant 1)

index = 0

if (index < 4)

then

glyph[data[index]]

else

{}
index = 0

if (index < 4) {
    glyph[data[index]]
} else {

}
Spectre-PHT (aka Spectre Variant 1)

index = 1

if (index < 4)
    glyph[data[index]]
else
    
Memory

Shared Memory

\[
\begin{array}{cccc}
A & B & C & D \\
E & F & G & H \\
I & J & K & L \\
M & N & O & P \\
Q & R & S & T \\
U & V & W & X \\
Y & Z & & \\
\end{array}
\]
index = 1

if (index < 4)
    glyph[data[index]]
else
    {}
Spectre-PHT (aka Spectre Variant 1)

index = 1

if (index < 4)

then

glyph[data[index]]

else

{ }

Shared Memory

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>G</td>
<td>H</td>
<td>I</td>
<td>J</td>
</tr>
<tr>
<td>K</td>
<td>L</td>
<td>M</td>
<td>N</td>
<td>O</td>
</tr>
<tr>
<td>P</td>
<td>Q</td>
<td>R</td>
<td>S</td>
<td>T</td>
</tr>
<tr>
<td>U</td>
<td>V</td>
<td>W</td>
<td>X</td>
<td>Y</td>
</tr>
<tr>
<td>Z</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Memory

<table>
<thead>
<tr>
<th>DATA</th>
</tr>
</thead>
<tbody>
<tr>
<td>key</td>
</tr>
<tr>
<td>data[0]</td>
</tr>
<tr>
<td>data[1]</td>
</tr>
<tr>
<td>data[2]</td>
</tr>
<tr>
<td>data[3]</td>
</tr>
</tbody>
</table>

Speculate

Michael Schwarz (@misc0110) — Graz University of Technology
index = 1

if (index < 4)
    glyph[data[index]]
else
    {}

Shared Memory

Memory

data[0]
data[1]
data[2]
data[3]

Glyph

Michael Schwarz (@misc0110) — Graz University of Technology
Spectre-PHT (aka Spectre Variant 1)

\[
\text{index} = 1
\]

\[
\text{if (index} < 4) \\
\text{then} \\
\text{glyph[data[index]]} \\
\text{else}
\]

Michael Schwarz (@misc0110) — Graz University of Technology
index = 1

if (index < 4)

then

glyph[data[index]]

else

{}
Spectre-PHT (aka Spectre Variant 1)

Index = 2

Shared Memory

_glyph[data[index]]_

if (index < 4) then

else

Memory

index = 2

if (index < 4) {
  glyph[data[index]]
} else {
  \text{\textbf{T}}
}

index = 2

Speculative cache hit: 

if (index < 4) 

then 

glyph[data[index]]

else 

{}
Speculative cache lookup when accessing shared memory:

1. **Speculate**
   - If `index < 4`:
     - `glyph[data[index]]` returns a value.
2. **Shared Memory**
   - `index = 2`
3. **Memory**
   - `DATASYSTEM`
index = 2

Spectre-PHT (aka Spectre Variant 1)

Memory

D
A
T
A
K
E
Y
· · ·
data[0]
data[1]
data[2]
data[3]

Shared Memory

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

Execute

if (index < 4)

then

glyph[data[index]]

else

{}

\[
\text{index} = 2
\]

\[
\text{if (index < 4)}
\]

\[
\text{then}
\]

\[
\text{ glyph[data[index]]}
\]

\[
\text{else}
\]

\[
\{}
\]

Michael Schwarz (@misc0110) — Graz University of Technology
index = 3

if (index < 4) then

glyph[data[index]]

else

{}
Spectre-PHT (aka Spectre Variant 1)

index = 3

if (index < 4) then
glyph[data[index]]

else

{ }

Shared Memory

A
B
C
D
E
F
G
H
I
J
K
L
M
N
O
P
Q
R
S
T
U
V
W
X
Y
Z

Memory

D
A
T
A
K
E
Y

...
index = 3

if (index < 4) then
glyph[data[index]]
else
{}

A

D

data[0]
data[1]
data[2]
data[3]

A

T
index = 3

if (index < 4)

then

glyph[data[index]]

else

{}
Spectre-PHT (aka Spectre Variant 1)

index = 3

if (index < 4)

else

glyph[data[index]]

Data

{ data[0]
data[1]
data[2]
data[3] }
Spectre-PHT (aka Spectre Variant 1)

index = 4

if (index < 4)

then

glyph[data[index]]

else

{}
index = 4

if (index < 4)
    glyph[data[index]]
else
    {}

Shared Memory

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

Memory

DATA
KEY

Spectre-PHT (aka Spectre Variant 1)

index = 4

if (index < 4)
    glyph[data[index]]
else
    {}

Michael Schwarz (@misc0110) — Graz University of Technology
index = 4

if (index < 4)
    glyph[data[index]]
else
    {}
index = 4

if (index < 4)

then

glyph[data[index]]

else

{}
operation #n
operation \#n

prediction
Spectre Root Cause

- Operation #n
- Prediction
- Operation #n+2
- Time
- Predict CF/DF
Spectre Root Cause

operation \#n

prediction

cf/df

predict
to

operation \#n+2

possibly
architectural

transient execution

time
Spectre Root Cause

- operation \#n
- retire
- prediction
- operation \#n+2
- transient execution
- possibly architectural
- predict CF/DF

Michael Schwarz (@misc0110) — Graz University of Technology
Spectre Root Cause

- Operation \( #n \)
- Retire
- Operation \( #n+2 \)
- Predict
- CF/DF
- Possibly architectural
- Transient execution
- Time

Flush pipeline on wrong prediction
Spectre Root Cause

- operation \#n
- prediction
- predict CF/DF
- possibly architectural
- transient execution
- operation \#n+2
- retire
- flush pipeline on wrong prediction

Michael Schwarz (@misc0110) — Graz University of Technology
• Many predictors in modern CPUs
• Many predictors in modern CPUs
  • Branch taken/not taken (PHT)
• Many predictors in modern CPUs
  • Branch taken/not taken (PHT)
  • Call/Jump destination (BTB)
Many predictors in modern CPUs
- Branch taken/not taken (PHT)
- Call/Jump destination (BTB)
- Function return destination (RSB)
• Many predictors in modern CPUs
  • Branch taken/not taken (PHT)
  • Call/Jump destination (BTB)
  • Function return destination (RSB)
  • Load matches previous store (STL)
- Many predictors in modern CPUs
  - Branch taken/not taken (PHT)
  - Call/Jump destination (BTB)
  - Function return destination (RSB)
  - Load matches previous store (STL)
- Most are even shared among processes
Spectre Mistraining

Victim

same address space/
in place

Victim
branch
Spectre Mistraining

- same address space/out of place
- Congruent branch
- Address collision
- same address space/in place
- Victim branch
Spectre Mistraining

same address space/
out of place

Congruent
branch

Address
collision

same address space/
in place

Victim
branch

Shared Branch Prediction State

Michael Schwarz (@misc0110) — Graz University of Technology
Spectre Mistraining

same address space/
out of place

same address space/
in place

Congruent branch

Address collision

Victim branch

Shared Branch Prediction State

Congruent branch

Victim

Attacker
Spectre Mistraining

same address space/out of place

same address space/in place

Victim

Congruent branch

Address collision

victim branch

Attacker

Shadow branch

cross address space/in place

Shared Branch Prediction State

Michael Schwarz (@misc0110) — Graz University of Technology
Spectre Mistraining

Victim

Congruent branch

Address collision

Victim branch

Attacker

Congruent branch

Address collision

Shadow branch

Shared Branch Prediction State

same address space/
out of place

cross address space/
out of place

same address space/
in place

cross address space/
in place

Michael Schwarz (@misc0110) — Graz University of Technology
Transient cause?
Spectre Variants
Spectre Variants

Transient cause?

- Spectre-PHT
- Spectre-BTB
- Spectre-RSB
- Spectre-STL

microarchitectural buffer

prediction

Spectre-type

Transient cause?
Spectre Variants

Transient cause?

- Spectre-PHT
- Spectre-BTB
- Spectre-RSB
- Spectre-STL

Microarchitectural buffer

Mistraining strategy

- Cross-address-space
- Same-address-space

Prediction

Cross-address-space
Same-address-space
Cross-address-space
Same-address-space
Cross-address-space
Same-address-space
Spectre Variants

- Spectre-PHT
- Spectre-BTB
- Spectre-RSB
- Spectre-STL

Microarchitectural buffer

Prediction

Transient cause?

Mistraining strategy

In-place (IP) vs. out-of-place (OP)

- PHT-CA-IP
- PHT-CA-OP
- PHT-SA-IP
- PHT-SA-OP
- BTB-CA-IP
- BTB-CA-OP
- BTB-SA-IP
- BTB-SA-OP
- RSB-CA-IP
- RSB-CA-OP
- RSB-SA-IP
- RSB-SA-OP
• Spectre is **not a bug**
Spectre Fix

- Spectre is not a bug
- It is an useful optimization
• Spectre is not a bug
• It is an useful optimization
→ Cannot simply fix it (as with Meltdown)
Spectre Fix

- Spectre is **not a bug**
- It is an useful **optimization**
  → Cannot simply fix it (as with Meltdown)
- **Workarounds** for critical code parts
Spectre defenses in 3 categories:

**C1** Mitigating or reducing the accuracy of covert channels

**C2** Mitigating or aborting speculation

**C3** Ensuring secret data cannot be reached
### Spectre: Defense Analysis

<table>
<thead>
<tr>
<th>Attack</th>
<th>Defense</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>InvisiSpec</td>
</tr>
<tr>
<td></td>
<td>SafeSpec</td>
</tr>
<tr>
<td></td>
<td>DAWG</td>
</tr>
<tr>
<td></td>
<td>RSB Stuffing</td>
</tr>
<tr>
<td></td>
<td>Retpoline</td>
</tr>
<tr>
<td></td>
<td>Poison Value</td>
</tr>
<tr>
<td></td>
<td>Index Masking</td>
</tr>
<tr>
<td></td>
<td>Site Isolation</td>
</tr>
<tr>
<td></td>
<td>SLH</td>
</tr>
<tr>
<td></td>
<td>YSNB</td>
</tr>
<tr>
<td></td>
<td>IBRS</td>
</tr>
<tr>
<td></td>
<td>STPBP</td>
</tr>
<tr>
<td></td>
<td>IBPB</td>
</tr>
<tr>
<td></td>
<td>Serialization</td>
</tr>
<tr>
<td></td>
<td>Taint Tracking</td>
</tr>
<tr>
<td></td>
<td>Timer Reduction</td>
</tr>
<tr>
<td></td>
<td>Sloth</td>
</tr>
<tr>
<td></td>
<td>SSBD/SSBB</td>
</tr>
</tbody>
</table>

Intel

- Spectre-PHT
- Spectre-BTB
- Spectre-RSB
- Spectre-STL

Attack is mitigated (●), partially mitigated (○), not mitigated (●), theoretically mitigated (■), theoretically impeded (□), not theoretically impeded (◇), or out of scope (◇).
<table>
<thead>
<tr>
<th>Attack</th>
<th>Defense</th>
<th>Spectre-PHT</th>
<th>Spectre-BTB</th>
<th>Spectre-RSB</th>
<th>Spectre-STL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel</td>
<td></td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
</tbody>
</table>

Attack is mitigated (●), partially mitigated (○), not mitigated (◇), theoretically mitigated (■), theoretically impeded (□), not theoretically impeded (◇), or out of scope (◇).
## Spectre: Defense Analysis

<table>
<thead>
<tr>
<th>Attack</th>
<th>Defense</th>
<th>InvisiSpec</th>
<th>SafeSpec</th>
<th>DAWG</th>
<th>RSB Stuffing</th>
<th>Retpoline</th>
<th>Poison Value</th>
<th>Index Masking</th>
<th>Site Isolation</th>
<th>SLH</th>
<th>YSNB</th>
<th>IBRS</th>
<th>STIPB</th>
<th>IBPB</th>
<th>Serialization</th>
<th>Taint Tracking</th>
<th>Timer Reduction</th>
<th>Sloth</th>
<th>SSBD/SSBB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spectre-PHT</td>
<td></td>
<td>●</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Spectre-BTB</td>
<td></td>
<td></td>
<td>○</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Spectre-RSB</td>
<td></td>
<td>●</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Spectre-STL</td>
<td></td>
<td></td>
<td>○</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Attack is mitigated (●), partially mitigated (○), not mitigated (◇), theoretically mitigated (■), theoretically impeded (□), not theoretically impeded (◇), or out of scope (◇).
<table>
<thead>
<tr>
<th>Attack</th>
<th>Defense</th>
<th>InvisiSpec</th>
<th>SafeSpec</th>
<th>DAWG</th>
<th>RSB Stuffing</th>
<th>Retpoline</th>
<th>Poison Value</th>
<th>Index Masking</th>
<th>Site Isolation</th>
<th>SLH</th>
<th>YSNB</th>
<th>IFRS</th>
<th>STIPB</th>
<th>IBPB</th>
<th>Serialization</th>
<th>Taint Tracking</th>
<th>Timer Reduction</th>
<th>Sloth</th>
<th>SSBD/SSBB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel</td>
<td>Spectre-PHT</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>○</td>
<td>●</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>●</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td></td>
<td>Spectre-BTB</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>○</td>
<td>●</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>●</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td></td>
<td>Spectre-RSB</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>○</td>
<td>●</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>●</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td></td>
<td>Spectre-STL</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>○</td>
<td>●</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>●</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
</tbody>
</table>

Attack is mitigated (●), partially mitigated (○), not mitigated (□), theoretically mitigated (■), theoretically impeded (■), not theoretically impeded (□), or out of scope (◇).
<table>
<thead>
<tr>
<th>Attack</th>
<th>Defense</th>
<th>InvisiSpec</th>
<th>SafeSpec</th>
<th>DAWG</th>
<th>RSB Stuffing</th>
<th>Poison Value</th>
<th>Index Masking</th>
<th>Site Isolation</th>
<th>SLH</th>
<th>YSNB</th>
<th>IBRS</th>
<th>STIPB</th>
<th>IBPB</th>
<th>Serialization</th>
<th>Taint Tracking</th>
<th>Timer Reduction</th>
<th>Sloth</th>
<th>SSBD/SSBB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel</td>
<td>Spectre-PHT</td>
<td>●</td>
<td>○</td>
<td>○</td>
<td>●</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>●</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Spectre-BTB</td>
<td>○</td>
<td>●</td>
<td>○</td>
<td>●</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>●</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Spectre-RSB</td>
<td>○</td>
<td>●</td>
<td>○</td>
<td>●</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>●</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Spectre-STL</td>
<td>○</td>
<td>●</td>
<td>○</td>
<td>●</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>●</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Attack is mitigated (●), partially mitigated (○), not mitigated (◇), theoretically mitigated (■), theoretically impeded (□), not theoretically impeded (◇), or out of scope (◇). 
### Spectre: Defense Analysis

**Intel**

<table>
<thead>
<tr>
<th>Attack</th>
<th>InvisiSpec</th>
<th>SafeSpec</th>
<th>DAWG</th>
<th>RSB Stuffing</th>
<th>Poison Value</th>
<th>Index Masking</th>
<th>Site Isolation</th>
<th>SLH</th>
<th>YSNB</th>
<th>IBRS</th>
<th>STIPB</th>
<th>IBPB</th>
<th>Serialization</th>
<th>Taint Tracking</th>
<th>Timer Reduction</th>
<th>Sloth</th>
<th>SSBD/SSBB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spectre-PHT</td>
<td>●</td>
<td>○</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Spectre-BTB</td>
<td>●</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>○</td>
<td>○</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Spectre-RSB</td>
<td></td>
<td>○</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>○</td>
<td>○</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Spectre-STL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>○</td>
<td>○</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Attack is mitigated (●), partially mitigated (○), not mitigated ( ○), theoretically mitigated (●), theoretically impeded (□), not theoretically impeded (□), or out of scope (◇).
### Spectre: Defense Analysis

<table>
<thead>
<tr>
<th>Attack</th>
<th>InvisiSpec</th>
<th>SafeSpec</th>
<th>DAWG</th>
<th>RSB Stuffing</th>
<th>Retpoline</th>
<th>Poison Value</th>
<th>Index Masking</th>
<th>Site Isolation</th>
<th>SLH</th>
<th>YSNB</th>
<th>IBRS</th>
<th>STIPB</th>
<th>IBPB</th>
<th>Serialization</th>
<th>Taint Tracking</th>
<th>Timer Reduction</th>
<th>Sloth</th>
<th>SSBD/SSBB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spectre-PHT</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>☐</td>
<td></td>
<td></td>
<td></td>
<td>☐</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Spectre-BTB</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td></td>
<td>☐</td>
<td></td>
<td></td>
<td></td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td></td>
<td>☐</td>
<td></td>
<td></td>
<td></td>
<td>☐</td>
<td></td>
</tr>
<tr>
<td>Spectre-RSB</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td></td>
<td>☐</td>
<td></td>
<td></td>
<td></td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td></td>
<td>☐</td>
<td></td>
<td></td>
<td></td>
<td>☐</td>
<td></td>
</tr>
<tr>
<td>Spectre-STL</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>☐</td>
<td></td>
<td></td>
<td></td>
<td>☐</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Attack is mitigated (●), partially mitigated (○), not mitigated (☐), theoretically mitigated (■), theoretically impeded (□), not theoretically impeded (◇), or out of scope (◇).
### Spectre: Defense Analysis

<table>
<thead>
<tr>
<th>Attack</th>
<th>Defense</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>InvisiSpec</td>
</tr>
<tr>
<td>Spectre-PHT</td>
<td>□</td>
</tr>
<tr>
<td>Spectre-BTB</td>
<td>□</td>
</tr>
<tr>
<td>Spectre-RSB</td>
<td>□</td>
</tr>
<tr>
<td>Spectre-STL</td>
<td>□</td>
</tr>
</tbody>
</table>

Attack is mitigated (●), partially mitigated (○), not mitigated (□), theoretically mitigated (■), theoretically impeded (■), not theoretically impeded (□), or out of scope (◇).
• Many countermeasures only consider the cache to get data...
• Many countermeasures only consider the cache to get data...
• ...but there are other possibilities, e.g.,
• Many countermeasures only consider the cache to get data...
• ...but there are other possibilities, e.g.,
  • Port contention (SMoTherSpectre)
• Many countermeasures only consider the cache to get data...
• ...but there are other possibilities, e.g.,
  • Port contention (SMoTherSpectre)
  • AVX (NetSpectre)
• Many countermeasures only consider the cache to get data...
• ...but there are other possibilities, e.g.,
  • Port contention (SMoTherSpectre)
  • AVX (NetSpectre)
• Cache is just the easiest
On Friday marked the release of the Linux 4.19.4 kernel as well as 4.14.83 and 4.9.139.

Greg Kroah-Hartman issued this latest round of stable point releases as basic maintenance updates. While these point releases don't tend to be too notable and generally go unmentioned on Phoronix, this round is worth pointing out since 4.19.4 and 4.14.83 are the releases that end up reverting the STIBP behavior that applied Single Thread Indirect Branch Predictors to all processes on supported systems. That is what was introduced in Linux 4.20 and then back-ported to the 4.19/4.14 LTS branches, which in turn hurt the performance a lot. So for now the code is removed.

As covered yesterday, there is improved STIBP code on the way for Linux 4.20 that by default just apply STIBP to SECCOMP threads and processes requesting it via prctl() but otherwise is off by default (that behavior can also be changed via kernel parameters).
On Friday marked the release of the Linux 4.19.4 kernel as well as 4.14.83 and 4.9.139.

Greg Kroah-Hartman issued this latest round of stable point releases as basic maintenance updates. While these point releases don't tend to be too notable and generally go unmentioned on Phoronix, this round is worth pointing out since 4.19.4 and 4.14.83 are the releases that end up reverting the STIBP behavior that applied Single Thread Indirect Branch Predictors to all processes on supported systems. That is what was introduced in Linux 4.20 and then back-ported to the 4.19/4.14 LTS branches, which in turn hurt the performance a lot. So for now the code is removed.

As covered yesterday, there is improved STIBP code on the way for Linux 4.20 that by default just apply STIBP to SECCOMP threads and processes requesting it via prctl() but otherwise is off by default (that behavior can also be changed via kernel parameters).
Linux 4.19.4 & 4.14.83 Released With STIBP Code Dropped

Written by Michael Larabel in Linux Kernel on 24 November 2018 at 09:00 AM EST. 6 Comments

On Friday marked the release of the Linux 4.19.4 kernel as well as 4.14.83 and 4.9.139.

Greg Kroah-Hartman issued this latest round of stable point releases as basic maintenance updates. While these point releases don't tend to be too notable and generally go unmentioned on Phoronix, this round is worth pointing out since 4.19.4 and 4.14.83 are the releases that end up reverting the STIBP behavior that applied Single Thread Indirect Branch Predictors to all processes on supported systems. That is what was introduced in Linux 4.20 and then back-ported to the 4.19/4.14 LTS branches, which in turn hurt the performance a lot. So for now the code is removed.

As covered yesterday, there is improved STIBP code on the way for Linux 4.20 that by default just apply STIBP to SECCOMP threads and processes requesting it via prctl() but otherwise is off by default (that behavior can also be changed via kernel parameters).
Linux 4.19.4 & 4.14.83 Released With STIBP Code Dropped

On Friday marked the release of the Linux 4.19.4 kernel as well as 4.14.83 and 4.9.139.

Greg Kroah-Hartman issued this latest round of stable point releases as basic maintenance updates. While these point releases don't tend to be too notable and generally go unmentioned on Phoronix, this round is worth pointing out since 4.19.4 and 4.14.83 are the releases that end up reverting the STIBP behavior that applied Single Thread Indirect Branch Predictors to all processes on supported systems. That is what was introduced in Linux 4.20 and then back-ported to the 4.19/4.14 LTS branches, which in turn hurt the performance a lot. So for now the code is removed.

As covered yesterday, there is improved STIBP code on the way for Linux 4.20 that by default just apply STIBP to SECCOMP threads and processes requesting it via prctl() but otherwise is off by default (that behavior can also be changed via kernel parameters).
Retpoline (compiler extension)
Retpoline (compiler extension)

```assembly
push <call_target>
call 1f
2:           ; speculation continues here
lfence       ; speculation barrier
jmp 2b       ; endless loop
1:           
    lea 8(%rsp), %rsp ; restore stack pointer
    ret                 ; the call to <call_target>
```

→ Always predict to enter an endless loop
Retpoline (compiler extension)

```assembly
push  <call_target>
call  1f
2: ; speculation continues here
lfence ; speculation barrier
jmp  2b ; endless loop
1:
lea  8(%rsp), %rsp ; restore stack pointer
ret ; the call to <call_target>
```

→ Always predict to enter an endless loop

• What if someone decides to fix the wrong prediction?
• Current mitigations are either incomplete or cost performance
Current mitigations are either incomplete or cost performance
→ More research required
• Current mitigations are either incomplete or cost performance
→ More research required
• Both on attacks and defenses
• Current mitigations are either incomplete or cost performance
→ More research required

• Both on attacks and defenses
→ Efficient defenses only possible when attacks are known
Transient Execution Attacks

Transient cause?

Spectre-type
- Spectre-PHT
- Spectre-BTB
- Spectre-RSB
- Spectre-STL

Meltdown-type
- Meltdown-NM
- Meltdown-AC
- Meltdown-DE
- Meltdown-PF
- Meltdown-UD
- Meltdown-SS
- Meltdown-BR
- Meltdown-GP

Prediction

Microarchitectural buffer

Fault

In-place (IP) vs., out-of-place (OP)
mistraining strategy

Cross-address-space
- PHT-CA-IP
- BTB-CA-IP
- RSB-CA-IP
- Meltdown-US-L1

Same-address-space
- PHT-SA-IP
- BTB-SA-IP
- RSB-SA-IP
- Meltdown-US-L3

Meltdown-US-LFB

Meltdown-AC
- Meltdown-US
- Meltdown-P
- Meltdown-RW
- Meltdown-PK
- Meltdown-XD
- Meltdown-SM
- Meltdown-MPX
- Meltdown-BND

Meltdown-DE
- Meltdown-US-L1

Meltdown-PF
- Meltdown-US-L3

Meltdown-UD
- Meltdown-US-LFB

Meltdown-SS
- Meltdown-US-L1

Meltdown-BR
- Meltdown-US-L3

Meltdown-GP
- Meltdown-US-LFB

Meltdown-NM
- Meltdown-US-L1

Meltdown-AC
- Meltdown-US-L3

Meltdown-DE
- Meltdown-US-LFB

Meltdown-PF
- Meltdown-US-L1

Meltdown-UD
- Meltdown-US-L3

Meltdown-SS
- Meltdown-US-LFB

Meltdown-BR
- Meltdown-US-L1

Meltdown-GP
- Meltdown-US-L3

Meltdown-NM
- Meltdown-US-LFB

Meltdown-AC
- Meltdown-US-L1

Meltdown-DE
- Meltdown-US-L3

Meltdown-PF
- Meltdown-US-LFB

Meltdown-UD
- Meltdown-US-L1

Meltdown-SS
- Meltdown-US-L3

Meltdown-BR
- Meltdown-US-LFB

Meltdown-GP
- Meltdown-US-L1

Meltdown-NM
- Meltdown-US-L3

Meltdown-AC
- Meltdown-US-LFB

Meltdown-DE
- Meltdown-US-L1

Meltdown-PF
- Meltdown-US-L3

Meltdown-UD
- Meltdown-US-LFB

Meltdown-SS
- Meltdown-US-L1

Meltdown-BR
- Meltdown-US-L3

Meltdown-GP
- Meltdown-US-LFB

Michael Schwarz (@misc0110) — Graz University of Technology
Transient Execution Attacks are...
Transient Execution Attacks are...
  ...a novel class of attacks
Transient Execution Attacks

- Transient Execution Attacks are...
  - ...a novel class of attacks
  - ...extremely powerful
• Transient Execution Attacks are...
  • ...a novel class of attacks
  • ...extremely powerful
  • ...only at the beginning
Transient Execution Attacks are...

- a novel class of attacks
- extremely powerful
- only at the beginning

Many optimizations introduce side channels → now exploitable
BRACE YOURSELVES
MORE BUGS ARE COMING
Exploiting the Microarchitecture: Transient Execution Attacks

Michael Schwarz (@misc0110)
April 11, 2019

Graz University of Technology