Transient Execution Attacks

Exploiting the CPU’s Microarchitecture

Michael Schwarz (@misc0110), Moritz Lipp (@mlqxyz)

October 7, 2019

Graz University of Technology
Michael Schwarz
PhD candidate @ Graz University of Technology

@misc0110

michael.schwarz@iaik.tugraz.at
Who am I?

Moritz Lipp
PhD candidate @ Graz University of Technology

@mlqxyz

mail@mlq.me
Let’s Read Kernel Memory from User Space!
• Find something human readable, e.g., the Linux version

```
# sudo grep linux_banner /proc/kallsyms
fffffffff81a000e0 R linux_banner
```
char data = *(char*) 0xffffffff81a000e0;
printf("%c\n", data);
 Compile and run

```
segbrcut at ffffffffb5000e0
ip 0000000000000000400535
sp 00007ffce4a80610
error 5 in reader
```
• Compile and run

```
segfault at ffffffff81a000e0
  ip 00000000000400535
  sp 00007ffce4a80610
error 5 in reader
```

• Kernel addresses are of course not accessible

Michael Schwarz (@misc0110), Moritz Lipp (@mlqxyz) — Graz University of Technology
• Compile and run

```
segfault at fffffffffff81a000e0
  ip 00000000000400535
  sp 00007ffce4a80610
  error 5 in reader
```

• Kernel addresses are of course not accessible

• Any invalid access throws an exception → segmentation fault
Memory Isolation

- Kernel is isolated from user space
- **Isolation**: combination of hardware and software
- Applications cannot access kernel
- Well-defined interface → **system calls**

![Diagram showing the separation between userspace and kernelspace with a wall as a metaphor for isolation.](image-url)
• Virtual address spaces isolate processes
• Virtual address spaces isolate processes
• Physical memory organized in page frames
Virtual address spaces isolate processes
Physical memory organized in page frames
Virtual memory pages are mapped to page frames using page tables
### 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>
</table>

- User/Supervisor bit defines in which **privilege level** the page can be accessed
Loading an address
Loading an address
Loading an address
Loading an address
Building the Code

• **Catch** the segmentation fault!
• **Catch** the segmentation fault!
→ Install a signal handler
• **Catch** the segmentation fault!
  
  → Install a signal handler

• On exception → jump back and continue
Still no kernel memory
Still no kernel memory
Privilege checks seem to work
• Still no kernel memory
• Privilege checks seem to work
• Back to the drawing board
• We cannot see architectural changes
• We cannot see architectural changes
• What about the CPU internals?
• We cannot see architectural changes
• What about the CPU internals?
→ The microarchitectural state
printf("%d", i);
printf("%d", i);
printf("%d", i);
printf("%d", i);
printf("%d", i);
printf("%d", i);
printf("%d", i);

printf("%d", i);
CPU Cache

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

Cache miss
Request
Response

Michael Schwarz (@misc0110), Moritz Lipp (@mlqxyz) — Graz University of Technology
`printf("%d", i);`

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

Cache miss

Cache hit
CPU Cache

DRAM access,
slow

printf("%d", i);
Cache miss

printf("%d", i);
Cache hit

Request
Response
CPU Cache

DRAM access,
slow

printf("%d", i);

Cache miss

printf("%d", i);

Cache hit

No DRAM access,
much faster

Request

Response

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

Access time [CPU cycles]

Number of accesses

Cache Hits

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

- Cache Hits
- Cache Misses

Access time [CPU cycles]

Number of accesses

Michael Schwarz (@misc0110), Moritz Lipp (@mlqxyz) — Graz University of Technology
Flush+Reload

Attacker

flush
access

Shared Memory

Victim

access

Michael Schwarz (@misc0110), Moritz Lipp (@mlqxyz) — Graz University of Technology
Flush+Reload

Attacker

flush

access

Shared Memory

cached

Victim

access

Shared Memory

cached

Michael Schwarz (@misc0110), Moritz Lipp (@mlqxyz) — Graz University of Technology
Flush+Reload

Attacker

flush
access

Shared Memory

Victim

access

Michael Schwarz (@misc0110), Moritz Lipp (@mlqxyz) — 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

\texttt{flush}

\texttt{access}

\texttt{Shared Memory}

Victim

\texttt{access}

Michael Schwarz (@misc0110), Moritz Lipp (@mlqxyz) — Graz University of Technology
Flush+Reload

Attacker

\texttt{flush}

\texttt{access}

\begin{align*}
\text{Shared Memory} & \quad \text{Victim} \\
\text{access} & \quad \text{Victim accessed (fast)} \quad \text{vs} \quad \text{Victim did not access (slow)}
\end{align*}

Michael Schwarz (@misc0110), Moritz Lipp (@mlqxyz) — Graz University of Technology
• Can we see illegal loads in the cache?
• Can we see illegal loads in the cache?
• What happens if the load cannot continue?
Out-of-order Execution
```c
int width = 10, height = 5;

float diagonal = sqrt(width * width + height * height);
int area = width * height;

printf("Area %d x %d = %d\n", width, height, area);
```
```c
int width = 10, height = 5;
float diagonal = sqrt(width * width + height * height);
int area = width * height;
printf("Area %d x %d = %d\n", width, height, area);
```
Out-of-order Execution

- Instructions are fetched and decoded in the **front-end**
- Instructions are dispatched to the **backend**
- Instructions are processed by individual execution units
Out-of-order Execution

- Instructions are executed **out-of-order**
- Instructions wait until their **dependencies are ready**
  - Later instructions might execute prior earlier instructions
- Instructions **retire in-order**
  - State becomes architecturally visible
• Adapted code

\[(volatile\ char\*) 0;\]
\[\text{array}[84\ \ast\ 4096] = 0;\]
• Adapted code

\[
*(volatile char*)0;
array[84 * 4096] = 0;
\]

• volatile because compiler was not happy

1 warning: statement with no effect [-Wunused-value]
2 \*(char*)0;
Building the Code

- Adapted code

  ```c
  *(volatile char *)0;
  array[84 * 4096] = 0;
  ```

- `volatile` because compiler was not happy

  1 `warning`: statement with no effect [−Wunused−value]
  2 `*(char *)0;`

- Static code analyzer is still not happy

  1 `warning`: Dereference of null pointer
  2 `*(volatile char *)0;`
• 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
Out-of-order instructions leave microarchitectural traces
Out-of-order instructions leave microarchitectural traces
We can see them for example in the cache
• Out-of-order instructions leave microarchitectural traces
• We can see them for example in the cache
• Give such instructions a name: transient instructions
• Out-of-order instructions leave microarchitectural traces
• We can see them for example in the cache
• Give such instructions a name: transient instructions
• We can indirectly observe the execution of transient instructions
• Maybe there is no permission check in transient instructions...
• Maybe there is no permission check in transient instructions...
• ...or it is only done when committing them
• Maybe there is no permission check in transient instructions...
• ...or it is only done when committing them
• Add another layer of indirection to test

```c
char data = *(char*) 0xffffffff81a000e0;
array[data * 4096] = 0;
```
• Maybe there is no permission check in transient instructions...
• ...or it is only done when committing them
• Add another layer of indirection to test

\[
\text{char} \; \text{data} = *(\text{char}*) \; 0xffffffff81a000e0; \\
\text{array}[\text{data} \times 4096] = 0;
\]

• 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 not fast enough
• Using out-of-order execution, we can read data at any address
• Using out-of-order execution, we can read data at any address
• Privilege checks are sometimes too slow
Meltdown

- Using out-of-order execution, we can read data at any address
- Privilege checks are sometimes too slow
- Allows to leak kernel memory
• Using out-of-order execution, we can read data at any address
• Privilege checks are sometimes too slow
• Allows to leak kernel memory
• Entire physical memory is typically also accessible in kernel address space
• Using out-of-order execution, we can read data at any address
• Privilege checks are sometimes too slow
• Allows to leak kernel memory
• Entire physical memory is typically also accessible in kernel address space
• Works on Intel CPUs and ARM Cortex-A75
• 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
Meltdown Mitigation: KAISER

Userspace

Kernelspace

Applications

Operating System

Memory

Michael Schwarz (@misc0110), Moritz Lipp (@mlqxyz) — Graz University of Technology
Kernel View

- Userspace
- Kernelspace

Operating System
Memory

User View

- Userspace
- Kernelspace

Applications

context switch

Michael Schwarz (@misc0110), Moritz Lipp (@mlqxyz) — Graz University of Technology
• **Linux**: Kernel Page-table Isolation (KPTI)
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
• Meltdown **fully mitigated** in software
Problem Solved?

- Meltdown fully mitigated in software
- Problem seemed to be solved
Meltdown fully mitigated in software
Problem seemed to be solved
No attack surface left
Problem Solved?

- 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
• There are more bits...
### 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**

Ignored  X
- **Present** bit is the next obvious bit
An even worse bug → Foreshadow-NG/L1TF
• An even **worse** bug → **Foreshadow-NG/L1TF**
• Exploitable from **VMs**
• An even worse bug → Foreshadow-NG/L1TF
• Exploitable from VMs
• Allows leaking data from the L1 cache
• 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 → Foreshadow-NG/L1TF
• Exploitable from **VMs**
• Allows **leaking** data from the **L1** cache
• Same mechanism as Meltdown
• Just a **different bit** in the PTE
Page Table

PTE 0
PTE 1

·

PTE #PTI

·

PTE 511

L1 Cache
<table>
<thead>
<tr>
<th>PTE 0</th>
<th>PTE 1</th>
<th>\vdots</th>
<th>PTE #PTI</th>
<th>\vdots</th>
<th>PTE 511</th>
</tr>
</thead>
</table>

Page Table

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
Page Table

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

Guest Physical to Host Physical

Physical Page

L1 Cache

L1 lookup with physical address
### Page Table

| PTE 0 |  
|-------|---
| PTE 1 |  
|       |  
| PTE #PTI |  
|       |  
| PTE 511 |  

not present

### L1 Cache
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
• KAISER/KPTI/KVA does not help
• Only software workarounds
  → Flush L1 on VM entry
• KAISER/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
What if the memory is not cached?
• What if the memory is not cached?
• No data $\rightarrow$ no leakage?
Flush the Target

- User mapping
  - Page $p$
  - 2 MB

- cache line
  - 4 KB
  - 2 MB
Flush the Target

Kernel address $k$

2 MB

cache line

Page $p$

2 MB

User mapping $\nu$

4 KB

4 KB

Michael Schwarz (@misc0110), Moritz Lipp (@mlqxyz) — Graz University of Technology
Flush the Target

Kernel address \(k\)

Page \(p\)

User mapping \(\nu\)

flush

2 MB

4 KB

4 KB

2 MB

2 MB

2 MB

Michael Schwarz (@misc0110), Moritz Lipp (@mlqxyz) — Graz University of Technology
Flush the Target

faulting load

Kernel address $k$

2 MB

cache line

Page $p$

2 MB

flush

User mapping $v$

4 KB

4 KB

Michael Schwarz (@misc0110), Moritz Lipp (@mlqxyz) — Graz University of Technology
THERE IS NO NOISE

NOISE IS JUST SOMEONE ELSE'S DATA
Line-fill Buffers

Execution Engine

Scheduler

Execution Units

Load data

Store data

ALU, AES, ...

ALU, FMA, ...

ALU, Vect, ...

ALU, Branch

Core Memory

Load Buffer

Store Buffer

L1 Data Cache

DTLB

LFB

Michael Schwarz (@misc0110), Moritz Lipp (@mlqxyz) — Graz University of Technology
Line-fill Buffers

CDB -> Reorder buffer -> Scheduler -> Execution Units

Execution Units:
- ALU, AES, ...
- ALU, FMA, ...
- ALU, Vect, ...
- ALU, Branch

Load data
Store data
AGU

Core Memory:
- Load Buffer
- Store Buffer
- L1 Data Cache
- DTLB
- LFB

... mov al, byte [rcx] ...

Michael Schwarz (@misc0110), Moritz Lipp (@mlqxyz) — Graz University of Technology
Line-fill Buffers

... mov al, byte [rcx] ...

Michael Schwarz (@misc0110), Moritz Lipp (@mlqxyz) — Graz University of Technology
Line-fill Buffers

mov al, byte [rcx]
Line-fill Buffers

Execution Engine

Reorder buffer

Scheduler

CDB

Execution Units

ALU, AES, ALU, FMA, ALU, Vect, ALU, Branch

Load data

Load data

Store data

AGU

Core Memory

Load Buffer

Store Buffer

L1 Data Cache

DTLB

LFB

... mov al, byte [rcx] ...

complex load situation! need to reissue this load! STOP!!
Line-fill Buffers

mov al, byte [rcx]

...
Line-fill Buffers

... mov al, byte [rcx] ...

- **Execution Engine**
  - Scheduler
  - Execution Units: ALU, AES, FMA, Vect, Branch
  - Load data
  - Store data
  - AGU

- **Core Memory**
  - Load Buffer
  - Store Buffer
  - L1 Data Cache
  - DTLB
  - LFB

- **CDB**
  - Reorder buffer
  - μOP μOP μOP μOP μOP μOP μOP μOP μOP μOP μOP μOP μOP μOP μOP μOP

- **Complex load situation!** Need to reissue this load! STOP!!
...mov al, byte [rcx]...

complex load situation! need to reissue this load! STOP!!
Line-fill Buffers

Execution Engine

Reorder buffer

Scheduler

Execution Units

ALU, AES, ...

ALU, FMA, ...

ALU, Vect, ...

ALU, Branch

Load data

Load data

Store data

AGU

CDB

Core Memory

Load Buffer

Store Buffer

L1 Data Cache

DTLB

LFB

... mov al, byte [rcx] ...
Line-fill Buffers

data can go to register

complex load situation! need to reissue this load! STOP!!
<table>
<thead>
<tr>
<th>Data Sampling</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Meltdown</strong></td>
</tr>
<tr>
<td>Physical</td>
</tr>
<tr>
<td>Virtual</td>
</tr>
<tr>
<td>Page Offset</td>
</tr>
<tr>
<td><strong>Foreshadow</strong></td>
</tr>
<tr>
<td>Physical</td>
</tr>
<tr>
<td>Virtual</td>
</tr>
<tr>
<td>Page Offset</td>
</tr>
<tr>
<td><strong>Fallout</strong></td>
</tr>
<tr>
<td>Physical</td>
</tr>
<tr>
<td>Virtual</td>
</tr>
<tr>
<td>Page Offset</td>
</tr>
<tr>
<td><strong>ZombieLoad/RIDL</strong></td>
</tr>
<tr>
<td>Physical</td>
</tr>
<tr>
<td>Virtual</td>
</tr>
<tr>
<td>Page Offset</td>
</tr>
</tbody>
</table>
Latest Meltdown Variant: ZombieLoad

- Leaks from the fill buffer
Latest Meltdown Variant: ZombieLoad

-Leaks from the fill buffer
- Crosses all privilege boundaries (Kernel, VM, SGX)
Latest Meltdown Variant: ZombieLoad

- Leaks from the fill buffer
- Crosses all privilege boundaries (Kernel, VM, SGX)
- Explored microcode assists as new type of faults

Michael Schwarz (@misc0110), Moritz Lipp (@mlqxyz) — Graz University of Technology
Latest Meltdown Variant: ZombieLoad

• Leaks from the fill buffer
• Crosses all privilege boundaries (Kernel, VM, SGX)
• Explored microcode assists as new type of faults
• Disadvantage: minimal control over leaked data
Mitigations?

- Disable HyperThreading
Mitigations?

- Disable HyperThreading
- Microcode Updates (to an extent)
Mitigations?

- Disable HyperThreading
- **Microcode** Updates (to an extent)
- `VERW` instruction → clear buffers on context switch
Overhead?

<table>
<thead>
<tr>
<th>Test Case</th>
<th>Pre Mitigation</th>
<th>Post Mitigation</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSmark 2014 SE Overall score</td>
<td>100%</td>
<td>99%</td>
</tr>
<tr>
<td>WebXPRT 3 Overall score</td>
<td>100%</td>
<td>97%</td>
</tr>
<tr>
<td>SPECint_rate_base 2017 (1 copy)</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>SPECint_rate_base 2017 (1 copy)</td>
<td>100%</td>
<td>101%</td>
</tr>
<tr>
<td>3DMark Skydiver Overall score</td>
<td>100%</td>
<td>100%</td>
</tr>
</tbody>
</table>

- **Intel® Core™ i9-9900K Processor; MDS mitigations not applied, Intel® HT enabled**
- **Intel® Core™ i9-9900K Processor; MDS mitigations applied, Intel® HT enabled**

View Full Image
Overhead?

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Overall Score 2014 SE</th>
<th>Overall Score WebXPR 3</th>
<th>SPECint_rate_base 2017 (n copy)</th>
<th>SPECint_rate_base 2017 (1 copy)</th>
<th>3DMark Skydive Overall Score</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
</tbody>
</table>

- Intel® Core™ i9-9900K Processor; MDS mitigations not applied, Intel® HT enabled
- Intel® Core™ i9-9900K Processor; MDS mitigations applied, Intel® HT enabled

Performance results are based on testing as of May 8, 2019 and may not reflect the publicly available security update. See configuration details for details. No product or component can be absolutely secure. Software and workloads used in performance tests may have been optimized for performance only on Intel® microprocessors. Results may vary for other microprocessors or workloads. Any change to any of these factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more complete information about performance and benchmark results, visit http://www.intel.com/benchmarks.
Software and workloads used in performance tests may have been optimized for the performance only on Intel microprocessors. [...] Any chance of any of those factors cause the results to vary.
Overhead?

About 16% lower performance on average (phoronix.com)

Michael Schwarz (@misc0110), Moritz Lipp (@mlqxyz) — Graz University of Technology
operation \#n
Meltdown Root Cause

operation \#n

data

data dependency

operation \#n+2

time

Michael Schwarz (@misc0110), Moritz Lipp (@mlqxyz) — Graz University of Technology
Meltdown Root Cause

operation \#n

exception

data

data dependency

operation \#n+2

time

possibly architectural

transient execution

Michael Schwarz (@misc0110), Moritz Lipp (@mlqxyz) — Graz University of Technology
Meltdown Root Cause

operation \#n → retire

data

exception

data dependency

operation \#n+2

possibly architectural

transient execution

time

Michael Schwarz (@misc0110), Moritz Lipp (@mlqxyz) — Graz University of Technology
Meltdown Root Cause

operation \#n

retire

exception

data

Meltdown

data dependency

operation \#n+2

possibly architectural

transient execution

time
Meltdown Root Cause

operation \#n

- retire

exception

raise

Meltdown

data

- retire

data dependency

- possibly architectural

- transient execution

operation \#n+2

time

Michael Schwarz (@misc0110), Moritz Lipp (@mlqxyz) — Graz University of Technology
Meltdown Tree

Transient cause

Meltdown-type

Meltdown-NM-REG
Meltdown-PF
Meltdown-BR
Meltdown-GP
Meltdown-MCA

Meltdown-US
Meltdown-US-L1
Meltdown-US-LFB
Meltdown-US-SB

Meltdown-P
Meltdown-P-L1
Meltdown-P-LFB
Meltdown-P-SB
Meltdown-P-LP

Meltdown-RW
Meltdown-PK-L1
Meltdown-SM-SB

Meltdown-NC-SB
Meltdown-CPL-REG
Meltdown-NC-SB

Meltdown-AD
Meltdown-AVX-LP

Meltdown-AD-LFB
Meltdown-AD-SB

Michael Schwarz (@misc0110), Moritz Lipp (@mlqxyz) — Graz University of Technology
• Meltdown is not a fully solved issue
Meltdown is not a fully solved issue

The tree is extensible
• Meltdown is not a fully solved issue
• The tree is extensible
• More Meltdown-type issues to come
Meltdown Outlook

- Meltdown is not a fully *solved* issue
- The tree is extensible
- *More* Meltdown-type *issues* to come
- Silicon fixes might not be complete
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
Speculative Execution

- 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
• 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
Spectre-PHT (aka Spectre Variant 1)

index = 0

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

Shared Memory

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

\[
\begin{array}{cccc}
  D & & & \\
  & A & T & A \\
  & & & E \\
  & & & F \\
  & & & G \\
  & & & H \\
  & & & I \\
  & & & J \\
\end{array}
\]

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

index = 0

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

Michael Schwarz (@misc0110), Moritz Lipp (@mlqxyz) — Graz University of Technology
**Spectre-PHT (aka Spectre Variant 1)**

```plaintext
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**
  - data[0]
  - data[1]
  - data[2]
  - data[3]
index = 0

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

Michael Schwarz (@misc0110), Moritz Lipp (@mlqxyz) — Graz University of Technology
index = 0

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

Michael Schwarz (@misc0110), Moritz Lipp (@mlqxyz) — Graz University of Technology
index = 1

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

Michael Schwarz (@misc0110), Moritz Lipp (@mlqxyz) — Graz University of Technology
index = 1

if (index < 4) then

glyph[data[index]]

else

{ }

D
DATA
TAKE
KEY...

Michael Schwarz (@misc0110), Moritz Lipp (@mlqxyz) — Graz University of Technology
index = 1

if (index < 4)

then

glyph[data[index]]

else

{}
index = 1

if (index < 4)

then

glyph[data[index]]

else

{}
index = 1

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

else

{ }

Michael Schwarz (@misc0110), Moritz Lipp (@mlqxyz) — Graz University of Technology
index = 1

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

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

index = 2

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


index = 2

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

Michael Schwarz (@misc0110), Moritz Lipp (@mlqxyz) — Graz University of Technology
index = 2

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

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

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

index = 2

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

T

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[0]
E data[1]
F data[2]
G data[3]
index = 2

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

{ }

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

```
index = 3

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

index = 3

Memory

Shared Memory

if (index < 4)

Speculate

glyph[data[index]]

else

{}

Memory

data[0]
data[1]
data[2]
data[3]
index = 3

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

Michael Schwarz (@misc0110), Moritz Lipp (@mlqxyz) — Graz University of Technology
index = 3

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

Michael Schwarz (@misc0110), Moritz Lipp (@mlqxyz) — Graz University of Technology
index = 3

if (index < 4)

then

glyph[data[index]]

else

{}
Spectre-PHT (aka Spectre Variant 1)

```
index = 4

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

Shared Memory

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

Memory

```
data[0]
data[1]
data[2]
data[3]
```
index = 4

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

Michael Schwarz (@misc0110), Moritz Lipp (@mlqxyz) — Graz University of Technology
index = 4

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

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

K

Michael Schwarz (@misc0110), Moritz Lipp (@mlqxyz) — Graz University of Technology
index = 4

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

Michael Schwarz (@misc0110), Moritz Lipp (@mlqxyz) — Graz University of Technology
index = 4

if (index < 4)

then

glyph[data[index]]

else

{}
operation \#n
operation \#n

prediction

time
Spectre Root Cause

operation #n

prediction

operation #n+2

predict CF/DF

time

Michael Schwarz (@misc0110), Moritz Lipp (@mlqxyz) — Graz University of Technology
Spectre Root Cause

operation \#n

decision

cf/df

possibly architectural

operation \#n+2

transient execution

time

Michael Schwarz (@misc0110), Moritz Lipp (@mlqxyz) — Graz University of Technology
Spectre Root Cause

operation #n

retire

prediction

predict CF/DF

possibly architectural

transient execution

operation #n+2

time

Michael Schwarz (@misc0110), Moritz Lipp (@mlqxyz) — Graz University of Technology
Spectre Root Cause

- Operation \#n
- Prediction
- Operation \#n+2
- Transient execution

Flux pipeline on wrong prediction
Spectre Root Cause

operation #n

flush pipeline on wrong prediction

prediction

retire

operation #n+2

transient execution

possibly architectural

predict CF/DF

retire

time

Michael Schwarz (@misc0110), Moritz Lipp (@mlqxyz) — 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)
Spectre Root Cause

- 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
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

Victim

same address space/
out of place

Congruent
branch

Address
collision

Victim
branch

same address space/
in place

Shared Branch Prediction State

Michael Schwarz (@misc0110), Moritz Lipp (@mlqxyz) — Graz University of Technology
Spectre Mistraining

Victim

Congruent branch

Address collision

Victim branch

Shared Branch Prediction State

Attacker

same address space/
out of place

same address space/
in place

Michael Schwarz (@misc0110), Moritz Lipp (@mlqxyz) — Graz University of Technology
Spectre Mistraining

Victim

same address space/out of place

Congruent branch

Attacker

same address space/in place

Victim branch

cross address space/in place

Address collision

Shared Branch Prediction State

Michael Schwarz (@misc0110), Moritz Lipp (@mlqxyz) — Graz University of Technology
Spectre Mistraining

same address space/
out of place

same address space/
in place

Victim
Congruent branch
Address collision
Victim branch

Attacker
Congruent branch
Address collision
Shadow branch

Shared Branch Prediction State

Michael Schwarz (@misc0110), Moritz Lipp (@mlqxyz) — Graz University of Technology
Spectre Variants

Transient cause?

Spectre-type

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

microarchitectural buffer

prediction
Spectre Variants

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

Transient cause?

Microarchitectural buffer

Mistraining strategy

Cross-address-space
- Same-address-space

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

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

- 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

prediction

microarchitectural buffer

Transient cause?

in-place (IP) vs., out-of-place (OP)
• Spectre is not a bug
• 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 is not a bug
• It is an useful optimization
→ Cannot simply fix it (as with Meltdown)
• Workarounds for critical code parts
Spectre Defense Categorization

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>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>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>Spectre-BTB</td>
<td>Spectre-RSB</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>
</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>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>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>
</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>
</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>
</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>
</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>STPB</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 (○).
## 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>InvisiSpec</th>
<th>SafeSpec</th>
<th>DAWG</th>
<th>RSB</th>
<th>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</th>
<th>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>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 (◇).
<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>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 (◇).
## 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 (◇).
• 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).
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).
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).
• 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

- Spectre-type
  - Spectre-PHT
    - Cross-address-space
      - PHT-CA-IP
      - PHT-CA-OP
    - Same-address-space
      - PHT-SA-IP
      - PHT-SA-OP
  - Spectre-BTB
    - Cross-address-space
      - BTB-CA-IP
      - BTB-CA-OP
    - Same-address-space
      - BTB-SA-IP
      - BTB-SA-OP
  - Spectre-RSB
    - Cross-address-space
      - RSB-CA-IP
      - RSB-CA-OP
    - Same-address-space
      - RSB-SA-IP
      - RSB-SA-OP
  - Spectre-STL

- Meltdown-type
  - Meltdown-NS-REG
  - Meltdown-PT
  - Meltdown-NP
  - Meltdown-NR
  - Meltdown-PKL-1
  - Meltdown-SM-SB
  - Meltdown-MPX
  - Meltdown-BND
  - Meltdown-NC-SB
  - Meltdown-AD-LFB
  - Meltdown-AD-SB

Transit causes:
- Cross-address-space
- Same-address-space

Michael Schwarz (@misc0110), Moritz Lipp (@mlqxyz) — Graz University of Technology
• Transient Execution Attacks are...
Transient Execution Attacks are...

- ...a novel class of 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

- 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
Transient Execution Attacks

Exploiting the CPU’s Microarchitecture

Michael Schwarz (@misc0110), Moritz Lipp (@mlqxyz)

October 7, 2019

Graz University of Technology