ConTExT

A Generic Approach for Mitigating Spectre

Michael Schwarz, Moritz Lipp, Claudio Canella, Robert Schilling, Florian Kargl, Daniel Gruss

February 26, 2020

Graz University of Technology
Transient Execution Attacks

Michael Schwarz (@misc0110) et al. — Graz University of Technology
Transient Execution Attacks

Michael Schwarz (@misc0110) et al. — Graz University of Technology
Transient Execution Attacks
Transient Execution Attacks

- Transient cause
  - Meltdown-type
    - Meltdown-NM-REG
    - Meltdown-AC
    - Meltdown-DE
    - Meltdown-PF
    - Meltdown-UD
    - Meltdown-SS
    - Meltdown-BR
    - Meltdown-GP
    - Meltdown-MCA
  - Spectre-type
    - Spectre-PHT
    - Spectre-BTB
    - Spectre-RSB
    - Spectre-CX
  - 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
- Cross-address-space
- Same-address-space

Michael Schwarz (@misc0110) et al. — Graz University of Technology
Transient Execution Attacks

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

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

Cross-address-space
- PHT-CA-IP
- PHT-CA-OP
- BTB-CA-IP
- BTB-CA-OP
- RSB-CA-IP
- RSB-CA-OP

Same-address-space
- PHT-SA-IP
- PHT-SA-OP
- BTB-SA-IP
- BTB-SA-OP
- RSB-SA-IP
- RSB-SA-OP

Meltdown-AC-LFB
Meltdown-AC-LP
Meltdown-US
Meltdown-P
Meltdown-RW
Meltdown-PK
Meltdown-SM-SB
Meltdown-MPX
Meltdown-BND
Meltdown-CPL-REG
Meltdown-NC-SB
Meltdown-AVX
Meltdown-AD
Meltdown-TAA
Meltdown-PRM-LFB
Meltdown-UC-LFB

Michael Schwarz (@misc0110) et al. — Graz University of Technology
operation #n
operation #n

prediction

time
Spectre Root Cause

operation \#n

prediction

predict CF/DF

operation \#n+2

time
Spectre Root Cause

operation \#n

prediction

operation \#n+2

possibly architectural transient execution

time

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

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

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

operation \#n

prediction

operation \#n+2

transient execution

flush pipeline on wrong prediction

time

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

operation #n

flush pipeline on wrong prediction

prediction

retire

predict CF/DF

operation #n+2

transient execution

possibly architectural

time

Michael Schwarz (@misc0110) et al. — Graz University of Technology
It's not a bug.
It's a feature.
if (x < array_len) {
    y = oracle[array[x] * 4096];
}
if (x < 4)

then

oracle[array[x]]

else

{ }

x = 4
Unprotected Execution

Unprotected

```assembly
cmp rdi, .array_len
jbe .else

mov (rax + rdi),al
shl 12,rax
and 0xff000,eax
mov (rdx + rax),al
mov 0,rax
retq
mov rax,(rsp + 8)
```
Unprotected Execution

Unprotected

```assembly
cmp rdi, .array_len
jbe .else

mov (rax + rdi),al
shl 12,rax
and 0xff000,eax
mov (rdx + rax),al
mov 0,rax
retq

mov rax,(rsp + 8)
```

Bounds check
Unprotected Execution

Unprotected

```assembly
cmp rdi, .array_len
jbe .else

mov (rax + rdi), al
shl 12, rax
and 0xff000, eax

mov (rdx + rax), al
mov 0, rax
retq

mov rax, (rsp + 8)
```

Bounds check

Access out-of-bounds array[x]
### Unprotected Execution

**Unprotected**

```
cmp rdi, .array_len
jbe .else

mov (rax + rdi),al
shl 12,rax
and 0xff000,eax

mov (rdx + rax),al
mov 0,rax
retq
mov rax,(rsp + 8)
```

- **Bounds check**
  - `cmp rdi, .array_len`
  - `jbe .else`

- **Access out-of-bounds array[x]**
  - `mov (rax + rdi),al`
  - `shl 12,rax`
  - `and 0xff000,eax`

- **Secret in rax**
  - `mov (rdx + rax),al`
  - `mov 0,rax`
  - `retq`
  - `mov rax,(rsp + 8)`
Unprotected Execution

Unprotected

cmp rdi, .array_len
jbe .else

mov (rax + rdi), al
shl 12, rax
and 0xff000, eax
mov (rdx + rax), al
mov 0, rax
retq
mov rax, (rsp + 8)

Bounds check
Access out-of-bounds array[x]
Secret in rax
Access secret-dependent memory location
if (x < array_len) {
    asm volatile("lfence");
    y = oracle[array[x] * 4096];
}
Serializing Barrier

```
cmp rdi, .array_len
jbe .else
lfence
mov (rax + rdi), al
shl 12, rax
and 0xff000, eax
mov (rdx + rax), al
mov 0, rax
retq
mov rax, (rsp + 8)
```
Memory Barriers

Serializing Barrier

```
cmp rdi, .array_len
jbe .else
lfence
mov (rax + rdi),al
shl 12,rax
and 0xff000,eax
mov (rdx + rax),al
mov 0,rax
retq
mov rax,(rsp + 8)
```

1. **Bounds check**
2. **Stop speculation**

---

Michael Schwarz (@misc0110) et al. — Graz University of Technology
Memory Barriers

Serializing Barrier

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cmp rdi, .array_len</td>
<td>Steps to check the bounds</td>
</tr>
<tr>
<td>jbe .else</td>
<td>Bound check</td>
</tr>
<tr>
<td>lfence</td>
<td>Stop speculation</td>
</tr>
<tr>
<td>mov (rax + rdi), al</td>
<td>Cannot access out-of-bounds array[x]</td>
</tr>
<tr>
<td>shl 12,rax</td>
<td>Bound check</td>
</tr>
<tr>
<td>and 0x0ff000,eax</td>
<td>Bound check</td>
</tr>
<tr>
<td>mov (rdx + rax), al</td>
<td>Bound check</td>
</tr>
<tr>
<td>mov 0,rax</td>
<td>Bound check</td>
</tr>
<tr>
<td>retq</td>
<td>Bound check</td>
</tr>
</tbody>
</table>

1. not executed

Michael Schwarz (@misc0110) et al. — Graz University of Technology
Performance Impact

- 62% – 74.8% overhead
• 62% – 74.8% overhead
• Additional overhead for other Spectre variants 5% – 50%
• 62% – 74.8% overhead
• Additional overhead for other Spectre variants 5% – 50%
• Identify leaking branches → difficult
From identifying branches...
From identifying branches...
From identifying branches... ...to identifying secrets
Annotated secrets...
Annotated secrets...
Annotated secrets... 

...stored in non-speculatable memory
Secrets can transiently enter registers...
Secrets can transiently enter registers... 

...but not transiently leave them
char nospec array[16];

if(x < array_len) {
    y = oracle[array[x] * 4096];
}

cmp rdi, .array_len
jbe .else

mov (rax + rdi),al
shl 12,rax
and 0xff000,eax

mov (rdx + rax),al
mov 0,rax
retq

mov rax,(rsp + 8)
```assembly
 cmp rdi, .array_len
 jbe .else

 mov (rax + rdi),al
 shl 12,rax
 and 0xff000,eax
 mov (rdx + rax),al

 mov 0,rax
 retq

 mov rax,(rsp + 8)
```

**Bounds check**
cmp rdi, .array_len
jbe .else

mov (rax + rdi),al
shl 12,rax
and 0xffffffff,fax

mov (rdx + rax),al
mov 0,rax
retq

mov rax,(rsp + 8)

Bounds check
Access out-of-bounds array[x]
```assembly
cmp rdi, .array_len
jbe .else

mov (rax + rdi),al
shl 12,rax
and 0xff000,eax
mov (rdx + rax),al
mov 0,rax
retq
mov rax,(rsp + 8)
```

- **Bounds check**
- **Access out-of-bounds array[x]**
- **Secret in rax, no operations on secret**
ConTExT

```
cmp rdi, .array_len
jbe .else

mov (rax + rdi),al
shl 12,rax
and 0xff000,eax
mov (rdx + rax),al

mov 0,rax
retq

mov rax,(rsp + 8)
```

- **Bounds check**
- **Access out-of-bounds array[x]**
- **Secret in rax, no operations on secret**
- **Independent operations continue out-of-order**
New Memory Type
ConTExT Requirements

New Memory Type

Simple Taint Tracking
ConTExT Requirements

- New Memory Type
- Simple Taint Tracking
- Compiler Support
ConTEXT Requirements

New Memory Type

Compiler Support

Simple Taint Tracking

OS Support
New Memory Type

Simple Taint Tracking

Compiler Support

OS Support

Michael Schwarz (@misc0110) et al. — Graz University of Technology
Uncachable Memory

Simple Taint Tracking

Compiler Support

OS Support
Uncachable Memory

No Taint Tracking

Compiler Support

OS Support
Uncachable Memory  
LLVM Support  
No Taint Tracking  
OS Support
Uncachable Memory

No Taint Tracking

LLVM Support

Linux Kernel Module
```
cmp rdi, .array_len
jbe .else
mov (rax + rdi),al
shl 12,rax
and 0xff000,eax
mov (rdx + rax),al
mov 0,rax
retq
mov rax,(rsp + 8)
```
ConTExT-light

```
not executed

cmp rdi, .array_len
jbe .else

mov (rax + rdi), al
shl 12, rax
and 0xff000, eax
mov (rdx + rax), al
mov 0, rax
retq
mov rax, (rsp + 8)

Bounds check
```
ConTExT-light

```
 cmp rdi, .array_len
  jbe .else

 mov (rax + rdi), al
  shl 12, rax
  and 0xff000, eax

 mov (rdx + rax), al
  mov 0, rax
  retq
  mov rax, (rsp + 8)

 Bounds check
 Out-of-bounds access array[x] stalls
```

Michael Schwarz (@misc0110) et al. — Graz University of Technology
ConTExT-light

```
cmp rdi, .array_len
jbe .else

mov (rax + rdi),al
shl 12,rax
and 0xff000,eax

mov (rdx + rax),al
mov 0,rax
retq

mov rax,(rsp + 8)
```

- Bounds check
- Out-of-bounds access array[x] stalls
- Dependent operations stall

Michael Schwarz (@misc0110) et al. — Graz University of Technology
cmp rdi, .array_len
jbe .else

mov (rax + rdi),al
shl 12,rax
and 0xff000,eax
mov (rdx + rax),al
mov 0,rax
retq

mov rax,(rsp + 8)

Bounds check
Out-of-bounds access array[x] stalls
Dependent operations stall
Independent operations continue out-of-order

Bounds check
Out-of-bounds access array[x] stalls
Dependent operations stall
Independent operations continue out-of-order

Not executed
char oracle[256 * 4096];
// nospec for ConTExT-light
char /*nospec*/ secret = 'X';

if(speculate()) {
    // LFENCE here for mitigation
    oracle[secret * 4096]; // encode secret
    oracle['E' * 4096]; // encode public value
}
```c
char oracle[256 * 4096];
// nospec for ConTExT-light
char /*nospec*/ secret = 'X';

if(speculate()) {
    asm volatile("lfence");
    oracle[secret * 4096]; // encode secret
    oracle['E' * 4096]; // encode public value
}
```

---

No secret 0x45

Secret 0x58
```c
char oracle[256 * 4096];  
// nospec for ConTExT-light  
char nospec secret = 'X'; 

if (speculate()) {  
    // LFENCE here for mitigation  
    oracle[secret * 4096];  // encode secret  
    oracle['E' * 4096];  // encode public value 
}
```

The graph shows latency in cycles for different protection methods:
- **unprotected**
- **lfence**
- **ConTExT**

The y-axis represents latency in cycles, and the x-axis represents the page of oracle. The graph indicates that the ConTExT method shows a significant improvement in latency compared to the unprotected and lfence methods, especially for pages containing a secret.
Performance

AES-NI

0%
Performance

- **AES-NI**: 0%

- **VeraCrypt**
  - Mount: 3.21%
  - Encrypt: 0%

---

Michael Schwarz (@misc0110) et al. — Graz University of Technology
### Performance

- **AES-NI**: 0%
- **VeraCrypt**
  - Mount: 3.21% / Encrypt: 0%
- **OpenSSH**
  - Init: 24.7% / Transfer: 5.4%
Performance

- **AES-NI**: 0%
- **VeraCrypt**: 3.21% (mount) / 0% (encrypt)
- **OpenSSH**: 24.7% (init) / 5.4% (transfer)
- **NGINX**: 7.3%
You can find our proof-of-concept implementation on:

- https://github.com/IAIK/contextlight
“A New Memory Type against Speculative Side Channel Attacks” [SBH19]
A New Memory Type

“A New Memory Type against Speculative Side Channel Attacks” [SBH19]

“Memory Type Which is Cacheable Yet Inaccessible by Speculative Instructions” [Bog+19]
More details in the paper [Sch+20]

- Compiler modifications
- Taint tracking in register/cache/TLB
- Handling context switches
- ...
• ConTExT is data-based instead of instruction-based
Conclusion

• ConTExT is data-based instead of instruction-based
• Mitigates the root cause (leakage) instead of the covert channel
Conclusion

- ConTExT is data-based instead of instruction-based
- Mitigates the root cause (leakage) instead of the covert channel
- Applicable to all Spectre variants with low overhead
Conclusion

- ConTExT is data-based instead of instruction-based
- Mitigates the root cause (leakage) instead of the covert channel
- Applicable to all Spectre variants with low overhead
- All changes are fully backward compatible
A Generic Approach for Mitigating Spectre

Michael Schwarz, Moritz Lipp, Claudio Canella, Robert Schilling, Florian Kargl, Daniel Gruss
February 26, 2020

Graz University of Technology
We thank our anonymous reviewers for their comments and suggestions that helped improving the paper. The project was supported by the European Research Council (ERC) under the European Union’s Horizon 2020 research and innovation programme (grant agreement No 681402). It was also supported by the Austrian Research Promotion Agency (FFG) via the K-project DeSSnet, which is funded in the context of COMET - Competence Centers for Excellent Technologies by BMVIT, BMWFW, Styria and Carinthia. This work has additionally been supported by the Austrian Research Promotion Agency (FFG) via the project ESPRESSO, which is funded by the Province of Styria and the Business Promotion Agencies of Styria and Carinthia. This work has also been supported by the Austrian Research Promotion Agency (FFG) via the competence center Know-Center (grant number 844595), which is funded in the context of COMET – Competence Centers for Excellent Technologies by BMVIT, BMWFW, and Styria. Additional funding was provided by generous gifts from ARM and Intel. Any opinions, findings, and conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of the funding parties.
