# ConTExT

## A Generic Approach for Mitigating Spectre

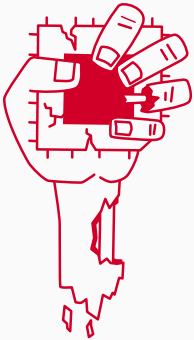
**Michael Schwarz, Moritz Lipp, Claudio Canella, Robert Schilling, Florian Kargl, Daniel Gruss**

February 26, 2020

Graz University of Technology


MD
S

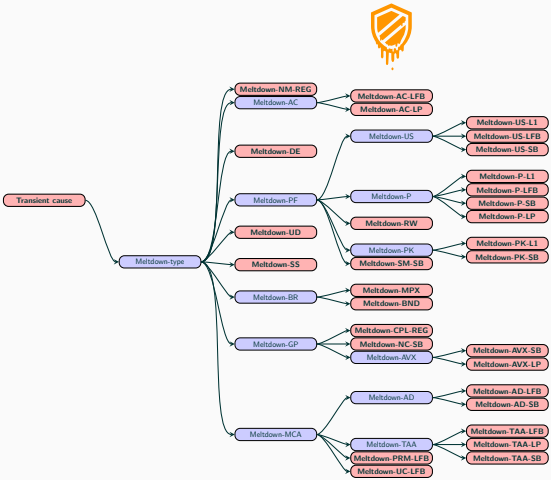

MD
S

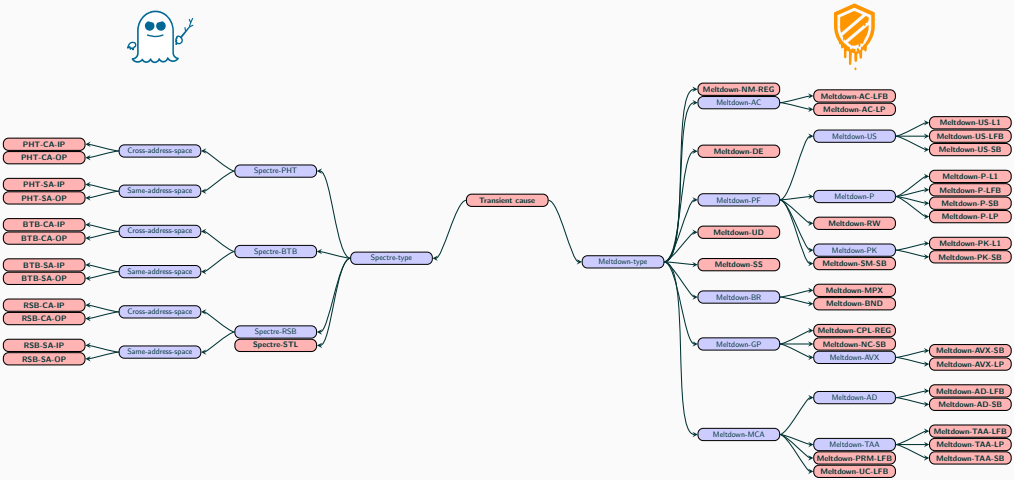


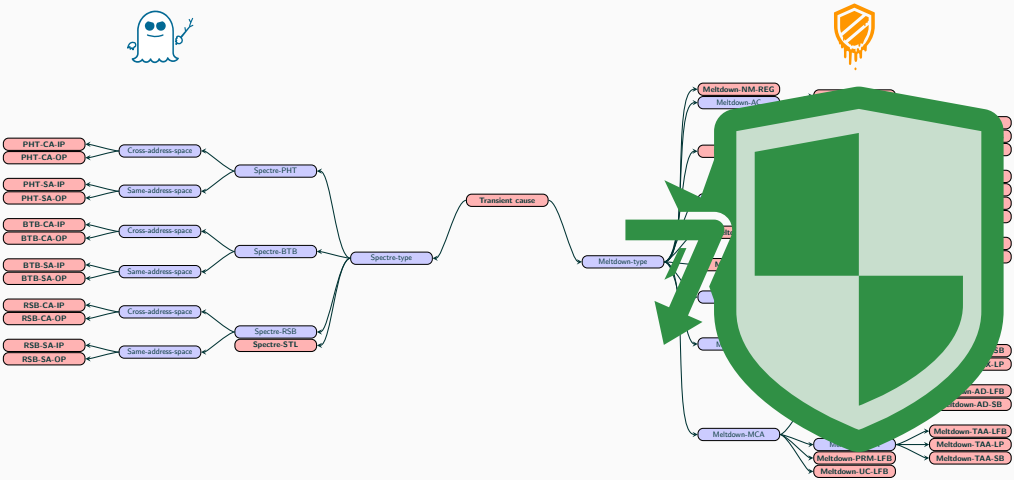

MD
S

# Transient Execution Attacks

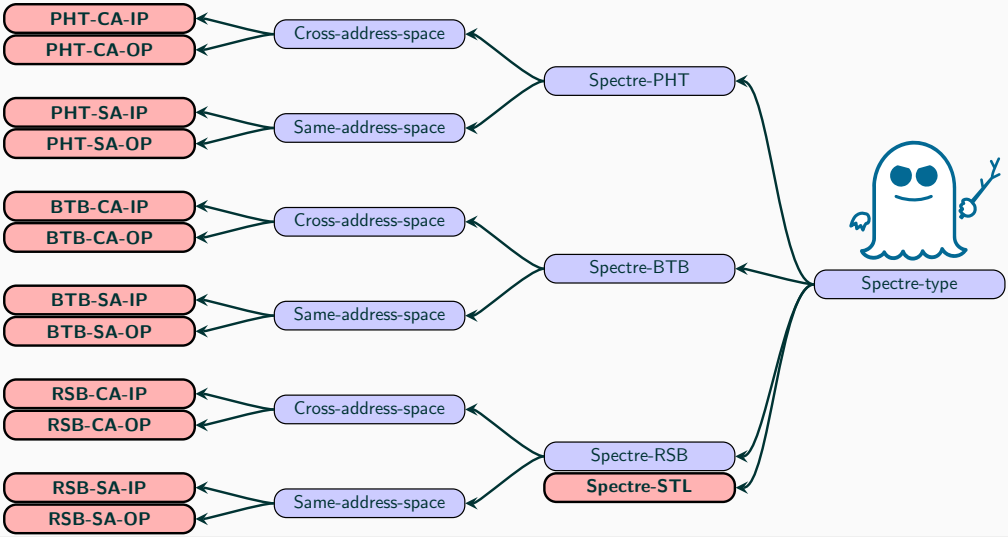
# Transient Execution Attacks

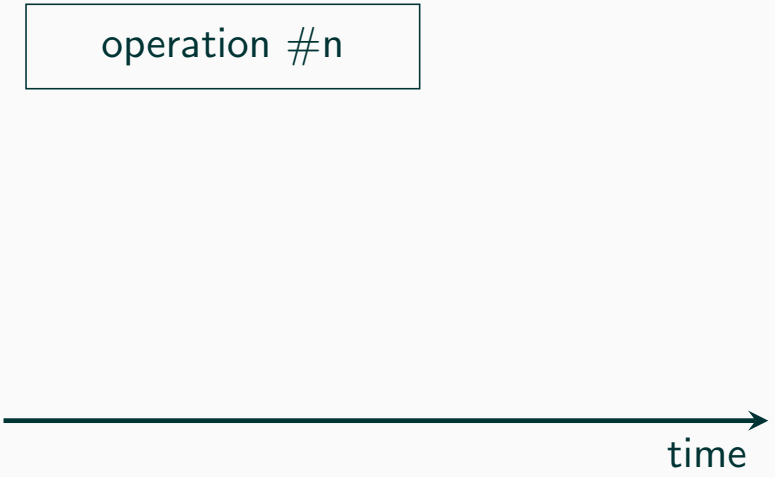


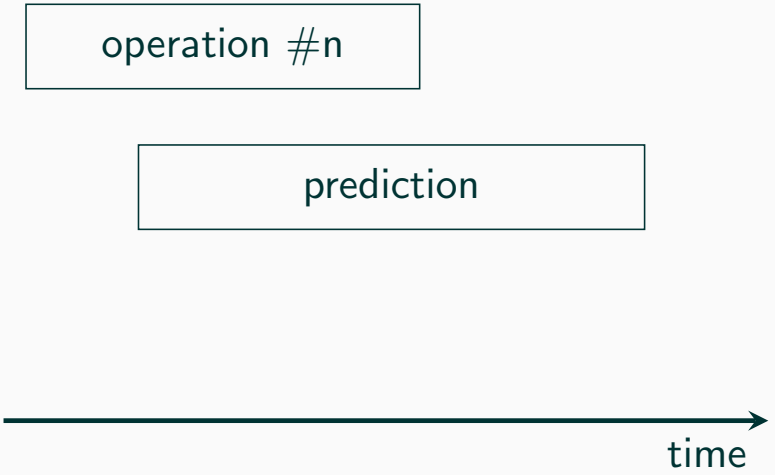

PHT-CA-IP
PHT-CA-OP
PHT-SA-IP
PHT-SA-OP
BTB-CA-IP
BTB-CA-OP
BTB-SA-IP
BTB-SA-OP
RSB-CA-IP
RSB-CA-OP
RSB-SA-IP
RSB-SA-OP
Cross-address-space
Same-address-space
Cross-address-space
Same-address-space
Cross-address-space
Same-address-space
Spectre-PHT
Spectre-BTB
Spectre-RSB
Spectre-STL
Spectre-type
Transient cause
Meltdown-type
Meltdown-NM-REG
Meltdown-AC
Meltdown-AC-LFB
Meltdown-AC-LP
Meltdown-DE
Meltdown-PF
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-UD
Meltdown-PK
Meltdown-PK-L1
Meltdown-PK-SB
Meltdown-SS
Meltdown-BR
Meltdown-MPX
Meltdown-BND
Meltdown-GP
Meltdown-CPL-REG
Meltdown-NC-SB
Meltdown-AVX
Meltdown-AVX-SB
Meltdown-AVX-LP
Meltdown-MCA
Meltdown-AD
Meltdown-AD-LFB
Meltdown-AD-SB
Meltdown-TAA
Meltdown-TAA-LFB
Meltdown-TAA-LP
Meltdown-TAA-SB
Meltdown-PRM-LFB
Meltdown-UC-LFB

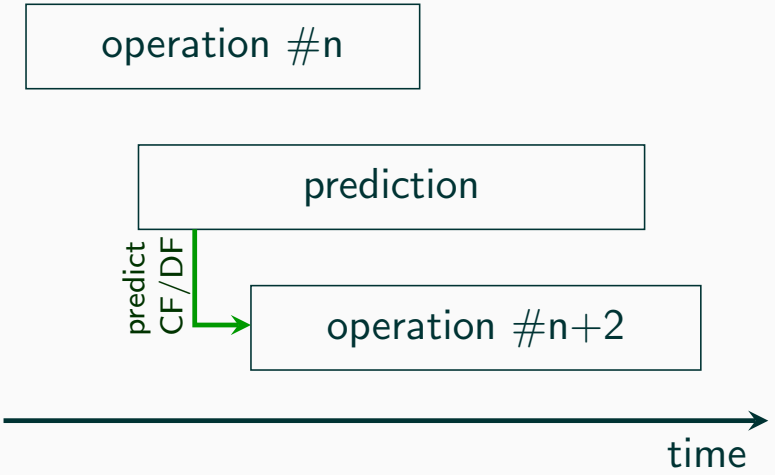

PHT-CA-IP
PHT-CA-OP
PHT-SA-IP
PHT-SA-OP
BTB-CA-IP
BTB-CA-OP
BTB-SA-IP
BTB-SA-OP
RSB-CA-IP
RSB-CA-OP
RSB-SA-IP
RSB-SA-OP
Cross-address-space
Same-address-space
Cross-address-space
Same-address-space
Cross-address-space
Same-address-space
Spectre-PHT
Spectre-BTB
Spectre-RSB
Spectre-STL
Spectre-type
Transient cause
Meltdown-type
Meltdown-NM-REG
Meltdown-AC
Meltdown-MCA
Meltdown-PRM-LFB
Meltdown-UC-LFB
Meltdown-TAA-LFB
Meltdown-TAA-LP
Meltdown-TAA-SB

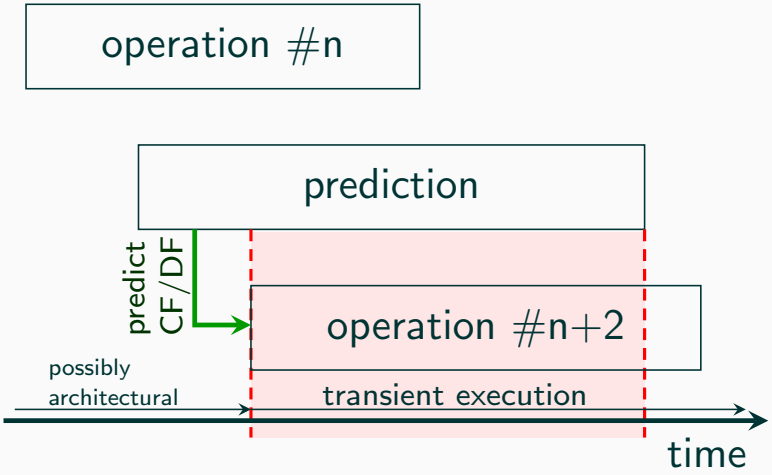
# Spectre Attacks

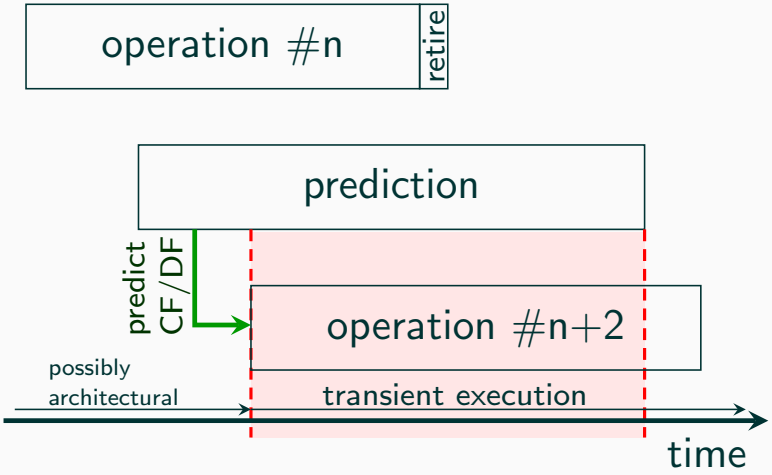


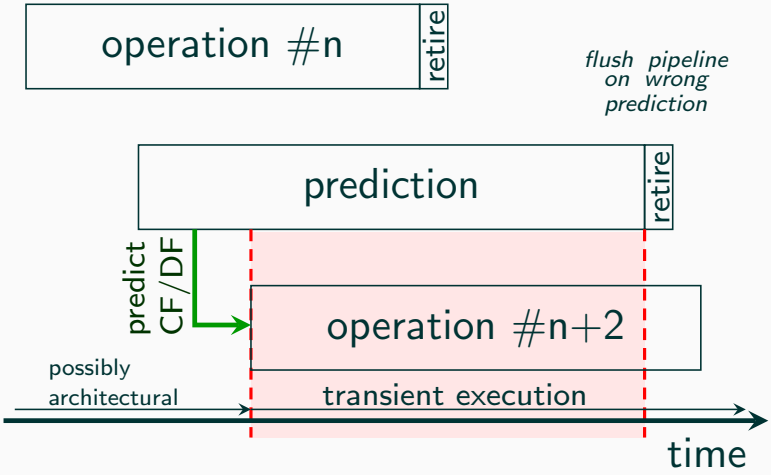

operation #n
time

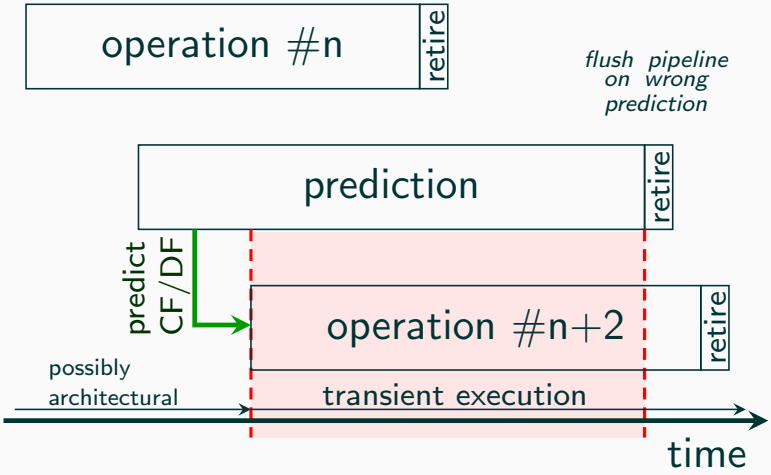

operation #n
prediction
time


operation #n
prediction
predict CF/DF
operation #n+2
time


operation #n
prediction
predict CF/DF
operation #n+2
possibly architectural
transient execution
time

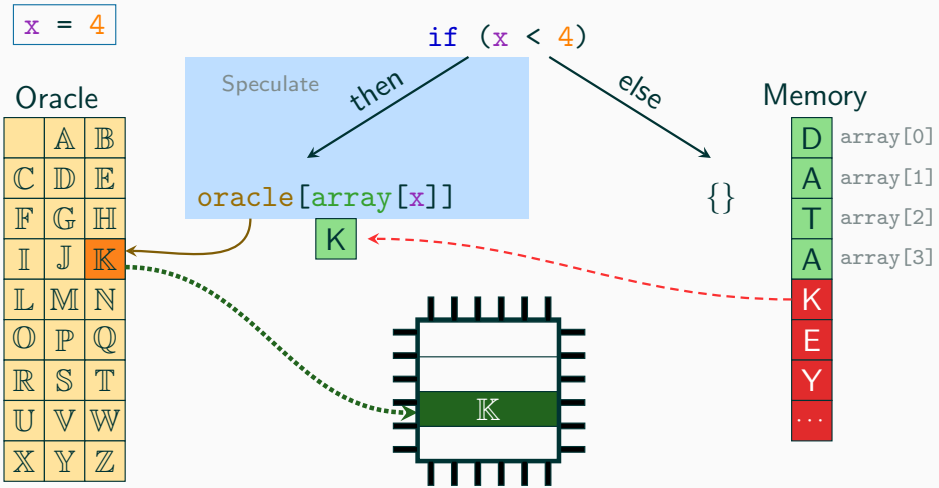

operation #n
retire
prediction
predict CF/DF
operation #n+2
possibly architectural
transient execution
time


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

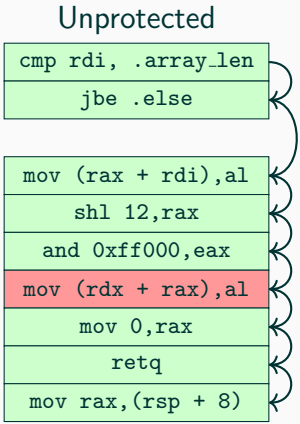

operation #n
retire
flush pipeline on wrong prediction
prediction
retire
predict CF/DF
operation #n+2
retire
possibly architectural
transient execution
time


IT'S NOT A
BUG
IT'S A
FEATURE

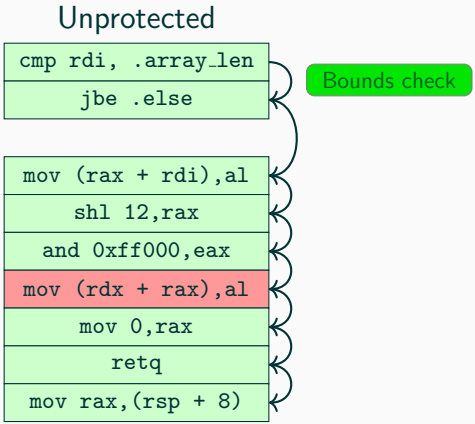
```
if(x < array_len) {
    y = oracle[array[x] * 4096];
}
```


x = 4
if (x < 4)
Speculate
then
else
Oracle
Memory
oracle[array[x]]
K
{}
array[0]
array[1]
array[2]
array[3]
D
A
T
A
K
E
Y
...
K

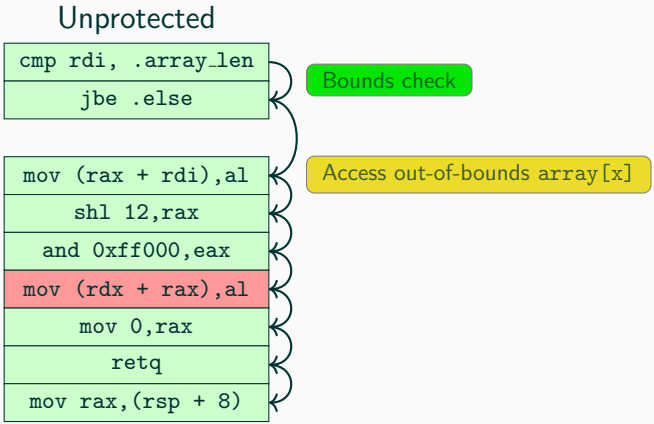
# Unprotected Execution



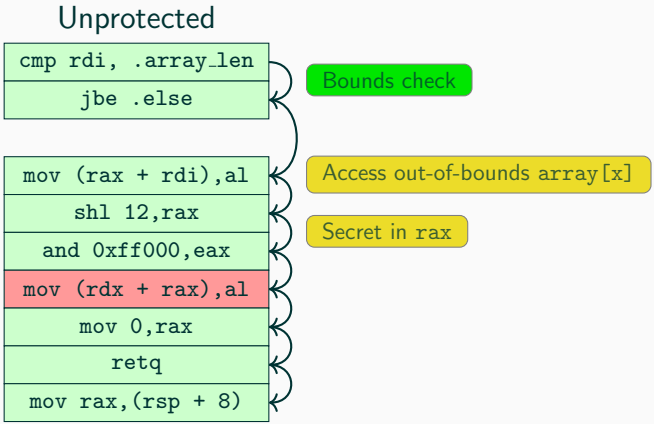
# Unprotected Execution



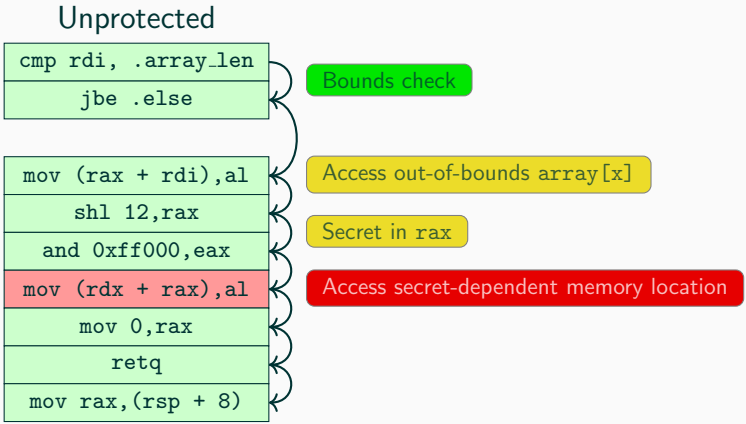
# Unprotected Execution



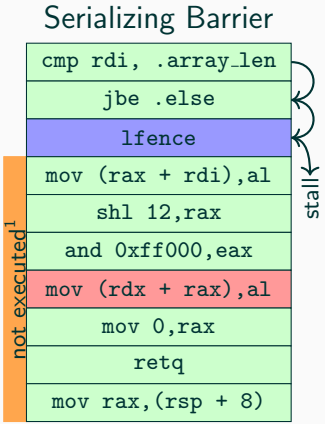
# Unprotected Execution

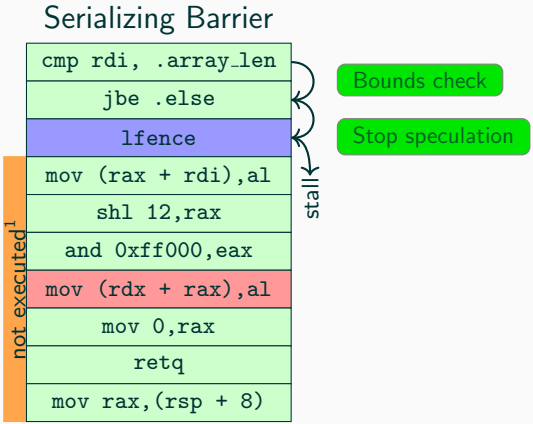


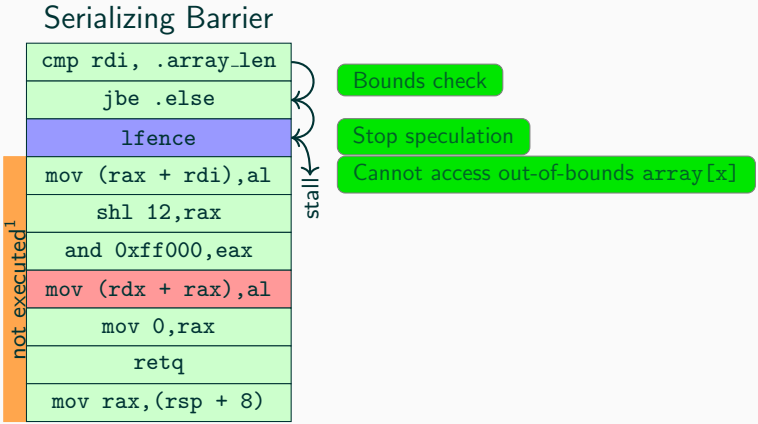
# Unprotected Execution



```
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)
stall
not executed[1]


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)
Bounds check
Stop speculation
stall
not executed[1]


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)
not executed[1]
stall
Bounds check
Stop speculation
Cannot access out-of-bounds array[x]

# Performance Impact

- 62 % – 74.8 % overhead

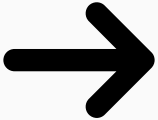
# Performance Impact

- 62 % – 74.8 % overhead
- Additional overhead for other Spectre variants 5 % – 50 %

# Performance Impact

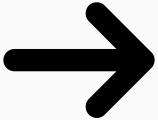
- 62 % – 74.8 % overhead
- Additional overhead for other Spectre variants 5 % – 50 %
- Identify leaking branches → difficult

From identifying branches...

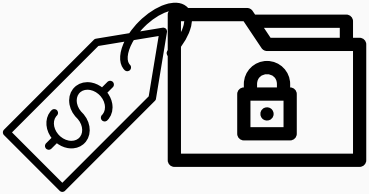


From identifying branches...

From identifying branches...

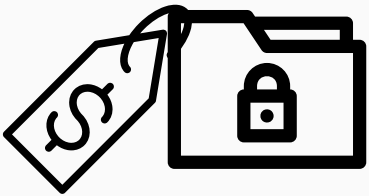


...to identifying secrets

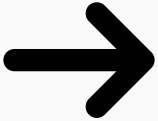


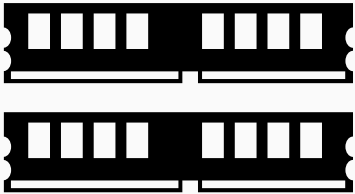
Annotated secrets...

Annotated secrets...

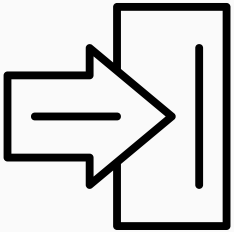


Annotated secrets...

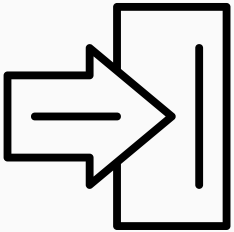




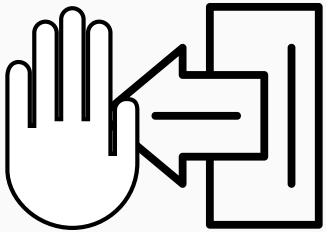
...stored in non-speculatable memory



Secrets can transiently enter registers...

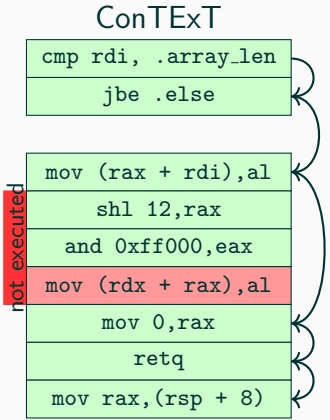


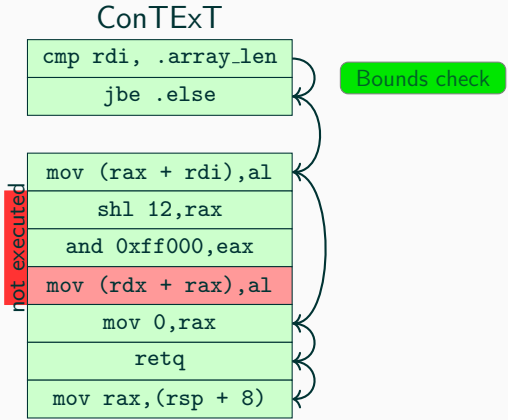
Secrets can transiently enter registers...

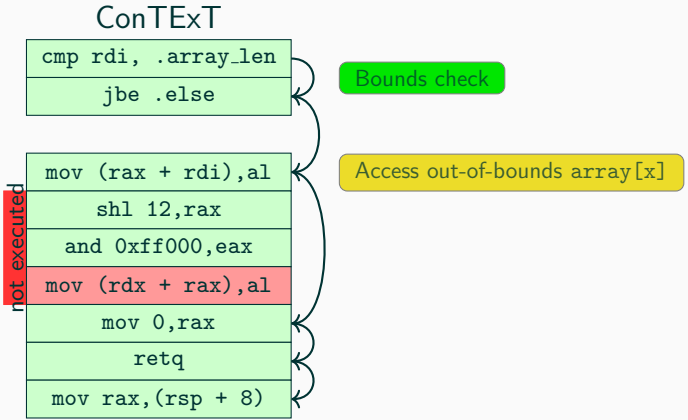


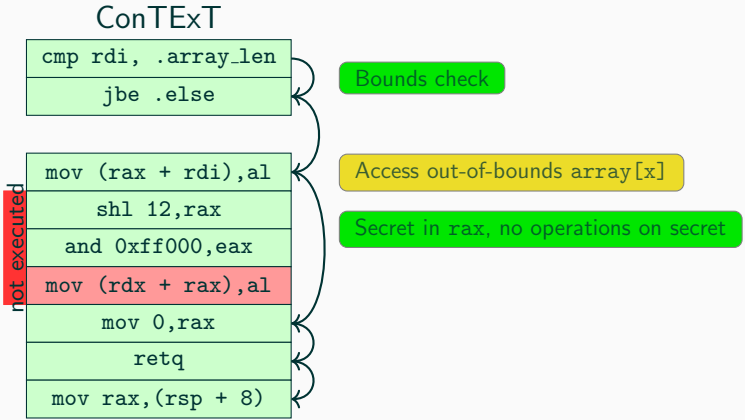
...but not transiently leave them

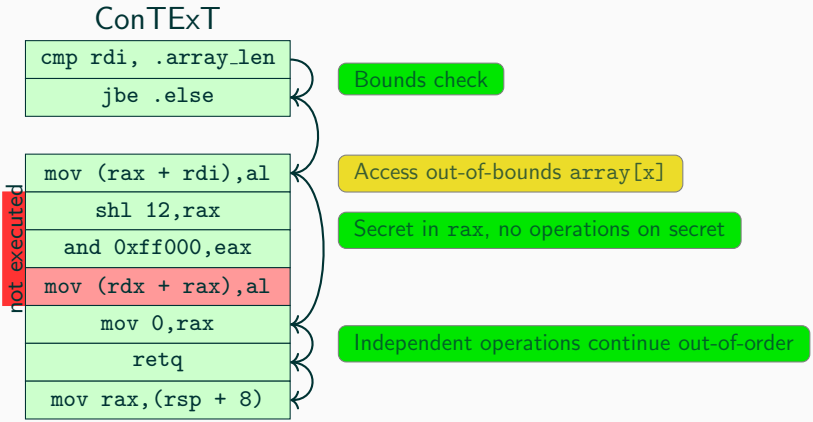
```
char nospec array[16];

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

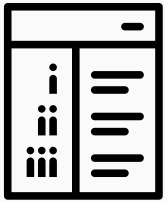

ConTExT
cmp rdi, .array_len
jbe .else
mov (rax + rdi),al
shl 12,rax
and 0xff000,eax
not executed
mov (rdx + rax),al
mov 0,rax
retq
mov rax,(rsp + 8)


ConTExT
cmp rdi, .array_len
jbe .else
Bounds check
mov (rax + rdi),al
shl 12,rax
and 0xff000,eax
not executed
mov (rdx + rax),al
mov 0,rax
retq
mov rax,(rsp + 8)


ConTExT
cmp rdi, .array_len
jbe .else
Bounds check
mov (rax + rdi),al
Access out-of-bounds array[x]
not executed
shl 12,rax
and 0xff000,eax
mov (rdx + rax),al
mov 0,rax
retq
mov rax,(rsp + 8)

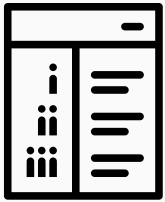

ConTExT
cmp rdi, .array_len
jbe .else
Bounds check
mov (rax + rdi),al
Access out-of-bounds array[x]
shl 12,rax
Secret in rax, no operations on secret
and 0xff000,eax
not executed
mov (rdx + rax),al
mov 0,rax
retq
mov rax,(rsp + 8)


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)
not executed
Bounds check
Access out-of-bounds array[x]
Secret in rax, no operations on secret
Independent operations continue out-of-order



New Memory Type

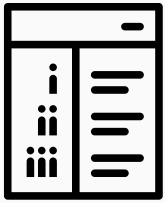
# ConTExT Requirements



New Memory Type

Simple Taint Tracking

# ConTExT Requirements

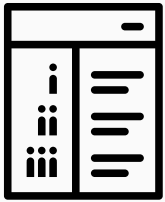


New Memory Type

Simple Taint Tracking

Compiler Support

# ConTExT Requirements

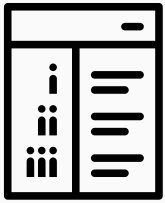


New Memory Type

Simple Taint Tracking

Compiler Support

OS Support

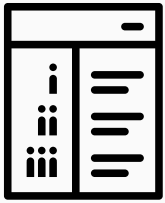


New Memory Type

Simple Taint Tracking

Compiler Support

OS Support

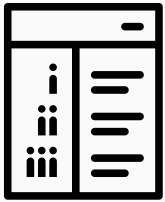


Uncachable Memory

Simple Taint Tracking

Compiler Support

OS Support

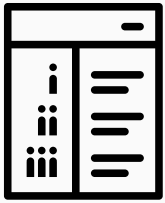


Uncachable Memory

No Taint Tracking

Compiler Support

OS Support

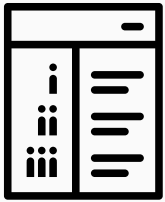


Uncachable Memory

No Taint Tracking

LLVM Support

OS Support

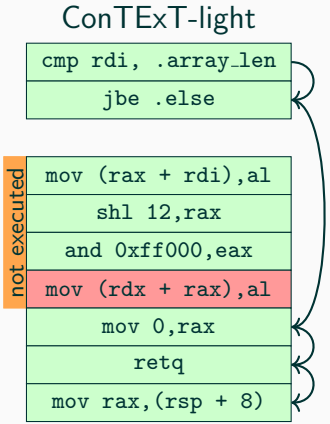


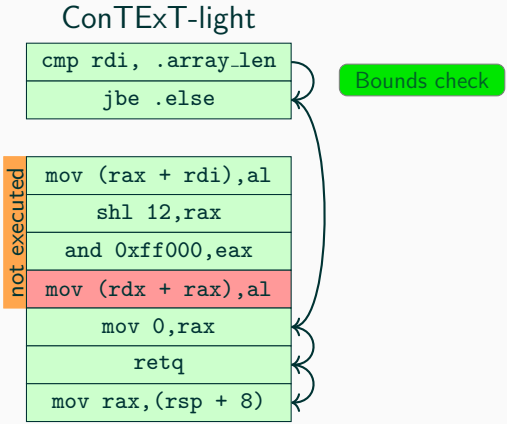
Uncachable Memory

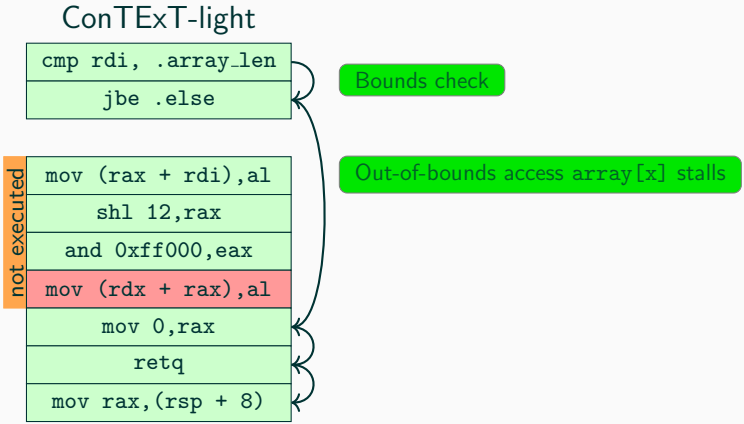
No Taint Tracking

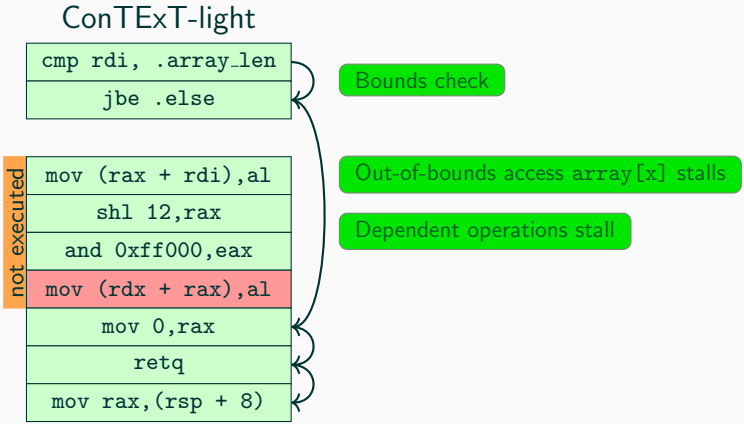
LLVM Support

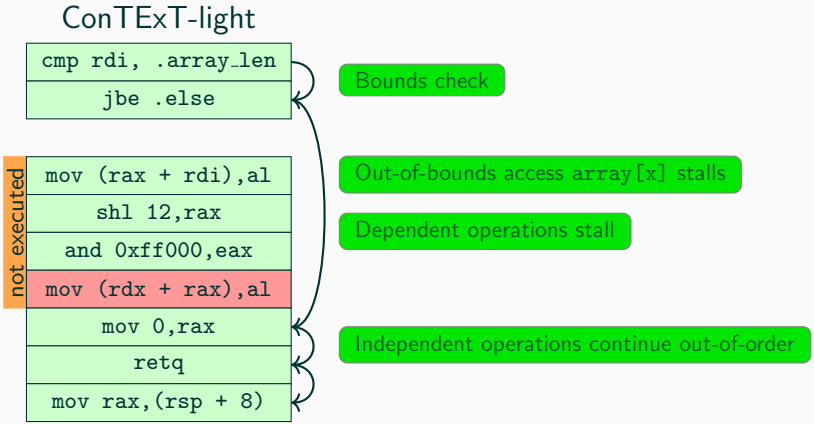
Linux Kernel Module


ConTExT-light
cmp rdi, .array_len
jbe .else
not executed
mov (rax + rdi),al
shl 12,rax
and 0xff000,eax
mov (rdx + rax),al
mov 0,rax
retq
mov rax,(rsp + 8)

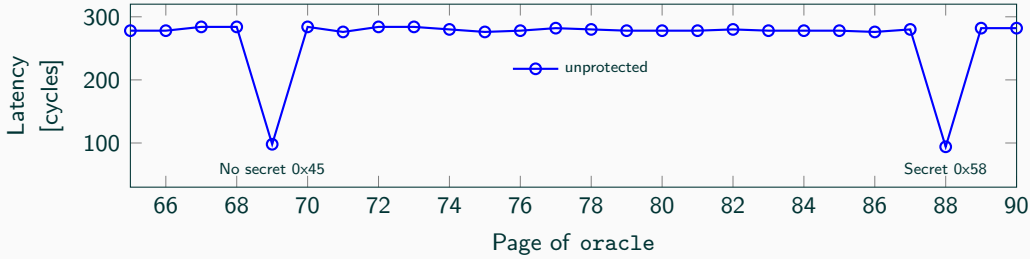

ConTExT-light
cmp rdi, .array_len
jbe .else
Bounds check
not executed
mov (rax + rdi),al
shl 12,rax
and 0xff000,eax
mov (rdx + rax),al
mov 0,rax
retq
mov rax,(rsp + 8)


ConTExT-light
cmp rdi, .array_len
jbe .else
Bounds check
not executed
mov (rax + rdi),al
shl 12,rax
and 0xff000,eax
mov (rdx + rax),al
Out-of-bounds access array[x] stalls
mov 0,rax
retq
mov rax,(rsp + 8)


ConTExT-light
cmp rdi, .array_len
jbe .else
Bounds check
not executed
mov (rax + rdi),al
shl 12,rax
and 0xff000,eax
mov (rdx + rax),al
Out-of-bounds access array[x] stalls
Dependent operations stall
mov 0,rax
retq
mov rax,(rsp + 8)

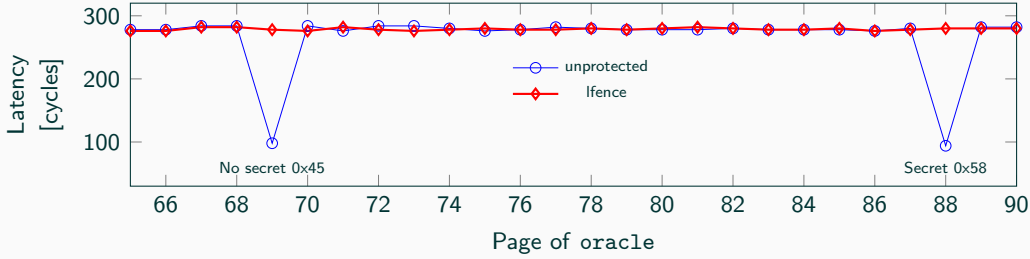

ConTExT-light
cmp rdi, .array_len
jbe .else
not executed
mov (rax + rdi),al
shl 12,rax
and 0xff000,eax
mov (rdx + rax),al
mov 0,rax
retq
mov rax,(rsp + 8)
Bounds check
Out-of-bounds access array[x] stalls
Dependent operations stall
Independent operations continue out-of-order

```
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
}
```

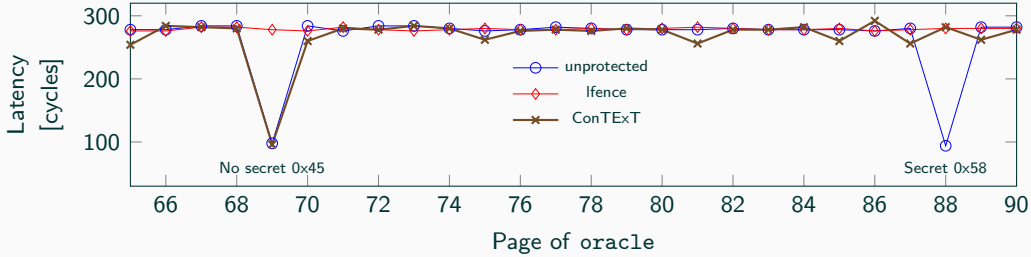


# Compare

```
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
}
```



```
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
}
```



AES-NI

0 %

# Performance

AES-NI

0 %

VeraCrypt

3.21 % (mount) / 0 % (encrypt)

# Performance

AES-NI

0 %

OpenSSH

24.7 % (init) / 5.4 % (transfer)


VeraCrypt

3.21 % (mount) / 0 % (encrypt)

# Performance

AES-NI

0 %


VeraCrypt

3.21 % (mount) / 0 % (encrypt)

OpenSSH

24.7 % (init) / 5.4 % (transfer)

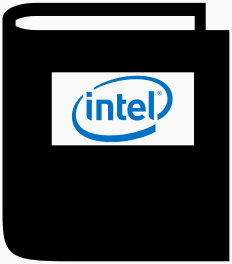
NGINX

7.3 %

You can find our proof-of-concept implementation on:

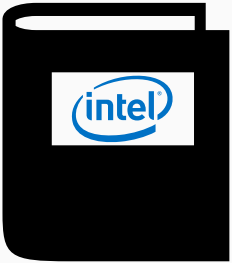
- `https://github.com/IAIK/contextlight`

# A New Memory Type

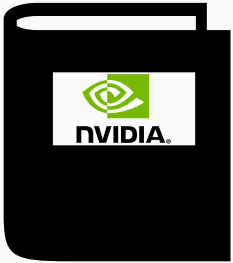



"A New Memory Type against Speculative Side Channel Attacks" [SBH19]

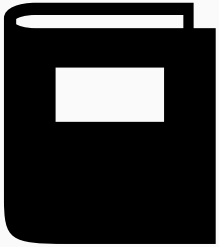
# A New Memory Type




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




"Memory Type Which is Cacheable Yet Inaccessible by Speculative Instructions" [Bog+19]

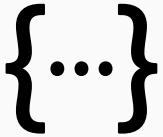


More details in the paper [Sch+20]

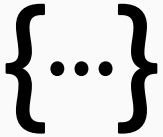
- Compiler modifications
- Taint tracking in register/cache/TLB
- Handling context switches
- ...

**NDSS'20**

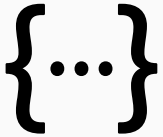
**Michael Schwarz**, Moritz Lipp, Claudio Canella, Robert Schilling, Florian Kargl, Daniel Gruss.
ConTExT: A Generic Approach for Mitigating Spectre.



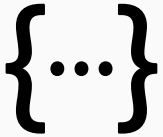
- ConTExT is data-based instead of instruction-based



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



- 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



- 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

# 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

# Acknowledgements



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.

# References


D. D. Boggs, R. Segelken, M. Cornaby, N. Fortino, S. Chaudhry, D. Khartikov, A. Mooley, N. Tuck, and G. Vreugdenhil. Memory type which is cacheable yet inaccessible by speculative instructions. US Patent App. 16/022,274. 2019.

K. Sun, R. Branco, and K. Hu. A New Memory Type Against Speculative Side Channel Attacks. 2019.

M. Schwarz, M. Lipp, C. Canella, R. Schilling, F. Kargl, and D. Gruss. ConTExT: A Generic Approach for Mitigating Spectre. In: NDSS. 2020.