Software-based Microarchitectural Attacks

Daniel Gruss
April 19, 2018

Graz University of Technology
• Daniel Gruss
• Post-Doc @ Graz University of Technology
• Twitter: @lavados
• Email: daniel.gruss@iaik.tugraz.at
• Both vulnerabilities existed for many years
• Both vulnerabilities existed for many years
• No one discovered it before
• Both vulnerabilities existed for many years
• No one discovered it before
• Suddenly, 4 independent teams discover it within 6 months
• Both vulnerabilities existed for many years
• No one discovered it before
• Suddenly, 4 independent teams discover it within 6 months
• Let’s create an evidence board
Why two names, two papers, etc?

- Two different problems
Why two names, two papers, etc?

- Two different problems
- They only have a very loose connection
Why two names, two papers, etc?

- Two different problems
- They only have a very loose connection
- Two different teams had already quite matured drafts ready when learning of each other
Why two names, two papers, etc?

- Two different problems
- They only have a very loose connection
- Two different teams had already quite matured drafts ready when learning of each other
- Initially we tried to merge, but all co-authors quickly agreed that it would mix things that don’t belong together

→ More on that after we understand the attacks
You realize it is something big when...
You realize it is something big when...

- it is in the news, all over the world
You realize it is something big when...

- it is in the news, all over the world
- you get a Wikipedia article in multiple languages
- there are comics, including xkcd
- you get a lot of Twitter followers after Snowden mentioned you

Daniel Gruss — Graz University of Technology
You realize it is something big when...

- it is in the news, all over the world
- you get a Wikipedia article in multiple languages
- there are comics, including xkcd
- you get a lot of Twitter followers after Snowden mentioned you

Daniel Gruss — Graz University of Technology
You realize it is something big when...

- it is in the news, all over the world
- you get a Wikipedia article in multiple languages
- there are comics, including xkcd
- you get a lot of Twitter followers after Snowden mentioned you

Daniel Gruss — Graz University of Technology
The Fallout

You realize it is something big when:

- It is in the news, all over the world
- You get a Wikipedia article in multiple languages
- There are comics, including xkcd
- You get a lot of Twitter follower after Snowden mentioned you

Daniel Gruss — Graz University of Technology
You realize it is something big when...

- it is in the news, all over the world
- you get a Wikipedia article in multiple languages
The Fallout
www.tugraz.at

You realize it is something big when...

• it is in the news, all over the world
• you get a Wikipedia article in multiple languages
• there are comics, including xkcd
• you get a lot of Twitter follower after Snowden mentioned you

Daniel Gruss — Graz University of Technology
The Fallout

You realize it is something big when...

- it is in the news, all over the world
- you get a Wikipedia article in multiple languages
- there are comics, including xkcd
- you get a lot of Twitter followers after Snowden mentioned you

5 Daniel Gruss — Graz University of Technology
You realize it is something big when...

- it is in the news, all over the world
- you get a Wikipedia article in multiple languages
- there are comics, including xkcd
You realize it is something big when...

- it is in the news, all over the world
- you get a Wikipedia article in multiple languages
- there are comics, including xkcd
- you get a lot of Twitter followers after Snowden mentioned you

Daniel Gruss — Graz University of Technology
You realize it is something big when...

- it is in the news, all over the world
- you get a Wikipedia article in multiple languages
- there are comics, including xkcd
- you get a lot of Twitter followers after Snowden mentioned you

Daniel Gruss — Graz University of Technology
You realize it is something big when...

- it is in the news, all over the world
- you get a Wikipedia article in multiple languages
- there are comics, including xkcd
- you get a lot of Twitter follower after Snowden mentioned you
The Fallout

You realize it is something big when...

• it is in the news, all over the world
• you get a Wikipedia article in multiple languages
• there are comics, including xkcd
• you get a lot of Twitter follower after Snowden mentioned you

Edward Snowden
@Snowden

You may have heard about @thehackernews Meltdown bug, but have you watched it in action? When your computer asks you to apply updates this month, don’t click ‘not now’.

via @speculation.com & @mscd110

23.07.4 Jun 2018

5152 17 6.947 0.718
The Wall
• Kernel is isolated from user space
• Kernel is isolated from user space
• This isolation is a combination of hardware and software
• Kernel is isolated from user space
• This isolation is a combination of hardware and software
• User applications cannot access anything from the kernel
The Core of Meltdown/Spectre

- Kernel is isolated from user space
- This isolation is a combination of hardware and software
- User applications cannot access anything from the kernel
- There is only a well-defined interface → syscalls

Daniel Gruss — Graz University of Technology
FOOD CACHE

Revolutionary concept!

Store your food at home, never go to the grocery store during cooking.

Can store ALL kinds of food.

ONLY TODAY INSTEAD OF $1,300

$1,299

ORDER VIA PHONE: +555 12345
printf("%d", i);
printf("%d", i);
printf("%d", i);
printf("%d", i);
printf("%d", i);
printf("%d", i);
printf("%d", i);  # Request

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

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

CPU Cache

Cache miss

Request

Response
printf("%d", i);

Cache miss

printf("%d", i);

Cache hit

i

Request

Response
printf("%d", i);

Cache miss
DRAM access, slow

printf("%d", i);

Cache hit

Request
Response

i
printf("%d", i);
Cache miss
DRAM access,
slow

printf("%d", i);
Cache hit
No DRAM access,
much faster
Flush+Reload

ATTACKER

Shared Memory

VICTIM

flush
access

access
Flush+Reload

ATTACKER

Shared Memory

VICTIM

flush
access
cached
access
cached
Flush+Reload

ATTACKER

VICTIM

Shared Memory

flush

access

access
Flush+Reload

ATTACKER

\textbf{flush}

access

Shared Memory

\textbf{access}

VICTIM

Daniel Gruss — Graz University of Technology
Flush+Reload

ATTACKER

flush
access

Shared Memory

VICTIM

access
Flush + Reload

ATTACKER

Shared Memory

VICTIM

flush

access

Shared Memory

access
Flush+Reload

ATTAkker

Shared Memory

VICTIM

flush

access

Shared Memory

access
Flush+Reload

ATTACKER

Shared Memory

VICTIM

flush
access

fast if victim accessed data, slow otherwise

Daniel Gruss — Graz University of Technology
Cache Template Attack Demo
7. Serve with cooked and peeled potatoes
Wait for an hour
Wait for an hour

LATENCY
1. Wash and cut vegetables

2. Pick the basil leaves and set aside

3. Heat 2 tablespoons of oil in a pan

4. Fry vegetables until golden and softened
1. Wash and cut vegetables

2. Pick the basil leaves and set aside

3. Heat 2 tablespoons of oil in a pan

4. Fry vegetables until golden and softened
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);
```
char data = *(char*)0xffffffff81a000e0;
printf("%c\n", data);
char data = *(char*)0xffffffff81a000e0;
printf("%c\n", data);

segfault at ffffffff81a000e0 ip 0000000000400535
sp 00007ffce4a80610 error 5 in reader
Building Meltdown

```c
char data = *(char*)0xffffffff81a000e0;
printf("%c\n", data);
```

- Kernel addresses are not accessible
char data = *(char*)0xffffffff81a000e0;
printf("%c\n", data);

segfault at ffffffff81a000e0 ip 0000000000400535
sp 00007ffce4a80610 error 5 in reader

- Kernel addresses are not accessible
- Are privilege checks also done when executing instructions out of order?
• Adapted code

```c
*(volatile char*)0;
array[84 * 4096] = 0; // unreachable
```
- Adapted code

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

- Static code analyzer is not happy

```c
warning: Dereference of null pointer
*(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
• Combine the two things

```c
char data = *(char*)0xffffffff81a000e0;
array[data * 4096] = 0;
```
- Combine the two things

```c
char data = *(char*)0xffffffff81a000e0;
array[data * 4096] = 0;
```

= sending end of a cache covert channel

- Then check whether any part of array is cached
• Combine the two things

```c
char data = *(char*)0xffffffff81a000e0;
array[data * 4096] = 0;
```

= sending end of a cache covert channel

• Then check whether any part of array is cached

= receiving end of a cache covert channel
• 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
pwd

Unlock Password Manager

Terminal

File  Edit  View  Search  Terminal  Help

mschwarz@lab06:~:/Documents$
meltdown@meltdown -/pmm2 % taskeet 1 ./imgdump 0x375a00000 14919 > output.flif
Reading from 0xffff880375a00000
Leaking Passwords from your Password Manager

[Image of a password manager window and a list of saved passwords]

Daniel Gruss — Graz University of Technology
AND IN OTHER NEWS...

WE'RE ALL DOOMED, SANDRA. HOW ABOUT THE WEATHER?
Not so fast...
• 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

- ...and remove them if not needed?
- User accessible check in hardware is not reliable
• Let’s just unmap the kernel in user space
• Let’s just unmap the kernel in user space
• Kernel addresses are then no longer present
Let’s just unmap the kernel in user space
Kernel addresses are then no longer present
Memory which is not mapped cannot be accessed at all
Kernel Address Isolation to have Side channels Efficiently Removed

Daniel Gruss — Graz University of Technology
Kernel Address Isolation to have Side channels Efficiently Removed

KAISER /ˈkʌɪzə/
1. [german] Emperor, ruler of an empire
2. largest penguin, emperor penguin

Daniel Gruss — Graz University of Technology
• We published KAISER in July 2017
• We published KAISER in July 2017
• Intel and others improved and merged it into Linux as KPTI (Kernel Page Table Isolation)
• We published KAISER in July 2017
• Intel and others improved and merged it into Linux as KPTI (Kernel Page Table Isolation)
• Microsoft implemented similar concept in Windows 10
We published KAISER in July 2017

Intel and others improved and merged it into Linux as KPTI (Kernel Page Table Isolation)

Microsoft implemented similar concept in Windows 10

Apple implemented it in macOS 10.13.2 and called it “Double Map”
• We published **KAISER** in July 2017
• Intel and others improved and merged it into Linux as **KPTI** (Kernel Page Table Isolation)
• Microsoft implemented similar concept in Windows 10
• Apple implemented it in macOS 10.13.2 and called it “Double Map”
• All share the same idea: switching address spaces on context switch
WAIT A MOMENT...

DUPLICATING EVERYTHING? THAT SOUNDS REALLY SLOW
• Depends on how often you need to switch between kernel and user space
• Depends on how often you need to switch between kernel and user space
• Can be slow, 40% or more on old hardware
- Depends on how often you need to switch between kernel and user space
- Can be slow, 40% or more on old hardware
- But modern CPUs have additional features
- Depends on how often you need to switch between kernel and user space
- Can be slow, 40% or more on old hardware
- But modern CPUs have additional features
- \( \Rightarrow \) Performance overhead on average below 2%
Meltdown and Spectre

MELTDOWN

SPECTRE
Prosciutto
Funghi
Diavolo
Diavolo
»A table for 6 please«
Speculative Cooking
»A table for 6 please«
What does Spectre do?

- Mistrains branch prediction
What does Spectre do?

- Mistrains branch prediction
- CPU speculatively executes code which should not be executed
What does Spectre do?

- Mistrains branch prediction
- CPU speculatively executes code which should not be executed
- Can also mistrain indirect calls
What does Spectre do?

- Mistrains branch prediction
- CPU speculatively executes code which should not be executed
- Can also mistrain indirect calls
  → Spectre “convinces” program to execute code
index = 0;

char* data = "textKEY";

if (index < 4)
    Prediction
else
    LUT[data[index] * 4096]

0
index = 0;

char* data = "textKEY";

if (index < 4)
then
LUT[data[index] * 4096]
else
Prediction

0
index = 0;
char* data = "textKEY";

if (index < 4)
then
LUT[data[index] * 4096]
else
Prediction
Speculate
0
index = 0;

char* data = "textKEY";

if (index < 4)
  then
    LUT[data[index] * 4096]
  else
    0
index = 1;

char* data = "textKEY";

if (index < 4)
then
LUT[data[index] * 4096]
else
Prediction

0
index = 1;

cchar* data = "textKEY";

if (index < 4)

then

LUT[data[index] * 4096]

else

0
index = 1;

char* data = "textKEY";

if (index < 4)
    Speculate
else
    Prediction

LUT[data[index] * 4096] 0
index = 1;

char* data = "textKEY";

if (index < 4)
    then
        LUT[data[index] * 4096]
    else
        0
index = 2;

char* data = "textKEY";

if (index < 4)
then
    Prediction
else
    LUT[data[index] * 4096] = 0
index = 2;

char* data = "textKEY";

if (index < 4)
then
Prediction
LUT[data[index] * 4096]
else
0
index = 2;

char* data = "textKEY";

if (index < 4)
    LUT[data[index] * 4096]
else
    0
index = 2;

char* data = "textKEY";

if (index < 4) then

LUT[data[index] * 4096]

else

0

Prediction
index = 3;

char* data = "textKEY";

if (index < 4)
    Prediction
else
    0
index = 3;

char* data = "textKEY";

if (index < 4)

then

LUT[data[index] * 4096]

else

0

Prediction
index = 3;

cchar* data = "textKEY";

if (index < 4)
then
Speculate
then
LUT[data[index] * 4096]
else
Prediction
else
0
index = 3;

char* data = "textKEY";

if (index < 4)
    then
        LUT[data[index] * 4096]
    else
        0
index = 4;

char* data = "textKEY";

if (index < 4)
then
LUT[data[index] * 4096]
else
Prediction
0
index = 4;

char* data = "textKEY";

if (index < 4)
then
LUT[data[index] * 4096]
else
Prediction

0
index = 4;

char* data = "textKEY";

if (index < 4)

Speculate

then

LUT[data[index] * 4096]

else

Prediction

0

Daniel Gruss — Graz University of Technology
index = 4;

char* data = "textKEY";

if (index < 4)

then

LUT[data[index] * 4096]

else

Prediction

Execute

0
index = 5;

char* data = "textKEY";

if (index < 4)
then
LUT[data[index] * 4096]
else
Prediction
0
index = 5;

char* data = "textKEY";

if (index < 4)
then
Prediction

LUT[data[index] * 4096] 0

else
index = 5;

char* data = "textKEY";

if (index < 4)
    Speculate
    then
    LUT[data[index] * 4096]
else
    Prediction
    0

Daniel Gruss — Graz University of Technology
index = 5;

char* data = "textKEY";

if (index < 4)
    then
        LUT[data[index] * 4096]
    else
        Prediction
        0
        Execute

Daniel Gruss — Graz University of Technology
index = 6;

char* data = "textKEY";

if (index < 4)
then
    Prediction
else
    0

LUT[data[index] * 4096]
index = 6;

char* data = "textKEY";

if (index < 4) {
    Prediction
    LUT[data[index] * 4096]
} else {
    0
}
index = 6;

char* data = "textKEY";

if (index < 4)
    LUT[data[index] * 4096]
else
    0
Spectre (variant 1)

index = 6;

char* data = "textKEY";

if (index < 4)

then

LUT[data[index] * 4096]

else

Prediction

Execute

0
Animal* a = bird;

a->move()

fly()

swim()

LUT[data[index] * 4096]

Prediction

0
```cpp
Animal* a = bird;

a->move();
```

LUT[data[index] * 4096]

Swim

Predict

Speculate

0
Animal* a = bird;

a->move()

fly()

LUT[data[index] * 4096]

swim()

swim()

Prediction

0
Animal* a = bird;

a->move();

Execute

LUT[data[index] * 4096]
Animal* a = bird;

a->move();

fly();

fly();

swim();

LUT[data[index] * 4096]
Animal* a = bird;

a->move();

Speculate
fly()
LUT[data[index] * 4096]

Prediction
fly()
swim()
0
`Animal* a = bird;`
Animal* a = fish;

a->move()

fly()

fly()

swim()

LUT[data[index] * 4096] 0
Animal* a = fish;

a->move();

Speculate

fly()

LUT[data[index] * 4096]

Prediction

fly()  swim()

0
Animal* a = fish;

a->move()

fly()

fly()

swim()

LUT[data[index] * 4096]

0
Animal* a = fish;

a->move();

fly();

LUT[data[index] * 4096]

fly() -> Prediction

swim() -> Execute

0
```cpp
Animal* a = fish;
```

```
LUT[data[index] * 4096]
```

Prediction

fly()

swim()  -> swim()  -> 0

`a->move()`
Trivial approach: disable speculative execution
• Trivial approach: disable speculative execution
• No wrong speculation if there is no speculation
• Trivial approach: disable speculative execution
• No wrong speculation if there is no speculation
• Problem: massive performance hit!
• Trivial approach: disable speculative execution
• No wrong speculation if there is no speculation
• Problem: massive performance hit!
• Also: How to disable it?
Mitigating Spectre

- Trivial approach: disable speculative execution
- No wrong speculation if there is no speculation
- Problem: massive performance hit!
- Also: How to disable it?
- Speculative execution is deeply integrated into CPU
Spectre Variant 1 Mitigations

Wrkaround: insert instructions stopping speculation
insert after every bounds check
x86: ARM:
Available on all Intel CPUs, retrofitted to existing ARMv7 and ARMv8
• Workaround: insert instructions stopping speculation
Workaround: insert instructions stopping speculation
→ insert after every bounds check
Spectre Variant 1 Mitigations

- Workaround: insert instructions stopping speculation
  → insert after every bounds check
- x86: LFENCE, ARM: CSDB
Spectre Variant 1 Mitigations

• Workaround: insert instructions stopping speculation
  → insert after every bounds check
  • x86: LFENCE, ARM: CSDB
  • Available on all Intel CPUs, retrofitted to existing ARMv7 and ARMv8
Speculation barrier requires compiler support.

Already implemented in GCC, LLVM, and MSVC.

Can be automated (MSVC) but not really reliable.

Explicit use by programmer:
• Speculation barrier requires compiler supported
Spectre Variant 1 Mitigations

- Speculation barrier requires compiler supported
- Already implemented in GCC, LLVM, and MSVC
- Speculation barrier requires compiler supported
- Already implemented in GCC, LLVM, and MSVC
- Can be automated (MSVC) → not really reliable
• Speculation barrier requires compiler supported
• Already implemented in GCC, LLVM, and MSVC
• Can be automated (MSVC) → not really reliable
• Explicit use by programmer: `_builtin_load_no_speculate`
// Unprotected

int array[4];

int get_value(unsigned int a) {
    int tmp;
    if (a < 4) {
        tmp = array[a]
    } else {
        tmp = FAIL;
    }
    return tmp;
}
// Unprotected

int array[N];

int get_value(unsigned int n) {  
  int tmp;
  if (n < N) {
    tmp = array[n]
  } else {  
    tmp = FAIL;
  }
  return tmp;
}

// Protected

int array[N];

int get_value(unsigned int n) {
  int *lower = array;
  int *ptr = array + n;
  int *upper = array + N;

  return
  __builtin_load_no_speculate  
  (ptr, lower, upper, FAIL);
}
Spectre Variant 1 Mitigations

Speculation barrier works if affected code constructs are known. Programmers must fully understand the vulnerability. Automatic detection is not reliable. There is a non-negligible performance overhead of barriers.
Speculation barrier works if affected code constructs are known
Speculation barrier works if affected code constructs are known

Programmer has to fully understand vulnerability
Spectre Variant 1 Mitigations

• Speculation barrier works if affected code constructs are known
• Programmer has to fully understand vulnerability
• Automatic detection is not reliable
• Speculation barrier works if affected code constructs are known
• Programmer has to fully understand vulnerability
• Automatic detection is not reliable
• Non-negligible performance overhead of barriers
Intel released microcode updates

- Indirect Branch Restricted Speculation (IBRS):
  - Do not speculate based on anything before entering IBRS mode
  - Lesser privileged code cannot influence predictions
- Indirect Branch Predictor Barrier (IBPB):
  - Flush branch-target buffer
- Single Thread Indirect Branch Predictors (STIBP):
  - Isolates branch prediction state between two hyperthreads
Intel released microcode updates

- Indirect Branch Restricted Speculation (IBRS):
  - Do not speculate based on anything before entering IBRS mode
Intel released microcode updates

- Indirect Branch Restricted Speculation (IBRS):
  - Do not speculate based on anything before entering IBRS mode
  - lesser privileged code cannot influence predictions
Intel released microcode updates

- Indirect Branch Restricted Speculation (IBRS):
  - Do not speculate based on anything before entering IBRS mode
    → lesser privileged code cannot influence predictions

- Indirect Branch Predictor Barrier (IBPB):
Intel released microcode updates

- Indirect Branch Restricted Speculation (IBRS):
  - Do not speculate based on anything before entering IBRS mode
  - lesser privileged code cannot influence predictions

- Indirect Branch Predictor Barrier (IBPB):
  - Flush branch-target buffer
Intel released microcode updates

- Indirect Branch Restricted Speculation (IBRS):
  - Do not speculate based on anything before entering IBRS mode
  - Lesser privileged code cannot influence predictions

- Indirect Branch Predictor Barrier (IBPB):
  - Flush branch-target buffer

- Single Thread Indirect Branch Predictors (STIBP):
Intel released microcode updates

- Indirect Branch Restricted Speculation (IBRS):
  - Do not speculate based on anything before entering IBRS mode
    → lesser privileged code cannot influence predictions

- Indirect Branch Predictor Barrier (IBPB):
  - Flush branch-target buffer

- Single Thread Indirect Branch Predictors (STIBP):
  - Isolates branch prediction state between two hyperthreads
Retpoline (compiler extension)
Retpoline (compiler extension)

```markdown
1. push <call_target>
2. call if
3. 2: ; speculation will continue here
4. 1fence ; speculation barrier
5. 2b ; endless loop
6. 1:
7. lea 8(%rsp), %rsp ; restore stack pointer
8. ret ; the actual call to <call_target>
```

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

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

→ always predict to enter an endless loop
  • instead of the correct (or wrong) target function
Retpoline (compiler extension)

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

→ always predict to enter an endless loop

• instead of the correct (or wrong) target function → performance?
Retpoline (compiler extension)

```assembly
push <call_target>
call 1f

2: ; speculation will continue here
lfence ; speculation barrier
jmp 2b ; endless loop

1:
lea 8(%rsp), %rsp ; restore stack pointer
ret ; the actual call to <call_target>
```

→ always predict to enter an endless loop

- instead of the correct (or wrong) target function → performance?
- On Broadwell or newer:
Retpoline (compiler extension)

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

→ always predict to enter an endless loop

• instead of the correct (or wrong) target function → performance?

• On Broadwell or newer:
  • `ret` may fall-back to the BTB for prediction

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

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

→ always predict to enter an endless loop

- instead of the correct (or wrong) target function → performance?
- On Broadwell or newer:
  - `ret` may fall-back to the BTB for prediction
  → microcode patches to prevent that
• ARM provides hardened Linux kernel
- ARM provides hardened Linux kernel
- Clears branch-predictor state on context switch
ARM provides hardened Linux kernel
Clears branch-predictor state on context switch
Either via instruction (BPIALL)...

Non-negligible performance overhead (≈ 200-300 ns)
• ARM provides hardened Linux kernel
• Clears branch-predictor state on context switch
• Either via instruction (BPIALL)...
• ...or workaround (disable/enable MMU)
• ARM provides hardened Linux kernel
• Clears branch-predictor state on context switch
• Either via instruction (BPIALL)...
• ...or workaround (disable/enable MMU)
• Non-negligible performance overhead ($\approx$ 200-300 ns)
• Prevent access to high-resolution timer
What does not work

- Prevent access to high-resolution timer
  → Own timer using timing thread
What does not work

- Prevent access to high-resolution timer
  → Own timer using timing thread
- Flush instruction only privileged
What does not work

- Prevent access to high-resolution timer
  → Own timer using timing thread
- Flush instruction only privileged
  → Cache eviction through memory accesses
What does not work

- Prevent access to high-resolution timer
  ➔ Own timer using timing thread
- Flush instruction only privileged
  ➔ Cache eviction through memory accesses
- Just move secrets into secure world
What does not work

- Prevent access to high-resolution timer
  → Own timer using timing thread
- Flush instruction only privileged
  → Cache eviction through memory accesses
- Just move secrets into secure world
  → Spectre works on secure enclaves
Meltdown vs. Spectre

Meltdown

Out-of-Order Execution has nothing to do with branch prediction turning off speculative execution completely. It can be mitigated by KAISER.

Spectre

Speculative Execution (subset of Out-of-Order Execution) fundamentally builds on branch (mis)prediction turning off speculative execution entirely would work.

KAISER has no effect on Spectre at all.
Meltdown vs. Spectre

Meltdown
- Out-of-Order Execution

Spectre
- Speculative Execution (subset of Out-of-Order Execution)
Meltdown vs. Spectre

Meltdown

- Out-of-Order Execution
- has nothing to do with branch prediction

Spectre

- Speculative Execution (subset of Out-of-Order Execution)
- fundamentally builds on branch (mis)prediction
Meltdown vs. Spectre

Meltdown

- Out-of-Order Execution
- has **nothing to do with branch prediction**
- turning off speculative execution **entirely** has no effect on Meltdown

Spectre

- Speculative Execution (subset of Out-of-Order Execution)
- fundamentally builds on branch (mis)prediction
- turning off speculative execution **entirely** would work
Meltdown

- Out-of-Order Execution
- has nothing to do with branch prediction
- turning off speculative execution entirely has no effect on Meltdown

→ melts down the isolation provided by the user_accessible-bit

Spectre

- Speculative Execution (subset of Out-of-Order Execution)
- fundamentally builds on branch (mis)prediction
- turning off speculative execution entirely would work
- has nothing to do with the user_accessible-bit
Meltdown

- Out-of-Order Execution
- has nothing to do with branch prediction
- turning off speculative execution entirely has no effect on Meltdown
  → melts down the isolation provided by the user_accessible-bit
- in theory: OoO not required, pipelining can be sufficient

Spectre

- Speculative Execution (subset of Out-of-Order Execution)
- fundamentally builds on branch (mis)prediction
- turning off speculative execution entirely would work
- has nothing to do with the user_accessible-bit
- KAISER has no effect on Spectre at all
Meltdown

- Out-of-Order Execution
- has nothing to do with branch prediction
- turning off speculative execution entirely has no effect on Meltdown
  → melts down the isolation provided by the user_accessible-bit
- in theory: OoO not required, pipelining can be sufficient
- mitigated by KAISER

Spectre

- Speculative Execution (subset of Out-of-Order Execution)
- fundamentally builds on branch (mis)prediction
- turning off speculative execution entirely would work
- has nothing to do with the user_accessible-bit
- KAISER has no effect on Spectre at all

Daniel Gruss — Graz University of Technology
Meltdown vs. Spectre

**Meltdown**

performs illegal memory accesses

- need to take care of processor exceptions
  - exception handling
  - exception suppression with TSX
  - exception suppression with branch misprediction

**Spectre**

performs only legal memory accesses

- has nothing to do with exception handling or suppression

4 papers, 2 names, etc.
Meltdown

- performs illegal memory accesses $\rightarrow$ we need to take care of processor exceptions

Spectre

- performs only legal memory accesses
Meltdown

- performs illegal memory accesses → we need to take care of processor exceptions
  - exception handling

Spectre

- performs only legal memory accesses
  - has nothing to do with exception handling
Meltdown

- performs illegal memory accesses → we need to take care of processor exceptions
  - exception handling
  - exception suppression with TSX

Spectre

- performs only legal memory accesses
  - has nothing to do with exception handling or suppression
Meltdown

• performs illegal memory accesses → we need to take care of processor exceptions
  • exception handling
  • exception suppression with TSX
  • exception suppression with branch misprediction

Spectre

• performs only legal memory accesses
  • has nothing to do with exception handling or suppression
Meltdown
- performs illegal memory accesses $\rightarrow$ we need to take care of processor exceptions
  - exception handling
  - exception suppression with TSX
  - exception suppression with branch misprediction

$\rightarrow$ two papers, two names, etc.

Spectre
- performs only legal memory accesses
  - has nothing to do with exception handling or suppression
But ... why were they named variant 1, and by Google?

"How can you use speculative execution maliciously?"

Intel had much interest in not fancy-naming them ;)

... why were they presented on the same date and on the same website?

We did not choose the date
We did not want to have one of them overshadow the other immediately

Daniel Gruss — Graz University of Technology
But ... why were they named variant 1, and by Google? "How can you use speculative execution maliciously?" Intel had much interest in not fancy-naming them ;)

... why were they presented on the same date and on the same website? We did not choose the date. We did not want to have one of them overshadow the other immediately.
... why were they named variant 1, 2 and 3 by Google?
... why were they named variant 1, 2 and 3 by Google?

- “How can you use speculative execution maliciously?”
... why were they named variant 1, 2 and 3 by Google?

- “How can you use speculative execution maliciously?”
- Intel had much interest in not fancy-naming them ;}
But ...

... why were they named variant 1, 2 and 3 by Google?
- “How can you use speculative execution maliciously?”
- Intel had much interest in not fancy-naming them ;)

... why were they presented on the same date and on the same website?
But ...

... why were they named variant 1, 2 and 3 by Google?

- “How can you use speculative execution maliciously?”
- Intel had much interest in not fancy-naming them ;)

... why were they presented on the same date and on the same website?
... why were they named variant 1, 2 and 3 by Google?
  • “How can you use speculative execution maliciously?”
  • Intel had much interest in not fancy-naming them ;)
... why were they presented on the same date and on the same website?
  • We did not choose the date
... why were they named variant 1, 2 and 3 by Google?
- “How can you use speculative execution maliciously?”
- Intel had much interest in not fancy-naming them ;)

... why were they presented on the same date and on the same website?
- We did not choose the date
- We did not want to have one of them overshadow the other immediately
We have ignored microarchitectural attacks for many many years:
We have ignored microarchitectural attacks for many many years:

- attacks on crypto
We have ignored microarchitectural attacks for many many years:

- attacks on crypto → “software should be fixed”
What do we learn from it?

We have ignored microarchitectural attacks for many many years:

- attacks on crypto → “software should be fixed”
- attacks on ASLR
We have ignored microarchitectural attacks for many many years:

- attacks on crypto → “software should be fixed”
- attacks on ASLR → “ASLR is broken anyway”
What do we learn from it?

We have ignored microarchitectural attacks for many many years:

- attacks on crypto → “software should be fixed”
- attacks on ASLR → “ASLR is broken anyway”
- attacks on SGX and TrustZone
We have ignored microarchitectural attacks for many years:

- attacks on crypto → “software should be fixed”
- attacks on ASLR → “ASLR is broken anyway”
- attacks on SGX and TrustZone → “not part of the threat model”
We have ignored microarchitectural attacks for many many years:

- attacks on crypto $\rightarrow$ “software should be fixed”
- attacks on ASLR $\rightarrow$ “ASLR is broken anyway”
- attacks on SGX and TrustZone $\rightarrow$ “not part of the threat model”

$\rightarrow$ for years we solely optimized for performance
After learning about a side channel you realize:
After learning about a side channel you realize:

- the side channels were documented in the Intel manual
After learning about a side channel you realize:

- the side channels were documented in the Intel manual
- only now we understand the implications
What do we learn from it?

Motor Vehicle Deaths in U.S. by Year

Sealbelts
More Sealbelts
Airbags
More Airbags
ABS
A unique chance to

- rethink processor design
A unique chance to

- rethink processor design
- grow up, like other fields (car industry, construction industry)
Conclusions

A unique chance to

- rethink processor design
- grow up, like other fields (car industry, construction industry)
- dedicate more time into identifying problems and not solely in mitigating known problems
Software-based Microarchitectural Attacks

Daniel Gruss
April 19, 2018

Graz University of Technology