Factor: Das Unfassbare

Die Geschichte von Meltdown und Spectre

Michael Schwarz (@misc0110)
Introduction
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
Is this all a conspiracy?

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

Not a conspiracy

- Tools to detect the bug only invented in 2014
- No broad interest in microarchitectural attacks before
Timeline of a Vulnerability

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” \( \Rightarrow \) a consequence of research in this area
  \[ \rightarrow \text{bug collision nearly inevitable} \]
You realize it is something big when...

[Image of the Earth with a satellite dish on top]
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
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

Michael Schwarz (@misc0110) — www.iaik.tugraz.at
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

SECURITY FLAW REVEALED

<table>
<thead>
<tr>
<th></th>
<th>Intel (Prev)</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>45.26</td>
<td>-1.59</td>
<td>[-3.39%]</td>
</tr>
<tr>
<td></td>
<td>Intel (After Hours)</td>
<td>44.85</td>
<td>-0.41</td>
</tr>
</tbody>
</table>

SHROUT: ISSUE NOT UNIQUE TO INTEL, BUT IT'S AFFECTED THE MOST
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.¹²³ 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 Linux⁵⁶, macOS⁴ 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,¹⁰ 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. On January 18, 2018, unwanted reboots, even for newer Intel chips, due to...
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
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 unneeded "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.

---

INTEL

Nooo, it’s just a tiny bug, barely even matters...

Oh! Look! A squirrel!

AMD

Haha! Well at least we don’t have to worry!

Uh, well, actually...

ARM

OK, so we’ll admit we might also have a small problem, but we’ll find a solution if we work together! #love #peace

INTEL CEO

I swear, when I sold my stock options in November...

...it was just because I wanted to buy this yacht!

CommitStrip.com
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

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 views, 6.547 likes, 6.512 dislikes
Kernel is isolated from user space
The Core of Meltdown

- 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

- 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**
• Breaks isolation between applications and kernel
- Breaks isolation between applications and kernel
- User applications can access kernel addresses
Meltdown Briefing

- Breaks isolation between applications and kernel
- User applications can access kernel addresses
- Entire physical memory is mapped in the kernel
• Breaks isolation between applications and kernel
• User applications can access kernel addresses
• Entire physical memory is mapped in the kernel
→ Meltdown can read whole DRAM
Meltdown Requirements

- Only on Intel CPUs and some ARM processors (Cortex A75)
Meltdown Requirements

- Only on Intel CPUs and some ARM (Cortex A75)
- AMD and other ARM seem to be unaffected
Meltdown Requirements

- Only on Intel CPUs and some ARM (Cortex A75)
- AMD and other ARMs seem to be unaffected
- Common cause: permission check done in parallel to load instruction
Meltdown Requirements

- Only on Intel CPUs and some ARM\textregistered{}s (Cortex A75)
- AMD and other ARMs seem to be unaffected
- Common cause: permission check done in parallel to load instruction
- Race condition between permission check and dependent operation(s)
**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);
Cache miss
printf("%d", i);
printf("%d", i);
printf("%d", i);
printf("%d", i);
printf("%d", i);

Cache miss
Request
Response
printf("%d", i);
printf("%d", i);
printf("%d", i);
Cache miss
printf("%d", i);
Cache hit
Request
Response
CPU Cache

printf("%d", i);

Cache miss

DRAM access, slow

printf("%d", i);

Cache hit

Request

Response

8
printf("%d", i);

Cache miss
Request
Response

printf("%d", i);

Cache hit
No DRAM access,
much faster

DRAM access,
slow

No DRAM access,
much faster
Flush+Reload

Shared Memory

ATTACKER

flush
access

VICTIM

access
Flush+Reload

ATTACKER

flush
access

cached

Shared Memory

VICTIM

flush
access

cached

cached

Michael Schwarz (@misc0110) — www.iaik.tugraz.at
Flush + Reload

ATTACKER

flush

access

Shared Memory

VICTIM

access
Flush+Reload

ATTACKER

Shared Memory

VICTIM

flush
access

access
Flush+Reload

ATTACKER

Shared Memory

VICTIM

flush

access

fast if victim accessed data, slow otherwise
Back to Work
6. Cook everything until 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
```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);
```
```
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
ffffffffffff81a000e0 R linux_banner
```
\texttt{char} data = *(\texttt{char]*)0xffffffff81a000e0;
printf("%c\n", data);
Compile and run

```
segfault at ffffffff81a000e0 ip 0000000000400535
sp 00007ffce4a80610 error 5 in reader
```
• Compile and run

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

• Kernel addresses are of course not accessible
Compile and run

```
segfault 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 straight forward
Privilege checks seem to work
• Still no kernel memory
• Maybe it is not that straightforward
• 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 straightforward
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

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

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

• volatile because compiler was not happy

warning: statement with no effect [−Wunused−value]
*(char*)0;
Building the Code

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

- `volatile` because compiler was not happy
  
  ```c
  warning: statement with no effect [-Wunused-value]
  *(char*)0;
  ```

- Static code analyzer is still 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
Out-of-order instructions leave microarchitectural traces
• Out-of-order instructions leave microarchitectural traces
• We can see them for example in the cache
Out-of-order instructions leave microarchitectural traces.
We can see them for example in the cache.
Give such instructions a name: transient instructions.
• Out-of-order instructions leave microarchitectural traces
• We can see them for example in the cache
• Give such instructions a name: **transient instructions**
• We can indirectly observe the execution of transient instructions
• Maybe there is no permission check in transient instructions...
Maybe there is no permission check in transient instructions...
...or it is only done when committing them
• Maybe there is no permission check in transient instructions...
• ...or it is only done when committing them
• Add another layer of indirection to test

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

```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
pwd

Unlock Password Manager

Terminal

mschwarz@lab06:~/Documents$
And now?...
• 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 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
Kernel View

User View

context switch
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
• ⇒ Performance overhead on average below 2%
Meltdown and Spectre

MELTDOWN

SPECTRE

Michael Schwarz (@misc0110) — www.iaik.tugraz.at
SPECTRE
- Mistrains branch prediction
- Mistrains branch prediction
- CPU speculatively executes code which should not be executed
• Mistrains branch prediction
• CPU speculatively executes code which should not be executed
• Can also mistrain indirect calls
• Mistrains branch prediction
• CPU speculatively executes code which should not be executed
• Can also mistrain indirect calls
→ Spectre “convinces” program to execute code
On Intel and AMD CPUs
- On Intel and AMD CPUs
- Some ARMs (Cortex R and Cortex A) are also affected
- On Intel and AMD CPUs
- Some ARMs (Cortex R and Cortex A) are also affected
- Common cause: speculative execution of branches
Spectre Requirements

- On Intel and AMD CPUs
- Some ARMs (Cortex R and Cortex A) are also affected
- Common cause: speculative execution of branches
- Speculative execution leaves microarchitectural traces which leak secret
Funghi
Diavolo
Diavolo
Diavolo
»A table for 6 please«
Speculative Cooking
»A table for 6 please«
index = 0;

char* data = "textKEY";

if (index < 4)

then

LUT[data[index] * 4096]

else

Prediction

0
index = 0;

char* data = "textKEY";

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

char* data = "textKEY";

if (index < 4)

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

char* data = "textKEY";

if (index < 4)
else
index = 1;

char* data = "textKEY";

if (index < 4)

then

Prediction

LUT[data[index] * 4096]

else

0
index = 1;

char* data = "textKEY";

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

char* data = "textKEY";

if (index < 4)
    Prediction
else
    Speculate

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

char* data = "textKEY";

if (index < 4)

then

Prediction

LUT[data[index] * 4096]

else

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

char* data = "textKEY";

if (index < 4)

then

Speculate

LUT[data[index] * 4096]

else

Prediction

0
index = 2;

char* data = "textKEY";

if (index < 4)
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 = 3;

char* data = "textKEY";

if (index < 4)

then

LUT[data[index] * 4096]

else

Prediction

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
        Prediction
        LUT[data[index] * 4096]
    else
        0
index = 4;

char* data = "textKEY";

if (index < 4)

then

Prediction

LUT[data[index] * 4096]

else

0
index = 4;

char* data = "textKEY";

if (index < 4)

then

Prediction

LUT[data[index] * 4096]

else

0
index = 4;

char* data = "textKEY";

if (index < 4)

Speculate

then

LUT[data[index] * 4096]

else

Prediction

0


```c
index = 4;

char* data = "textKEY";

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

Execute
```

Michael Schwarz (@misc0110) — www.iaik.tugraz.at
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

LUT[data[index] * 4096]

else

0

Prediction
index = 5;

char* data = "textKEY";

if (index < 4)

Speculate

then

Prediction

LUT[data[index] * 4096]

else

0
index = 5;

char* data = "textKEY";

if (index < 4)
else
Prediction

LUT[data[index] * 4096]
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

0
index = 6;

char* data = "textKEY";

if (index < 4) then

LUT[data[index] * 4096]

Prediction

else

0
index = 6;

`char* data = "textKEY";`

`if (index < 4)`

`then`  
`Prediction`

LUT[data[index] * 4096]

`else`  
`Execute`

0
Animal* a = bird;

a->move()

fly()
swim()
swim()

LUT[data[index] * 4096]

Prediction

0
Animal* a = bird;

a->move();

fly()

swim()

LUT[data[index] * 4096]

swim()

Prediction

Speculate

0
\texttt{Animal* a = bird;}

\texttt{a->move();}

\texttt{LUT[data[index] * 4096]}
Animal* a = bird;

```
Execute
a->move();

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

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

```
LUT[data[index] * 4096] 0
```
```c
Animal* a = bird;

a->move();

fly();

LUT[data[index] * 4096] 0

swim();
```

Prediction
\texttt{Animal}\* \texttt{a} = \texttt{fish};

\texttt{a->move()}

\texttt{LUT[data[index] \ast 4096]}

\texttt{0}
```c
Animal* a = fish;
a->move();
```
```cpp
Animal* a = fish;

a->move();

fly()
fly()

Prediction

LUT[data[index] * 4096]

0
```
```cpp
Animal* a = fish;
a->move();
```

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

Prediction

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

Execute

0
Animal* a = fish;

a->move();

fly();
swim();
swim();

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
• 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?
• 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

Workaround: insert instructions stopping speculation

x86: LFENCE
     ARM: CSDB

Available on all Intel CPUs, retrofitted to existing ARMv7 and ARMv8
• Workaround: insert instructions stopping speculation
Spectre Variant 1 Mitigations

- 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).

Explicit use by programmer:

```c
builtin
load
no
speculate
```
• Speculation barrier requires compiler supported
• 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[N];

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

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

    return tmp;
}
```

```c
// 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);
}
```
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.
Spectre Variant 1 Mitigations

- Speculation barrier works if affected code constructs are known
Spectre Variant 1 Mitigations

- Speculation barrier works if affected code constructs are known
- Programmer has to fully understand vulnerability
Speculation barrier works if affected code constructs are known

Programmer has to fully understand vulnerability

Automatic detection is not reliable
Spectre Variant 1 Mitigations

- 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)

```
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
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
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 → 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:

On Broadwell or newer:
- `ret` may fall-back to the BTB for prediction
- microcode patches to prevent that
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:
  - `ret` may fall-back to the BTB for prediction
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:
  - `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)...
ARM provides hardened Linux kernel
Clears branch-predictor state on context switch
Either via instruction ($\texttt{BPIALL}$)...
...or workaround (disable/enable MMU)
Spectre Variant 2 Mitigations (Software)

- 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 ($\approx 200\text{-}300\text{ ns}$)
What does not work

- 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
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 → “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”
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”
- for years we solely optimized for performance
When you read the Intel manuals...

After learning about a side channel you realize:
When you read the Intel manuals...

After learning about a side channel you realize:

- the side channels were documented in the Intel manual
When you read the Intel manuals...

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

- Seatbelts
- More Seatbelts
- Airbags
- More Airbags
- ABS
A unique chance to
- rethink processor design
- grow up, like other fields (car industry, construction industry)
- find good trade-offs between security and performance
• 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”
Fact or Fiction?

MELTDOWN

SPECTRE
Any Questions?
Factor:
Das Unfassbare

Die Geschichte von Meltdown und Spectre

Michael Schwarz (@misc0110)