• Meltdown and Spectre are two CPU vulnerabilities
Meltdown and Spectre are two CPU vulnerabilities
Discovered in 2017 by 4 independent teams
• Meltdown and Spectre are two CPU vulnerabilities
• Discovered in 2017 by 4 independent teams
• Due to an embargo, released at the beginning of 2018
Vulnerability Assessment

• Meltdown and Spectre are two CPU vulnerabilities
• Discovered in 2017 by 4 independent teams
• Due to an embargo, released at the beginning of 2018
• News coverage followed by a lot of panic
NEWS ALERT

INTEL REVEALS DESIGN FLAW THAT COULD ALLOW HACKERS TO ACCESS DATA
DEVELOPING STORY

COMPUTER CHIP FLAWS IMPACT BILLIONS OF DEVICES
GLOBAL

COMPUTER CHIP SCARE

The bugs are known as 'Spectre' and 'Meltdown'
SECURITY FLAW REVEALED

<table>
<thead>
<tr>
<th></th>
<th>Intel (Prev)</th>
<th>Intel (After Hours)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Price</td>
<td>45.26</td>
<td>44.85</td>
</tr>
<tr>
<td>Change</td>
<td>-1.59</td>
<td>-0.41</td>
</tr>
<tr>
<td>Percent Change</td>
<td>-3.39%</td>
<td>-0.91%</td>
</tr>
</tbody>
</table>

SHROUT: ISSUE NOT UNIQUE TO INTEL, BUT IT'S AFFECTED THE MOST
A lot of confusion fueled the panic

- Which CPUs/vendors are affected?
A lot of confusion fueled the panic

- Which CPUs/vendors are affected?
- Are smartphones/IoT devices affected?
A lot of confusion fueled the panic

- Which CPUs/vendors are affected?
- Are smartphones/IoT devices affected?
- Can the vulnerabilities be exploited remotely?
A lot of confusion fueled the panic

- Which CPUs/vendors are affected?
- Are smartphones/IoT devices affected?
- Can the vulnerabilities be exploited remotely?
- What data is at risk?
A lot of confusion fueled the panic

- Which CPUs/vendors are affected?
- Are smartphones/IoT devices affected?
- Can the vulnerabilities be exploited remotely?
- What data is at risk?
- How hard is it to exploit the vulnerabilities?
A lot of confusion fueled the panic

- Which CPUs/vendors are affected?
- Are smartphones/IoT devices affected?
- Can the vulnerabilities be exploited remotely?
- What data is at risk?
- How hard is it to exploit the vulnerabilities?
- Is it already exploited?
Let’s try to clarify these questions
• Kernel is isolated from user space
Hardware Isolation

- 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
• 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*
Meltdown Briefing

- Breaks isolation between applications and kernel
• Breaks isolation between applications and kernel
• User applications can access kernel addresses
• Breaks isolation between applications and kernel
• User applications can access kernel addresses
• Entire physical memory is mapped in the kernel
Meltdown Briefing

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

- Only on Intel CPUs and some unreleased 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 unreleased ARM (Cortex A75)
- AMD and other ARM seem to be unaffected
- Common cause: permission check done in parallel to load instruction
- Race condition between permission check and dependent operation(s)
Meltdown Variant Requirements

- Meltdown variant: read privileged registers

Michael Schwarz, Moritz Lipp — www.iaik.tugraz.at
Meltdown Variant Requirements

- Meltdown variant: read privileged registers
- Limited to some registers, no memory content
Meltdown Variant Requirements

- Meltdown variant: read privileged registers
- Limited to some registers, no memory content
- Reported by ARM
- Meltdown variant: read privileged registers
- Limited to some registers, no memory content
- Reported by ARM
- Affects some ARMv8 (Cortex A15, A57, and A72)
Meltdown Exploitability

- Meltdown requires code execution on the device (e.g. Apps)
Meltdown requires code execution on the device (e.g. Apps)
Untrusted code can read entire memory of device
Meltdown Exploitability

- Meltdown requires code execution on the device (e.g. Apps)
- Untrusted code can read entire memory of device
- Cannot be triggered remotely
- Meltdown requires code execution on the device (e.g. Apps)
- Untrusted code can read entire memory of device
- Cannot be triggered remotely
- Proof-of-concept code available online
• Meltdown requires code execution on the device (e.g. Apps)
• Untrusted code can read entire memory of device
• Cannot be triggered remotely
• Proof-of-concept code available online
• No info about environment required → easy to reproduce
• 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
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.
Spectre Requirements

- 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
• Spectre requires code execution on the device (e.g. Apps)
Spectre Exploitability

- Spectre requires code execution on the device (e.g. Apps)
- Untrusted code can convince trusted code to reveal secrets
• Spectre requires code execution on the device (e.g. Apps)
• Untrusted code can convince trusted code to reveal secrets
• Can be triggered remotely (e.g. in the browser)
Spectre Exploitability

- Spectre requires code execution on the device (e.g. Apps)
- Untrusted code can convince trusted code to reveal secrets
- Can be triggered remotely (e.g. in the browser)
- Proof-of-concept code available online
Spectre Exploitability

- Spectre requires code execution on the device (e.g. Apps)
- Untrusted code can convince trusted code to reveal secrets
- Can be triggered remotely (e.g. in the browser)
- Proof-of-concept code available online
- Info about environment required → hard to reproduce
Background
printf("%d", i);
printf("%d", i);
printf("%d", i);

Cache miss

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

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

```
printf("%d", i);
printf("%d", i);
```

Cache miss

Request

Response

i
CPU Cache

printf("%d", i);

Cache miss

i

printf("%d", i);

Cache hit

Request

Response

Cache miss

Cache hit
CPU Cache

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

**Cache miss**

**Cache hit**

DRAM access, slow

Request

Response
printf("%d", i);

Cache miss

DRAM access,
slow

i

Cache hit

No DRAM access,
much faster

Request

Response
Flush+Reload

ATTACKER

flush
access

Shared Memory

VICTIM

access
Flush+Reload

ATTACKER

flush
access

Shared Memory

cached

access

VICTIM

cached

Michael Schwarz, Moritz Lipp — www.iaik.tugraz.at
Flush+Reload

ATTACKER

**flush**

**access**

Shared Memory

VICTIM

**access**
Flush + Reload

ATTACKER

flush

access

Shared Memory

VICTIM

access
Flush + Reload

Shared Memory

ATTACKER

flush
access

Victim

access
Flush+Reload

ATTACKER

flush

access

fast if victim accessed data, slow otherwise

Shared Memory

VICTIM

access

Michael Schwarz, Moritz Lipp — www.iaik.tugraz.at
Out-of-order Execution
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);
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

- Instructions are fetched and decoded in the front-end
- Instructions are dispatched to the backend
- Instructions are processed by individual execution units
- Instructions are executed out-of-order
- Instructions wait until their dependencies are ready
  - Later instructions might execute prior earlier instructions
- Instructions retire in-order
  - State becomes architecturally visible
We are ready for the gory details of Meltdown
• Find something human readable, e.g., the Linux version

```
# sudo grep linux_banner /proc/kallsyms
ffffffffffffff81a000e0 R linux_banner
```
char data = *(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 0000000000400535
    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 straightforward
• 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

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

\[
\ast (\text{volatile char} \ast) \ 0;
\]

array[0] = 0;

- `volatile` because compiler was not happy

**warning**: statement with no effect [-Wunused-value]

\[
\ast (\text{char} \ast) \ 0;
\]
• Adapted code

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

• \texttt{volatile} because compiler was not happy

\texttt{warning: statement with no effect [-Wunused-value]}
\star\texttt{(char*) 0;}

• Static code analyzer is still not happy

\texttt{warning: Dereference of null pointer}
\star\texttt{(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
Building Meltdown

- 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
• Combine the two things

```c
char data = *(char*)0xfffffffff81a000e0;
array[data * 4096] = 0;
```
• Combine the two things

```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
- Using out-of-order execution, we can read data at any address
• Using out-of-order execution, we can read data at any address
• Privilege checks are sometimes too slow
Using out-of-order execution, we can read data at any address
Privilege checks are sometimes too slow
Allows to leak kernel memory
• Using out-of-order execution, we can read data at any address
• Privilege checks are sometimes too slow
• Allows to leak kernel memory
• Entire physical memory is typically also accessible in kernel address space
Dumping memory

used with authorization from Sun Microsystems, Inc. However, the authors make no claim that Mesa is in any way a compatible replacement for OpenGL or associated with Silicon Graphics, Inc.

... This version of Mesa provides GLX and DRI capabilities: it is capable of both direct and indirect rendering. For direct rendering, it can use DRI modules from the libg
if <access in bounds>
if <access in bounds>

true

predicted
if <access in bounds>

predicted

true

true
if <access in bounds>
if <access in bounds>
if <access in bounds>
if <access in bounds>
if <access in bounds>
if <access in bounds>
if <access in bounds>
if <access in bounds>
We are ready for the gory details of Spectre
index = 0;

define data as a character array

char* data = "textKEY";

if (index < 4)

then

Prediction

LUT[data[index] * 4096]

else

0
index = 0;

char* data = "textKEY";

if (index < 4)
then

LUT[data[index] * 4096]
else

0
index = 0;

char* data = "textKEY";

if (index < 4)

then

LUT[data[index] * 4096]

else

Speculate

Prediction

0
index = 0;

char* data = "textKEY";

if (index < 4)

  LUT[data[index] * 4096]

else
  0
index = 1;

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

```
if (index < 4)

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

Prediction

```
else

0
```
index = 1;

char* data = "textKEY";

if (index < 4)

then

LUT[data[index] * 4096]

else

0
index = 1;

char* data = "textKEY";

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

Michael Schwarz, Moritz Lipp — www.iaik.tugraz.at
index = 1;

char* data = "textKEY";

if (index < 4)

then

Prediction

LUT[data[index] * 4096]

else

0

Michael Schwarz, Moritz Lipp — www.iaik.tugraz.at
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
\begin{verbatim}
index = 2;
char* data = "textKEY";

if (index < 4)
    LUT[data[index] * 4096]
else
    0
\end{verbatim}
index = 2;

char* data = "textKEY";

if (index < 4)

then

Prediction

LUT[data[index] * 4096]

else

0
index = 3;

char* data = "textKEY";

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

char* data = "textKEY";

if (index < 4)

then

LUT[data[index] * 4096]

else

0

Prediction

Michael Schwarz, Moritz Lipp — www.iaik.tugraz.at
index = 3;

char* data = "textKEY";

if (index < 4)

Speculate
then
LUT[data[index] * 4096]

Prediction
else
0
index = 3;

char* data = "textKEY";

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

\texttt{char* data} = "textKEY";

\texttt{if (index < 4)}

\texttt{then}

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

\texttt{else}

\texttt{0}
index = 4;

char* data = "textKEY";

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

char* data = "textKEY";

if (index < 4)

Speculate
then
LUT[data[index] * 4096]

Prediction
else
0
index = 4;

char* data = "textKEY";

if (index < 4)
    LUT[data[index] * 4096]
index = 5;

char* data = "textKEY";

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

Michael Schwarz, Moritz Lipp — www.iaik.tugraz.at
index = 5;

char* data = "textKEY";

if (index < 4)

then

LUT[data[index] * 4096]

else

Prediction

Execute

0
\texttt{index} = 6;

\texttt{char* data} = "textKEY";

\texttt{if (index < 4)}

\texttt{then}

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

\texttt{else}

\texttt{0}

\texttt{Prediction}
index = 6;

char* data = "textKEY";

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

char* data = "textKEY";

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

char* data = "textKEY";

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

0

Execute
```
Animal* a = bird;

a->move()

fly()  swim()  swim()

LUT[data[index] * 4096]  0

Prediction
Spectre (variant 2)

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

```python
a->move()
```

LUT[data[index] * 4096]

Speculate

- fly()
- swim()
- Prediction

0
 Animal* a = bird;

LUT[data[index] * 4096]

0
Animal* a = bird;

Execute

a->move();

LUT[data[index] * 4096]

Prediction

0

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

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

a->move();

Speculate

fly()

LUT[data[index] * 4096]

Prediction

fly()  swim()  0
Animal* a = bird;

a->move()

fly()
fly()
swim()

Prediction

LUT[data[index] * 4096] 0
Spectre (variant 2)

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

Prediction

LUT[data[index] * 4096]

0
Animal* a = fish;

a->move();

Speculate

LUT[data[index] * 4096]

fly()

Prediction

fly()

swim()

0
Animal* a = fish;

a->move();

fly();

fly();

swim();

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

a->move();

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

a->move()
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 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

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

• Does not directly access kernel
• Does not directly access kernel
• “Convinces” other programs to reveal their secrets
● Does not directly access kernel
● “Convinces” other programs to reveal their secrets
● Much harder to fix, KAISER does not help
Spectre

• Does not directly access kernel
• “Convinces” other programs to reveal their secrets
• 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 → insert after every bounds check
- 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
• 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
Spectre Variant 1 Mitigations

• 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`
• Speculation barrier requires compiler supported
Spectre Variant 1 Mitigations

- Speculation barrier requires compiler supported
- Already implemented in GCC, LLVM, and MSVC
Spectre Variant 1 Mitigations

- Speculation barrier requires compiler supported
- Already implemented in GCC, LLVM, and MSVC
- Can be automated (MSVC) \(\rightarrow\) 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;
}
Spectre Variant 1 Mitigations

// 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
- Programmer has to fully understand vulnerability
- Automatic detection is not reliable
- Non-negligible performance overhead of barriers
• 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):**
  - Isolates branch prediction state between two hyperthreads
Retpoline (compiler extension)
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
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)

```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:
Spectre Variant 2 Mitigations (Software)

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
Spectre Variant 2 Mitigations (Software)

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)...

Spectre Variant 2 Mitigations (Software)
• ARM provides hardened Linux kernel
• Clears branch-predictor state on context switch
• Either via instruction \texttt{(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 (≈ 200-300 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
What to do now?
• Is the used hardware even affected?
• Is the used hardware even affected?
• Can untrusted users run code on affected hardware?
Don’t panic

- Is the used hardware even affected?
- Can untrusted users run code on affected hardware?
- Is a software attack even in the threat model?
Don’t panic

- Is the used hardware even affected?
- Can untrusted users run code on affected hardware?
- Is a software attack even in the threat model?
- Is confidentiality required on the hardware?
We have ignored software side-channels for many many years:
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”
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
We have ignored software side-channels 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”
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 manuals...

After learning about a side channel you realize:

- the side channels were documented in the Intel manual
- only now we understand the implications
When you read the manuals...

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
  - 705.005 System-On-Chip
- find good trade-offs between security and performance
Conclusion

• Underestimated microarchitectural attacks for a long time
• Meltdown and Spectre exploit performance optimizations
  • Allow to leak arbitrary memory
• Countermeasures come with a performance impact
• Find trade-offs between security and performance