Spectre and Meltdown on x86 and ARM

Michael Schwarz, Moritz Lipp, Stefan Mangard
15.02.2018

www.iaik.tugraz.at
Meltdown and Spectre are two CPU vulnerabilities
Meltdown and Spectre are two CPU vulnerabilities
Discovered in 2017 by 4 independent teams
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
• 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
COMPUTER CHIP SCARE
The bugs are known as 'Spectre' and 'Meltdown'.
<table>
<thead>
<tr>
<th>Security Flaw Revealed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel (Prev)</td>
</tr>
<tr>
<td>45.26</td>
</tr>
<tr>
<td>-1.59</td>
</tr>
<tr>
<td>[-3.39%]</td>
</tr>
<tr>
<td>Intel (After Hours)</td>
</tr>
<tr>
<td>44.85</td>
</tr>
<tr>
<td>-0.41</td>
</tr>
<tr>
<td>[-0.91%]</td>
</tr>
</tbody>
</table>

Shrou: 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
- User applications cannot access anything from the kernel
- There is only a well-defined interface → syscalls

### Diagram:

- Userspace
- Kernelspace
- Applications
- Operating System
- Memory

Michael Schwarz, Moritz Lipp, Stefan Mangard — www.iaik.tugraz.at
- Kernel is isolated from user space
- This *isolation* is a combination of hardware and software
Hardware Isolation

- 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
• 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
• 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 unreleased ARM (Cortex A75)

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

- 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 ARMv8 CPUs (Cortex A75)
- AMD and other ARMv8 CPUs 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: read privileged registers
• 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 Requirements

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

- Meltdown requires code execution on the device (e.g. Apps)

Michael Schwarz, Moritz Lipp, Stefan Mangard — www.iaik.tugraz.at
Meltdown Exploitability

- Meltdown requires code execution on the device (e.g. Apps)
- Untrusted code can read entire memory of device
• Meltdown requires code execution on the device (e.g. Apps)
• Untrusted code can read entire memory of device
• Cannot be triggered remotely
Meltdown Exploitability

- 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
SPECTRE
Spectre Briefing

- 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

Michael Schwarz, Moritz Lipp, Stefan Mangard — www.iaik.tugraz.at
• On Intel and AMD CPUs
• Some ARMs (Cortex R and Cortex A) are also affected
Spectre Requirements

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

Cache miss

DRAM access,
slow

Request

Response

Cache hit

No DRAM access,
much faster

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

ATTACKER

flush
access

Shared Memory

VICTIM

access
Flush+Reload

Shared Memory

ATTACKER

flush
access
cached

VICTIM

cached
access
Flush+Reload

ATTACKER

flush

access

Shared Memory

VICTIM

access

Michael Schwarz, Moritz Lipp, Stefan Mangard — 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

12

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

ATTACKER

flush

access

fast if victim accessed data,
slow otherwise

Shared Memory

access

VICTIM
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);
Out-of-order Execution

```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);
```
We are ready for the gory details of Meltdown
char data = *(char*)0xffffffff81a000e0;
printf("%c\n", data);
`char` data = *(char*)0xffffffff81a000e0;
printf("%c\n", data);

segfault at fffffffff81a000e0 ip 0000000000400535
sp 00007ffce4a80610 error 5 in reader
```c
char data = *(char*)0xffffffff81a000e0;
printf("%c\n", data);
```

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

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

segfault at ffffffff81a000e0 ip 000000000400535
sp 00007ffce4a80610 error 5 in reader

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

\texttt{* (volatile char*) 0;}
array[84 * 4096] = 0; // unreachable
• Adapted code

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

• Static code analyzer is not happy

    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
• Combine the two things

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

    char data = *(char*)0xfffffffff81a000e0;
    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
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>
if <access in bounds>
if <access in bounds>
if <access in bounds>
We are ready for the gory details of Spectre
index = 0;

char* data = "textKEY";

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

char* data = "textKEY";

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

Prediction
index = 0;

char* data = "textKEY";

if (index < 4)
    LUT[data[index] * 4096]
else
    0
\begin{itemize}
\item \texttt{index} = 0;
\item \texttt{char* data = "textKEY";}
\item \texttt{if} (\texttt{index} < 4)
\item \texttt{LUT[data[index] \times 4096]}
\item \texttt{Prediction}
\item \texttt{else}
\item \texttt{0}
\end{itemize}
index = 1;

char* data = "textKEY";

if (index < 4)

then

LUT[data[index] * 4096]

else

0

Prediction
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

Speculate

then

Prediction

else
index = 1;

char* data = "textKEY";

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

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)

then

LUT[data[index] * 4096]

else

Prediction

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

0

Prediction

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

char* data = "textKEY";

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

`Speculate`

`then`

`LUT[data[index] * 4096]`

`else`

`Prediction`

`0`
index = 3;

char* data = "textKEY";

if (index < 4)

then

LUT[data[index] * 4096]

else

0
index = 4;

char* data = "textKEY";

if (index < 4)

then

LUT[data[index] * 4096]

else

0
if (index < 4)

char * data = "textKEY";

index = 4;

LUT[data[index] * 4096]
index = 4;

char* data = "textKEY";

if (index < 4)

    Speculate

    then

    LUT[data[index] * 4096]

else

    Prediction

    0
index = 4;

char* data = "textKEY";

if (index < 4)

then

LUT[data[index] * 4096]

else

Prediction

Execute

0
index = 5;

char* data = "textKEY";

if (index < 4)

then

Prediction

else

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

char* data = "textKEY";

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

char* data = "textKEY";

if (index < 4)

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

char* data = "textKEY";

if (index < 4) then

LUT[data[index] * 4096]

else Prediction

Execute

0
index = 6;

char* data = "textKEY";

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

char* data = "textKEY";

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

0
index = 6;

char* data = "textKEY";

if (index < 4)

then

Speculate

Prediction

LUT[data[index] * 4096]

else

0
index = 6;

char* data = "textKEY";

if (index < 4)

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

a->move();

fly()

swim()

swim()

LUT[data[index] * 4096] 0

Prediction
Spectre (variant 2)

\[
\text{Animal}^* \ a = \text{bird};
\]

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

```
fly()
```

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

```
swim()
```

```
Prediciton
```

```
Speculate
```

```
0
```

```
swim()
```
Animal* a = bird;

a->move()

fly()

LUT[data[index] * 4096] 0

Prediction

swim()
Animal* a = bird;

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

a->move()

fly()

fly()

swim()

LUT[data[index] * 4096] 0
\( \text{Animal}\* \; a = \text{bird}; \)

\[
\text{Speculate} \\
\text{LUT}[\text{data}[\text{index}] \times 4096] \\
\text{fly}() \\
\text{Prediction} \\
\text{fly}() \\
\text{swim}() \\
0 \\
\text{a} \rightarrow \text{move}() 
\]
Animal* a = bird;

a->move()

fly()

fly()  swim()

Prediction

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

Prediction

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

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

Speculate

LUT[data[index] * 4096]

fly()

fly()

Prediction

swim()

0
Animal* a = fish;

a->move();

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

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

```
LUT[data[index] * 4096]  0
```
• Idea: unmap the kernel in user space
Idea: unmap the kernel in user space
Kernel addresses are then no longer present
• Idea: 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 View

- Userspace
- Kernelspace
- Applications
- Operating System
- Memory

User View

- Userspace
- Kernelspace
- Applications
• 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
- 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.
Spectre Variant 1 Mitigations

- Workaround: insert instructions stopping speculation
  - x86: LFENCE, ARM: CSDB
  - Available on all Intel CPUs, retrofitted to existing ARMv7 and ARMv8
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 supported
• Already implemented in GCC, LLVM, and MSVC
• Can be automated (MSVC) → not really reliable

Explicit use by programmer: `builtin load no speculate`
Spectre Variant 1 Mitigations

• Speculation barrier requires compiler supported
• 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) → 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;
}
// 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
Speculation barrier works if affected code constructs are known
Programmer has to fully understand vulnerability
Spectre Variant 1 Mitigations

- Speculation barrier works if affected code constructs are known
- Programmer has to fully understand vulnerability
- Automatic detection is not reliable
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
    \[ \rightarrow \text{lesser privileged code cannot influence predictions} \]

- 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)

```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:

Michael Schwarz, Moritz Lipp, Stefan Mangard — www.iaik.tugraz.at
Retpoline (compiler extension)

```
push <call_target>
call 1f
2:
  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 (BPIALL)...
- ...or workaround (disable/enable MMU)
• ARM provides hardened Linux kernel
• Clears branch-predictor state on context switch
• Either via instruction ($BPIALL$)...
• ...or workaround (disable/enable MMU)
• Non-negligible performance overhead ($\approx 200\text{-}300 \text{ ns}$)
• Prevent access to high-resolution timer
What does not work

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

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

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

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

- Prevent access to high-resolution timer
  → Own timer using timing thread
- Flush instruction only privileged
  → Cache eviction through memory accesses
- Just move secrets into secure world
  → Spectre works on secure enclaves
What to do now?
• Is the used hardware even affected?
Don’t panic

- 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?
• 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 $\rightarrow$ “software should be fixed”
- attacks on ASLR $\rightarrow$ “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 → “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
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

- 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”
Any Questions?
Spectre and Meltdown on x86 and ARM

Michael Schwarz, Moritz Lipp, Stefan Mangard
15.02.2018

www.iaik.tugraz.at