A Christmas Carol

The Spectres of the Past, Present, and Future

Claudio Canella
Moritz Lipp

Daniel Gruss
Michael Schwarz
Acknowledgements

Background music for the choir song kindly provided by Kerbo-Kev.

Cooking photos kindly provided by Becca Lee (ladyfaceblog).

Santa Clause images by http://www.thefvectorart.com/

Some picture components are included from "Mickey's Christmas Carol" under fair use.
We want to thank our collaborators: Anders Fogh, Benjamin von Berg, Daniel Genkin, Dmitry Evtyushkin, Frank Piessens, Jann Horn, Jo Van Bulck, Mike Hamburg, Paul Kocher, Philipp Ortner, Stefan Mangard, Thomas Prescher, Werner Haas, and Yuval Yarom.

The research behind this talk was partially funded by a generous gift from ARM and a generous gift from Intel. Any opinions, findings, and conclusions or recommendations are those of the authors and do not necessarily reflect the views of the funding parties.
Performance is awesome!

- 1995
- 150 MHz
- RISC emulating CISC
- 256 KB L2 cache integrated!
- branch prediction
- out-of-order execution
Performance is awesome!

- 1995
- 150 MHz
- RISC emulating CISC
- 256 KB L2 cache integrated!
- branch prediction
- out-of-order execution
Performance is awesome!

• 1995

1995

• 150 MHz
• RISC emulating CISC
• 256 KB L2 cache integrated!
• branch prediction
• out-of-order execution
Performance is awesome!

- 1995
- 150 MHz
Performance is awesome!

- 1995
- 150 MHz
- RISC emulating CISC
Performance is awesome!

- 1995
- 150 MHz
- RISC emulating CISC
- 256 KB L2 cache integrated!
Performance is awesome!

- 1995
- 150 MHz
- RISC emulating CISC
- 256 KB L2 cache integrated!
- branch prediction
Performance is awesome!

- 1995
- 150 MHz
- RISC emulating CISC
- 256 KB L2 cache integrated!
- branch prediction
- out-of-order execution
The future is going to be fast:
The future is going to be fast:

- Apple A12 Bionic (iPhone X): 16 KB pages → 128 KB caches
A Christmas Carol

Spectres of the Past
Side-Channel Attacks

- Bug-free software does not mean safe execution
Side-Channel Attacks

- Bug-free software does not mean safe execution
- Information leaks due to underlying hardware
Side-Channel Attacks

- Bug-free software does not mean safe execution
- Information leaks due to underlying hardware
- Exploit leakage through side-effects
Side-Channel Attacks

- Bug-free software does not mean safe execution
- Information leaks due to underlying hardware
- Exploit leakage through side-effects

Power consumption
Execution time
CPU caches
Instruction Set Architecture (ISA) is an abstract model of a computer (x86, ARMv8, SPARC, ...)
• Instruction Set Architecture (ISA) is an abstract model of a computer (x86, ARMv8, SPARC, …)
• **Interface** between hardware and software
• Instruction Set Architecture (ISA) is an abstract model of a computer (x86, ARMv8, SPARC, ...)  
• **Interface** between hardware and software  
• Microarchitecture is an ISA implementation
Architecture and Microarchitecture

- Instruction Set Architecture (ISA) is an abstract model of a computer (x86, ARMv8, SPARC, ...)
- Interface between hardware and software
- Microarchitecture is an ISA implementation
Microarchitectural Elements

- Modern CPUs contain multiple microarchitectural elements
Microarchitectural Elements

- Modern CPUs contain multiple microarchitectural elements

Caches and buffer

Predictor
Modern CPUs contain multiple microarchitectural elements:

- Caches and buffer
- Predictor

*Transparent* for the programmer
Microarchitectural Elements

- Modern CPUs contain multiple microarchitectural elements
- Transparent for the programmer
- Timing optimizations $\rightarrow$ side-channel leakage
Food Cache

8
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
What could possibly go wrong with \texttt{<insert x86 instruction here>}?

Side effects include side-channel attacks and bypassing kernel ASLR

\textcopyright Clémentine Maurice and Moritz Lipp

---

What could possibly go wrong with \texttt{<insert x86 instruction here>}?

Clémentine Maurice, Moritz Lipp

December 2016—33rd Chaos Communication Congress
printf("%d", i);
printf("%d", i);
CPU Cache

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

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

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

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

[Diagram of CPU cache with 'Cache miss' and 'Request' and 'Response']
CPU Cache

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

Cache miss

Cache hit

Request

Response
CPU Cache

DRAM access, slow

printf("%d", i);

printf("%d", i);

Cache miss

Cache hit

Request

Response

DRAM access, slow
CPU Cache

DRAM access, slow

printf("%d", i);

Cache miss

printf("%d", i);

Cache hit

No DRAM access, much faster

Request

Response

DRAM access, slow
Caching speeds up Memory Accesses

Access time [CPU cycles]

Number of accesses

Cache Hits
Caching speeds up Memory Accesses

Access time [CPU cycles]

Number of accesses

Cache Hits

Cache Misses
Flush+Reload

Attacker

flush
access

Shared Memory

Victim

access
Flush+Reload

Attacker
flush
access

Shared Memory

Cached

Victim
access

Cached

Shared Memory
Flush+Reload

Attacker

Shared Memory

Flush

access

Victim

Shared Memory

access
Flush+Reload

Attacker

flush
access

Shared Memory

Victim
access
Flush+Reload

Attacker
flush
access

Shared Memory

Victim
access
Flush+Reload

Attacker
flush
access

Shared Memory

Victim
access
Flush+Reload

Attacker
flush
access

Shared Memory

Victim
access
Flush+Reload

Attacker

flush

access

Shared Memory

Victim

access

Victim accessed (fast)

Victim did not access (slow)
• Just by looking at cache hits/misses, we can ...
Attacks

- Just by looking at cache hits/misses, we can ...
  - Leak AES keys from the cache
Attacks

- Just by looking at cache hits/misses, we can ...
  - Leak AES keys from the cache
  - Leak keystroke timings via the cache
Attacks

- Just by looking at cache hits/misses, we can ...
  - Leak **AES keys** from the cache
  - Leak **keystroke timings** via the cache
  - **Covertly send data** through the cache
Attacks

- Just by looking at cache hits/misses, we can ...
  - Leak **AES keys** from the cache
  - Leak **keystroke timings** via the cache
  - **Covertly send data** through the cache
- Browser, Cloud, TEEs, ...
shell@zeroflt:e:/data/local/tmp $ ./keyboard_spy -c 0
More and More Performance

The future is going to be fast:
- Apple A12 Bionic (iPhone X): 16 KB pages → 128 KB caches
The future is going to be fast:

- Apple A12 Bionic (iPhone X): 16 KB pages → 128 KB caches
- Intel → more out-of-order parallelism
A Christmas Carol

Spectres of the Present
int width = 10, height = 5;

float diagonal = sqrt(width * width
                        + height * height);
int area = width * height;

printf("Area %d x %d = %d\n", width, height, area);
```c
int width = 10, height = 5;
float diagonal = sqrt(width * width + height * height);
int area = width * height;
printf("Area %d x %d = %d\n", width, height, area);
```
Out-of-Order Execution

Instructions are
• fetched and decoded in the **front-end**
Instructions are

- fetched and decoded in the **front-end**
- dispatched to the **backend**
Instructions are

- fetched and decoded in the **front-end**
- dispatched to the **backend**
- processed by **individual execution units**
Building the Code

- An experiment

```c
*(volatile char*) 0;
array[84 * 4096] = 0;
```
Building the Code

- An experiment

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

- \textbf{volatile because compiler was not happy}

\[
\text{warning: statement with no effect [-Wunused-value]} \\
*\text{(char} \*) 0;
\]
Building the Code

- **An experiment**

  ```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
Building the Code

- Flush+Reload over all pages of the array

- "Unreachable" code line was actually executed
Building the Code

- 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.
Building the Code

• Out-of-order instructions leave microarchitectural traces
  • We can see them for example in the cache
Building the Code

- Out-of-order instructions leave microarchitectural traces
  - We can see them for example in the cache
  - We call them transient instructions
Out-of-order instructions leave microarchitectural traces
  - We can see them for example in the cache
  - We call them transient instructions
  - Execution indirectly observable
Loading an address
Loading an address
Loading an address
Loading an address
Loading an address
Loading an address
Add another **layer of indirection** to test

```c
char data = *(char*) 0xffffffff81a000e0;
array[data * 4096] = 0;
```
• Add another layer of indirection to test

```c
char data = *(char*) 0xfffffffff81a00e0;
array[data * 4096] = 0;
```

• Then check if any part of array is cached
Building the Code

- **Flush•Reload over all pages of the array**

- **Index of cache hit reveals data**
Building the Code

- Flush+Reload over all pages of the array

![Graph showing access time in cycles against page number]

- Index of cache hit reveals data
- Permission check fails sometimes
• Kernel addresses in user space are a problem
• Kernel addresses in user space are a problem
• Why don’t we take the kernel addresses...
• ...and remove them if not needed?
Meltdown Mitigation

- ...and remove them if not needed?
- User accessible check in hardware is not reliable
Meltdown-MS Mitigation

- Unmap the kernel in user space
Meltdown-US Mitigation

- Unmap the kernel in user space
- Kernel addresses are then no longer present
Meltdown-US Mitigation

- Unmap the kernel in user space
- Kernel addresses are then **no longer present**
- Memory which is not mapped **cannot be accessed at all**
Meltdown-P (aka Foreshadow-NG)

Page Table

- PTE 0
- PTE 1
- PTE #PTI
- PTE 511

L1 Cache
Meltdown-P (aka Foreshadow-NG)

Page Table

- PTE 0
- PTE 1
- PTE #PTI
- PTE 511

Present

L1 Cache
Meltdown-P (aka Foreshadow-NG)

Page Table

- PTE 0
- PTE 1
- ...
- PTE #PTI
- ...
- PTE 511

Guest Physical to Host Physical

present

LI Cache
Meltdown-P (aka Foreshadow-NG)

Page Table
- PTE 0
- PTE 1
- ...
- PTE #PTI
- PTE 511

Guest Physical to Host Physical

Physical Page

L1 Cache

L1 lookup with physical address
Meltdown-P (aka Foreshadow-NG)

Page Table

- PTE 0
- PTE 1
- ...
- PTE #PTI
- ...
- PTE 511

not present

L1 Cache
Meltdown-P (aka Foreshadow-NG)

Page Table

PTE 0
PTE 1
...
PTE #PTI
...
PTE 511

not present

L1 lookup with virtual address

L1 Cache
The future is going to be fast:

- Apple A12 Bionic (iPhone X): 16 KB pages → 128 KB caches
- Intel → more ports, more parallelism, larger reorder buffer
More and More Performance

The future is going to be fast:

- Apple A12 Bionic (iPhone X): 16 KB pages → 128 KB caches
- Intel → more ports, more parallelism, larger reorder buffer
- AMD → perceptron-based prediction mechanisms
robm@homebox ~ $ sudo su
Password: robm
is not in the sudoers file. This incident will be reported.
robm@homebox ~ $ !

Hey — Who doessudo report these "incidents" to?

You know, I've never checked.

https://xkcd.com/838/
Let us get rid of bottlenecks
Speculative Execution

Use the naughty/nice list of last year
Finally, check predictions with list of this year
Speculative Execution

Throwing away wrongly manufactured presents
Speculative Execution

Correct predictions result in free time
index = 0;

char* data = "textKEY";

if (index < 4)
index = 0;

char* data = "textKEY";

if (index < 4)
   Prediction
else

LUT[data[index] * 4096] 0
Spectre-PHT (aka Spectre Variant I)

```c
index = 0;
char* data = "textKEY";
if (index < 4)
    LUT[data[index] * 4096]
else
    Prediction
Speculate
LUT[0]
```

Prediciton
Speculate
index = 0;

char* data = "textKEY";

if (index < 4) {
    // Execute
    LUT[data[index] * 4096]
} else {
    Prediction
    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)
else

LUT[data[index] * 4096] 0
Spectre-PHT (aka Spectre Variant I)

index = 1;

char* data = "textKEY";

if (index < 4)

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

char* data = "textKEY";

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

LUT[0]
index = 2;

cchar* data = "textKEY";

if (index < 4)

LUT[data[index] * 4096]

else

Prediction 0
index = 2;

char* data = "textKEY";

if (index < 4)
index = 2;

char* data = "textKEY";

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

Speculate

then

Index 'x'

else

0
index = 2;

char* data = "textKEY";

if (index < 4)
    Prediction
else
    0
Spectre-PHT (aka Spectre Variant I)

```c
index = 3;

char* data = "textKEY";

if (index < 4)
```

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
else
    Prediction

LUT[data[index] * 4096]
index = 3;

char* data = "textKEY";

if (index < 4)

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

char* data = "textKEY";

if (index < 4)

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

char* data = "textKEY";

if (index < 4)

LUT[data[index] * 4096] 0
index = 4;
char* data = "textKEY";

if (index < 4)
    Speculate
else
    Prediction

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

char* data = "textKEY";

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

char* data = "textKEY";

if (index < 4)

LUT[data[index] * 4096]
Spectre-PHT (aka Spectre Variant II)

```c
index = 5;

char* data = "textKEY";

if (index < 4)

LUT[data[index] * 4096]

else

Prediction

0
```
Spectre-PHT (aka Spectre Variant 1)

index = 5;

char* data = "textKEY";

if (index < 4)

else
Spectre-PHT (aka Spectre Variant I)

index = 5;

char* data = "textKEY";

if (index < 4)

LUT[data[index] * 4096]

else

Execute 0
index = 6;

char* data = "textKEY";

if (index < 4)

else
index = 6;

char* data = "textKEY";

if (index < 4)

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

char* data = "textKEY";

if (index < 4)

LUT[data[index] * 4096]

else

0
spectre-PHT (aka spectre variant I)

index = 6;

char* data = "textKEY";

if (index < 4)

LUT[data[index] * 4096]

else

execute

0
Animal* a = bird;

a->move()

fly()

swim()

Prediction

LUT[data[index] * 4096]

0
Animal* a = bird;

a->move()

fly()  

LUT[data[index] * 4096]

swim()  

Prediction  

Speculate  

swim()  

0
Animal* a = bird;

a->move()

fly()  swim()  swim()  Prediction

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

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

fly()
fly()
swim()

Prediction

LUT[data[index] * 4096] 0
```cpp
Animal* a = bird;
a->move();
```

**Speculate**

- `fly()`

**LUT**

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

**Prediction**

- `fly()`, `swim()`, `0`
Animal* a = bird;

a->move()

fly()

fly()

swim()

Prediction

LUT[data[index] * 4096]

0
Animal* a = fish;

a->move()

fly()

Prediction

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

a->move();
Spectre-BTB (aka Spectre Variant 2)

Animal* a = fish;

a->move()

fly()  Prediction  fly()  swim()

LUT[data[index] * 4096]  0
```cpp
Animal* a = fish;
a->move();
```

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

a->move()

fly()

swim()

Prediction

LUT[data[index] * 4096]

0
function()

...
Spectre-RSB

Victim

\[
\text{reg} = \text{secret} \\
\text{call function(SHORT)}
\]

Typing: 

\[
\text{reg} = \text{dummy}
\]

RSB

&victim

function()
Spectre-RSB

Victim

reg = secret
call function(SHORT)

RSB

&attacker
&victim

Attacker

reg = dummy
call function(LONG)
data[reg * 4096]
Spectre-RSB

Victim

\[ \text{reg} = \text{secret} \]
\[ \text{call function(SHORT)} \]

Attacker

\[ \text{reg} = \text{dummy} \]
\[ \text{call function(LONG)} \]
\[ \text{data[reg * 4096]} \]

RSB

\&\text{attacker}
\&\text{victim}
Spectre-RSB

Victim

reg = secret
call function(SHORT)

Attacker

reg = dummy
call function(LONG)
data[reg * 4096]

RSB

&victim
Victim

reg = secret
call function(SHORT)

Attacker

reg = dummy
call function(LONG)
data[reg * 4096]
Meltdown vs. Spectre

operation #n

time
Meltdown vs. Spectre

operation #n

prediction

time
Meltdown vs. Spectre

- operation \#n
- prediction
- predict CF/DF
- operation \#n+2

Time
Meltdown vs. Spectre

- Operation #n
- Prediction
- Operation #n+2
- Transient execution
- Possibly architectural
- CF/DF
Meltdown vs. Spectre

![Diagram showing the relationship between operations and time]

- Operation #n
- Retire
- Prediction
- Predict CF/DF
- Operation #n+2
- Possibly architectural
- Transient execution

Time
Meltdown vs. Spectre

operation \#n

prediction

operation \#n+2

transient execution

flush pipeline on wrong prediction

possibly architectural
Meltdown vs. Spectre

- Operation #n
- Retire
- Time
- Prediction
- Retire
- Operation #n+2
- Retire
- Transient execution
- Time

Flush pipeline on wrong prediction

Possibly architectural
Meltdown vs. Spectre

- Operation #n
- Retire
- Flush pipeline on wrong prediction
- Prediction
- Operation #n+2
- Transient execution
- Possibly architectural
- Data

Time
Meltdown vs. Spectre

Operation #n

Prediction

Operation #n+2

Flush pipeline on wrong prediction

Transient execution

Data

Operation #n+2

Data dependency

Possibly architectural
Meltdown vs. Spectre

Operation #n

Retire

Prediction

Operation #n+2

Transient execution

Possible architectural

Flush pipeline on wrong prediction

Operation #n

Retire

Data

Exception

Data dependency

Operation #n+2

Transient execution

Possible architectural
Meltdown vs. Spectre

Operation #n retire

Prediction

Operation #n+2 retire

Flush pipeline on wrong prediction

Operation #n retire

Data dependency

Exception

Data

Operation #n+2 retire

Possibly architectural

Transient execution

Possibly architectural

Transient execution
Meltdown vs. Spectre

Operation #n retirement

Prediction

Operation #n+2 retirement

Flush pipeline on wrong prediction

Time

Data

Exception

Meltdown

Data dependency

Operation #n+2 retirement

Transient execution

Possibly architectural

Possibly architectural

Transient execution
Meltdown vs. Spectre

Operation #n

Prediction

Operation #n+2

Transient execution

Flush pipeline on wrong prediction

Operation #n

Data

Exception

Operation #n+2

Data dependency

Meltdown

Transient execution

Possibly architectural

Possibly architectural
More and More Performance

The future is going to be fast:

- Apple A12 Bionic (iPhone X): 16 KB pages → 128 KB caches
- Intel → more ports, more parallelism, larger reorder buffer
- AMD → perceptron-based prediction mechanisms
A Christmas Carol

Spectres of The Future
• Protection key for a group of pages
• Protection key for a **group of pages**
• 4 bits in PTE **identify key** for protected memory regions
• Protection key for a **group of pages**
• 4 bits in PTE **identify key** for protected memory regions
• **Quick update** of access rights
• Protection keys are lazily enforced
- Protection keys are **lazily enforced**
- Protected value is forwarded to transient instructions
Meltdown-BR

- x86 provides dedicated instruction raising #BR exception if bound-range is exceeded
• x86 provides dedicated instruction raising #BR exception if bound-range is exceeded
• Data used in transient execution
• x86 provides dedicated instruction raising #BR exception if bound-range is exceeded
• Data used in transient execution
• Attacker determines accessed cache line using Flush+Reload
From: Tom Lendacky <tom.lendacky@amd.com>

Subject: [PATCH] x86/cpu, x86/pti: Do not enable PTI on AMD processors

Date: Tue, 26 Dec 2017 23:43:54 -0600

AMD processors are not subject to the types of attacks that the kernel page table isolation feature protects against. The AMD microarchitecture does not allow memory references, including speculative references, that access higher privileged data when running in a lesser privileged mode when that access would result in a page fault.

Disable page table isolation by default on AMD processors by not setting the X86_BUG_CPU_INSECURE feature, which controls whether X86_FEATURE_PTI is set.

Signed-off-by: Tom Lendacky <tom.lendacky@amd.com>

---
arch/x86/kernel/cpu/common.c | 4 ++-
1 file changed, 2 insertions(+), 2 deletions(-)

diff --git a/arch/x86/kernel/cpu/common.c b/arch/x86/kernel/cpu/common.c
index c47de4e..7d9e3b0 100644
--- a/arch/x86/kernel/cpu/common.c
+++ b/arch/x86/kernel/cpu/common.c
@@ -923,8 +923,8 @@ static void __init early_identify_cpu(struct cpuinfo_x86 *c)
    setup_force_cpu_cap(X86_FEATURE_ALWAYS);

- /* Assume for now that ALL x86 CPUs are insecure */
-       setup_force_cpu_bug(X86_BUG_CPU_INSECURE);
+ if (c->x86_vendor != X86_VENDOR_AMD)
+       setup_force_cpu_bug(X86_BUG_CPU_INSECURE);
    fpu_init_system(c);
From: Tom Lendacky <tomlendacky@angstrom.com>

Subject: [PATCH] x86/cpu, x86/pti: Do not enable PTI on AMD processors

Date: Tue, 26 Dec 2017 23:43:54 -0600

AMD processors are not subject to the types of attacks that the kernel page table isolation feature protects against. The AMD microarchitecture does not allow memory references, including speculative references, that access higher privileged data when running in a lesser privileged mode when that access would result in a page fault.

Disable page table isolation by default on AMD processors by not setting the X86_BUG_CPU_INSECURE feature, which controls whether X86_FEATURE PTI is set.

Signed-off-by: Tom Lendacky <tomlendacky@angstrom.com>

---
arch/x86/kernel/cpu/common.c | 4 ++-
1 file changed, 2 insertions(+), 2 deletions(-)

diff --git a/arch/x86/kernel/cpu/common.c b/arch/x86/kernel/cpu/common.c
index c4f7e4e..7d9e3b0 100644
--- a/arch/x86/kernel/cpu/common.c
+++ b/arch/x86/kernel/cpu/common.c
@@ -923,8 +923,8 @@ static void __init early_identify_cpu(struct cpuinfo_x86 *c)
    setup_force_cpu_cap(X86_FEATURE ALWAYS);

    /* Assume for now that ALL x86 CPUs are insecure */
-   setup_force_cpu_bug(X86_BUG_CPU_INSECURE);
+   if (c->vendor != X86_VENDOR_AMD)
+      setup_force_cpu_bug(X86_BUG_CPU_INSECURE);

    fpu_init_system(c);
From: Tom Lendacky <tom.lendacky@amd.com>
Subject: [PATCH] x86/cpu, x86/pti: Do not enable PTI on AMD processors
Date: Tue, 26 Dec 2017 23:43:54 -0600

AMD processors are not subject to the types of attacks that the kernel page table isolation feature protects against. The AMD microarchitecture does not allow memory references, including speculative references, that access higher privileged data when running in a lesser privileged mode when that access would result in a page fault.

Disable page table isolation by default on AMD processors by not setting the X86_BUG_CPU_INSECURE feature, which controls whether X86_FEATURE_PTI is set.

Signed-off-by: Tom Lendacky <tom.lendacky@amd.com>

---
arch/x86/kernel/cpu/common.c | 4 ++-
1 file changed, 2 insertions(+), 2 deletions(-)

diff --git a/arch/x86/kernel/cpu/common.c b/arch/x86/kernel/cpu/common.c
index c47de4e..7d9e3b0 100644
--- a/arch/x86/kernel/cpu/common.c
+++ b/arch/x86/kernel/cpu/common.c
@@ -923,8 +923,8 @@ static void __init early_identify_cpu(struct cpuinfo_x86 *c)

 setup_force_cpu_cap(X86_FEATURE_ALWAYS);

- /* Assume for now that ALL x86 CPUs are insecure */
- setup_force_cpu_bug(X86_BUG_CPU_INSECURE);
+ if (c->vendor != X86_VENDOR_AMD)
+ setup_force_cpu_bug(X86_BUG_CPU_INSECURE);

 fpu_init_system(c);
Meltdown-BR

- x86 provides dedicated instruction raising #BR exception if bound-range is exceeded
- Data used in transient execution
- Attacker determines accessed cache line using Flush+Reload
- First Meltdown-type attack on AMD
## Vulnerable Vendors

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>ARM</td>
<td>●</td>
<td>O</td>
<td>●</td>
<td>_</td>
<td>●</td>
<td>_</td>
<td>_</td>
<td>_</td>
<td>_</td>
<td>_</td>
<td>_</td>
<td>_</td>
<td>_</td>
<td>_</td>
</tr>
<tr>
<td>AMD</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>_</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
</tbody>
</table>
Meltdown Defense Categorization

Meltdown defenses in 2 categories:
Meltdown Defense Categorization

Meltdown defenses in 2 categories:

D1 Architecturally inaccessible
data is also microarchitecturally inaccessible
Meltdown Defense Categorization

Meltdown defenses in **2 categories:**

**D1** Architecturally inaccessible data is also microarchitecturally inaccessible

**D2** Preventing occurrence of faults
Meltdown-P Mitigation

Clear physical address field of unmapped PTEs

Flush L1 upon switching protection domains
Clear physical address field of unmapped PTEs
Meltdown-P Mitigation

**Clear** physical address field of unmapped PTEs

**Flush** L1 upon switching protection domains
Transient Execution Attacks: Classification

Transient cause?
Transient Execution Attacks: Classification

- Spectre-type
- Meltdown-type

Transient cause?
- prediction
- fault
Transient Execution Attacks: Classification

Spectre-type
- Spectre-PHT
- Spectre-BTB
- Spectre-RSB
- Spectre-STL

Meltdown-type

Transient cause?
prediction
fault
Spectre: Mistraining Strategies

Victim

same address space/in place

Spectre-vulnerable branch
Spectre: Mistraining Strategies

- Same address space / out of place
- Same address space / in place

Victim

Aliased branch

Address collision

Spectre-vulnerable branch
Spectre: Mistraining Strategies

- same address space/out of place
- same address space/in place

Victim

Aliased branch

Address collision

Spectre-vulnerable branch

Shared Branch Prediction State
Spectre: Mistraining Strategies

Victim

same address space/out of place

Aliased branch

same address space/in place

Spectre-vulnerable branch

Attacker

Shared Branch Prediction State

Address collision
Spectre: Mistraining Strategies

Victim

Aliased branch

Address collision

Spectre-vulnerable branch

Attacker

Same address

cross address space/in place

same address space/out of place

Same address space/in place

Shared Branch Prediction State
Spectre: Mistraining Strategies

Victim

Aliased branch

Spectre-vulnerable branch

Attacker

Aliased address

Same address

Shared Branch Prediction State

same address space/
out of place

cross address space/
out of place

same address space/
in place

cross address space/
in place

Address collision
Spectre Mistraining: Vulnerable Vendors

Method

Attack

Spectre-PHT
Spectre-BTB
Spectre-RSB
Spectre-STL

Intel
- same-address-space in-place
- out-of-place
- cross-address-space in-place
- out-of-place

ARM
- same-address-space in-place
- out-of-place
- cross-address-space in-place
- out-of-place

AMD
- same-address-space in-place
- out-of-place
- cross-address-space in-place
- out-of-place
## Spectre Mistraining: Vulnerable Vendors

<table>
<thead>
<tr>
<th>Method</th>
<th>Attack</th>
<th>Spectre-PHT</th>
<th>Spectre-BTB</th>
<th>Spectre-RSB</th>
<th>Spectre-STL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel same-address-space</td>
<td>in-place</td>
<td>★</td>
<td>★</td>
<td>★</td>
<td>★</td>
</tr>
<tr>
<td>Intel cross-address-space</td>
<td>out-of-place</td>
<td>★</td>
<td>★</td>
<td>★</td>
<td>★</td>
</tr>
<tr>
<td></td>
<td>in-place</td>
<td>★</td>
<td>★</td>
<td>★</td>
<td>★</td>
</tr>
<tr>
<td></td>
<td>out-of-place</td>
<td>★</td>
<td>★</td>
<td>★</td>
<td>★</td>
</tr>
<tr>
<td>Method</td>
<td>Attack</td>
<td>Spectre-PHT</td>
<td>Spectre-BTB</td>
<td>Spectre-RSB</td>
<td>Spectre-STL</td>
</tr>
<tr>
<td>----------------</td>
<td>-----------------</td>
<td>-------------</td>
<td>-------------</td>
<td>-------------</td>
<td>-------------</td>
</tr>
<tr>
<td>Intel</td>
<td>same-address-space</td>
<td>in-place</td>
<td>★</td>
<td>★</td>
<td>•</td>
</tr>
<tr>
<td></td>
<td></td>
<td>out-of-place</td>
<td>★</td>
<td>★</td>
<td>•</td>
</tr>
<tr>
<td></td>
<td>cross-address-space</td>
<td>in-place</td>
<td>★</td>
<td>★</td>
<td>•</td>
</tr>
<tr>
<td></td>
<td></td>
<td>out-of-place</td>
<td>★</td>
<td>★</td>
<td>•</td>
</tr>
<tr>
<td>ARM</td>
<td>same-address-space</td>
<td>in-place</td>
<td>★</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td></td>
<td></td>
<td>out-of-place</td>
<td>★</td>
<td>•</td>
<td>O</td>
</tr>
<tr>
<td></td>
<td>cross-address-space</td>
<td>in-place</td>
<td>★</td>
<td>•</td>
<td>O</td>
</tr>
<tr>
<td></td>
<td></td>
<td>out-of-place</td>
<td>★</td>
<td>•</td>
<td>O</td>
</tr>
</tbody>
</table>
## Spectre Mistraining: Vulnerable Vendors

<table>
<thead>
<tr>
<th>Method</th>
<th>Attack</th>
<th>Spectre-PHT</th>
<th>Spectre-BTB</th>
<th>Spectre-RSB</th>
<th>Spectre-STL</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Intel</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>same-address-space</td>
<td>in-place</td>
<td>★★</td>
<td>★★</td>
<td>★★</td>
<td>★★</td>
</tr>
<tr>
<td></td>
<td>out-of-place</td>
<td>☆☆☆☆</td>
<td>★☆☆☆</td>
<td>☆☆☆☆</td>
<td>☆☆☆☆</td>
</tr>
<tr>
<td>cross-address-space</td>
<td>in-place</td>
<td>★☆☆☆</td>
<td>★☆☆☆</td>
<td>☆☆☆☆</td>
<td>☆☆☆☆</td>
</tr>
<tr>
<td></td>
<td>out-of-place</td>
<td>☆☆☆☆</td>
<td>☆☆☆☆</td>
<td>☆☆☆☆</td>
<td>☆☆☆☆</td>
</tr>
<tr>
<td><strong>ARM</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>same-address-space</td>
<td>in-place</td>
<td>★☆☆☆</td>
<td>★☆☆☆</td>
<td>☆☆☆☆</td>
<td>☆☆☆☆</td>
</tr>
<tr>
<td></td>
<td>out-of-place</td>
<td>☆☆☆☆</td>
<td>☆☆☆☆</td>
<td>☆☆☆☆</td>
<td>☆☆☆☆</td>
</tr>
<tr>
<td>cross-address-space</td>
<td>in-place</td>
<td>★☆☆☆</td>
<td>★☆☆☆</td>
<td>☆☆☆☆</td>
<td>☆☆☆☆</td>
</tr>
<tr>
<td></td>
<td>out-of-place</td>
<td>☆☆☆☆</td>
<td>☆☆☆☆</td>
<td>☆☆☆☆</td>
<td>☆☆☆☆</td>
</tr>
<tr>
<td><strong>AMD</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>same-address-space</td>
<td>in-place</td>
<td>★☆☆☆☆</td>
<td>★☆☆☆☆</td>
<td>☆☆☆☆☆</td>
<td>☆☆☆☆☆</td>
</tr>
<tr>
<td></td>
<td>out-of-place</td>
<td>☆☆☆☆☆</td>
<td>☆☆☆☆☆</td>
<td>☆☆☆☆☆</td>
<td>☆☆☆☆☆</td>
</tr>
<tr>
<td>cross-address-space</td>
<td>in-place</td>
<td>★☆☆☆☆</td>
<td>★☆☆☆☆</td>
<td>☆☆☆☆☆</td>
<td>☆☆☆☆☆</td>
</tr>
<tr>
<td></td>
<td>out-of-place</td>
<td>☆☆☆☆☆</td>
<td>☆☆☆☆☆</td>
<td>☆☆☆☆☆</td>
<td>☆☆☆☆☆</td>
</tr>
</tbody>
</table>
Super Effective Solution: Drilling template

Drilling template (@kreon_nrw)
Spectre Defense Categorization

Spectre defenses in 3 categories:

C1: Mitigate or reduce accuracy of covert channels
C2: Mitigate or abort speculation
C3: Ensure secret cannot be reached
Spectre Defense Categorization

Spectre defenses in **3 categories:**

- C1 Mitigate or reduce accuracy of covert channels
- C2 Mitigate or abort speculation
- C3 Ensure secret cannot be reached
Spectre Defense Categorization

Spectre defenses in 3 categories:

C1 Mitigate or reduce accuracy of covert channels

C2 Mitigate or abort speculation
Spectre Defense Categorization

Spectre defenses in **3 categories**:

**C1** Mitigate or reduce accuracy of covert channels

**C2** Mitigate or abort speculation

**C3** Ensure secret cannot be reached
<table>
<thead>
<tr>
<th>Defense</th>
<th>InvisiSpec</th>
<th>SafeSpec</th>
<th>DAWG</th>
<th>Taint Tracking</th>
<th>Timer Reduction</th>
<th>RSB Stuffing</th>
<th>Retpoline</th>
<th>SLH</th>
<th>YSNB</th>
<th>IBRS</th>
<th>STIPB</th>
<th>IBPB</th>
<th>Serialization</th>
<th>Sloth</th>
<th>Poison Value</th>
<th>Index Masking</th>
<th>Site Isolation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cache</td>
<td>● ● ● ● ●</td>
<td>○ ○ ○ ○ ○</td>
<td>○ ○ ○ ○ ○</td>
<td>○ ○ ○ ○ ○</td>
<td>○ ○ ○ ○ ○</td>
<td>●</td>
<td>○ ○ ○ ○ ○</td>
<td>○ ○ ○</td>
<td>○ ○ ○</td>
<td>○ ○ ○</td>
<td>○ ○ ○</td>
<td>○ ○ ○</td>
<td>○ ○ ○ ○ ○ ○</td>
<td>○ ○ ○</td>
<td>○ ○ ○ ○ ○ ○</td>
<td>○ ○ ○ ○ ○ ○</td>
<td>○ ○ ○ ○ ○ ○</td>
</tr>
<tr>
<td>TLB</td>
<td>○ ● ○ ○ ○</td>
<td>○ ○ ○ ○ ○</td>
<td>○ ○ ○ ○ ○</td>
<td>○ ○ ○ ○ ○</td>
<td>○ ○ ○ ○ ○</td>
<td>○ ○ ○ ○ ○</td>
<td>○ ○ ○ ○ ○</td>
<td>○ ○ ○</td>
<td>○ ○ ○</td>
<td>○ ○ ○</td>
<td>○ ○ ○</td>
<td>○ ○ ○</td>
<td>○ ○ ○ ○ ○ ○</td>
<td>○ ○ ○</td>
<td>○ ○ ○ ○ ○ ○</td>
<td>○ ○ ○ ○ ○ ○</td>
<td>○ ○ ○ ○ ○ ○</td>
</tr>
<tr>
<td>BTB</td>
<td>○ ○ ○ ○ ○</td>
<td>○ ○ ○ ○ ○</td>
<td>○ ○ ○ ○ ○</td>
<td>○ ○ ○ ○ ○</td>
<td>○ ○ ○ ○ ○</td>
<td>● ● ● ● ●</td>
<td>○ ○ ○ ○ ○</td>
<td>○ ○ ○</td>
<td>○ ○ ○</td>
<td>○ ○ ○</td>
<td>○ ○ ○</td>
<td>○ ○ ○</td>
<td>○ ○ ○ ○ ○ ○</td>
<td>○ ○ ○</td>
<td>○ ○ ○ ○ ○ ○</td>
<td>○ ○ ○ ○ ○ ○</td>
<td>○ ○ ○ ○ ○ ○</td>
</tr>
<tr>
<td>BHB</td>
<td>○ ○ ○ ○ ○</td>
<td>○ ○ ○ ○ ○</td>
<td>○ ○ ○ ○ ○</td>
<td>○ ○ ○ ○ ○</td>
<td>○ ○ ○ ○ ○</td>
<td>○ ○ ○ ● ●</td>
<td>○ ○ ○ ○ ○</td>
<td>○ ○ ○</td>
<td>○ ○ ○</td>
<td>○ ○ ○</td>
<td>○ ○ ○</td>
<td>○ ○ ○</td>
<td>○ ○ ○ ○ ○ ○</td>
<td>○ ○ ○</td>
<td>○ ○ ○ ○ ○ ○</td>
<td>○ ○ ○ ○ ○ ○</td>
<td>○ ○ ○ ○ ○ ○</td>
</tr>
<tr>
<td>PHT</td>
<td>○ ○ ○ ○ ○</td>
<td>○ ○ ○ ○ ○</td>
<td>○ ○ ○ ○ ○</td>
<td>○ ○ ○ ○ ○</td>
<td>○ ○ ○ ○ ○</td>
<td>○ ○ ○ ○ ○</td>
<td>○ ○ ○ ○ ○</td>
<td>○ ○ ○</td>
<td>○ ○ ○</td>
<td>○ ○ ○</td>
<td>○ ○ ○</td>
<td>○ ○ ○</td>
<td>○ ○ ○ ○ ○ ○</td>
<td>○ ○ ○</td>
<td>○ ○ ○ ○ ○ ○</td>
<td>○ ○ ○ ○ ○ ○</td>
<td>○ ○ ○ ○ ○ ○</td>
</tr>
<tr>
<td>RSB</td>
<td>○ ○ ○ ○ ○</td>
<td>○ ○ ○ ○ ○</td>
<td>○ ○ ○ ○ ○</td>
<td>○ ○ ○ ○ ○</td>
<td>○ ○ ○ ○ ○</td>
<td>○ ○ ○ ○ ○</td>
<td>○ ○ ○ ○ ○</td>
<td>○ ○ ○</td>
<td>○ ○ ○</td>
<td>○ ○ ○</td>
<td>○ ○ ○</td>
<td>○ ○ ○</td>
<td>○ ○ ○ ○ ○ ○</td>
<td>○ ○ ○</td>
<td>○ ○ ○ ○ ○ ○</td>
<td>○ ○ ○ ○ ○ ○</td>
<td>○ ○ ○ ○ ○ ○</td>
</tr>
<tr>
<td>AVX</td>
<td>○ ○ ○ ○ ○</td>
<td>○ ○ ○ ○ ○</td>
<td>○ ○ ○ ○ ○</td>
<td>○ ○ ○ ○ ○</td>
<td>○ ○ ○ ○ ○</td>
<td>○ ○ ○ ○ ○</td>
<td>○ ○ ○ ○ ○</td>
<td>○ ○ ○</td>
<td>○ ○ ○</td>
<td>○ ○ ○</td>
<td>○ ○ ○</td>
<td>○ ○ ○</td>
<td>○ ○ ○ ○ ○ ○</td>
<td>○ ○ ○</td>
<td>○ ○ ○ ○ ○ ○</td>
<td>○ ○ ○ ○ ○ ○</td>
<td>○ ○ ○ ○ ○ ○</td>
</tr>
<tr>
<td>FPU</td>
<td>○ ○ ○ ○ ○</td>
<td>○ ○ ○ ○ ○</td>
<td>○ ○ ○ ○ ○</td>
<td>○ ○ ○ ○ ○</td>
<td>○ ○ ○ ○ ○</td>
<td>○ ○ ○ ○ ○</td>
<td>○ ○ ○ ○ ○</td>
<td>○ ○ ○</td>
<td>○ ○ ○</td>
<td>○ ○ ○</td>
<td>○ ○ ○</td>
<td>○ ○ ○</td>
<td>○ ○ ○ ○ ○ ○</td>
<td>○ ○ ○</td>
<td>○ ○ ○ ○ ○ ○</td>
<td>○ ○ ○ ○ ○ ○</td>
<td>○ ○ ○ ○ ○ ○</td>
</tr>
</tbody>
</table>

Microarchitectural Element: Cache, TLB, BTB, BHB, PHT, RSB, AVX, FPU
Execution Ports

Category: C1, C2, C3
Site Isolation

- Each site executed in its own process
Site Isolation

- Each site executed in its own process
  → limits amount of data that is exposed
Site Isolation

• Each site executed in its own process
  → limits amount of data that is exposed
• Chrome 67: default, Firefox: work in progress
Serialization

- Insert instructions **stopping** speculation
Insert instructions stopping speculation:
- insert after every bounds check
Serialization

- Insert instructions **stopping** speculation
  → insert after **every** bounds check
- x86: LFENCE, ARM: CSDB with conditional selects or moves
• Make transient loads **invisible** in the cache hierarchy
InvisiSpec

- Make transient loads **invisible** in the cache hierarchy
  - all transient loads use a **speculative buffer**
InvisiSpec

- Make transient loads **invisible** in the cache hierarchy
- All transient loads use a **speculative buffer**
- Correct prediction: buffer content loaded into cache
- Make transient loads **invisible** in the cache hierarchy
  - all transient loads use a **speculative buffer**
- Correct prediction: buffer content loaded into cache
- Wrong prediction: transient load is **reverted**
Spectre: Defense Analysis

- InvisiSpec
- SafeSpec
- DAWG
- RSB Stuffing
- Retpoline
- Poison Value
- Index Masking
- Site Isolation
- SLH
- YSNB
- IBRS
- STIPB
- IBPB
- Serialization
- Taint Tracking
- Timer Reduction
- Sloth
- SSBD/SSBB

Intel: Spectre-PHT, Spectre-BTB, Spectre-RSB, Spectre-STL

ARM: Spectre-PHT, Spectre-BTB, Spectre-RSB, Spectre-STL

AMD: Spectre-PHT, Spectre-BTB, Spectre-RSB, Spectre-STL
## Spectre: Defense Analysis

<table>
<thead>
<tr>
<th>Attack</th>
<th>Defense</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>InvisiSpec</td>
</tr>
<tr>
<td>Intel</td>
<td>Spectre-PHT</td>
</tr>
<tr>
<td></td>
<td>Spectre-BTB</td>
</tr>
<tr>
<td></td>
<td>Spectre-RSB</td>
</tr>
<tr>
<td></td>
<td>Spectre-STL</td>
</tr>
</tbody>
</table>
# Spectre: Defense Analysis

<table>
<thead>
<tr>
<th>Attack</th>
<th>Defense</th>
<th>InvisiSpec</th>
<th>SafeSpec</th>
<th>DAWG</th>
<th>RSB Stuffing</th>
<th>Retpoline</th>
<th>Poison Value</th>
<th>Index Masking</th>
<th>Site Isolation</th>
<th>SLH</th>
<th>YSNB</th>
<th>IBRS</th>
<th>STPB</th>
<th>IBPB</th>
<th>Serialization</th>
<th>Taint Tracking</th>
<th>Timer Reduction</th>
<th>Sloth</th>
<th>SSBD/SSBB</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Intel</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Spectre-PHT</td>
<td></td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td></td>
<td>□</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>□</td>
<td>□</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>□</td>
</tr>
<tr>
<td>Spectre-BTB</td>
<td></td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td></td>
<td>□</td>
<td></td>
<td></td>
<td>□</td>
<td></td>
<td>□</td>
<td>□</td>
<td>□</td>
<td></td>
<td></td>
<td></td>
<td>□</td>
<td>□</td>
</tr>
<tr>
<td>Spectre-RSB</td>
<td></td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td></td>
<td>□</td>
<td></td>
<td></td>
<td>□</td>
<td></td>
<td>□</td>
<td>□</td>
<td>□</td>
<td></td>
<td>□</td>
<td></td>
<td>□</td>
<td>□</td>
</tr>
<tr>
<td>Spectre-STL</td>
<td></td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td></td>
<td>□</td>
<td></td>
<td></td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td></td>
<td>□</td>
<td></td>
<td>□</td>
<td>□</td>
</tr>
<tr>
<td><strong>ARM</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Spectre-PHT</td>
<td></td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td></td>
<td>□</td>
<td></td>
<td></td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td></td>
<td>□</td>
<td></td>
<td>□</td>
<td>□</td>
</tr>
<tr>
<td>Spectre-BTB</td>
<td></td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td></td>
<td>□</td>
<td></td>
<td></td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td></td>
<td>□</td>
<td></td>
<td>□</td>
<td>□</td>
</tr>
<tr>
<td>Spectre-RSB</td>
<td></td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td></td>
<td>□</td>
<td></td>
<td></td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td></td>
<td>□</td>
<td></td>
<td>□</td>
<td>□</td>
</tr>
<tr>
<td>Spectre-STL</td>
<td></td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td></td>
<td>□</td>
<td></td>
<td></td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td></td>
<td>□</td>
<td></td>
<td>□</td>
<td>□</td>
</tr>
</tbody>
</table>
## Spectre: Defense Analysis

<table>
<thead>
<tr>
<th>Attack</th>
<th>Defense</th>
<th>Intel</th>
<th>ARM</th>
<th>AMD</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>InvisiSpec</td>
<td>Spectre-PHT</td>
<td>Spectre-PHT</td>
<td>Spectre-PHT</td>
</tr>
<tr>
<td></td>
<td>SafeSpec</td>
<td>Spectre-BTB</td>
<td>Spectre-BTB</td>
<td>Spectre-BTB</td>
</tr>
<tr>
<td></td>
<td>DAWG</td>
<td>Spectre-RSB</td>
<td>Spectre-RSB</td>
<td>Spectre-RSB</td>
</tr>
<tr>
<td></td>
<td>RSB Stuffing</td>
<td>Spectre-STL</td>
<td>Spectre-STL</td>
<td>Spectre-STL</td>
</tr>
<tr>
<td></td>
<td>Retpoline</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Poison Value</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Index Masking</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Site Isolation</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SLH</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>YSNB</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>IBRS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>STIPB</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>IBPB</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Serialization</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Taint Tracking</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Timer Reduction</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Sloth</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SSBD/SSBB</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
On Friday marked the release of the Linux 4.19.4 kernel as well as 4.14.83 and 4.9.139.

Greg Kroah-Hartman issued this latest round of stable point releases as basic maintenance updates. While these point releases don't tend to be too notable and generally go unmentioned on Phoronix, this round is worth pointing out since 4.19.4 and 4.14.83 are the releases that end up reverting the STIBP behavior that applied Single Thread Indirect Branch Predictors to all processes on supported systems. That is what was introduced in Linux 4.20 and then back-ported to the 4.19/4.14 LTS branches, which in turn hurt the performance a lot. So for now the code is removed.

As covered yesterday, there is improved STIBP code on the way for Linux 4.20 that by default just apply STIBP to SECCOMP threads and processes requesting it via prctl() but otherwise is off by default (that behavior can also be changed via kernel parameters).
On Friday marked the release of the Linux 4.19.4 kernel as well as 4.14.83 and 4.9.139.

Greg Kroah-Hartman issued this latest round of stable point releases as basic maintenance updates. While these point releases don't tend to be too notable and generally go unmentioned on Phoronix, this round is worth pointing out since 4.19.4 and 4.14.83 are the releases that end up reverting the STIBP behavior that applied Single Thread Indirect Branch Predictors to all processes on supported systems. That is what was introduced in Linux 4.20 and then back-ported to the 4.19/4.14 LTS branches, which in turn hurt the performance a lot. So for now the code is removed.

As covered yesterday, there is improved STIBP code on the way for Linux 4.20 that by default just apply STIBP to SECCOMP threads and processes requesting it via prctl() but otherwise is off by default (that behavior can also be changed via kernel parameters).
Linux 4.19.4 & 4.14.83 Released With STIBP Code Dropped

Written by Michael Larabel in Linux Kernel on 24 November 2018 at 09:00 AM EST. 6 Comments

On Friday marked the release of the Linux 4.19.4 kernel as well as 4.14.83 and 4.9.139.

Greg Kroah-Hartman issued this latest round of stable point releases as basic maintenance updates. While these point releases don't tend to be too notable and generally go unmentioned on Phoronix, this round is worth pointing out since 4.19.4 and 4.14.83 are the releases that end up reverting the STIBP behavior that applied Single Thread Indirect Branch Predictors to all processes on supported systems. That is what was introduced in Linux 4.20 and then back-ported to the 4.19/4.14 LTS branches, which in turn hurt the performance a lot. So for now the code is removed.

As covered yesterday, there is improved STIBP code on the way for Linux 4.20 that by default just apply STIBP to SECCOMP threads and processes requesting it via prctl() but otherwise is off by default (that behavior can also be changed via kernel parameters).
Linux 4.19.4 & 4.14.83 Released With STIBP Code Dropped

Written by Michael Larabel in Linux Kernel on 24 November 2018 at 09:00 AM EST. 6 Comments

On Friday marked the release of the Linux 4.19.4 kernel as well as 4.14.83 and 4.9.139.

Greg Kroah-Hartman issued this latest round of stable point releases as basic maintenance updates. While these point releases don't tend to be too notable and generally go unmentioned on Phoronix, this round is worth pointing out since 4.19.4 and 4.14.83 are the releases that end up reverting the STIBP behavior that applied Single Thread Indirect Branch Predictors to all processes on supported systems. That is what was introduced in Linux 4.20 and then back-ported to the 4.19/4.14 LTS branches, which in turn hurt the performance a lot. So for now the code is removed.

As covered yesterday, there is improved STIBP code on the way for Linux 4.20 that by default just apply STIBP to SECCOMP threads and processes requesting it via prctl() but otherwise is off by default (that behavior can also be be changed via kernel parameters).
Transient Execution Attacks: Classification

Transient cause?
Transient Execution Attacks: Classification

- Spectre-type
- Meltdown-type

- Transient cause?
- Prediction
- Fault
Transient Execution Attacks: Classification

- Spectre-type
  - Spectre-PHT
  - Spectre-BTB
  - Spectre-RSB
  - Spectre-STL

- Meltdown-type

Transient cause?

Microarchitectural buffer

Prediction

Fault
Transient Execution Attacks: Classification

Transient cause?

Spectre-type

- Spectre-PHT
  - Cross-address-space
  - Same-address-space
- Spectre-BTB
  - Cross-address-space
  - Same-address-space
- Spectre-RSB
  - Cross-address-space
  - Same-address-space
- Spectre-STL
  - Cross-address-space
  - Same-address-space

Microarchitectural buffer

Prediction

Fault

Meltdown-type
Transient Execution Attacks: Classification

**Transient cause?**

- Spectre-type
  - Spectre-PHT
  - Spectre-BTB
  - Spectre-RSB
  - Spectre-STL

**Microarchitectural buffer**

- Cross-address-space
  - Same-address-space

- Mistraining strategy

- In-place (IP) vs. out-of-place (OP)

**Fault**

- Meltdown-type
Transient Execution Attacks: Classification

- Spectre-type
  - Spectre-PHT
  - Spectre-BTB
  - Spectre-RSB
  - Spectre-STL

- Meltdown-type

- Microarchitectural buffer
  - Cross-address-space
    - Same-address-space
  - In-place (IP) vs. out-of-place (OP)
    - PHT-CA-IP
    - PHT-CA-OP
    - PHT-SA-IP
    - PHT-SA-OP
    - BTB-CA-IP
    - BTB-CA-OP
    - BTB-SA-IP
    - BTB-SA-OP
    - RSB-CA-IP
    - RSB-CA-OP
    - RSB-SA-IP
    - RSB-SA-OP

- Mistraining strategy
  - Prediction fault
Transient Execution Attacks: Classification

- **Spectre-type**
  - Spectre-PHT
  - Spectre-BTB
  - Spectre-RSB
  - Spectre-STL

- **Meltdown-type**
  - Meltdown-NM
  - Meltdown-PF
  - Meltdown-BR
  - Meltdown-GP

- **Microarchitectural buffer**
  - Cross-address-space
  - Same-address-space

- **Mistraining strategy**
  - In-place (IP) vs. out-of-place (OP)

- **Prediction**
  - Cross-address-space
  - Same-address-space

- **Fault type**
  - Cross-address-space
  - Same-address-space

- **Fault**
  - Cross-address-space
  - Same-address-space
Transient Execution Attacks: Classification

Transient cause?

Microarchitectural buffer

Spectre-type

Spectre-PHT

Spectre-BTB

Spectre-RSB

Spectre-STL

Meltdown-type

Meltdown-NM

Meltdown-AC

Meltdown-DE

Meltdown-PF

Meltdown-UD

Meltdown-SS

Meltdown-BR

Meltdown-GP

Prediction

Fault

Fault type

Cross-address-space vs. Same-address-space

Mistraining strategy

In-place (IP) vs. out-of-place (OP)

Meltdown-NM

Meltdown-AC

Meltdown-DE

Meltdown-PF

Meltdown-UD

Meltdown-SS

Meltdown-BR

Meltdown-GP

PHT-CA-IP

PHT-CA-OP

PHT-SA-IP

PHT-SA-OP

BTB-CA-IP

BTB-CA-OP

BTB-SA-IP

BTB-SA-OP

RSB-CA-IP

RSB-CA-OP

RSB-SA-IP

RSB-SA-OP
Transient Execution Attacks: Classification

**Transient cause?**
- Spectre-type
  - Spectre-PHT
  - Spectre-BTB
  - Spectre-RSB
  - Spectre-STL

**Microarchitectural buffer**
- Spectre-PHT
- Spectre-BTB
- Spectre-RSB
- Spectre-STL

**Fault type**
- Meltdown-NM
- Meltdown-AC
- Meltdown-DE
- Meltdown-PF
- Meltdown-UD
- Meltdown-SS
- Meltdown-BR
- Meltdown-GP

**Meltdown-type**
- Meltdown-NM
- Meltdown-AC
- Meltdown-DE
- Meltdown-PF
- Meltdown-UD
- Meltdown-SS
- Meltdown-BR
- Meltdown-GP

**Cross-address-space**
- Meltdown-US
- Meltdown-P
- Meltdown-RW
- Meltdown-MPX

**Same-address-space**
- Meltdown-US
- Meltdown-P
- Meltdown-RW
- Meltdown-MPX

**Prediction fault**
- Meltdown-US
- Meltdown-P
- Meltdown-RW
- Meltdown-MPX

**In-place (IP) vs. out-of-place (OP)**
- Meltdown-US
- Meltdown-P
- Meltdown-RW
- Meltdown-MPX

**Mistraining strategy**
- Meltdown-US
- Meltdown-P
- Meltdown-RW
- Meltdown-MPX
Transient Execution Attacks: Classification

Microarchitectural buffer:
- Spectre-type
  - Spectre-PHT
  - Spectre-BTB
  - Spectre-RSB
  - Spectre-STL
- Meltdown-type
  - Meltdown-NM
  - Meltdown-AC
  - Meltdown-DE
  - Meltdown-PF
  - Meltdown-UD
  - Meltdown-SS
  - Meltdown-BR
  - Meltdown-GP

Fault:
- Spectre-type
- Meltdown-type

Fault type:
- Cross-address-space
- Same-address-space

Mistraining strategy:
- In-place (IP) vs. out-of-place (OP)

Faults:
- PHT-CA-IP*
- PHT-CA-OP*
- PHT-SA-IP
- PHT-SA-OP*
- BTB-CA-IP
- BTB-CA-OP
- BTB-SA-IP*
- BTB-SA-OP*
- RSB-CA-IP
- RSB-CA-OP
- RSB-SA-IP
- RSB-SA-OP

Prediction:
- Meltdown-US
- Meltdown-P
- Meltdown-RW
- Meltdown-XD
- Meltdown-SM
- Meltdown-MPX
- Meltdown-BND

54
Transient Execution Attacks: Classification

**Transient cause?**

- Spectre-type
  - Spectre-PHT
  - Spectre-BTB
  - Spectre-RSB
  - Spectre-STL

**Spectre-PHT**
- Cross-address-space
- Same-address-space

**Spectre-BTB**
- Cross-address-space
- Same-address-space

**Spectre-RSB**
- Cross-address-space
- Same-address-space

**Spectre-STL**
- Cross-address-space
- Same-address-space

**Meltdown-type**
- Meltdown-NM
  - Meltdown-US
  - Meltdown-AC
  - Meltdown-P
  - Meltdown-DE
  - Meltdown-PF
  - Meltdown-RW
  - Meltdown-SS
  - Meltdown-BR
  - Meltdown-GP
  - Meltdown-MPX
  - Meltdown-BND

**Meltdown-NM**
- Meltdown-US
- Meltdown-AC
- Meltdown-DE
- Meltdown-PF
- Meltdown-RW
- Meltdown-SS
- Meltdown-BR
- Meltdown-GP
- Meltdown-MPX
- Meltdown-BND

**Meltdown-US**
- Meltdown-AC
- Meltdown-PF
- Meltdown-RW
- Meltdown-SS
- Meltdown-BR
- Meltdown-GP
- Meltdown-MPX
- Meltdown-BND

**Meltdown-AC**
- Meltdown-DE
- Meltdown-PF
- Meltdown-RW
- Meltdown-SS
- Meltdown-BR
- Meltdown-GP
- Meltdown-MPX
- Meltdown-BND

**Meltdown-DE**
- Meltdown-PF
- Meltdown-RW
- Meltdown-SS
- Meltdown-BR
- Meltdown-GP
- Meltdown-MPX
- Meltdown-BND

**Meltdown-PF**
- Meltdown-RW
- Meltdown-SS
- Meltdown-BR
- Meltdown-GP
- Meltdown-MPX
- Meltdown-BND

**Meltdown-RW**
- Meltdown-SS
- Meltdown-BR
- Meltdown-GP
- Meltdown-MPX
- Meltdown-BND

**Meltdown-SS**
- Meltdown-BR
- Meltdown-GP
- Meltdown-MPX
- Meltdown-BND

**Meltdown-BR**
- Meltdown-GP
- Meltdown-MPX
- Meltdown-BND

**Meltdown-GP**
- Meltdown-MPX
- Meltdown-BND

**Meltdown-MPX**
- Meltdown-BND

**Meltdown-BND**

**prediction fault**

- in-place (IP) vs., out-of-place (OP)
- Cross-address-space
- Same-address-space
Lessons learned

We have ignored microarchitectural attacks for many years:

- attacks on crypto → software should be fixed
- attacks on ASLR → ASLR is broken anyway
- attacks on TEEs → not within threat model
- Rowhammer → only some cheap modules → for years we solely optimized for performance
Lessons learned

We have ignored microarchitectural attacks for many years:

- attacks on crypto
Lessons learned

We have ignored microarchitectural attacks for many years:

- attacks on crypto → "software should be fixed"
Lessons learned

We have ignored microarchitectural attacks for many years:

- attacks on crypto → “software should be fixed”
- attacks on ASLR
Lessons learned

We have ignored microarchitectural attacks for many years:

- **attacks on crypto** → “software should be fixed”
- **attacks on ASLR** → “ASLR is broken anyway”
Lessons learned

We have ignored microarchitectural attacks for many years:

- attacks on crypto → “software should be fixed”
- attacks on ASLR → “ASLR is broken anyway”
- attacks on TEEs
Lessons learned

We have ignored microarchitectural attacks for many years:

- attacks on crypto → “software should be fixed”
- attacks on ASLR → “ASLR is broken anyway”
- attacks on TEEs → “not within threat model”
Lessons learned

We have ignored microarchitectural attacks for many years:

- **attacks on crypto** → “software should be fixed”
- **attacks on ASLR** → “ASLR is broken anyway”
- **attacks on TEEs** → “not within threat model”
- **Rowhammer**
Lessons learned

We have ignored microarchitectural attacks for many years:

- **attacks on crypto** → “software should be fixed”
- **attacks on ASLR** → “ASLR is broken anyway”
- **attacks on TEEs** → “not within threat model”
- **Rowhammer** → “only some cheap modules”
Lessons learned

We have ignored microarchitectural attacks for many years:

- **attacks on crypto** → “software should be fixed”
- **attacks on ASLR** → “ASLR is broken anyway”
- **attacks on TEEs** → “not within threat model”
- **Rowhammer** → “only some cheap modules”

→ for years we solely optimized for performance
Conclusion

• Optimizations always come at a cost
• Some mitigations cost more than gained by the feature they defend
• Transient-execution attacks will keep us busy for a while
• **Optimizations always come at a cost**
Conclusion

- Optimizations always come at a cost
- Some mitigations cost more than gained by the feature they defend
Conclusion

- Optimizations always come at a cost
- Some mitigations cost more than gained by the feature they defend
- Transient-execution attacks will keep us busy for a while
A Christmas Carol
The Spectres of the Past, Present, and Future

Moritz Lipp
"Past"
@mlqxyz

Michael Schwarz
"Present"
@misc0110

Claudio Canella
"Future"
@cc0x1f

Daniel Gruss
"Scrooge"
@lavados