A Christmas Carol

The Spectres of the Past, Present, and Future

Claudio Canella
Daniel Gruss
Moritz Lipp
Michael Schwarz

Acknowledgements I

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.thevectorart.com/

Some picture components are included from "Mickey's Christmas Carol" under fair use.

Acknowledgements II

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!

REFRESHING MEMORIES

Performance is awesome!

REFRESHING MEMORIES

Performance is awesome!

REFRESHING MEMORIES

- 1995

Performance is awesome!

REFRESHING MEMORIES

- 1995
- 150 MHz

Performance is awesome!

- 1995
- 150 MHz
- RISC emulating CISC

Performance is awesome!

REFRESHING MEMORIES

- 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!

REFRESHING MEMORIES

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

More and More Performance

REFRESHING MEMORIES

The future is going to be fast:

More and More Performance

The future is going to be fast:

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

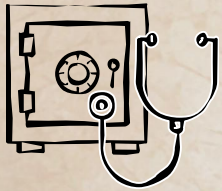
A Christmas Carol

Spectres of The Past

Side-Channel Attacks

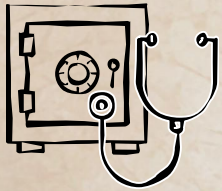
REFRESHING MEMORIES

- Bug-free software does not mean safe execution



Side-Channel Attacks

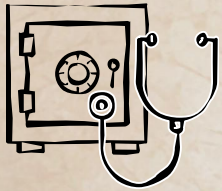
- Bug-free software does not mean safe execution
- Information leaks due to underlying hardware



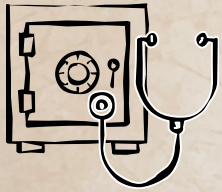
Side-Channel Attacks

REFRESHING MEMORIES

- 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

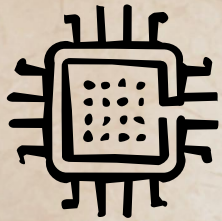
Power consumption

Execution time

CPU caches

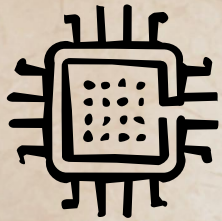
●●●

Architecture and Microarchitecture



- Instruction Set Architecture (ISA) is an abstract model of a computer (x86, ARMv8, SPARC, ...)

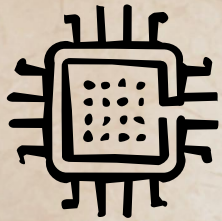
Architecture and Microarchitecture



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

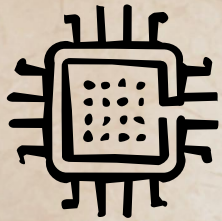
Architecture and Microarchitecture

REFRESHING MEMORIES



- 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

REFRESHING MEMORIES

- Modern CPUs contain multiple microarchitectural elements

Microarchitectural Elements

REFRESHING MEMORIES

- Modern CPUs contain multiple microarchitectural elements

Caches and buffer

Predictor

●●●

Microarchitectural Elements

REFRESHING MEMORIES

- Modern CPUs contain multiple microarchitectural elements

Caches and buffer

Predictor

●●●

- Transparent for the programmer

Microarchitectural Elements

REFRESHING MEMORIES

- Modern CPUs contain multiple microarchitectural elements

Caches and buffer

Predictor

●●●

- Transparent for the programmer
- Timing optimizations → side-channel leakage


Looking for nutritious choices?
Reach for the Stars!

1337 4242

FOOD CACHE

Revolutionary concept!

Store your food at home, never go to the grocery store during cooking.

Can store **ALL** kinds of food.

ONLY TODAY INSTEAD OF ~~$1,300~~

$1,299

ORDER VIA PHONE: +555 12345

E3

WORKS FOR ME

What could possibly go wrong with <insert x86 instruction here>?

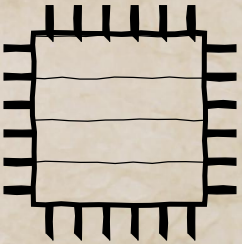
Clémentine Maurice, Moritz Lipp

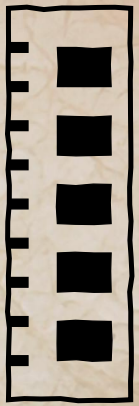
December 2016—33rd Chaos Communication Congress

CPU Cache

REFRESHING MEMORIES

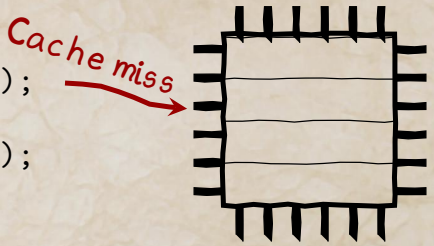
```
printf("%d", i);

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

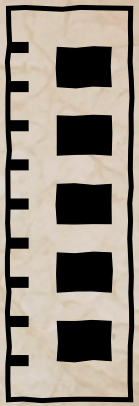




CPU Cache

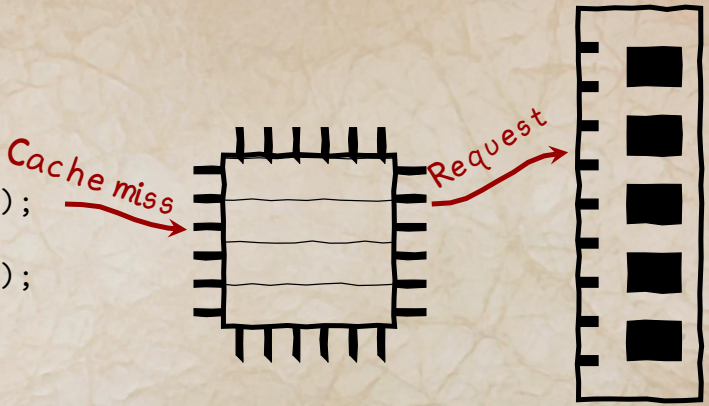

```
printf("%d", i);

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





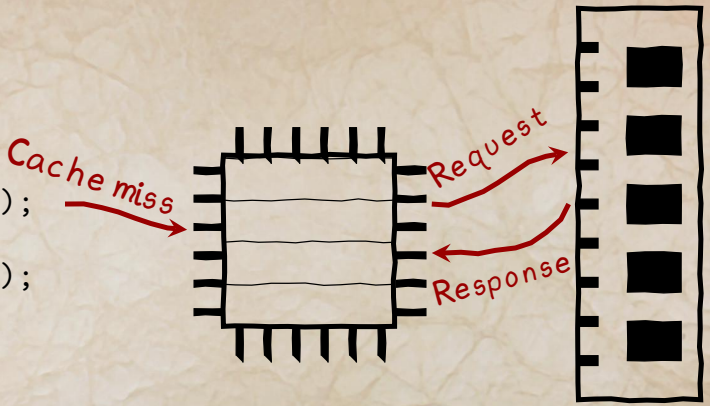
CPU Cache


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

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



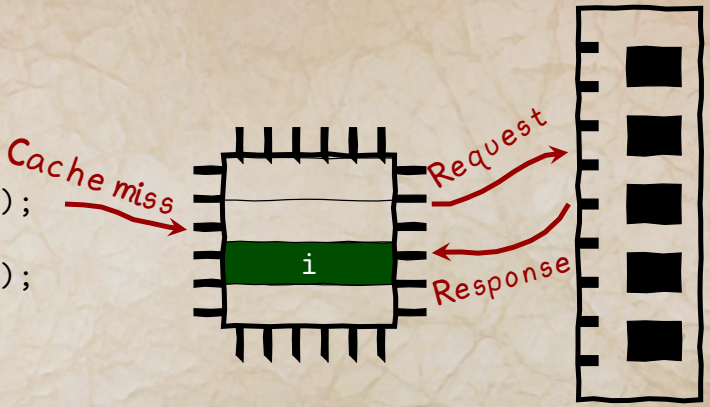
CPU Cache


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

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



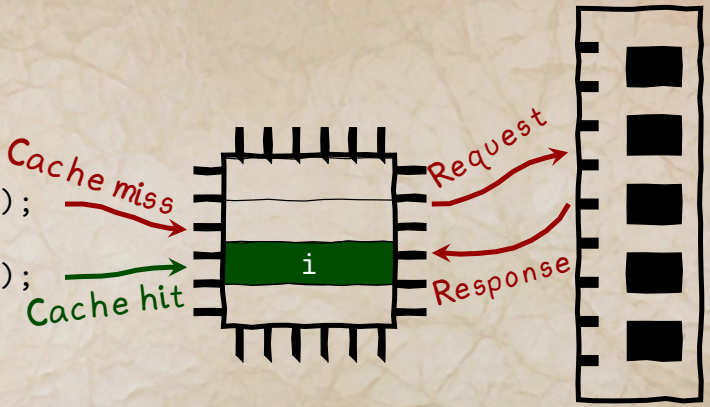
CPU Cache


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

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

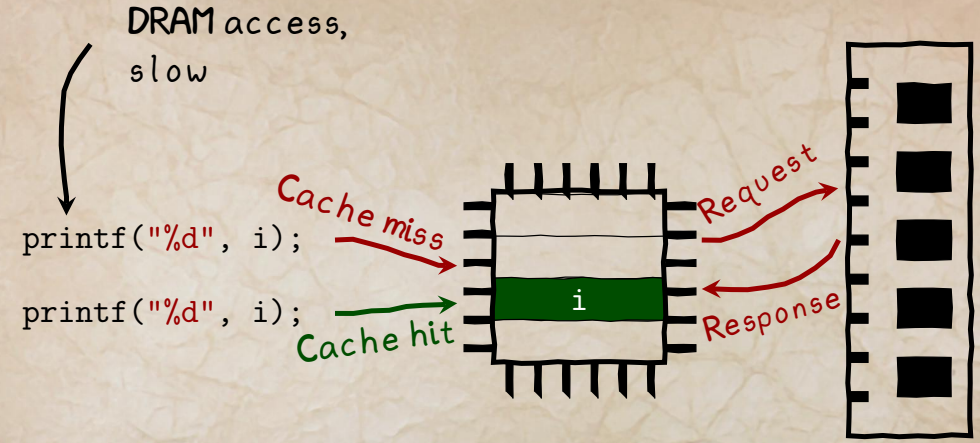


CPU Cache

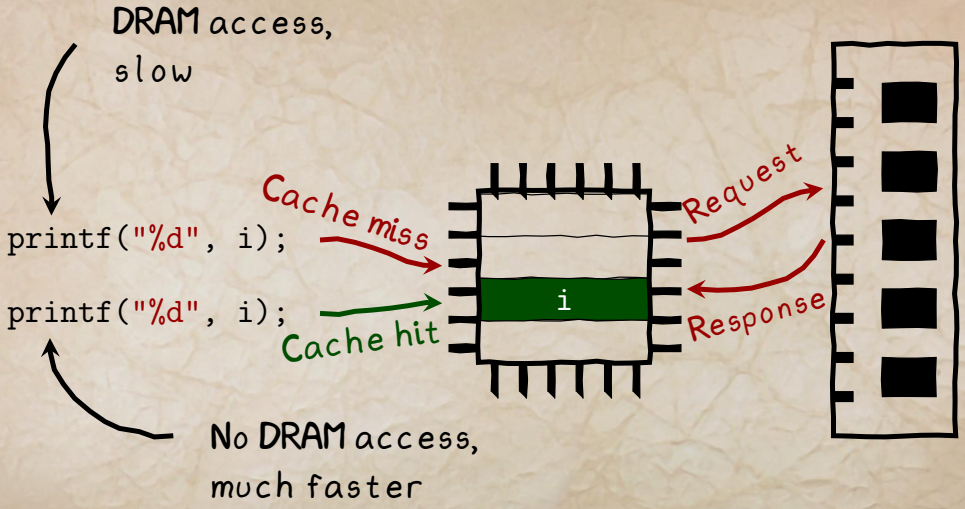

```
printf("%d", i);

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



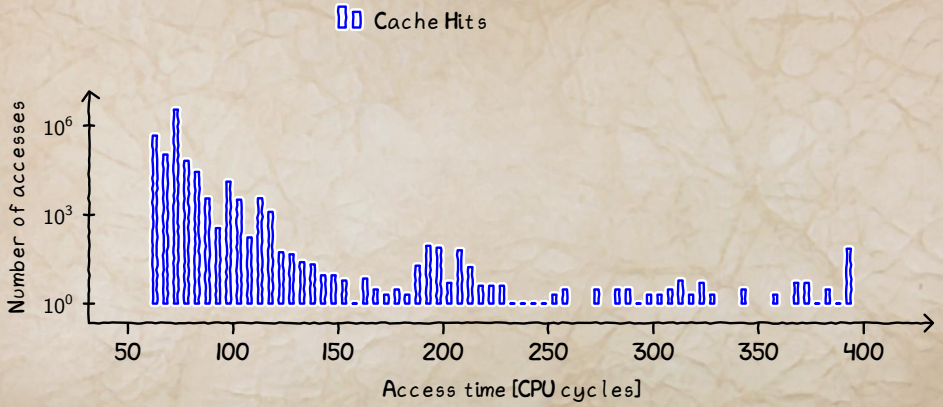
CPU Cache


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

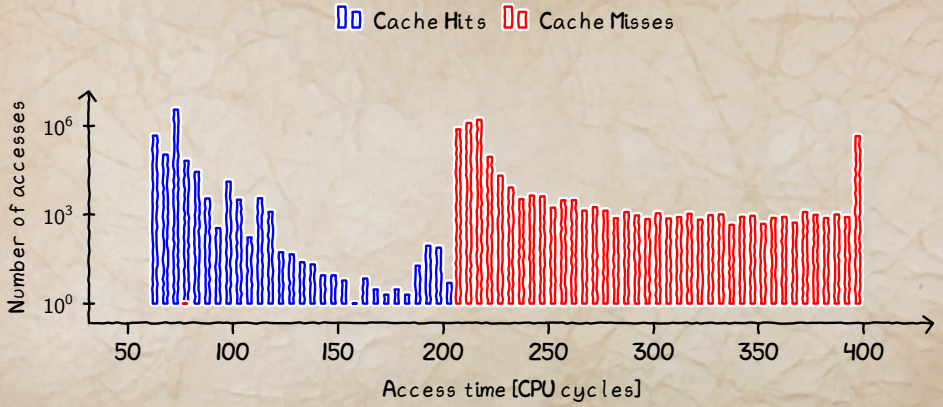
CPU Cache


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

Caching speeds up Memory Accesses



Caching speeds up Memory Accesses

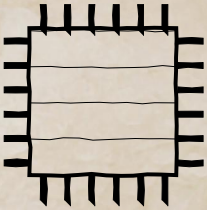


Shared Memory

Attacker

flush

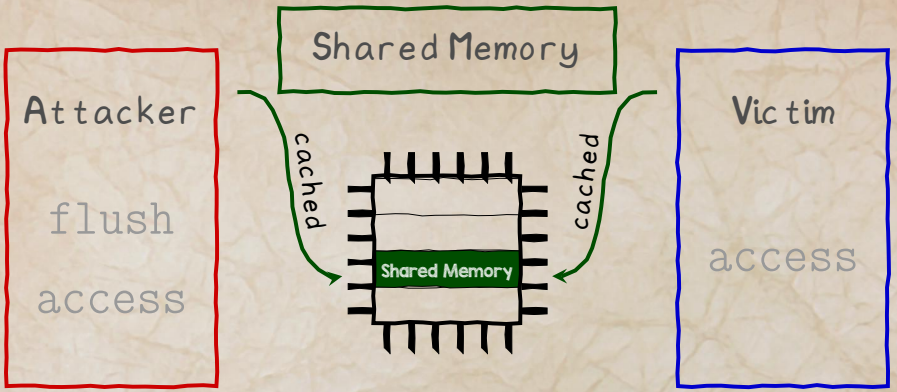
access

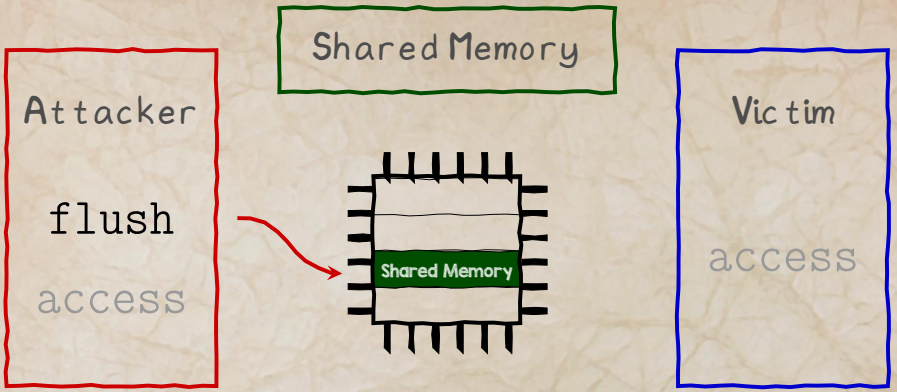


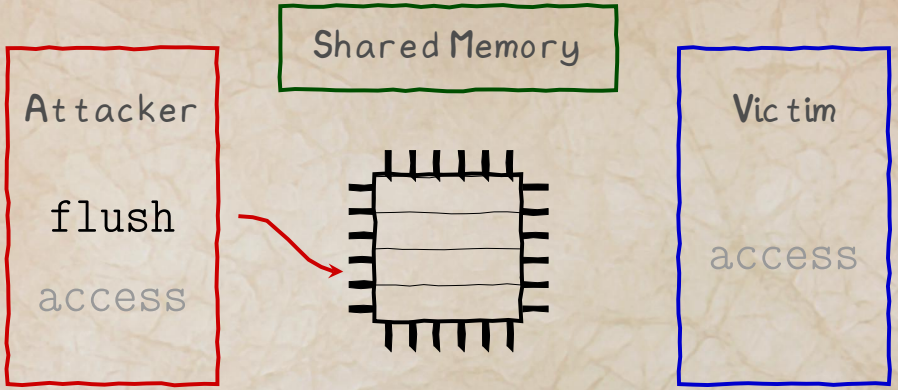
Victim

access

Flush+Reload

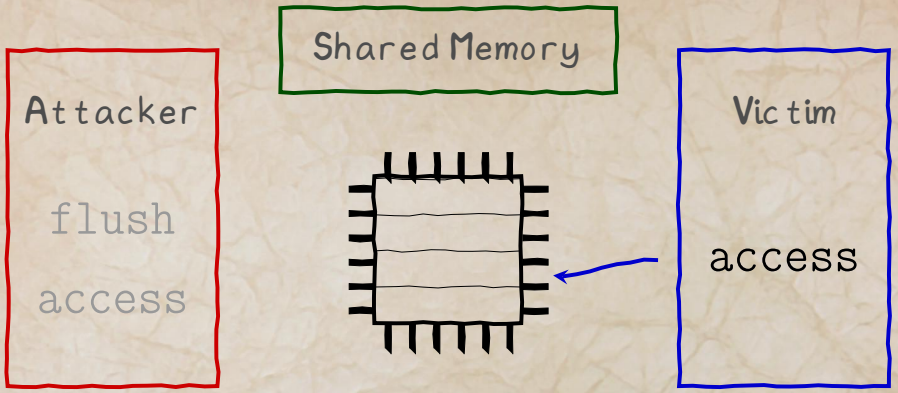


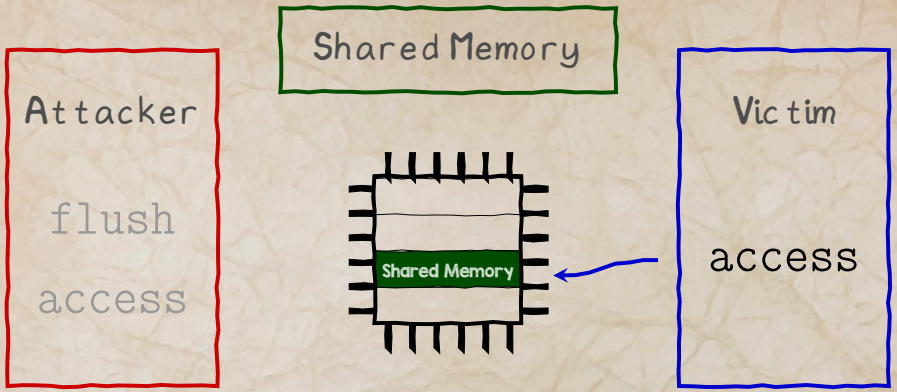

Shared Memory
Attacker
flush
access
Shared Memory
Victim
access

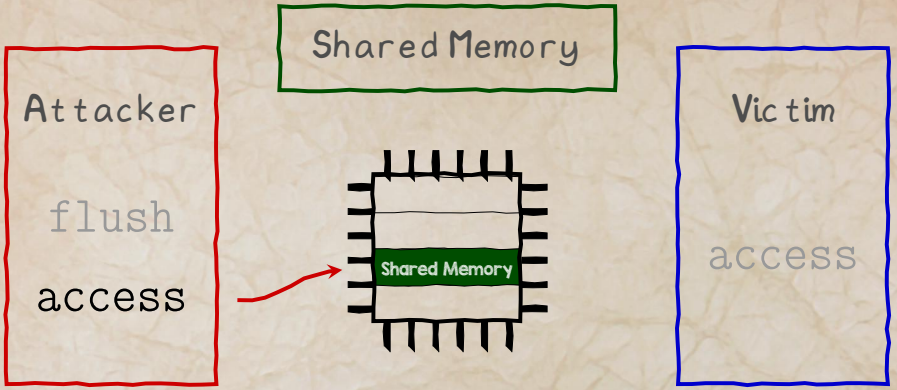

Shared Memory
Attacker
flush
access
Victim
access

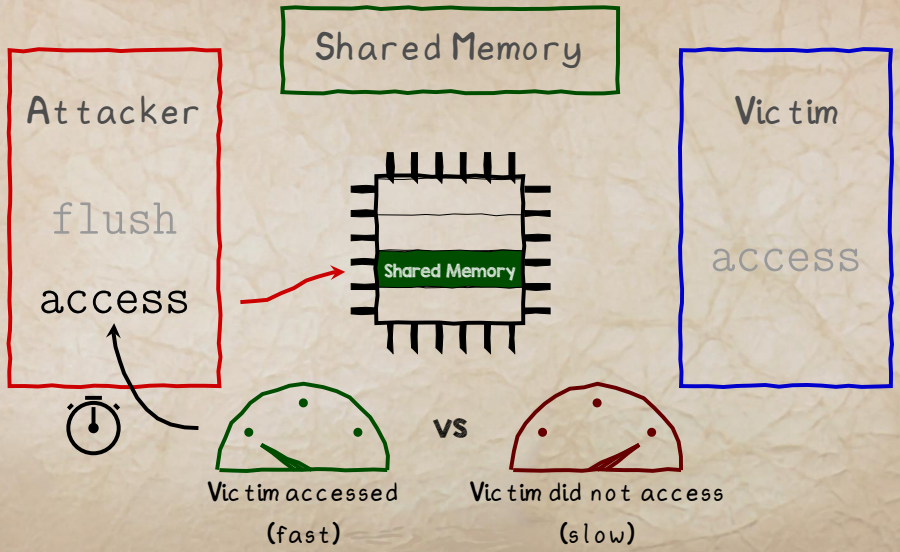
Flush+Reload

REFRESHING MEMORIES




Shared Memory
Attacker
flush
access
Shared Memory
Victim
access


Shared Memory
Attacker
flush
access
Shared Memory
Victim
access


Shared Memory
Attacker
flush
access
Shared Memory
Victim
access
vs
Victim accessed
(fast)
Victim did not access
(slow)

Attacks

REFRESHING MEMORIES

- 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

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

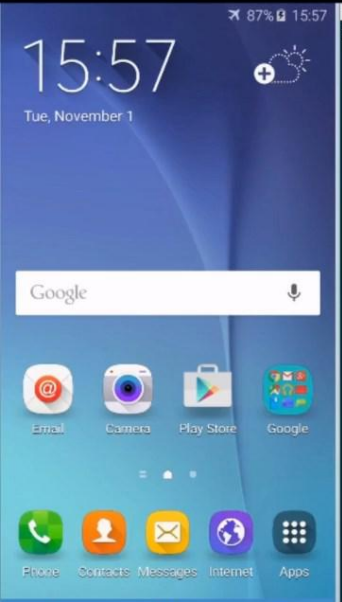
Attacks

- 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

REFRESHING MEMORIES

- 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, ...



File Edit View Search Terminal Help

```
shell@zeroflte:/data/local/tmp $ ./keyboard_spy -c 0
```

More and More Performance

REFRESHING MEMORIES

The future is going to be fast:

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

More and More Performance

REFRESHING MEMORIES

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

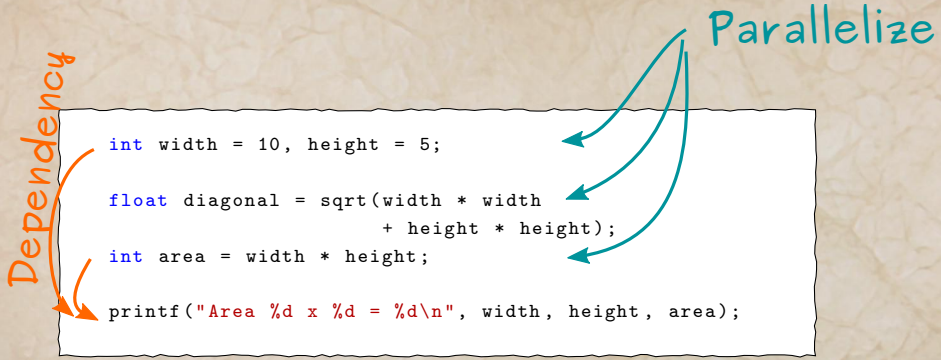
Spectres of The Present

Out-of-Order Execution

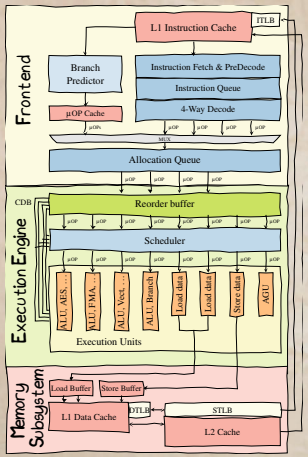
REFRESHING MEMORIES

```
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



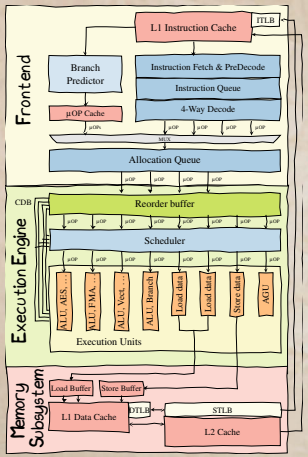
Out-of-Order Execution



Instructions are

- fetched and decoded in the front-end

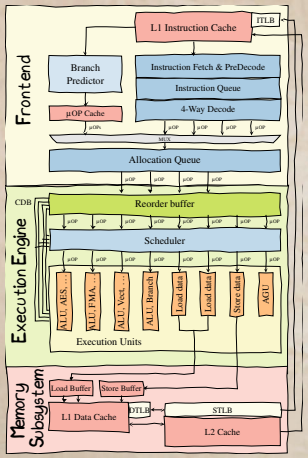
Out-of-Order Execution



Instructions are

- fetched and decoded in the front-end
- dispatched to the backend

Out-of-Order Execution



Instructions are

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

Building the Code

- An experiment

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

Building the Code

- An experiment

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

- volatile because compiler was not happy

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

Building the Code

- An experiment

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

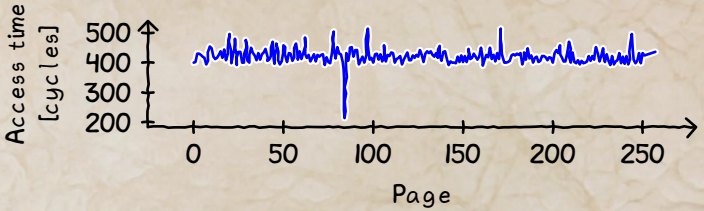
- volatile because compiler was not happy

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

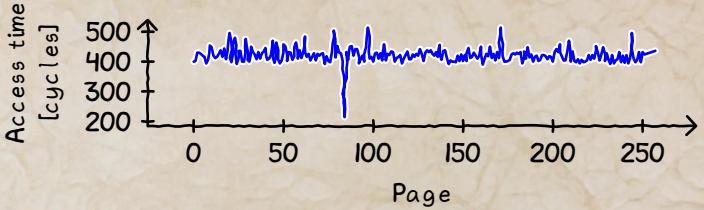
- Static code analyzer is still not happy

```
warning: Dereference of null pointer
         *(volatile char*)0;
```

- Flush+Reload over all pages of the array



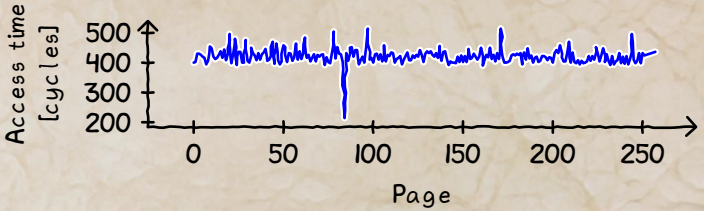
- Flush+Reload over all pages of the array



- "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

Building the Code

- 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

Building the Code

- 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

REFRESHING GK MEMORIES

Loading an address

REFRESHING MEMORIES

Loading an address

Loading an address

Loading an address

Loading an address

Building the Code

- Add another layer of indirection to test

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

Building the Code

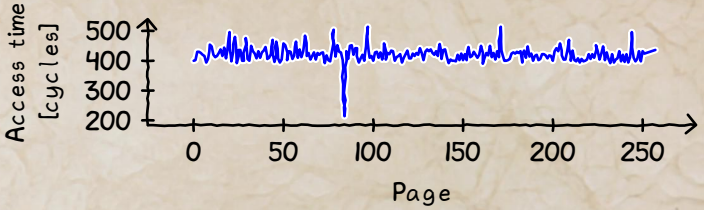
- Add another layer of indirection to test

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

- Then check if any part of array is cached

Building the Code

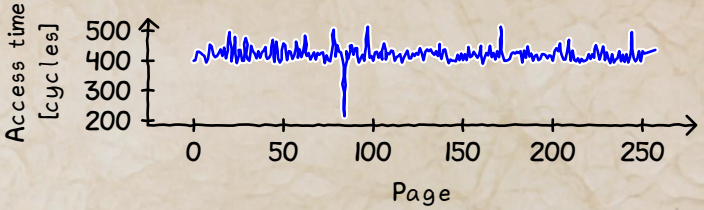
- Flush+Reload over all pages of the array



- Index of cache hit reveals data

Building the Code

- Flush+Reload over all pages of the array



- Index of cache hit reveals data
- Permission check fails sometimes

```
e01d8150: 69 6c 69 63 6f 6e 20 47  72 61 70 68 69 63 73 2c |ilicon Graphics,|
e01d8160: 20 49 6e 63 2e 20 20 48  6f 77 65 76 65 72 2c 20 | Inc.  However, |
e01d8170: 74 68 65 20 61 75 74 68  6f 72 73 20 6d 61 6b 65 |the authors make|
e01d8180: 20 6e 6f 20 63 6c 61 69  6d 20 74 68 61 74 20 4d | no claim that M|
e01d8190: 65 73 61 0a 20 69 73 20  69 6e 20 61 6e 79 20 77 |esa. is in any w|
e01d81a0: 61 79 20 61 20 63 6f 6d  70 61 74 69 62 6c 65 20 |ay a compatible |
e01d81b0: 72 65 70 6c 61 63 65 6d  65 6e 74 20 66 6f 72 20 |replacement for |
e01d81c0: 4f 70 65 6e 47 4c 20 6f  72 20 61 73 73 6f 63 69 |OpenGL or associ|
e01d81d0: 61 74 65 64 20 77 69 74  68 0a 20 53 69 6c 69 63 |ated with. Silic|
e01d81e0: 6f 6e 20 47 72 61 70 68  69 63 73 2c 20 49 6e 63 |on Graphics, Inc|
e01d81f0: 2e 0a 20 2e 0a 20 54 68  69 73 20 76 65 72 73 69 |.. .. This versi|
e01d8200: 6f 6e 20 6f 66 20 4d 65  73 61 20 70 72 6f 76 69 |on of Mesa provi|
e01d8210: 64 65 73 20 47 4c 58 20  61 6e 64 20 44 52 49 20 |des GLX and DRI |
e01d8220: 63 61 70 61 62 69 6c 69  74 69 65 73 3a 20 69 74 |capabilities: it|
e01d8230: 20 69 73 20 63 61 70 61  62 6c 65 20 6f 66 0a 20 | is capable of. |
e01d8240: 62 6f 74 68 20 64 69 72  65 63 74 20 61 6e 64 20 |both direct and |
e01d8250: 69 6e 64 69 72 65 63 74  20 72 65 6e 64 65 72 69 |indirect renderi|
e01d8260: 6e 67 2e 20 20 46 6f 72  20 64 69 72 65 63 74 20 |ng.  For direct |
e01d8270: 72 65 6e 64 65 72 69 6e  67 2c 20 69 74 20 63 61 |rendering, it ca|
e01d8280: 6e 20 75 73 65 20 44 52  49 0a 20 6d 6f 64 75 6c |n use DRI. modul|
```

MELTDOWN

Meltdown Mitigation

REFRESHING MEMORIES

- Kernel addresses in user space are a problem

Meltdown Mitigation

REFRESHING MEMORIES

- Kernel addresses in user space are a problem
- Why don't we take the kernel addresses...

Meltdown Mitigation

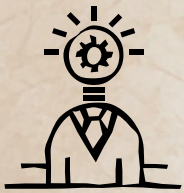
- ...and remove them if not needed?

Meltdown Mitigation

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

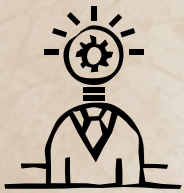
Meltdown-US Mitigation

REFRESHING MEMORIES



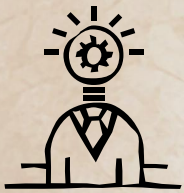
- Unmap the kernel in user space

Meltdown-US Mitigation

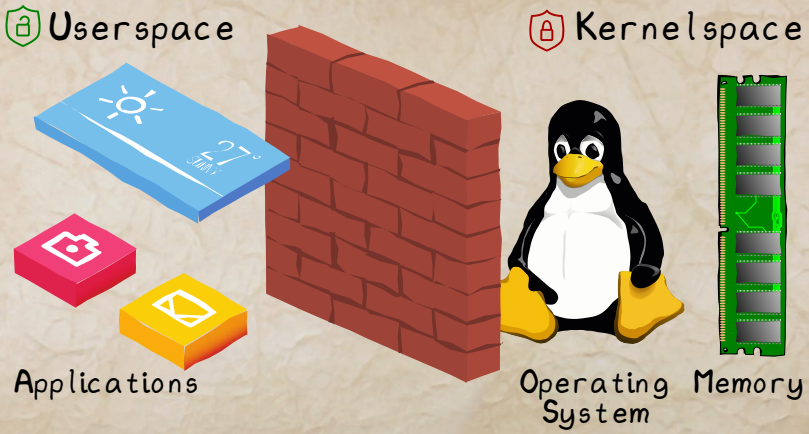


- Unmap the kernel in user space
- Kernel addresses are then no longer present

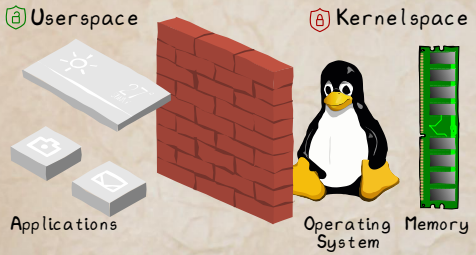
Meltdown-US Mitigation



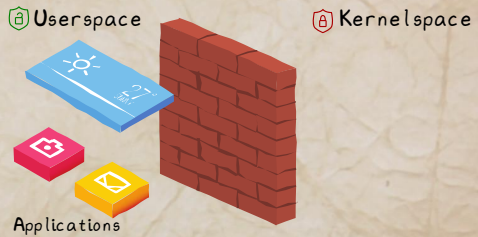
- Unmap the kernel in user space
- Kernel addresses are then no longer present
- Memory which is not mapped cannot be accessed at all


Userspace
Kernelspace
Applications
Operating System
Memory

Kernel View

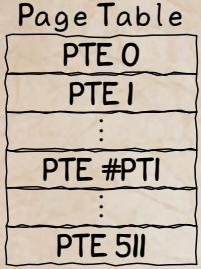


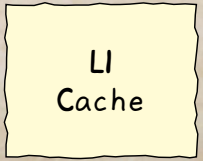
User View



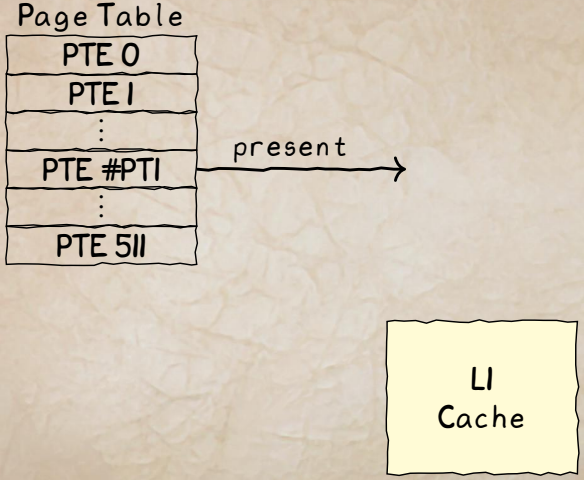
context switch

Meltdown-P (aka Foreshadow-NG)

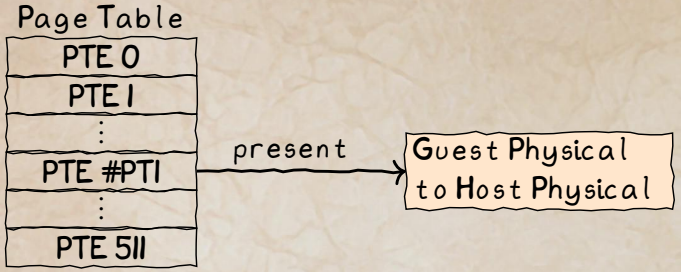




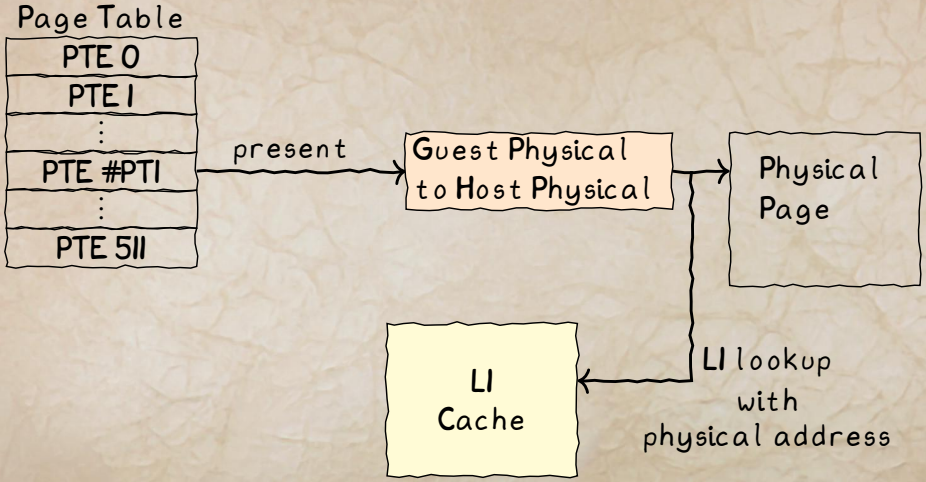
Meltdown-P (aka Foreshadow-NG)



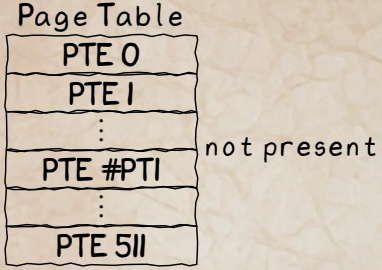
Meltdown-P (aka Foreshadow-NG)

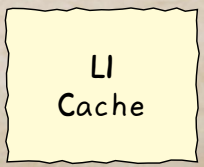


Meltdown-P (aka Foreshadow-NG)

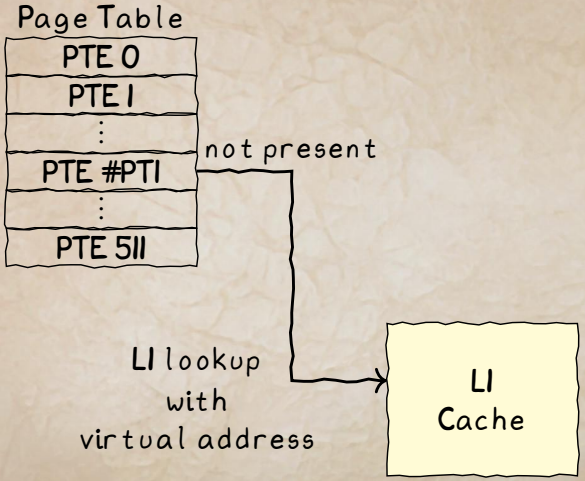


Meltdown-P (aka Foreshadow-NG)





Meltdown-P (aka Foreshadow-NG)



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

More and More Performance

REFRESHING MEMORIES

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

https://xkcd.com/838/

Speculative Execution

Let us get rid of bottlenecks

Use the naughty/nice list of last year

Speculative Execution

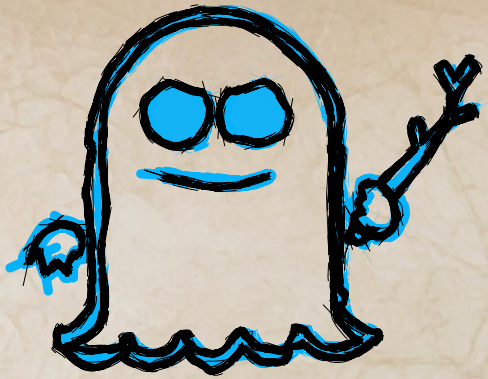
REFRESHING MEMORIES

Finally, check predictions with list of this year

Throwing away wrongly manufac-tured presents

Speculative Execution

Correct predictions result in free time



SPECTRE

Spectre-PHT (aka Spectre Variant 1)


```
index = 0;

char* data = "textKEY";

if (index < 4)
```

then

Prediction

else

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

0

Spectre-PHT (aka Spectre Variant 1)


```
index = 0;

char* data = "textKEY";

if (index < 4)
```

then

else

Prediction

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

0

Spectre-PHT (aka Spectre Variant 1)


```
index = 0;

char* data = "textKEY";

if (index < 4)
```

then

else

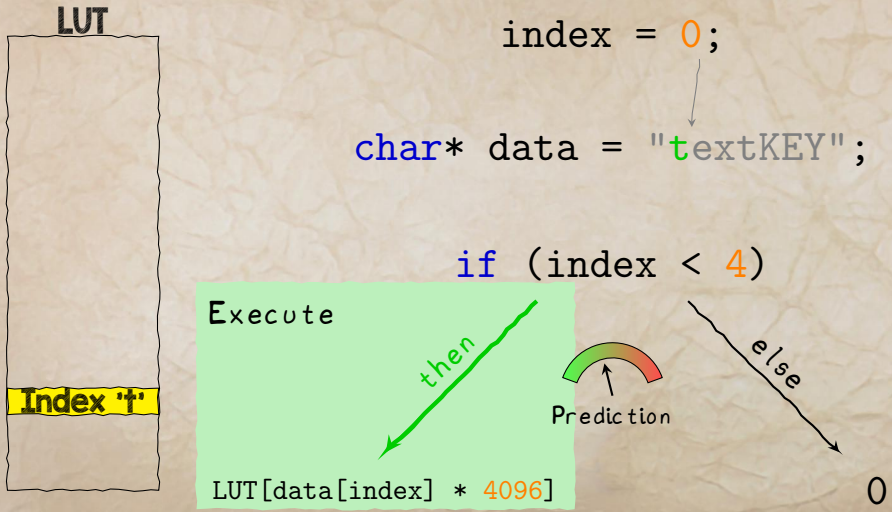
Prediction

Speculate

LUT[data[index] * 4096]

0

Spectre-PHT (aka Spectre Variant 1)



Spectre-PHT (aka Spectre Variant 1)


```
index = 1;

char* data = "textKEY";

if (index < 4)
```

then

else

Prediction

`LUT[data[index] * 4096]`

0

Spectre-PHT (aka Spectre Variant 1)

REFRESHING MEMORIES


```
index = 1;

char* data = "textKEY";

if (index < 4)
```

then

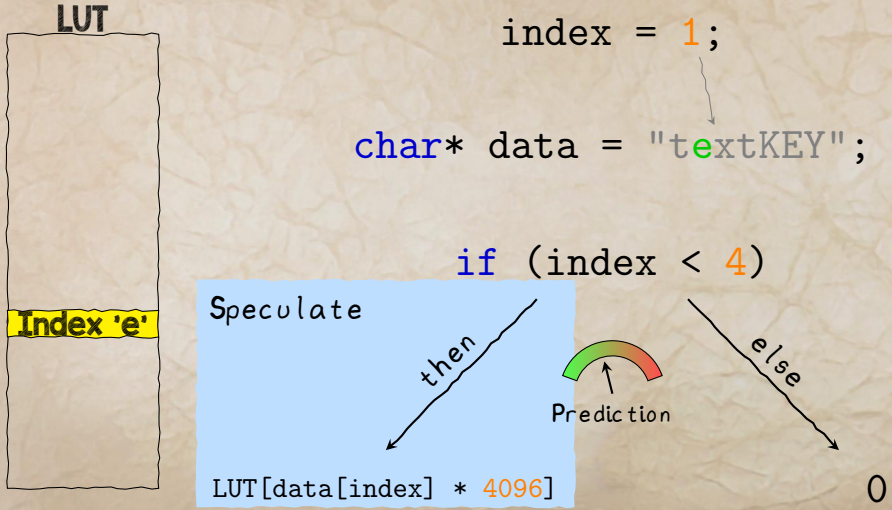
else

Prediction

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

0

Spectre-PHT (aka Spectre Variant 1)



Spectre-PHT (aka Spectre Variant 1)


```
index = 1;

char* data = "textKEY";

if (index < 4)
```

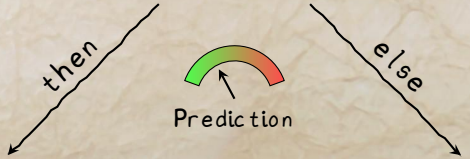
then → `LUT[data[index] * 4096]`

Prediction

else → 0

Spectre-PHT (aka Spectre Variant 1)


```
index = 2;

char* data = "textKEY";

if (index < 4)
```



`LUT[data[index] * 4096]`

0

Spectre-PHT (aka Spectre Variant 1)

REFRESHING MEMORIES


```
index = 2;

char* data = "textKEY";

if (index < 4)
```

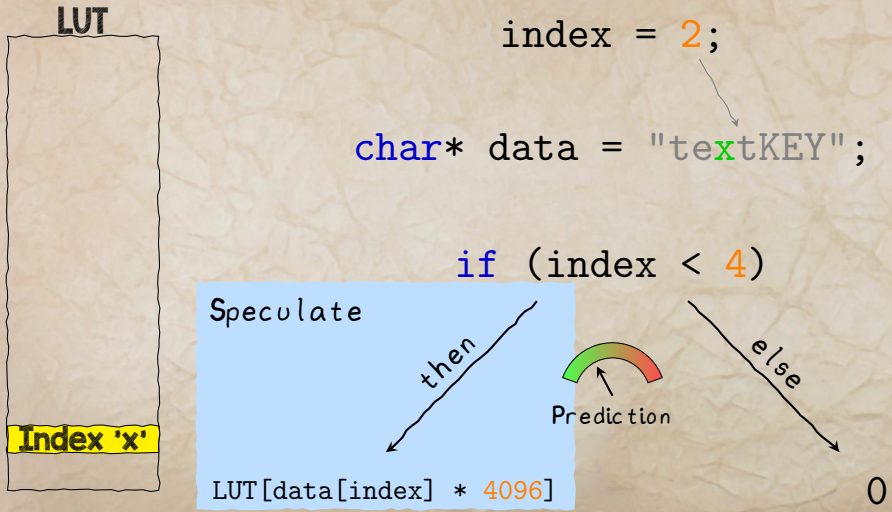
then

else


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

0

Spectre-PHT (aka Spectre Variant 1)



Spectre-PHT (aka Spectre Variant 1)

REFRESHING MEMORIES


```
index = 2;

char* data = "textKEY";

if (index < 4)
```

then → `LUT[data[index] * 4096]`

Prediction

else → 0

Spectre-PHT (aka Spectre Variant 1)


```
index = 3;

char* data = "textKEY";

if (index < 4)
```

then

Prediction

else

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

0

Spectre-PHT (aka Spectre Variant 1)


```
index = 3;

char* data = "textKEY";

if (index < 4)
```

then

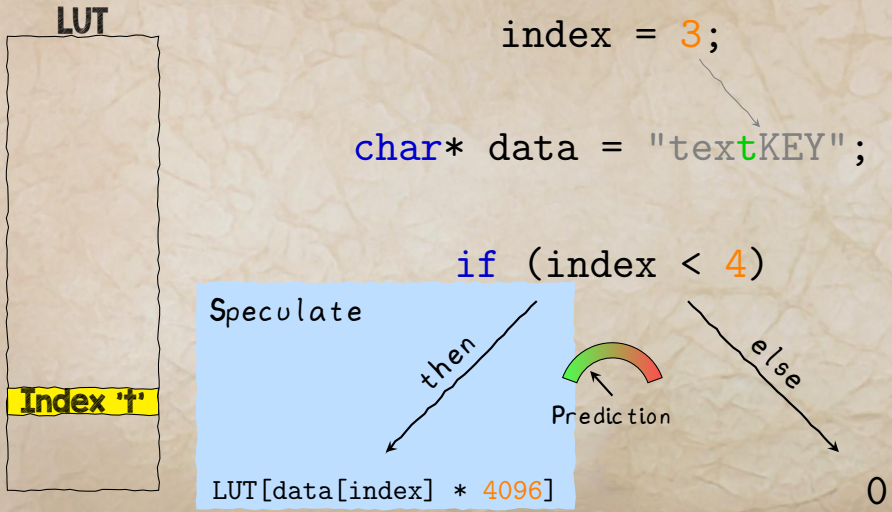
else

Prediction

`LUT[data[index] * 4096]`

0

Spectre-PHT (aka Spectre Variant 1)



Spectre-PHT (aka Spectre Variant 1)


```
index = 3;
char* data = "textKEY";
if (index < 4)
```

then → `LUT[data[index] * 4096]`

Prediction

else → 0

Spectre-PHT (aka Spectre Variant 1)


```
index = 4;

char* data = "textKEY";

if (index < 4)
```

then

else

Prediction

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

0

Spectre-PHT (aka Spectre Variant 1)

REFRESHING MEMORIES


```
index = 4;

char* data = "textKEY";

if (index < 4)
```

then

Prediction

else

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

0

Spectre-PHT (aka Spectre Variant 1)


```
index = 4;

char* data = "textKEY";

if (index < 4)
```

Speculate

then

else

Prediction

LUT[data[index] * 4096]

0

Spectre-PHT (aka Spectre Variant I)


```
index = 4;

char* data = "textKEY";

if (index < 4)
```

then

Prediction

else

Execute

0

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

Spectre-PHT (aka Spectre Variant 1)


```
index = 5;

char* data = "textKEY";

if (index < 4)
```

then

else


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

0

Spectre-PHT (aka Spectre Variant 1)

REFRESHING MEMORIES


```
index = 5;

char* data = "textKEY";

if (index < 4)
```

then

else

Prediction

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

0

Spectre-PHT (aka Spectre Variant 1)


```
index = 5;

char* data = "textKEY";

if (index < 4)
```

Speculate

then

else

Prediction

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

0

Spectre-PHT (aka Spectre Variant 1)


```
index = 5;

char* data = "textKEY";

if (index < 4)
```

then

Prediction

else

Execute

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

0

Spectre-PHT (aka Spectre Variant 1)


```
index = 6;

char* data = "textKEY";

if (index < 4)
```

then

Prediction

else

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

0

Spectre-PHT (aka Spectre Variant 1)

REFRESHING MEMORIES


```
index = 6;

char* data = "textKEY";

if (index < 4)
```

then

else

Prediction

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

0

Spectre-PHT (aka Spectre Variant 1)


```
index = 6;

char* data = "textKEY";

if (index < 4)
```

Speculate

then

Prediction

else

LUT[data[index] * 4096]

0

Spectre-PHT (aka Spectre Variant 1)


```
index = 6;

char* data = "textKEY";

if (index < 4)
```

then

Prediction

else

Execute

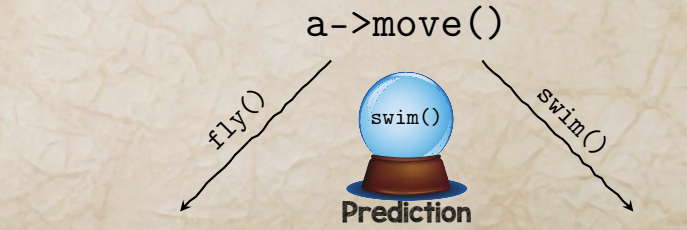
```
LUT[data[index] * 4096]
```

0

Spectre-BTB (aka Spectre Variant 2)

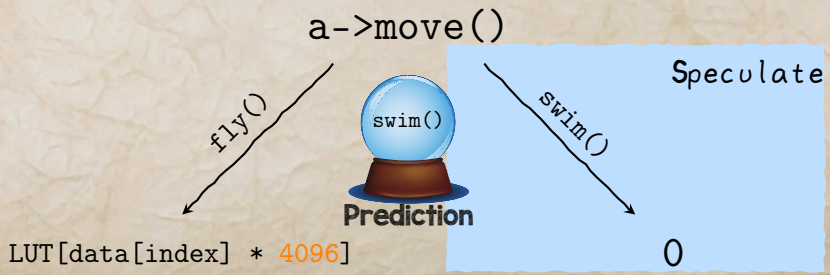
REFRESHING MEMORIES

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



Spectre-BTB (aka Spectre Variant 2)

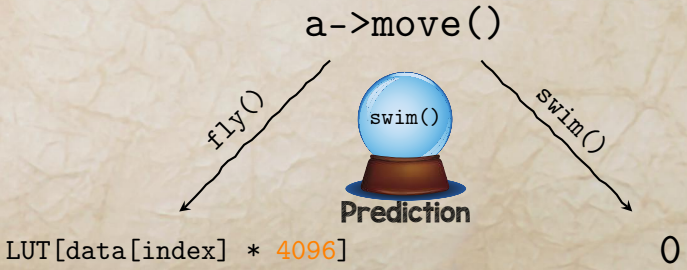

```
Animal* a = bird;
```



Spectre-BTB (aka Spectre Variant 2)

REFRESHING MEMORIES

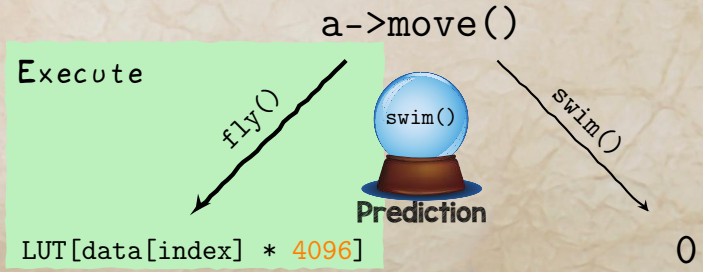
```
Animal* a = bird;
```



Spectre-BTB (aka Spectre Variant 2)

REFRESHING MEMORIES

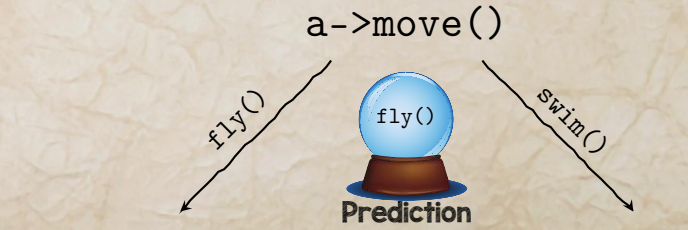
```
Animal* a = bird;
```



Spectre-BTB (aka Spectre Variant 2)

REFRESHING MEMORIES

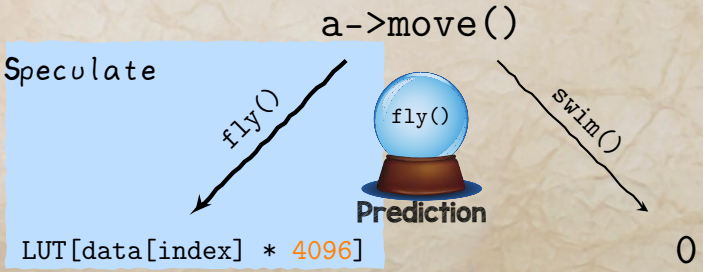
```
Animal* a = bird;
```



Spectre-BTB (aka Spectre Variant 2)

REFRESHING MEMORIES

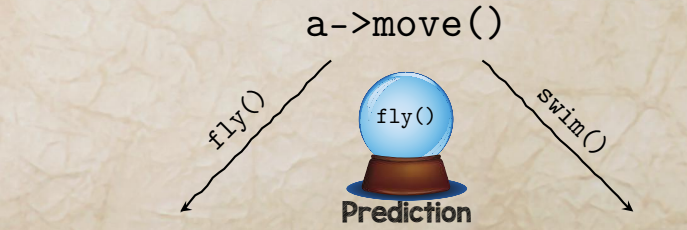
```
Animal* a = bird;
```



Spectre-BTB (aka Spectre Variant 2)

REFRESHING MEMORIES

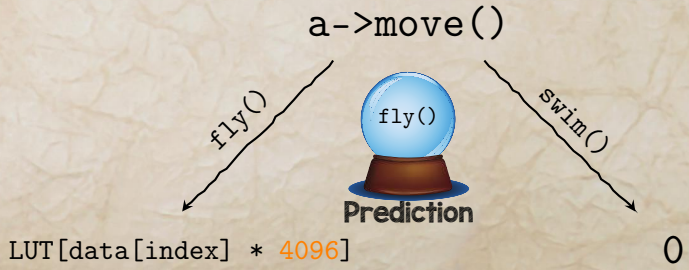
```
Animal* a = bird;
```



Spectre-BTB (aka Spectre Variant 2)

REFRESHING MEMORIES

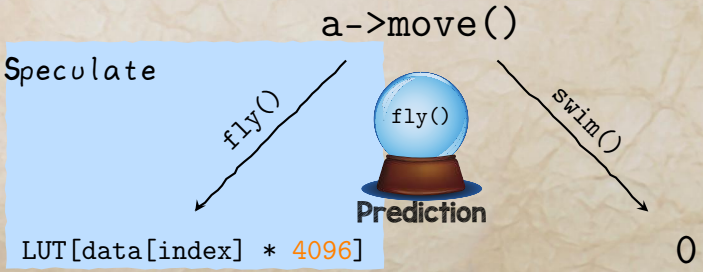
```
Animal* a = fish;
```



Spectre-BTB (aka Spectre Variant 2)

REFRESHING MEMORIES

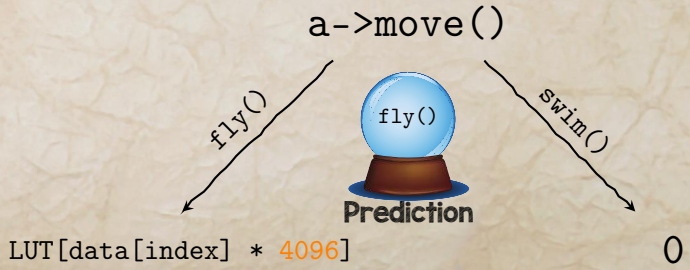
```
Animal* a = fish;
```



Spectre-BTB (aka Spectre Variant 2)

REFRESHING MEMORIES

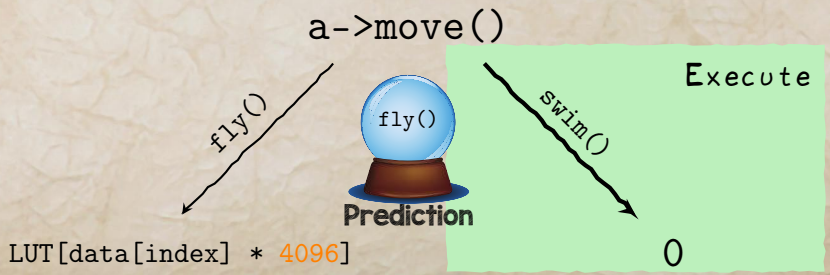
```
Animal* a = fish;
```



Spectre-BTB (aka Spectre Variant 2)

REFRESHING MEMORIES

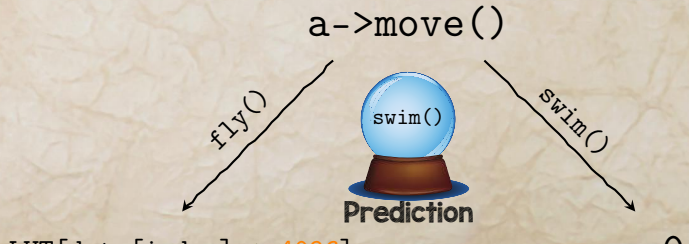
```
Animal* a = fish;
```

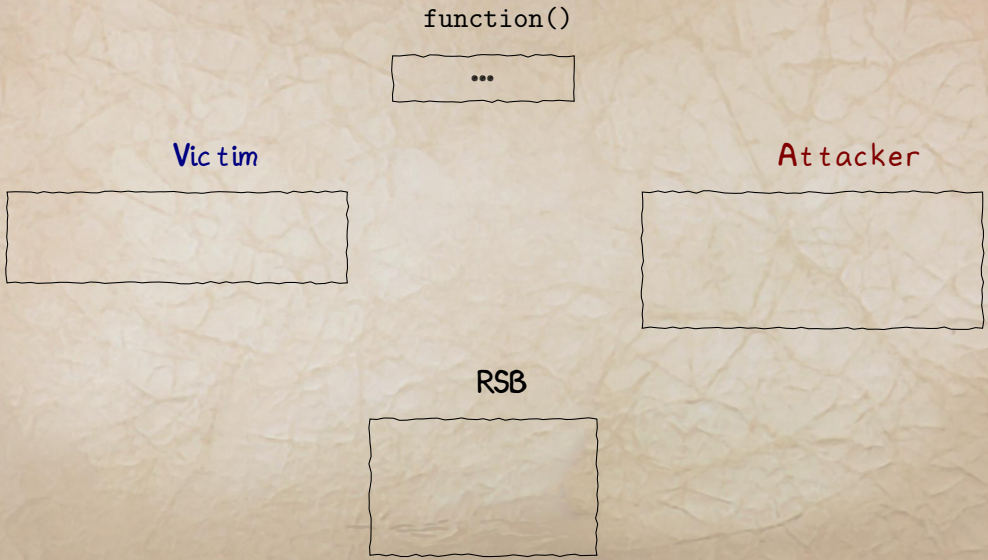


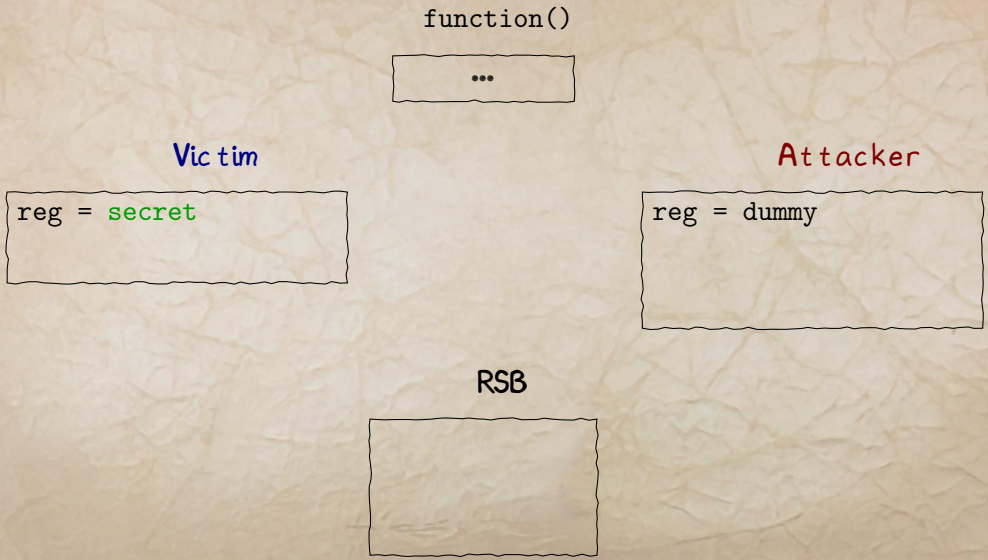
Spectre-BTB (aka Spectre Variant 2)

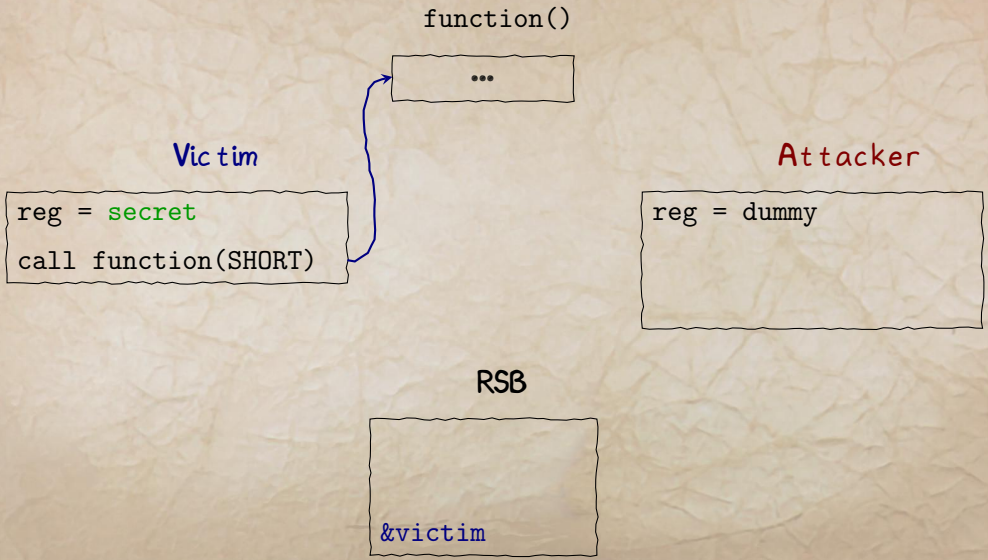
REFRESHING MEMORIES

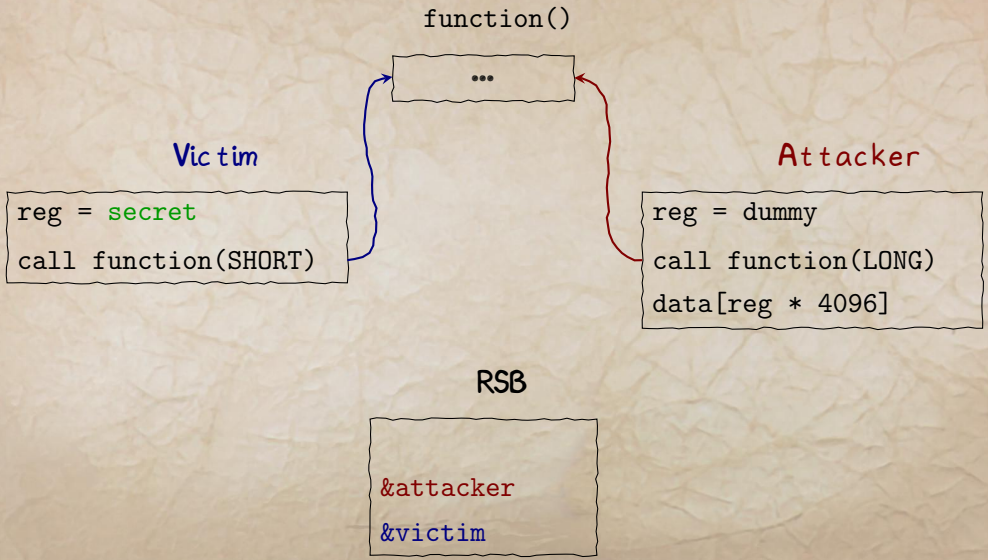
```
Animal* a = fish;
```

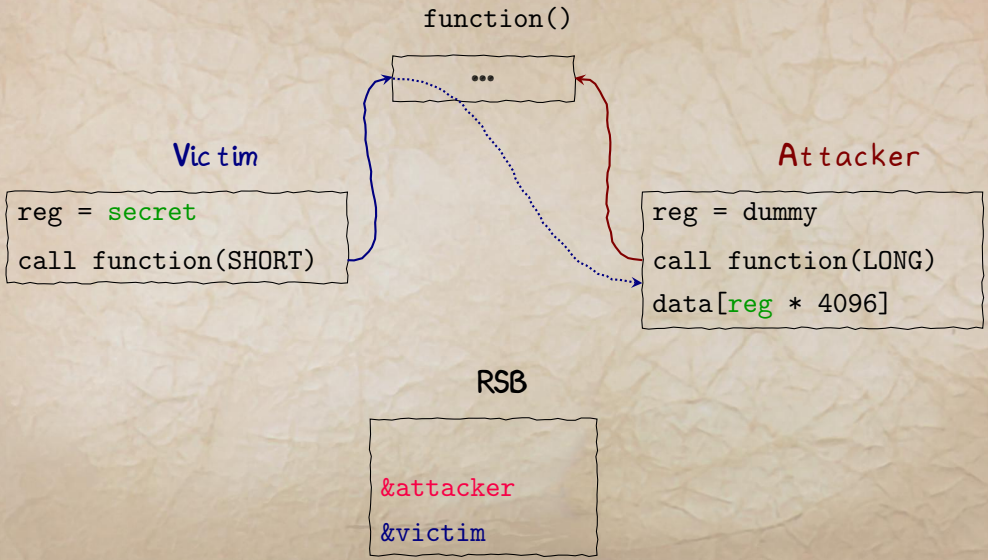


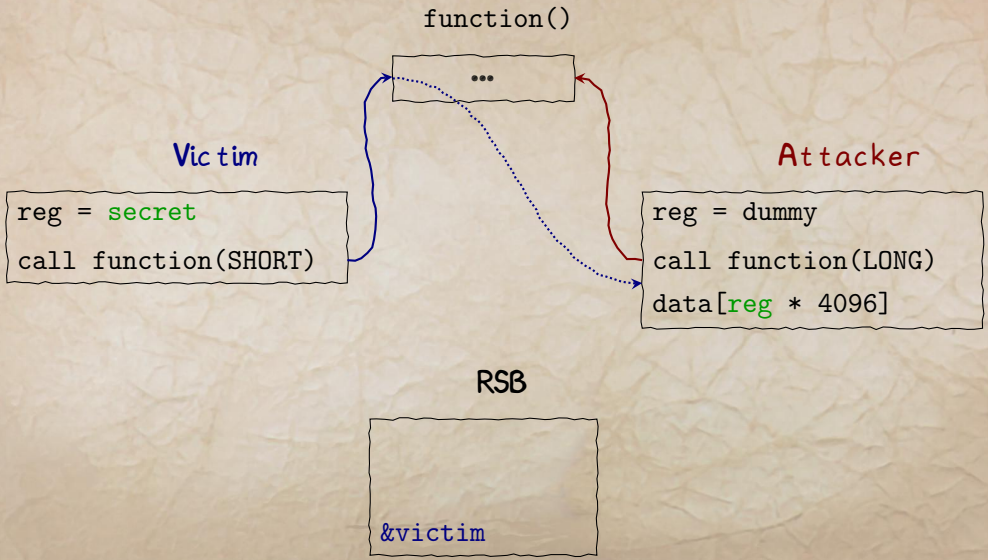

function()
...
Victim
Attacker
RSB

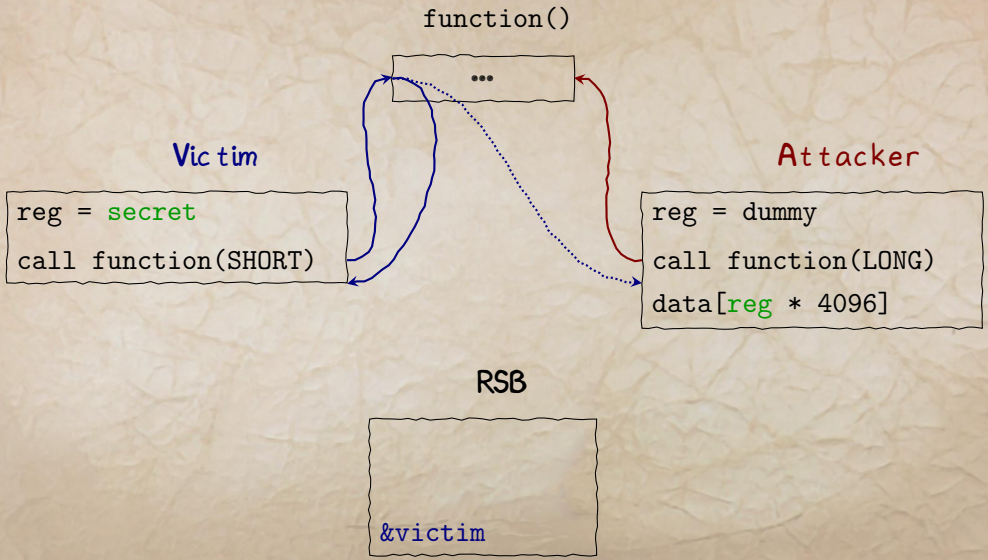

function()
...
Victim
reg = secret
Attacker
reg = dummy
RSB


function()
...
Victim
reg = secret
call function(SHORT)
Attacker
reg = dummy
RSB
&victim

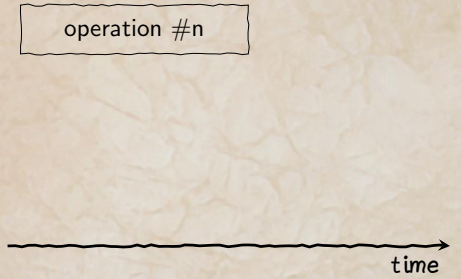

function()
...
Victim
reg = secret
call function(SHORT)
Attacker
reg = dummy
call function(LONG)
data[reg * 4096]
RSB
&attacker
&victim


function()
...
Victim
reg = secret
call function(SHORT)
Attacker
reg = dummy
call function(LONG)
data[reg * 4096]
RSB
&attacker
&victim

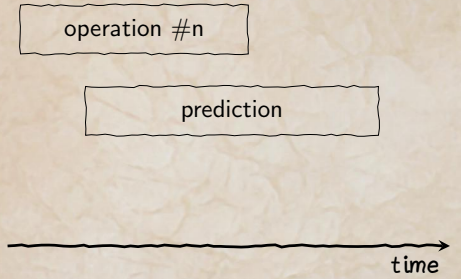

function()
...
Victim
reg = secret
call function(SHORT)
Attacker
reg = dummy
call function(LONG)
data[reg * 4096]
RSB
&victim


function()
...
Victim
reg = secret
call function(SHORT)
Attacker
reg = dummy
call function(LONG)
data[reg * 4096]
RSB
&victim

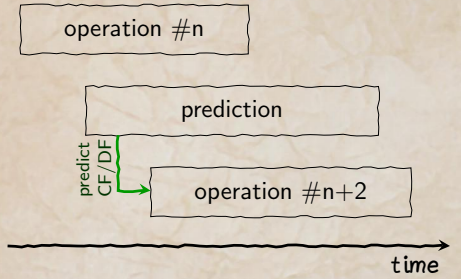
Meltdown vs. Spectre



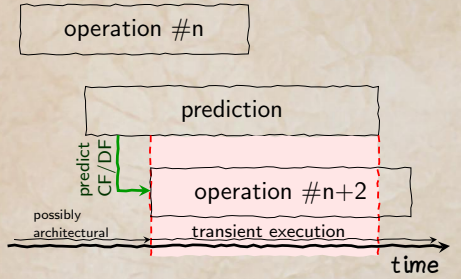
Meltdown vs. Spectre



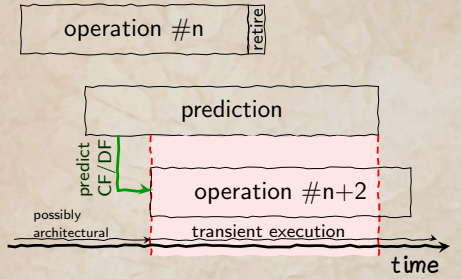
Meltdown vs. Spectre



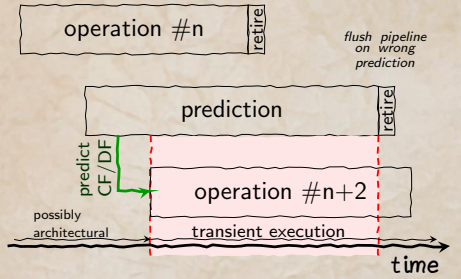
Meltdown vs. Spectre



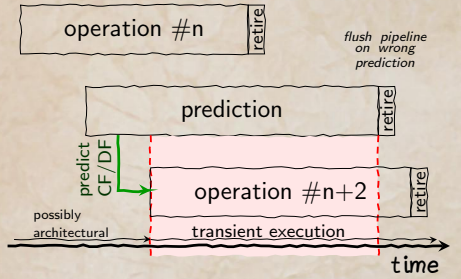
Meltdown vs. Spectre



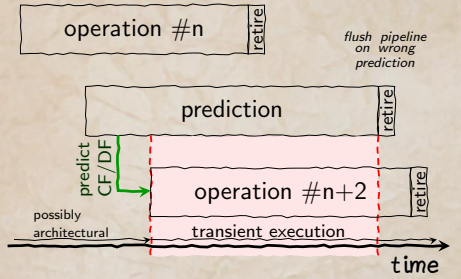
Meltdown vs. Spectre

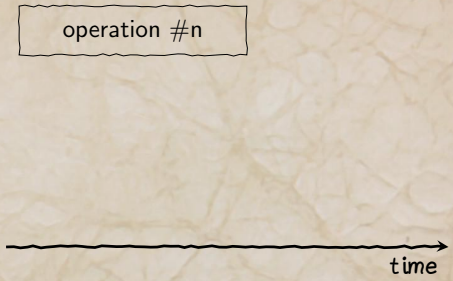


Meltdown vs. Spectre

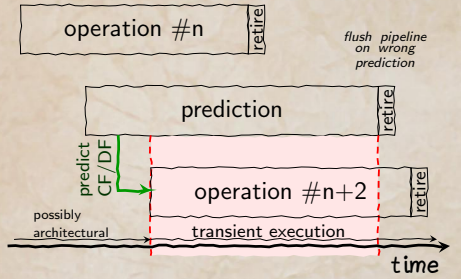


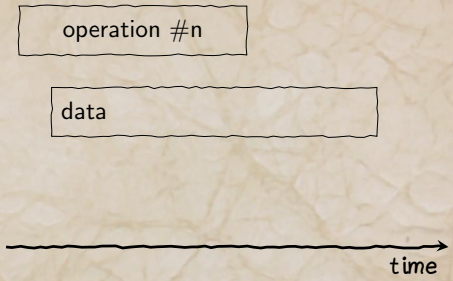
Meltdown vs. Spectre



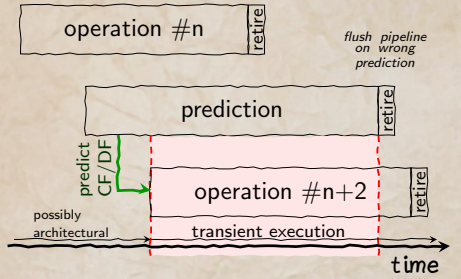


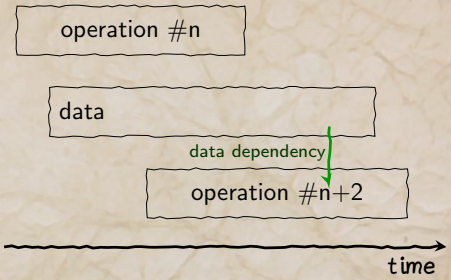
Meltdown vs. Spectre



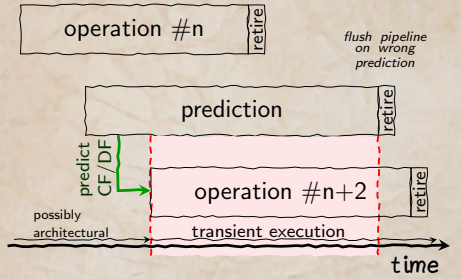


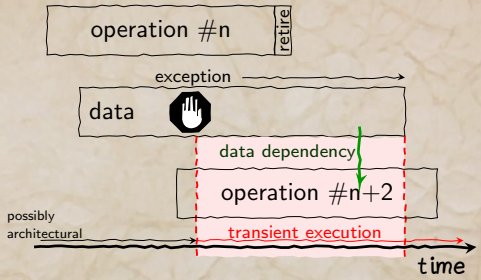
Meltdown vs. Spectre



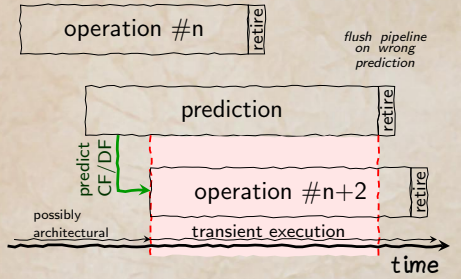


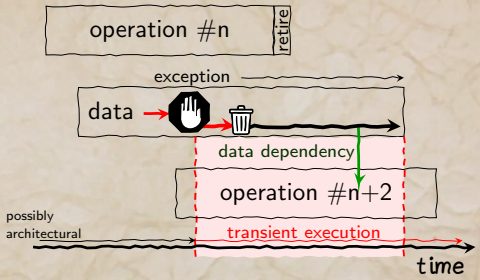
Meltdown vs. Spectre



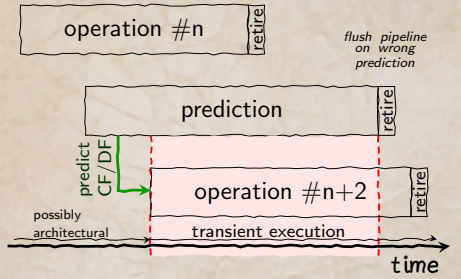


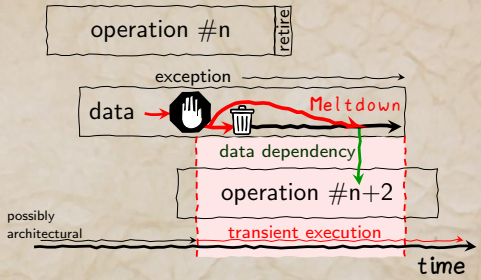
Meltdown vs. Spectre



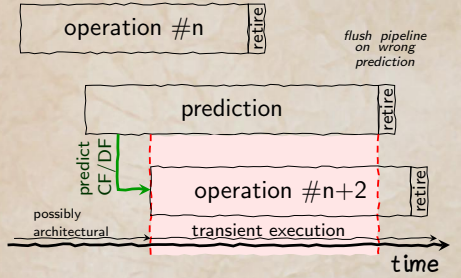


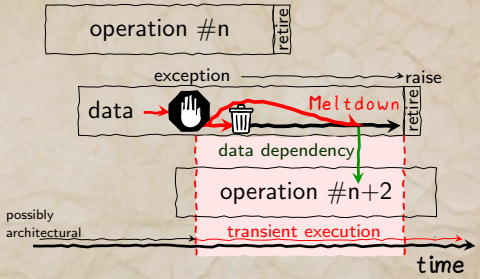
Meltdown vs. Spectre





Meltdown vs. Spectre





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

Intel MPK

- Protection key for a group of pages

Intel MPK

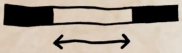
- Protection key for a group of pages
- 4 bits in PTE identify key for protected memory regions

Intel MPK

- Protection key for a group of pages
- 4 bits in PTE identify key for protected memory regions
- Quick update of access rights

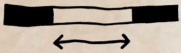
- Protection keys are lazily enforced

- Protection keys are lazily enforced
- Protected value is forwarded to transient instructions

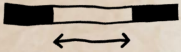


- x86 provides dedicated instruction raising #BR exception if bound-range is exceeded

Meltdown-BR



- 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

Messages in this thread

- *First message in thread*
- **Tom Lendacky**
  - *Dave Hansen*
    - *Tom Lendacky*
  - *Borislav Petkov*
  - *tip-bot for Tom Lendacky*
  - *Pavel Machek*
    - *Brian Gerst*
    - *Thomas Gleixner*

Patch in this message

- *Get diff 1*

From Tom Lendacky <>

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 <thomas.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);
```

Messages in this thread

- *First message in thread*
- **Tom Lendacky**
  - *Dave Hansen*
    - *Tom Lendacky*
  - *Borislav Petkov*
  - *tip-bot for Tom Lendacky*
  - *Pavel Machek*
    - *Brian Gerst*
    - *Thomas Gleixner*

Patch in this message

- *Get diff 1*

From Tom Lendacky <>
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 <thomas.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);
```

Messages in this thread

- *First message in thread*
- **Tom Lendacky**
  - *Dave Hansen*
    - *Tom Lendacky*
  - *Borislav Petkov*
  - *tip-bot for Tom Lendacky*
  - *Pavel Machek*
    - *Brian Gerst*
    - *Thomas Gleixner*

Patch in this message

- *Get diff 1*

From Tom Lendacky <>
Subject [PATCH] x86/cpu, x86/pti: Do not enable PTI on AMD processors
Date Tue, 26 Dec 2017 23:43:54 -0600

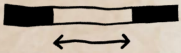
AMD processors are not subject to the types of attacks that the kernel
page table isolation feature protects against. The AMD microarchitecture
does not allow memory references, including speculative references, that
access higher privileged data when running in a lesser privileged mode
when that access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

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

REFRESHING MEMORIES

Vendor \ Attack	Meltdown-US	Meltdown-P	Meltdown-GP	Meltdown-NM	Meltdown-RW	Meltdown-PK	Meltdown-BR	Meltdown-DE	Meltdown-AC	Meltdown-UD	Meltdown-SS	Meltdown-XD	Meltdown-SM
Intel	●	●	●	●	●	★	★	☆	☆	☆	☆	☆	☆
ARM	●	○	●	_	●	_	_	☆	☆	☆	_	☆	☆
AMD	○	○	○	○	○	_	★	☆	☆	☆	☆	☆	☆

Meltdown Defense Categorization

REFRESHING MEMORIES

Meltdown defenses in 2 categories:

Meltdown Defense Categorization

REFRESHING MEMORIES

Meltdown defenses in 2 categories:

DI Architecturally inaccessible data is also microarchitecturally inaccessible

Meltdown Defense Categorization

REFRESHING MEMORIES

Meltdown defenses in 2 categories:

D1 Architecturally inaccessible data is also microarchitecturally inaccessible

D2 Preventing occurrence of faults

Meltdown-P Mitigation

REFRESHING MEMORIES

Meltdown-P Mitigation

REFRESHING MEMORIES

Clear phyiscal address field of unmapped PTEs

Meltdown-P Mitigation

Clear phyiscal address field of unmapped PTEs

Flush L1 upon switching protection domains

Demo

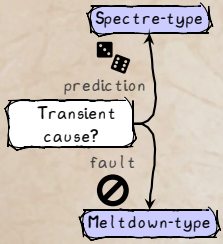
Foreshadow-NG

Transient Execution Attacks: Classification

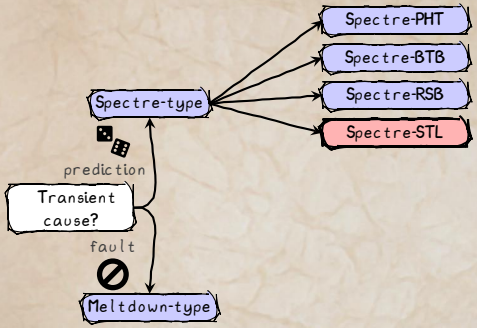
REFRESHING MEMORIES

Transient cause?

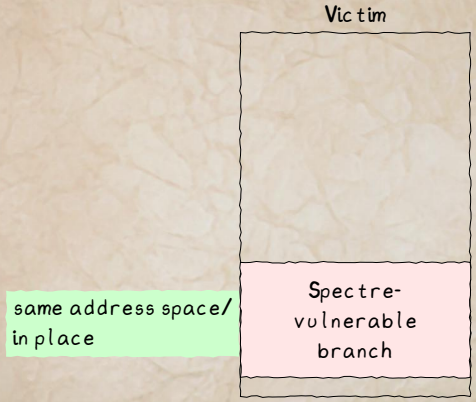
Transient Execution Attacks: Classification



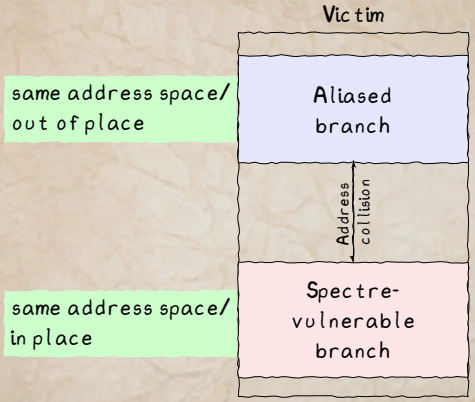
Transient Execution Attacks: Classification



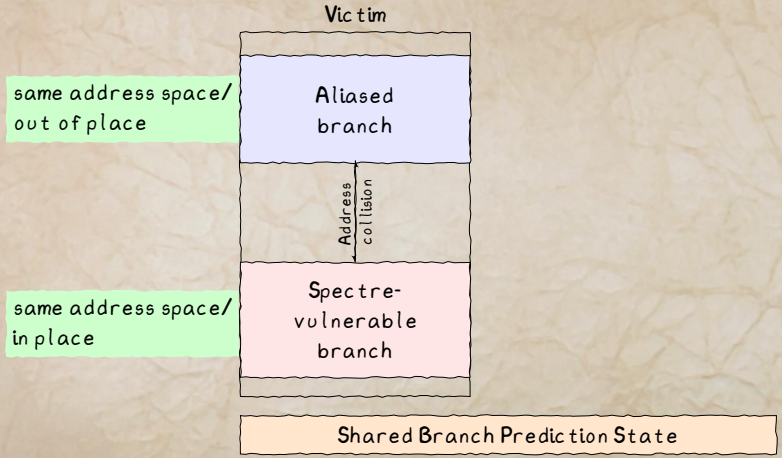
Spectre: Mistraining Strategies



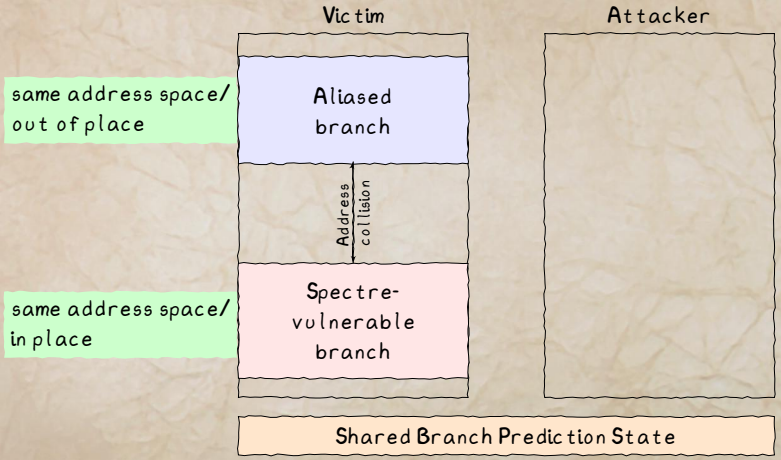
Spectre: Mistraining Strategies



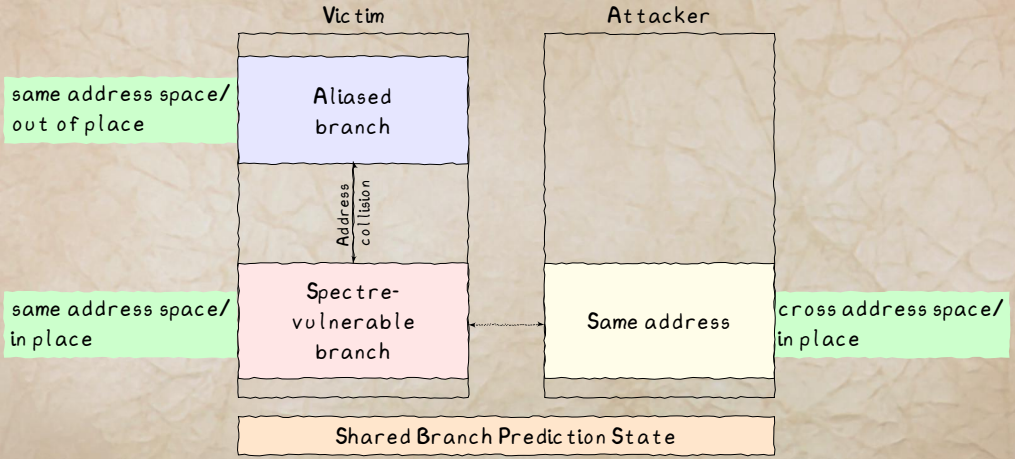
Spectre: Mistraining Strategies



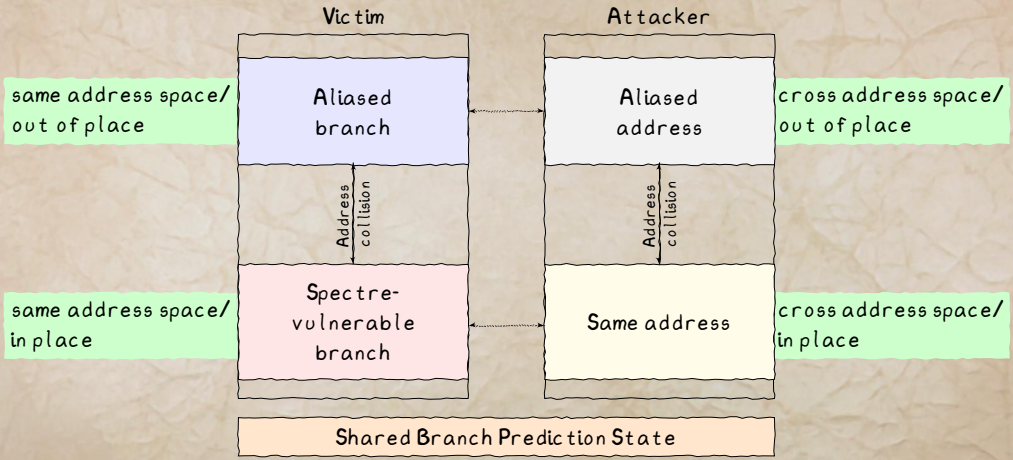
Spectre: Mistraining Strategies



Spectre: Mistraining Strategies

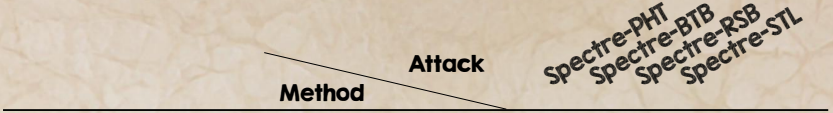


Spectre: Mistraining Strategies



Spectre Mistraining: Vulnerable Vendors

REFRESHING MEMORIES



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	★	●	●	○

Spectre Mistraining: Vulnerable Vendors

REFRESHING MEMORIES

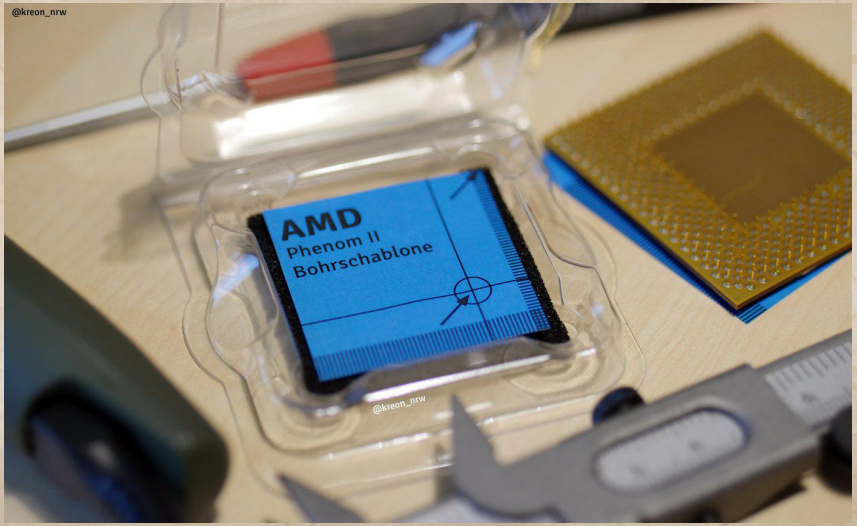
	Method	Attack	Spectre-PHT	Spectre-BTB	Spectre-RSB	Spectre-STL
Intel	same-address-space	in-place	●	★	●	●
		out-of-place	★	★	●	○
	cross-address-space	in-place	★	●	●	○
		out-of-place	★	●	●	○
ARM	same-address-space	in-place	●	★	●	●
		out-of-place	★	☆	●	○
	cross-address-space	in-place	★	●	☆	○
		out-of-place	★	☆	☆	○

Spectre Mistraining: Vulnerable Vendors

REFRESHING MEMORIES

	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	★	☆	★	○

Super Effective Solution: Drilling template



Drilling template (@kreon_nrw)

Spectre Defense Categorization

REFRESHING MEMORIES

Spectre defenses in 3 categories:

Spectre Defense Categorization

REFRESHING MEMORIES

Spectre defenses in 3 categories:

CI Mitigate or reduce accuracy of covert channels

Spectre Defense Categorization

REFRESHING MEMORIES

Spectre defenses in 3 categories:

C1 Mitigate or reduce accuracy of covert channels

C2 Mitigate or abort speculation

Spectre Defense Categorization

REFRESHING MEMORIES

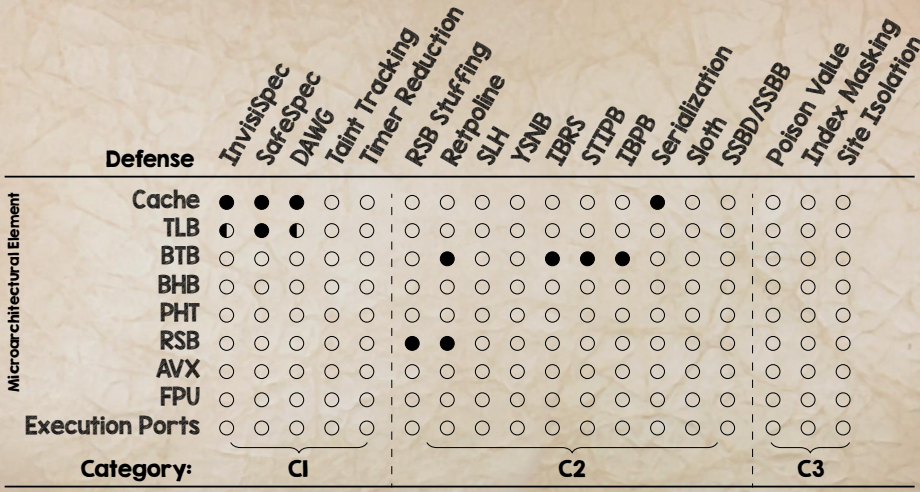
Spectre defenses in 3 categories:

C1 Mitigate or reduce accuracy of covert channels

C2 Mitigate or abort speculation

C3 Ensure secret cannot be reached

Spectre Defenses: Microarchitectural Target



Site Isolation

- Each site executed in its own process

Site Isolation

REFRESHING MEMORIES

- 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

Serialization

- 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

- Make transient loads invisible in the cache hierarchy

→ all transient loads use a speculative buffer

- 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

REFRESHING MEMORIES

Attack \ Defense	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

Spectre: Defense Analysis

REFRESHING MEMORIES

Attack \ Defense		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	□	□	□	◇	◇	◇	◇	◐	◇	◇	◇	◇	◇	◇	■	◐	■	●

Spectre: Defense Analysis

REFRESHING MEMORIES

Attack \ Defense		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	□	□	□	◇	◇	◇	◇	◐	◇	◇	◇	◇	◇	◇	■	◐	■	●

Spectre: Defense Analysis

REFRESHING MEMORIES

Attack \ Defense		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	□	□	□	◇	◇	◇	◇	◐	◇	◇	◇	◇	◇	◇	■	◐	■	●

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 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.

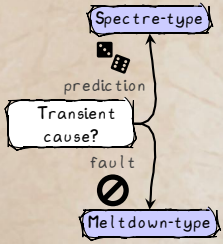
As covered yesterday, there is improved STIBP code on the way for Linux 4.20 that by default just apply STIBP to SECCOMP threads and processes requesting it via prctl() but otherwise is off by default (that behavior can also be changed via kernel parameters).

Transient Execution Attacks: Classification

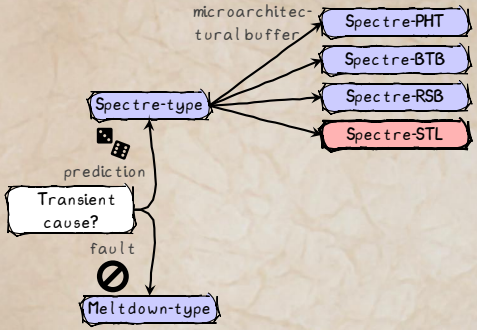
REFRESHING MEMORIES

Transient cause?

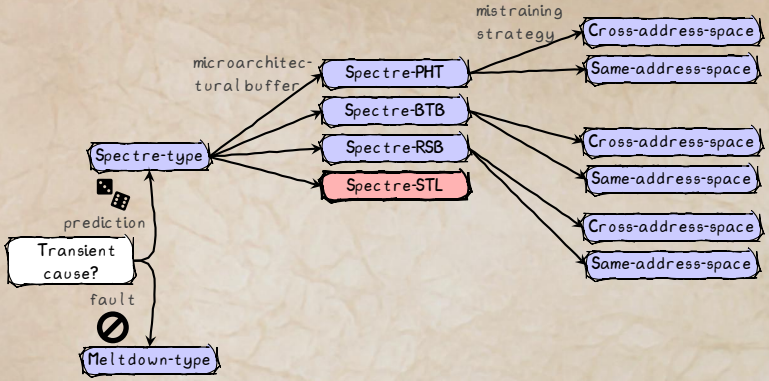
Transient Execution Attacks: Classification



Transient Execution Attacks: Classification

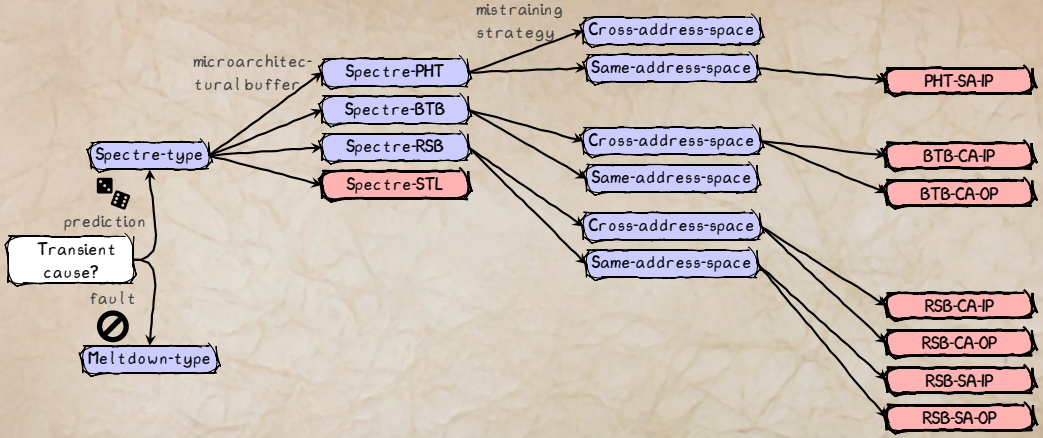


Transient Execution Attacks: Classification

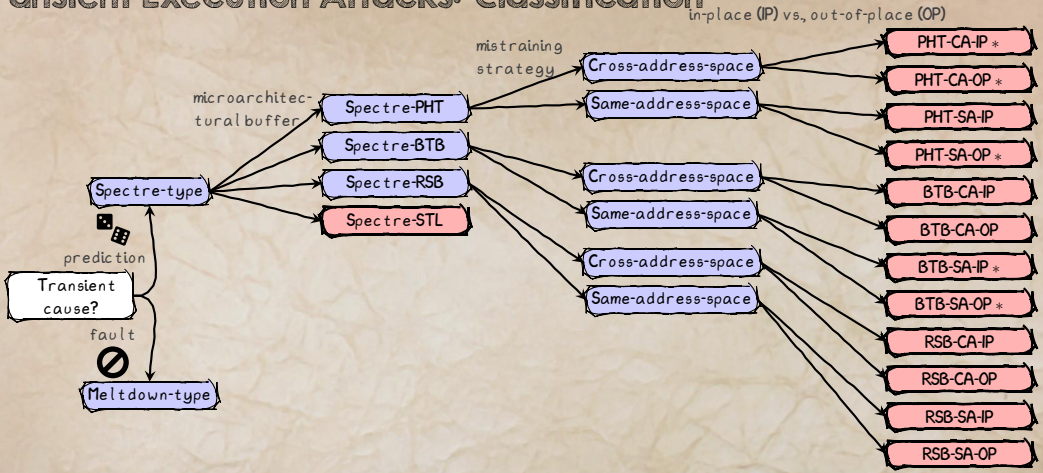


Transient Execution Attacks: Classification

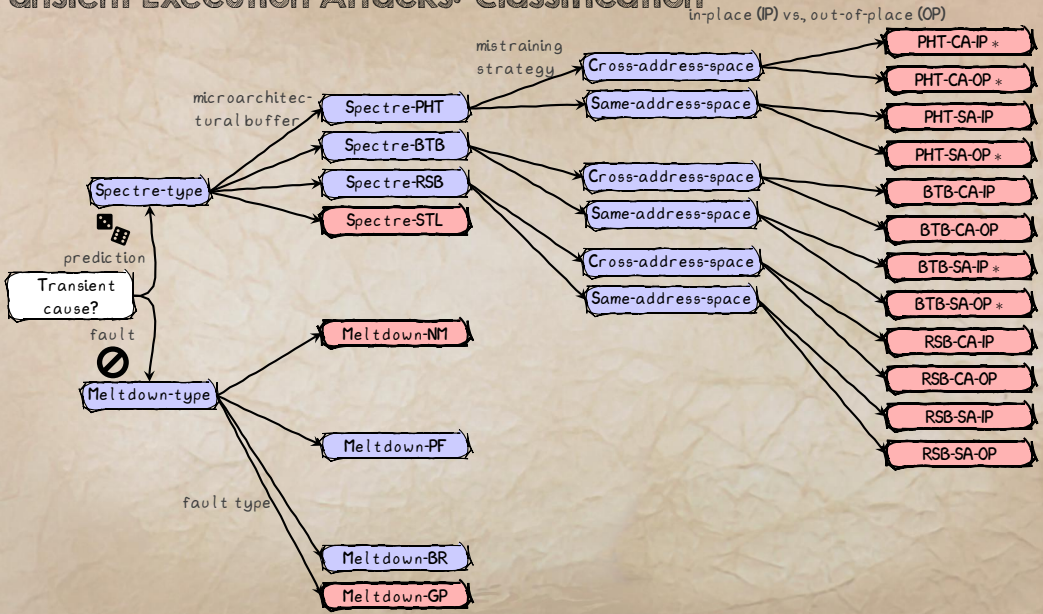
in-place (IP) vs. out-of-place (OP)



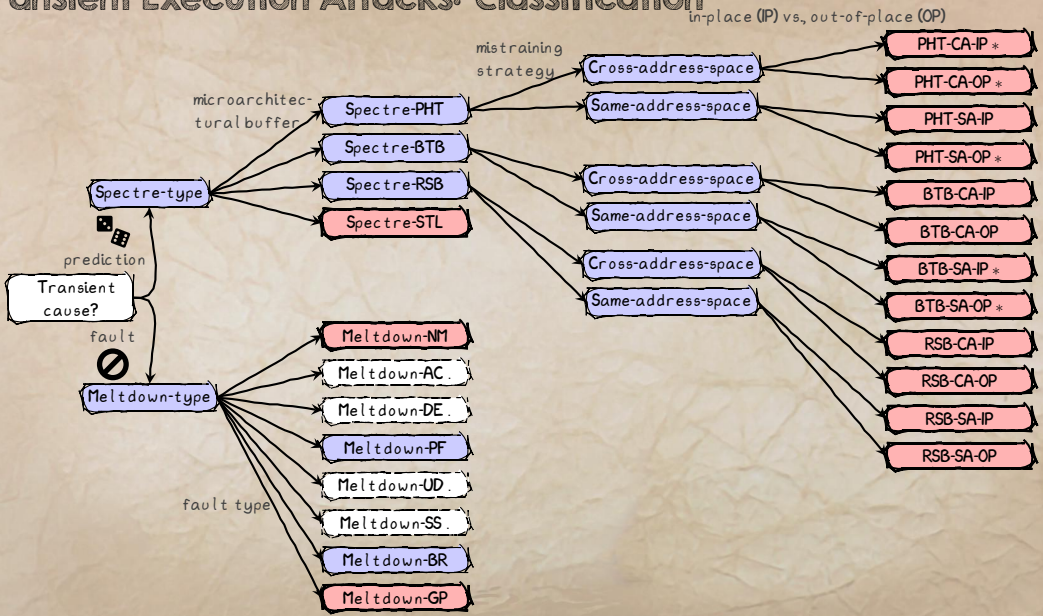
Transient Execution Attacks: Classification



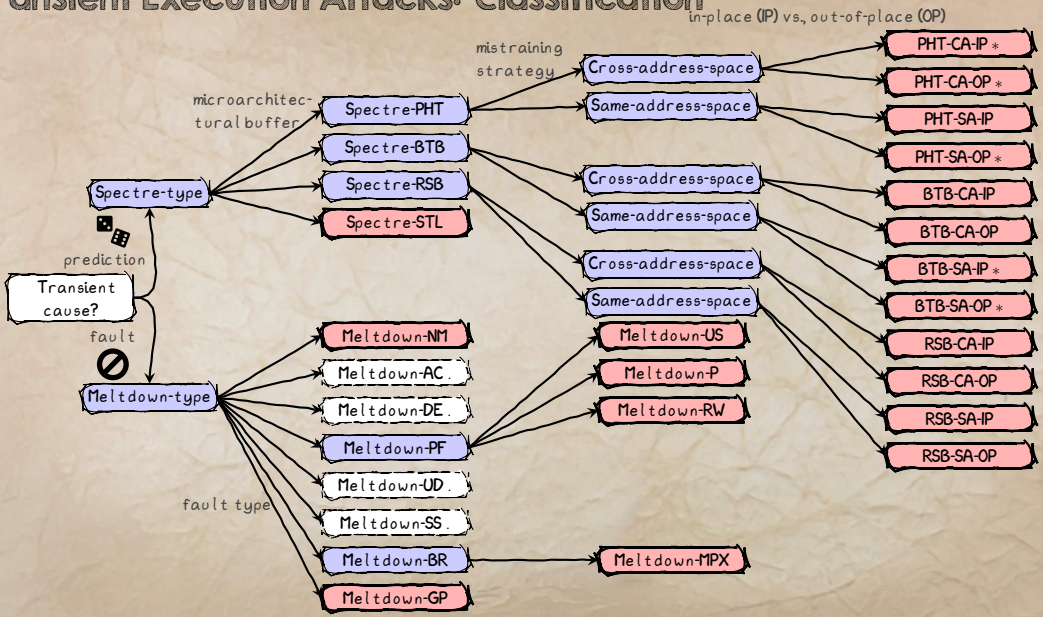
Transient Execution Attacks: Classification



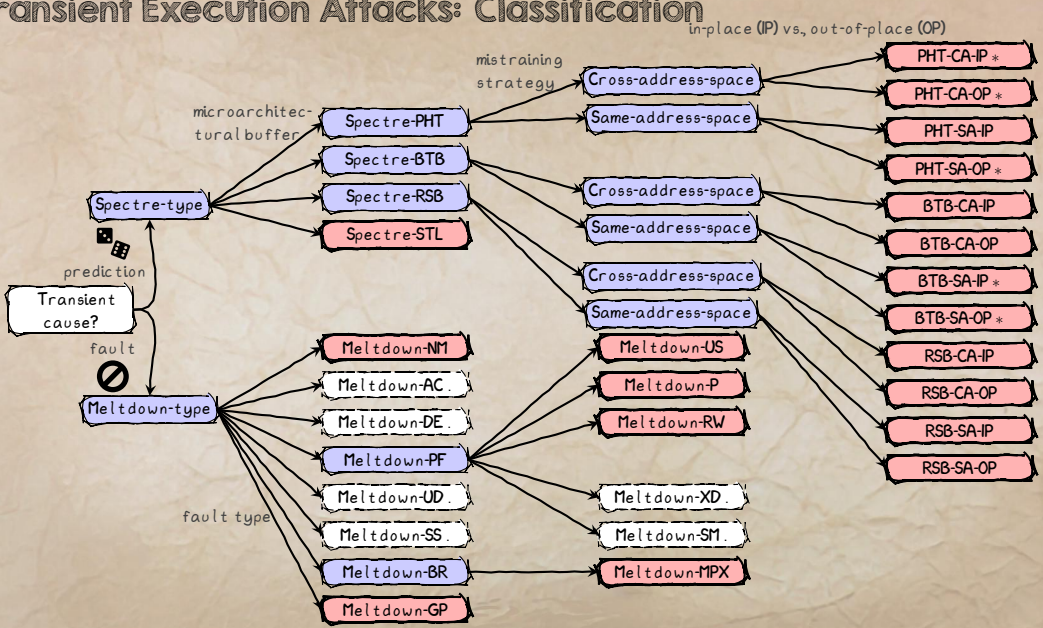
Transient Execution Attacks: Classification



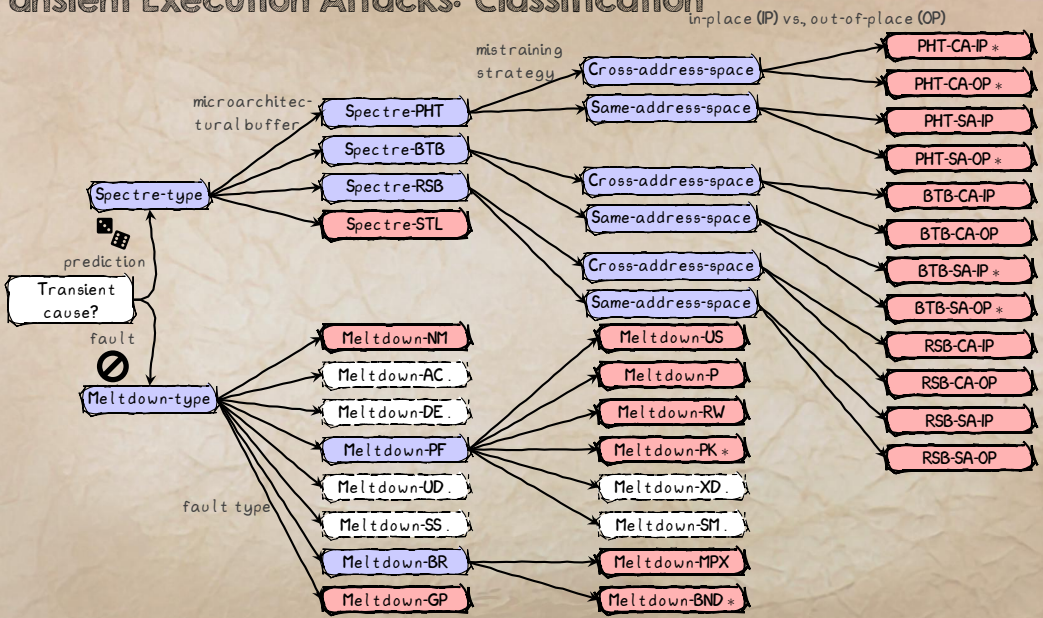
Transient Execution Attacks: Classification



Transient Execution Attacks: Classification

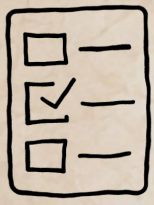


Transient Execution Attacks: Classification



Lessons learned

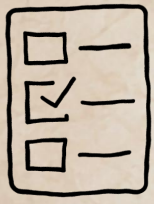
REFRESHING MEMORIES



We have ignored microarchitectural attacks for many years:

Lessons learned

REFRESHING MEMORIES

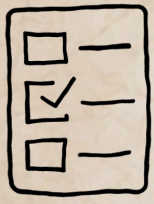


We have ignored microarchitectural attacks for many years:

- attacks on crypto

Lessons learned

REFRESHING MEMORIES

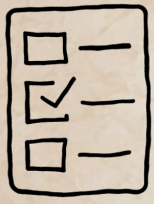


We have ignored microarchitectural attacks for many years:

- attacks on crypto → "software should be fixed"

Lessons learned

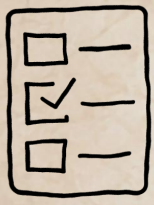
REFRESHING MEMORIES



We have ignored microarchitectural attacks for many years:

- attacks on crypto → "software should be fixed"
- attacks on ASLR

Lessons learned

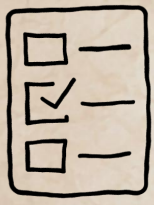


We have ignored microarchitectural attacks for many years:

- attacks on crypto → "software should be fixed"
- attacks on ASLR → "ASLR is broken anyway"

Lessons learned

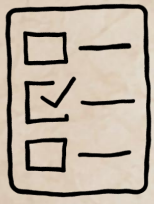
REFRESHING MEMORIES



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

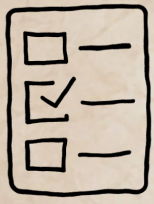


We have ignored microarchitectural attacks for many years:

- attacks on crypto → "software should be fixed"
- attacks on ASLR → "ASLR is broken anyway"
- attacks on TEEs → "not within threat model"

Lessons learned

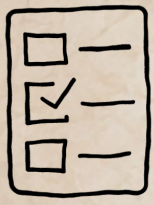
REFRESHING MEMORIES



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

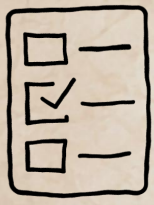
Lessons learned



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



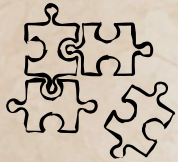
We have ignored microarchitectural attacks for many years:

- 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

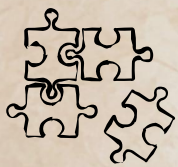
Conclusion

REFRESHING MEMORIES



Conclusion

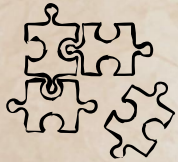
REFRESHING MEMORIES



- Optimizations always come at a cost

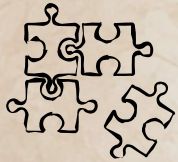
Conclusion

REFRESHING MEMORIES



- 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