Breaking through walls

How performance optimizations shatter security boundaries

Moritz Lipp

Mar 05, 2018—QCon London 2018

IAIK, Graz University of Technology

Whoami

- **Moritz Lipp**
- PhD student @ Graz University of Technology
- @mlqxyz
- mail@mlq.me


FOX
BUSINESS
WASHINGTON, D.C.
WASHINGTON, D.C.
WINTER STORM
NEWS
ALERT
INTEL REVEALS DESIGN FLAW THAT
COULD ALLOW HACKERS TO ACCESS DATA

DEVELOPING STORY

COMPUTER CHIP FLAWS IMPACT BILLIONS OF DEVICES

LIVE

CNN

DAX ▲164.69

NEWS STREAM


GLOBAL
COMPUTER CHIP SCARE
The bugs are known as 'Spectre' and 'Meltdown'
BBC WORLD NEWS
£:HK$ 10.58
EURO:£ 0.891

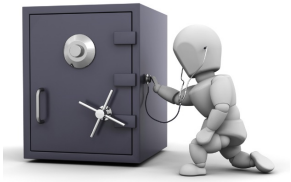
Motivation

- Two major vulnerabilities in processors have been disclosed
- Affecting every CPU vendor and, thus, billions of devices
- Discovered in 2017 by 4 independent teams
- News coverage followed by a lot of panic
- What is this all about and what are the consequences?

Performance optimizations

- Modern computers are amazingly fast
- Get faster and faster every year
- Smaller and smaller
- Include many clever optimizations to maximize performance
- What are the downsides?

Side-channel attacks

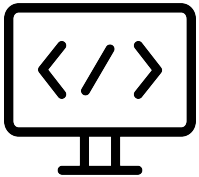


- Safe software infrastructure does not mean safe execution
- Information leaks because of the underlying hardware

Side-channel attacks

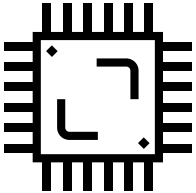
- Exploit unintentional information leakage by side-effects
  - Power consumption
  - Execution time
  - CPU cache
  - ...
- Performance optimizations often induce side-channel leakage

Software-based side-channel attacks



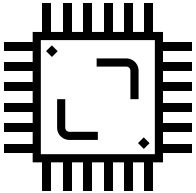
- Do not require physical access
- Mounted solely by software
  - native code
  - within the browser

Architecture and Microarchitecture



- Instruction Set Architecture (ISA) is an abstract model of a computer (x86, ARMv8, SPARC, ...)
- Serves as the interface between hardware and software
- Microarchitecture is an actual implementation of the instruction set
  - Vary in performance, size, costs, ...
  - Intel (Pentium, Sandy Bridge, Skylake, ...)
  - AMD (Athlon, Bobcat, Zen, ...)

Microarchitectural Side-channel attacks

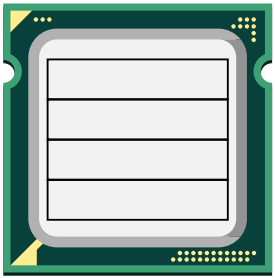


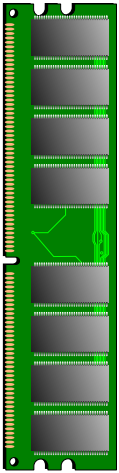
- Side-channel attacks on the implementation of an ISA
- Expose internal state of the hardware
  - depending on secret data
  - to infer the secret data

Caches and Cache Attacks

CPU Cache

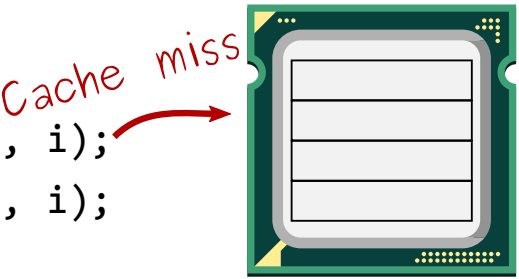
```
printf("%d", i);
printf("%d", i);
```

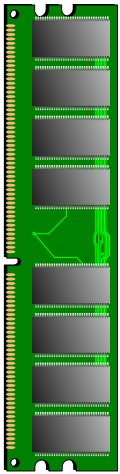




CPU Cache

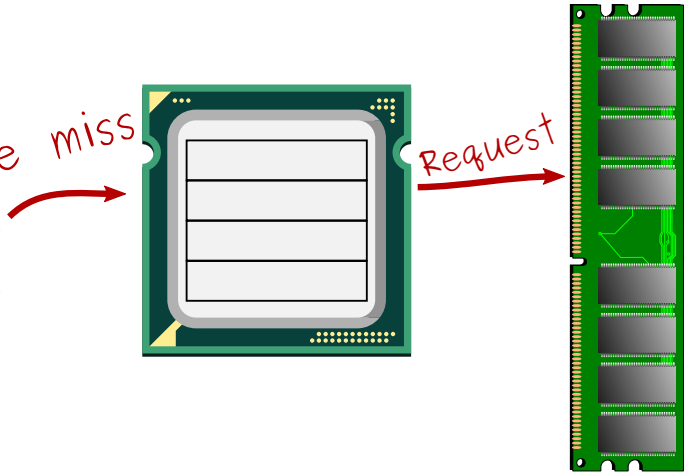
```
printf("%d", i);
printf("%d", i);
```





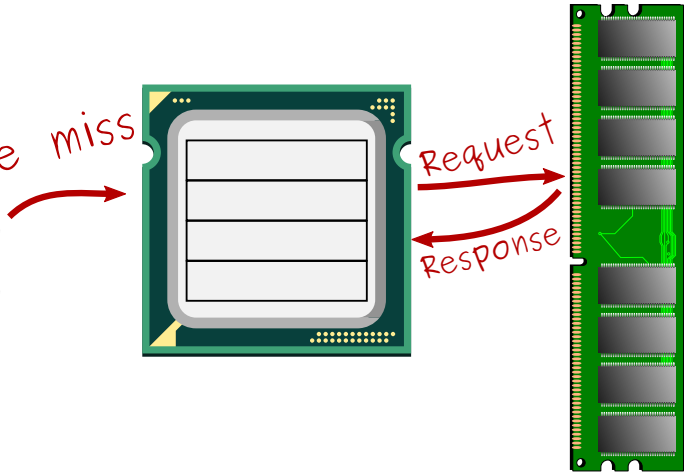
CPU Cache

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



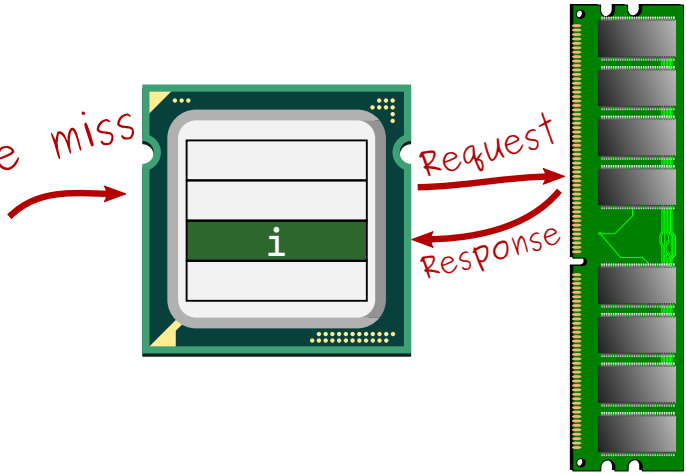
CPU Cache

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

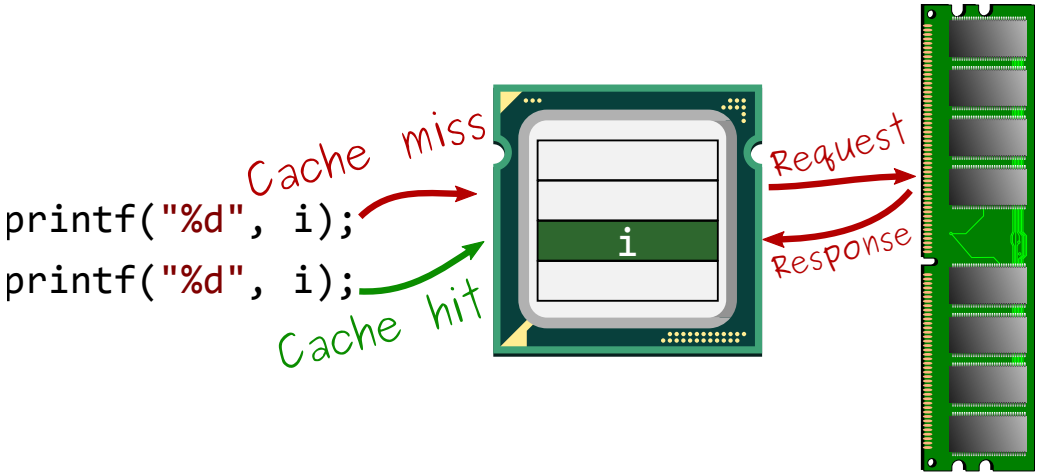


CPU Cache

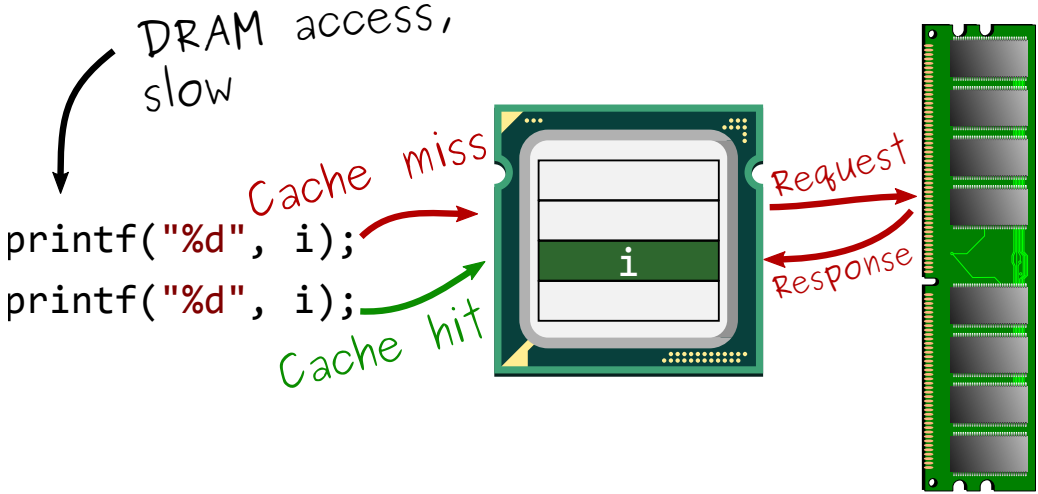
```
printf("%d", i);
printf("%d", i);
```



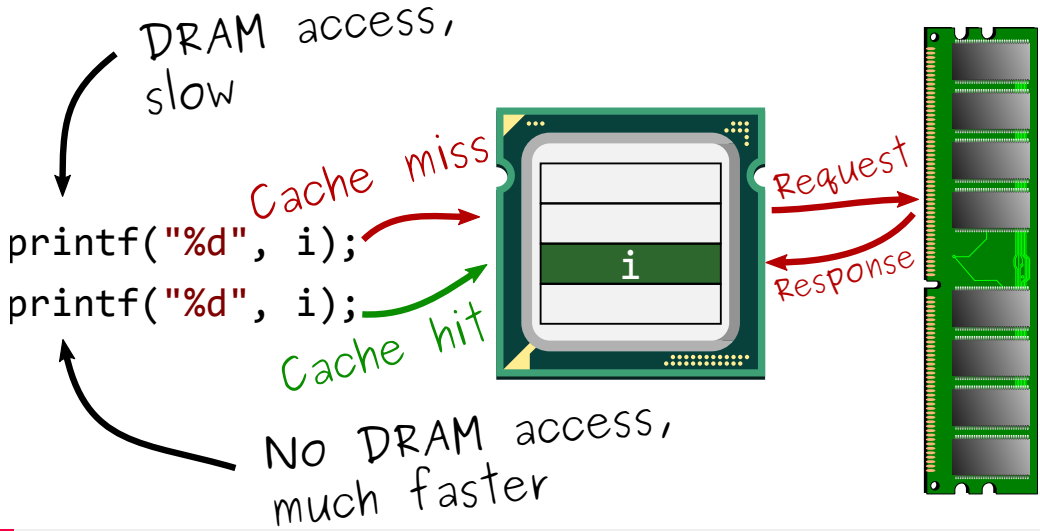
CPU Cache



CPU Cache


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

CPU Cache

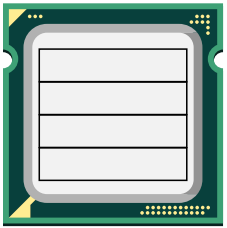

DRAM access, slow
printf("%d", i);
printf("%d", i);
Cache miss
Cache hit
i
Request
Response
No DRAM access, much faster

Shared Memory

ATTACKER

flush

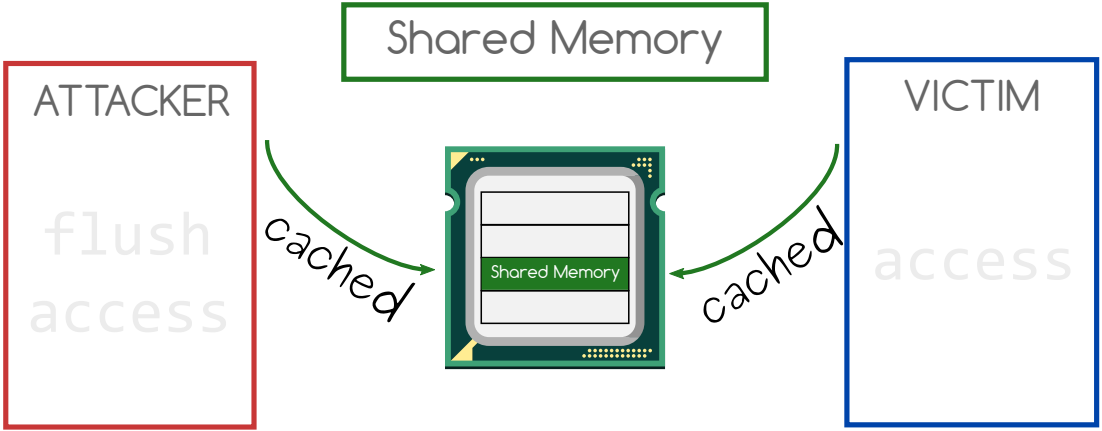
access



VICTIM

access

Flush+Reload

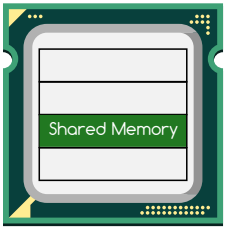


Shared Memory

ATTACKER

flush

access



VICTIM

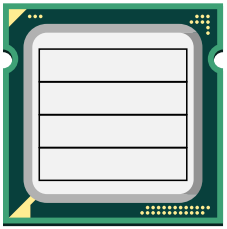
access

Shared Memory

ATTACKER

flush

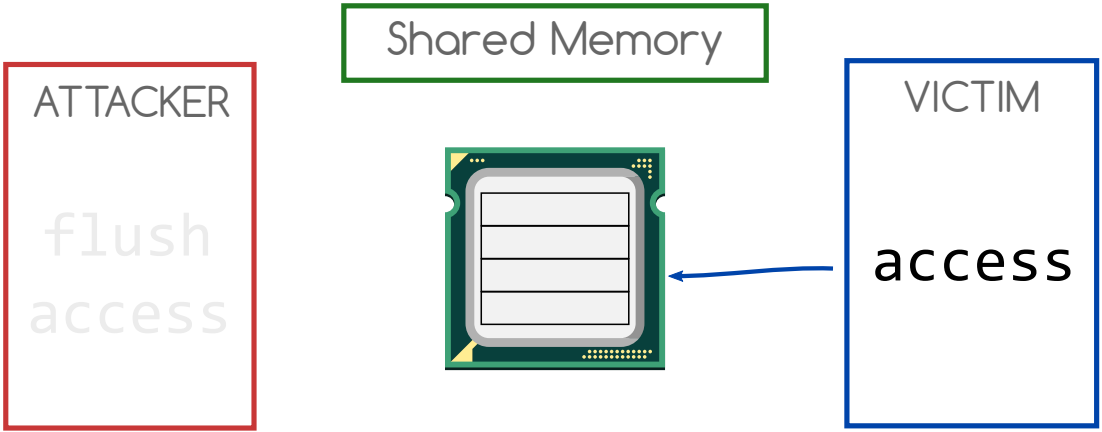
access



VICTIM

access

Flush+Reload



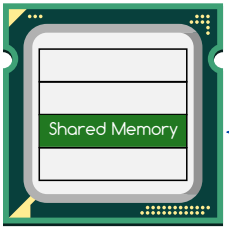
Flush+Reload

Shared Memory

ATTACKER

flush

access



VICTIM

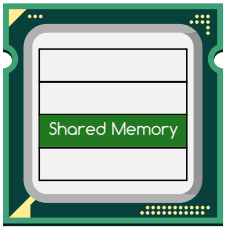
access

Shared Memory

ATTACKER

flush

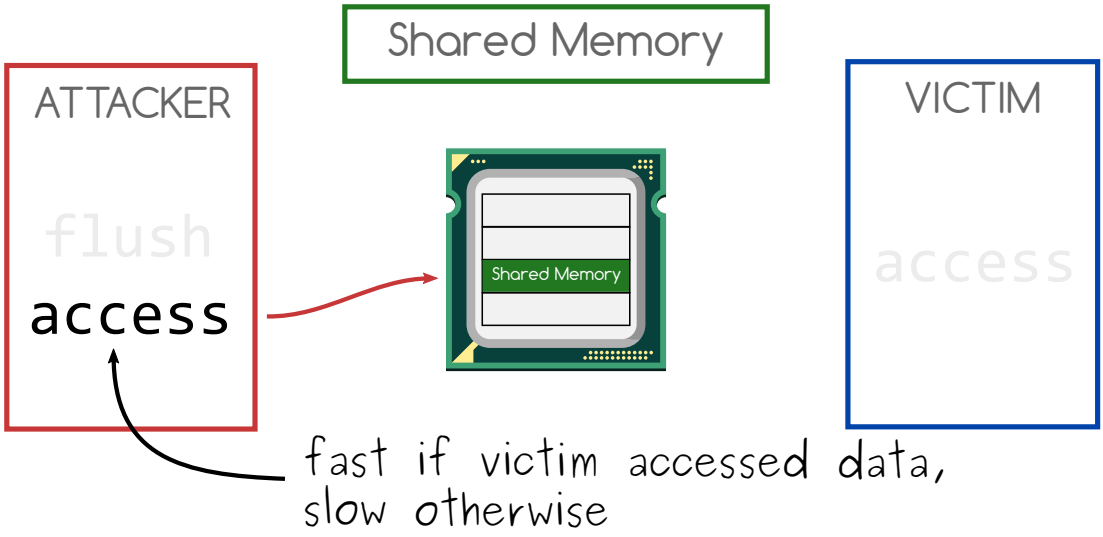
access



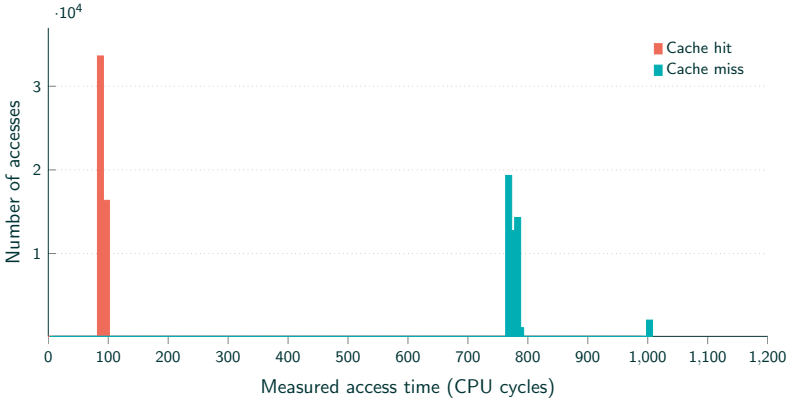
VICTIM

access

Flush+Reload



Memory Access Latency



Cache Attacks

- Leak cryptographic keys
- Leak information on co-located virtual machines
- Monitor function calls of other applications
- Break (K)ASLR
- Allow Rowhammer attack in software
- Build covert communication channels

File Edit View Search Terminal Help

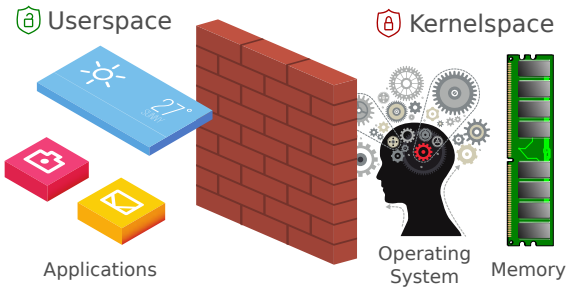
```
shell@zeroflte:/data/local/tmp $ ./keyboard_spy -c 0
```

Cache Attack Demo

Operating Systems 101

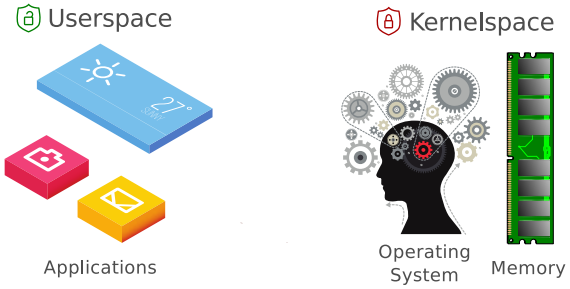
Core of Meltdown and Spectre

- Kernel is isolated from user space
- This isolation is a combination of hardware and software
- User applications cannot access anything from the kernel
- There is only a well-defined interface called system calls



Core of Meltdown and Spectre

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



Out-of-order execution and Meltdown

7. Serve with cooked and peeled potatoes

Wait for an hour

Wait for an hour

LATENCY

1. Wash and cut vegetables

2. Pick the basil leaves and set aside

3. Heat 2 tablespoons of oil in a pan

4. Fry vegetables until golden and softened

Dependency

1. Wash and cut vegetables

2. Pick the basil leaves and set aside

3. Heat 2 tablespoons of oil in a pan

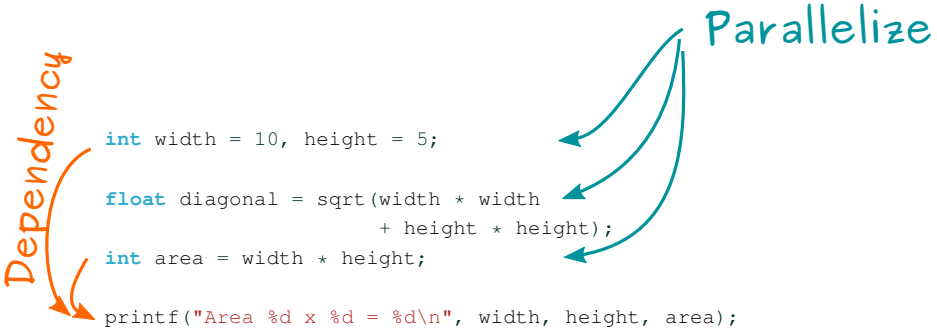
4. Fry vegetables until golden and softened

Parallelize

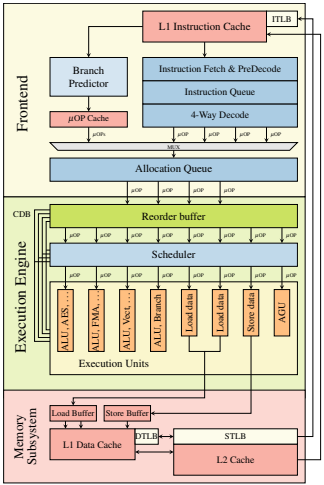
Out-of-order Execution

```
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

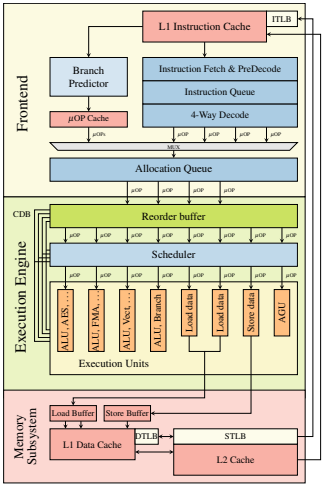


Out-of-order execution



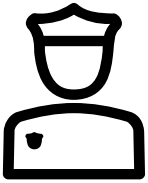
- Instructions are fetched and decoded in the front-end
- Instructions are dispatched to the backend
- Instructions are processed by individual execution units

Out-of-order execution



- Instructions are executed out-of-order
- Instructions wait until their dependencies are ready
  - Later instructions might execute prior earlier instructions
- Instructions retire in-order
  - State becomes architecturally visible

Reading memory



- If an application reads memory, ...
  - ... permissions are checked
  - ... data is loaded
- If an application tries to read inaccessible memory, ...
  - ... an error occurs
  - ... application is stopped
- But what does the CPU really do?

Let’s try to read kernel memory

Building the Code

- Find something human readable, e.g., the Linux version

```
# sudo grep linux_banner /proc/kallsyms
ffffffff81a000e0 R linux_banner
```

Building the Code

```
char data = *(char*) 0xffffffff81a000e0;
printf("%c\n", data);
```

Building the Code

- Compile and run

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

Building the Code

- Compile and run

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

- Kernel addresses are of course not accessible

Building the Code

- Compile and run

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

- Kernel addresses are of course not accessible
- Any invalid access throws an exception → segmentation fault

Building the Code

- Just catch the segmentation fault!

Building the Code

- Just catch the segmentation fault!
- We can simply install a signal handler

Building the Code

- Just catch the segmentation fault!
- We can simply install a signal handler
- And if an exception occurs, just jump back and continue

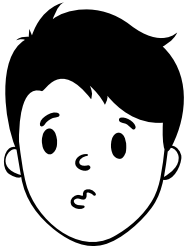
Building the Code

- Just catch the segmentation fault!
- We can simply install a signal handler
- And if an exception occurs, just jump back and continue
- Then we can read the value

Building the Code

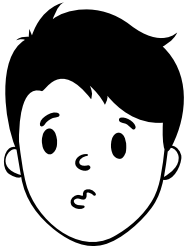
- Just catch the segmentation fault!
- We can simply install a signal handler
- And if an exception occurs, just jump back and continue
- Then we can read the value
- Sounds like a good idea

Building the Code



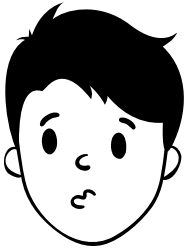
- Still no kernel memory

Building the Code



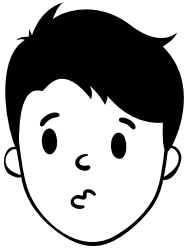
- Still no kernel memory
- Maybe it is not that straight forward

Building the Code



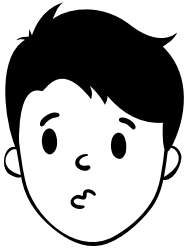
- Still no kernel memory
- Maybe it is not that straight forward
- Privilege checks seem to work

Building the Code



- Still no kernel memory
- Maybe it is not that straight forward
- Privilege checks seem to work
- Are privilege checks also done when executing instructions out of order?

Building the Code



- Still no kernel memory
- Maybe it is not that straight forward
- Privilege checks seem to work
- Are privilege checks also done when executing instructions out of order?
- Problem: out-of-order instructions are not visible

Building the Code

- Adapted code

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

Building the Code

- Adapted code

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

- `volatile` because compiler was not happy

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

Building the Code

- Adapted code

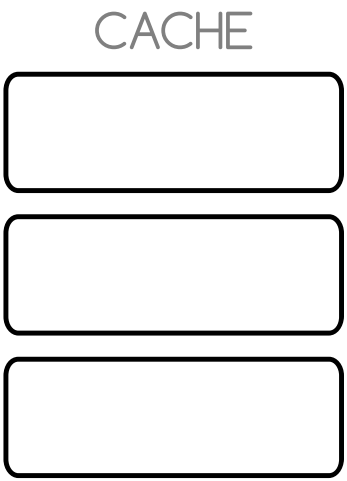
```
*(volatile char*) 0;
array[0] = 0;
```

- `volatile` because compiler was not happy

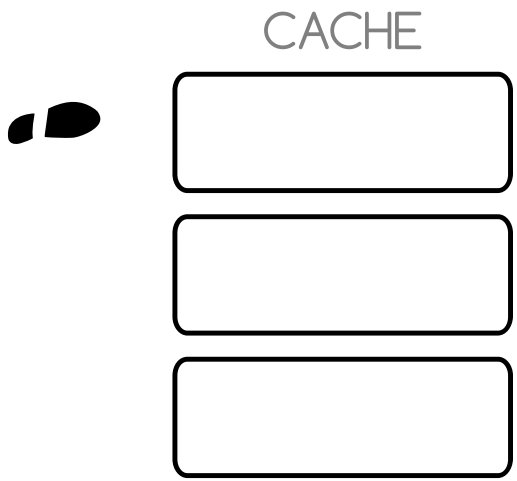
```
warning: statement with no effect [-Wunused-
    value]
          *(char*) 0;
```

- Static code analyzer is still not happy

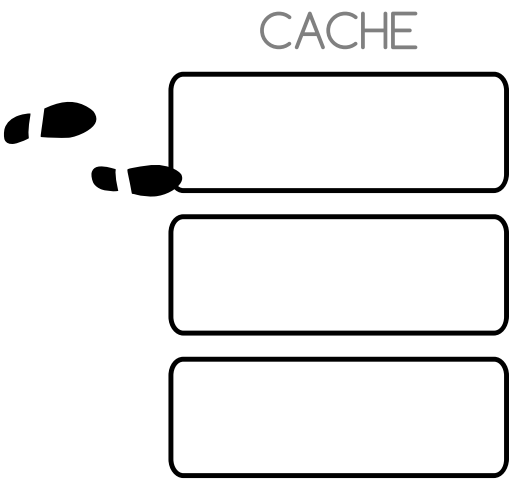
Traces in the Cache



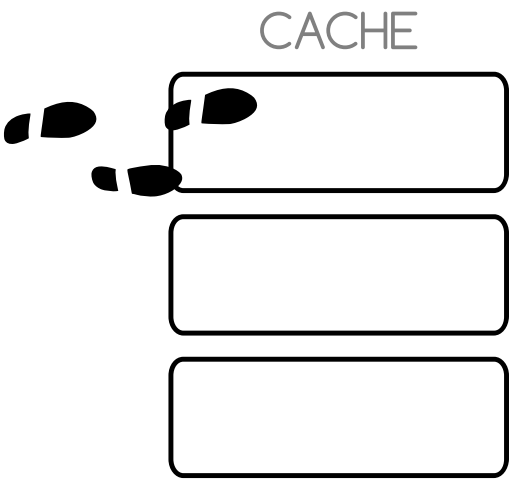
Traces in the Cache



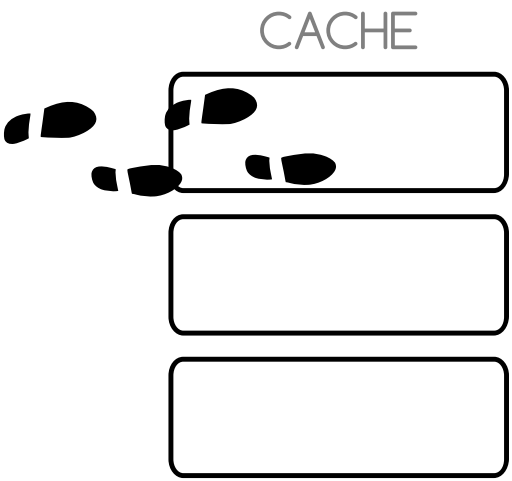
Traces in the Cache



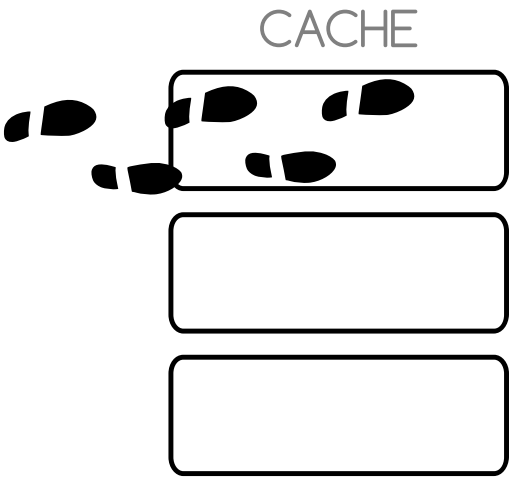
Traces in the Cache



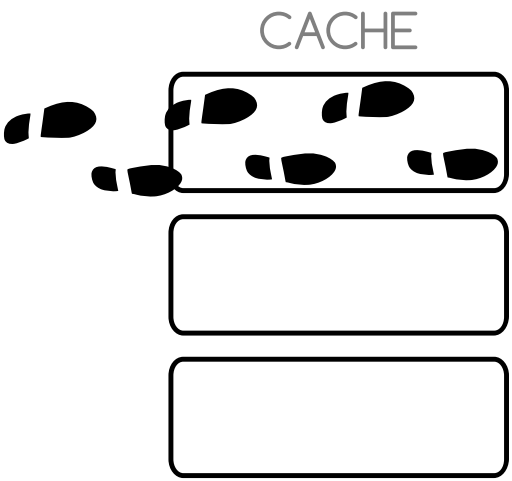
Traces in the Cache



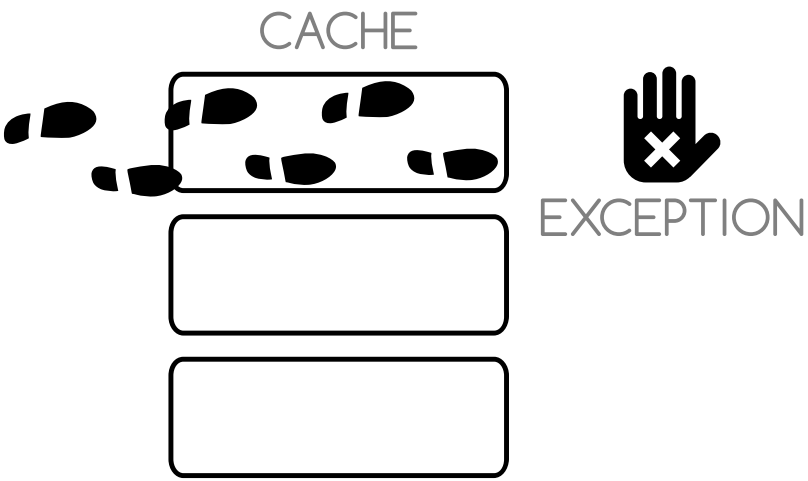
Traces in the Cache



Traces in the Cache



Traces in the Cache



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
- Give such instructions a name: transient instructions

Building the Code

- Out-of-order instructions leave microarchitectural traces
- We can see them for example in the cache
- Give such instructions a name: transient instructions
- We can indirectly observe the execution of transient instructions

Building the Code

- Maybe there is no permission check in transient instructions...

Building the Code

- Maybe there is no permission check in transient instructions...
- ...or it is only done when commiting them

Building the Code

- Maybe there is no permission check in transient instructions...
- ...or it is only done when commiting them
- Add another layer of indirection to test

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

Building the Code

- Maybe there is no permission check in transient instructions...
- ...or it is only done when commiting them
- Add another layer of indirection to test

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

- Then check whether 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 is in some cases not fast enough

Building Meltdown

- Using out-of-order execution, we can read data at any address

Building Meltdown

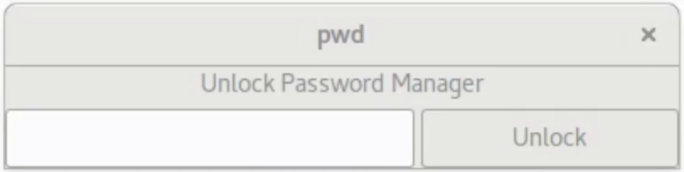
- Using out-of-order execution, we can read data at any address
- Privilege checks are sometimes too slow

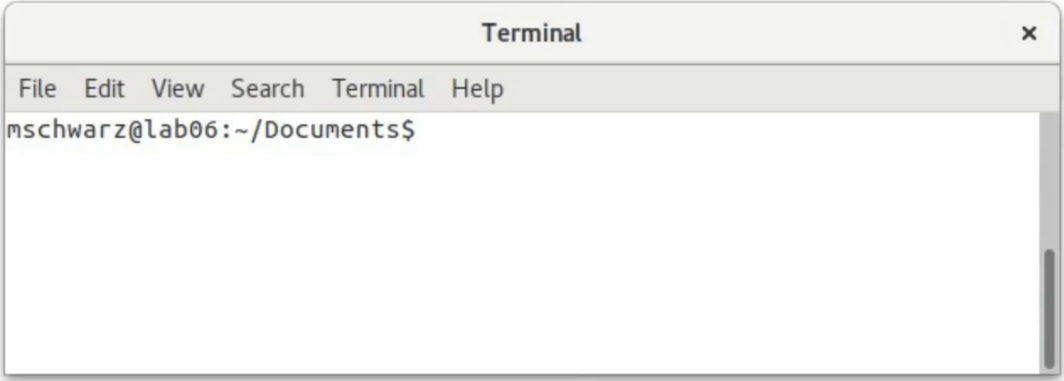
Building Meltdown

- Using out-of-order execution, we can read data at any address
- Privilege checks are sometimes too slow
- Allows to leak kernel memory

Building Meltdown

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


pwd
Unlock Password Manager
Unlock


Terminal
File Edit View Search Terminal Help
mschwarz@lab06:~/Documents$

```
e01d8130: 20 75 73 65 64 20 77 69  74 68 20 61 75 74 68 6f | used with autho|
e01d8140: 72 69 7a 61 74 69 6f 6e  20 66 72 6f 6d 0a 20 53 |rization from. S|
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 20 0a | 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|
e01d8290: 65 73 20 66 72 6f 6d 20  74 68 65 20 6c 69 62 67 |es from the libg|
```

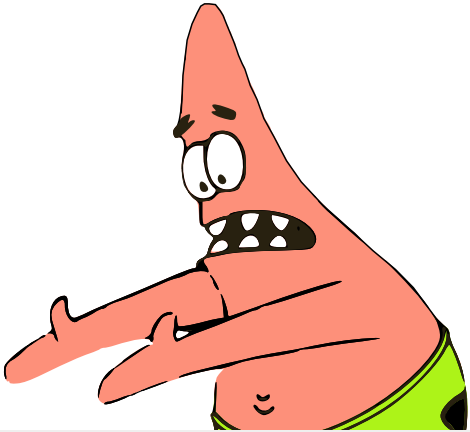
Can we fix that?

Take the kernel addresses...

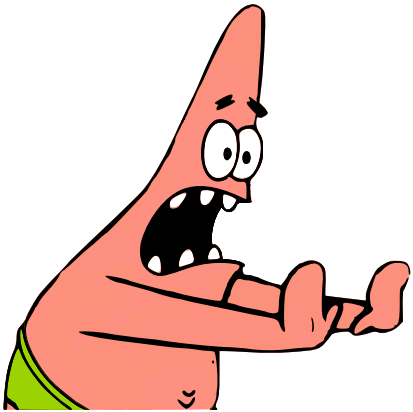
- Kernel addresses in user space are a problem

Take the kernel addresses...

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

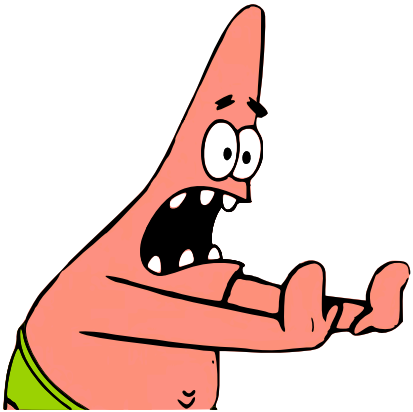


...and remove them



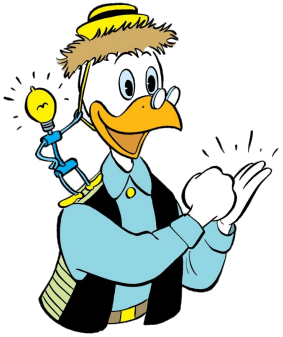
- ...and remove them if not needed?

...and remove them



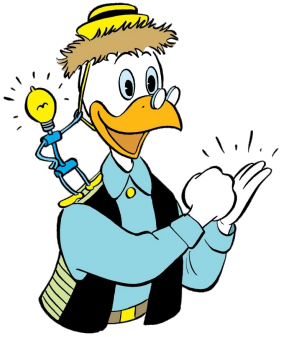
- ...and remove them if not needed?
- User accessible check in hardware is not reliable

Idea



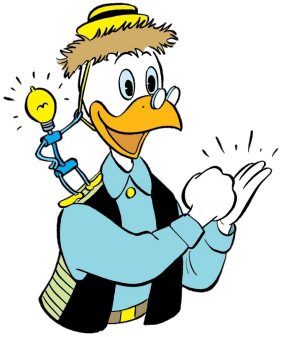
- Let's just unmap the kernel in user space

Idea

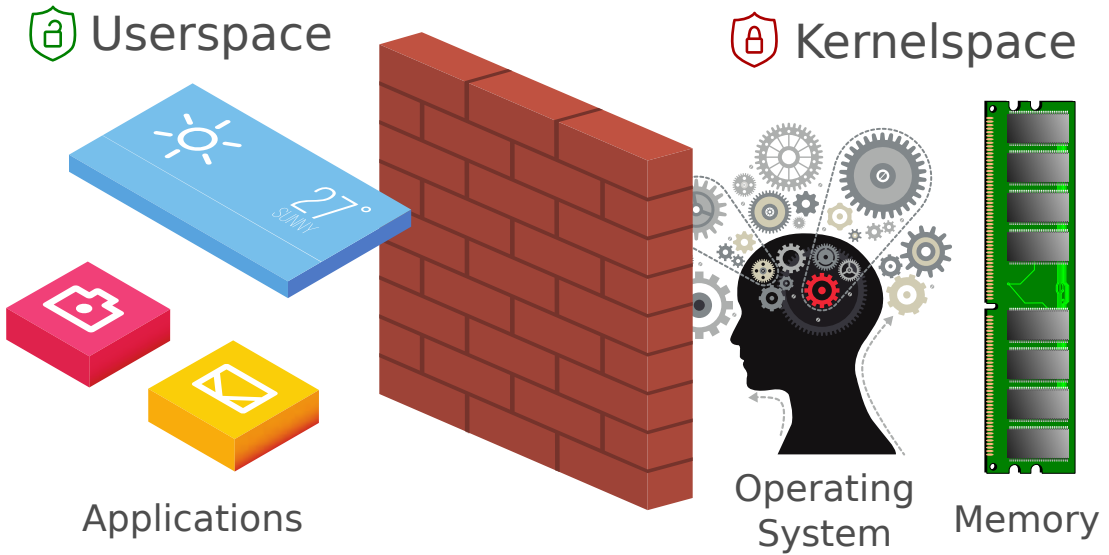


- Let's just unmap the kernel in user space
- Kernel addresses are then no longer present

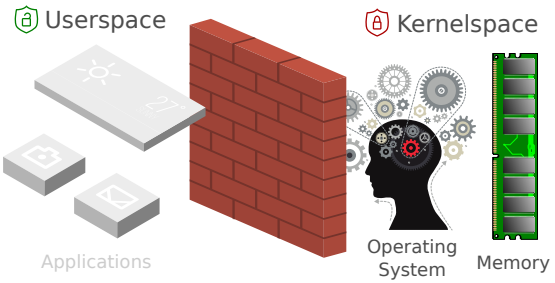
Idea



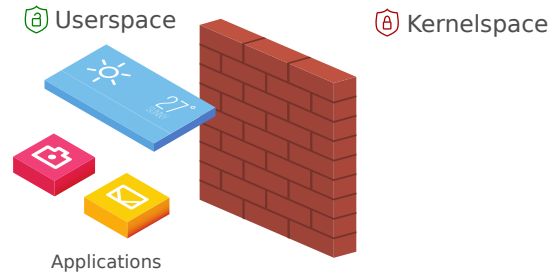
- Let's just unmap the kernel in user space
- Kernel addresses are then no longer present
- Memory which is not mapped cannot be accessed at all


Userspace
Kernelspace
27°
SUNNY
Applications
Operating System
Memory

Kernel View

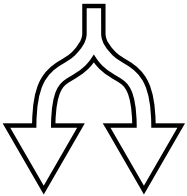


User View

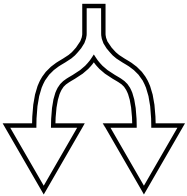


Kernel Address Space Isolation

- We published KAISER in July 2017

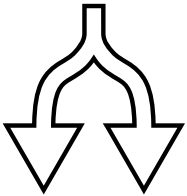


Kernel Address Space Isolation



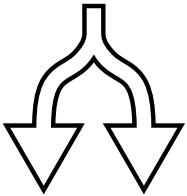
- We published KAISER in July 2017
- Intel and others improved and merged it into Linux as KPTI (Kernel Page Table Isolation)

Kernel Address Space Isolation



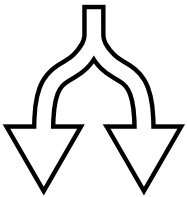
- We published KAISER in July 2017
- Intel and others improved and merged it into Linux as KPTI (Kernel Page Table Isolation)
- Microsoft implemented similar concept in Windows 10

Kernel Address Space Isolation



- We published KAISER in July 2017
- Intel and others improved and merged it into Linux as KPTI (Kernel Page Table Isolation)
- Microsoft implemented similar concept in Windows 10
- Apple implemented it in macOS 10.13.2 and called it "Double Map"

Kernel Address Space Isolation



- We published KAISER in July 2017
- Intel and others improved and merged it into Linux as KPTI (Kernel Page Table Isolation)
- Microsoft implemented similar concept in Windows 10
- Apple implemented it in macOS 10.13.2 and called it "Double Map"
- All share the same idea: switching address spaces on context switch

Performance

- Depends on how often you need to switch between kernel and user space

Performance

- Depends on how often you need to switch between kernel and user space
- Can be slow, 40% or more on old hardware

Performance

- Depends on how often you need to switch between kernel and user space
- Can be slow, 40% or more on old hardware
- But modern CPUs have additional features

Performance

- Depends on how often you need to switch between kernel and user space
- Can be slow, 40% or more on old hardware
- But modern CPUs have additional features
- $\Rightarrow$ Performance overhead on average below 2%

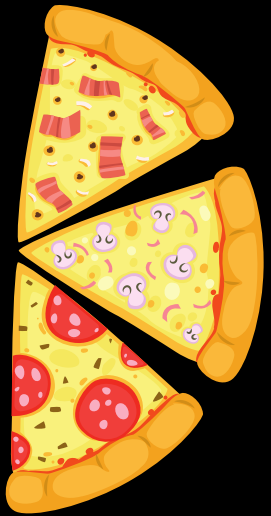
Speculative Execution and Spectre

PIZZA

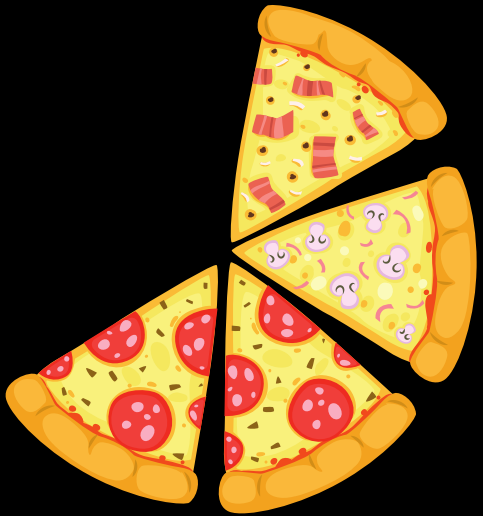
SPECIAL RECIPES

Procsciutto

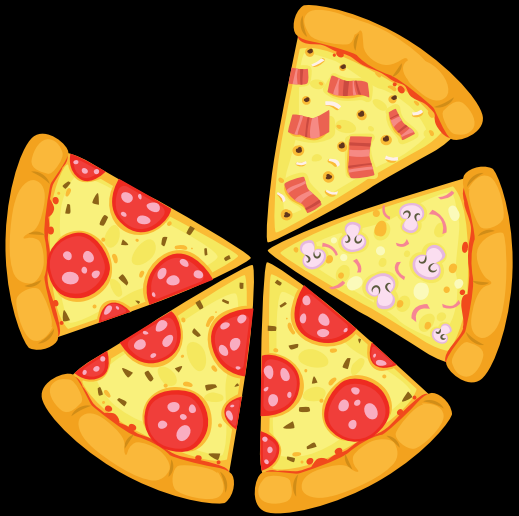
Funghi



Diavolo



Diavolo



Diavolo

Diavolo


»A table for 6 please«

Speculative Cooking


»A table for 6 please«


PIZZA
SPECIAL RECIPES


PIZZA
SPECIAL RECIPES


Pizza


PIZZA
SPECIAL RECIPES


Pizza


PIZZA
SPECIAL RECIPES


Pizza

Speculative Execution

- CPU tries to predict the future (branch predictor), ...
  - ... based on events learned in the past
- Speculative execution of instructions
- If the prediction was correct, ...
  - ... very fast
  - otherwise: Discard results
- Measurable side-effects?

Spectre (Variant 1: Bounds-check bypass)

```
index = 0;

char* data = "textKEY";

if (index < 4)
```


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

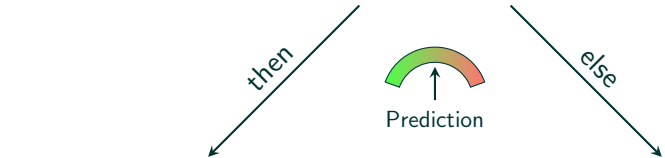
0

Spectre (Variant 1: Bounds-check bypass)

```
index = 0;
```

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

```
if (index < 4)
```




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

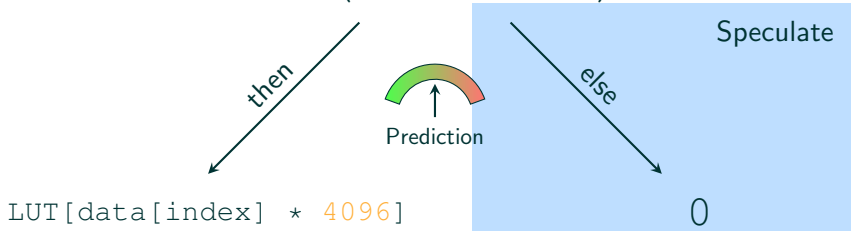
0

Spectre (Variant 1: Bounds-check bypass)

```
index = 0;
```

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

```
if (index < 4)
```

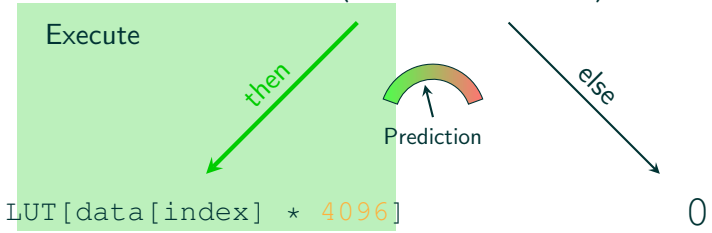


Spectre (Variant 1: Bounds-check bypass)

```
index = 0;
```

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

```
if (index < 4)
```



Spectre (Variant 1: Bounds-check bypass)

```
index = 1;

char* data = "textKEY";

if (index < 4)
```

then

else

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

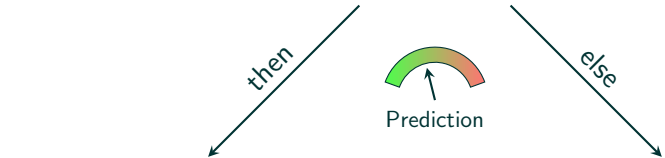
0

Spectre (Variant 1: Bounds-check bypass)

```
index = 1;
```

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

```
if (index < 4)
```

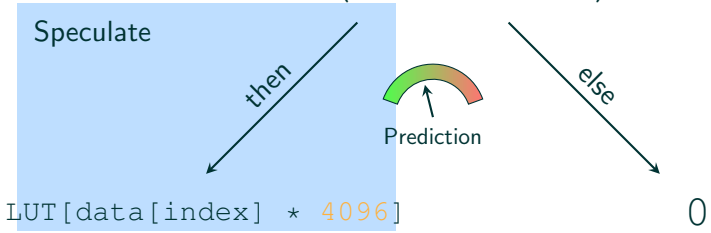


Spectre (Variant 1: Bounds-check bypass)

```
index = 1;
```

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

```
if (index < 4)
```



Spectre (Variant 1: Bounds-check bypass)

```
index = 1;
```

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

```
if (index < 4)
```

Spectre (Variant 1: Bounds-check bypass)

```
index = 2;

char* data = "textKEY";

if (index < 4)
```


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

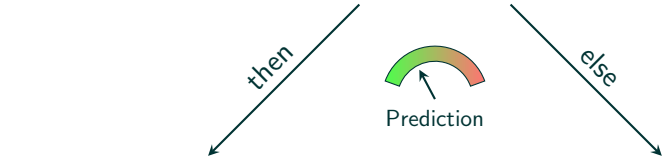
0

Spectre (Variant 1: Bounds-check bypass)

```
index = 2;
```

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

```
if (index < 4)
```

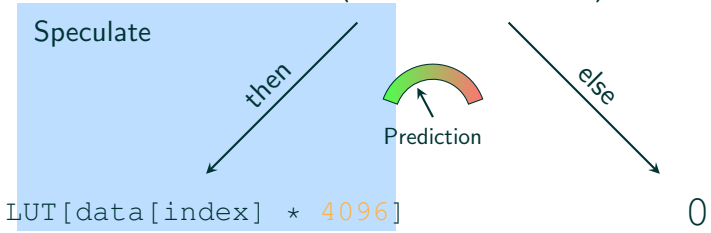


Spectre (Variant 1: Bounds-check bypass)

```
index = 2;
```

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

```
if (index < 4)
```



Spectre (Variant 1: Bounds-check bypass)

```
index = 2;
```

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

```
if (index < 4)
```

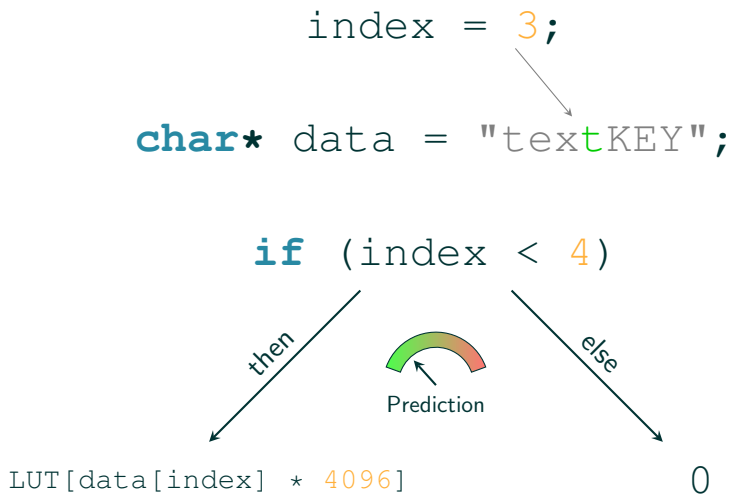
Spectre (Variant 1: Bounds-check bypass)

```
index = 3;

char* data = "textKEY";

if (index < 4)
```


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

0

Spectre (Variant 1: Bounds-check bypass)

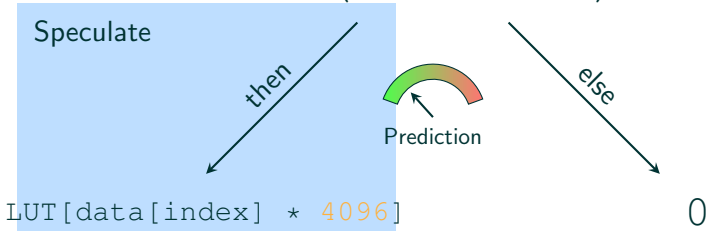


Spectre (Variant 1: Bounds-check bypass)

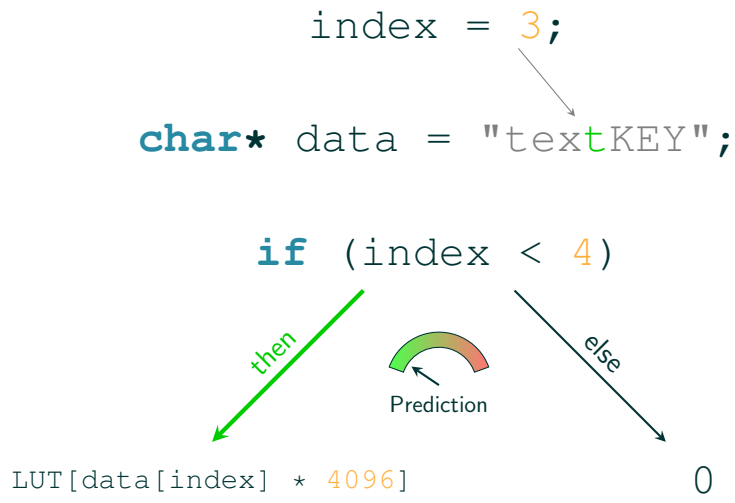
```
index = 3;
```

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

```
if (index < 4)
```



Spectre (Variant 1: Bounds-check bypass)



Spectre (Variant 1: Bounds-check bypass)

```
index = 4;

char* data = "textKEY";

if (index < 4)
```

`LUT[data[index] * 4096]` 0

Spectre (Variant 1: Bounds-check bypass)

```
index = 4;
```

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

```
if (index < 4)
```


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

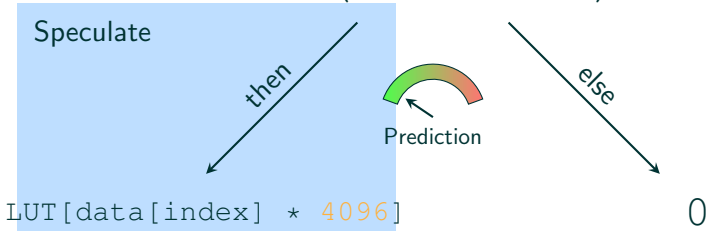
0

Spectre (Variant 1: Bounds-check bypass)

```
index = 4;
```

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

```
if (index < 4)
```

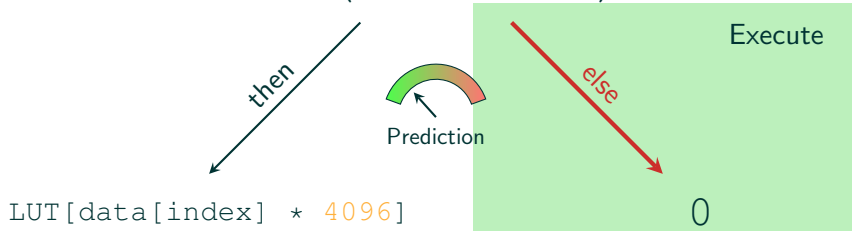


Spectre (Variant 1: Bounds-check bypass)

```
index = 4;
```

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

```
if (index < 4)
```

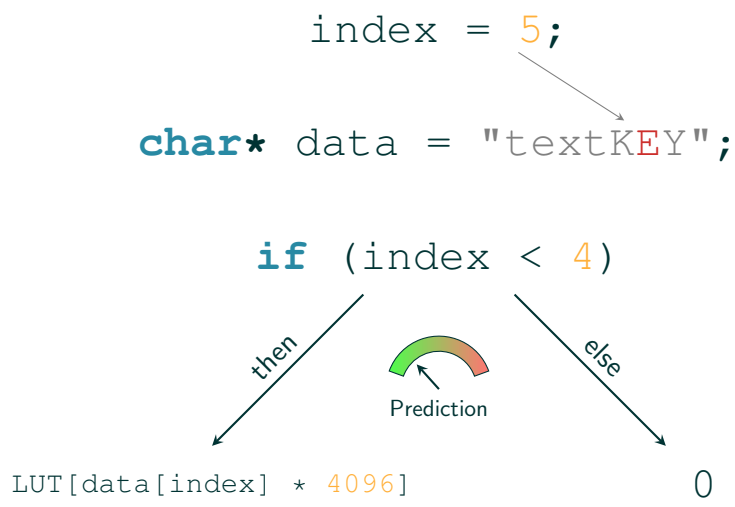


Spectre (Variant 1: Bounds-check bypass)

```
index = 5;

char* data = "textKEY";

if (index < 4)
```

`LUT[data[index] * 4096]` 0

Spectre (Variant 1: Bounds-check bypass)

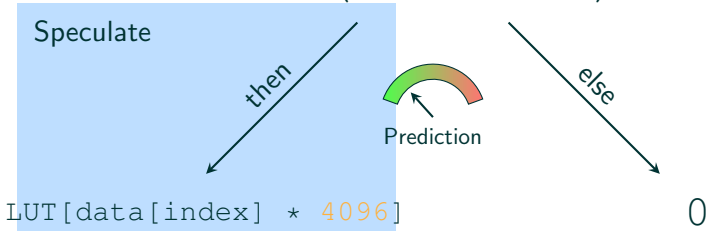


Spectre (Variant 1: Bounds-check bypass)

```
index = 5;
```

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

```
if (index < 4)
```

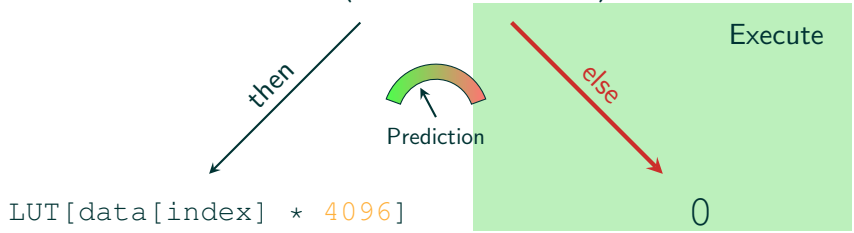


Spectre (Variant 1: Bounds-check bypass)

```
index = 5;
```

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

```
if (index < 4)
```



Spectre (Variant 1: Bounds-check bypass)

```
index = 6;

char* data = "textKEY";

if (index < 4)
```


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

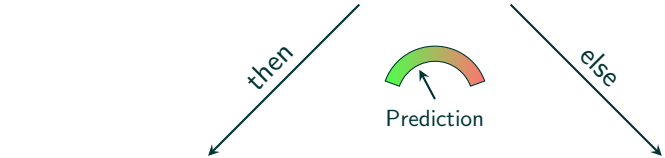
0

Spectre (Variant 1: Bounds-check bypass)

```
index = 6;
```

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

```
if (index < 4)
```

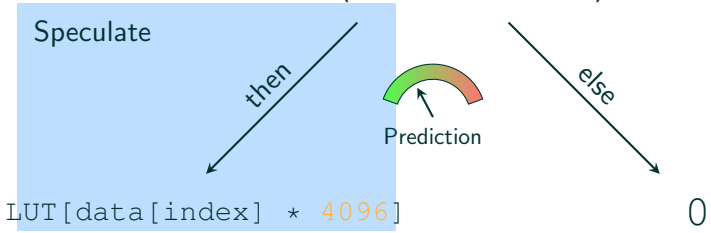


Spectre (Variant 1: Bounds-check bypass)

```
index = 6;
```

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

```
if (index < 4)
```

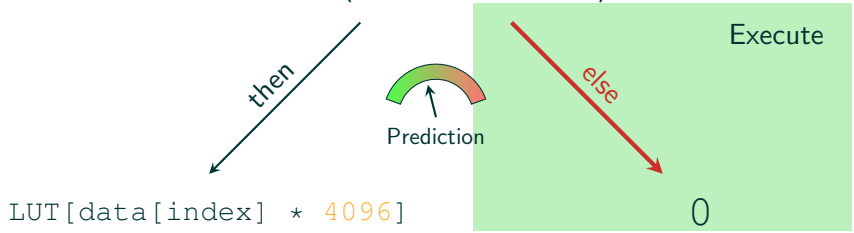


Spectre (Variant 1: Bounds-check bypass)

```
index = 6;
```

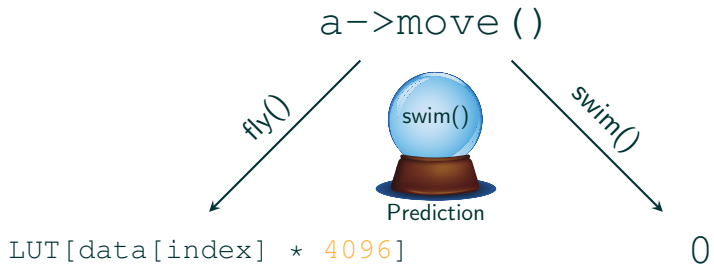
```
char* data = "textKEY";
```

```
if (index < 4)
```



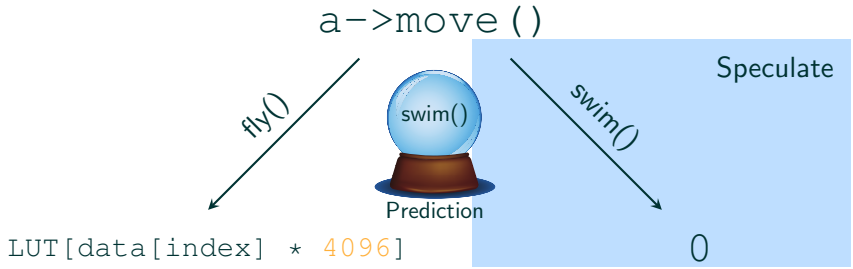
Spectre (Variant 2: Branch target injection)

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



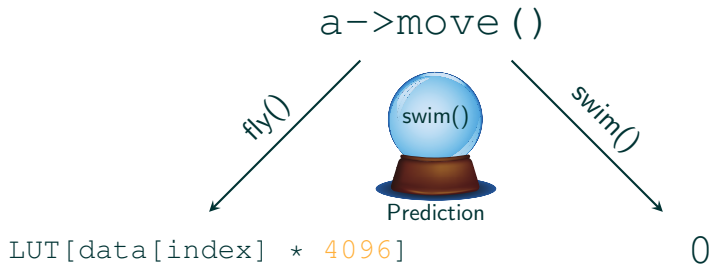
Spectre (Variant 2: Branch target injection)

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



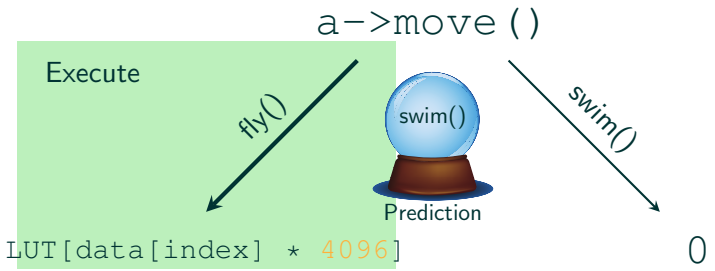
Spectre (Variant 2: Branch target injection)

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



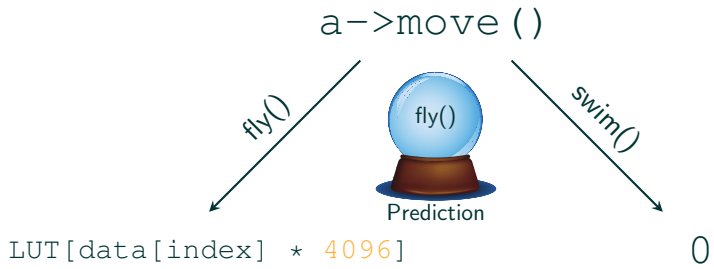
Spectre (Variant 2: Branch target injection)

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



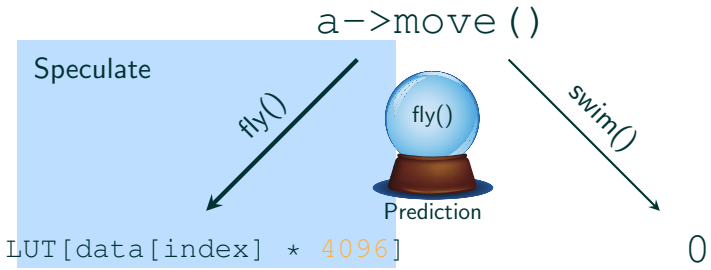
Spectre (Variant 2: Branch target injection)

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



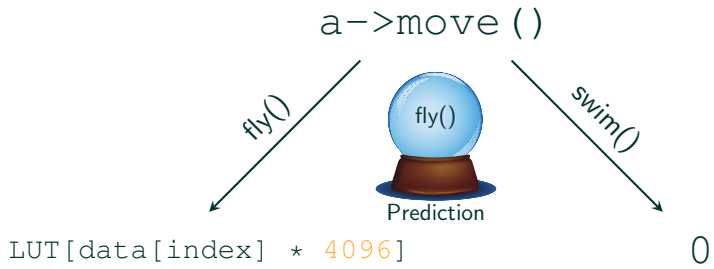
Spectre (Variant 2: Branch target injection)

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



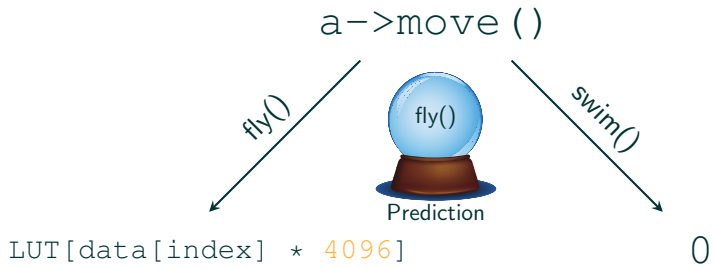
Spectre (Variant 2: Branch target injection)

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



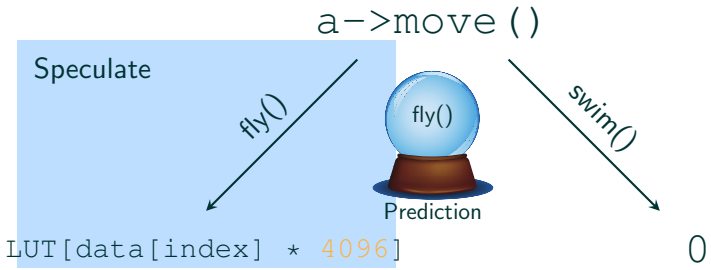
Spectre (Variant 2: Branch target injection)

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



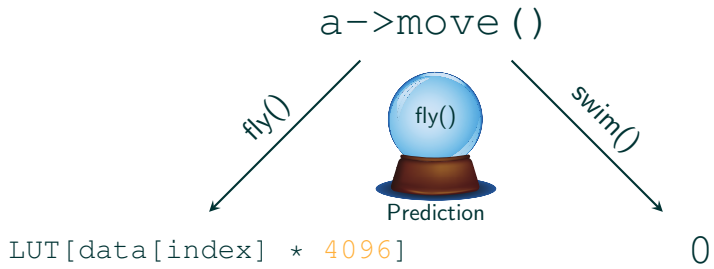
Spectre (Variant 2: Branch target injection)

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



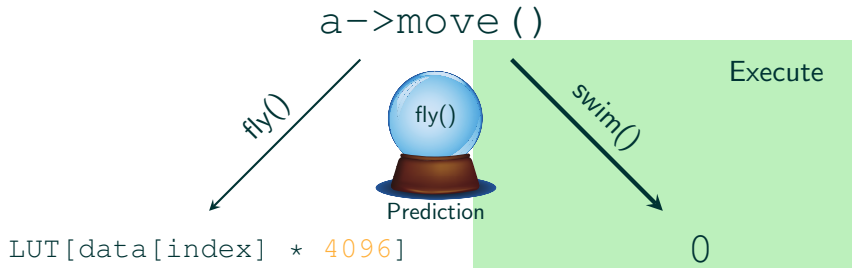
Spectre (Variant 2: Branch target injection)

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



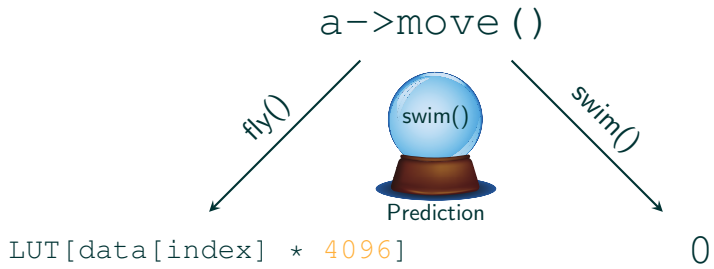
Spectre (Variant 2: Branch target injection)

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



Spectre (Variant 2: Branch target injection)

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



Spectre

- We can influence the CPU to mispredict the future
- CPU speculatively executes code that should never be executed
- Read own memory (e.g., sandbox escape)

Spectre

- We can influence the CPU to mispredict the future
- CPU speculatively executes code that should never be executed
- Read own memory (e.g., sandbox escape)
- “Convince” other programs to reveal their secrets

Spectre

- We can influence the CPU to mispredict the future
- CPU speculatively executes code that should never be executed
- Read own memory (e.g., sandbox escape)
- "Convince" other programs to reveal their secrets
- Again, a cache attack (Flush+Reload) is used to read the secret

Spectre

- We can influence the CPU to mispredict the future
- CPU speculatively executes code that should never be executed
- Read own memory (e.g., sandbox escape)
- "Convince" other programs to reveal their secrets
- Again, a cache attack (Flush+Reload) is used to read the secret
- Much harder to fix, KAISER does not help

Spectre

- We can influence the CPU to mispredict the future
- CPU speculatively executes code that should never be executed
- Read own memory (e.g., sandbox escape)
- "Convince" other programs to reveal their secrets
- Again, a cache attack (Flush+Reload) is used to read the secret
- Much harder to fix, KAISER does not help
- Ongoing effort to patch via microcode update and compiler extensions

Can we fix that?

Spectre Mitigations

- Trivial approach: disable speculative execution

Spectre Mitigations

- Trivial approach: disable speculative execution
- No wrong speculation if there is no speculation

Spectre Mitigations

- Trivial approach: disable speculative execution
- No wrong speculation if there is no speculation
- Problem: Massive performance hit!

Spectre Mitigations

- Trivial approach: disable speculative execution
- No wrong speculation if there is no speculation
- Problem: Massive performance hit!
- Also: How to disable it?

Spectre Mitigations

- Trivial approach: disable speculative execution
- No wrong speculation if there is no speculation
- Problem: Massive performance hit!
- Also: How to disable it?
- Speculative execution is deeply integrated into CPU

Spectre Variant 1 Mitigations

Spectre Variant 1 Mitigations

- Workaround: insert instructions stopping speculation

Spectre Variant 1 Mitigations

- Workaround: insert instructions stopping speculation

$\rightarrow$ insert after every bounds check

Spectre Variant 1 Mitigations

- Workaround: insert instructions stopping speculation

→ insert after every bounds check

- x86: `LFENCE`, ARM: `CSDB`

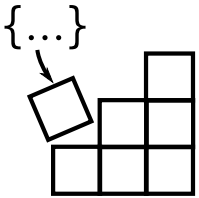
Spectre Variant 1 Mitigations

- Workaround: insert instructions stopping speculation

→ insert after every bounds check

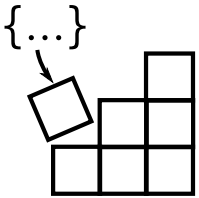
- x86: `LFENCE`, ARM: `CSDB`
- Available on all Intel CPUs, retrofitted to existing ARMv7 and ARMv8

Spectre Variant 1 Mitigations



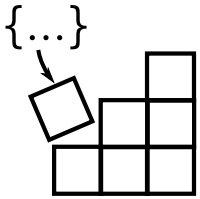
- Speculation barrier requires compiler supported

Spectre Variant 1 Mitigations



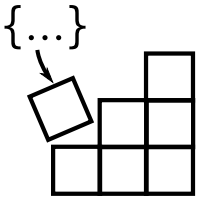
- Speculation barrier requires compiler supported
- Already implemented in GCC, LLVM, and MSVC

Spectre Variant 1 Mitigations



- Speculation barrier requires compiler supported
- Already implemented in GCC, LLVM, and MSVC
- Can be automated (MSVC) $\rightarrow$ not really reliable

Spectre Variant 1 Mitigations



- Speculation barrier requires compiler supported
- Already implemented in GCC, LLVM, and MSVC
- Can be automated (MSVC) $\rightarrow$ not really reliable
- Explicit use by programmer:
  `__builtin_load_no_speculate`

Spectre Variant 1 Mitigations

```
// Unprotected

int array[N];

int get_value(unsigned int n) {
  int tmp;

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

  return tmp;
}
```

Spectre Variant 1 Mitigations

```
// Unprotected

int array[N];

int get_value(unsigned int n) {
  int tmp;

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

  return tmp;
}
```

```
// Protected

int array[N];

int get_value(unsigned int n) {

  int *lower = array;
  int *ptr = array + n;
  int *upper = array + N;

  return
    __builtin_load_no_speculate
    (ptr, lower, upper, FAIL);

}
```

Spectre Variant 1 Mitigations

- Speculation barrier works if affected code constructs are known

Spectre Variant 1 Mitigations

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

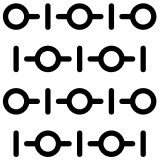
Spectre Variant 1 Mitigations

- Speculation barrier works if affected code constructs are known
- Programmer has to fully understand vulnerability
- Automatic detection is not reliable

Spectre Variant 1 Mitigations

- Speculation barrier works if affected code constructs are known
- Programmer has to fully understand vulnerability
- Automatic detection is not reliable
- Non-negligible performance overhead of barriers

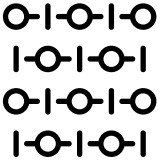
Spectre Variant 2 Mitigations (Microcode/MSRs)



Intel released microcode updates

- **Indirect Branch Restricted Speculation (IBRS)**:

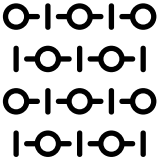
Spectre Variant 2 Mitigations (Microcode/MSRs)



Intel released microcode updates

- **Indirect Branch Restricted Speculation (IBRS)**:
  - Do not speculate based on anything before entering IBRS mode

Spectre Variant 2 Mitigations (Microcode/MSRs)

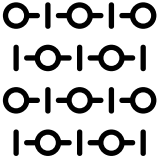


Intel released microcode updates

- **Indirect Branch Restricted Speculation (IBRS)**:
  - Do not speculate based on anything before entering IBRS mode

  $\rightarrow$ lesser privileged code cannot influence predictions

Spectre Variant 2 Mitigations (Microcode/MSRs)

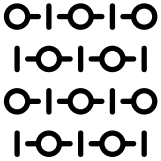


Intel released microcode updates

- **Indirect Branch Restricted Speculation (IBRS)**:
  - Do not speculate based on anything before entering IBRS mode

  $\rightarrow$ lesser privileged code cannot influence predictions
- **Indirect Branch Predictor Barrier (IBPB)**:

Spectre Variant 2 Mitigations (Microcode/MSRs)

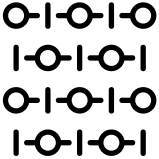


Intel released microcode updates

- **Indirect Branch Restricted Speculation (IBRS)**:
  - Do not speculate based on anything before entering IBRS mode

  $\rightarrow$ lesser privileged code cannot influence predictions
- **Indirect Branch Predictor Barrier (IBPB)**:
  - Flush branch-target buffer

Spectre Variant 2 Mitigations (Microcode/MSRs)

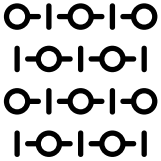


Intel released microcode updates

- **Indirect Branch Restricted Speculation (IBRS)**:
  - Do not speculate based on anything before entering IBRS mode

  $\rightarrow$ lesser privileged code cannot influence predictions
- **Indirect Branch Predictor Barrier (IBPB)**:
  - Flush branch-target buffer
- **Single Thread Indirect Branch Predictors (STIBP)**:

Spectre Variant 2 Mitigations (Microcode/MSRs)



Intel released microcode updates

- **Indirect Branch Restricted Speculation (IBRS)**:
  - Do not speculate based on anything before entering IBRS mode

  $\rightarrow$ lesser privileged code cannot influence predictions
- **Indirect Branch Predictor Barrier (IBPB)**:
  - Flush branch-target buffer
- **Single Thread Indirect Branch Predictors (STIBP)**:
  - Isolates branch prediction state between two hyperthreads

Spectre Variant 2 Mitigations (Software)

Retpoline (compiler extension)

Spectre Variant 2 Mitigations (Software)

Retpoline (compiler extension)

```
    push <call_target>
    call 1f
2:                      ; speculation will continue here
    lfence              ; speculation barrier
    jmp 2b              ; endless loop
1:
    lea 8(%rsp), %rsp   ; restore stack pointer
    ret                 ; the actual call to <call_target>
```

→ always predict to enter an endless loop

Spectre Variant 2 Mitigations (Software)

Retpoline (compiler extension)

```
    push <call_target>
    call 1f
2:                      ; speculation will continue here
    lfence              ; speculation barrier
    jmp 2b              ; endless loop
1:
    lea 8(%rsp), %rsp   ; restore stack pointer
    ret                 ; the actual call to <call_target>
```

→ always predict to enter an endless loop

- instead of the correct (or wrong) target function

Spectre Variant 2 Mitigations (Software)

Retpoline (compiler extension)

```
    push <call_target>
    call 1f
2:                      ; speculation will continue here
    lfence              ; speculation barrier
    jmp 2b              ; endless loop
1:
    lea 8(%rsp), %rsp   ; restore stack pointer
    ret                 ; the actual call to <call_target>
```

→ always predict to enter an endless loop

- instead of the correct (or wrong) target function → performance?

Spectre Variant 2 Mitigations (Software)

Retpoline (compiler extension)

```
    push <call_target>
    call 1f
2:                      ; speculation will continue here
    lfence              ; speculation barrier
    jmp 2b              ; endless loop
1:
    lea 8(%rsp), %rsp   ; restore stack pointer
    ret                 ; the actual call to <call_target>
```

→ always predict to enter an endless loop

- instead of the correct (or wrong) target function → performance?
- On Broadwell or newer:

Spectre Variant 2 Mitigations (Software)

Retpoline (compiler extension)

```
    push <call_target>
    call 1f
2:                      ; speculation will continue here
    lfence              ; speculation barrier
    jmp 2b              ; endless loop
1:
    lea 8(%rsp), %rsp   ; restore stack pointer
    ret                 ; the actual call to <call_target>
```

→ always predict to enter an endless loop

- instead of the correct (or wrong) target function → performance?
- On Broadwell or newer:
  - **ret** may fall-back to the BTB for prediction

Spectre Variant 2 Mitigations (Software)

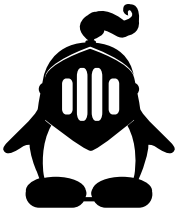
Retpoline (compiler extension)

```
  push <call_target>
  call 1f
2:                      ; speculation will continue here
  lfence                ; speculation barrier
  jmp 2b                ; endless loop
1:
  lea 8(%rsp), %rsp     ; restore stack pointer
  ret                   ; the actual call to <call_target>
```

→ always predict to enter an endless loop

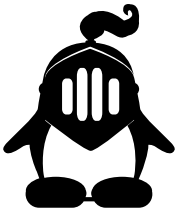
- instead of the correct (or wrong) target function → performance?
- On Broadwell or newer:
  - **ret** may fall-back to the BTB for prediction → microcode patches to prevent that

Spectre Variant 2 Mitigations (Software)



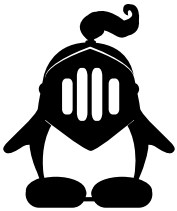
- ARM provides hardened Linux kernel

Spectre Variant 2 Mitigations (Software)



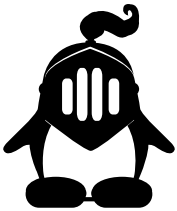
- ARM provides hardened Linux kernel
- Clears branch-predictor state on context switch

Spectre Variant 2 Mitigations (Software)



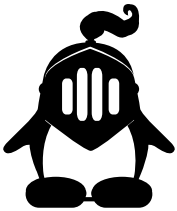
- ARM provides hardened Linux kernel
- Clears branch-predictor state on context switch
- Either via instruction (`BPIALL`)...

Spectre Variant 2 Mitigations (Software)



- ARM provides hardened Linux kernel
- Clears branch-predictor state on context switch
- Either via instruction (`BPIALL`)...
- ...or workaround (disable/enable MMU)

Spectre Variant 2 Mitigations (Software)



- ARM provides hardened Linux kernel
- Clears branch-predictor state on context switch
- Either via instruction (`BPIALL`)...
- ...or workaround (disable/enable MMU)
- Non-negligible performance overhead ($\approx$ 200-300 ns)

What does not work

- Prevent access to high-resolution timer

What does not work

- Prevent access to high-resolution timer

→ Own timer using timing thread

What does not work

- Prevent access to high-resolution timer

→ Own timer using timing thread

- Flush instruction only privileged

What does not work

- Prevent access to high-resolution timer

→ Own timer using timing thread

- Flush instruction only privileged

→ Cache eviction through memory accesses

What does not work

- Prevent access to high-resolution timer

→ Own timer using timing thread

- Flush instruction only privileged

→ Cache eviction through memory accesses

- Just move secrets into secure world

What does not work

- Prevent access to high-resolution timer

→ Own timer using timing thread

- Flush instruction only privileged

→ Cache eviction through memory accesses

- Just move secrets into secure world

→ Spectre works on secure enclaves

What to do now?

Learn from it

We have ignored software side-channels for many many years:

Learn from it

We have ignored software side-channels for many many years:

- attacks on crypto

Learn from it

We have ignored software side-channels for many many years:

- attacks on crypto $\rightarrow$ "software should be fixed"

Learn from it

We have ignored software side-channels for many many years:

- attacks on crypto $\rightarrow$ "software should be fixed"
- attacks on ASLR

Learn from it

We have ignored software side-channels for many many years:

- attacks on crypto $\rightarrow$ "software should be fixed"
- attacks on ASLR $\rightarrow$ "ASLR is broken anyway"

Learn from it

We have ignored software side-channels for many many years:

- attacks on crypto $\rightarrow$ "software should be fixed"
- attacks on ASLR $\rightarrow$ "ASLR is broken anyway"
- attacks on SGX and TrustZone

Learn from it

We have ignored software side-channels for many many years:

- attacks on crypto → "software should be fixed"
- attacks on ASLR → "ASLR is broken anyway"
- attacks on SGX and TrustZone → "not part of the threat model"

Learn from it

We have ignored software side-channels for many many years:

- attacks on crypto → "software should be fixed"
- attacks on ASLR → "ASLR is broken anyway"
- attacks on SGX and TrustZone → "not part of the threat model"

→ for years we solely optimized for performance

When you read the manuals...

After learning about a side channel you realize:

When you read the manuals...

After learning about a side channel you realize:

- the side channels were documented in the Intel manual

When you read the manuals...

After learning about a side channel you realize:

- the side channels were documented in the Intel manual
- only now we understand the implications

A unique chance

A unique chance to

- rethink processor design
- grow up, like other fields (car industry, construction industry)
- find good trade-offs between security and performance

Conclusion

Conclusion

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

Breaking through walls

How performance optimizations shatter security boundaries

Moritz Lipp

Mar 05, 2018—QCon London 2018

IAIK, Graz University of Technology