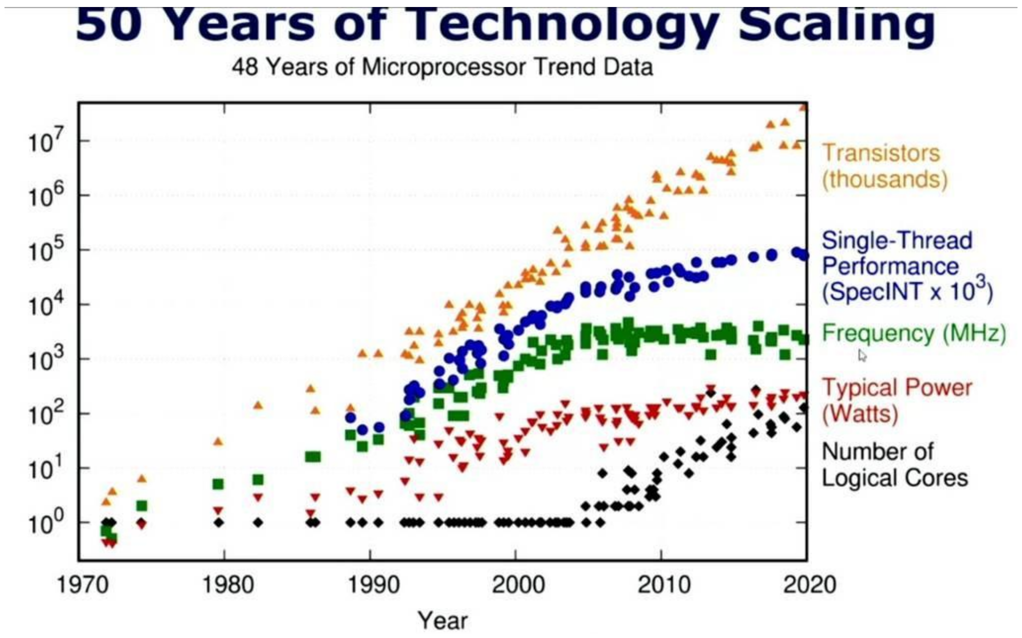
The curse of shortcuts to be MOoRE happy!

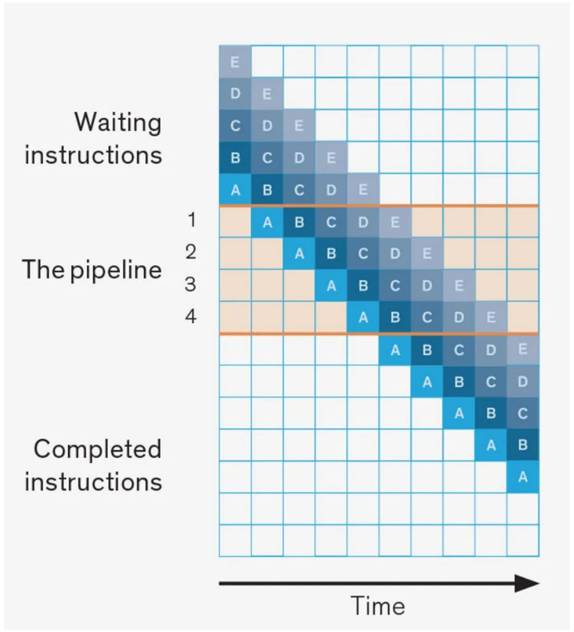
Holistic Software Security

Aravind Machiry

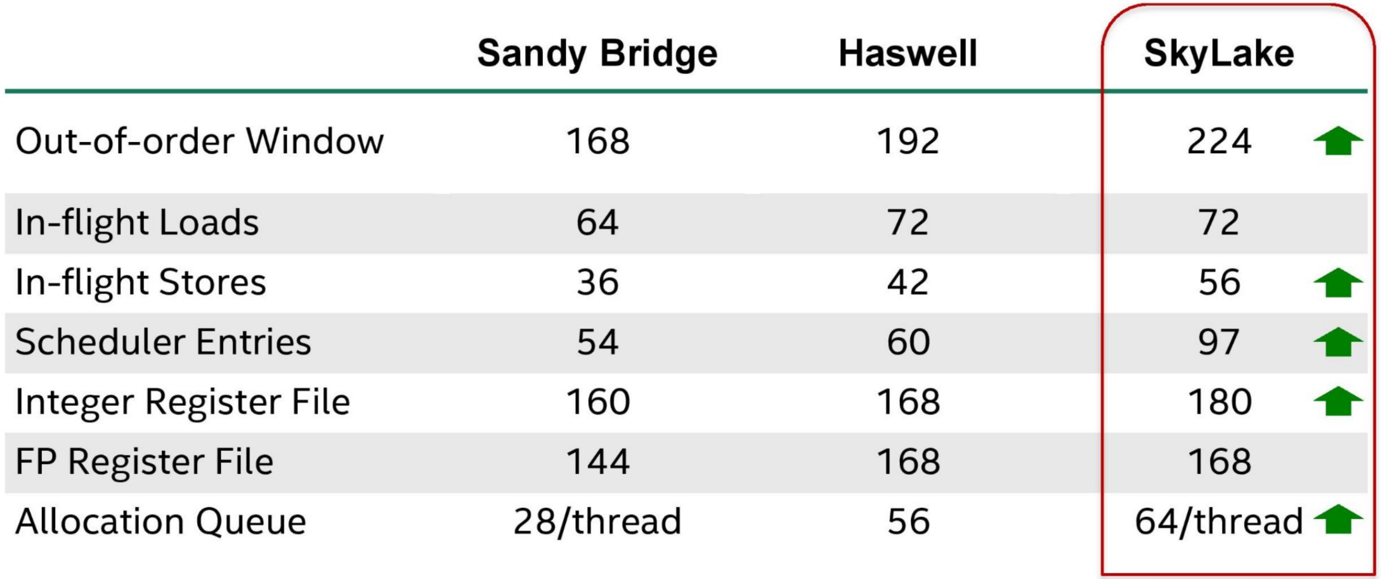
Hard to pack more transistors!



Pipeline Execution

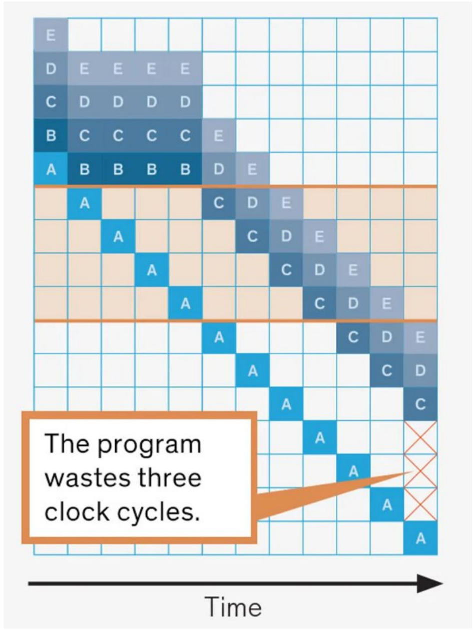


Pipeline length keeps on increasing!

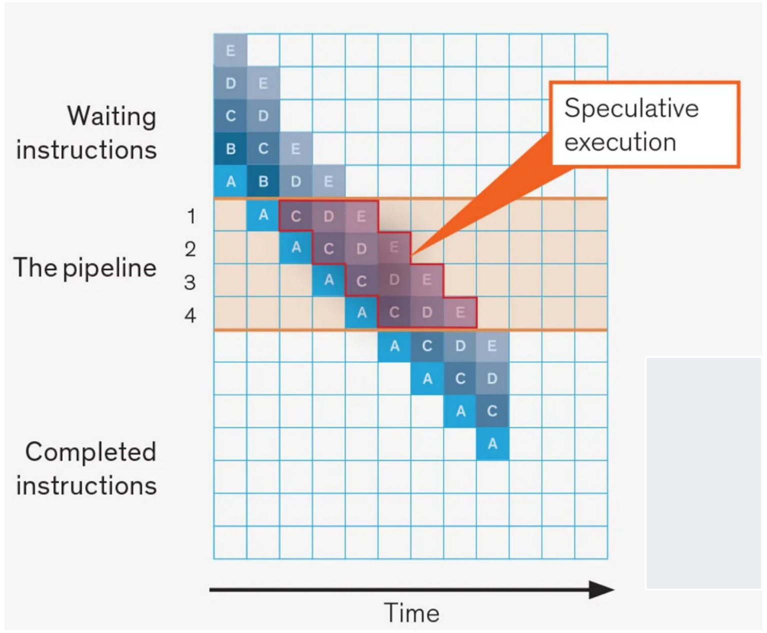


	Sandy Bridge	Haswell	SkyLake
Out-of-order Window	168	192	224 ⬆
In-flight Loads	64	72	72
In-flight Stores	36	42	56 ⬆
Scheduler Entries	54	60	97 ⬆
Integer Register File	160	168	180 ⬆
FP Register File	144	168	168
Allocation Queue	28/thread	56	64/thread ⬆

The need for speculation!



Speculative Execution



Types of Speculation

- Out of order execution

- Branch Prediction based Speculative Execution.

Out of order execution

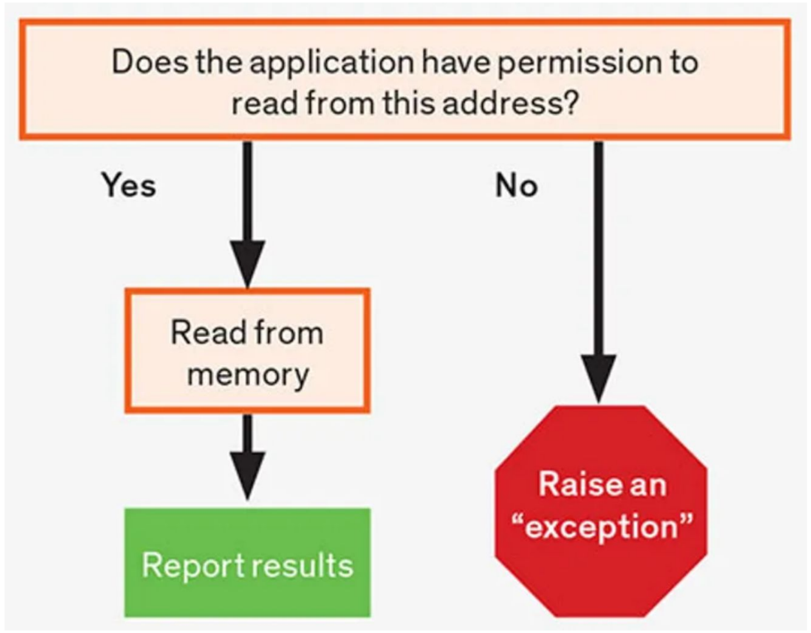
```
xor rax, rax
retry:
 mov al, byte [rcx]
 shl rax, 0xc
jz retry
mov rbx, qword [rbx + rax]
```

Out of order execution

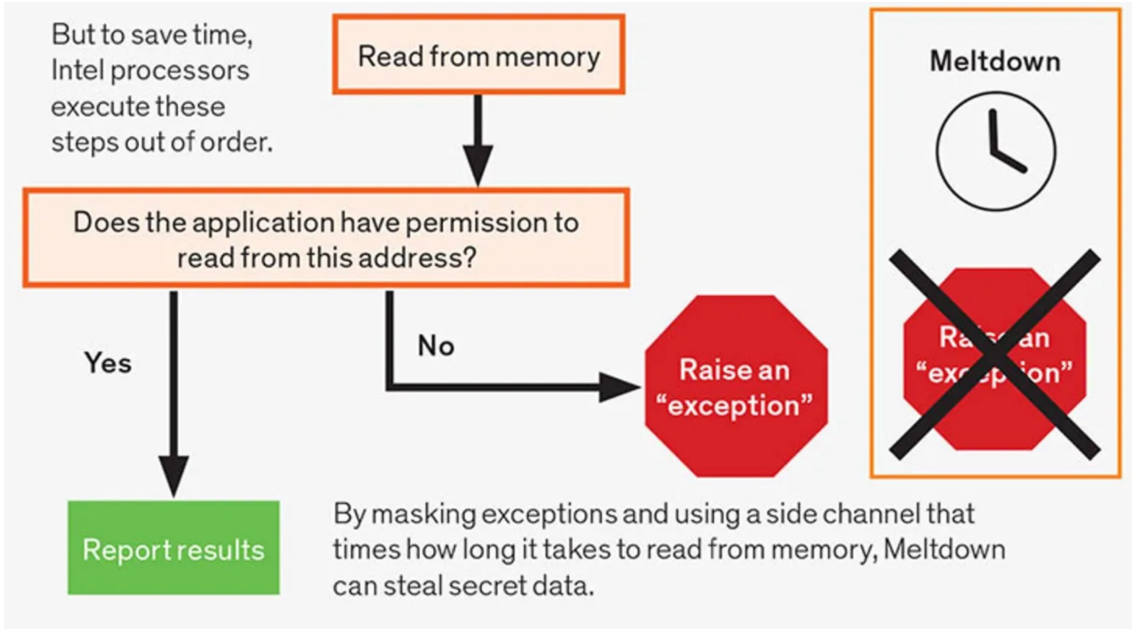
```
xor rax, rax

retry:

 mov al, byte [rcx]

 shl rax, 0xc

jz retry

mov rbx, qword [rbx + rax]
```

What happens if **rcx contains kernel address**?

Expectation


Does the application have permission to read from this address?
Yes
No
Read from memory
Report results
Raise an "exception"

Reality: Meltdown


But to save time, Intel processors execute these steps out of order.
Read from memory
Does the application have permission to read from this address?
Yes
No
Raise an "exception"
Meltdown
Raise an "exception"
Report results
By masking exceptions and using a side channel that times how long it takes to read from memory, Meltdown can steal secret data.

Meltdown

```
xor rax, rax

retry:

 mov al, byte [rcx]

 shl rax, 0xc

jz retry

mov rbx, qword [rbx + rax]
```

rcx = kernel address,

rbx = probe array

Depending on the value of `al`, corresponding entry in `rbx` will be loaded into cache.

Spectra: Abusing Branch Predictor

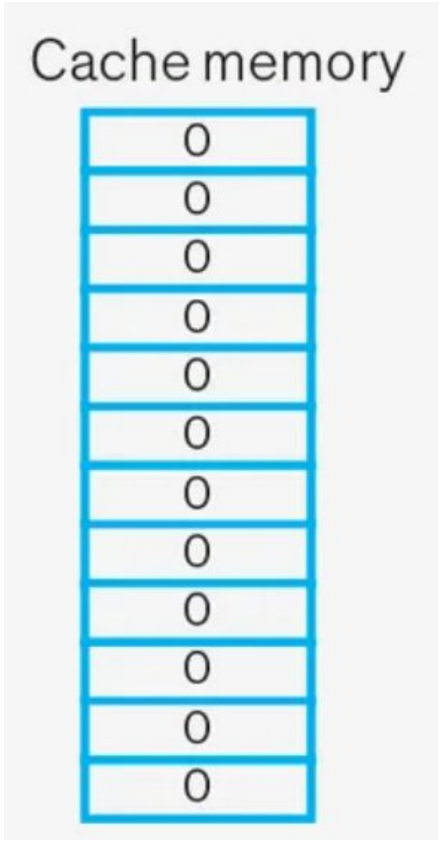
```
if (x < 256) {

  secret = array1[x];

  y = array2[secret];

}
```

Priming Branch Predictor

Run the code several times with x less than 256 to prime the branch predictor.

Setting up Cache.

Flush the cache.

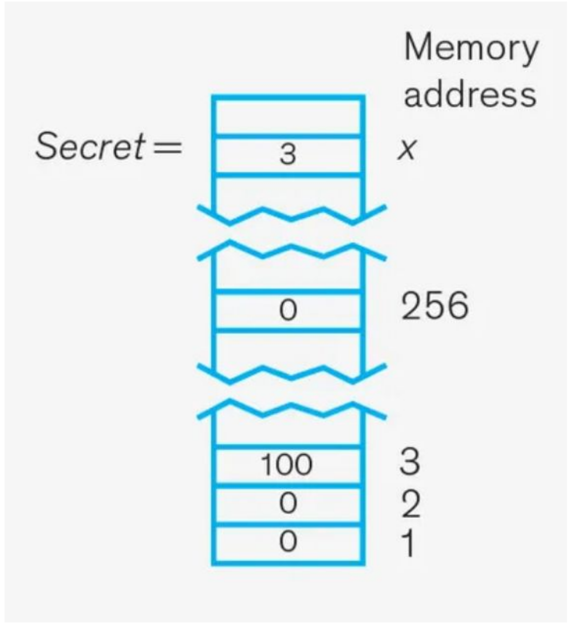


Triggering Branch Predictor

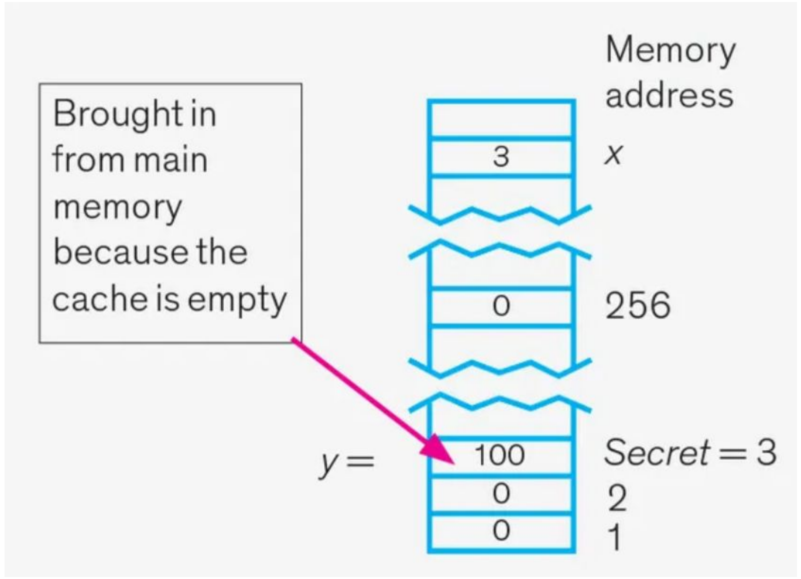
Run the code with **x > 256**

Branch predictor:

Take the branch?
Yes
No

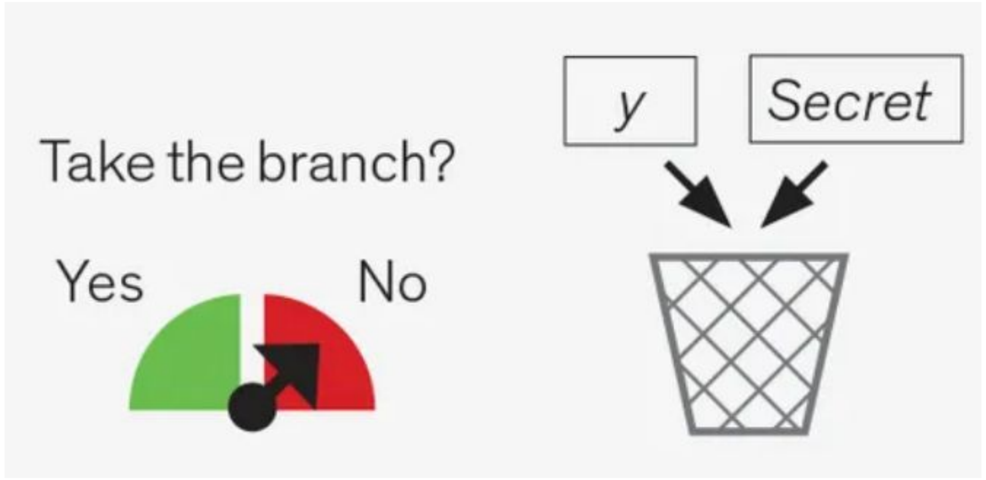


Loading into Cache.

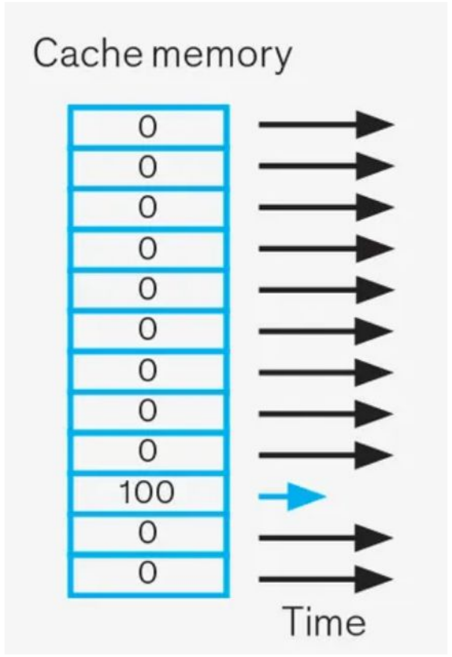


Mispredict realization

Realizing that the branch was mispredicted.

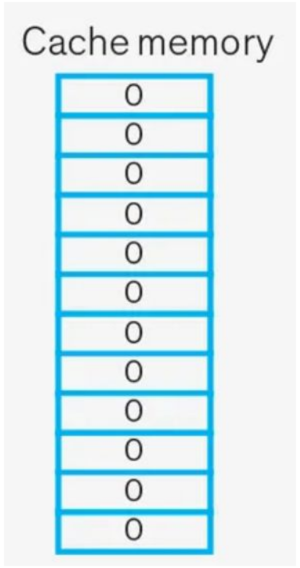


Cache is not flushed!!

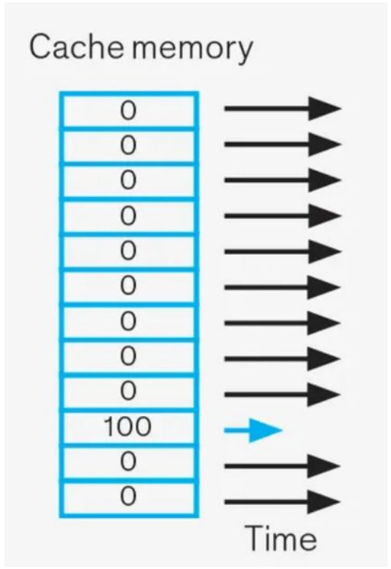


Flush and Reload

Before



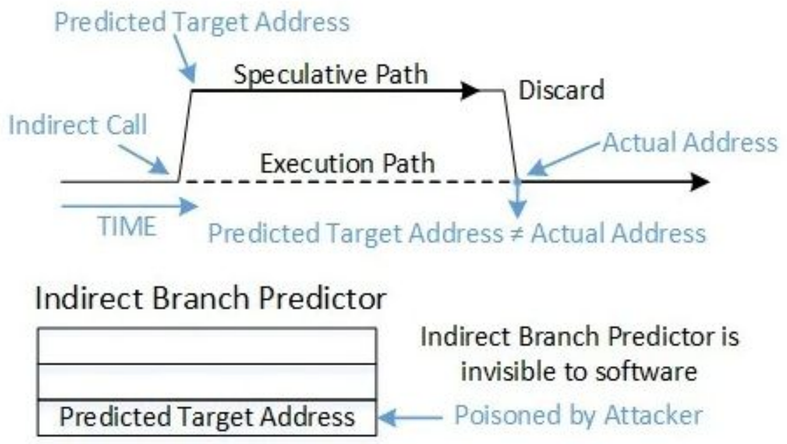
After



What is the secret value?

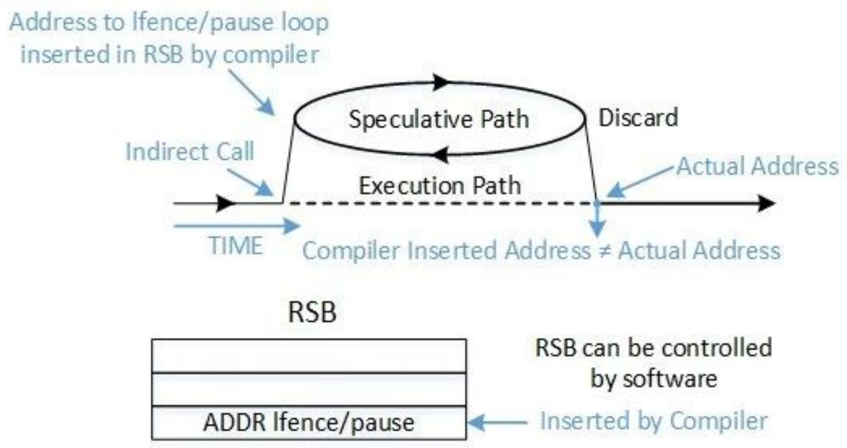
Mitigating Spectra: retpoline

Speculative Execution



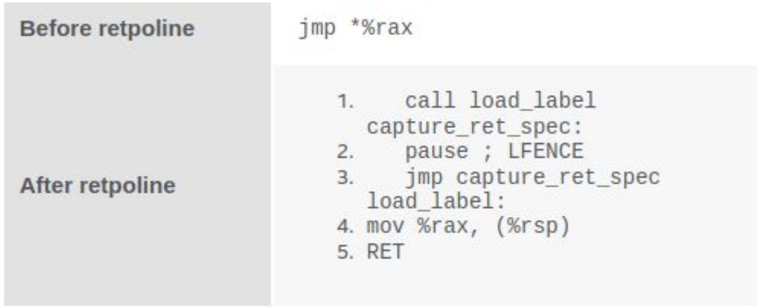
Mitigating Spectra: retpoline

Speculative Execution with retpoline



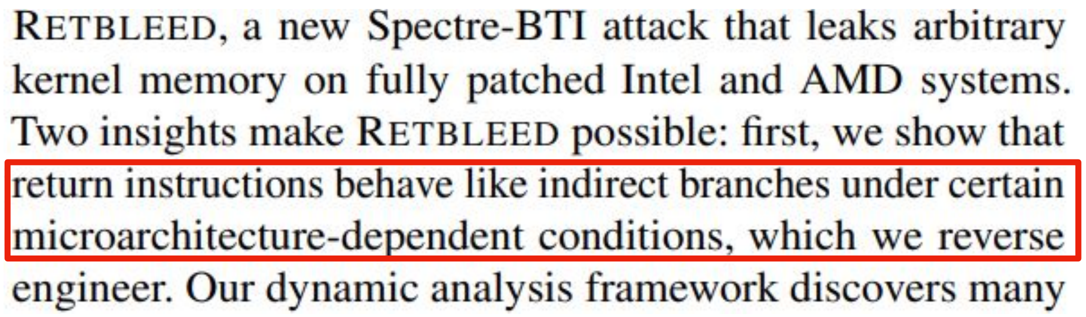
Address to lfence/pause loop inserted in RSB by compiler
Speculative Path
Discard
Indirect Call
Actual Address
Execution Path
TIME
Compiler Inserted Address ≠ Actual Address
RSB
ADDR lfence/pause
RSB can be controlled by software
Inserted by Compiler

Mitigating Spectra: retpoline



Before retpoline	`jmp *%rax`
After retpoline	`call load_label` `capture_ret_spec:` `pause ; LFENCE` `jmp capture_ret_spec` `load_label:` `mov %rax, (%rsp)` `RET`

Problem with Retpoline


RETBLEED, a new Spectre-BTI attack that leaks arbitrary kernel memory on fully patched Intel and AMD systems. Two insights make RETBLEED possible: first, we show that return instructions behave like indirect branches under certain microarchitecture-dependent conditions, which we reverse engineer. Our dynamic analysis framework discovers many

Preventing Spectra!

Disable Branch Prediction!

Preventing Spectra!

Disable Branch Prediction!

Keep track of “all” the side effects caused by an execution and clean them up on a mispredicted branch!!

Preventing Spectra!

Disable Branch Prediction!

Keep track of “all” the side effects caused by an execution and clean them up on a mispredicted branch!!

Prevent speculation on accessing secret data:

- `mlfence`

Many other CPU bugs found since!

- `cat /proc/cpuinfo | grep -i bugs`