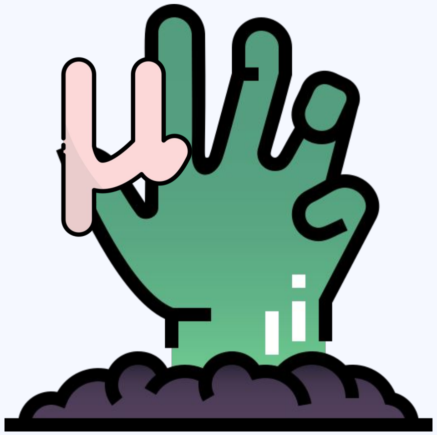
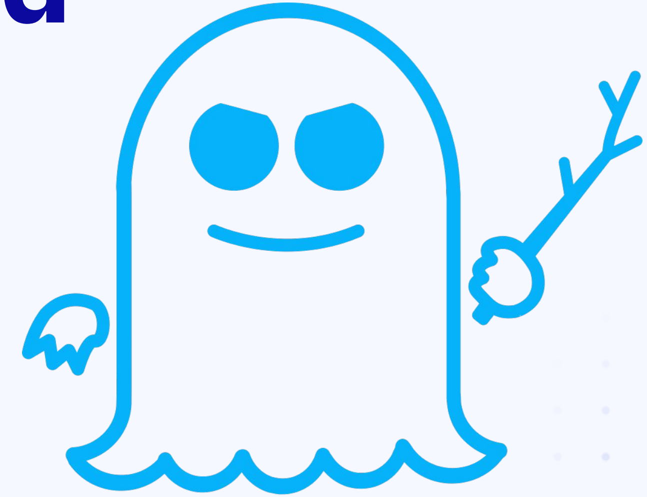
HORRORS OF ILP : Spectre and dead µops





Reading

Spectre

- Paper
- Webpage

I see dead μops

- Paper
- Article

Bonus: Meltdown

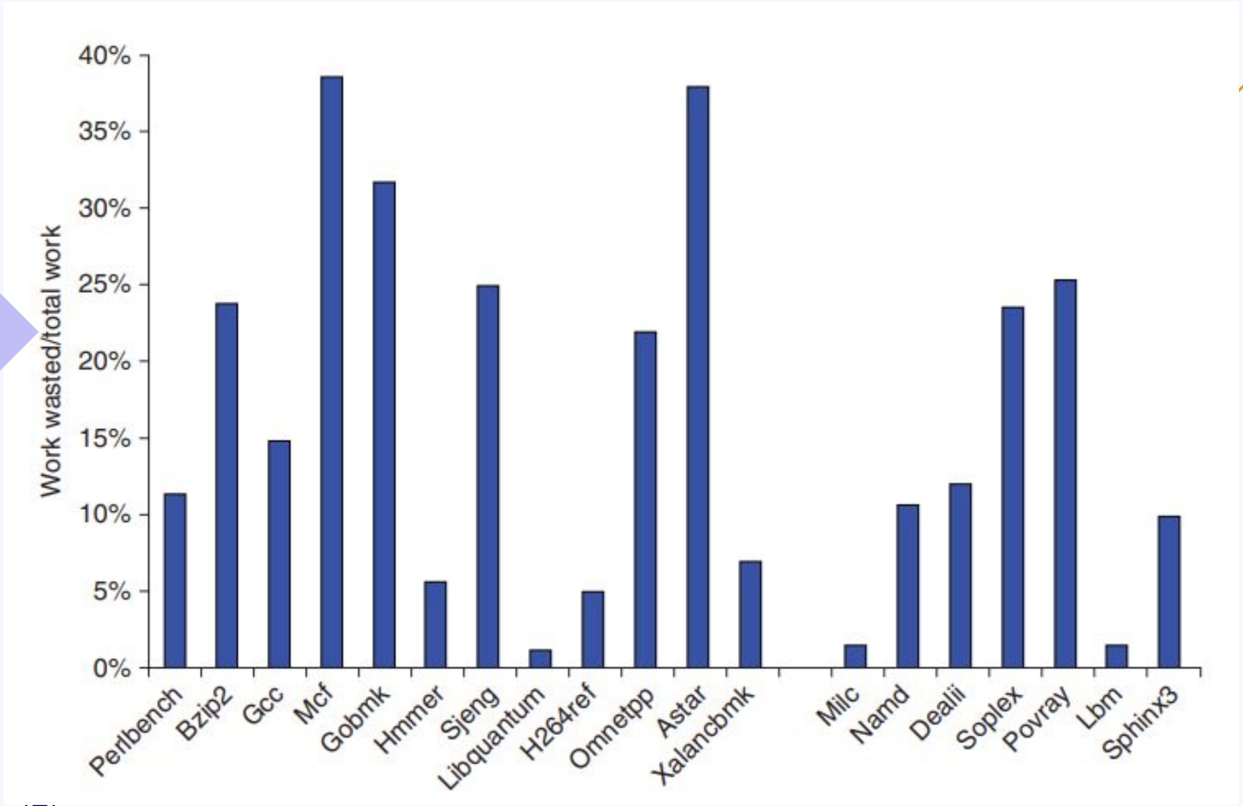
- Paper
- Usenix page (slides, presentation)

? ? ?

What performance metrics (beyond CPI) might become important in a speculative CPU?

H&P fig. 3.42

% of executed µops that were not committed



? ? ?

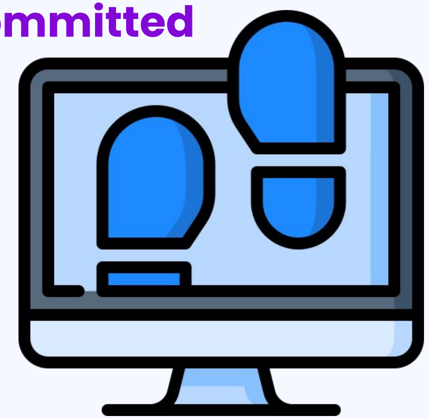
What unintended effects could OOO/speculative execution have on the memory system?

OOO/speculative attacks

Two key ideas:

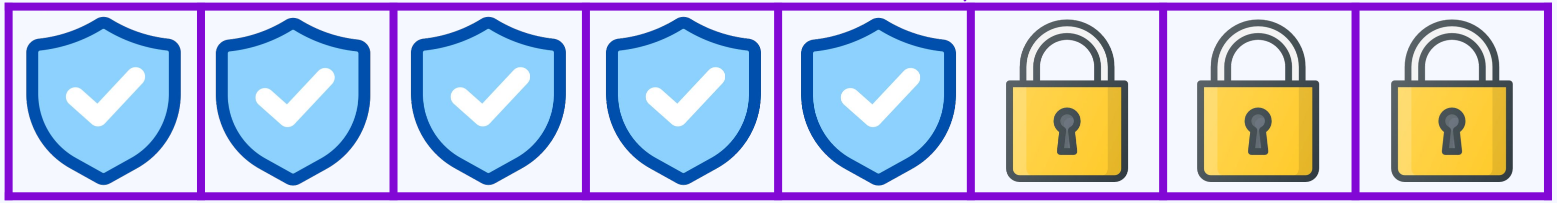
- Instructions can be executed but not committed (**transient** instructions)
- Transient instructions leave footprints in the microarchitecture

By forcing an unsafe instruction to execute speculatively, we can observe what *would* have happened if it had been committed



Spectre

anything past this array bound is privileged



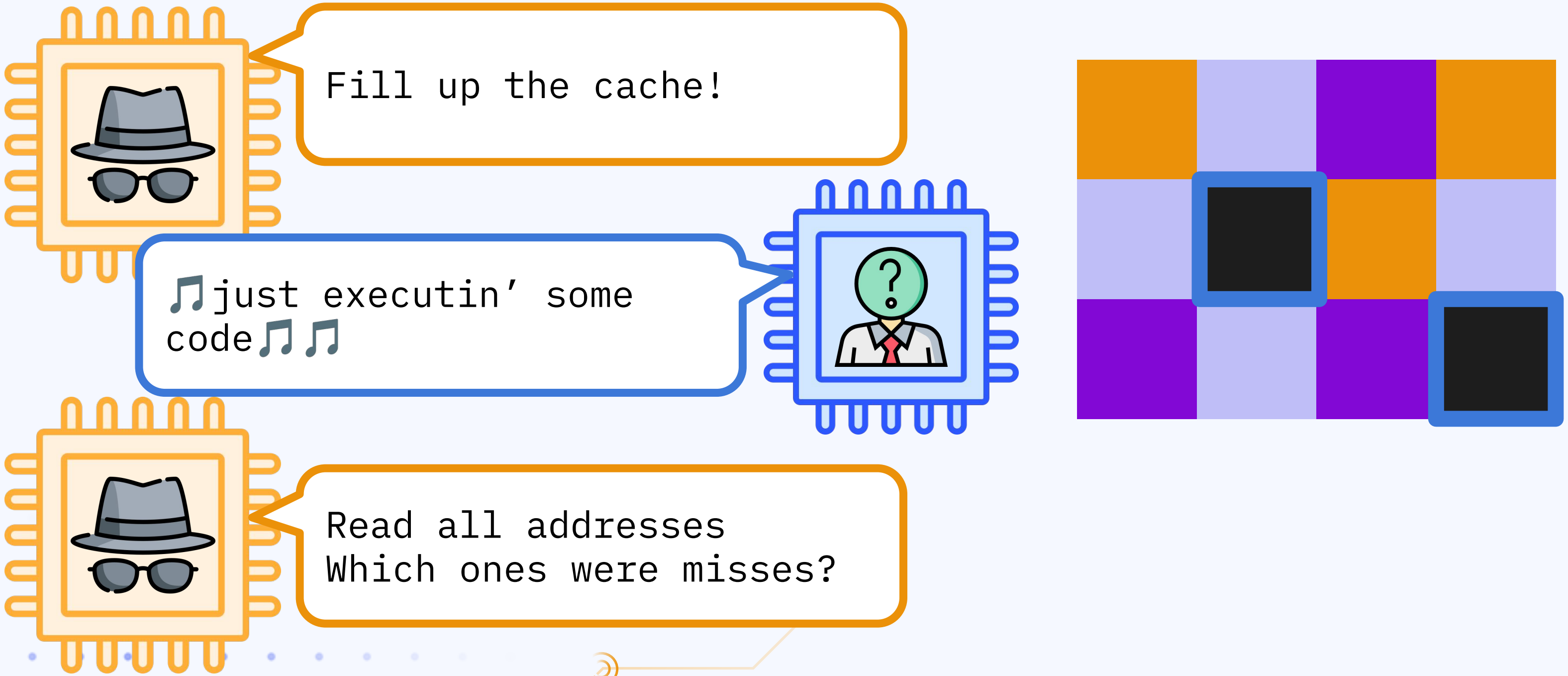
```
libFunc(x) {
    if (x < safeBound) {
        key = arr1[x];
        return arr2[key];
    }
}
```

get the CPU to run this line of code even when x is past the safeBound

Can we learn these keys just by calling libFunc?

Reminder: cache side channel

Prime and probe

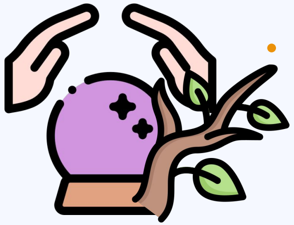


Spectre setup

```
libFunc(x) {
    if (x < safeBound) {
        key = arr1[x];
        return arr2[key];
    }
}
```

1. Train the branch predictor

```
for (int i = 0 ; i < safeBound; i++) {
    libFunc(i);
}
```

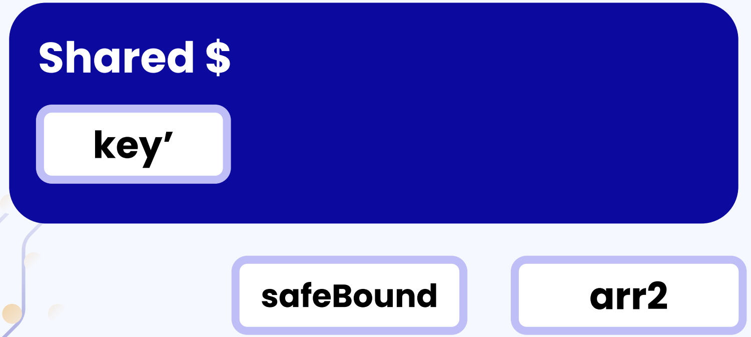


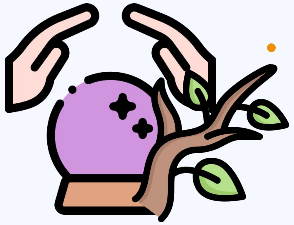
The branch will be taken

Spectre setup

```
libFunc(x) {
    if (x < safeBound) {
        key = arr1[x];
        return arr2[key];
    }
}
```

1. Train the branch predictor
2. Prime the cache
   - evict by walking addresses
   - call library functions that access *secret* key' but not safeBound, arr2





The branch will be taken

Spectre attack

```
libFunc(x) {
    if (x < safeBound) {
        key = arr1[x];
        return arr2[key];
    }
}
```

libFunc(x’)

Predict taken
Takes a while to resolve due to uncached safeBound

executes speculatively; fast because key’ is in cache

executes speculatively; brings arr2[key’] into cache

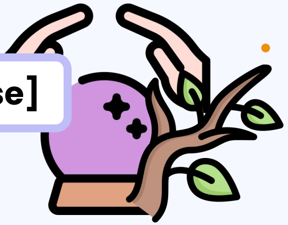
Shared $

key’

arr2[key’]

safeBound

arr2[anything else]



The branch will be taken

Spectre measure

```
libFunc(x) {
    if (x < safeBound) {
        key = arr1[x];
        return arr2[key];
    }
}
```

probe addresses that were used to prime
time results
one access slower than the rest: **conclude arr2[key'] maps to that block**
narrows down what key' value could be!

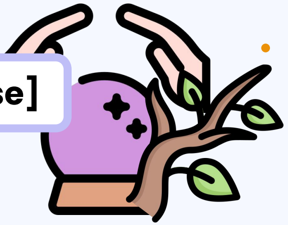
Shared $

key'

arr2[key']

safeBound

arr2[anything else]



oh #@%&

? ? ?

How do we mitigate spectre?

Avoid branches entirely?

```
libFuncSafer(x) {
    x = x & (x < safeBound ? ~0 : 0); // compiled into something
                                       // like SLT and AND
    key = arr1[x];
    return arr2[key];
}
```

Barrier instructions

Instructions that provide serialization/stop speculation

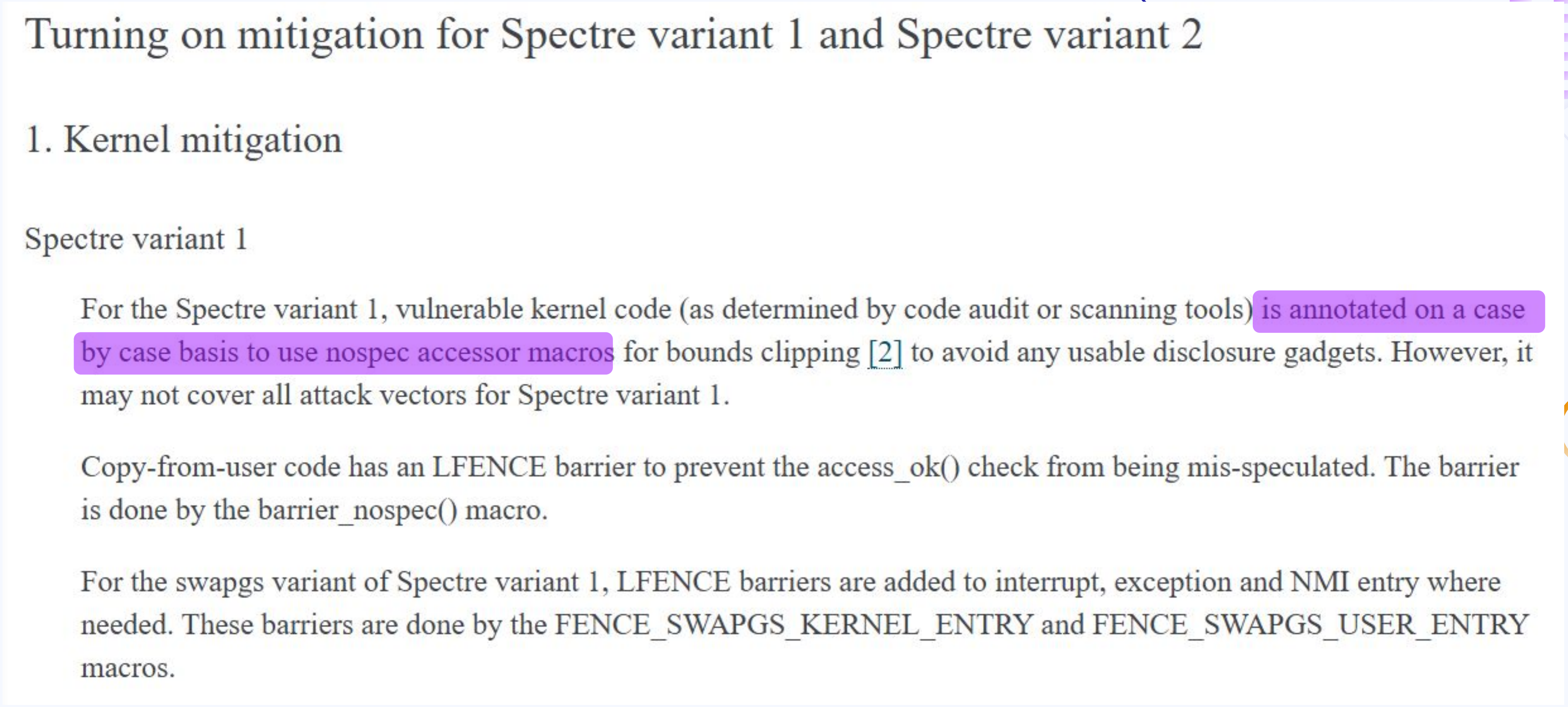
Different mechanisms and restrictions

- Some *weaker* (only serialize instruction memory operations)
- Some *stronger* (require flushing of µarch structures such as caches)

x86: LFENCE, arm: ISB/DMB/DSB; RISCV: FENCE.I

```
libFunc(x) {
    if (x < arraySize) {
        asm volatile("lfence");
        key = arr1[x];
        return arr2[key];
    }
}
```

Tradeoff between security and performance: we lose speed of speculation on in-bounds accesses
Active research area on how to make this performant!



Turning on mitigation for Spectre variant 1 and Spectre variant 2

1. Kernel mitigation

Spectre variant 1

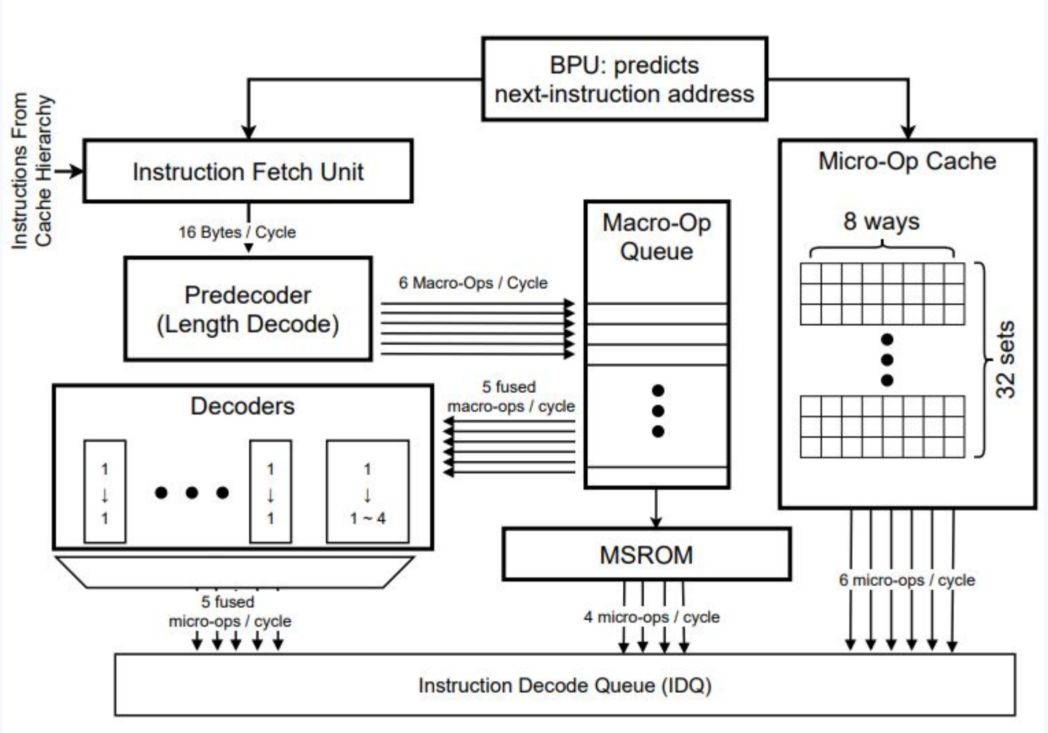
For the Spectre variant 1, vulnerable kernel code (as determined by code audit or scanning tools) is annotated on a case by case basis to use nospec accessor macros for bounds clipping [2] to avoid any usable disclosure gadgets. However, it may not cover all attack vectors for Spectre variant 1.

Copy-from-user code has an LFENCE barrier to prevent the access_ok() check from being mis-speculated. The barrier is done by the barrier_nospec() macro.

For the swapgs variant of Spectre variant 1, LFENCE barriers are added to interrupt, exception and NMI entry where needed. These barriers are done by the FENCE_SWAPGS_KERNEL_ENTRY and FENCE_SWAPGS_USER_ENTRY macros.

source

Micro-op cache



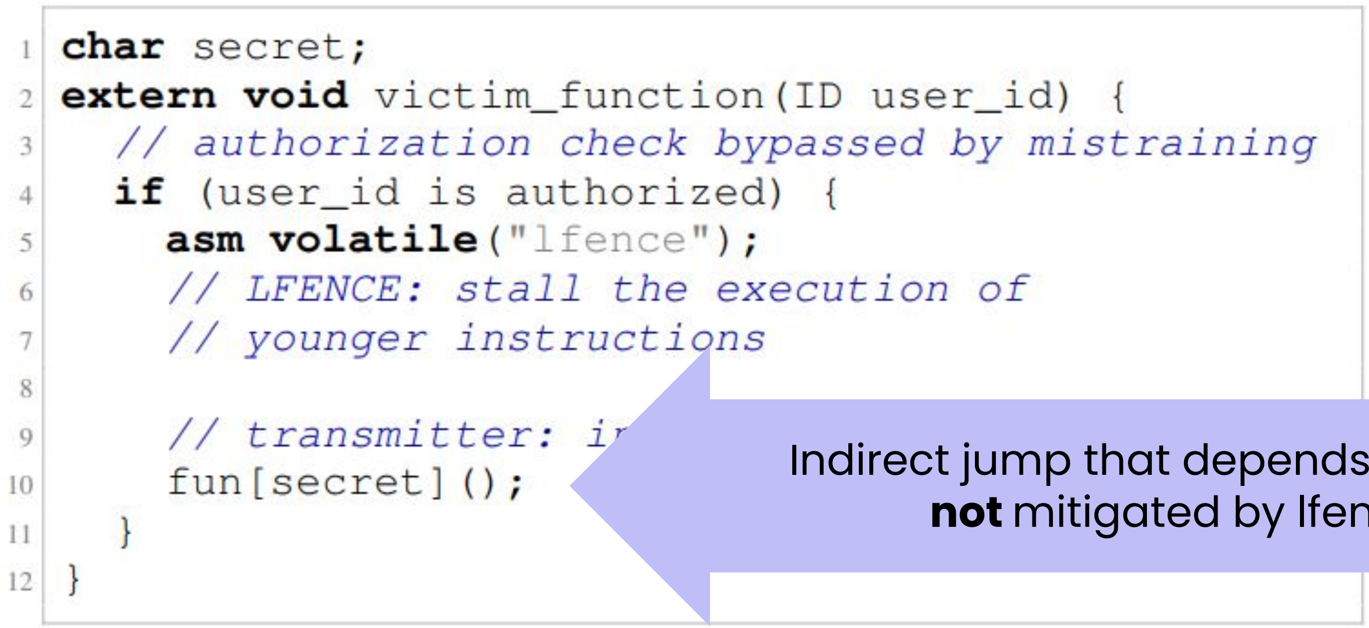
BPU: predicts next-instruction address
Instructions From Cache Hierarchy
Instruction Fetch Unit
16 Bytes / Cycle
Predecoder (Length Decode)
6 Macro-Ops / Cycle
Macro-Op Queue
5 fused macro-ops / cycle
Decoders
1 ↓ 1
1 ↓ 1
1 ↓ 1 ~ 4
5 fused micro-ops / cycle
MSROM
4 micro-ops / cycle
Micro-Op Cache
8 ways
32 sets
6 micro-ops / cycle
Instruction Decode Queue (IDQ)

Fig. 1: x86 micro-op cache and decode pipeline

Ren, Xida, et al. "I see dead µops: Leaking secrets via intel/amd micro-op caches." 2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA). IEEE, 2021.

µop $ is populated on fetch (prior to instructions executing)

I see dead µops attack



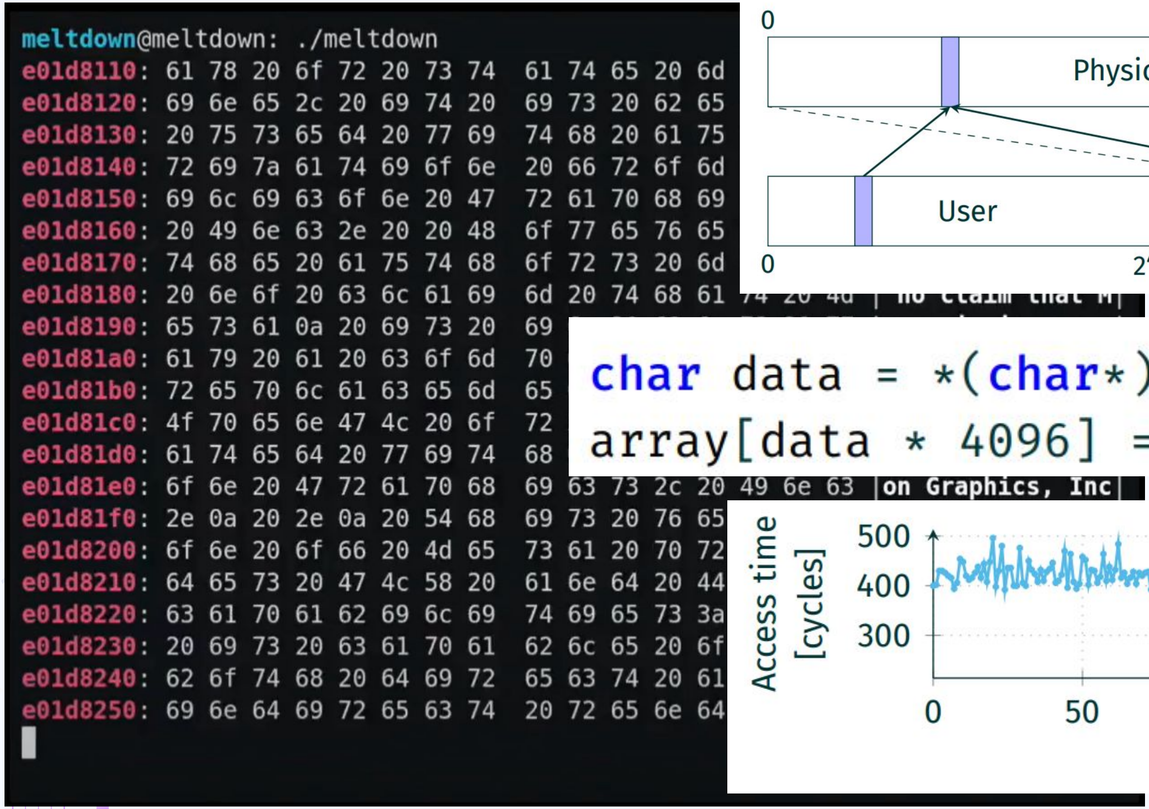
```
char secret;
extern void victim_function(ID user_id) {
  // authorization check bypassed by mistraining
  if (user_id is authorized) {
    asm volatile("lfence");
    // LFENCE: stall the execution of
    // younger instructions

    // transmitter: i
    fun[secret]();
  }
}
```

Listing 5: Victim Method for our Variant-2 Attack

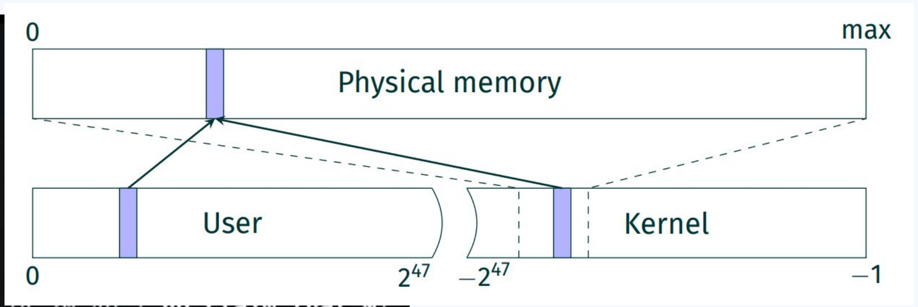
Indirect jump that depends on secret
not mitigated by lfence!

source

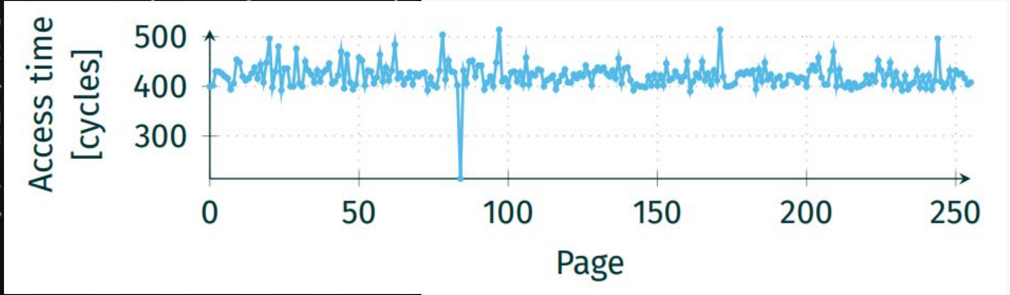
Bonus: meltdown



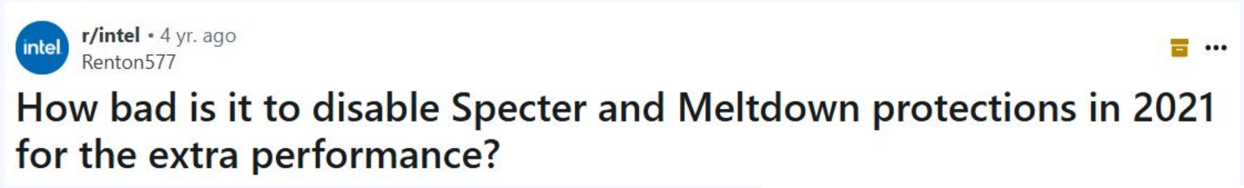
```
meltdown@meltdown: ./meltdown
e01d8110: 61 78 20 6f 72 20 73 74  61 74 65 20 6d
e01d8120: 69 6e 65 2c 20 69 74 20  69 73 20 62 65
e01d8130: 20 75 73 65 64 20 77 69  74 68 20 61 75
e01d8140: 72 69 7a 61 74 69 6f 6e  20 66 72 6f 6d
e01d8150: 69 6c 69 63 6f 6e 20 47  72 61 70 68 69
e01d8160: 20 49 6e 63 2e 20 20 48  6f 77 65 76 65
e01d8170: 74 68 65 20 61 75 74 68  6f 72 73 20 6d
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
e01d81a0: 61 79 20 61 20 63 6f 6d  70
e01d81b0: 72 65 70 6c 61 63 65 6d  65
e01d81c0: 4f 70 65 6e 47 4c 20 6f  72
e01d81d0: 61 74 65 64 20 77 69 74  68
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
e01d8200: 6f 6e 20 6f 66 20 4d 65  73 61 20 70 72
e01d8210: 64 65 73 20 47 4c 58 20  61 6e 64 20 44
e01d8220: 63 61 70 61 62 69 6c 69  74 69 65 73 3a
e01d8230: 20 69 73 20 63 61 70 61  62 6c 65 20 6f
e01d8240: 62 6f 74 68 20 64 69 72  65 63 74 20 61
e01d8250: 69 6e 64 69 72 65 63 74  20 72 65 6e 64
```



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

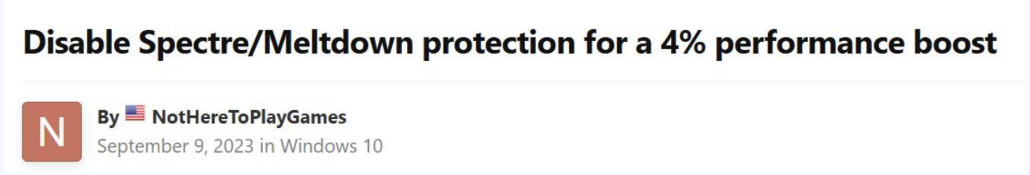


Bonus: users



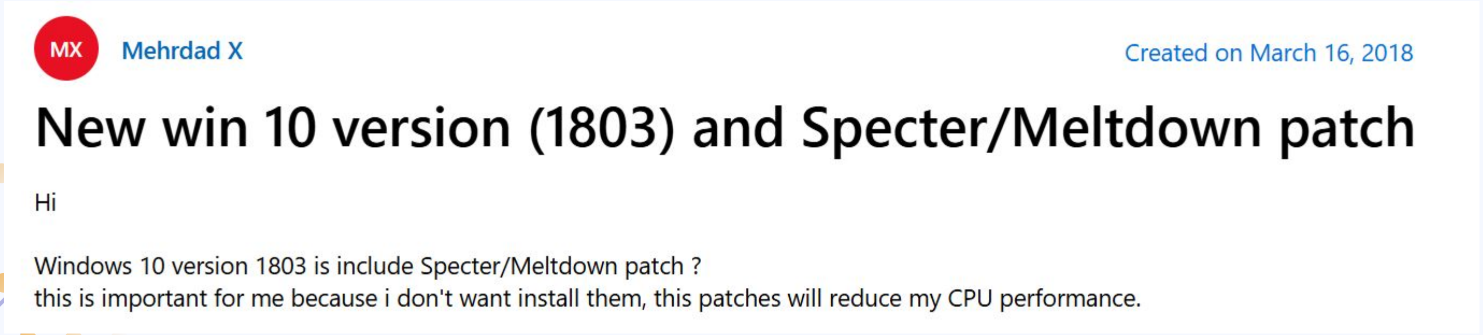
r/intel • 4 yr. ago
Renton577

How bad is it to disable Specter and Meltdown protections in 2021 for the extra performance?



Disable Spectre/Meltdown protection for a 4% performance boost

By NotHereToPlayGames
September 9, 2023 in Windows 10



Mehrdad X

Created on March 16, 2018

New win 10 version (1803) and Specter/Meltdown patch

Hi

Windows 10 version 1803 is include Specter/Meltdown patch ?
this is important for me because i don't want install them, this patches will reduce my CPU performance.