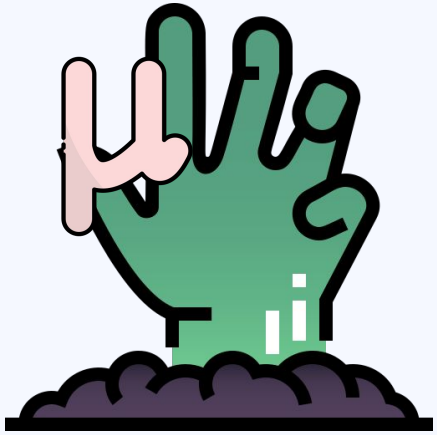
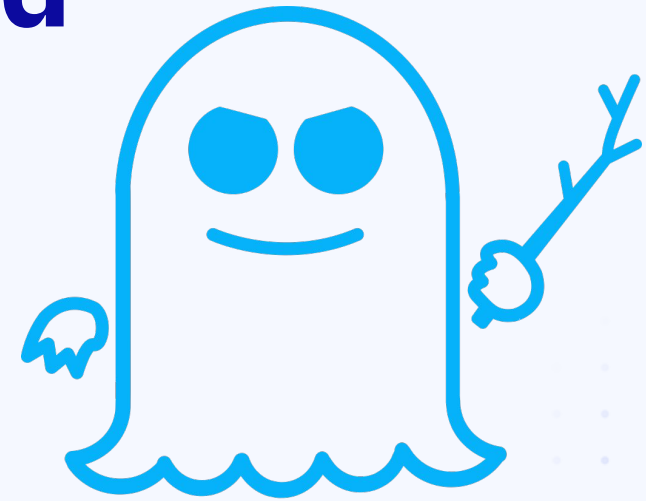
HORRORS OF ILP:

Spectre and dead µops





Reading

Spectre

- Paper
- Webpage

I see dead uops

- Paper
- Article

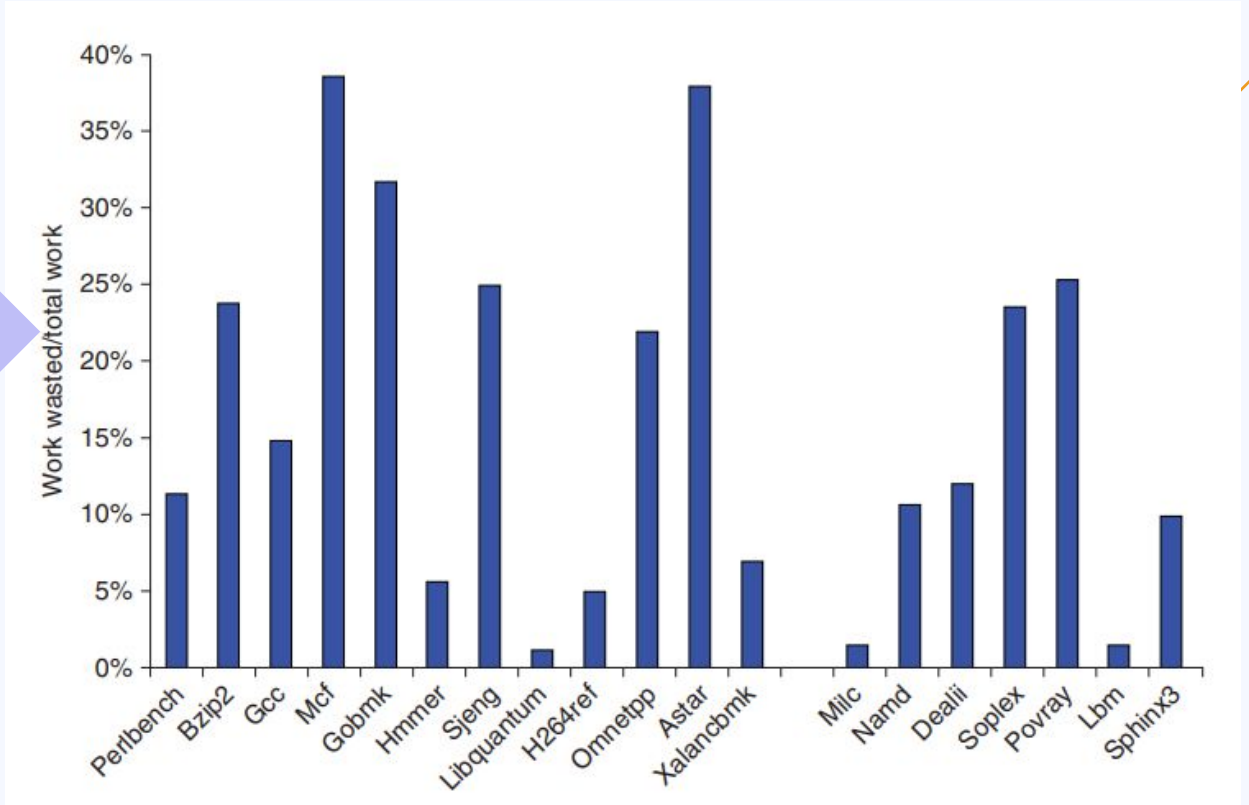
Bonus: Meltdown

- Paper
- Usenix page (slides, presentation)

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

H&P fig. 3.42

% of executed uops that were not committed



What effects would OOO/speculative execution have on the memory system?
(Hint: think protected access and/or caches)

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

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

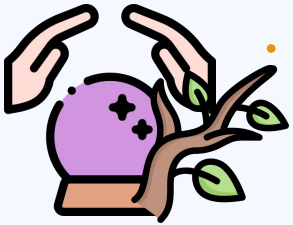
Can we learn a secret key' stored at some index j >= arraySize of arr1?

Spectre setup

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

1. Train the branch predictor

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



The branch will be taken

Spectre setup

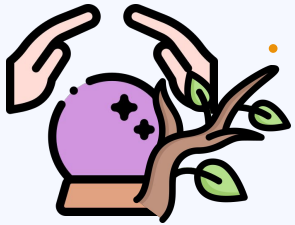
```
libFunc(x) {
    if (x < arraySize) {
        key = arr1[x];
        return arr2[key];
    }
}
```

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

Shared $

key'

arraySize **arr2**



The branch will be taken

Spectre attack

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

libFunc(large x)

Speculate taken
Takes a while to resolve due to uncached arraySize

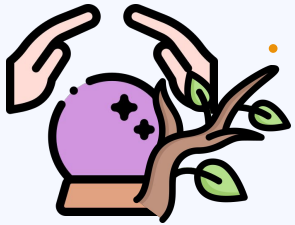
executes speculatively; fast

executes speculatively; brings arr2[key'] into cache

Shared $

key' **arr2[key']**

arraySize **arr2**



The branch will be taken

Spectre measure

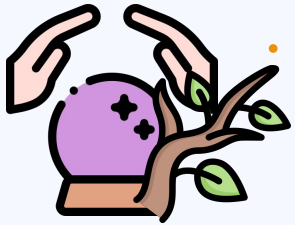
```
libFunc(x) {
    if (x < arraySize) {
        key = arr1[x];
        return arr2[key];
    }
}
```

access addresses that map to the same blocks as possible values of arr2[key’]
time results
if fast: can guess value of key’

Shared $

key’ **arr2[key’]**

arraySize **arr2**



The branch will be taken

How do we mitigate spectre?

Barrier instructions

Instructions that provide serialization/stop speculation

Different mechanisms and restrictions

Some only serialize memory operations

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

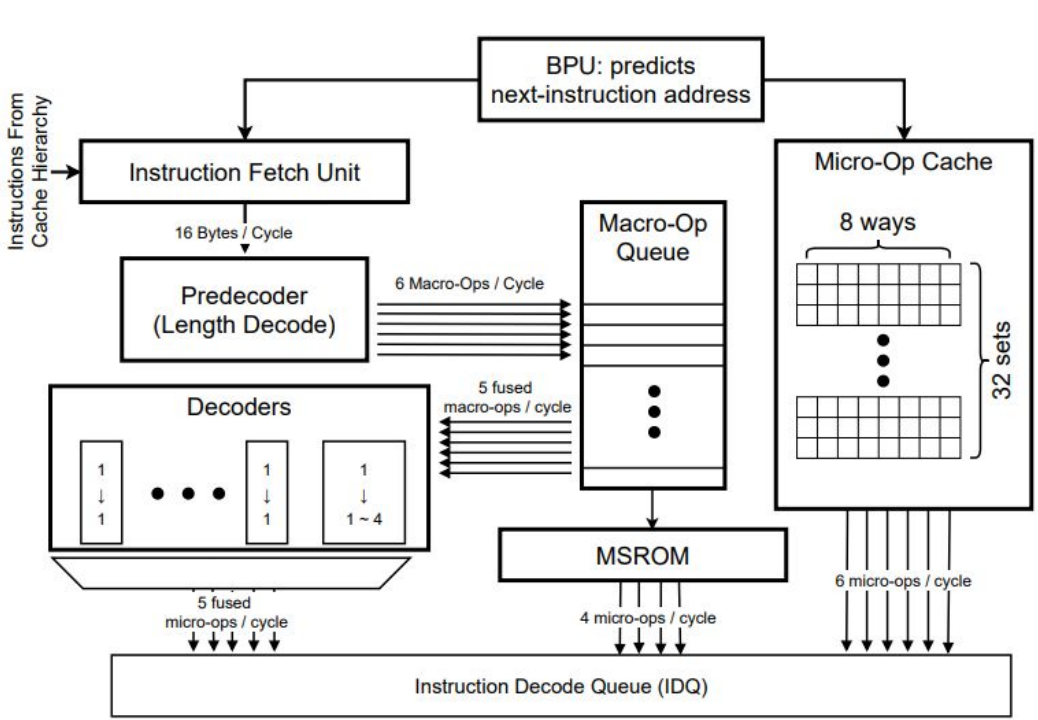


Fig. 1: x86 micro-op cache and decode pipeline

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

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

I see dead uops attack

```
1  char secret;
2  extern void victim_function(ID user_id) {
3    // authorization check bypassed by mistraining
4    if (user_id is authorized) {
5      asm volatile("lfence");
6      // LFENCE: stall the execution of
7      // younger instructions
8
9      // transmitter: ir
10     fun[secret]();
11   }
12 }
```

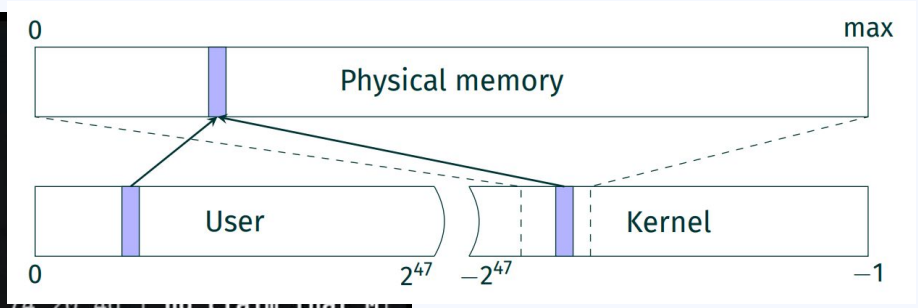
Indirect jump that depends on secret **not** mitigated by lfence!

Listing 5: Victim Method for our Variant-2 Attack

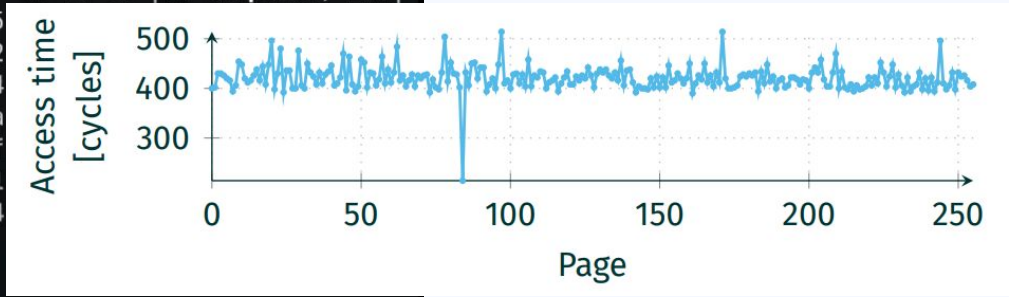
source

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
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
```

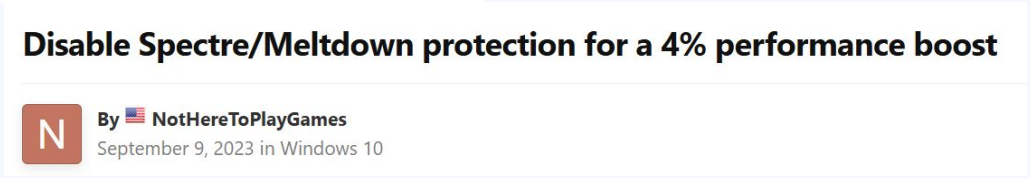



```
char data = *(char*) 0xffffffff81a000e0;
array[data * 4096] = 0;
```



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.