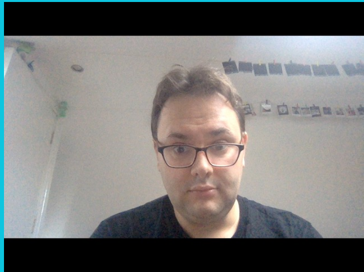
Meltdown and Spectre

Joseph Hallett



Last time…

We spoke about *speculative execution* and *branch prediction*

This time we're going to discuss a family of *real world* side channel attacks based on speculative execution

Meltdown and Spectre

Meltdown *melts down* security barriers...

- CVE-2017-5754

Spectre makes *speculative execution* scary…

- CVE-2017-5753 and CVE-2017-5715

Attacks allow us to leak secure memory (keys)

Effect nearly all OSs and processor architectures

- We do have **software** mitigations available now…
- …But they come at a considerable cost

So what does Spectre look like?

```
if (x < array1_size)
    y = array2[array1[x] * 4096];
```

So what does Spectre look like?

```
if (x < array1_size)
    y = array2[array1[x] * 4096];
```

`array1` is a pointer to some region of memory

We are allowed to access `array1_size` bytes of it

x is controlled by an attacker

4096 is the size of a cache page

Speculative Execution

```
if (x < array1_size)
    y = array2[array1[x] * 4096];
```

`if (x < array1_size)` should fail…

…but it is going to take some time to get through the CPU pipeline

- (We can engineer a cache for example)

We can trick the branch predictor into guessing that the branch will be taken

- (Make it succeed repeatedly beforehand!)

The second line will be executed speculatively…

Exceptions…

```
if (x < array1_size)
    y = array2[array1[x] * 4096];
```

`array2[array1[x] * 4096]` is going to cause a page miss

So a *new page* will be loaded into the CPU's cache

And that page will depend on the value of `array1[x]`

…which would normally trigger a segmentation fault

…which happens during the **LAST** stage of a CPU's pipeline

Rollback…

```
if (x < array1_size)
    y = array2[array1[x] * 4096];
```

Nevermind! The branch predictor has caught up now!

The **if** statement should never have been taken!

Lets roll back everything:

- Registers
- Exception flags
- Memory writes

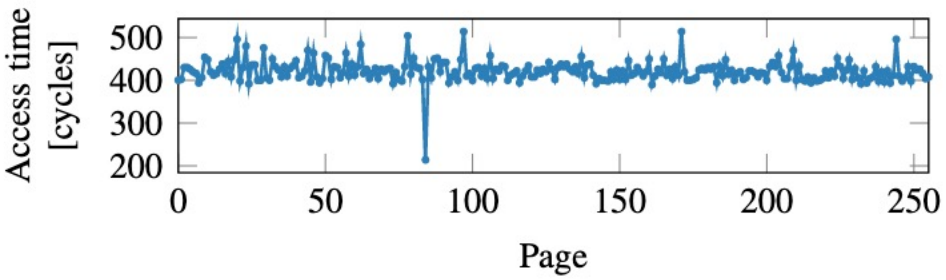
Oh no…

```
if (x < array1_size)
    y = array2[array1[x] * 4096];
```

…but not the CPU's cache

So the currently cached memory page is dependent on whatever was at that illegally accessed memory address…

So if you timed accessing each page of memory you could find out which page was quick to access and that would leak the value or `array1[x]`



Whoops.

Spectre

And it works via JavaScript too...

```
if (index < simpleByteArray.length) {
  index = simpleByteArray[index | 0];
  index = (((index * 4096)|0) & (32*1024*1024-1))|0;
  localJunk ^= probeTable[index|0]|0;
}
```

Listing 2: Exploiting Speculative Execution via JavaScript.

So if you can host a webpage on a shared server... (e.g. AWS)

…you can dump all of memory (given a few hours)

…for every process and user on there

…you can leak **every** key in memory

Spectre and Meltdown

Like the current pandemic... there are **lots** of variants

…some are more deadly than others

See the Meltdown and Spectre papers for more details

Same *basic* idea as shown here, different ways of triggering it…

So how are we going to fix this?

```
$ cat /sys/devices/system/cpu/vulnerabilities/{meltdown,spectre*}
Mitigation: PTI
Mitigation: usercopy/swapgs barriers and __user pointer sanitization
Mitigation: Full generic retpoline, STIBP: disabled, RSB filling
```

Some software mitigations we can do at the OS level

- see `/sys/devices/system/cpu/vulnerabilities/` on Linux
- Not perfect, but makes in **MUCH** harder to exploit

But on an Intel CPU Hyperthreading (SMT) makes exploiting **MUCH** easier…

RIP Hyperthreading