University of BRISTOL

Systems & Software Security

COMSM0050

2020/2021

bristol.ac.uk

University of BRISTOL

Spectre & Meltdown

bristol.ac.uk

Spectre & Meltdown

- Late 2017/early 2018
- Lots of discussion
- Can be fixed in the OS

- Spectre category of Attack
- Meltdown one instance
  - Speculative execution has proven problematic

Spectre & Meltdown

- Late 2017/early 2018
- Lots of discussion
- Can be fixed in the OS
- Computer get slower ☹

- Spectre category of Attack
- Meltdown one instance
  - Speculative execution has proven problematic

Spectre & Meltdown

- Category of attack affect:
  - all OSes
  - All hardware (at different degree)
- Hardware cannot be patched

Spectre & Meltdown

- Category of attack affect:
  - all OSes
  - All hardware (at different degree)
- Hardware cannot be patched
- Can be exploited via simple javascript

The problem?

- Speculative execution
  - Try to guess what will be executed
  - If correct all good!
  - If wrong roll back!

The problem?

- Speculative execution
  - Try to guess what will be executed
  - If correct all good!
  - If wrong roll back!
- CPU registers are rolled back…
- … but not (always) the CPU cache

The problem?

- Speculative execution
  - Try to guess what will be executed
  - If correct all good!
  - If wrong roll back!
- CPU registers are rolled back…
- … but not (always) the CPU cache
- … and authorization is checked at the end of the pipeline

The problem?

- Speculative execution
  - Try to guess what will be executed
  - If correct all good!
  - If wrong roll back!
- CPU registers are rolled back…
- … but not (always) the CPU cache
- … and authorization is checked at the end of the pipeline

Attacker can get memory value he should not access in the CPU cache

Meltdown phases

1. **Reading secret**: inaccessible memory content chosen by an attacker is loaded into a register
2. **Transmit secret**: instruction access cache line based on the secret value
3. **Receive secret**: attacker use fetch and reload to determine the accessed cache line

Repeat to dump entire kernel memory

Meltdown example

```
if (spec_cond) {
        unsigned char value = *(unsigned char *)ptr;
        unsigned long index2 = (((value>>bit)&1)*0x100)+0x200;
        maccess(&data[index2]);
}
```

Meltdown example

```
if (spec_cond) {
        unsigned char value = *(unsigned char *)ptr;
        unsigned long index2 = (((value>>bit)&1)*0x100)+0x200;
        maccess(&data[index2]);
}
```

Under attacker control, we want this branch to be speculatively executed

Meltdown example

```
if (spec_cond) {
        unsigned char value = *(unsigned char *)ptr;
        unsigned long index2 = (((value>>bit)&1)*0x100)+0x200;
        maccess(&data[index2]);
}
```

data under attacker control

Meltdown example

```
if (spec_cond) {
        unsigned char value = *(unsigned char *)ptr;
        unsigned long index2 = (((value>>bit)&1)*0x100)+0x200;
        maccess(&data[index2]);
}
```

ptr address we don't have access to

Meltdown example

```
if (spec_cond) {
        unsigned char value = *(unsigned char *)ptr;
        unsigned long index2 = (((value>>bit)&1)*0x100)+0x200;
        maccess(&data[index2]);
}
```

bit shift to access a bit of data

Meltdown example

```
if (spec_cond) {
        unsigned char value = *(unsigned char *)ptr;
        unsigned long index2 = (((value>>bit)&1)*0x100)+0x200;
        maccess(&data[index2]);
}
```

generate address from data value

Meltdown example

```
if (spec_cond) {
        unsigned char value = *(unsigned char *)ptr;
        unsigned long index2 = (((value>>bit)&1)*0x100)+0x200;
        maccess(&data[index2]);
}
```

use address to pull in cache we control

Meltdown example

- char value = *secret_kernel_pointer
- mask bit you want to read
- calculate offset in data

addres	
0x100	
0x200	
0x300	

Meltdown example

- char value = *secret_kernel_pointer
- mask bit you want to read
- calculate offset in data

IF 1

addres	
0x100	
0x200	
0x300	DATA

Meltdown example

```
time = rdtsc();
maccess(&data[0x300]);
delta3 = rdtsc() - time;

time = rdtsc();
maccess(&data[0x200]);
delta2 = rdtsc() - time;
```

access time will depend if value in cache or not.

Cache side channel.