


Plan

- Recitation Qs

- Meltdown

↳ Load kernel data into cache.

↳ Read kernel data from cache.

Logistics

* Design project due May 11

* If you like this, consider G.S060 in Fall 2021

Recitation Questions

1. What is the meltdown attack?

- Use OOO execution to read "privileged data" from unprivileged process.
- Side-channel attack... exploiting unspecified behavior

2. How does it work?

- "Transient instruction sequence" -----

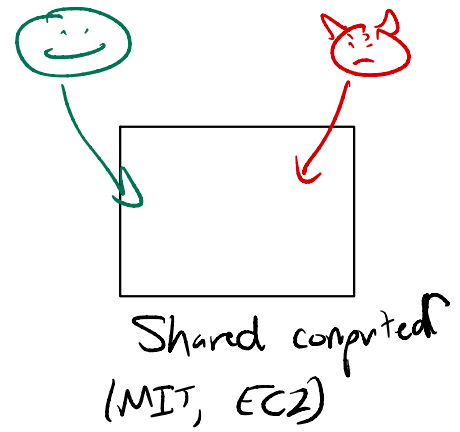
3. Why is this attack possible?

- CPU designers prioritize speed
-

Meltdown

Goal: Read data of another user on the same machine.

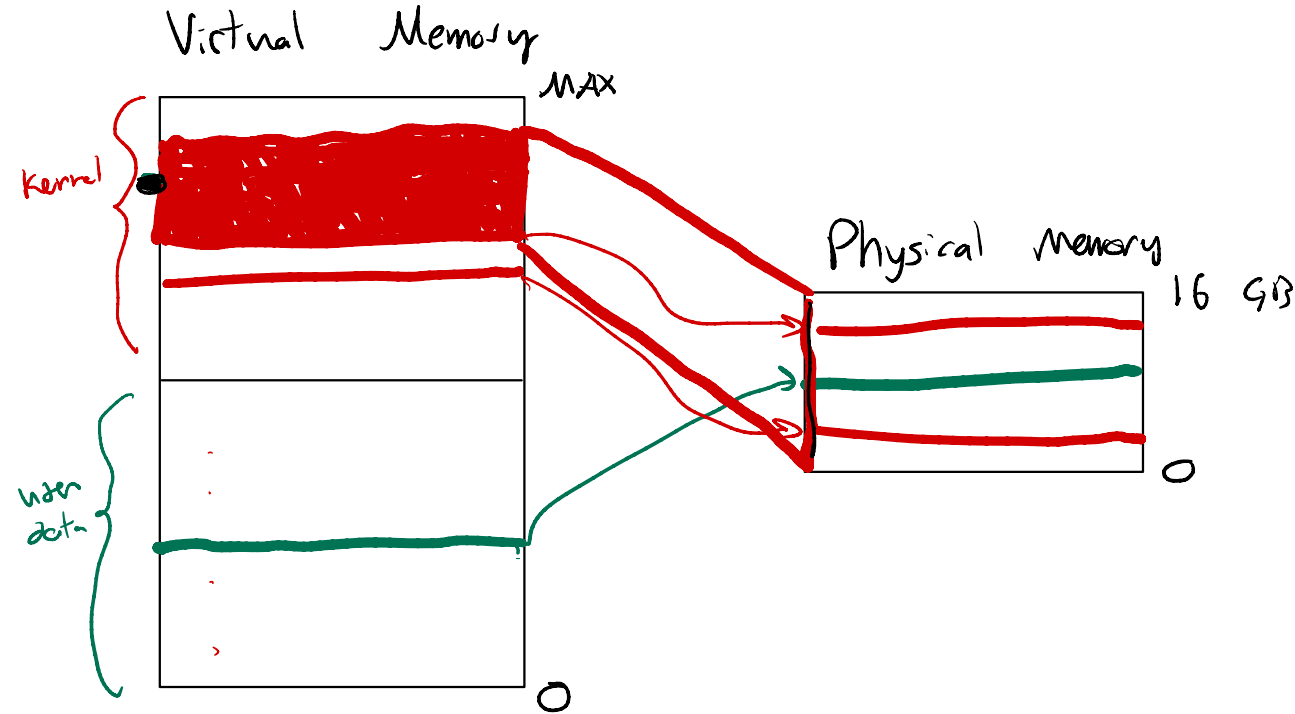
- Email
- Password
- Crypto key
- ...



~~This~~ This attack will no longer work.

↳ "Spectre" attacks. 

Meltdown (restated): Read arbitrary addr in memory, bypassing H/w permissions checks.



Step 1: Load kernel data into register.

```
int main() {  
    char buf[256 * 128];  
    byte k = * kernel - addr;  
    char stuff = buf[k * 128];  
}
```

CPU will...

- Load data from memory
- Cache the data

* - Check permissions

- Crash prog if perm check fails.

* - Execute next instruction
(Speculative execution)

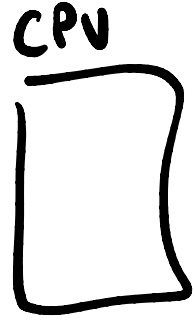
The diagram illustrates the memory access flow during a cache miss. On the left, the **CPU** contains an **Execution unit** with a memory access pattern: `char k = buf[k]` (highlighted in orange), followed by `buf[0]` (highlighted in green), `buf[1]`, `buf[2]`, `buf[3]`, and `buf[255]`. A green line connects `buf[0]` to the **Cache**. The **Cache** contains `buf[k]` (red line) and a green wavy line representing `buf`. A red arrow points from `buf[k]` in the cache to `buf[k]` in the **Physical Mem RAM**. A green arrow points from the green wavy line in the cache to the green wavy line in the **Physical Mem RAM**. A grey arrow points from the **Cache** to the **Physical Mem RAM**. The **Physical Mem RAM** contains `buf` (green wavy line). The flow is labeled **FAST** for the cache hit and **SLOW** for the cache miss.

A hand-drawn diagram illustrating a buffer. At the top, a horizontal red line is labeled "buf". Below this, a green wavy line represents the buffer's state. A small black dot is positioned on the left side of the green wavy line, near the "buf" label.

Possible pros to avoid crashing

⇒ 256 possible
vals of k
... try them all

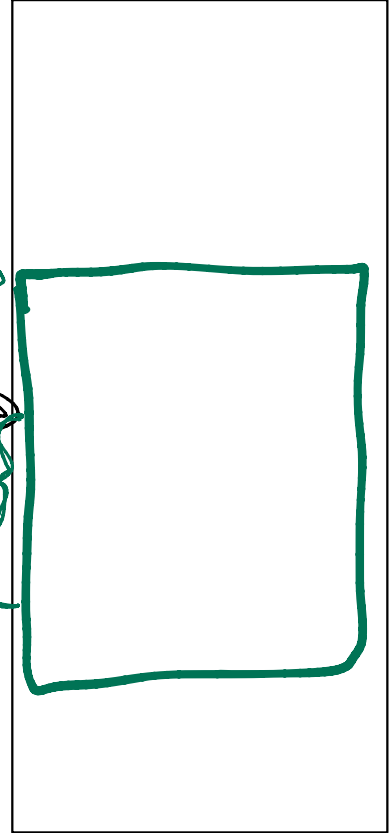




"Prefetching"

buf[3], ..., buf[3+128]

RAM



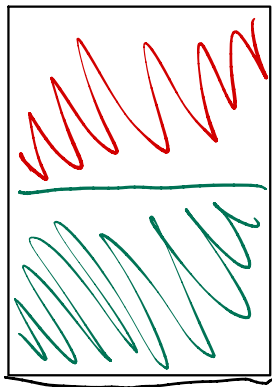
Mitigations

SW/OS

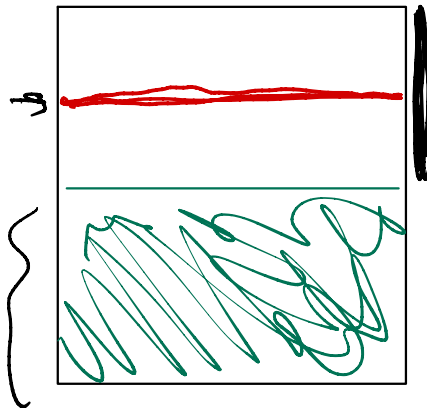
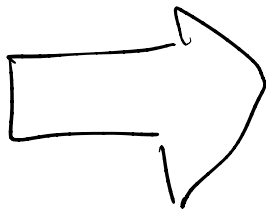
kernel
data

user
data

HW



protected
by
HW perm
bits



Plan

- Recitation Qs

- Meltdown

↳ Load kernel data into cache.

↳ Read kernel data from cache.

Logistics

* Design project due May 11

* If you like this, consider G.S060 in Fall 2021

1. what is MeltDown?

↳ attack that uses OOO execution

2. How does it work?

↳ Read cached results.... covert channel

3. Why is the attack possible?

↳ perf benefits

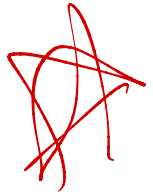
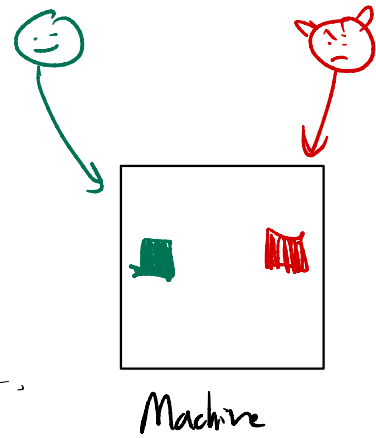
↳ race condition.

Meltdown

Goal:

Read data of another user

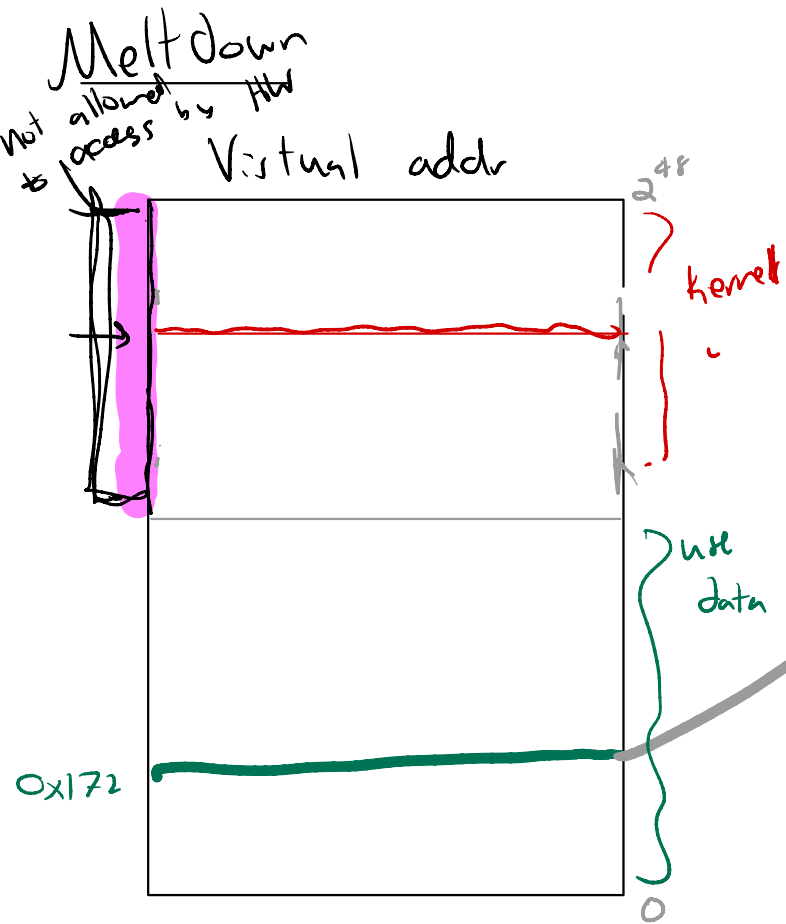
- email, password, crypto key, ...



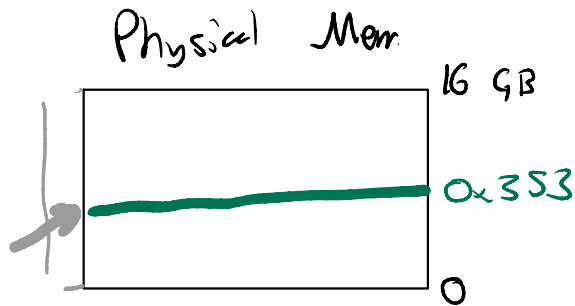
This attack no longer works



↳ "Spectre attack"



KAISER / KPTI



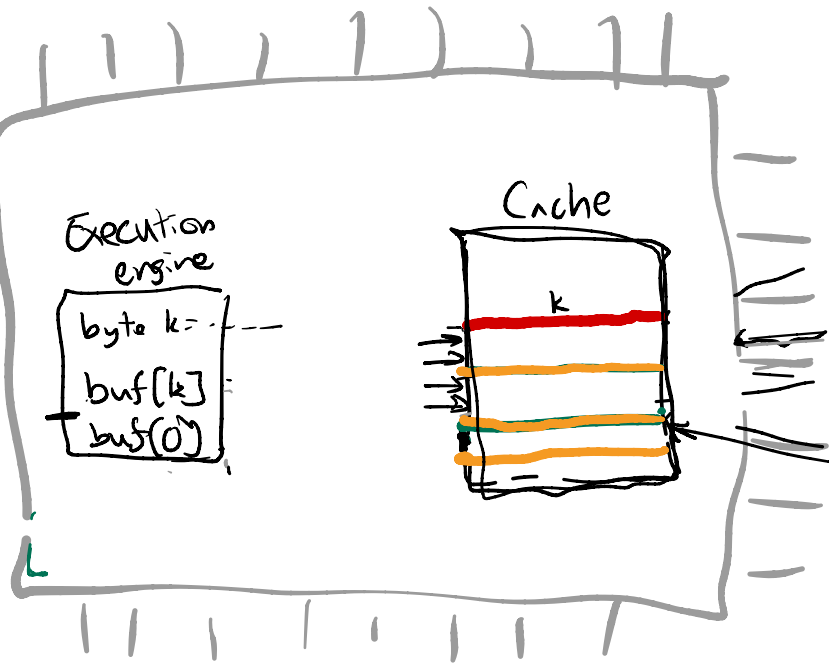
Step 1: Load kernel data into register.

```
int main() {  
    [ byte buf[256];  
    byte k = *kernel - addr;  
    byte stuff = buf[k];  
    x  
}
```

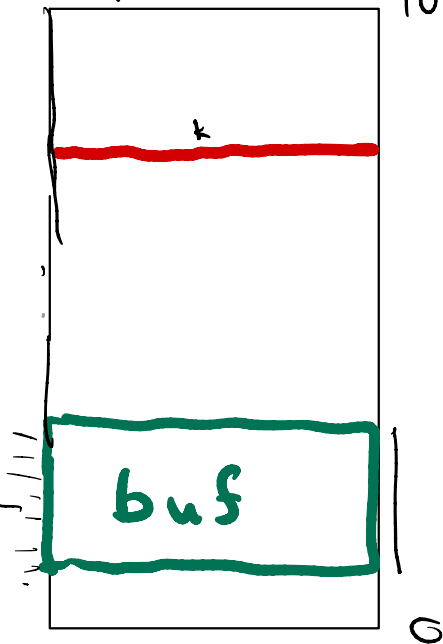
Speculative
execution
ooo execution

CPU will...

- Load data from mem
- Check perm bits
- Crash prog if perm check fail kernel mem say "no"
- Execute next instruction Making coffee



RAM (memory)
16 GB



→ buf(0) buf(1), ...
clflush

buf(256)

byte buf [128 × 256]

"Prefetching"

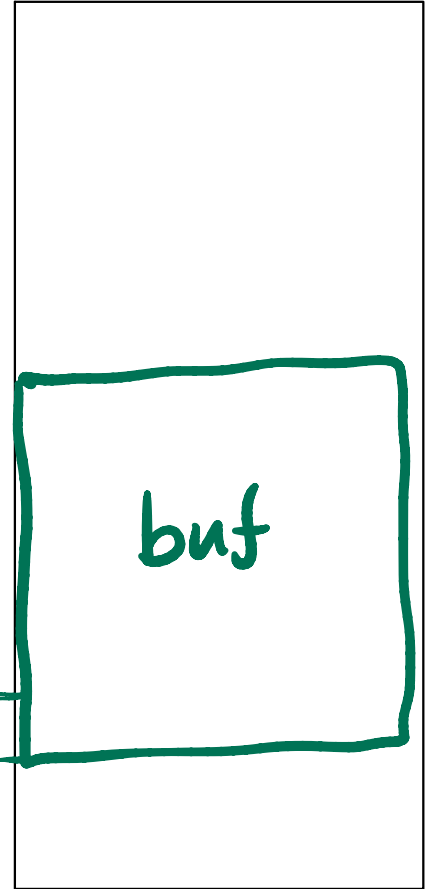
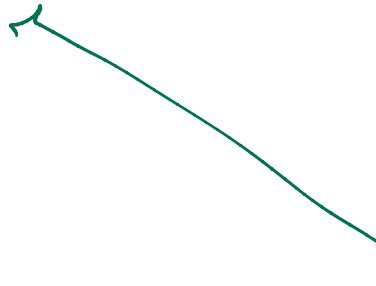
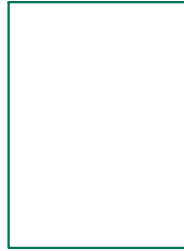
buf[0]

buf[1]

buf[2]

⋮

buf[127]



Mitigation

HW: - Don't speculate at all. ☹️

- Separate caches

▶ Don't speculate past permission checks (?)

Software: