Recitation 20: Meltdown
Plan

- Recitation Qs

**Meltdown**

→ Load kernel data into cache
→ Read kernel data out of cache

---

**Logistics**

* Design project due May 21
* If you like this, consider 6.5060 in Fall '21
Poll:

Do you know what a cache is?
Recitation Questions

1. What is the Meltdown attack?
   - Technique to read kernel memory from user space
   - Doesn’t work on modern processors or fully patched OSes (Linux, MacOS, etc.)

2. How does it work?
   a) Trick CPU into loading item into cache whose address depends on kernel data
   b) Use cache-timing to extract this info from cache
   c) Repeat

3. Why is this attack possible?
   - CPU designers prioritize speed
     - Didn’t really expect this “side-channel” leakage to be so problematic.
   - CPU “speculates past” permissions checks
Melt down

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

- Email
- Cryptographic keys
- Passwords

Assumes: Attacker running as unprivileged user

- e.g. two MIT users on the same cluster machine
- e.g. two users on Amazon EC2

This particular attack will no longer work on a modern CPU/OS.

Other related attacks ("Spectre") still do...
**Meltdown (Restated)**

**Goal:** Read arbitrary address in memory, bypassing HW permissions checks.

→ All of physical mem is mapped in vaddr space
→ Most of this is not available to user proc
→ Reading arbitrary vaddr is enough to read any location in physical memory.

---

*Virtual memory*

<table>
<thead>
<tr>
<th>Kernel data</th>
<th>OS kernel data</th>
</tr>
</thead>
</table>

| User data   | Other user's data |

| Physical Mem | 16 GB |

| 0            | 0     |
A useful analogy:

- Go to Dr. Lacwts’ favorite cafe, ask “I’ll have the same thing Dr. Lacwts usually gets.”

- Barista calls Dr. Lacwts to ask if he can divulge her usual order.

- While the phone is ringing, he pulls 4 shots of espresso and froths 8 oz of almond milk.

- Dr. Lacwts finally answers the phone. She tells the barista to not reveal her secret coffee order.

- Barista won’t give you a coffee.

The barista leaked the secret info before performing the permissions check.
Step 1: Load kernel data into register.

```c
int main() {
    char k = * kernel_addr;
    // print data
}
```

CPU will:

- Load data from memory
- Check permissions bits
- Crash program (exception) if permission check fails
Step 2: Access data in cache based on register contents. [victim data]

```c
int main() {
  char buf [4096];
  char k = * kernel_addr;
  char stuff = buf [k];

  CPU will...
  - Load data from memory
  - Check permissions bits
  - Execute next instruction (speculatively)
  - Crash program (exception) if perm check fails
```
What happened here?

The data `bus[k]` gets loaded into CPU cache.

Then program crashes. (SegFault)

Cache stays as is.

Learning which element of `buf` got cached reveals `k`!

Key: possible for program to handle the exception and continue running?
Step 3: Figure out which element of buf the CPU accessed.

* Access to buf[\(k\)] $\Rightarrow$ Fast (CACHED!)

* Access to all other parts of buf $\Rightarrow$ Slow

$\rightarrow$ 256 possible values of \(k\).

Try them all and time accesses!
Game: Cache-timing attacks

One student is memory subsystem (fetch pages into cache, reply to response)

One student is honest user
- accesses a page by DM to memory subsystem

One student is attacker
- tries to guess which page the honest user accessed
- Can also flush cache pages.
Mitigations

Can't trust HW to enforce perm perm checks

Software/OS: KAISER/KPTI

HW was too greedy

CPU design: Do not speculate past permissions checks

CPU will...
- Load data from memory
- Check permissions bits
- Crash program (exception) if perm check fails
- Execute next instruction