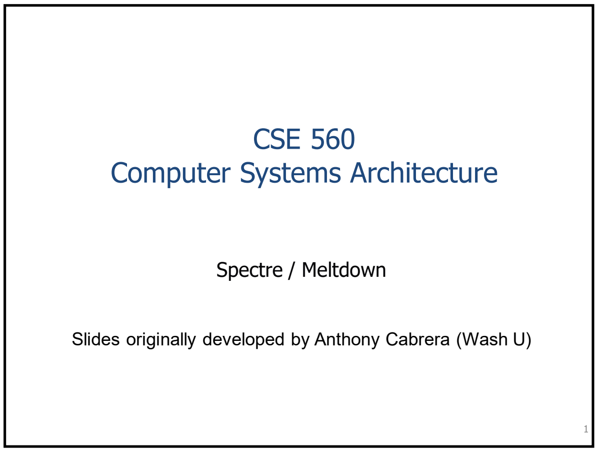
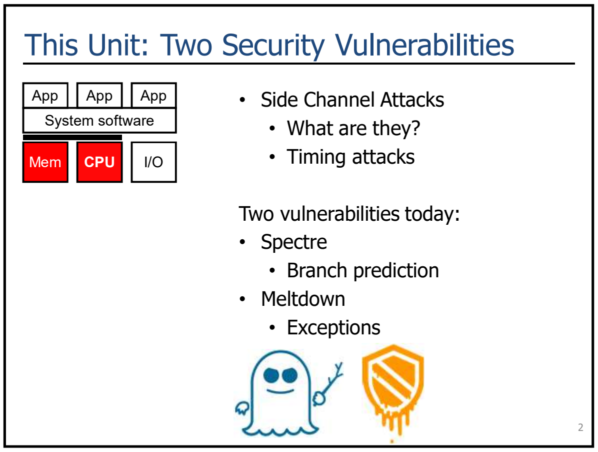


CSE 560
Computer Systems Architecture

Spectre / Meltdown

Slides originally developed by Anthony Cabrera (Wash U)

1



This Unit: Two Security Vulnerabilities

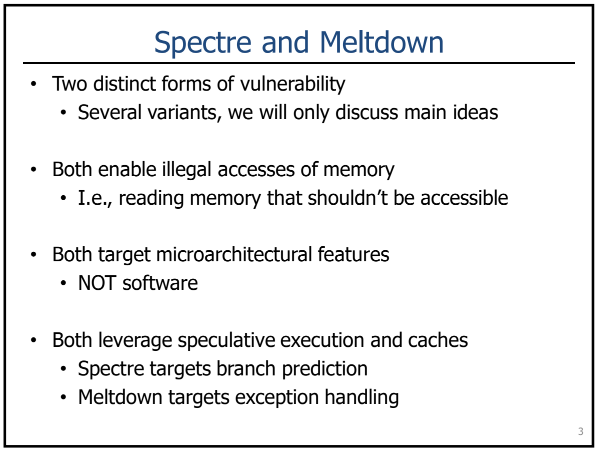
App
App
App
System software
Mem
CPU
I/O

- Side Channel Attacks
  - What are they?
  - Timing attacks

Two vulnerabilities today:

- Spectre
  - Branch prediction
- Meltdown
  - Exceptions

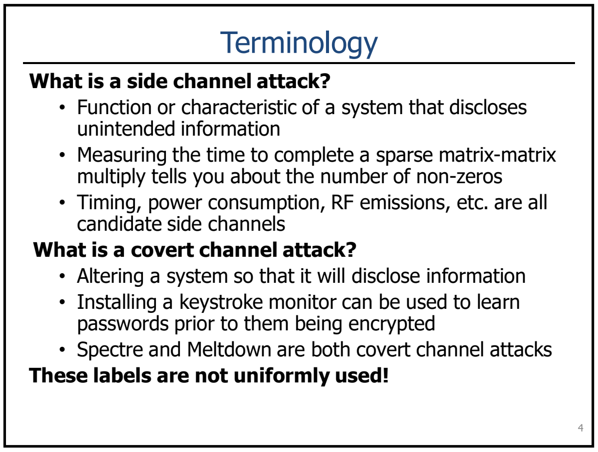
2



Spectre and Meltdown

- Two distinct forms of vulnerability
  - Several variants, we will only discuss main ideas
- Both enable illegal accesses of memory
  - I.e., reading memory that shouldn't be accessible
- Both target microarchitectural features
  - NOT software
- Both leverage speculative execution and caches
  - Spectre targets branch prediction
  - Meltdown targets exception handling

3



Terminology

What is a side channel attack?

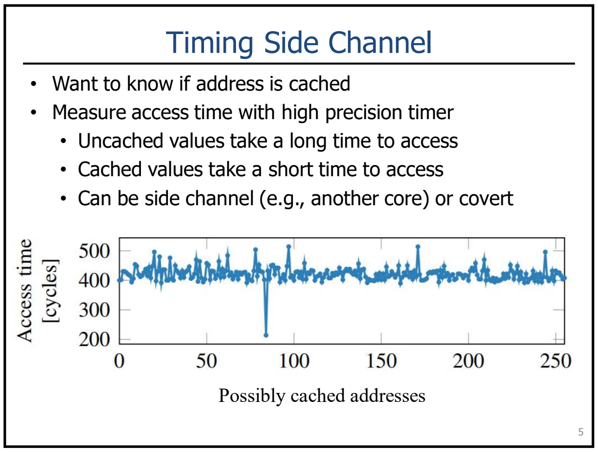
- Function or characteristic of a system that discloses unintended information
- Measuring the time to complete a sparse matrix-matrix multiply tells you about the number of non-zeros
- Timing, power consumption, RF emissions, etc. are all candidate side channels

What is a covert channel attack?

- Altering a system so that it will disclose information
- Installing a keystroke monitor can be used to learn passwords prior to them being encrypted
- Spectre and Meltdown are both covert channel attacks

These labels are not uniformly used!

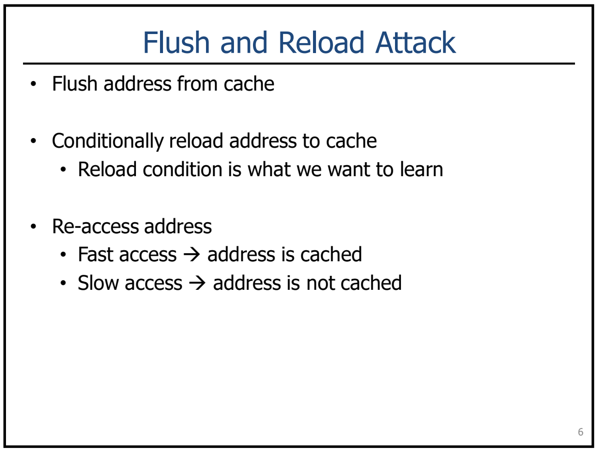
4



Timing Side Channel

- Want to know if address is cached
- Measure access time with high precision timer
  - Uncached values take a long time to access
  - Cached values take a short time to access
  - Can be side channel (e.g., another core) or covert

5



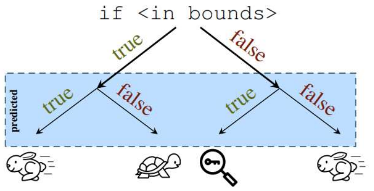
Flush and Reload Attack

- Flush address from cache
- Conditionally reload address to cache
  - Reload condition is what we want to learn
- Re-access address
  - Fast access → address is cached
  - Slow access → address is not cached

6

Spectre

- Exploit the branch predictor in OoO pipelines
  - Train predictor to take one path
  - Switch to other path and access illegal memory (e.g., via out of bounds array access)



7

Exploiting Branch Prediction

```
byte a[N];
int size_a = N;
byte b[256];
…
foo (int x){
    if (x < size_a)
        y = b[a[x]];
}
```

Do this many times (with good `x`) so branch predicts true

Next, flush `b` and `size_a`

Then call `foo(bad_illegal_x)`

- Result is `b[a[bad_illegal_x]]` is loaded into cache
- Even though `if` test ultimately fails!

8

Exploiting Branch Prediction

```
byte a[N];
int size_a = N;
byte b[256];
…
foo (int x){
    if (x < size_a)
        y = b[a[x]];
}
```

With `size_a` not in cache, `if` takes many cycles to resolve

Initially `b` not cached, but speculative execution puts one entry of `b` into cache

- Use timing side channel to determine which entry cached
- That entry is value of `a[bad_illegal_x]`!

9

Notes about Spectre

- What if accessed address is protected?

  `a[bad_illegal_x]`

  might trigger exception!
  - Nope, because bad load never commits
  - It does trigger cache fill, however, which is the problem
- Variant is indirect branch exploitation
  - Train predictor to make speculative jump to bad code
  - Bad code alters microarchitecture state (e.g., cache)
  - Leak information via side channel

10

What addresses are vulnerable?

- Certainly addresses within same process:

  `a[bad_illegal_x]`

  can be any logical address readable by the process
- But what about other processes?
  - Flush and reload must be in process (covert)
  - Timing side channel can be another process
- And what about kernel memory?
  - eBPF (extended Berkeley packet filter) is code provided by user application that runs in kernel space
  - Static analysis performed, but attack code looks good wrt array bounds (they are explicitly checked)!

11

What machines are vulnerable?

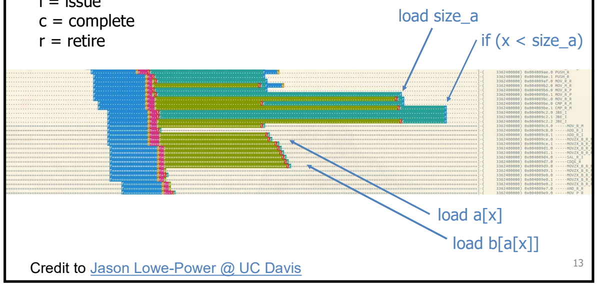
- Any microarchitecture that has the following:
  - Cache
  - Out of order execution
  - Branch prediction
- Manufacturers include
  - Intel
  - AMD
  - Arm
  - Others (including gem5 stock OoO model!)

12

OoO Simulation in gem5

f = fetch
d = dispatch
n = rename
p = dispatch
i = issue
c = complete
r = retire

"." one clock cycle of normal execution
"=" instruction is eventually squashed



Credit to Jason Lowe-Power @ UC Davis

What can be done?

- Spectre mitigation is difficult
- It represents a whole class of vulnerabilities
- Disabling microarchitectural features has substantial performance impact
  - 14% slowdown reported by several groups
- Patches available for specific variants

Meltdown

- Exploits features of virtual memory
- Often, all of physical memory is mapped into user space (and protected from user access) to speed system calls (user ←→ kernel transitions)
- Core idea is to access illegal memory and (like Spectre) impact the cache state before load is rolled back

Exploit

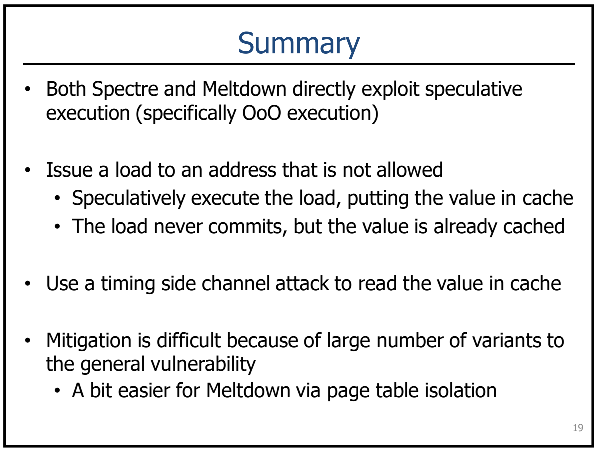
- Processor attempts a load instruction
  - Address is in protected memory
  - The instruction issues, which triggers both:
    1. Reading address from main memory
    2. Checking the privilege bits in the virtual memory
- Privilege check fails, so load never commits
  - By the time of the failure, cache can already be updated
- Value is now in cache, proceed to use timing side channel
  - Same as Spectre here on out

What machines are vulnerable?

- Any microarchitecture that has the following:
  - Cache
  - Out of order execution
  - Accessible page table concurrent with permission check
- Manufacturers include
  - Intel
  - Arm
  - NOT AMD – check permissions when reading page table

What can be done?

- Meltdown mitigation is more straightforward
- Main memory read happens concurrently with privilege check (which is a VM function)
- But main memory read requires access to page table
  - Specifically, a page table entry that is in the kernel!
- Solution is to separate user and kernel space page tables
  - Called "kernel page table isolation" or KPTI
  - Access to kernel page table from user mode refuses to provide physical address
  - Therefore, it never gets cached



Summary

- Both Spectre and Meltdown directly exploit speculative execution (specifically OoO execution)
- Issue a load to an address that is not allowed
  - Speculatively execute the load, putting the value in cache
  - The load never commits, but the value is already cached
- Use a timing side channel attack to read the value in cache
- Mitigation is difficult because of large number of variants to the general vulnerability
  - A bit easier for Meltdown via page table isolation