Topic 10: Side Channels, Meltdown, and Spectre, Oh My!

ECE 4750 Computer Architecture

Prof. Anne Bracy

Based on slides by D. Zagieboylo, M. Hill, K. Sekniqi

Side Channels

- An *extra* way to learn information about a program's execution
- Usually a way for an *attacker* to bypass security mechanisms

Side Channels

- An *extra* way to learn information about a program's execution
- Usually a way for an *attacker* to bypass security mechanisms
  - Power consumption
  - Electromagnetic Radiation
  - Responsiveness / Faults
  - Timing

- Timing attacks are a BIG concern:
  - Can be executed remotely
  - Hard to prevent all secret-dependent timing
  - Small differences can be amplified with repetition
  - Very *stealthy*

Timing Side Channels

What influences a program's execution time?

- Dynamic instruction count
  - Which branches get executed
- Cycles per instruction
  - Variable latency instructions (e.g., division)
  - TLB Hit or Miss (Page Fault)
  - Cache Hit or Miss
  - Correct vs. Incorrect Speculation
- Clock frequency
  - DVFS (Dynamic Voltage-Frequency Scaling)

Cache Timing Channel

- *very* common side channel
  - Fast/easy to execute
  - High signal to noise (don't have to repeat much to be sure it worked)

- How it works: Prime + Probe:
  1. Setup cache state
  2. Run victim
  3. Time memory accesses

"Which cache set did the victim access?"

Prime + Probe Example

```
//Attacker: (e.g., user process)
char arr[N_CACHE_SETS*LINE_SIZE];
for (int i = 0; i < N_CACHE_SETS; i++) {
      arr[i*LINE_SIZE] = 0;
}
```

Cache is now completely filled with attacker's array.

idx	Tag
63	&arr[63]
62	&arr[62]
61	&arr[61]
...	...
2	&arr[0]
1	&arr[1]
0	&arr[0]

Prime + Probe Example

```
//Attacker: (e.g., user process)
char arr[N_CACHE_SETS*LINE_SIZE];
for (int i = 0; i < N_CACHE_SETS; i++) {
      arr[i*LINE_SIZE] = 0;
}
//Call Victim Code (e.g., via syscall)
  ...
  victim[secret] = data;
  ...
```

idx	Tag
63	&arr[63]
62	&arr[62]
61	&victim[secret]
...	...
2	&arr[0]
1	&arr[1]
0	&arr[0]

Prime + Probe Example

```
//Attacker: (e.g., user process)
char arr[N_CACHE_SETS*LINE_SIZE];
for (int i = 0; i < N_CACHE_SETS; i++) {
      arr[i*LINE_SIZE] = 0;
}
//Call Victim Code (e.g., via syscall)
  ...
  victim[secret] = data;
  ...
//Return to Attacker:
for (int i = 0; i < N_CACHE_SETS; i++) {
      time_start();
      arr[i*LINE_SIZE] = 0;
      time_end();
}
```

idx	Tag	
63	&arr[63]	Hit
62	&arr[62]	Hit
61	&arr[61]	MISS
...	...	
2	&arr[0]	Hit
1	&arr[1]	Hit
0	&arr[0]	Hit

Prime + Probe Example

Cache Hit (Fast!)
- Victim was not here

Cache MISS (Slow)
- Attacker learns **index bits** of secret memory address

`&victim[secret] is 0x???? 3f ??`

Can be helpful:
- if you already know `&victim`
- or if you only need to limit the number of possibilities for `secret`

idx	Tag	
63	&arr[63]	Hit
62	&arr[62]	Hit
61	&arr[61]	MISS
...	...	
2	&arr[0]	Hit
1	&arr[1]	Hit
0	&arr[0]	Hit

Cache Timing Channels

In reality, more complicated
- Multi-level caches
- Associativity
- Hardware Prefetchers
- Virtual Memory (Address Translation)
- Non-secret memory accesses (noise)

Can still execute $ timing attacks
- Reverse Engineering of HW
- Repeated execution of attack
- Statistical analysis
- Other attacks (e.g., Flush+Reload)

Solutions?

👎 Add more noise
(you'll lose the arms race usually)

⛔ Partition Cache
(doesn't help if victim & attacker are in same user-space process - costs efficiency)

⛔ Avoid secret-dependent LW/SW
(hard (or impossible) to do)

Recent Events – Transient Execution Attacks

- 2018
  - Meltdown & Spectre – [Jann Horn, Google Project Zero]
    Also , independently, Paul Kocher
  - Both are *microarchitectural attacks* that allow the user to exploit **speculative execution** to learn secret data
  - Make $ timing channels super easy to exploit – nearly NO statistical analysis necessary, can pick *any address you want to leak*
  - Meltdown affects almost every Intel chip made since 1995, and some ARM chips
    Spectre affects Everychip, Everywhere, All at once.
  - Intel® pushes out several microcode (HW) patches that…don't work and cause BSOD
  - OS, Compiler & Browser Mitigations (KPTI, SLH, Retpoline) start to be rolled out

Recent Events – Transient Execution Attacks

- 2018
  - Meltdown & Spectre – [Jann Horn, Google Project Zero]
    Also , independently, Paul Kocher
- 2019
  - Spectre Variants (Speculative Store Bypass, Foreshadow, Zombieload) continue to haunt us
  - Numerous *new microarchitectural designs to avoid Spectre* are proposed at high profile research conferences
  - No new word from Intel, AMD, ARM, etc. on *Spectre-secure* designs
- 2020-2022
  - ***Even more Spectre attacks. Old defenses broken. New defenses proposed. Repeat.***

Recent Events – Transient Execution Attacks

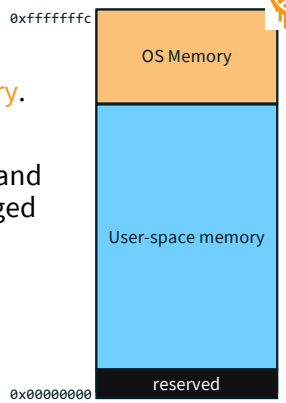
- 2018
  - Meltdown & Spectre – [Jann Horn, Google Project Zero]
    Also , independently, Paul Kocher
- 2018-19
  - OS patches for Meltdown released
  - Chipmakers plan to fix Meltdown in future HW
  - SW patches for Spectre_v1 & v2 developed.
    Mostly unused outside Google Chrome & Cryptographic libraries
- 2020-2022
  - Spectre patches gain more traction, incorporated into LLVM
  - More variants discovered, highlights need for new design, not just adhoc patches
  - Still an open problem, the attack-defense vicious cycle continues.

Background on Memory space

The virtual address space of each process contains user-level memory and OS memory.

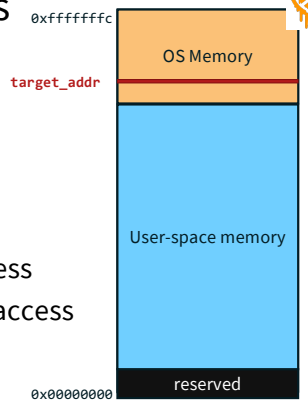
This is convenient for handling exceptions and making system calls (just change to privileged mode and start fetching OS code).

User-level process cannot load from OS memory. This is a permission violation.



Background on Memory Checks

```
x = *target_addr; // user-level code
```



→TLB detects illegal memory violation

→instruction will throws an exception

→seg fault kills the process**WHEN does detection & suppression happen??**

EARLY: AMD seems to suppress at TLB access

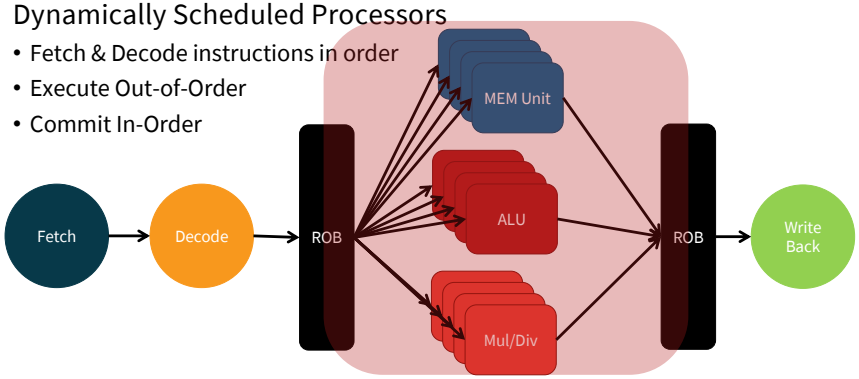
LATE: Intel seems to suppress *after* cache access

- Architectural state not changed
- *Micro-architectural state is changed!*

Meltdown – In Detail

Dynamically Scheduled Processors

- Fetch & Decode instructions in order
- Execute Out-of-Order
- Commit In-Order

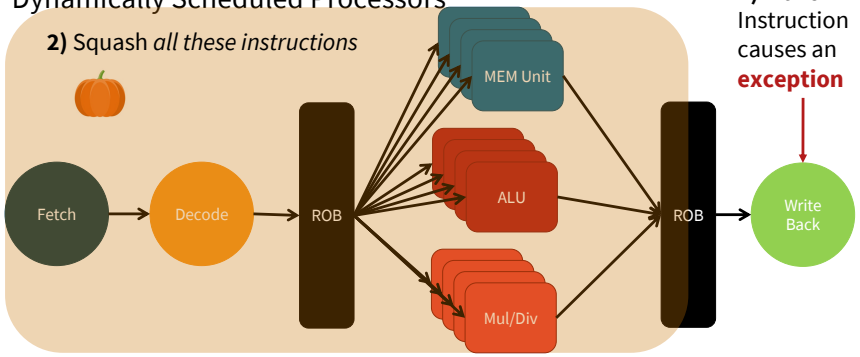


Meltdown – In Detail

Dynamically Scheduled Processors

2) Squash *all these instructions*

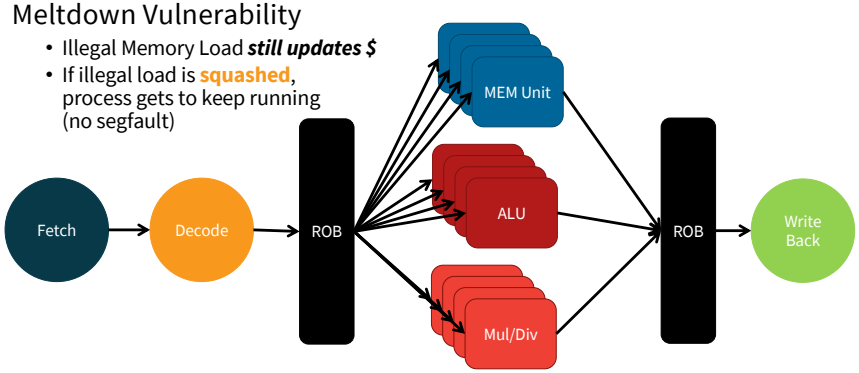
1) If this Instruction causes an **exception**



Meltdown – In Detail

Meltdown Vulnerability

- Illegal Memory Load ***still updates $***
- If illegal load is **squashed**, process gets to keep running (no segfault)



Meltdown – In Detail

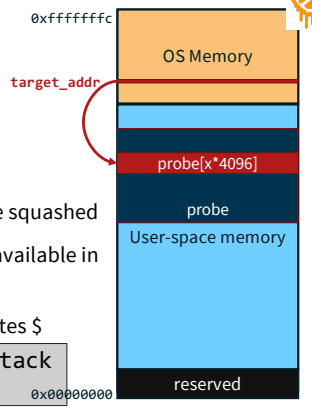
- Meltdown Vulnerability
  - Illegal Memory Load ***still updates $***
  - If illegal load is **squashed**, **process** gets to keep running (no segfault)

```
1. syscall();
2. x = *target_addr;
3. y = probe[x*4096];
4. //another thread executes $ timing attack
   (prime+probe)to learn some bits of x
```

← causes PC+4 etc. to be squashed

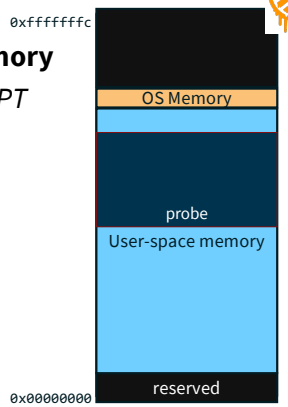
← Executes OoO, value available in bypass network

← dependent load updates $



Meltdown Consequences

- User process can *easily* read **all of OS Memory**
- Solution: *unmap most of OS memory from PT*
  - Syscalls take longer to handle
    - Trap to OS
    - Trap handler loads OS page table, **flushes TLB**
    - Handle trap
    - Loads User page table, **flushes TLB**
    - Return to User
  - 5% overhead most programs
  - 30% for syscall-heavy programs



Spectre in 1 Slide

```
unsigned int a;
if (a < xarray_len) {
  // Should only execute if x[a] is in bounds
      b = x[a];
      z = *b;
}
```

Bounds-check-bypass

- Extremely common check
- *Speculation* allows body to *temporarily execute* when `a >= xarray_len`
- Speculative execution modifies $ state (just like meltdown)
- Attacker can read arbitrary (user space) memory via $ timing channel

software & hardware fixes exist

scary

both leak data through $ timing channel

- Exploits out-of-order execution *after exceptions*
- Illegal memory accesses after an exception still update $
- Breaks *Kernel Isolation*: Allows user process to read any part of OS's memory (if mapped)

- Exploits *speculative execution across branches*
- Attacker manipulates branch predictor to speculatively execute target instructions
- Breaks *software sandboxing*: Allows user process to violate application-level isolation (within a single process)

Miessler Blog (https://danielmiessler.com/blog/simple-explanation-difference-meltdown-spectre/)

Takeaways for Computer Architects

Architecture: timing-independent functional behavior of a computer
Micro-architecture: implementation techniques to ⇧performance

These choices have consequences!

What if a computer that is architecturally correct can leak protected information via its micro-architecture?

Perhaps our definition of "architecturally correct" needs re-thinking…

Some References

New York Times: https://www.nytimes.com/2018/01/03/business/computer-flaws.html

Meltdown paper: https://meltdownattack.com/meltdown.pdf
Spectre paper: https://spectreattack.com/spectre.pdf

A blog separating the two bugs: https://danielmiessler.com/blog/simple-explanation-difference-meltdown-spectre/

Google Blog: https://security.googleblog.com/2018/01/todays-cpu-vulnerability-what-you-need.html and https://googleprojectzero.blogspot.com/2018/01/reading-privileged-memory-with-side.html

Industry News Sources: https://arstechnica.com/gadgets/2018/01/whats-behind-the-intel-design-flaw-forcing-numerous-patches/ and https://www.theregister.co.uk/2018/01/02/intel_cpu_design_flaw/