DATA AND NETWORK SECURITY

Case Study IV: Hardware Vulnerabilities (Meltdown & Spectre)

1

Overview

- Classification: Vulnerability
  - Leverages features of high performance CPUs
  - Ubiquitous
    - Nearly every system is vulnerable to Meltdown, Spectre, or both!
  - Detectability by AV tools
    - low/none
    - Exercises a hardware feature
    - Malicious software looks normal
  - Revealed: Jan 2018
  - Present in every Intel CPU since at least 2010, probably longer

2

Overview

- Meltdown
  - Exploits out-of-order execution
  - Leaking memory through stale cache
  - Leverages (uses) 'side-channels' (unintended information channels)
  - Primarily Intel processors. ARM Cortex-A75 and IBM Power
  - NOT AMD processors
- Spectre
  - Exploits speculative execution
  - Tricking processor to speculatively execute instructions
  - Leverages (uses) 'side-channels' (unintended information channels)
  - Intel processors, Apple, ARM, IBM Power, AMD

3

4

Overview (cont)

- Both
  - Use microarchitectural traces or 'residue' [sub-architectural state that is not 'undone']
  - Use covert side-channels to leak information
  - Can be addressed with software patches
    - Meltdown is easier to fix than Spectre

4

5

Architecture 'features' ('weaknesses')

1. Modern Processors rely on out-of-order execution for speed
2. Branch prediction keeps processor busy by predicting branch direction
   - Allows instructions to 'speculatively' execute.
     - When wrong, changes are discarded
     - When right, changes are committed
3. Also for speed, instructions/data from memory are loaded into cache (very high speed memory) before use
4. Memory layout
   - Most operating systems map kernel (privileged code/data) addresses into user address space
   - Rely on 'mode' bit to catch illegal memory access

5

6

Meltdown - Details

- Combination of the above 'features' permits unprivileged user code to leak all of kernel memory
- Relies on 'features' 1, 3, and 4
- Meltdown 'Steps'
  1. Try to read kernel memory (causes an 'exception')
  2. Uses 'transient instructions' to access cache
  3. Attacker uses a cache Flush&Reload or access timing check operation to determine the accessed 'cache line'
- Accessed 'cache line' is directly related to secret kernel memory

6

7

Meltdown - Example

- Train the 'if' with conditions that execute 'then' part
- Now provide data that will cause an 'exception'

```
If (cond) {
    value = *ptr; // ptr is to kernel memory!
    index = ((value >> bit)&1)*0x100 + 0x200;
    legalvalue = array[index];   // We have access to this data
}
```

- Protection exception happens after array is read
- Now, read values from array [offset 0x100 and 0x300]
- Time the reads. This tells us if the value was in cache
- Tells us if the bit is 1 or 0

7

8

Meltdown - Mitigation (Prevention)

- Mostly preventable
- KAISER – Kernel Address Isolation to have Sidechannels Efficiently Removed
  - Separating the address space mapping blocks reading most of the critical kernel memory
  - Introduced in 2017/2018
  - Reduces performance due to page table changes on mode switch

8

9

Spectre - Background

- Based on mis-predicting branches leading to executing unintended instructions
- Unintended instructions leave private/secret data in cache
- Microarchitectural side effects left in place – Side channel can read this data from cache

9

10

Spectre - Details

- Several Variants
  - Conditional Branch Exploit
  - Indirect Branch Exploit
  - Others
    - Mis-training return instructions
    - Leaking information based on timing
    - Contention for arithmetic units

10

11

Spectre – Conditional Branches

- Attacker 'mistrains' a conditional branch
- Then feeds a value to cause branch to be mispredicted
- Example:

```
if (x < array1_size)
   y = array2[array1[x] * 4096]
```

- Attacker controls x and feeds good values (within array1_size) numerous times
- Attacker now changes x to be out of range
  - 'if' mispredicts and starts accessing array2 data that is out of bounds
  - Information is reverted but cache holds residual info from victim's address space!
  - Attacker can recover data from cache

11

12

Spectre – Indirect Branches

- Attacker finds a 'gadget' (some user code) in victim's address space they want to execute
- Mistrains the 'Branch Target Buffer' (BTB) with an indirect branch to the gadget's address
  - Indirect branch is a branch to an address stored in a register or memory
  - This mis-training can be done inside the attacker's address space
- Victim's process does an indirect branch and mispredicts causing 'speculative' execution of the 'gadget'
- Residue or Traces from speculative code left in cache to be read by attacker

12

13

Mitigation of Spectre

- Most options are bad:
  - Speculation
    - Turn off speculation (future hw design)
      - This will result in severe performance degradation
    - Serializing/Speculation blocking – Use instructions that force ordering of instructions (i.e., lfence)
      - Place lfence on each of the resulting code paths
      - This would also degrade performance (disables 'branch prediction')
  - Protecting Secret Data
    - Using masks for indices into arrays (limits how far out of bounds a reference can be)
    - Using 'poisoned' pointers – Pointers must be combined (XOR) with a 'poison' value that is 'random'.
  - Limiting Data Extraction from Covert channels
  - Preventing Branch Poisoning

13

14

Conclusions

- Spectre at least will plague us for a while
- Meltdown can be mitigated pretty easily

14

15

Sources

- https://meltdownattack.com/
- https://research.cs.wisc.edu/multifacet/papers/hill_mark_wisconsin_meltdown_spectre.pdf
- https://lwn.net/Articles/738975/
- https://spectrum.ieee.org/tech-talk/semiconductors/processors/how-the-intel-processor-meltdown-vulnerability-was-thwarted
- https://www.techarp.com/guides/complete-meltdown-spectre-cpu-list/
- https://spectrum.ieee.org/computing/hardware/how-the-spectre-and-meltdown-hacks-really-worked
- https://www.usenix.org/sites/default/files/conference/protected-files/lisa18_slides_masters.pdf

15