Spectre and Meltdown

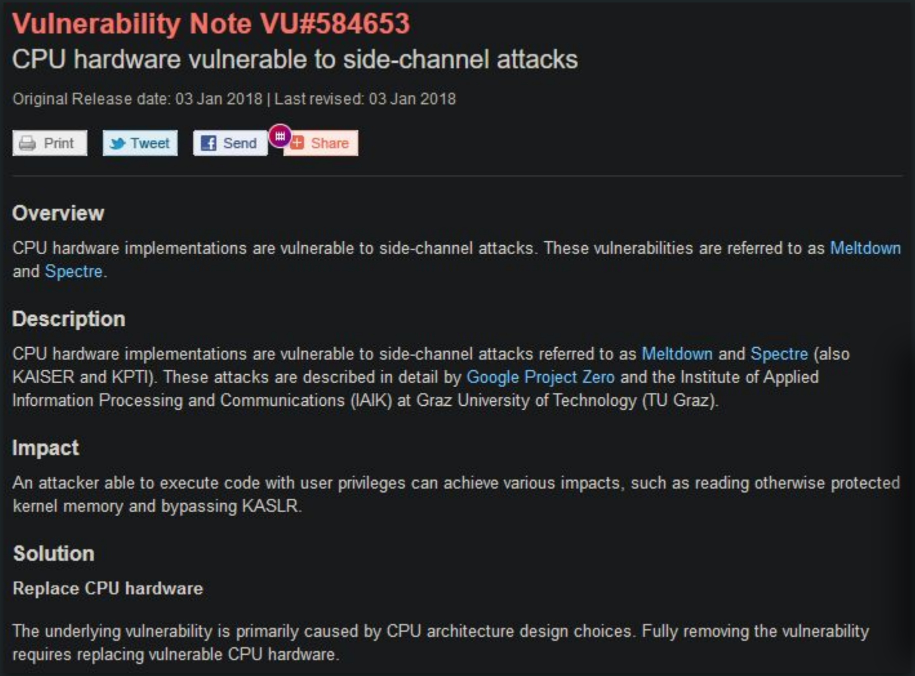
By Max Weimer

Overview

- Two distinct attacks
- Spectre
  - User ↔ User memory breakdown
  - Sandbox breakout
  - Less serious, harder to mitigate
- Meltdown
  - User ↔ Kernel memory breakdown
  - Access to kernel memory
  - More serious, easy to mitigate

Severity

- BAD



Vulnerability Note VU#584653

CPU hardware vulnerable to side-channel attacks

Original Release date: 03 Jan 2018 | Last revised: 03 Jan 2018

Print Tweet Send Share

Overview

CPU hardware implementations are vulnerable to side-channel attacks. These vulnerabilities are referred to as Meltdown and Spectre.

Description

CPU hardware implementations are vulnerable to side-channel attacks referred to as Meltdown and Spectre (also KAISER and KPTI). These attacks are described in detail by Google Project Zero and the Institute of Applied Information Processing and Communications (IAIK) at Graz University of Technology (TU Graz).

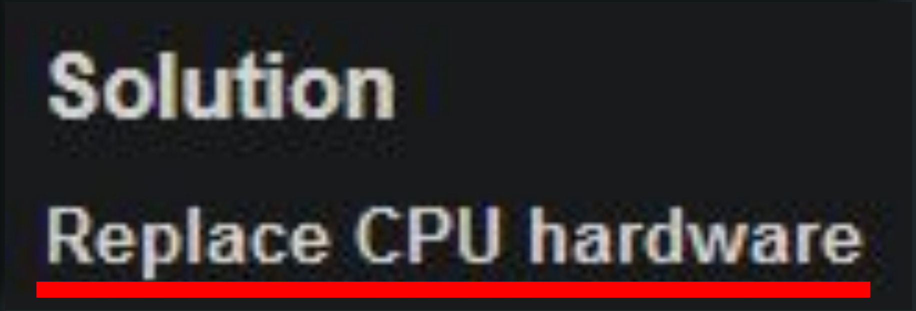
Impact

An attacker able to execute code with user privileges can achieve various impacts, such as reading otherwise protected kernel memory and bypassing KASLR.

Solution

Replace CPU hardware

The underlying vulnerability is primarily caused by CPU architecture design choices. Fully removing the vulnerability requires replacing vulnerable CPU hardware.



Background: CPU Pipelining

How do we execute an instruction?

Fetch	→	Decode	→	Execute
What are we running?	→	How do we run it?	→	Run it.

Optimize?

`mov ax, 5`	Fetch	→ Decode	→ Execute		
`mov bx, 6`		Fetch	→ Decode	→ Execute	
`add ax, bx`			Fetch	→ Decode	→ Execute

Spectre: Branch Prediction

```
int read(int i) {
    int result = -1;
    if (i < array_size)
        result = array[i];

    return result;
}
```

- Resolving condition takes time
- Where is array_size in memory?
  - Uncached is beneficial
- Are we going to run the code in the branch?
  - Do we wait for the condition?
  - Do we guess?

Of course we guess.

Spectre: Branch Prediction

```
int read(int i) {
    int result = -1;
    if (i < array_size)
        result = array[i];

    return result;
}
```

- What's doing the predicting?
  - Branch predictor in hardware
- Evil mistraining
- What if we're wrong?
  - Rewind state
  - *Can't rewind cache*

Why does it matter?

We evaluate cache, read secrets

Break out of sandbox

Spectre: Mitigations

- No fix-all solution
- Javascript engines have been updated
  - Chrome uses individual processes to bolster sandboxing
  - Webkit
    - Index masking to prevent the bounds from getting *too* out of hand
    - Pointers are XORed with a pseudo random “poison” value, poisoned pointers can’t be used to read arbitrary memory unless the poison is known

Meltdown Background: µOPs & OOE

- What does decode do?
- Breaks down a command like add into micro operations (µOPs)
- Ex: add ax, bx breaks down to the following µOPs roughly:
  1. Put contents of ax on bus, read bus into ALU input a
  2. Put contents of bx on bus, read bus into ALU input b
  3. Put ALU output on system bus, read bus into acc
- Independent operations may be executed in a different order than written
  - Exception handling gets muddy
  - Worse yet, µOPs may run out of order

KEY: Errors are handled at the **end** of the pipeline

Meltdown Background: Out of Order Execution

```
; rcx = kernel address, rbx = probe array
xor rax, rax
retry:
  mov al, byte [rcx]         ; yes officer, this line right here
  shl rax, 0xC
jz retry                     ; Still alive?  Go again
mov rbx, qword [rbx + rax]   ; OOE: quick throw data somewhere!
```

- Meltdown causes an exception on purpose by attempting to read kernel memory
  - Kernel memory is read into register before exception is raised
  - With dying breath the program passes the register value out through a side-channel, run out of order
  - Rinse, repeat, dump kernel memory out at ~500KB/s

Meltdown Mitigation: Kernel PTI

- For performance, modern OSes map kernel memory into user space
  - I/O calls, etc are faster
- Solution? Kernel Page Table Isolation (KPTI)
  - Don’t map kernel memory into user space *on affected CPUs*
    - Speed decrease, but only really viable method
    - ~10% performance hit on Intel CPUs
      - Workload dependent

Fin