Meltdown Note

KyoKeun Park

Background: Out-of-Order Execution

In most (if not all) modern CPUs, there is an optimization called "out-of-order execution" implemented in order to run multiple lines of code. This is done by reading the instructions ahead of time to see which instructions can be executed alongside current instruction.

For example, take this snippet of code for example:

```
int a = 3;
int b = 4;
int c = 5;
int d = 6;
int e = a + b;
```

Although `int d = 6;` instruction is ahead of `int e = a + b;`, it is very possible that the latter will start executing before the former. This is because `int e = a + b;` requires the use of ALU while `int d = 6;` strictly just requires registers to complete. In fact, ALU will not be in use other than the last instruction.

This is where out-of-order execution comes in. This feature allows the program to execute an instruction like `int e = a + b;` earlier in this case in order to utilize the processor as much as possible.

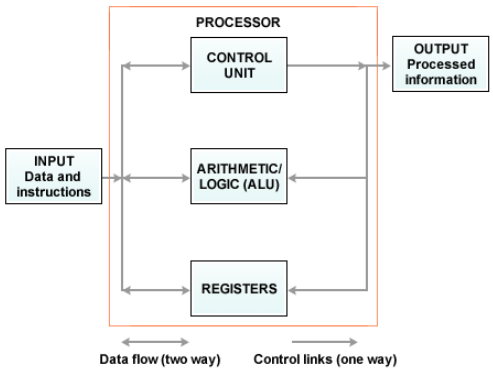

Figure 1: Recall, CPU has different units for different tasks

As useful as this is, out-of-order execution is the backbone of the exploits, *Meltdown*.

Intel Processors and Out-of-Order Execution

Just like other modern processors, Intel processors also have out-of-order execution implemented in all of their modern processors. However, unlike other processors (ie. AMD), there's a huge flaw in the way they use out-of-order execution.

Note that memory is separated into kernel space and user space. This is to isolate the sensitive data. However, when Intel processors go through out-of-order execution, it does not check for address space privilage. For the most part, this is not a big problem, since at some point, OS realizes that the program is attempting to access a memory that it shouldn't be accessing, hence sends a signal (SIGSEGV) and discards the instruction.

The problem arises from the fact that, although the malicious instruction gets discarded, (hence instruction gets reverted) cache within the CPU is left untouched. This means that, whatever the malicious instruction has done through out-of-order execution stays in the cache.

How Does Meltdown Work?

Meltdown exploits this flaw of Intel process with its interaction with out-of-order execution. It follows these steps:

1. Use out-of-order execution in order to execute an instruction that accesses protected address space indirectly.
2. Handle or suppress exception (SIGSEGV) so that the program does not crash when the signal is sent.
3. Knowing that the content of the protected address space is (indirectly) stored in cache, use that knowledge to "guess" what the content is by measuring the time of the memory access

Example

Consider an address space of size 4000 bytes, where first 1000 bytes is a kernel space and the rest (1000-4000 bytes) is user space. Let's say at memory location 100, there's a value "10".

Now consider this snippet of code:

```
int a = 2000;
int b = 100;
int c = 50;
int d = 10;
int e = read(a + read(b));
```

`read(x)` in this case is a function to read a memory at position `x`. The line `int e = read(a + read(b));` will go through out-of-order execution and store value that is located in memory 2010 before it stops the instructions and then sends SIGSEGV signal to indicate segmentation fault.

Like described above, let's the say program handles the exception, thus continuing to run after SIGSEGV. We now know that the value in memory location "2000 + x" is cached, where "x" is the value that is stored in location 100.

So now, we can read the memory starting from location 2000 and time each read. Once we get to location 2010, the read speed will be a lot faster than previous reads, since the value of 2010 was stored in the cache already. Now we know that the value that was stored in location 100 was 2010 - 2000, which is 10.

We can repeat this process until we read the entire memory if we wanted to.

Prevention

When this exploit was first publicly announced, Intel was quick to look into the problem and send out a patch. Microsoft and Apple has released a patch for their operating systems (Windows and MacOSX) not long after. Linux also released patches for both long-term support kernel and their latest kernel. Hence, the best prevention against meltdown will be to update the OS/kernel to the latest version.

KASLR

Even before the patches, Linux kernels had a feature called kernel address space layout randomization (KASLR). This feature randomizes where kernel space resides within the physical memory on every boot. This makes meltdown exploit a lot harder on Linux kernels with KASLR, since the program will need to try to find out where the kernel space is even before it can pull the exploit.