Author: John Sully
Date: 1/9/2018

Spectre and Meltdown

The first week of the new year got off to a fast start with a pair of security vulnerabilities which affect just about every computer built in the last 20 years. The reason for this broad reach is that unlike a run of the mill virus or malware vulnerability, which attacks a weakness in software, Spectre and Meltdown attack fundamental features of modern CPU and operating system architecture. Spectre exploits a feature called speculative execution, which allows a CPU to execute instructions which might not be needed but which recent experience says will be; Meltdown exploits a feature called out of order execution which allows instructions which do not have a dependency on a previous instruction to execute speculatively. Spectre affects all processor architectures, while Meltdown affects only Intel architectures.

These attacks both allow the attacker to read memory which it does not have access to, such as kernel memory and memory belonging to other processes. But it is important to realize that the damage done by these attacks is limited solely to the expropriation of secrets such as a passwords, cryptographic keys and credit card numbers which are stored in memory. These attacks cannot be used to destroy or alter data.

Recommendations

Although Spectre and Meltdown exploit a subtle flaw in modern processor architectures, it is possible to mitigate the harm caused through software patches.

Microsoft released an out of band patch to Windows 10 over the weekend to address the Meltdown vulnerability, and UIT is currently evaluating how to respond. Microsoft is also releasing patches for Windows 7 and 8 but will not be releasing patches for earlier versions. Apple has released a patch to macOS (version 10.13.2) and iOS (version 11.2), but earlier versions of macOS do not have patches available. Most versions of Linux have patches available.

There is some performance impact associated with the mitigation technique, which is similar for all operating systems, but it appears to be generally minor, measuring less than 10% on most benchmarks and in many cases less than 1%. In some cases the impact can be severe – the Postgres database server has been observed to suffer up to a 30% performance penalty under some workloads.

Spectre, because of a large number of caveats about conditions which must hold true for it to be successful, is a considerably less effective attack. It can, however be easily deployed to read memory outside of a browser page's "sandbox". All major browsers have released versions which have patches intended to mitigate the effects of Spectre and you should update your browser to the latest version if you haven't done so recently. Additionally, you might want to try using an ad blocker since advertisements on the web can carry a payload consisting of this bit of malware.

As always, the best advice for combatting Spectre and Meltdown is the same as usual: keep your system's software up to date and use a good anti-virus.

Meltdown Details

Consider a code sequence like the following where rcx contains an address in the kernel (which a normal process is not allowed to access) and rbx contains an array address in the calling process' address space (which is legal for the process to access)

```
1                                   ; rcx = kernel address
2                                   ; rbx = probe_array
3 retry:
4 mov al, byte [rcx]                ; make illegal access
5 shl rax, 0xc                      ; multiply (illegal) value by 4096
6 jz retry
7 mov rbx, qword [rbx + rax]        ; read (legal) memory address based
8                                   ; on (illegal) value
```

The instruction at line 4 attempts to read at an illegal address, but the other instructions in the sequence execute speculatively as soon as the value in register al is available. The illegal access causes an exception, which is not raised until the mov instruction at line 4 is retired. Once the exception is raised the results of the instructions on lines 5, 6 and 7 are discarded so, in theory, it is impossible for the attacker to know the value which was illegally read at line 4. Line 7 has a side effect, however. When the read at line 7 is executed, in addition to reading a value from memory it also creates an entry in the cache at an address determined by the value which was read at line 4 and multiplied by 4096 at line 5.

Because the value at this address was cached, a subsequent read at this address will be much faster than reads at surrounding addresses. Since all modern Intel processors have high resolution performance counters it is possible to time how long an access takes, a fast access indicates a cache hit and therefore the illegally read value can be inferred from the offset from the base of the array accessed in line 7. So a bit of code like this:

```
for (i = 0; i < 256; i++)
       x = probe_array[i * 4096];
```

will give you the value read when the speed of the array read is fast compared to other accesses into the array, with *i* being the value which was read at line 4. Since the operating system typically provides a fixed map of all physical memory starting at a specific virtual address, by iterating over each address in this map it is possible to read all physical memory. It should be obvious the problems this would present to cloud vendors, although this also is problematic for desktops.

Spectre Details

Spectre uses a technique similar to Meltdown to communicate ill-gotten gains, but is somewhat more limited in what it can read. Spectre works by exploiting a feature called branch prediction. Branch prediction takes advantage of the fact that programs tend to contain a lot of loops which will usually take a main course of execution (execute the body of the loop) and only comparatively rarely will take the alternative course (terminate the loop). Once the branch prediction engine in the processor learns what course of action a program usually takes at a branch point it will execute the code path normally taken on the assumption that is what the program will do. So a bit of code like this:

```
if (x < probe_array_size)         // bounds check
       y = probe_array[x];        // make possibly illegal access
```

when called repeatedly with valid values of x will train the branch prediction engine to take the branch. When this code is called with an illegal value of x, the access will proceed speculatively and then

techniques similar to Meltdown can be used to cache a value at an address dependent on the illegally read value. It is interesting to note that Spectre has a proof of concept in JavaScript.