Meltdown and Spectre Hack

Recent press reports (January 2018) talk about a newly discovered form of security threat that involves attackers exploiting common features of modern microprocessors (aka chips) that power our computers, tablets, smartphones, and other gadgets. These attacks, known as “Meltdown” and “Spectre”, are getting a lot of attention. People are (rightly) concerned, and it’s of course very important to apply all of the necessary software updates that have been carefully produced and made available. Technology leaders are working together to address these exploits and minimize the risk of potential attacks.

To get going, let’s understand a bit about “speculative execution” by looking at an everyday analogy. Suppose a regular customer visits the same coffee shop and orders the same caffeinated beverage every morning. Over time, the customer gets to know the baristas, who become familiar with the customer’s order. Seeking to offer good service (and save their valued customer some time standing in line) the baristas eventually decide to begin preparing the customer’s order when they wave at them as they enter through the front door. But one day, the customer changes their order. Now the barista has to throw away the previously prepared coffee and make a new one while the customer waits.

Taking the analogy one step further, suppose the baristas know the customer’s name, and they like to write that name using a permanent marker on their cup. When they speculatively prepare the usual beverage, they write the customer’s name on the cup. If the customer comes in with a different order, the speculated cup is thrown away along with its contents. But in so doing, the cup’s personally identifiable information is briefly visible to anyone watching. This coffee shop scenario involves speculation. The staff doesn’t know for sure when the customer comes in that they’re going to order a latte or an Americano, but they know from historical data what the customer usually orders and they make an educated guess to save the customer waiting. Similar speculation happens throughout our everyday lives because such guesses often turn out to be true, and we can get more done in the same amount of time as a result. It’s like this with our computers. They use a technique known as “speculative execution” to perform certain processing operations before it is known for certain that those operations will be required, on the premise that these guesses often turn out to save time.

In the case of computers, speculative execution is used to decide what to do when confronted by a test like *“if A, do this; otherwise, do that”*. We call these tests conditions, and the code that executes as a result is part of what we term a conditional branch. A branch just means a section of the program that we choose to run in response to whatever the result of the condition turns out to be. Modern computer chips have sophisticated “branch predictors” that use fancy algorithms to determine what the result of the conditional test is likely to be while that test is still being calculated. In the interim, they speculatively execute code in the branch that seems to be most likely to run. If the guess turns out to be right, the chip appears to run faster than waiting for the test to complete. If the guess is wrong, the chip has to throw away any speculative results and run the other branch. Branch predictors are often over 99% accurate at guessing.

As you can see, the potential performance benefit from a chip speculatively executing the correct branch of code is significant. Indeed, speculative execution is one of the many optimizations that have helped to dramatically speed up our computers over the past couple

of decades. When implemented correctly, the resulting performance benefit is substantial. The source of the newly discovered problems come from the chip design attempts to further optimize by assuming that speculation process is a black box that is completely invisible to outside observers (or bad guys).

Conventional industry wisdom was that whatever happened during the process of speculation (known as a “speculative execution window”) was either later confirmed and the results were used by the program, or it was not used and completely discarded. But it turns out that there are ways attackers can view what happened within the speculation window and manipulate the system as a result. An attacker can also steer the behavior of branch predictors to cause certain code sequences to run speculatively that should never normally have been executed. We expect these vulnerabilities and other similar flaws which could exploit speculative execution to lead to fundamental changes in the way that future chips are designed so that we can have speculative execution without security risks.

Let’s dive a bit deeper into the attacks, starting with Meltdown (variant 3) which received a lot of attention because of its broad impact. In this form of attack, the chip is fooled into loading secured data during a speculation window in such a way that it can later be viewed by an unauthorized attacker. The attack relies upon a commonly-used, industry-wide practice that separates loading in-memory data from the process of checking permissions. Again, the industry’s conventional wisdom operated under the assumption that the entire speculative execution process was invisible, so separating these pieces wasn’t seen as a risk. In Meltdown, a carefully crafted branch of code first arranges to execute some attack code speculatively. This code loads some secure data to which the program doesn’t ordinarily have access. Because it’s happening speculatively, the permission check on that access will happen in parallel (and not fail until the end of the speculation window), and as a consequence special internal chip memory known as a cache becomes loaded with the privileged data. Then, a carefully constructed code sequence is used to perform other memory operations based upon the value of the privileged data.

While the normally observable results of these operations aren’t visible following the speculation (which ultimately is discarded), a technique known as cache side-channel analysis can be used to determine the value of the secure data. Mitigating Meltdown involves changing how memory is managed between application software and the operating system. We introduce a new technology, known as KPTI (Kernel Page Table Isolation), which separates memory such that secure data cannot be loaded into the chip’s internal caches while running user code. Taking extra steps every time application software asks the operating system to do something on its behalf (we call these “system calls”) results in a performance hit. The degree of performance hit varies roughly in line with how frequently an application needs to use such operating system services.

The Spectre attack has two parts. The first (variant 1) has to do with “bounds check” violation. Once again, when speculatively executing code, the chip might load some data that is later used to locate a second piece of data. As part of a performance optimization, the chip might attempt to speculatively load the second piece of data before it has validated that the first is within a defined range of values. If this happens, it is possible to arrange for code to execute speculatively and read data it should not into the system caches, from where it can be extracted using a side-channel attack similar to the one discussed before.

Mitigating the first part of Spectre involves adding what we call “load fences” throughout the kernel. They prevent the speculation hardware from attempting to perform a second load based upon a first load. These require small, trivial, and not particularly performance-impacting changes throughout the kernel source. Our toolchain team has developed some tooling and worked with others to help determine where these load fences should be located.

The second part of Spectre (variant 2) is in some ways the most interesting. It has to do with “training” the branch predictor hardware to favor speculatively executing pieces of code over those it should be executing. A common hardware optimization is to base the behavior of a given branch choice upon the location in memory of the branch code itself. Unfortunately, the way in which this memory location is stored isn’t unique between an application and the operating system kernel. This allows for the predictor to be trained to speculatively run whatever code the attacker would like. By carefully choosing a “gadget” (existing code in the kernel that has access to privileged data) the attacker can load sensitive data in the chip caches, where the same kind of side-channel attack once again serves to extract it.

One of the biggest problems posed by this second part of Spectre is its potential to exploit the boundary between the operating system kernel and a hypervisor, or between different virtual machines running on the same underlying hardware. The branch predictor can be trained by one virtual machine to cause privileged code in the hypervisor (or another virtual machine instance) to access trusted hypervisor data which can be extracted using a side channel. This poses a significant risk to private and public cloud environments running unpatched servers.

Mitigating this second part of Spectre requires that the operating system (selectively) disable branch prediction hardware whenever a program requests operating system (system call) or hypervisor services, so that any attempt by malicious code to train the predictor won’t carry over into the operating system kernel, the hypervisor, or between untrusted virtual machines running on the same server. This approach works well, but it comes at a performance penalty that is not insignificant.

Many OS manufacturers' patches will default to implementing the security change and accepting the performance impact.

Taken from:

January 5, 2018

Jon Masters, chief ARM architect, Red Hat