Spectre and Meltdown

A Look into Mitigation and Detection

Caroline Hickey
University of Virginia
Charlottesville, VA
ch6yg@virginia.edu

ABSTRACT

Throughout the years, companies have been working to increase processor speed and performance. Through the implementation of out-of-order execution, optimal performance was achieved, but at a cost. This optimization led to security holes. In January of 2018, the Meltdown and Spectre vulnerabilities were published to the public. These vulnerabilities exploit out of-order execution to read kernel and user memory and access sensitive data. Because the vulnerabilities only exploit hardware issues, they affect almost all processor and therefore almost all computer systems. As the future of cloud progresses, these vulnerabilities cause significant fear for the future of the cloud as attackers could gain access to not just one users' memory, but any user using the same resources. So far, the only solution to these vulnerabilities is to disable out-of-order execution. This would significantly slow CPU speed by up to 30%, making it an unfeasible solution. It is also difficult to provide software patches as it is a hardware issue and not a software issue. Combining mitigation and detection methods would be the only way to less the chances on an attack. Even these methods would slow the performance of CPUs. Any solution to these dangerous vulnerabilities will have a trade-off between performance and security. The discovery of Spectre and Meltdown will change the way that new modern processors are designed and optimized.

ACM Reference format:

Caroline Hickey. 2021. Spectre and Meltdown: A Look into Mitigation and Detection.

1 Background Knowledge

1.1 Out-of-Order Execution

In order for processors to achieve optimal runtimes, many processors use optimization techniques. Instead of executing instructions linearly, instructions are executed as resources become available to execute them. This makes it possible for the CPU to run instructions in parallel, significantly increasing performance of machines by cutting out the time that instructions would sit idle. While this is incredibly helpful for increasing performance, it leads the CPU to execute instructions that might not be needed. This is known as speculative execution.

1.2 Speculative Execution

Speculative execution is another optimization technique used by many processors in order to increase performance. In cases of branches and conditional statements, the processor predicts which instructions will be executed and executes them before knowing which instructions should actually be executed. This allows the processor to eliminate the time those instructions sit idle waiting for the results of the branch [6]. If the processor is wrong, the CPU reverts back to the state before the wrong instruction was executed and continues with no problem. This instruction that was wrongly executed is referred to as a transient instruction. While most changes made by the CPU are reset, the cache still holds values used by the wrongly executed instruction, as these are not reset after a transient instruction. This trace of a mis-executed instruction is what causes vulnerabilities [5].

1.3 Cache-based side channel attacks

Cache-based side channel attacks aim at deriving secrets from a system through the measurement of observable behavior of a process. If an attacker and victim are using the same resources, attackers can exploit cache timing to retrieve sensitive information. This information might be cryptographic keys, details of executed instructions, or passwords.

For example, an attacker can retrieve an AES secret encryption key using cache-based side channel attack [8]. During encryption, entries from the AES T-table, at an index based on a calculation between part of the plain text and a bit from the secret key, are cached. Subsequent accesses of the T-table reveal information about the secret which done through cache access timing. In order to find the index of T-table to cache, parts of the plain text, p_n, is xored with a bit

of secret key, k_n. A cache hit based on a different (p_* xor k_*) would show that (p_n xor k_n) = (p_* xor k_*), meaning (p_n xor p_*) = (k_n xor k_*). The attacker is able to gain information about this secret key just from observing the cache [8].

Since loading something from main memory takes significantly more time than loading something from cache, attackers can observe the time taken to retrieve data from a given memory address. Based on the time taken, attackers can observe whether or not a memory address is in the cache. There are many variants of this attack, but on common example of this attack is the FLUSH+RELOAD [7] attack. Memory lines are flushed from all levels of the cache using non-privileged instructions such as cflush in x-86 architecture [5]. The attacker than waits for the victim to execute some program which will load memory into the cache. Next, the attacker reloads the cache line and observes the time taken to retrieve the data for the memory address. If the data is accessed quickly, the memory address was used, and therefore cached, by the victim. Otherwise, the attacker knows that the memory was loaded from main memory and was not used by the victim. The attacker repeats the flush and reload for the cache lines that were cached in the attacks [3]. If desired, the attacker can iterate over all addresses and dump the entire memory.

2 Background of Spectre and Meltdown

In January of 2018, the team from Google's Project Zero published their findings that a hardware vulnerability existed in many processors that could allow attackers to read kernel memory, even from an unprivileged user. These vulnerabilities became known as Spectre and Meltdown [1]. They showed that due to CPUs' use of speculative execution allowed attackers to access unauthorized cached memory and preform cache-based side channel attacks. This is due to the fact that after a transient instruction is executed, any changes, besides changes made to the cache, are reverted, allowing attackers to access this cached memory. This meant that attackers could gain access to passwords, encryption keys, and other sensitive data without leaving evidence in system logs making it practically untraceable. Not only do these attacks affect individual computing systems, it also affects cloud computing allowing attackers to retrieve data not just form the user, but also data from any other users utilizing the same CPU. Both Meltdown and Spectre do not exploit any software issues, making it extremely difficult to patch [6].

3 Meltdown attack

Meltdown attacks occur in two steps. First, memory isolation is bypassed by executing an instruction. Second, meltdown preforms a cache-based side channel attack, such as Flush + Reload, to retrieve kernel memory [5].
In the first step, the attacker chooses an unauthorized memory location to load into a register. Next, an out-of-order instruction is executed causing the memory that is inaccessible to the attacker to be loaded into memory. Soon, the processor realizes it has made a mistake and throws an exception causing a segmentation or page fault. Before this, however, Meltdown executes step two, a cache-based side channel attack.

```
1 ; rcx = kernel address, rbx = probe array
2 xor rax, rax
3 retry:
4 mov al, byte [rcx]
5 shl rax, 0xc
6 jz retry
7 mov rbx, qword [rbx + rax]
```

Listing 1: Example of a Meltdown attack process [6]

3.1 Step 1 of Meltdown

The attacker first chooses a memory location that is inaccessible to the user in the victim's address space. The attacker accesses this memory location by indexing outside of some probe array and into the victim's address space, which is inaccessible to the attacker. Because access to kernel memory must be checked via a permission bit, the entire kernel is mapped into the virtual address space. This means that the CPU is able to find the valid physical address of the kernel space and access its contents, only if the user is given permission to do so. Else, it will raise an exception and stop a user from accessing the kernel memory. Next the attacker executes a sequence of instructions such as Listing 1. The instruction must cause some exception to be thrown. In this example, the processor will raise an exception because the attacker is attempting to access memory outside the bounds of the probe array and into unauthorized memory. Due to out-of-order execution, the instruction will be executed and the kernel memory will be cached without checking permissions in the attacker's cache. The CPU will then check the permission bit and throw and exception. Before this, however, the Meltdown attack is executed [5].

3.2 Step 2 of Meltdown

In step 2, Meltdown attackers perform the FLUSH+RELOAD [7] attack in order to retrieve the desired sensitive information [5]. Before running the

malicious program, the attacker flushes its entire cache. After the malicious program is run, the attacker’s cache now holds the value of some secret information from the victim’s process, whether it is a part of a password or cryptographic key. The attacker attempts to access each value of its probe array in its cache and observes the time taken to retrieve the value. If the time to retrieve information from cache was quick, the value at that cache position is the value cached from the speculative execution. Based on the index of the probe array that resulted in the cache hit, attackers can leak the value in the cache.

3.3 Execution handling/suppression

In Meltdown attacks, an exception is always thrown, causing the process to terminate. This causes a race condition for an attacker to leak the data before the program is terminated. Often, exceptions are handled differently or suppressed. One way to get around this race condition is to fork the application before accessing the invalid memory location and only do the access in the child process [5]. This will cause the child process to terminate, but not the parent. The attacker can then recover the information through a side-channel attack. Another way is to install a signal handler to prevent the process from terminating upon an exception.

4 Spectre attack

Spectre also occurs in two steps. First, the attacker mistrains the branch predictor to execute unprivileged instructions using speculative execution. Second, similar to Meltdown, Spectre preforms a cache-based side channel attack, such as Flush + Reload [7], to retrieve kernel memory [6]. Unlike Meltdown, Spectre does not take advantage of speculative execution to bypass processor-level permissions checks. Additionally, Spectre attacks do not cause any exceptions. There are a few ways Spectre executes step one. We will focus on variant 1 and 2.

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

Listing 2: An example of a Spectre attack process [6]

4.1 Variant 1

Variant 1 of Spectre, also known as Bounds Check Bypass, exploits conditional branch misprediction. First, the attacker mistrains the branch predictor to incorrectly execute the unprivileged instructions. This is done by running the program with correct inputs so that the branch is always taken. The branch predictor is now trained to always take the conditional branch. Some program with a bounds check is chosen to run. The attacker then chooses some maliciously chosen index value, x, such as in listing 2, that leads to an out-of-bounds failure [6]. Normally, the x would be chosen to index into some value in an array. In the attack however, the x is chosen so that it indexes out of the attacker’s address space and into the victim’s address space. Next, because the branch predictor is mistrained, when the program is executed, the branch instruction is executed as taken. Instead of waiting for the results of the conditional check, the processor assumes that the following instructions should be executed and speculatively executes them. The instruction inside the conditional statement, like the code shown in Listing 2, will then preform some memory access dependent on x under the assumption that the bounds check will return as valid. This memory access will load the data based on x, which would be the victim’s memory, into the cache. When the CPU finally realizes the branch was mispredicted it will revert any changes made and continue executing the rest of the instructions, throwing no exception. The data, however, is still cached. The attacker can then preform a FLUSH+RELOAD [7] attack to find the value of sensitive data and leak the memory via a covert channel [6], in the same way as Meltdown attacks as described in section 3.2.

4.2 Variant 2

Variant 2 of Spectre exploits the misprediction of targets of indirect branches. Like variant 1, the attacker mistrains the CPU to later execute the wrong instructions. In variant 2, the attacker does this through mistraining the Branch Target Buffer [6]. To do this, the attacker locates the virtual address of the user’s data in the victim’s address space. It then preforms multiple indirect branch instructions to this address, which trains the BTB to always take the indirect branch. In the attack, the CPU mispredicts a branch to the victim’s address space from an indirect branch instruction which executes instructions in the victim’s address space. This is done by attempting to index into the victim’s address space by going beyond the bounds of some probe array, which is not allowed. The CPU will soon realize its misprediction and will revert any changes made. Changes in the cache, however, are not reverted. The attacker can then access this memory through a cache side channel, as described in section 3.2.

Both variant 1 and variant 2 take advantage of branch misprediction to access unauthorized memory.

5 Implicated Systems

The implications of the Spectre and Meltdown vulnerabilities are far reaching. The Meltdown vulnerability affects all Intel and some ARM processors. Due to architectural differences in AMD and many ARM processors versus Intel processors, they were not affected by the Meltdown attacks [3]. It is believed that the privileges for load are checked differently, and thus the users is unable to load kernel memory without checking the kernel permission bit. The Spectre vulnerability affects all Intel, ARM, AMD, Apple, Fujitsu, IBM, Nvidia, and Qualcomm processors [3]. This is effectively every computer system including laptops, smartphones, and other devices. This means that almost any computer system, including the cloud, could be the target of one these attacks.

6 Solutions

The solution to fixing the Spectre and Meltdown vulnerabilities is not as straight forward as it might seem. For many past vulnerabilities, software patches were sufficient to fix the issue. The Spectre and Meltdown vulnerabilities are hardware vulnerabilities, so updates to their software is not a full solution. There is one way to completely close the Spectre and Meltdown vulnerabilities. To do this, companies would have to make processors that don't utilize out-of-order execution [6]. While it seems like a simple solution, in-order execution would slow down processor performance by up to 30% [1]. As we live in a world that highly values performance, this is not a feasible solution. Additionally, there are millions of systems already out there that cannot be fixed with this solution as their hardware cannot be updated. The solution to these vulnerabilities will have to be a trade-off between performance and security.

Another solution similar to disabling out-of-order execution would be to have the option to disable out-of-order execution while running certain processes [6]. This would allow the user to protect their sensitive data, while still keeping optimal performance speeds during regular execution.

There are less-extreme solutions to fixing Spectre and Meltdown. On such solution proposed to make speculation invisible in the data cache hierarchy [4]. This would be done by creating a new special buffer called the Speculative Buffer [4]. Data from unsafe speculative loads would be loaded into this new buffer instead of the cache. This blocks attackers from leaking information through cache side channels. At a certain point after the instruction is determined to be the correct instruction, the loaded data will become visible. This method, like many other solutions to Spectre and Meltdown, causes overhead which decreases performance. It is significantly better than removing out-of-order execution all together, making it a more feasible solution.

7 Mitigation Techniques

Meltdown is protected in some cases by KASLR, kernel address space layout randomization [5]. This feature randomizes the location of the kernel on each boot making it more difficult to guess the location of the kernel. Attackers would need the value of a randomized offset in order to find kernel memory. It was originally created to protect against side-channel attacks and inadvertently protects against Meltdown attacks. While it protects most computing systems against Meltdown, it has been proven that Meltdown attacks can still occur even with KASLR. Since the offset randomization is limited to 40 bits, any machine with 8GB of RAM is able to test for addresses in 8GB steps [5]. Attackers are able to successfully find the kernel space with at most 128 tests. Nonetheless, it is advised for everyone to enable the KALSR protection on their computing systems.

Another mitigation against Meltdown is kernel page-table isolation, or KPTI [3]. Prior to this patch, page tables were shared between user and kernel space. With this patch, page tables are divided into two sets of tables, one for kernel space and one for user space. This prevents users from accessing kernel address space, which would not allow Meltdown attackers to access kernel memory.

In order to mitigate Spectre attacks, a new mechanism, called retpoline, was introduced by Google [6]. This mechanism prevents the poisoning of indirect branches, which is the method used to executed variant 2 of Spectre attacks, as described in section 4.2. A retpoline causes indirect branches to be isolated from speculative execution by replacing indirect branches with a sequence of instructions that causes an infinite loop catching any speculative execution [3]. This occurs by first calling some function that replaces the return address with the desired address to jump to, rather than the line following the call instruction, and returns to the intended address. In the case of any speculative execution, the processor would speculatively execute to return to the line after the call function. These series of instructions form an infinite loop, which the processor will be stuck in until it realizes it returned to the wrong address. A retpoline prevents the Spectre attack by ensuring speculative execution does not occur in indirect jumps.

8 Detection Techniques

Meltdown and Spectre leave almost no trace behind after attacks. During attacks, however, researchers noticed anomalies as compared to a benign process. In Meltdown attacks, unprivileged memory lookups lead to an unusually high number of segmentation faults or page faults [2]. In addition, due to the FLUSH+RELOAD [7] attacks constantly flushing and re-accessing the cached data, Meltdown attacks cause a high rate of cache misses. Similarly, Spectre attacks also utilize FLUSH+RELOAD [7] attacks so malignant processes produce a high cache miss rate. Cache miss rate is significant in both attacks, but what is also significant to look at is the cache miss rate in relation to the total number of executed instructions [2]. Both Spectre and Meltdown attacks are performed by running relatively small programs. Small programs that produce a high cache miss rate is indicative of these attacks. Unlike Meltdown, Spectre doesn’t produce any exceptions or page faults. Spectre takes advantage of branch mispredictions to load unauthorized memory into the cache. Because of this, Spectre attacks may produce up to 90% more branch mispredictions than a benign program [2].

Some researchers have begun utilizing this information in order to produce real-time detection tools for Spectre and Meltdown. While a fix to these vulnerabilities is necessary, there is no current one that closes both vulnerabilities. Being able to detect if there is a Spectre or Meltdown attack in real-time will allow CPUs to stop attacks before any sensitive data is accessed. One way to do this is to use machine learning to detect Spectre and Meltdown attacks by training them on features indicative of Spectre and Meltdown as described above.

As mentioned before, any solution to Spectre and Meltdown vulnerabilities will be a tradeoff between performance and security. Researchers found that detection overhead ends up being less than 2% with a sampling rate of 100ms [2]. Ideally, this sampling rate would be much faster, however, the performance degradation would be too much to justify. This sampling rate does lead to some false negatives as the indicators might not be picked up quickly enough for the detector to analyze.

9 Conclusion

The discovery of Meltdown and Spectre shook up the future of computing. Due to optimization techniques, Spectre and Meltdown attacks could potentially infiltrate almost any system and leak sensitive data. While no full solution has been found, smaller patches as well as real-time detection is needed to mitigate the potential effects of these vulnerabilities.

ACKNOWLEDGMENTS

I would like to acknowledge the contributors of the research used for this paper.

REFERENCES

[1] Ahmet Efe and Muhammaed Onur Gungor. 2019. The Impact of Spectre and Meltdown. International Journal of Multidisciplinary Studies and Innovative Technologies 3, 1 (2019), 38-43.

[2] Bilal Ali Ahmad. 2020. Real time Detection of Spectre and Meltdown Attacks Using Machine Learning. arXiv:2006.01442. Retrieved from https://arxiv.org/abs/2006.01442

[3] Marc Löw. 2018. Overview of Meltdown and Spectre patches and their impacts. In Proceedings of Workshop on Advanced Microkernel Operating Systems (WAMOS). WAMOS, Wiesbaden, Hessen, Germany, 9 pages.

[4] Mengjia Yan, Jiho Choi, Dimitrios Skarlatos, Adam Morrison, Christopher W. Fletcher, and Josep Torrellas. 2018. InvisiSpec: making speculative execution invisible in the cache hierarchy. In Proceedings of the 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-51). IEEE Press, 428–441. DOI:https://doi.org/10.1109/MICRO.2018.00042

[5] Moritz Lipp, Michael Schwarz, Daniel Gruss,Thomas Prescher, Werner Haas, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, and Mike Hamburg. 2018. Meltdown. Corr, abs/1801.01207. arXiv: 1801 . 01207. http : / / arxiv . org / abs / 1801.01207.

[6] P. Kocher, J. Horn, A. Fogh, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, and Y. Yarom. 2018. Spectre attacks: exploiting speculative execution. In 2019 ieee symposium on security and privacy. Volume 00, 19–37. doi: 10 . 1109 / SP . 2019 . 00002. https : / / spectreattack.com/spectre.pdf.

[7] Yarom, Y., and Falkner, K. Flush+Reload: a High Resolution, Low Noise, L3 Cache Side-Channel Attack. In USENIX Security Symposium (2014)

[8] D. J. Bernstein, “Cache-Timing Attacks on AES,” 2005. [Online]. Available: http://cr.yp.to/antiforgery/ cachetiming-20050414.pdf