Hunter McVey – hjm98f

Prof. Ronna Bazan

University of Missouri - Columbai

Abstract

This paper will go into detail what a Meltdown and Spectre attack are, and how to implement the Spectre attack. The setup necessary for this will also be documented within.

Meltdown and Spectre Attack implementation

Final Project – INFOTC 3910 Advanced Cyber Security

Table of Contents

[Topic 2](#_Toc57604913)

[Research 2](#_Toc57604914)

[Similarities Between Meltdown and Spectre 2](#_Toc57604915)

[How They Work 3](#_Toc57604916)

[Potential Fixes to Both 4](#_Toc57604917)

[Applications 4](#_Toc57604918)

[Citations: 12](#_Toc57604919)

# Topic

The major topic that was decided upon for this research was the Meltdown and Spectre attacks. The goal was to see if successful implementation could be done with current knowledge from the INFOTC 3910 class at the University of Missouri. Without any relevant background to hardware security, I wanted to test my knowledge of virtual machines and attack methodology on new areas of cybersecurity. The reason that this topic was chosen was because there is also a need in cybersecurity to understand the physical security of machines. I also chose this attack due to its recency. These attacks surfaced in 2018 and were major causes for advancement in processor security. I wanted to know how they work, and how they were protected. I believe it is pertinent to study both old and new attacks to prevent future ones from happening. The Meltdown and Spectre attacks are great examples of how the physical aspects of the machine can lead a system to be vulnerable as well. This vulnerability was discovered to be applicable to all intel x86 microprocessors and IBM Power processors. Thus, most people were vulnerable due to just having a specific component within their systems.

The attack went into a virtual machine and attacked my own machine that has an intel processor. This was done because my system houses a processor that was vulnerable to the attack and I had not even known it. As a consumer, and as a prospective member of the cybersecurity industry, I desire to understand the issues that effect a wide spread of people. The Meltdown and Spectre attacks were a widespread threat that could impact almost anybody with a system at the time of its discovery. These attacks represented a major threat to how processors ran on speculative execution, and they were threats to computing speeds. The implementations that I went into were how the attacks worked, and how the attacks were protected against. I only implemented the Spectre attack against myself since I wanted to make sure I did not get in trouble with anyone as well as the fact that the processor I am using does not have Meltdown vulnerabilities. The implementation is mostly a precursory glance at how these two attacks functioned. I plan on going deeper into vulnerabilities like these in the future.

# Research

## Similarities Between Meltdown and Spectre

Since both Meltdown and Spectre are both operating under exploiting speculative execution, the initial implementation of both follows the same general procedure. “First, they trigger speculation to execute code desired by the attacker. This code reads secret data without permission. Then, the attacks communicate the secret using Flush and Reload or a similar side channel” (Abu-Ghazaleh, Ponomarev, & Evtyushkin, 2018). The second half of the attack, the flush and reload section, are generally the same within both attacks and other selective execution exploits. They differ in the beginning portion: The trigger. To understand how they exploit speculative execution, one needs to know what it is doing. “Usually, when an application needs some data that is not in memory, it will issue a disk request and then stall waiting for that request to complete. Rather than simply wasting unused processing cycles while applications are stalled on I/O, the speculative execution approach uses these cycles to try to discover and initiate prefetching for the future data needs of stalled applications by running ahead of their stalled executions” (USENIX, 2020). So essentially, speculative execution is the processor trying to rush ahead of the current execution process and guess at what is coming next.

This cause major issues with the exploit and side-channels. By using a side-channel, the attacker can listen for the timing that it takes a flush and reload command to take place. By listening to these times, the attacker can understand where the referenced data is coming from. Thus, this makes the Meltdown and Spectre attacks like a side-channel attack, or “An attack enabled by leakage of information from a physical cryptosystem. Characteristics that could be exploited in a side-channel attack include timing, power consumption, and electromagnetic and acoustic emissions” (*Side-Channel Attack - Glossary* 2020). They ultimately function by understanding how the hardware is referencing memory. By manipulating the CPU cache, the attacker can send new requests for memory with speculative execution. Since these requests are done before the CPU checks if the user has authorized access, when the user is determined unauthorized, they can flush and reload. The quicker response time will indicate where the information that was referenced is located. This is since the memory in the first call was already loaded into the CPU cache. Since the system can do a faster response from the cached memory, the attacker then knows where that information is. Specific attacks can then go further into the information like the Foreshadow exploit which can read from a specific cache itself making it much more effective. Depending on how the initial step is done within a speculative execution exploit, the attacker can gain valuable information from the target system.

## How They Work

Meltdown work through a single instruction implementation. It works because Intel microprocessors process memory before they for a privilege check. Intel does this methodology to save time for the computer. However, since the information has already been retrieved, it does not matter if the application, or attacker, has permissions. The attacker can simply use the flush and reload implementation and listening on a side-channel to obtain the secret information they were not supposed to receive. “As a result, an attacker can dump the entire kernel memory by reading privileged memory in an out-of-order execution stream, and subsequently transmitting the data via a covert channel, for example, by modulating the state of the cache. As the CPU's internal state is not fully reverted, the receiving end of the covert channel can later recover the transmitted value, for example, by probing the state of the cache” (Schwarz et al., 2020). This shows how dangerous these attacks are. A user with normal access to the system can gain information from memory without any escalation. All they must do is send the correct instruction, and then access a side-channel, and they can recall the information that they were not supposed to have gotten. This exploit shows how cybersecurity professionals must look at every aspect of a computer to protect it. To protect a system, one must look inwards as well as outwards. Sometimes a vulnerability could be right there, and they never knew about it.

With Spectre, as stated before, the actual retrieval of information is the same once the attack has been done. However, the difference between the two is in the first step. “Spectre attacks manipulate the branch-prediction system. This system has three parts: the branch-direction predictor, the branch-target predictor, and the return stack buffer” (Abu-Ghazaleh, Ponomarev, & Evtyushkin, 2018). Instead of a single instruction, Spectre uses the branch system to gain access to information. It essentially tricks the systems predictions into doing something it should not. For example, if something is trained that it should go a certain direction five times, the next time it is called it believes it should go there again. Then, by misleading the system, the attacker can then lead the predictor into a section of code that the attacker wants it to execute. Another Spectre attack is, “the attacking program can fill the predictor structures with carefully chosen bad data as it executes. When an unwitting victim executes their program either at the same time as the attacker or afterward, the victim will wind up using the predictor state that was filled in by the attacker and unwittingly set off a gadget” (Abu-Ghazaleh, Ponomarev, & Evtyushkin, 2018). Essentially, the Spectre attack works with supplying bad data to the processor to trick the speculative execution instead of listening like Meltdown. By mis-training or setting up an execution attack, the attacker tricks the predictor into using the attacker’s code.

## Potential Fixes to Both

The major reason that these attacks were so detrimental, when they were found, was because they affected almost everyone with a computer system and the known fixes at the time would severely limit hardware functionality. Some quick fixes were done around the time that this exploit was published. “Microsoft, Intel, and Antivirus vendors have managed to get on the same page for most up-to-date Windows 10-era devices.  Antivirus makers have incorporated the changes Microsoft needed, Intel has given Microsoft the code they needed, and Microsoft has pushed out the patches for Windows systems” (Peck, 2018). With this, the code for the processors, and protections from malicious web code were implemented for version two of Spectre. However, this left a time frame of almost 3 months between the beginning of January and late March that did not have concrete solutions. After a period as well, it was noted, “In April, however, Intel deemed some of its affected CPUs too tough to patch. The chip manufacturer released a revision notice listing 16 microcode updates as "stopped", meaning that while patches had previously been released, they were no longer being rolled out” (*What are Meltdown and Spectre and are you affected?* 2018). This shows that there are still chips that are being used by consumers that are still affected by this. The only other implementation for this fix id to go into your BIOS and make sure that it is completely up to date. There are other fixes that are temporary, but only certain chipsets are completely free of the vulnerabilities.

# Applications

A screenshot of a computer

Description automatically generated  
The implementation that I followed for this exercise of Spectre began with understanding and implementation of how Spectre works and abuses speculative execution. With the first code, I implemented a script that would functionally call the memory of 2 arrays that simulated addresses. Once the flush function had been called, the 2 addresses were referenced within the code. Now, once you go through the array of the addresses, the two that had been called beforehand were able to be access much faster within the code. This shows how you can listen for information that had been stored in the CPU cache.

A screenshot of a computer screen

Description automatically generated

Now, once you go through the array of the addresses, the two that had been called beforehand were able to be access much faster within the code. This shows how you can listen for information that had been stored in the CPU cache. When you call the addresses that are within the CPU, you can see that certain ones are seen in less CPU cycles; Thus, as an attacker, we now know were certain information is located within the system based on the location of where it is located. This forms the foundation for how to implement the Spectre attack itself and how to find the address of the information that we are looking for.

Graphical user interface, text

Description automatically generated

With the foundation of the memory done, I moved onto the Flush+Reload methodology. The purpose was to get a value associated with a function that had been compromised. The goal was to use the secret value the target function used and using its memory access to gain that information without permission. To begin, you must Flush the side channel so you can avoid any values within the cache from previous attempts. Then, there was a “victim” function that called the secret value. This is the important part. By the target function calling the value, we now have a reference within the CPU cache. Once the function has been called, we can use the Reload function to call the addresses and listen to the time it takes the CPU to access them. By comparing those times, we can find the address for the secret value within the cache. Once we hit the target threshold found from the first function, we can find the value. Once it is in practice, we can see that the value is within the cache, and that the value itself was held within the cache as well.

A screenshot of a computer screen

Description automatically generated

A screenshot of a computer screen

Description automatically generated

The next setup in the process was to implement the speculative execution manipulation. To do this, I had to trick the CPU into speculating the next branch of code that would be processed. By doing so, you can cause functions to come back true even if they are false. The CPU in the example would believe that the next call would be true and would begin executing the next information. Once the code realizes that the value is false, it cancels everything out and discards the information. However, the information is now stored in the cache of the CPU. Hence, the CPU left no physical evidence, but the information is still there. Once this has been executed, the Spectre attack has succeeded. We can then do the Flush+Reload that was implemented above to gain the information that is now in the cache.

A screenshot of a computer screen

Description automatically generated

As seen above in the execution, the value that was stored in the cache is successfully found and the number associated with it as well. This information should not have been accessed by the code, but because of the speculative execution mis-training that was done, the value can be found.

A screenshot of a computer screen

Description automatically generated

With the experiment out of the way, you could then move on to the actual full on Spectre attack. This is the main portion of the code that allowed for the full Spectre attack. There are two areas within a sandbox environment for web browsers the restricted and unrestricted. In an actual implementation, we can use it on this sandbox environment that web browsers use. By using the same out-of-order execution from the previous step, we can use it on a real-world example. To begin, the value held within the address is offset with a buffer value. The buffer value is made large to make sure that the value will always work for the offset purpose. Based of the previous iteration, we have the implementation to trick the CPU with the speculative execution process. Once both of those tasks are completed, the information within is then called into the cache from the buffer call. Once it has been called, it is just a matter of using the Flush+Reload function to find the information just like from before. Through that you can get the information as shown below and the secret value is revealed.

Text

Description automatically generated

A screenshot of a computer screen

Description automatically generated

Finally, the last implementation for the Spectre attack was to improve upon the implementation and get the actual number of hits that we can get but automatically running the code multiple times. Instead of just a single run, which could provide misinformation, we run it multiple times with a time buffer to see the actual number of address hits we can get within a specific data set. To do this, you simply change a couple of lines from the original attack. The first change is in the reload, we need to keep track of every time an address is considered the correct one. This is simply a counter function addition. Then later, we compare the number of hits within a loop to find the largest one. Finally, output the largest address, value, and hits. By doing this on the total two-hundred and fifty-six possibilities, you can find that we can get to about fifteen to twenty precent confidence in a single address. Thus, we can then have higher certainty that the address that we find is the actual one with the value. Now, the best version of the Spectre attack that I implemented was complete.

A screenshot of a computer screen

Description automatically generated

# Citations:

Abu-Ghazaleh, N., Ponomarev, D., & Evtyushkin, D. (2018, February 28). How the Spectre and Meltdown Hacks Really Worked. Retrieved November 29, 2020, from <https://spectrum.ieee.org/computing/hardware/how-the-spectre-and-meltdown-hacks-really-worked>

CSRC Content Editor. (2020). Side-Channel Attack - Glossary. Retrieved November 29, 2020, from <https://csrc.nist.gov/glossary/term/Side_Channel_Attack>

Du, W. (2018). Retrieved November 30, 2020, from <https://seedsecuritylabs.org/Labs_16.04/System/Spectre_Attack/>

IT Pro Team. (2018, August 29). What are Meltdown and Spectre and are you affected? Retrieved November 30, 2020, from <https://www.itpro.co.uk/exploits/30478/what-are-meltdown-and-spectre-and-are-you-affected>

Peck, J. (2018). Meltdown and Spectre: The Risk Manager's Guide. Retrieved November 30, 2020, from <https://ci.security/resources/news/article/meltdown-and-spectre-the-risk-managers-guide>

Schwarz, M., Strackx, R., Hamburg, M., Yarom, Y., Genkin, D., Kocher, P., . . . Lipp, M. (2020, June 01). Meltdown: Reading Kernel Memory from User Space. Retrieved November 30, 2020, from <https://cacm.acm.org/magazines/2020/6/245161-meltdown/fulltext>

USENIX. (2020). Speculative execution. Retrieved November 29, 2020, from <https://www.usenix.org/legacy/publications/library/proceedings/usenix03/tech/fraser/fraser_html/node3.html>