A Security Analysis of the Meltdown and Spectre Vulnerabilities

Gladys Diaz
College of Engineering
SUNY Polytechnic Institute

Abstract This paper will detail the context and mechanisms behind the Meltdown and Spectre security exploits revealed by Google's Project Zero research team early in 2018. After a brief introduction and definition of some terminology, there will be an explanation of how each exploit uses processor optimization techniques to circumvent the fundamental security features of memory isolation and user privilege. The paper then presents the scope of the security risk posed by the exploits and explains the steps being taken to mitigate the effects. The paper goes on to explain Google's disclosure of security vulnerabilities, and the industry's response. Finally, there will be a brief discussion of future design considerations with respect to processor architecture that take these new exploits into consideration.

1. Introduction

New security vulnerabilities allow shared data to be leaked, by taking advantage of CPU optimization techniques to avoid security measures that are supposed to be implemented by processor microarchitecture. These security flaws affect the vast majority of Intel, AMD, ARM, and other processors on the market including desktops, laptops, and servers, but also mobile devices, tablets, and smart-appliances. While the discovery of new weaknesses can be highly disruptive, it allows cyber-security experts to harden operating system's security measures, implement new hardware-based solutions and keep important data from falling into malevolent hands.

Computer systems are often times designed and built with speed and capacity as their top priorities and their main selling points. While developments in semiconductor technology often boost the speed and efficiency of computer hardware, chip makers are constantly searching for new methodologies to make more effective use of existing technology. Until the discovery of Meltdown and Spectre, security has, often times, been overlooked. Both flaws have existed and been "unknown" for at least the past 10 years and possibly dating as far back as 1995.

The Meltdown exploit allows a user to view the data of all users on an operating system and even the cloud. Furthermore, a user can read system files, without the need of elevated permissions. Spectre attacks allow an intruder on your physical machine to clearly read sensitive data such as passwords, SSN, banking numbers on a web browser. Both vulnerabilities open possibilities for catastrophic attacks.

Analyzing the approach of the attack aids CPU developers in searching for proper solutions and mitigation techniques to halt or minimize the damage done by Meltdown and Spectre. Countermeasures have been quickly rolled out, and even more refinements to these mitigations are released every week. Consequently, if left unpatched, the results could be devastating, which gives you an idea of the severity of the vulnerabilities. However, since the flaw is in the microarchitecture it, the solution to stop these vulnerabilities is to create a new chipset.

This paper will go over terminology needed to understand the security implications and exploitation of optimization techniques. As well as provide information on identifying vulnerable systems and applying patches to minimize damage of vulnerabilities.

2. Background

This section explains the terminology in technical detail, that is a part of exploiting the recently discovered vulnerabilities. I'll describe processor design and the process of retrieving data from memory, as well as Intel's microarchitecture. I'll describe the security techniques being circumvented, the optimization methods taken advantage of and side-cache channels used to leak data.

2.1 Memoy Hierarchy

Overall, processors are divided into three parts: the main memory, CPU cache, and CPU core. The main memory is also known as RAM that stores data being used by the processor. While the CPU cache is a small reservoir between the RAM and CPU core for frequently used data, it allows for faster retrieval of data.

All modern processors have three cache levels called L1, L2, L3. The lowest level cache has the highest data retrieval speed, but the smallest storage. Main Memory(RAM), on the other hand, is larger in storage but transfers data at a much slower speed. All cores share the Last Level Cache(L3) and Main memory, but each CPU core has their own L1 and L2 cache as illustrated in Figure 1.

When data is retrieved from the memory, it first checks the L1 through L3 cache. If the data is not found, it is retrieved from main memory and placed in the suitable level cache. In order to increase memory speed, data that is frequently used is stored in either L1 and L2 cache, while

larger data sets can be stored in L3 memory for quick access [12].

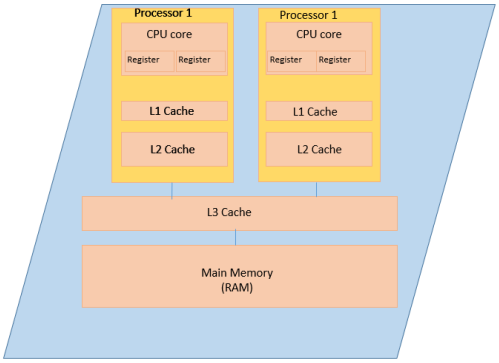

Figure 1: Shows a simplified diagram of components in CPU.

2.2 *Intel's Processor Microarchitecture*

In a single core of Intel's processor microarchitecture there are three main parts: the front-end, execution engine, and memory subsystem that share a pipeline, as Figure 2 depicts. In the frontend, x86 instructions are fetched and translated to micro-operations. Then these micro-operations are pushed into the execution engine, where out-of-order execution occurs and then get forwarded to the Reorder Buffer. The reorder buffer controls register allocation, naming and retiring. Next the Unified Reservation Station, or the scheduler [13], queues the micro-operations into the Execution Units. Each execution unit can execute different jobs like ALU operations, AES operations or address generation units, etc.… Next, the data is loaded into the Load or Store Buffer where it will be passed to the L1 Data cache or L2 Cache. [15]

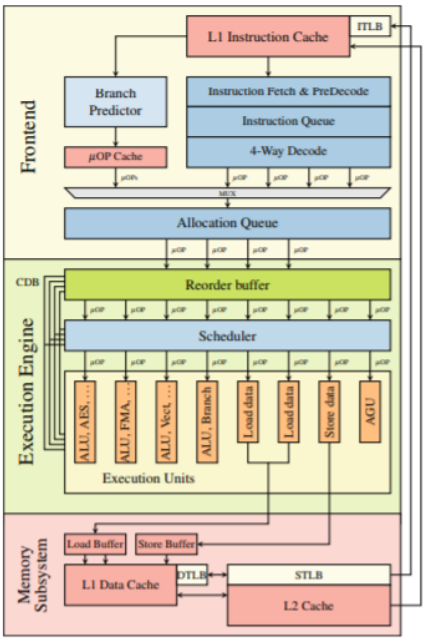

Figure 2: A diagram that shows a single core of Intel's processor and its instruction process [15].

2.3 *Memory Isolation*

Memory isolation is a security technique used to isolate the user and kernel space and protects kernel memory from user processes. Processes are marked by a supervisor bit that either allows or denies access to the kernel memory. This determines whether a process can access kernel or privileged data [15].

2.4 *Address Spaces*

To implement memory isolation, CPUs support virtual address spaces that are translated into physical address. Each virtual address space is divided into pages and each page is mapped to physical memory through a multi-level page translation table. This translation table enforces privilege checks (read, write, execute) on the physical address and the table gets stored on a special CPU register, as demonstrated in Figure 3. Every time a process switch occurs the table is updated in the special CPU register [5]. Due to these updates, each process can only recall data from their own virtual space address. Within this virtual space address two sections exist, user and kernel. Recall that user address space can be accessed by programs and the kernel address space by the operating system that contains mapped physical memory.

Because the kernel process may need to run certain operations on user processes, most of the physical memory will be found mapped in the kernel. Attacking kernel address spaces can allow data to be read, leaked and manipulated. [14]

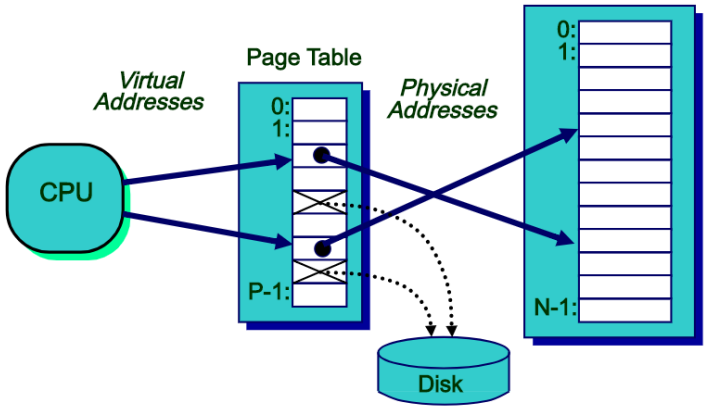

Figure 3: Illustrates the process of implementing memory isolation, using virtual address mapped to physical addresses [1].

2.5 *Adress Space Layout Randomization(ASLR) and Kernel Addres Space Layout Randomization (KASLR)*

ASLR randomizes the offset of address spaces on every boot [15], while KASLR randomizes the offset of kernel address spaces. To protect address spaces in memory from being retrieved by an unknown process, ASLR and KASLR are implemented.

2.6 *Out-of-order execution*

Out-of-order execution is an optimization technique that maximizes the use of CPU cores by getting "an operation executed before the processor has committed the results of all prior instructions [15]" This allows processing instructions to run in parallel as soon as resources are made available. In Intel's CPU core, the Scheduler, allows the processor to use

computed data as soon as they are available, rather than mapping it to a register and then reading it (older chipsets do this).

2.7 Exception Handling and Suppression

When a user task attempts to read kernel data this usually leads to an exception being raised, which causes the operation to terminate. The exception can be handled through exception handling or suppression. Exception handling, controls the error by notifying the user in a dialog window to change the task instructions that can allow a user to continue their instruction through other manipulations. Exception suppression, prevents the error by rolling back to a previous state through user-group privileges, thus a user processes can continue with their operations without the exception interruption [15].

2.8 Branch prediction

Branch prediction guesses the likely instructions that processor will perform in the future. On most modern processors a two-level adaptive prediction is implemented, which use statistics based on past events to predict the next instructions [15]. The branch predictor sets up multiple paths of instruction that may be committed by the processor when it receives the next request. On these predicted branches, there are several checkpoints created to allow the CPU cores to be recalled for a wrongly guessed path. For example, the branch predictor will guess multiple outcomes for a variable x, the unknown request. The branch predictor may create paths to retrieve information from memory address 1, 2, or 3.

2.9 Speculative Execution

Speculative execution is an optimization technique that prematurely executes instructions on the branch predictor, while the CPU core is waiting for a request. This cuts down idle time and maximizes the use of cores. Once a command is received, the processor executes the correct speculatively executed path and continues the branch. While the other incorrect speculatively executed paths (transient instructions) are discontinued and its execution engines are recalled to a checkpoint on the branch and resume on the corrected path. A quick note, that only wrong instructions were halted and removed, but the data retrieved is still in the CPU cache [13].

2.10 Return Oriented Programing(ROP)

Return Oriented Programing(ROP) uses small instructions of binary code, gadgets, that are combined together to perform computations and return data. These computations can load and store data from memory, where the return function feedbacks results to the program. It is possible to use this ROP to exploit buffer overflow flaws, by using the return function to initiate the next gadget address into executing the next gadget [14].

2.11 Side Channel attack: Flush + Reload & Evict +Flush

Side channel attacks based on Rambus definition are “any attack based on information gained from the physical implementation of a cryptosystem, rather than brute force or theoretical weaknesses in the algorithm [22].” Meaning using an attack to gather information from physical components; in the case of Meltdown and Spectre, the physical component is the CPU cache.

Cache-based side channel attacks are Flush + Reload and Evict+Reload. They both leak data in CPU cache through cache lines and measure the read data speed of the L3 cache. A key difference between the two techniques is that Flush + Reload uses machine instruction “clflush” and Evict + Reload uses cache contention to “replace the memory by loading other data [25]” to evict data from the cache. It is recommended to use Flush + Reload since it does not suffer from false positive data thus making it more reliable for a low error reading.

2.12 Cache-timing:

Using a cache side-channel attack, allows you to determine whether the data is being reloaded on the victim’s end, through cache or RAM memory. Data that is retrieved from the cache will have a fast retrieval speed with a low error rate, while data retrieved from RAM will have a high retrieval time with low success rate.

3 ATTACKS

In this section, I will explain the overall execution of the attack and the inner workings. I’ll explain which security features are bypassed by these exploits, along with results discovered by Google’s research team, Project Zero.

3.1 Exploiting Meltdown

Meltdown bypasses memory isolation by taking advantage of out of order execution in order to allow a user process to read kernel memory. Meltdown breaks down all security fundamentals assumed by hardware implementation [15].

To exploit Meltdown, the attacker must be on the physical machine to execute the malicious code that attempts to read kernel address spaces as an unprivileged process. The attacker can target addresses that it wants to obtain from the kernel. Regardless, if he/she does not, executing this attack on a random kernel address can lead to the dumping of the entire physical memory. All processes are translated into a virtual address that is mapped to a physical address, where the supervisor bit is set. When this code is launched, the processor checks the privilege table to either continue or halt the malicious code. Since the malicious code, does not have the right permissions, an exception is raised, and the process is terminated [15].

However, because of out-of-order executions, the CPU core already ran the instructions from the malicious code, between the time of checking the table and raising the exception. The execution of these instructions get data from the memory without checking the privilege table and stores it

into the CPU cache. As the processor core realizes that the code is not privileged it terminates the code, by discarding the incoming instructions. To the user, the execution of the code is halted. On the microarchitecture level, however, the kernel data retrieved is still store in the cache. Data in the cache is susceptible to side-channel attacks like Flush + Reload, Prime + Probe, Evict + Reload. The attacker can use the most efficient and accurate side-channel attack, Flush + Reload, which recovers the data in a low-noise channel, with a high transfer rate [26]. Allowing the attacker to leak the kernel data. By repeating this attack in different addresses, the attacker can dump the entire physical memory. This attack can also be launched in virtualized environments for the same results [15].

The malicious code can also implement exception handling and suppression to deal with the exception for further transient instruction execution before termination. Even though security techniques like ASLR, KASLR exist these randomization offsets are limited to 40bits. 40 bits of randomization for 8GB of RAM takes about 128 guesses to retrieve the offset value in seconds, thus this security feature is does not stall a Meltdown attack. It is important to note, that Linux kernels before 4.12 did not have KASLR enabled by default [15].

Meltdown attacks have been successfully launched on personal computers secured with ASLR, KASLR (to decrease memory address attacks), SMAP (Supervisor Mode Access Prevention), SMEP (Supervisor Mode Execution Protection), etc.… and have no software vulnerabilities. The Meltdown attack works effectively on both Windows and Linux OS. This concept was further tested with clouds services like Docker, LXC, and OpenVZ that use Intel CPUs. These services allow users to store and access their data on the web. For cloud hosting services, many of them separate users through the use of containers, instead of VMs since its leverages lower cost for the company [23]. However, these containers, run on the same kernel. Thus, exploiting meltdown in a container will allow a user to leak data of other containers on the same kernel as Figure 4 illustrates.

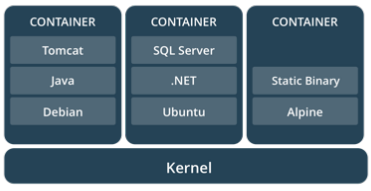

Figure 4: Shows the implementation that most cloud services use. [17].

Researchers verified that the data obtained from the cache dump is from the actual kernel memory by using a kernel debugger that contains the kernel address, confirming the effectiveness of the attack. By using the attack process stated above the attacker can obtain a dump of the entire memory by iterating through different address spaces. In Google's experiment, the kernel memory was dumped at up to 503 KB/s with an error rate as low as 0.02% [15].

3.2 Exploiting Spectre

Spectre abuses branch prediction and speculative execution in order to leak sensitive information about its own program. Spectre attacks work by tricking the CPU core into running the wrong execution paths when the processors have already received correct execution path from programs. An attacker can use these wrong speculative execution paths to leak information within memory address space. There are two variants of Spectre: Variant 1, which is called "bounds check bypass", and Variant 2, which is called "branch target injection" [6].

To execute the "branch target injection" variant, the attacker must first train the branch predictor into creating paths that can be "speculatively executed". By executing repetitive instructions, the user can train the processor to guess the next likely paths, eventually the user will be able to "predict outcomes" before the processor branch predictor. Once the attacker-influenced branch outcomes are predicted they are preemptively executed, thus if malicious code (transient instructions) is inserted in the predicted branches it can also be executed. These transient instructions could be written in binary code for a specific CPU using operating system libraries or in JavaScript language to read users memory address. In this case, the transient instructions are to retrieve sensitive data from its own user address and place it in the CPU cache. Once a the correctly executed branch is selected and run, other incorrect outcomes are discarded. Once again, the contents of the cache are untouched. The attacker can then use Flush+Reload or Evict +Reload cache-based covert channels to recover the data.

The idea is the same with variant 1 of Spectre but bypassing the bounds check. For example, use the code from Figure 5, to grasp this idea. If the value in variable X is less than the array1 size it may proceed into setting a value for variable Y. Variable Y's value would be set in Array2 in the (X's value in array1 multiplied by 256) element. This code first checks the bounds of the variable X, which prevents access to Array 2, if the condition is not met. Upon reaching this conditional branch, the processor preemptively executes possibilities which the attacker can mistrained the branch predictor to guess. If X is set to a greater value (out of bound check) then the processor will read array1 to notice that the value does not exist and will proceed to check an address from the memory that adds X value to the base address of array1, due to speculative execution logic. This would result in a value existing(k), fetching data and returning it to the CPU cache. The code continues to preemptively execute line 2, and reads data using the value retrieved above and attempts to read the data at the new address in Array2. Around here the processor realizes that the bound check was not met and discards the reminding instructions; however, the cache state reminds the same. Now the attacker can retrieve this secret value k, through side-channel attack. For more data values, the

attacker can iterate through a loop of an array of memory address, to reveal more memory contents.

```
if (x < array1_size)
   y = array2[array1[x] * 256];
```

Figure 5: A conditional branch example [14].

Project Zero launched Spectre attacks in C for x86 processors, on both 32- and 64- bit Linux and Windows OS with Intel, ARM, and AMD processors that successfully leak secret data. Secret data can be personal information (SSN, banking numbers, login keys, PINs), computer logs, passwords, etc….. Furthermore, the concept was pushed to abuse browser sandboxing, by using JavaScript code mounted on a web browser, that also leaked secret data. Unlike meltdown, no exceptions occur in Spectre attacks, since the user process is trying to read its own memory, not the kernel [14]. Although, it may seem that Spectre is easier to execute, its complexity is in launching the attack with transient instructions that must be tailored to each system.

A similar exploit like Spectre is ROP, which can obtain data by running malicious gadgets continuously with the return function until the processor core terminates the sequence. Unlike Spectre, ROP can only read data and works on bug-free programs. In addition, Spectre exploits optimization techniques to launch more sophisticated attacks that allow data to be read, computed, and memory to be accessed.

In Googles' experiment, it was confirmed that the branch predictor can be mistrained by executing code on one hyper thread. The mistrained branch predictor also affects other hyper-threads on the same CPU, due to a shared cache that maintains records of past events with their predicted sequences. It was also observed that the branch predictor pays attention to branch destination virtual address, applications using the same DLL (Dynamic Link Libraries) that load one instance mapped to the same virtual address for all apps. Mistraining the branch predictor can influence the branch predictor into committing illegal destination address, which raise an exception that can be handled through a try...catch function. Although this leads the branch predictor to guess other processes into illegal destinations [14]. In sum, to run a Spectre attack, malicious code must be launched within the victim's system to reveal sensitive data in address spaces chosen by the attacker.

3.3 Comparing these vulnerabilities

Meltdown and Spectre are both vulnerabilities in microarchitecture that allow data to be leaked from memory. They both exploit optimization techniques to fetch data not accessible to malicious process to obtain secret data. After the exploitation occurs both use a side-channel cache attack to recover the data from the address space targeted. Both attacks cannot be remotely executed yet or modify data. Meltdown only affects Intel processors due to the way they are built to handle out-of-order executions. Exploiting Meltdown allows an unprivileged process to gain kernel data access. Meltdown works on personal machines allowing a user to read other users' data on the system. Taking the exploitation, a step further, a user on a cloud service can view other virtual users' data on the same physical kernel. Of the two vulnerabilities, Spectre is much more sophisticated to launch, but affects all modern processors. Spectre abuses branch prediction and speculative execution to run code that fetches secret user data from its own address space. Allowing an attacker to visibly see passwords, cookies, personal information and much more that should have been protected. Furthering this attack, could allow attackers to use the obtained information to gain system privileges or launch further attacks such as bank transfers, spam, data disclosure, etc…..

3.4 Processors affected:

All Intel Processors dating to up to decade ago have been affected by Meltdown except the Intel Itanium and Intel Atom made before 2013 [6]. Meltdown does not affect AMD processors, due to their microarchitecture design, they do not use speculative references. A couple of server ARM CortexR and -A model processors are affected by Meltdown [2]. All brand processors (Intel, ARM and AMD) are affected by Spectre. In addition, every electronic device containing modern processors (smartphones, smart appliances, cloud servers, etc...) are vulnerable to Spectre.

3.5 Identifying system vulnerability

Identifying whether your system has these vulnerabilities brings you one step closer to securing your system. There are a couple of methods for determining if your machine is affected. The first is to manually figure out the processor brand model. For Windows computers, you can find your CPU information by right-clicking the start button and clicking on "System", this would open a window with information regarding your computer including processor model. On Apple computers, selecting the option "About This Mac" on the Apple menu, will reveal general information about the CPU. More detailed information can be found by opening a terminal and typing "sysctl -n machdep.cpu.brand_string", giving more detailed CPU info. In Linux systems, typing "cat /proc/cpuinfo" in a terminal, lists CPU information.

The second method is downloading software to determine your vulnerability. For windows OS, downloading Gibson Research Corporation tool "InSpectre", determines if the system is vulnerable to these new exploits using a simple interface[9]. For a technical person, or a script lover they can use Windows Speculation Control Validation PowerShell Script, created by Microsoft [20]. This PowerShell script also determines if prevention measures are installed for these vulnerabilities. While on Linux systems, a user can download the spectre-meltdown-checker.sh script from GitHub [19].

4 MITIGATIONS

In this section, I will discuss the mitigation techniques being implemented to combat these vulnerabilities, and their effects on CPU performance. Since this flaw exist in the CPU's hardware, I will also be discussing the long-term changes that need to be implemented in CPU architecture, as software and microcode-based mitigations will not be effective in the long run.

4.1 Meltdown Mitagations

To totally prevent the abuse of the Meltdown, exploit in the first place, one obvious solution is to disable out-of-order execution. However, by disabling this feature processor performance would be severely hampered, which is not feasible considering the processing needs of modern hardware, especially mobile devices. Inspecting, the out-of-order execution technique, we note that data is fetched before the privilege check is done. Implementing a privilege check for each data address being fetch before the execution engine, would prevent privileged data from being obtained. Yet once again this negatively affects processor performance, because it requires more resources to examine the permissions of each memory address retrieved [15].

A software mitigation that sufficiently minimizes the leakage of kernel memory is KPTI (kernel page table isolation) or KAISER (kernel address isolation to have side-channels efficiently removed). This patch creates a stronger memory isolation between the user and kernel. With KAISER, only necessary x86 architecture kernel mappings are placed in the user space. Without most of kernel mappings, the user cannot obtain the necessary addresses to exploit Meltdown. It has been verified by Project Zero that the implementation of KAISER strengthens protection of the kernel and physical memory [15]. The downfall of isolating the user and kernel space further, is an increase in use of system resources, thus slowing down computer processes all together. According to PostgreSQL, they tested these patches on 16 clients running i7-6820HQ CPU processors and saw a 7% to 20% decrease in performance [8]. Take into consideration that the biggest performance impact would be seen in servers running several applications and data calculations. This mitigation was launched as an automatic update in PCs via the operating system.

It has been stated in Project Zero's paper, to strengthen KPTI/KAISER to its full potential, one can completely get rid of kernel memory mappings in the user space, by using a trampoline function for kernel pointers that must be in the user space. As of right now, this technique is theoretical. The implementation would need to be developed and an assessment of system performance would need to be analyzed.

The ultimate solution is to create a new processor with a solid split between user and kernel space. If implemented on new chipsets, chipset makers would allow users to enable the option on processor by a hard-split supervisor bit. The processor would then split the address space into two halves, the kernel space would reside on the top while user space on the bottom. In applying this hard-split technique, a process is clearly defined as being from the user or kernel, and instructions can be executed using out-of-order execution. The CPU performance of these new processors would be slightly slower than the chipsets that currently have the hardware flaws. Although if implemented as an optional feature on new chipsets, the user would be allowed to weigh the options [15]. This is a viable solution, but it will take time for chipset makers to develop and test new processors, before they are released to the public.

4.2 Spectre Migations

Several methods have been considered for preventing the exploitation of Spectre. Preventing speculative execution has been considered, however this is a necessary feature for modern processors especially on mobile hardware. Implementing a boundary check for instructions would prevent malicious code from obtaining sensitive data; however, creating a patch would prove to be challenging to further verify a user process instruction and would cause a noticeable performance hit in affected CPUs. Speculative execution can be limited to execute a maximum number of instructions ahead of a wrong path, yet this patch would severely degrade performance. Preventing Spectre will also require implementing countermeasures against abusing the branch predictor. Indirect branch poisoning is difficult to prevent since it would require disabling of the analysis of the statistics gathered, which at this time is not a feasible solution either [14].

It is possible to patch Spectre with some of the methods described above but would require the manipulation of microcode. However, not all the theoretical microcode can be created because there is no mechanism to implement these fixes [14].

A recently developed countermeasure specifically for Spectre variant 2, is Redpoline. This mitigation was developed at Google with binary code that uses a "separate predictor for function returns, to allow a construction of an indirect branch which is safe from speculation-based attacks [21]." This technique is applied to sensitive addresses (such as operating system libraries) so that the constructed return function would bounce the speculative execution until it is halted. The binary code uses call (to accept the branch instructions) and ret (return) functions, to return the call function. Creating an endless trampoline function, that keeps speculative execution from continuing its instruction. There is a negligible overhead of performance, that significantly less than the KPTI performance hit [11].

Site Isolation, which is implemented in a web browser, is a mitigation technique that provides additional protection against both flaws. It was created by Google's developer team as an experimental feature that creates another defense in isolating different web processes. This works by isolating different website processes into different pages. This blocks data from being transferred from one site to another as

well as making it difficult to obtain data through attacks like Meltdown and Spectre [18]. Although it is an effective countermeasure in preventing data leakage, it requires a higher overall memory use by the browser, causing a performance drop for other system resources. In addition, the feature is still being debugged by developers, which may cause JavaScript files to break, iframes to be clipped, or the scrolling feature to not work properly. If implemented on an unpatched system, it can provide protection for data on web browsers

4.3 What users and administrators can do

Update personal computers and systems to get the most recent software, firmware and microcode updates for your OS. For IT professionals, it is important to check with system vendors for more information regarding security patches against these attacks. Implement techniques such as site-isolation, KPTI, and Redpoline to further secure your data. Install and/or update antivirus programs to avoid running malicious code that can exploit the vulnerability on your machine. It is important for you to take the proper steps towards minimizing the risk posed by these vulnerabilities.

5 Google's Disclosures and handling of security crisis

According to Intel's disclosure report about the new security findings that were discovered by the Google Team, an industry-wide collaboration took place to mitigate the attack through multiple, concerted approaches updating system software, CPU firmware, OS system updates [16]. It should be noted that recently, U.S. representative Greg Walden has questioned Intel's non-disclosure of Meltdown and Spectre to the government. The government issued a statement that their "non-disclosure can pose a threat to national security" Intel along with other chipset makers stated they did not inform the governments because "zero-day vulnerabilities are not known by hackers and government officials could not implement solutions in preventing attacks[3]."

Google's Project Zero research team saw to it that this critical, zero-day vulnerability was disclosed in the most responsible way possible. This stands in contrast to past events where government organizations, such as the NSA, were informed first only to have vulnerabilities and attacks used against private citizens and the public, including the government. Professionals should learn about the disclosure methods that the researchers have taken. This serves as an excellent example of companies working together to resolve a potential security threat to all organizations in an ethical and effective manner.

6 Conclusion

The discovery of the Spectre and Meltdown attacks exposed several flaws with modern-day processors directly resulting from our need for speed, rather than security. These security vulnerabilities are especially disruptive, firstly because they take advantage of the inherent architecture of the processor in an unexpected and highly effective way, allowing shared data to leaked. Secondly because of the vast majority of processor models affected, dating back ten years. As far as we know these types of attacks do not leave a trail, and cannot be detected, making it more imperative to apply patches to mitigate effects [24].

Society's desire for performance has too often been prioritized in favor of better security. It is likely that many of the outdated processors will never see a proper fix for these vulnerabilities, meaning that users will have no choice but to upgrade their hardware if they want their data to remain secure. To develop a secure and powerful CPU, processor architects and software engineers must collaborate in order to prevent catastrophic vulnerabilities, such as Meltdown and Spectre from going unnoticed in the design process.

References

[1] A System with Virtual Memory, http://slideplayer.com/slide/9445838/#

[2] Arm Processor Security Update, ARM Developer, March 2018, https://developer.arm.com/support/security-update

[3] Campbell, M., Intel failed to disclose Meltdown and Spectre to government until flaws made public, Apple and others confirm, Apple Insider, February 2018, https://appleinsider.com/articles/18/02/22/intel-failed-to-disclose-meltdown-and-spectre-to-government-until-flaws-made-public-apple-and-others-confirm

[4] Corbet, J., KAISER: hiding the kernel from user space, November 2017, https://lwn.net/Articles/738975/

[5] Context Switch Definition, The Linux Information Project, May 2006, http://www.linfo.org/context_switch.html

[6] Crowe, J. The Meltdown and Spectre CPU Bugs, Explained, Barkly, Jan 2018, https://blog.barkly.com/meltdown-spectre-bugs-explained

[7] Fiser, D., and Gamazo, W., Detecting Attacks that Exploit Meltdown and Spectre with Performance Counters, Trend Micro, March 2013, https://blog.trendmicro.com/trendlabs-security-intelligence/detecting-attacks-that-exploit-meltdown-and-spectre-with-performance-counters/

[8] Freund, A. , Heads up: Fix for intel hardware bug will lead to performance regressions, PostgreSQL, January 2018, https://www.postgresql.org/message-id/20180102222354.qikjmf7-dvnjgbkxe@alap3.anarazel.de

[9] Gibson, S., InSpectre, Gibson Research Corporation, https://www.grc.com/inspectre.htm

[10] Gruss, D., Maurice, C., Wagner, K., Flush+Flush: A Stealthier Last-Level Cache Attack, https://online.tugraz.at/tug_online/voe_main2.getvolltext?pCurrPk=88379

[11] Intel Corp to Discuss Reports of New Security Research Findings, Janurary 2018, https://s21.q4cdn.com/600692695/files/doc_downloads/INTC-USQ_Transcript_2018-01-03.pdf

[12] Jenkov, J., Modern Hardware, Tech and Media Labs, September 2015, http://tutorials.jenkov.com/java-performance/modern-hardware.html

[13] Kanter, D., Silvermont, Intel's Low Power Architecture,real world technologies, May 2013, https://www.realworldtech.com/silvermont/4/

[14] Kocher, P., Genkin, D., Gruss, D., Haas, W., Hamburg, M., Lipp, M., et al, Spectre Attacks: Exploiting Speculative Execution, https://spectreattack.com/spectre.pdf

[15] Lipp, M., Schwarz, M., Gruss, D., Prescher, T., Haas, W., Mangard, S., et al, Meltdown, https://meltdownattack.com/meltdown.pdf

[16] Product Security at Intel, https://www.intel.com/content/www/us/en/corporate-responsibility/product-security.html

[17] Rubens, P., Containers and Cloud Computing, Datamation, April 2017, https://www.datamation.com/cloud-computing/containers.html

[18] Site Isolation, The Chromium Projects, http://www.chromium.org/Home/chromium-security/site-isolation#TOC-Current-Status

[19] Spectre and Meltdown Checker, GitHub, https://github.com/speed47/spectre-meltdown-checker

[20] Spectrulation Control Validation Powershell Script, Microsoft, https://gallery.technet.microsoft.com/scriptcenter/Speculation-Control-e36f0050

[21] Turner, P.,Retpoline: a software construct for preventing branch-target-injection,Google Support, https://support.google.com/faqs/answer/7625886

[22] Understanding the evolution of side-channel attacks, Rambus, April 2017, https://www.rambus.com/blogs/understanding-the-evolution-of-side-channel-attacks/

[23] Vaughan-Nichols, S., Containers vs. virtual machines: How to tell which is the right choice for your enterprise, IT WORLD, April 2015, https://www.itworld.com/article/2915530/virtualization/containers-vs-virtual-machines-how-to-tell-which-is-the-right-choice-for-your-enterprise.html

[24] Wagenseil, P., Meltdown and Spectre: How to Protect Your PC, Mac and Phone, Tom's Guide, January 2018, https://www.tomsguide.com/us/meltdown-spectre-fixes,news-26326.html

[25] Wong, D., CHES 2016 tutorial part 2: Micro-Architectural Side-Channel Attacks, Cryptologie, August 2016, https://cryptologie.net/article/367/ches-2016-tutorial-part-2-micro-architectural-side-channel-attacks/

[26] Yarom, Y., Falkner K, FLUSH +RELOAD : a High Resolution, Low Noise, L3 Cache Side-Channel Attack,2014, https://eprint.iacr.org/2013/448.pdf