BILINGUAL PUBLISHING CO.
Pioneer of Global Academics Since 1984

Semiconductor Science and Information Devices
https://ojs.bilpublishing.com/index.php/ssid

REVIEW

Security Vulnerabilities in Microprocessors

Benjamin Ashby Smith   Kevin Curran[*]

School of Computing, Engineering & Intelligent Systems, Ulster University, UK

ARTICLE INFO

Article history
Received: 26 April 2021
Accepted: 19 May 2021
Published Online: 30 May 2021

Keywords:
Microprocessors
Cybersecurity
Microarchitecture security

ABSTRACT

Microprocessors such as those found in PCs and smartphones are complex in their design and nature. In recent years, an increasing number of security vulnerabilities have been found within these microprocessors that can leak sensitive user data and information. This report will investigate microarchitecture vulnerabilities focusing on the Spectre and Meltdown exploits and will look at what they do, how they do it and, the real-world impact these vulnerabilities can cause. Additionally, there will be an introduction to the basic concepts of how several PC components operate to support this.

1. Introduction

A CPU is often referred to as the "brain" of a computer system. It carries out instructions that are as dictated by the CPUs' instruction set architecture. These instructions are executed when a computer program performs arithmetic or logic, and when controlling input and output operations. These instructions are executed during a clock cycle which is a single electronic pulse within a CPU [1]. The speed at which a CPU can execute these cycles is dictated primarily by its clock speed which is measured in "hertz". Modern CPUs are so fast they are measured in "gigahertz (GHz)" which is represented mathematically as 10^9 Hz. As microprocessor companies battled to try and obtain higher clock speeds, they began to hit a limit around 3-4 Ghz. To continue making processors faster whilst avoiding clock speed limitations, chip manufacturers had to come up with some new solutions. The two solutions that will be relevant to this report are Simultaneous Multithreading (SMT) [2] and "Speculative Execution" [3].

1.1 Simultaneous Multithreading (SMT)

SMT is a technology that allows CPUs to subdivide their physical cores into multiple logical cores. Traditionally this division of cores would be 2 logical cores per 1 physical core. However, even when a physical CPU core has 2 logical cores, this does not double the capabilities of the physical core as both logical cores share many resources with the physical core, however, they do provide enhanced performance. The way this performance enhancement is achieved is that logical cores can operate in parallel with each other which allows them to execute instructions in multiple threads at the same time. A common implementation used with simultaneous multithreading is speculative execution.

1.2 Speculative Execution

A CPU can execute programs in multiple ways but the

**Corresponding Author:*
Kevin Curran,
School of Computing, Engineering & Intelligent Systems, Ulster University, UK;
Email: kj.curran@ulster.ac.uk

DOI: https://doi.org/10.30564/ssid.v3i1.3151

two relevant to this report are standard instruction execution and speculative execution. With standard instruction execution, the microprocessor does not execute any instructions until the program requests those instructions to be carried out and cannot carry out new instructions until prior instructions have been completed. This is called "In Order Execution". Instead, speculative execution allows one logical core to pre-emptively begin working on an instruction that the program may use soon, whilst the other logical core is working on executing the current instruction it has been given. This is called "Out of Order Execution". If it becomes apparent the instructions were not required, they can be discarded. A common method used in computer programs is called "control flow" (Figure 1). With control flow, a program will meet a condition whereby the program will wait until the condition resolves to be true or false. When it has been resolved the program will follow the consequent path of the result.

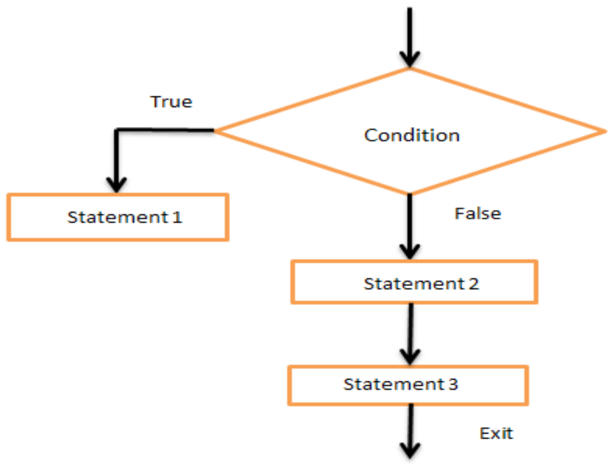

Figure 1. Control Flow Diagram [4]

It is common for speculative execution to predict the control flow of a program and execute it in parallel. This is done via a set of extremely complex algorithms and mechanisms within the CPU that can learn which path the control flow normally follows.

2. Memory

Memory is broken up into address spaces which are allocated to system processes and devices. The address space allows a CPU to know where in the memory it needs to look to access the data a program or device requires. Memory in a computer is generally separated into two categories; main memory also known as Random Access Memory (RAM), and cache memory which is stored on the CPU itself. Memory can be Physical or virtual; a physical memory address space is one that exists in system memory or cache memory whereas virtual memory (often known as a page file) is a file that is created on a storage drive such as a Hard Disk Drive (HDD) or a Solid-State Drive (SSD). When a process runs, it is assigned a pool of virtual memory. The process will be given the impression that it is working with large contiguous sections of memory, however, the memory of the process may be dispersed across many different sectors of the physical memory. When a CPU instruction is executed, it will either create data and store it in memory or if it requires data it will retrieve it from memory. When either of these requests is made, the page file will translate the mapping of the virtual memory address to the physical memory address that the data are to be stored in or is currently stored in.

To understand further how memory is used within a system, the following example provides simple pseudocode to show a basic calculation and explain what happens during the calculation:

```
integer X = 1 +2 (X now holds the value 3)
integer Result = X + 2 (Result now holds the value 5)
Print Result (The value stored in result is displayed on screen)
```

In this example, the sum of 1 + 2 is stored in an integer called X. Then a new integer called Result adds 2 to whatever the value stored in X is. This value is stored in the system memory so when the program moves to the next line, it can fetch the value of X from the memory and add 2 to it. The value of Result is now 5 and again this is stored in system memory. Finally, the value stored in Result is retrieved from system memory and is printed on screen, displaying the number 5 to the user.

The performance of memory relates to how quickly it can be accessed which is defined by two factors, its speed and its latency. The speed of memory indicates how fast it can process data whereas the latency dictates how long it will take between a command being entered and executed [5]. RAM speeds have historically not matched CPU Speeds and it is common to see even modern RAM only operate at speeds around 2.5-3 Ghz. This created a situation where the CPU would be left waiting around for the slower RAM to catch up with it which wasted CPU cycles. One measure taken to help address this issue was for CPU manufacturers to begin integrating cache memory onto the CPU itself. Cache comes in much lower capacities than RAM with even modern CPUs in 2019 only featuring around 100Mb of cache memory. However, due to cache memory living on the CPU package, the latency for cache memory is usually only a few nanoseconds versus potentially 80 nanoseconds or higher when accessing RAM [6]. When a CPU wishes to access something from RAM, it will copy that data into its cache so in the event it requires access to it again. It is readily available and much quicker to access. When there is no more space or the data are no longer required, it will be cleared from the cache.

3. Kernel and Side-Channel Attacks

All operating systems have a computer program known as the "Kernel". The kernel is the core of a computers operating system and operates with complete control over all aspects of the system. As such it is imperative that it receives its own dedicated and protected memory allocation as if a program or outside system could access the kernel, they could potentially gain access to private or sensitive data stored in the kernel. The operating system handles this allocation of memory via a process called "Memory Isolation" [7]. Memory Isolation ensures that a program cannot access another programs memory, the kernel memory or anything else stored in protected memory that is not allocated to the program that is trying to access it. This allows computers to safely run multiple programs at the same time whilst ensuring the security of a program or the entire system is not compromised. However, the system must still be able to interact with the kernel and so this is handled by a supervisor bit on the CPU that defines whether an aspect of the kernel can be accessed or not. This bit is only set when entering the kernel program and it is cleared when switching to user processes. This ensures a level of security as it means no program will definitively know the supervisor bit due to it being set during entry to the kernel and cleared during the exit from the kernel.

A traditional exploit would attack by looking for a vulnerability within a programs code. Modern operating systems will provide a software implementation to ensure that this sort of direct exploit cannot occur by ensuring the kernel memory cannot be accessed by outside programs due to being stored in protected memory. In the event such an exploit was found, this would be rectified via a software patch very quickly. A side-channel attack instead exploits how an operating system communicates with hardware upon which the operating system runs. A side-channel attack monitors the physical emissions produced by electronic circuits including power consumption, electromagnetic fields, time to execute commands and even the sounds the CPU makes when it's running. The information gathered about these emissions can be reverse engineered to decipher what the computer is doing. This means side-channel attacks are hardware and software agnostic and can target any hardware or software that may be vulnerable [8].

4. Meltdown

In 2017 several security experts discovered 3 hardware vulnerability exploits within most modern CPUs that utilised side-channel attacks. Referred to initially as variants 1, 2 and, 3, they later received the public names Spectre and Meltdown. The Meltdown exploit (Variant 3) was independently reported and discovered by 3 teams, Jann Horn from Google Project Zero, Werner Haas and Thomas Prescher from Cyberus Technology and, Daniel Gruss, Moritz Lipp, Stefan Mangard and, Michael Schwarz of the Graz University of Technology [9]. The vulnerability was reported to the public on the 3rd of January 2018 [10]. Meltdown operates using a cache attack. A cache attack is one that occurs when speculative code execution moves data between RAM and cache memory. To explain this further, an example is provided.

A user visits a website that has a malicious cache attack script on it. The script is designed to steal the users' Wi-Fi password. The script will load an image that will be used later in the attack. The script first must make sure that the Wi-Fi password is not stored in the cache. To achieve this, it will write junk data to the main memory which will then be copied into the cache ultimately flushing it of its current data. The script will then attempt to read the Wi-Fi password from protected memory. At this time the value will be copied into the cache from the RAM as there has been a read request for it. The memory in the cache now consists of the junk data the attacker has assigned and the Wi-Fi password. The program will now attempt to retrieve 1 value in the Wi-Fi password from the cache and load 1 pixel of the image based on the time it takes for this value to return. To do this they will check the first memory address after their junk data they filled the cache with earlier. The attacker can discern that a fast load of the pixel indicates a fast read which would mean the value is stored in the cache and hence must be the value they are seeking. If the read is slow it means it was retrieved from RAM and hence is not the data they are looking for. The following pseudocode shows an idea of how this setup is achieved with the presumption the memory pool is 1020 bytes in size.

```
"If (memoryAddress1001 contains the letter S) {
read the first pixel of the image that was loaded earlier
}
```

Repeat this incrementing 1 byte to the memory address each time.

However, the protected memory will obviously stop this from occurring as it knows this program shouldn't be able to access that data and the data would not be copied from RAM to the cache as the program will not be allowed to access the protected memory where the Wi-Fi password is stored. Unfortunately, speculative execution is not protected. If the CPU executes the malicious code speculatively, then the data will be moved to the cache as the speculative execution takes place. The attacker is then free to proceed with the attack and obtain the Wi-Fi password as outlined in the example.

Fundamentally the flaw exists due to certain processors not using protected memory during speculative execution. This issue applied to every Intel CPU since 1995 [11] which meant that the majority of consumer and server PCs were vulnerable to this sort of attack. Additionally, ARM who provide CPU designs to Qualcomm and Apple also confirmed that some of their CPUs were also vulnerable to the attack which meant many smartphones were also vulnerable [12].

5. Spectre

The Spectre vulnerability was independently reported and discovered by two people, Jann Horn from Google Project Zero and Paul Kocher in collaboration with Daniel Genkin from the University of Pennsylvania and University of Maryland, Mike Hamburg from Rambus, Moritz Lipp from Graz University of Technology, and Yuval Yarom from the University of Adelaide and Data61 [7]. Spectre is related to Meltdown in its execution. The exploit is again a side-channel attack and again targets vulnerabilities with how speculative execution manages memory. However, unlike Meltdown, both variants of Spectre have been confirmed to exist on nearly all CPUs created by Intel, AMD, ARM and most other CPU manufacturers [11].

Spectre has two variants; variant 1 is a "Bounds check bypass" and variant two is a "Branch target injection". These will both be covered going forward, starting with the variant 1: bounds check bypass.

5.1 Variant 1: Bounds Check Bypass

A "bounds check bypass" in a simple form is a side-channel whereby a malicious program will exploit speculative execution to access data stored in a protected memory store that lives next to the memory allocated to the malicious code. To explain this further, an example is provided. A user has a program that creates an array of data that is 4 bytes in size. They then create an integer variable called length with a value of 1000. The program performs an if statement that checks if the length value is less than the value of the array. If the statement is true, then the input will be stored in the data. This may look like the following:

```
data = [1, 2, 3, 4]
input = 1000
if (input < data.size) {
        data[input]
}
```

On the surface, this standard block of code may appear innocuous and with a standard in order execution method, it would be completely safe. However, due to how speculative execution functions this can be exploited with an out of bounds read. The attacker will follow a set of steps, like those used in the Meltdown exploit.

Initially, the attacker will train the branch predictor to expect the statement to normally resolve to true by changing the input value to something that is less than the size of the data array and executing the program multiple times. The attacker will then clear out the cache to ensure it no longer stores the protected value. When the attacker has completed this they will then change the input value to be much higher than the length of the data array, in this example, the input value is changed to 1000. The data array size is stored in RAM prior to being copied to the cache memory which means it will take a long time to access prior to executing the code due to the CPU being much faster than the RAM. To avoid wasting CPU cycles the speculative execution will execute the if statement prior to evaluating the size of the data array. It will predict that the if statement will resolve to true due to the prediction training performed earlier. It is at this point that the exploit can occur. When the CPU speculatively executes the if statement, it doesn't know the actual length of the array yet and hence it will perform the read based on the input variable size. This means that whilst the data array may only be 4 bits in length, it will read 1000 bits which can allow the program to read memory past what it should be allocated such as protected memory where private data is stored. The value that is read from private memory will be then stored in the cache memory. At some point, the CPU will realise that the actual size of the data array is less than the input value and hence determines the result of the if statement to be false. When this happens, the CPU will clear the stored data in RAM from the speculatively executed code and the private data that was accessed during the speculative execution is also cleared. However, whilst the memory is cleared from RAM, it is not cleared from the cache and much like Meltdown, the attacker can use a time-based attack to figure out what value is stored in the cache.

In this example, the attacker can simply perform a read request that tries to guess a value that was in the protected memory. If the value is returned very quickly, they can determine the value is stored in the cache memory which indicates it must be one of the protected values that were copied from RAM during the out of bounds read. If the value takes a longer amount of time, they know it is being returned from RAM. At this point, the attacker can simply repeat this process using different values for the input variable until they have obtained as much private data as they want to [13].

5.2 Variant 2: Branch Target Injection

The second variant of Spectre is known as a "Branch target injection" exploit. With Spectre variant 1, the attack is based on the conditional branch prediction. In the examples shown in both the Meltdown and Spectre Variant 1 sections of this report, this was an if statement. Variant 2 of Spectre instead uses indirect branching. Indirect branching is a feature built into all modern CPUs that allows the program to go to any memory address that is dictated as per the program. This allows an attacker to target where the speculative execution occurs rather than having to execute it in a conditional statement and then access the protected memory addresses in a similar fashion to the example provided in variant 1. The destination at which a speculative attack occurs is referred to as the "gadget". A gadget is a small group of instructions that end with a return instruction. It will usually be a piece of code that existed already within the victims' code. The gadget will be at a destination in the program that allows access to the memory address space the attacker wishes to read.

The concept behind the attack is as follows:

The attacker will train the branch predictor so the speculative execution will go to the gadget in the code. The attacker then clears out the cache to ensure it no longer stores the protected memory. The code will then run the gadget speculatively to gain access to the memory address space they wish to obtain protected data from. At this point, the same methods used by variant 1 are implemented such as a time-based attack to determine which values are in the cache by checking how long a read request takes.

6. Mitigations and Performance Impact

Due to Meltdown and Spectre being hardware-level flaws, they cannot be conclusively patched as the vulnerability is one that requires changes in the physical architecture of the CPUs to fix. However, the problem is incredibly severe and dangerous, so it is also impossible to ignore it. The only solution that would provide complete protection would be to replace the vulnerable CPUs with newer models where the hardware vulnerability has been fixed at an architectural level. Another solution was to disable SMT on the affected CPUs, the drawbacks of this solution will be detailed in the Apple section of this report. Given the number of processors that are vulnerable to these attacks, it is unfeasible to ask every consumer or customer to replace their CPUs, so an alternative solution had to be created.

The result was a set of mitigations implemented via patches to the operating systems that were created by the OS vendors, and microcode updates that were created by the CPU manufacturers in conjunction with motherboard vendors that were shipped in the form of a Basic Input Output System (BIOS) update. However, mitigation by the Oxford Dictionary definition is "The action of reducing the severity, seriousness, or painfulness of something." [14]. By this definition, it can be seen clearly that mitigation is not an absolute solution to the problem. The operating system level mitigations were handled by the organisations behind each operating system. How each operating system handles the mitigation varies and hence there must be an independent review conducted for each platform.

6.1 Linux

Ever since Linux was created, it mapped its kernel memory into the address space of every running process. This choice was made mostly for performance reasons as it means if a process needs to interact with something in the kernel, it has a mapping of the kernel address already and doesn't need to seek it out. This may sound dangerous but due to how CPUs manage memory, they can usually be trusted to prevent the user space from accessing the kernel memory. To try and ensure that the kernel address space was not easily accessed, a system called "kernel address-space layout randomization" (KASLR) was used [15]. KASLR randomized where the kernel is placed in the virtual memory address space every time a machine is booted. In theory, this meant an attacker would not be able to easily find out where the kernel address space was, even if it was mapped to a process. However, KASLR was not absolute and it suffered from information leaks that allowed attackers to then gain access to where the information may be stored. KASLR was able to be patched to close off these leaks but this is only a software solution and it does not provide any security if the exploits exist at a hardware level, such as Meltdown and Spectre.

The only feasible solution to mitigate the hardware level exploits was to discontinue the practice of mapping the kernel address space to a process, effectively making it inaccessible from the userspace. The solution to this was dubbed kernel address isolation to have side-channels efficiently removed" (KAISER) [16]. KAISER provides an implementation of separated address spaces for the kernel. What this means is that rather than the entire kernel address space being mapped to a process, the kernel mapping will instead be limited to only map the required addressed that are needed to enter and exit the kernel. As this is only a mitigation, it does not provide complete protection from the leaks and even the entry/exit information could be used to reveal where the memory address space that the kernel is stored in. However, this minimal kernel data is trusted meaning it falls under a set of security policies that prevent trusted components from being controlled my malicious code [17].

As it has been established that the Linux kernel mapped the kernel address to processes for performance, it is implied that there would be a performance regression when moving to KAISER. This performance impact depends on several systems which are as follows:

• System Call rates: The rate at which a program requests a service from the kernel [18].

• Context Switches: How a CPU can change from one process to another while ensuring they do not conflict (this is how multitasking is achieved on PCs or smart devices) [19].

• Page fault rate: The rate at which a program may try to access a memory address that is not currently mapped to the virtual address space of that process [20].

• Working set size: Defines the amount of memory that a process requires in a given time interval [21].

• Cache access pattern: The pattern in which a system reads and writes to the cache [22].

With the change to KAISER, these systems can suffer from a reduction in performance that may range from as little as a 1% regression to as high as an 800% performance regression based on the program and which system or combination of systems it uses and how often it uses them [23]. These performance regressions will be lessened as programs are updated to try and reduce the usage of any systems that are causing severe performance regressions. It is noted in Brendan Greggs article that at the time of writing he was working for Netflix, one of the largest media-services providers in the world. He stated, “I'm expecting the cloud systems at my employer (Netflix) to experience between 0.1% and 6% overhead with KPTI due to our syscall rates, and I'm expecting we'll take that down to less than 2% with tuning” [23]. It could be determined that 6% performance regression does not sound like a large amount but for a company like Netflix with many servers and extensive infrastructure, it means a loss in efficiency and that ultimately costs money. As evidenced by the quote, they intend to tune their software to try and ensure the impact is less than 2%. This indicates that Netflix perceives a 6% performance regression to be more than they are willing to accept and would rather invest their resources in trying to reduce this performance regression than living with it.

6.2 Windows

Windows does not manage its kernel in the same way as Linux did and hence it had a different approach to mitigating the Meltdown and Spectre vulnerabilities. With the kernel, Microsoft implemented a form of mitigation known as Kernel Virtual Address Shadow (KVA Shadow). This implementation creates two separate page directories for each process. The first page maps the user page tables which only contain the user mode mappings and a small number of kernel transition pages. The second page maps the kernel page tables which contain the user and kernel mappings for a process. In a simple form, this means sensitive kernel memory content is removed from the virtual address space for a process meaning a process cannot see or access sensitive kernel memory. This is a similar concept to that applied in the KAISER approach and in Microsofts' blog post about the matter, they say “This mitigation draws inspiration from prior research known as KAISER” [24]. In more technical terms, the kernel mappings for a process will only have limited functions such and enter/exit that are trusted. The two page tables are used alongside each other so if a process is only executing code in the user mode, the user-mode page table is used. If a process needs to execute code in the kernel mode, the code is trapped and executed inside the kernel mode page table on behalf of the process.

Additionally, Microsoft created a speculation barrier flag that was implemented within Visual Studio; the Integrated Development Environment developed by Microsoft for creating software for windows. The flag enables the identification of some areas of the code that may be susceptible to a speculative execution attack. The flag will identify these areas when compiling the code and can insert instructions that would prevent these executions from being exploitable. They then rebuilt any Windows software that would be potentially susceptible to such an exploit and pushed the update to users in the January 2018 security update [24]. However, this flag and the compiler cannot guarantee 100% coverage of these exploit scenarios so, in an attempt to improve the coverage they introduced a bounty program offering rewards to anyone who could find any instances of the exploit that remained [25]. Additionally, the Just in Time (JIT) compilers that operate within Microsoft Edge and Internet Explorer were both patched to provide mitigations to the exploits as well.

Outside of technical mitigations Microsoft developed, they also provided a selection of suggestions regarding writing software to mitigate against the attacks. One such suggestion is to ensure a program removes sensitive content from memory to ensure if a speculative attack were to occur, there would be no sensitive information for it to access. Microsoft claims that with 2016 era CPUs from intel such as Skylake, Kabylake or newer, the performance regression was single digit which would be negligible for the average consumer although again, a concern for larger industry infrastructure. Whilst not reporting exact numbers, a quote from a blog post claimed that 2015 era CPUs from Intel such as Haswell or older "show more significant slowdowns, and we expect that some users will notice a decrease in system performance." [26]. Perhaps more concerning is another quote from the same blog that claims, "Windows Server on any silicon, especially in any IO-intensive application, shows a more significant performance impact when you enable the mitigations to isolate untrusted code within a Windows Server instance". Due to Microsoft being a private company with closed source software and extensive business partnerships within both the consumer and, industry space, they have been less forthcoming with direct numbers. Microsoft claims that mitigations against variants 1 and 3 saw little performance regression and that the bulk of the impact was in their patch relating to variant 2. In May 2019, Microsoft released a security update for windows that included Retpoline; a mitigation strategy developed by google to prevent branch target injection. This mitigation was claimed to reduce the performance regression; especially the regression found in SSD read/write scenarios down to 1-2% from the previous 30%.

6.3 iOS and macOS

In January 2018, Apple released security updates to both macOS, an x86 based operating system for desktop and iOS, an ARM based operating system for iPhone and iPad. macOS operates on top of Unix which could indicate their mitigations are more in line with the Linux mitigations and iOS is based off macOS in some respects so it may be similar. Apple has not released the technical details of how they have implemented their mitigations, however, they did release a list of the security fixes implemented in the updates [27].

With these software mitigations fixes Apple claims the Meltdown mitigation resulted in "no measurable reduction in performance" for either macOS or iOS when testing using GeekBench 4, Speedometer, Jetstream and, ARES-6 [28] which are a collection of benchmarking tools that target various aspects of a PC to gain an understanding of the performance of the system. With regards to Spectre, Apple claims that at worst they saw a 2.5% performance regression which would be borderline negligible. Apple does not traditionally have a large foothold in larger IT infrastructure and as such the performance penalties generally apply to consumer-level hardware. However, Apple did suggest a solution for those in more security-sensitive positions that they claim offers "full protection". This solution was to disable Hyperthreading [29]. Hyperthreading is the branding Intel uses for SMT. The logic behind this solution is that the Spectre and Meltdown flaws both rely on simultaneous multithreading to perform the speculative execution and hence by disabling Hyperthreading, the speculative execution can never occur. This solution has also been proposed by Intel, Microsoft and, Linus Torvalds who is the creator of the Linux kernel.

However, the reason this solution is not generally recommended or implemented is due to it being generally unfeasible for most users. Disabling Hyperthreading has the potential for signification performance reductions that exceed that of the software mitigations that have been provided but would provide complete mitigation that would not require a hardware fix. As to whether this can be called a solution is debatable. It is not so much a fix as complete avoidance of the issue, however, it does provide "full protection" in a sense. According to Apple, the performance impact of disabling hyperthreading could be as high as 40% [29].

7. Conclusions

In summary, the technologies created to increase the performance of CPUs such as speculative execution have been implemented with a large oversight with regards to security. The Spectre and Meltdown vulnerabilities showed how hardware-level vulnerabilities could be exploited on modern CPUs, with potentially devastating consequences regarding sensitive data, both in the consumer space and the larger IT industry. With all proposed solutions bearing an impact on the performance of affected CPUs, there is a corresponding financial cost for the companies this performance regression affects. With looming technologies on the horizon such as RISC-V that seek to upset the status quo of the x86 and ARM duopoly, the CPU manufacturers will bear the brunt of this oversight as partnered companies will no doubt either seek reimbursement or reconsider their partnerships with the manufacturers going forward.

References

[1] TechTerms, "Clock Cycle," 2010. [Online]. Available: https://techterms.com/definition/clockcycle.

[Accessed 15 11 2019].

[2] S. Eggers, J. Emer, H. Levy, J. Lo, R. Stamm and D. Tullsen, "Simultaneous multithreading: a platform for next-generation processors," IEEE Micro, vol. 17, no. 5, pp. 12-19, 1997.

[3] Intel, "Deep Dive: Introduction to Speculative Execution Side Channel Methods," 2019. [Online]. Available: https://software.intel.com/security-software-guidance/insights/deep-dive-introduction-speculative-execution-side-channel-methods. [Accessed 11 11 2019].

[4] Wideskills, "C++ Control Flow Structures," 2015. [Online]. Available: https://www.wideskills.com/c-plusplus/c-plusplus-control-flow-structures. [Accessed 13 11 2019].

[5] Micron, "Speed vs. Latency (Whitepaper)," 01 05 2015. [Online]. Available: https://pics.crucial.com/wcsstore/CrucialSAS/pdf/en-us-c3-whitepaper-speed-vs-latency-letter.pdf. [Accessed 12 11 2019].

[6] J. Hruska, "How L1 and L2 CPU Caches Work, and Why They're an Essential Part of Modern Chips," 2018. [Online]. Available: https://www.extremetech.com/extreme/188776-how-l1-and-l2-cpu-caches-work-and-why-theyre-an-essential-part-of-modern-chips. [Accessed 11 11 2019].

[7] J. Horn, M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, A. Fogh, S. Mangard, P. Kocher, D. Genkin, Y. Yarom and M. Hamburg, "Meltdown: Reading Kernel Memory from User Space," 2017. [Online]. Available: https://meltdownattack.com/meltdown.pdf. [Accessed 11 11 2019].

[8] M. Rouse, "side-channel attack," 2019. [Online]. Available: https://searchsecurity.techtarget.com/definition/side-channel-attack. [Accessed 11 11 2019].

[9] J. Horn, W. Haas, T. Prescher, D. Gruss, M. Lipp, S. Mangard and M. Schwarz, "Spectre Attacks: Exploiting Speculative Execution," 2017. [Online]. Available: https://spectreattack.com/spectre.pdf. [Accessed 12 11 2019].

[10] J. Horn, "Reading privileged memory with a side-channel," 2018. [Online]. Available: https://googleprojectzero.blogspot.com/search?q=spectre. [Accessed 12 11 2019].

[11] Graz University of Technology, "Meltdown and Spectre," 2018. [Online]. Available: https://meltdownattack.com/. [Accessed 14 11 2019].

[12] ARM, "Vulnerability of Speculative Processors to Cache Timing Side-Channel Mechanism," 2019. [Online]. Available: https://developer.arm.com/support/arm-security-updates/speculative-processor-vulnerability. [Accessed 14 11 2019].

[13] P. Kocher, "Spectre Attacks: Exploiting Speculative Execution," 17 04 2018. [Online]. Available: https://www.rsaconference.com/usa/us-2018/agenda/spectre-attacks-exploiting-speculative-execution-4. [Accessed 12 11 2019].

[14] Oxford Dictionary of English, Mitigation, 3rd ed., Oxford: Oxford University Press, 2019.

[15] J. Edge, "Kernel address space layout randomization," 2013. [Online]. Available: https://lwn.net/Articles/569635/. [Accessed 13 11 2019].

[16] J. Corbet, "KAISER: hiding the kernel from user space," 2017. [Online]. Available: https://lwn.net/Articles/738975/. [Accessed 13 11 2019].

[17] Technopedia, "Trusted Computing Base (TCB)," 2019. [Online]. Available: https://www.techopedia.com/definition/4145/trusted-computing-base-tcb. [Accessed 13 11 2019].

[18] "syscalls - Linux system calls," 2019. [Online]. Available: http://man7.org/linux/man-pages/man2/syscalls.2.html. [Accessed 13 11 2019].

[19] M. Rouse, "context switch," 2012. [Online]. Available: https://whatis.techtarget.com/definition/context-switch. [Accessed 13 11 2019].

[20] Redhat, "4.4. Virtual Memory: The Details," N/A. [Online]. Available: https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/4/html/Introduction_To_System_Administration/s1-memory-virt-details.html. [Accessed 13 11 2019].

[21] P. J. Denning, "The Working Set Model for Program Behavior," Massachusetts Institute of Technology, Cambridge, Massachusetts, 1968.

[22] B. Jang, D. Schaa, P. Mistry and D. Kaeli, "Exploiting Memory Access Patterns to Improve Memory Performance in Data-Parallel Architectures," IEEE Transactions on Parallel and Distributed Systems, vol. 22, no. 1, pp. 105-118, 2011.

[23] B. Gregg, "KPTI/KAISER Meltdown Initial Performance Regressions," 2018. [Online]. Available: http://www.brendangregg.com/blog/2018-02-09/kpti-kaiser-meltdown-performance.html. [Accessed 13 11 2019].

[24] Swiat, "Mitigating speculative execution side channel hardware vulnerabilities," 2018. [Online]. Available: https://msrc-blog.microsoft.com/2018/03/15/mitigating-speculative-execution-side-channel-hardware-vulnerabilities/. [Accessed 13 11 2019].

[25] Microsoft, "Microsoft Bug Bounty Program," 2019. [Online]. Available: https://www.microsoft.com/en-us/msrc/bounty?rtc=1. [Accessed 13 11 2019].

[26] T. Myerson, "Understanding the performance impact of Spectre and Meltdown mitigations on Windows Systems," 2018. [Online]. Available: https://www.microsoft.com/security/blog/2018/01/09/understand-

ing-the-performance-impact-of-spectre-and-meltdown-mitigations-on-windows-systems/. [Accessed 13 11 2019].

[27] Apple, "About the security content of macOS High Sierra 10.13.3, Security Update 2018-001 Sierra, and Security Update 2018-001 El Capitan," 2019. [Online]. Available: https://support.apple.com/en-us/HT208465. [Accessed 14 11 2019].

[28] Apple, "About speculative execution vulnerabilities in ARM-based and Intel CPUs," 2018. [Online]. Available: https://support.apple.com/en-us/HT208394. [Accessed 13 11 2019].

[29] Apple, "How to enable full mitigation for Microarchitectural Data Sampling (MDS) vulnerabilities," 2019. [Online]. Available: https://support.apple.com/en-gb/HT210108. [Accessed 13 11 2019].