Software-Based Side Channel Attacks and the Future of Hardened Microarchitecture

Nathaniel Hatfield

A Senior Thesis submitted in partial fulfillment
of the requirements for graduation
in the Honors Program
Liberty University
Spring 2021

Acceptance of Senior Honors Thesis

This Senior Honors Thesis is accepted in partial
fulfillment of the requirements for graduation from the
Honors Program of Liberty University.

____
Michael Lehrfeld, Ph.D.
Thesis Chair

____
Melesa Poole, Ph.D.
Committee Member

____
David Schweitzer, Ph.D.
Assistant Honors Director

____
May 5, 2021

Abstract

Side channel attack vectors found in microarchitecture of computing devices expose systems to potentially system-level breaches. This thesis consists of a comprehensive report on current exploits of this nature, describing their fundamental basis and usage, paving the way to further research into hardware mitigations that may be utilized to combat these and future vulnerabilities. It will discuss several modern software-based side channel attacks, describing the mechanisms they utilize to gain access to privileged information. Attack vectors will be exemplified, along with applicability to various architectures utilized in modern computing. Finally, discussion of how future architectural changes must successfully harden chips against attacks of this type will occur, ending with a reinforced call for development of these integral architectural revisions to resolve the threat.

Keywords: Microarchitecture, Architecture, Speculative Execution, Side Channel Attack, Vulnerability, Exploit, Spectre, Meltdown, ZombieLoad, Foreshadow, Fallout, Intel, AMD, ARM, Apple Silicon, x86-64

Software-Based Side Channel Attacks and the Future of Hardened Microarchitecture

Introduction

Computers today are incredibly advanced. Each year, Central Processing Unit (CPU) manufacturers release new versions of their computing products which are faster, more tangibly refined, and purportedly more secure. However, no products are without their flaws. Critical side channel exploits found in many modern chips have entered the public eye over the past few years, resulting in unauthorized disclosure of private data. Even though (generally speaking) the exploits themselves are software implementations, side channel attacks are classified as hardware vulnerabilities. This is because they take advantage of flaws inherent in the architecture of the CPU of the computer. CPU microarchitecture is referred to as the hardware implementation of the instruction sets a processor can execute. “For example, x86-64 is the [instruction set] used by most modern laptop and desktop computers” [1]. This instruction set shares commonalities in each implementation such that many CPUs currently produced for application in the commercial and consumer domains are vulnerable to critical side channel attacks. These vulnerabilities have the potential to expose private user data to malicious actors, undermining the security, trust, and reliance upon computing that has become the norm today.

Modern Software-Based Side Channel Attacks

Side channel attacks induce exploitable hardware vulnerabilities in otherwise ostensibly secure systems; these attacks are considered software-based exploits of hardware architectural vulnerabilities. The numerous exploits of this type that have been most prevalent recently in the security sphere have been made possible by a computing feature known as speculative execution. This is a performance optimization technique utilized in modern processors. Speculative execution occurs when the processor executes instructions it knows it will eventually need to complete in advance of being asked to do so [2]. It removes the requirement for previous instructions to complete before executing new ones and increases performance by reducing latency and increasing parallelism [3]. The potential drawback is that speculative execution can allow the CPU to “speculatively” and completely unknowingly execute code that may be malicious.

Speculative Execution Exploits

Speculative execution is a vital performance optimization technique utilized in a suite of modern processors, not just chips manufactured by Intel. However, an unintended complication of speculative execution is creation of avenues for side channel attack methods. This allows for malicious exploits to access privileged data on running processes and programs that would otherwise be kept private. Such information is normally safe within main memory, but these exploits allow attackers to expose a covert side channel, enabled through flawed architecture design, providing unauthorized access to data as it is loaded into the CPU cache and registers.

Side channel attacks can and have been demonstrated to be capable of stealing passwords, encryption keys, and more, in both lab and real-world environments. One such exploit, nicknamed “Meltdown” by the security researchers who uncovered it, allows attackers to effectively remove the restrictions placed on memory addresses of different programs. Bad actors can extract data from these memory areas that they otherwise would not have access to [4]. “Spectre,” another side channel attack enabled by speculative execution, allows rogue programs to fool legitimate ones into sharing private information voluntarily [5]. These are but a few examples, and each have multiple variants designed to exploit specific architectural vulnerabilities via unique side channels, some repairable and some not. The root commonality among all such exemplified exploits is a covert, architecturally induced side channel, typically in the form of speculative execution.

Exposing Side Channels

What exactly is a side channel? The term itself is innocuous enough; merely referring to data overlap such that information can be identified via methods other than the primary delivery mechanism. In this context, "…a side channel is an unintended pathway that leaks information from one entity to another (usually both are software programs), typically through a shared resource such as a hard drive or memory." As a generic example, consider the information transferred to a page via a printer. A side channel would be a separate recording device which could interpret the data the printer was printing from sound alone, not through observing the printed page [6]. Speculative execution creates incidental side channels, allowing speculatively executed instructions to read from CPU caches, registers, Translation Lookaside Buffers (TLBs), microarchitectural predictors, and prefetchers [2]. The security researchers who identified the Meltdown exploit describe out-of-order execution as a "new, extremely powerful software-based side channel," [4], and variant 2 of Spectre creates a return-oriented programming (ROP) cache side channel [5].

Speculative Execution Deep Dive

To better understand how speculative execution exploits are possible, a breakdown of how speculative execution functionally works must be conducted. "When programmers (or [the] compiler) write assembly code, they make the assumption that each instruction is executed before execution of the subsequent instruction is begun. This assumption is invalidated by pipelining," [3]. Pipelining is an optimization that has been present in CPUs for decades and is responsible for much of the non-silicon-based performance improvements that have occurred over time. At their core, assembly instructions have not changed in a tangible manner since the x86_64 instruction set was first defined. Implementations of pipelining include multithreading and Instruction Level Parallelism (ILP) and are responsible for significant increases to Instructions Per Clock (IPC), a common metric of processor performance [3].

As described above, parallelism is still constrained by a program's control flow. Even if there are available resources to pipeline a program's instructions, it is often true that consequent instructions rely upon the result of an instruction currently being executed. Because of this, early implementations of parallelism made processors exponentially better at executing multiple programs simultaneously, without necessarily improving the performance of a single program, bound by control flow and limited ability to conduct parallel compute. This is still true today, especially in software applications like video games, where game logic is run on a single core and often cannot make use of multiple threads in an effective manner. Other constraints include resource conflicts, along with nontrivial data and control dependence [3]. Speculative execution is the proverbial golden ticket, enabling parallelism to occur where otherwise it could not. "Speculative execution is one of the main techniques used by most modern high performance processors to improve performance," [2].

Branch prediction is a simple and common method of implementing speculative execution, and involves analyzing the control flow of a program, identifying and predicting the outcomes of branches in the program logic. "Speculative execution is the processor's ability to execute instructions that exist beyond a conditional branch that has not yet been resolved, and ultimately to commit the results in the order of the original instruction stream" [3].

Rather than waiting for each branch instruction to resolve, extra processing cycles found on idle cores, integer, or floating-point engines can be dedicated to executing what is predicted to be the next instruction. Hence, the requirement for previous instructions to complete before new ones are executed, normally slowing process flow, is bypassed. As these predictions have been greatly optimized over time, their accuracy is exceptional. Branch prediction and other speculative execution optimizations significantly reduce latency in program control flow, resulting in massive performance gains. Speculative execution is built into architecture through hardware, the implementation of which is composed of branch prediction, dynamic scheduling, out-of-order execution, machine state commitment, and robust exception handling. "By executing instructions speculatively, performance can be increased by minimizing latency and extracting greater parallelism. The results may be discarded if it is discovered that the instructions were not needed after all," [2]. However, speculative execution may expose a vulnerable state by inducing a processor to speculatively execute malicious code, performing an exploit completely without user or process knowledge.

The Ring Zero Rabbit Hole

Central to processor functionality is the interaction between main memory, where programs are stored in runtime, and its cache and registers, where instructions are written into and read from during their instanced execution. To the processor, all memory and all caches are viable address space [2]. This is not the case for all programs that enter runtime, because of the concept of least privileged execution. Programs are typically only allowed to access information in memory that they have allocated or requested access to and been granted by an application running at a lower level. These computing "levels" are actually hierarchical protection contexts to isolate privileges in program execution and are commonly described as "protection rings." These rings most typically range from ring three to ring zero (see Fig. 1). Ring three is the least privileged and refers to user applications; the subsequent rings two and one refer to device drivers with increasing privileges. Ring zero is where the operating system kernel runs, holding the highest privileges on a system. The protection ring concept is relevant to speculative execution-based side channel attacks in that they bypass this least-privileged security concept completely, as speculatively executed instruction error handling does not occur until later in the program control flow/pipeline and is therefore already at or beyond the kernel level [6].

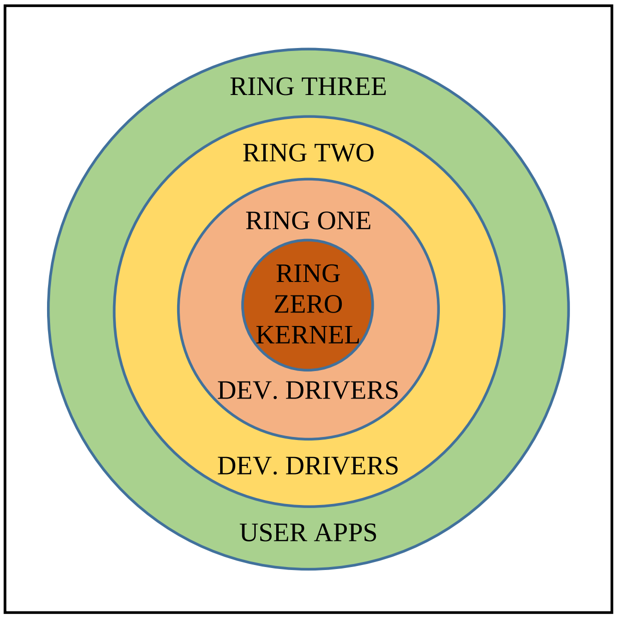


Fig. 1: Computing Protection Rings.

Negative Rings: Follow the White Rabbit

There are also conceptual negative rings, but as these are architecture specific and only found in Intel processors, they will not be discussed at length in this research [7]. However, it is important to note that some vulnerabilities exposed through Intel's negative rings also make use of side channel exploits. Intel's negative rings were created as firmware security implementations – nevertheless, as these firmware implementations run at a level of privilege higher than the kernel itself, they expose further opportunity for exploitation. Intel's defensive firmware tools are the Intel Management Engine (IME), Active Management Technology (AMT), and Software Guard Extensions (SGX). IME consists of firmware loaded on an embedded microcontroller, running its own microkernel and operating system at a sub-kernel level, supplying features like anti-theft protection [8]. Intel's SGX provide(s) a set of contextual runtime extensions to CPU architectural instructions, purportedly protecting data even in the event of partial or even complete breaches of trust in software [9].

Security researchers classify IME as Ring -3. Ring -1 threats occur in the "hypervisor" level, one below the ring 0 kernel, and ring -2 threats occur in a unique CPU state called System-Management-Mode (SMM). Exploits which address these negative rings and proprietary Intel firmware security technologies already exist, and as discussed, are based in the same speculative execution side channels as Meltdown and Spectre. Foreshadow is an advanced derivative of Meltdown that has two variants: the first extracting data from a side channel in SGX enclaves, and the second affecting virtual machines, the hypervisor (Ring -1), along with both kernel (OS) and SMM memory [10].

Speculative Execution: The Security Trade-Offs

Performance advantages to modern computing provided by speculative execution are greatly overshadowed by the many drawbacks, those shortcomings taking the form of attacks like Meltdown, Spectre, and the many others that have come to light recently. With increasing competition in the performant architectural development space, chip giants like Intel, AMD, Qualcomm, and more recently, Apple, the overwhelming drive has been toward process shrinks and increases in transistor density. With companies progressively jockeying for who gets to hold the performance crown, if even briefly, perhaps some corners have been cut, or edge cases not investigated, allowing speculative execution-based exploits to rise to prevalence.

> We performed experiments on multiple x86 processor architectures, including Intel Ivy Bridge (i7-3630QM), Intel Haswell (i7-4650U), Intel Broadwell (i7-5650U), Intel Skylake (unspecified Xeon on Google Cloud, i5-6200U, i7-6600U, i7-6700K), Intel Kaby Lake (i7-7660U), and AMD Ryzen. The Spectre vulnerability was observed on all of these CPUs. Similar results were observed on both 32- and 64-bit modes, and both Linux and Windows. [5]

As noted above, these types of vulnerabilities have been identified in nearly all consumer chips up to certain generations, widely deployed today in applications from internet of things (IoT) to smartphones, personal computers, and even large-scale datacenter deployments.

Side Channel Exploits Characterized

To better illustrate side channel exploits, consider this allegory: a strategic game of cat and mouse. Except, in this case, the mouse is much smarter than the cat. The cat is merely an obstacle found between the mouse and his cheese. The cat exhibits predictable behaviors on a repeatable basis. Therefore, the mouse knows what the cat will do if certain conditions exist, or if presented with information in advance of it occurring – just like speculative execution. The mouse will repetitively mistrain the cat to make an erroneous, speculative prediction of where the mouse will emerge, leaving the avenue to the coveted cheese wide open. A CPU, like the cat in the example, is mistrained by malicious software to speculatively execute exploitable instructions. “The attacker, from its own context, performs a mistraining of the branch predictors to trick the processor into speculatively executing the gadget when it runs the victim code,” [5]. The private data the attacker desires can then be obtained as the CPU is in an exploitable state by timing the latency of addressing pieces of memory in the CPU cache [2], like a hyper-intelligent mouse timing a cat’s comings and goings to triangulate exactly where the cheese is most likely to be. Amusing images of the classic Tom and Jerry cartoons may come to mind, but the illustration paints a chilling picture.

More technically speaking, mistraining is another form of misspeculation – what occurs when an instruction that did not need to (or should not have) run, actually does. Typically, this isn’t a big deal; as mentioned previously, “…the results may be discarded if it is discovered that the instructions were not needed after all,” [2]. Indeed, the results of speculatively executed instructions are not even directly observable. However – latency between instruction executions is observable, measurable, and predictable. Covert channel senders in malicious code implementations can observe and report on the latency of memory reads and writes to and from the CPU registers, cache, main memory, and hard disk, resulting in an unintentional leakage and disclosure of the information being accessed in speculatively executed code [4].

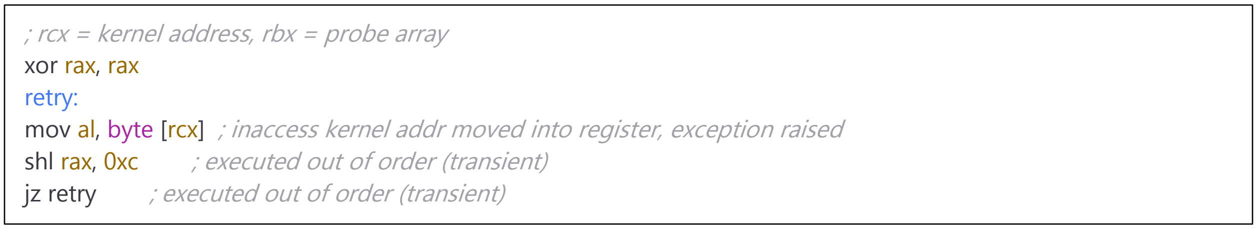
Meltdown and Spectre

Meltdown allows attackers to remove safeguards on memory addresses of different programs. This would enable a malicious program to gain access to another program’s memory and make them vulnerable to stealing passwords, encryption keys, and other personal data in both lab and real-world environments. Fundamentally, Meltdown’s exploits are reliant upon three strategies: out-of-order execution, known address spaces, and cache attacks. Within

Meltdown's context, speculative execution is known as a transient instruction. Its authors wrote, "…transient instructions introduce an exploitable side channel if their operation depends on a secret value." This secret value could be a password, key, or some other form of information it would be desirable for an attacker to obtain. More importantly, this data could be located in a user-inaccessible memory page, cache, or register, and would still be accessible through the Meltdown exploit [4].

Meltdown's operation can be generalized into three steps. First, a program running on the target system must load the content of a memory location the attacker wishes to read into a CPU register. Second, a transient instruction will access something in the cache, utilizing the secret content which has now been moved directly into a CPU register. Third, the attacker will issue flush, execute, and reload instructions (which they may do without elevated privileges), which will allow them to identify the accessed location in the cache, which in turn allows a dump of the targeted memory location. This process may be repeated ad nauseum, such that a system's entire physical memory may be dumped given enough time [4].

The assembly code snippet found below forms the core of the Meltdown exploit [4]:



```
; rcx = kernel address, rbx = probe array
xor rax, rax
retry:
mov al, byte [rcx]  ; inaccess kernel addr moved into register, exception raised
shl rax, 0xc        ; executed out of order (transient)
jz retry       ; executed out of order (transient)
```

Spectre enables rogue programs to trick legitimate, privileged applications into sharing private information voluntarily. As a result, Spectre is much more reliant on the mistraining instruction concept; misleading highly advanced and optimized branch prediction and speculative execution technologies to incorrectly predict the outcome of a given instruction.

Dangerously, some Spectre attack vectors can even be performed in languages like JavaScript, meaning all it could take to fall victim to a Spectre attack would be to visit a malicious website requiring no user interaction to trigger [5].

There are multiple variants to the original Spectre exploit, split into two main categories. The first category exploits conditional branch misprediction, and the second leverages controlled misprediction of the targets of indirectly executed branches. Variant one, a derivative of the conditional branch type, can be implemented in JavaScript to read private memory from the browser process, or in any more conventional compiled x86 language to read from any memory address on a system. The unoptimized Proof of Concept (PoC) code written in C by Spectre's researchers could read roughly 10KB/s on a low-power 4th generation mobile i7 processor with an error rate less than 0.01% [5].

JavaScript implementation of Spectre "variant one" [5]:

```
if (index < simpleByteArray.length) {
    index = simpleByteArray[index | 0];
    index = (((index * 4096) | 0) & (32 * 1024 * 1024 – 1)) | 0;
    localJunk ^= probeTable[index | 0] | 0;
}
```

Disassembly of JavaScript implementation [5]:

```
cmpl r15, [rbp-0xe0]          ; Compare index (r15) against simpleByteArray.length
jnc 0x24dd099bb870           ; If index >= length, branch to instr after movq below
REX.W leaq rsi, [r12+rdx*1] ; Set rsi=r12+rdx=addr of first byte in simpleByteArray
movzxbl rsi, [rsi+r15*1]     ; Read byte from addr rsi+r15 (= base address + index)
shll rsi,12                  ; Multiply rsi by 4096 by shifting left 12 bits
andl rsi, 0x1ffffff          ; AND assures JIT that next op is in bounds (it isn't)
movzxbl rsi, [rsi+r8*1] ; Read from probeTable (this is the malicious read)
xorl rsi,rdi                 ; XOR the read result onto localJunk
REX.W movq rdi,rsi           ; copy localJunk into rdi (now accessible in user space)
```

This is just one of the four Spectre variants. Alternative implementations rely on poisoning indirect branches that are known to be executed in the control flow based on speculative execution. However, all attacks, like Meltdown, can be broken into three phases. First comes the setup phase, wherein an adversarial program or code snippet performs instructions that progressively mistrain the processor’s speculative execution model within a given context. Speculative execution may be induced more frequently than it would be otherwise in this phase to get the processor to a desired cache and control flow state. In some cases, a covert channel for exfiltration of data is also established in this phase [5].

Phase two involves the erroneous speculative execution. In this phase, the attacker takes advantage of an architectural flaw to transfer confidential information into the pre-established covert channel. This exploit can be triggered by either the attacker or the user (unknowingly, in the case of the user). The most common target of this phase is a memory address known to the attacker that contains desirable private information to be extracted. The third and final phase is just like Meltdown, wherein the attacker triggers a flush+reload or evict+reload to exfiltrate the identified data, timing the access to memory addresses based on monitored latencies during phase two [5].

ZombieLoad

Speculative execution-based attacks are certainly the most common but are by no means the only side channel attacks made possible by architectural flaws which have emerged in recent years. ZombieLoad is one such side channel attack. It affects Intel CPUs two generations newer than Spectre and Meltdown but does not apply to ARM, AMD, or other x86_64 silicon due to differences in architecture. Specifically, Intel’s specific cache fill buffer mechanism is what is architecturally vulnerable. ZombieLoad is known as a data sampling attack, taking advantage of leaked data from the CPU’s fill buffer accessed either on the same or a sibling hyper-thread. The attack executes transient instructions designed to observe values of memory loads/stores on a given CPU core, exploiting the fact that the fill buffer used by logical cores within a physical core do not distinguish between processes or privileges to access data. An exploit can be crafted to leak values from any concurrently running application to another, across user-space, kernel-space, SGX, and even the hypervisor (virtual machines). However, identifying how ZombieLoad works is a bit more tenuous than Meltdown or Spectre, which have clear architectural behaviors in speculative execution which explain why the behavior is possible. Even the exploit developers only have hypotheses as to the underlying architectural flaw that enables ZombieLoad’s vulnerability [11].

To facilitate the exploit, a “zombie load” is requested via a transient instruction; the load is dubbed such because the data being requested may not actually exist in the given context. This load triggers an assist in the microcode of the processor, such that the load is designed to contain incorrect data, so that it will fail. However, the fault state will read as an incomplete load, meaning the processor will schedule an automatic re-issue of the load at a later time. When the re-issued load occurs, it does so in the correct context, leaking otherwise private data through the fill buffer side channel. ZombieLoad enables cross-VM covert channels at a rate of 26.8KB/s with a net 0% error rate. However, this exploit is limited in what it may target, as it is a one-dimensional side channel; this means that leakage can occur solely from running processes, and a specific targeted address is not achievable. Malicious data retrieval is instead achieved through data sampling at a larger scale, dumping mass memory regions in an iterative fashion [11]. In the case of ZombieLoad, the limiting factor for what can be extracted is simply time.

The clear and present danger making ZombieLoad a concern even today is its viability on both older and newer Intel platforms, along with its lethal ability to breach the virtualization barrier, which was not possible with exploits like Spectre and Meltdown. ZombieLoad has had the most discernible negative effect in datacenter and large-scale virtualization deployments, where data that was previously thought to be protected through virtualization is no longer considered safe. Further, the patches which enable “protections” against ZombieLoad incur massive performance penalties, especially in virtualized scenarios [12].

Most damagingly, an embargo was lifted on January 27th, 2020, wherein the ZombieLoad security researchers disclosed that the existing MDS patches and microcode updates were not sufficient to patch ZombieLoad on any vulnerable processors. At time of writing (Q1 2021), the L1D (CPU Level 1 Data cache) eviction sampling exploit method still enables ZombieLoad to leak data in a maliciously advantageous way. As such, the security researchers for ZombieLoad have not disclosed this particular attack vector, and it is unconfirmed if it has been exploited in the wild [12].

Side Channel Attack Risk Factors

Since speculative execution is utilized in virtually all modern chips, whether they are manufactured by Intel, Advanced Micro Devices (AMD), or in the fundamentally different ARM architecture [6], they all share commonalities specific to 64-bit architectures. This makes side channel attacks like Meltdown, Spectre, and others at the very least conceptually viable. Even Apple’s innovative custom M1 processor and future Apple Silicon chips may be vulnerable, due to the fact that they are 64-bit and ARM-based [6]. Apple Silicon vulnerability also depends on its implementation of branch prediction, due to an architectural inclusion of the x86 memory consistency model and Total Store Ordering (TSO) [13].

Side channel attack-based exploits pose a clear and present danger. As demonstrated by variants of Spectre and Zombieload which are still viable under certain circumstances today [12], [14], you cannot make architectural revisions to existing hardware. Chips vulnerable to these exploits are employed in all levels of industry, from consumer to enterprise, and have been for years. Especially with the widespread use of virtualization (ZombieLoad bears particular relevance to large-scale datacenter applications due to its ability to breach the VM barrier) and industry focus on cloud hosting, the capacity for malicious actors to make use of these exploits, inducing data breaches and accessing valuable, private information – is extreme.

Side Channel Side Effects

The underlying concern raised by these vulnerabilities is that they are implicitly enabled by latent flaws in the hardware architecture of the chips. For instance, in the ZombieLoad exploit, the hardware pathways facilitating the memory-to-cache fill buffer expose a side channel across peer hyper-threads [11]. Given that this pathway exists in hardware, not software, no alternative avenue is available. Software patches designed to mitigate this class of exploits often measurably reduce the performance of the chips they are implemented on. In March of 2019, Red Hat published an article surveying speculative execution exploit performance impacts for the Spectre and Meltdown patches, ranking them from minimal (<1%) to measurable (8-19%) in various bare metal compute workloads exemplified as typical on Red Hat systems. In particular, memory intensive workloads were observed to have suffered the most meaningful performance impact. However, it was noted that heavily virtualized applications would suffer more drastic impacts than measured in this study. More significant performance impacts most assuredly exist, affecting primarily the data center industry, in a highly scenario-dependent (and therefore not easily testable) fashion [15].

At time of writing (Q1 2021), Meltdown has been fully patched by Intel through software for older generations, and in hardware through newer generations. Fortunately, AMD silicon is considered architecturally unaffected by all Meltdown variants. However, Spectre's core vulnerability cannot be fully patched by either chipmaking giant in its existing CPUs, because its particular exploit avenue takes advantage of a logical flaw which does not have a software resolution [16]. More frighteningly, these exploits aren't going anywhere anytime soon. Even Intel's hardware fixes are considered workarounds, merely a physical implementation of a software patch, incurring the same performance deficits [6]. Long-term mitigation of speculative execution and side channel attacks will require a congress of hardware architectural revisions, trusted firmware, and software protections now and into the future: a full systems approach.

Ongoing Threats

Exemplifying the truly band-aid (read: temporary, not a true long-term solution) nature of both hardware and software mitigations is the Fallout exploit, which was disclosed in 2019 and affects current 10th generation Intel processors which are not vulnerable to Meltdown or Spectre. "Short-term software mitigations for Meltdown are only a temporary solution with a significant performance overhead. Due to hardware fixes, these mitigations are disabled on recent processors…we show that Meltdown-like attacks are still possible on recent CPUs which are not vulnerable to Meltdown." Fallout takes advantage of yet another microarchitectural vulnerability in the form of the store buffer on Intel processors, which enables a side channel allowing data forwarding from unvalidated memory addresses. Fallout attacks have proven to be viable in causing data leakage, recovery of faulted control flow, and even attacks on Address Space Layout Randomization (ASLR), a security feature designed to randomize how applications store data [17].

Fallout's security research team rightfully posited the following question: "Are new generations of processors adequately protected against transient execution attacks? If so, can ad-hoc software mitigations be safely disabled on post-Meltdown Intel hardware?" They had a swift and ready answer in their research: a resounding no. Fallout's method of exploitation, write transfer forwarding, enabled a renewed application of previously mitigated Spectre exploits when applied in a fatal combination [17]. The existence of the Fallout exploit as another progressively evolved threat from the same foundational basis certainly raises concerns, especially given that it utilizes an architectural side channel that had been considered completely patched through hardware and software mitigations.

Beyond the known is the terrifying unknown. The discovery of architectural side channel vulnerabilities like Meltdown, Spectre, Fallout, and more suggests the existence of as-yet unidentified, or non-publicized exploits. Some may already have exploitable code "in the wild," utilized without the awareness of system administrators or even cutting-edge security researchers. Consider variant two of Zombieload (L1D eviction sampling leakage): though the embargo against releasing the exploit's details was lifted on January 27, 2020, the security team who discovered it still refuse to release proof-of-concept code, as no patches to fix the underlying vulnerability are available [11]. Unfortunately, unlike the Google Project Zero team (who contributed heavily to the discovery of the Meltdown and Spectre exploits), not all those who discover side channel exploits like these consider the ethical ramifications [15]. Cyber-crimes are lucrative – if they were not, the landscape of the computer security industry would look vastly different. It would be pure foolishness to consider the exploits of this nature discovered thus far to be the only such vulnerabilities to exist.

Future of Hardened Microarchitecture

Since these vulnerabilities are architecturally inherent to the hardware design of the CPUs, will it ever be possible to patch speculative execution bugs in all exploitable chips? This question seems especially relevant today, as so-called "security measures" like Intel's SGX and secure enclaves have often introduced more vulnerabilities of late than they have mitigated [15]. Speculative execution, though a core functionality for enhancing performance in modern CPUs, is a hot topic for re-evaluation due to the many vulnerabilities that have continued to rear their ugly heads of late. Just in the past two years, attacks like ZombieLoad, which makes use of an architectural flaw in Intel's store buffer [11], and flaws in Intel's CSME (Converged Security Management Engine) present in every Intel processor from the past six years, have undermined the long reputation of trust and security that Intel has long built their brand upon [18].

Dependence and Trust Development

Attackers of all natures often greatly benefit from dependence – a Merriam-Webster definition of trust [19]. Dependence may be considered use of a system which, whether trusted or not, is reliant upon the degree at which the user acquires a sense of security (which may or may not be well-founded), also known as complacency. Quantifying the source of this complete dependence involves understanding that the user has limited choice in whether they use a system or not. Developing a better understanding of trust within the subject context, user dependence upon processor systems must be quantified. In the case of side channel vulnerabilities resultant from architectural flaws, the root of trust is the processor, and the exploits are a critical breach of that trust. Dependence is exemplified in the widespread deployment of vulnerable processors. As the security researchers who discovered Meltdown wrote in 2018, “Desktop, Laptop, and Cloud computers may be affected by Meltdown. More technically, every Intel processor which implements out-of-order execution is potentially affected, which is effectively every processor since 1995 (except Intel Itanium and Intel Atom before 2013). We successfully tested Meltdown on Intel processor generations released as early as 2011,” [15].

Proliferation of any technology will result in a corresponding spike in interest for development of exploits in that field. Apple’s Mac OS X operating system certainly has a good reputation and has historically been known for its security and stability, but this cannot be credited exclusively to the proficiency of Apple’s developers and security researchers. As of January 2021, OS X (across its various versions), only holds a 7.01% global market share, compared to the 31.6% held by Windows (also across all versions). Apple’s mobile operating system, iOS, holds more than double, at 16.89% [20]. Exploit developers will be much more successful in targeting so-called “low-hanging fruit;” that is to say, more commonplace targets. The law of averages is on their side, especially when taking human factors into consideration.

When a minute 7% share is held by an operating system, compared to greater percentages in others, which will provide the greater return on investment for threat actors developing exploits? In the case of side channel attacks, the ubiquity of x86_64 processors with reliance on speculative execution and other exploitable architectural quirks make it very likely that these attacks will not be the last of their kind. The only way forward is through innovation, to architecture, and potentially even the underlying core. The x86_64 assembly and instruction set has existed for over twenty years, and at the end of the day, is simply not designed for the hugely parallel processing now possible with CPUs that release with more cores and threads every year.

There must be critical awareness of a whole systems view for CPU development and architectural analysis. This is essential to conceptualize at a granular level the many interactions between subsystem components (caches, registers, instruction sets, speculative execution) – inadequate understanding of these interactions is what enabled side channel vulnerabilities in the first place. More importantly, such a view informs the levels of trust that may be established and maintained for each component. After all, a system is only the sum of its parts – and the user must be considered a part of the system. Many users have little awareness of vulnerabilities, and do not even know (or care) to update device software, firmware, or hardware, even when it may be vital to do so. In point of fact, research has revealed that the majority of consumers who are non-experts in the fields of computer science and cyber security simply do not update systems at all, “perhaps due to lack of understanding of their effectiveness or bad past experiences caused by software updates,” [21].

Architectural channels are subsystems within a processor system (design) with high levels of dependence. Whether trusted or not, understanding the role of these subsystems and quantifying the degree to which dependence exists is crucial to effectively mitigate the threats brought about by reliance. Trust must be architected and maintained at every system’s level and design, across its full life span. Are speculative execution and out-of-order execution singularly necessary components to achieve improved processor performance? From a dependence perspective, these are either zero or single point of failure systems, destined to trigger whole system collapse.

Silicon Mitigation Strategies

To truly mitigate these vulnerabilities, key architectural changes are necessary – which poses quite a problem for companies like Intel, whose legacy of architecture has made them billions in both their own manufacturing and in patent licensing. Intel has been buffeted by a storm of these vulnerabilities recently, which have often had fewer or no applications on competing architectures. AMD's new Ryzen architecture has required far fewer patches to mitigate these side channel attacks and has even been proven to be architecturally immune to some variants of Spectre and all variants of Meltdown, due in no small part to its more recent architectural development basis. Spectre/Meltdown vulnerability variant one (bounds check bypass) was identified as fully software-patched for Ryzen processors with a negligible performance impact. Variants two and three (branch target injection, rogue data cache load) are not applicable to AMD processors at all due to architectural variations between Intel and AMD chips [22].

It has already proven more difficult for Intel to evolve their microarchitecture to combat these attacks than AMD, as their low-level core architecture has been reused year-over-year for nearly a decade. Some of Intel's patches have merely been to processor microcode, only introducing a behavioral alteration. This is not likely to change any time soon, as even recent Intel processors with supposed hardware mitigations (Table I) still prove vulnerable to some variants, while also incurring up to a 60% reduction in performance in synthetic, worst-case load tests based heavily in I/O computation [23]. Side channel attacks are not a new concept, but

TABLE I
SPECTRE AND MELTDOWN MITIGATIONS ON INTEL PROCESSOR FAMILIES

Processor Family			(CFL-R)	(CLX-AP)	(WHL-U)	(SKX-R)	(AML-Y)
Spectre	Variant 1	Bounds Check Bypass	OS/VMM	OS/VMM	OS/VMM	OS/VMM	OS/VMM
Spectre	Variant 2	Branch Target Injection	SW + FW	SW + HW	SW + FW	SW + FW	SW + FW
Meltdown	Variant 3	Rogue Data Cache Load	Hardware	Hardware	Hardware	Firmware	Firmware
Meltdown	Variant 3a	Rogue System Register Load	Firmware				
Meltdown	Variant 4	Speculative Store Bypass	Software + Firmware				
Meltdown	Variant 5	L1 Terminal Fault	Hardware	Hardware	Hardware	Firmware	Firmware

exploiting architectural design certainly is. The way forward is to implement hardened microarchitectures that are designed with these vulnerabilities in mind; whether that means abandoning speculative execution, seeking performance gains elsewhere, architecting hardware safeguards to ensure data privacy and integrity are upheld, or something else entirely has yet to be determined. These are vital questions that must be investigated with further research.

Non-Silicon Mitigation Strategies

Beyond silicon enhancements and manufacturing considerations, architectural modules can be built in an additive fashion to improve hardware security. These are components like Trusted Platform Modules (TPMs), which enables full-disk encryption on a system [24], Virtual Machine Extensions (VMX), which provides a mechanism for virtualization of software or even full operating systems [25], and Intel Trusted Execution Technology (TXT), which allows applications and services to run sandboxed in software through architectural features [26]. Apple’s Secure Enclave, implemented in many recent devices sold by the company across their mobile, desktop, and accessory product lines within the system on chip (SoC) or via an add-in module (the T1/T2 security coprocessor), is a shining example of how an external hardware component can directly enhance system security:

> The Secure Enclave is a secure coprocessor that includes a hardware-based key manager, which is isolated from the main processor to provide an extra layer of security…Secure Enclave also maintains the integrity of its cryptographic operations even if the device kernel has been compromised. Communication between the Secure Enclave and the application processor is tightly controlled by isolating it to an interrupt-driven mailbox and shared memory data buffers. [27]

Much of the latency and performance overhead resultant from side channel exploit patches have been due to the added checks verifying that code to be speculatively executed is not malicious. This interrupts program control flow and reintroduces the same latencies and reduction of parallelism that speculative execution and branch prediction was originally intended to reduce. A novel security coprocessor, either on-die or as a separate interconnected module, could be developed with direct access to main memory along with the various CPU caches and registers. This module would be responsible for arbitration of branch prediction, speculative execution, and integrity checks of running processes, and could be architecturally designed to excel at this particular type of computation. This work could be offloaded from the CPU to this coprocessor, integrating a secure hardware revision and increase to performance in one fell swoop. Coprocessors of this kind can be prototyped using FPGAs (Field-Programmable Gate Arrays), reprogrammable processors purpose-built for testing architecture and instruction set development.

Conclusion

While Meltdown, Spectre, ZombieLoad, and other side channel attacks may have just been blips on the radar to the average consumer, software developer, or even system administrator, they should really be held at a much higher level of continued concern. Survey the aftermath of these exploits: processors which had already entered mass production and the consumer market when these exploits were disclosed have been patched, but at the cost of significant performance deficits in certain workloads [6]. Such software and microcode fixes are band-aids, and do not deal with the root of the problem. Hence, Spectre's root vulnerability remains unpatched [15], ZombieLoad is still viable on up to 9th generation Intel processors in both consumer and enterprise domains, while the existing insufficient patches incur performance hits in heavily virtualized deployments. Further, Fallout has emerged as a viable side channel attack on current Intel silicon despite Meltdown hardware revisions [17]. Despite their profoundly negative impact, the discovery and subsequent analysis of these exploits should be considered a net positive. Never before has the architecture development industry been so scrutinized and well-researched in terms of security and trust development.

Taking the undisputed lead in architectural innovation and silicon development is the singular clear path to a future where it is not a given that the computers used every day by individuals, businesses, and government organizations are vulnerable to some devastating form of side channel attack. Side channel attacks and vulnerabilities inherent in CPU architecture are a very relevant topic for today and will be into the foreseeable future. The next generations of secure microarchitecture cannot merely be iterative improvements with band-aid fixes; they will rely on hardening designs against such attacks at an architectural level, with a full-systems view of how each component works together in congress to create a secure, performant, and trusted platform. Should the utilization of speculative execution be re-evaluated to promote security, despite the performance benefits it provides? Can those performance benefits be found elsewhere? Further research will illuminate the differences between the various side channel attack methods and aid discovery of strategies that may be employed in the security endeavors of chipmakers for years to come.

References

[1] "What is Microarchitecture?" ComputerHope. [Online]. Available: https://www.computerhope.com/jargon/m/microarchitecture.htm.

[2] Introduction to Speculative Execution Side Channel Methods. [Online]. Available: https://software.intel.com/security-software-guidance/deep-dives/deep-dive-introduction-speculative-execution-side-channel-methods.

[3] C. Dewan, "Study of Speculative Execution and Branch Prediction," M.S. report, Dept. Comp. Sci. and Eng., Indian Inst. of Tech. Bombay, Bombay, India, 2006. [Online]. Available: https://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=EBE980ABF71E4B8C0055F14D3DDAC3F2?doi=10.1.1.119.2934&rep=rep1&type=pdf.

[4] M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, A. Fogh, J. Horn, S. Mangard, P. Kocher, D. Genkin, Y. Yarom, and M. Hamburg, (Aug. 2018). Meltdown: Reading Kernel Memory from User Space Presented at: Proc. of the 27th USENIX Sec. Symp., August 15-17, 2018, pp. 972-990. [Online]. Available: https://www.usenix.org/system/files/conference/usenixsecurity18/sec18-lipp.pdf.

[5] P. Kocher, J. Horn, A. Fogh, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, and Y. Yarom, (May 2019). Spectre Attacks: Exploiting Speculative Execution. Presented at: 2019 IEEE Symp. on Sec. and Priv. (SP), May 19-23, 2019, doi: 10.1109/SP.2019.00002. [Online]. Available: https://ieeexplore.ieee.org/document/8835233.

[6] N. Abu-Ghazaleh, D. Ponomarev, D. Evtyushkin, "How the Spectre and Meltdown Hacks Really Worked." IEEE Spectrum. [Online]. Available:

https://spectrum.ieee.org/computing/hardware/how-the-spectre-and-meltdown-hacks-really-worked.

[7] “Negative Rings in Intel Architecture: The Security Threats You've Probably Never Heard Of.” Medium. [Online]. Available: https://medium.com/swlh/negative-rings-in-intel-architecture-the-security-threats-youve-probably-never-heard-of-d725a4b6f831.

[8] “What is Intel® Management Engine?” Intel. [Online]. Available: https://www.intel.com/content/www/us/en/support/articles/000008927/software/chipset-software.html.

[9] “Intel® Software Guard Extensions (Intel® SGX).” Intel. [Online]. Available: https://www.intel.com/content/www/us/en/architecture-and-technology/software-guard-extensions.html.

[10] “Foreshadow: Breaking the Virtual Memory Abstraction with Transient Out-of-Order Execution.” ForeshadowAttack. [Online]. Available: https://foreshadowattack.eu/.

[11] M. Schwarz, M. Lipp, D. Moghimi, J. V. Bulck, J. Stecklina, T. Prescher, and D. Gruss, (Nov. 2019). ZombieLoad: Cross-Privilege-Boundary Data Sampling. Presented at: Proc. of the 2019 ACM SIGSAC Conf. on Comp. and Comm. Sec., 2019, pp. 753-768, doi: 10.1145/3319535.3354252. [Online]. Available: https://dl.acm.org/doi/10.1145/3319535.3354252.

[12] “ZombieLoad Attack: Return of the Leaking Dead.” ZombieLoad Attack, Graz Univ. of Tech. [Online]. Available: https://zombieloadattack.com/.

[13] S. D. Simone, "How x86 to arm64 Translation Works in Rosetta 2." InfoQ. [Online]. Available: https://www.infoq.com/news/2020/11/rosetta-2-translation/.

[14] "Meltdown and Spectre." Meltdown Attack, Graz Univ. of Tech. [Online]. Available: https://meltdownattack.com/.

[15] "Speculative Execution Exploit Performance Impacts - Describing the performance impacts to security patches for CVE-2017-5754 CVE-2017-5753 and CVE-2017-5715." Red Hat Knowledgebase, Red Hat Inc. [Online]. Available: https://access.redhat.com/articles/3307751.

[16] G. Chen, S. Chen, Y. Xiao, Y. Zhang, Z. Lin, and T. H. Lai, "SgxPectre: Stealing Intel Secrets from SGX Enclaves Via Speculative Execution," *2019 IEEE Euro. Symp. on Security and Privacy (EuroS&P)*, vol. 18, no. 3, 2020, pp. 28–37, doi: 10.1109/MSEC.2019.2963021. [Online]. Available: https://ieeexplore.ieee.org/document/8967194.

[17] C. Canella, D. Genkin, L. Giner, D. Gruss, M. Lipp, M. Minkin, D. Moghimi, F. Piessens, M. Schwarz, B. Sunar, J. V. Bulck, and Y. Yarom, (Nov. 2019). Fallout: Leaking Data on Meltdown-resistant CPUs. Presented at: Proc. of the 2019 ACM SIGSAC Conf. on Comp. and Comm. Sec., 2019, pp. 769-784, doi: 10.1145/3319535.3363219. [Online]. Available: https://dl.acm.org/doi/10.1145/3319535.3363219.

[18] D. Martin, "Researchers Uncover New Vulnerabilities In Intel, AMD Processors." CRN. [Online]. Available: https://www.crn.com/news/components-peripherals/researchers-uncover-new-vulnerabilities-in-intel-amd-processors.

[19] "Trust." Merriam-Webster. [Online]. Available: https://www.merriam-webster.com/dictionary/trust.

[20] “Operating System Market Share Worldwide.” StatCounter. [Online]. Available: https://gs.statcounter.com/os-market-share.

[21] I. Ion, R. Reeder, and S. Consolvo, (July 2015). “...No one Can Hack My Mind”: Comparing Expert and Non-Expert Security Practices. Presented at: Proc. of the 2015 Symp. On Usable Priv. and Sec., 2015, pp. 327-346, doi: 10.5555/3235866.3235893. [Online]. Available: https://www.usenix.org/system/files/conference/soups2015/soups15-paper-ion.pdf.

[22] “An Update on AMD Processor Security.” AMD. [Online]. Available: https://www.amd.com/en/corporate/speculative-execution-previous-updates.

[23] I. Cuttress, “Analyzing Core i9-9900K Performance with Spectre and Meltdown Hardware Mitigations.” AnandTech. [Online]. Available: https://www.anandtech.com/show/13659/analyzing-core-i9-9900k-performance-with-spectre-and-meltdown-hardware-mitigations.

[24] “Trusted Platform Module.” Microsoft. [Online]. Available: https://docs.microsoft.com/en-us/windows/security/information-protection/tpm/trusted-platform-module-top-node.

[25] “Virtual Machine Extension (VMX) Operation.” Intel. [Online]. Available: https://software.intel.com/content/www/us/en/develop/documentation/debug-extensions-windbg-hyper-v-user-guide/top/virtual-machine-extension-vmx-operation.html.

[26] “Intel® Trusted Execution Technology (Intel® TXT) Overview.” Intel. [Online]. Available: https://www.intel.com/content/www/us/en/support/articles/000025873/technologies.html.

[27] “Secure Enclave overview.” Apple. [Online]. Available:

https://support.apple.com/guide/security/secure-enclave-overview-sec59b0b31ff/web.