Software based side-channel attacks on CPUs*

Their history and how we behaved

Harald Heckmann
RheinMain University - University of Applied Sciences
Wiesbaden, Hessen
Harald.Heckmann@hs-rm.de

ABSTRACT

This paper focuses on presenting the chronological order of software based side-channel attacks on processors. This is achieved by introducing key attacks since 2003 in the chronological order, explaining how these attacks work and how they were fixed or worked around. Additionally, a small part of this paper focuses on the behaviour of computer scientists and computer users after the publication of such attacks. Why did we wait until the catastrophe happened?

CCS CONCEPTS

• **Security and privacy → Side-channel analysis and countermeasures**; *Hardware reverse engineering*; Operating systems security;

KEYWORDS

software based side-channels, branch prediction unit, CPU optimization, cache, speculative execution, out-of-order execution, branch target buffer, pages

ACM Reference Format:
Harald Heckmann. 2018. Software based side-channel attacks on CPUs: Their history and how we behaved. In *Proceedings of Wamos conference (WAMOS 2018).* WAMOS, Wiesbaden, Hessen, Germany , 6 pages.

1 INTRODUCTION

In the early 2000s computer scientists began to develop software methods to leak sensitive information using side channel attacks targeting the underlying hardware. This allows to leak data which would otherwise be hidden due to missing privileges. This proofed that the architecture of the hardware is vulnerable against software based side channel attacks. Although these attacks succeeded and were documented, developers did not begin to discuss this problems and find solutions for them from early on. Additionally the masses were not sensibilized for the potential dangers they were exposed to. Furthermore the masses were poorly informed about the steps they could take to make such attacks on their systems unlikely. In the following years many more such attacks were developed and it took more than a decade to get where we are now - a catastrophic error in the design of modern processors.
In this paper the historical development of software based side channel attacks on the underlying hardware, as well as the proximate reaction to their detection is documented. In the first section an overview of such attacks is presented and exemplified in the correct chronological order. After that the reader is informed about the abrasive process of those attacks and their scope. The survey of the history of software based side-channel attacks on CPUs is complete after showing known fixes or workarounds for those attacks in the third section. In the fourth section the focus lies on presenting how those attacks where published and what impact they had. Finally, this paper will be completed with a conclusion containing a ranking representing how dangerous the attacks were and further, how we as computer scientists could change our future behaviour to effectively address these issues earlier.

2 OVERVIEW OF ATTACKS

The amount of newly arriving side channel attacks as well as the general approach they use evolved increasingly faster during the last 15 years. The first successful side channel attack between two general purpose computers seems to be executed in the year 2003 and is documented in the paper ”remote attacks are practical” [2]. Using timings in a local network, one is able to receive secrets from an openssh server the attacker is connected to. Timings have proved themselves as a good source for side channel attacks on a computer or between computers.
Two years later, in the year 2005, a paper containing an attack called ”Prime+Probe” [10] was released. This attack can be used to compare cache states before and after the execution of code to retrieve information of the memory segments used. Finally, memory access pattern of the victim process can be deduced, potentially leading to implicit leakage of secret information. It is one of the fundamental attacks to retrieve information from a cache using a covert channel. A covert channel allows to transfer information between processes which are not allowed to communicate.
Three years later, in the year 2006, the paper ”predicting secret keys via branch prediction” [1] was published. In this paper the authors present a new side channel attack which can be used to leak currently processed sensitive information, like a private key, of another process running on the same processor core. In contrast to the first paper, this attack uses knowledge about the underlying hardware architecture, specifically the Branch Prediction Unit (BPU). The attack is able to passively gain information through the BPU or even to force the BPU into a specific state to subsequently leak the sensitive information using timing side channels. One can

* Short Research Paper

WAMOS 2018, August 2018, Wiesbaden, Hessen, Germany

still fix this vulnerability at the level of the algorithm which handles the secrets.
Five years later, in the year 2011, the paper "Cache games — bringing access-based cache attacks on AES to practice" [6] was released. This attack makes use of shared pages, cache flush mechanisms and timing covert channels of the cache. By using this attack, the attacker is able to spy another process to gather the information about which execution paths were executed and in what order they were executed. This is achieved by detecting whether a specific memory location was cached (e.g. code). By knowing the execution order, the attacker might be able to extract secret contents which were processed in a specific time frame during the runtime. It can only be applied on processes running on the same core and the same virtual machine. Since it uses x86 specific instructions, it cannot be applied on any other processor not supporting those instructions.
Two years later, in the year 2013, the paper "FLUSH+RELOAD: a High Resolution, Low Noise, L3 Cache Side-Channel" [11] was released. Flush+Reload can be seen as an extension of the attack described in the paper "cache game - [...]" [6]. It uses the same mechanism but differs in the following ways:

(1) It uses a timing covert channel targeting the Last Level Cache (LLC). This enables cross core attacks.
(2) It uses the instruction "mfence" instead of "cpuid", which implicitly enables attacks on virtual machine running on the same host.
(3) It is noise resistant and has a relatively high transmission rate.

This attack delivers one core mechanism which is used by the most dangerous and powerful attacks to this date, as the next attack introduced does likewise.
The paper "Last-Level Cache Side-Channel Attacks are Practical" [9] was released in 2015 and describes an attack called "Evict+Reload", which can be used to spy any execution path used as well as data usage of another known process. The idea is similar to Flush+Reload, but it does not use dedicated instructions to flush the cache. Instead of that it evicts the relevant cache lines by accessing memory which content will be stored in the same cache line. This way it can be applied on a broader range of modern processors. Further, it does not rely on memory sharing.
In the year 2018, an avalanche of dangerous attacks appeared. A paper describing one of those "most dangerous and powerful" attacks was released in January 2018 and is called "Meltdown" [8]. It is a two staged attack which enables to read data from any memory location, even on other virtual machines running on the same host, by bypassing the privileged mode isolation of the CPU. In the first stage, out-of-order execution of instructions is abused to cache data which implicitly contains one byte of the leaked data. Before the data is fetched, relevant cache lines will be flushed. It is constructed in a way that no data prefetching in a single page occurs. Further it is constructed to use a new cache line for each possible 256 values of one secret byte of the secret data to be leaked. The out-of-order execution violates privilege checks on Intel CPUs. In the second stage, a timing attack (as described in Flush+Reload) is used to find out which of the 256 cache lines was used. The offset (0 to 255) of the cache line equals the secret byte. This way the whole physical memory can be dumped on most systems, since it is completely mapped in virtual kernel memory space. This attacks can only be successfully executed on Intel processors.
At the same time, another paper called "Spectre Attacks: Exploiting Speculative Execution" [7] was published. It presents the basis for the most powerful and dangerous attacks to date. The attacker requires knowledge about the process being executed, from now on called the victim. Instead of using out-of-order execution, which seems to be missing the correct privilege check only on Intel processors, the attacker wants to get the processor to speculatively execute a specific execution path. This attack works with direct (variant 1) and indirect (variant 2) branches. In this paper, the focus lies on variant 1. Nevertheless, both variants share the same general idea. The attacker tries to spot branches in the victims code where memory access is dependent on a variable "x", which is received from an untrusted source. Afterwards the BPU has to be trained to speculatively execute the branch containing the memory access. More precise, a modification of the Branch Target Buffer (BTB) inside the BPU has to be realised. After achieving this, the attacker can use Flush+Reload or Evict+Reload (if the target processor is not a x86 processor) to remove all data from the relevant cache line. Afterwards the attacker selects a desired value for "x" and then lets the victim execute the code, which speculatively reads the victims memory using a malicious "x". The data can be retrieved using a cache side channel as described in Flush+Reload or Evict+Reload. This attack can be applied on all processors using caches and speculative execution, including newer processors from Intel, AMD and Arm.
The release of Meltdown and Spectre lead to a publication of many attacks, including highly dangerous ones which cannot be fixed but just worked around. Branchscope [4] was released in March 2018, showing that it is also possible to abuse the shared directional Branch Predictor instead of the BTB. Earlier in the same month an attack called SgxPectre [3] was presented, showing that Spectre can be used to read data in Intels "secure" enclave. In May 2018 Spectre-NG (Next Generation) was announced, containing eight attacks which represent extensions of Spectre - some even more dangerous than Spectre. Spectre-NG contains attacks like "Rogue System Register Read", "Speculative Store Bypass" or "Lazy FP Restore". In June 2018 it was proofed, that not only the BPU contains security flaws, but the Translation Lookaside Buffer (TLB) does likewise. This was demonstrated with an attack called TLBleed. To the date of the release of this paper, neither Spectre-NG nor TLBleed have been described in papers.

3 ATTACKS EXPLAINED

In this chapter the most important attacks, regarding their scope, usability and difficulties to fix them, will be presented in detail. Beginning with two attacks, Flush+Reload and Evict+Reload, which are used to figure out where relevant cachelines containing secret data can be located to afterwards be cleared and probed for the targeted data. This section is completed with the description of the two major attacks which initiated the avalanche of dangerous attacks, Meltdown and Spectre.

3.1 Flush+Reload - 2013

Idea: Using shared pages, the attacker can obtain memory addresses used by a targeted process. This addresses can contain code or data. The attacker clears the cache and lets the victim execute a specific portion of the code. He then reads the relevant cache lines - since he knows the addresses this is a simple undertaking. By measuring the time, the attacker knows which memory was accessed. Consequently, he might have implicitly leaked secret information, for example how a secret was calculated and therefore which bits it contains.
Crucial knowledge: Shared pages, cache hierarchy, address translation, timing side-channel attacks
Steps:

(1) Measure how long memory access takes if the data is cached in the LLC in case of a hit. Use this value leveraged by a small percentage as a threshold.
(2) Choose a victim process.
(3) Find out which memory addresses are of interest. The example attack uses page sharing for this - achieved by calling mmap(...) to map the targeted code into own memory.
(4) Find out when the victim executes the code of interest. Loops are practical for this case. The example attack uses debugging symbols.
(5) Before the victim executes the relevant code, clear the cache by using the x86 instruction "clflush".
(6) Let the victim execute the relevant code.
(7) Access the cache line (by reading the memory addresses of step 3) and measure the time. Ensure that those commands are not executed speculatively or out-of-order by using "mfence" and "lfence" instructions.
(8) Compare the access times to the threshold to gain the information which memory regions where accessed
(9) (Not part of Flush+Reload) Reconstruct the secret.

Weakness:

(1) Uses x86 instruction and is therefore limited to x86 processors.
(2) Oblivious to address diversification like ASLR.

3.2 Evict+Reload - 2015

Idea: This attack aims to evict cache lines before the victim process fills them during his execution with data. The access to the cache lines will be probed. Measuring the time the attacker implicitly gains information regarding the data access pattern of the victim process. In contrast to Flush+Reload, this attack creates "eviction sets", which contain enough relevant addresses to clear exactly one cache set in the LLC. This attack is separated in a training phase, where eviction sets are created and an evaluation phase, where cache set clearing and probing access time thresholds are defined. A lot of problems have to be solved. Two major problems are that the LLC uses physical addresses instead of virtual addresses and that the last level cache is sliced on modern processor, where the slicing is determined by an unknown hashing function.
Crucial knowledge: Pages in detail, cache in detail, timing side-channel attacks
Steps: E - Evaluation, T - Training

(1) E: Access an address not accessed before, measure the time (cache miss). Access the same memory location from a different core. Measure the time for accessing the initial data again (cache hit on LLC).
(2) T: Use page sizes large enough so that the set index bits of the address are contained within the page offset of the address. Doing so ensures that all index bits are contained in one page - eliminating the need to figure out the virtual to physical address mapping and therefore effectively enabling virtual indexing of the LLC.
(3) T: Allocate a buffer at least twice the size of the LLC, which is backed by large pages. The buffer has to be at least twice the size to circumvent the need to know the secret hashing function inside the LLC which maps a slice to an address.
(4) T: Create one empty conflict set and as many evictions sets as there are sets in the LLC.
    - Conflict set - will contain enough addresses to completely evict the LLC.
    - Eviction set - will contain enough addresses to completely evict a specific set in each slice of the LLC.
(5) T: Test for every address A in buffer: load A, load every address in the conflict set, probe A. If A was evicted (cache miss access time) continue. Otherwise add A to the conflict set.
(6) T: Test for every remaining addresses in buffer: If they get evicted using the conflict set, try to find out which elements in the conflict set are relevant for the eviction by removing one and retrying the eviction. Every element which removal lead to a failed eviction is required to clear the cache set at the given index in every slice of the LLC. Add this element to the eviction set for the given cache set.
(7) E: How long does it take to evict a cache set in all slices of the LLC, i.e. to access every address in one eviction set?
(8) An attack can now be executed. One is able to clear any specific cache set inside the LLC, including data from another process. The attacker can then probe those cache lines (using the eviction sets in reverse order) to find out if the data was accessed.

Weakness: Large page size required, cache inclusiveness property required

3.3 Meltdown - 2018

Idea: Meltdown has the ability to read the whole physical memory on systems using Intel CPUs, if the physical memory is completely mapped into kernel space (that was the default case before the release of this attack). This is achieved by reading memory during out-of-order execution which is dependent on a secret byte value x at address y (like $probe\_array[*y * pagesize]$, whereas $*y$ represents the data located at address y, namely x). Usually the CPU should intervene and abort the out-of-order execution if the memory access requires elevated privileges, but since the CPU does not check this during out-of-order execution, any memory location can be read. The microarchitectural effects get reverted, but the microarchitectural side effects, in this case cache entries, still consist. By checking which of the 256 possible cache lines was accessed, the attacker has uncovered the secret byte.

Crucial knowledge: Out-of-order execution, pages, cache hierarchy, Flush+Reload
Steps: P - Parent process, C - Child process

(1) Create a byte array "*probe_array*" with size $256 \cdot pagesize$ using shared memory.
(2) Fork the process.
(3) P: Execute the "Flush" part from Flush+Reload.
(4) P: Define the secret address to read as y.
(5) P: read byte value from y called x.
(6) P: (this is executed before the segmentation fault happens) read $probe\_array[x * pagesize]$.
(7) P: Handle or suppress the exception.
(8) C: Execute the "Reload" part from Flush+Reload for every 256 possible values for x.
(9) C: Reveal which iteration of the 256 iterations contained a cache hit, the number equals the secret byte value.
(10) By repeating all steps for the whole virtual address room, the whole physical memory can be dumped.

Weakness:

(1) Relies on a permission bug during out-of-order execution in Intel CPUs.
(2) Relies on address translation by kernel driver.

3.4 Spectre - 2018

Idea: Speculative execution appears when branches are executed, but the condition is not evaluated yet. The BPU collects previously executed program paths for a branch inside the BTB. The more often one path is taken, the higher the probability that this path is contained in the BTB as the path to speculatively execute in question. If there is a memory access inside one path of the branch which contains a variable from an untrusted source, there is room to abuse the speculative execution. By training the BTB to execute one specific path, the attacker can force the processor to speculatively execute the path containing the memory access. The memory access can look like this: $array\_to\_probe[array[x * 256]]$, with *array_to_probe* being an unsigned byte array, *array* being a byte array and x being the untrusted variable. By injecting a malicious variable x, which is used for the memory access, the attacker can read the victims memory. This is achieved by clearing the cache beforehand in the relevant sets (address range of *array_to_probe*), speculatively executing the malicious memory access ($array[x * 256]$) and afterwards probing all relevant cachelines (address range of *array_to_probe*) again. The relative offset of the cacheline starting from the cacheline assigned to the base address of *array_to_probe* is equal to the secret byte.
Crucial knowledge: BPU (BTB), speculative execution, Evict+Reload, Prime+Probe, optionally Flush+Reload
Steps: E - Evaluation, T - Training, A - Attack

(1) E: Search the victim process for eligible branch and a suitable memory access.
(2) E: Figure out the base address of the array to be probed. If not directly available, use Prime+Probe.
(3) E: If possible, reduce the scope of possible malicious values to values leading to a memory access to the desired data.
(4) T: Train the BTB. Do this by executing the target branch multiple times using values for x which lead to the desired execution path.
(5) A: Use Flush+Reload on x86 CPUs, use Evict+Reload otherwise. Execute the "Flush" part or the "Evict" part to clear the relevant cache lines from *array_to_probe*.
(6) A: Let the victim execute the branch with your malicious value x.
(7) A: Execute the "Reload" part to find the secret byte value.

Weakness: memory loads during speculative execution, those weaknesses inherited from Flush+Reload or Evict+Reload and Prime+Probe

4 MITIGATIONS

Since every attack presented here is a real danger to the security of a vast amount of processing units, immediate mitigations have to be rolled out as fast as possible. This section focuses on the attacks presented in the previous chapters. The attack Flush+Reload relies on shared pages to successfully address cache lines in the LLC (whose position is determined by the physical address). Disabling shared pages can be a solution to mitigate this attack, but since it would result in a significant amount of additionally required memory, it is not a very practical solution. Another long term fix is to restrict the clflush instruction further or to add a permission check in the next generation processors. Since this would not mitigate currently vulnerable processors, this is not an ideal approach for short-term mitigations.
Meltdown, which uses Flush+Reload, abuses the circumstance that the whole physical memory is usually mapped in kernel space. The "Kernel Address Isolation to have Side-channels Efficiently Removed" (KAISER) [5] kernel patch mitigates Meltdown, because it only maps the necessary memory into kernel space - like interrupt or required device drivers. The fact that Meltdown relies on a missing permission check in Intel processors - and therefore is limited to those - also implies this attack cannot be executed on future Intel processors, since they will carefully check the permissions.
Evict+Reload is a tricky attack. By analysing the LLC cache layout, the attack is aware of the information which addresses can be used to evict cachelines - at cache set granularity. Since it does not use any operating system dependent features or processor instructions, mitigating this procedure is going to be a difficult task. Evict+Reload uses big pages, pages big enough that the addresses fully cover the cache set bits, so that the attacker can address any cacheline in the LLC. If the pagesize will be reduced to a certain limit so that the available addresses do not fully cover the cache set bits, not every cacheline can be targeted in the LLC anymore. This can weaken or even impede the attack.
Spectre does not rely on a missing permission check. In fact, Spectre does not even violate memory access. The attack does just force a victim process to speculatively execute a memory access containing an address in the victims address space. Therefore, no processor fix is obvious. Even mitigations in form of kernel patches are not obvious. The most basic mitigation ideas simply suggest to turn of features like speculative execution or caching. This is no solution of course, since execution times rise significantly. It is possible to instruct the processor to not use speculative execution for specific parts of the code. Developers of security critical applications

could modify their code in a way that the security critical execution paths do not execute speculatively. Another method is to remove the branch and directly calculate the index in the array, using arithmetic expressions to define the valid range. This solutions require software developers to gain knowledge about this workaround and to recompile their software. This is cumbersome and after all, old vulnerable software will still be used a lot. Kernel developers have created a function, called *array_index_masc_nospec* [1], which disables speculative execution for values which are out of bounds and therefore is an effective countermeasure against Spectre variant 1. This is achieved by using a mask on the addresses accessed, which is 0 in case of an address which is out of bounds or the negation of 0 (every bit is 1) otherwise. One mitigation which is used against Spectre variant 2 is called "Retpoline" [2]. It avoids (memory) load instructions during speculative execution of indirect branches, therefore avoiding that the attacker can read the victims memory.

5 REACTION AND AFTERMATH OF THE PUBLICATION OF ATTACKS

The following questions are covered in this chapter: "What were the warning signals given be the researchers?", "How did manufacturer of CPUs react to the publication of such attacks?" and finally, "Have special communication canals next to papers been used to sensitize journalist or computer users to the problems?". Beginning with the first question, "What were the warning signals given by the researchers?", this part summarizes key sentences in the papers earlier presented in this paper. The paper "Remote Timing Attacks are Practical" (2003) states, that cache-side channel attacks are a real threat: "Our experiments show that, counter to current belief, the timing attack is effective when carried out between machines separated by multiple routers. Similarly, the timing attack is effective between two processes on the same machine and two Virtual Machines on the same computer.". The paper presenting the Prime+Probe (2005) attack, which is an essential key side-channel attack against caches, states that the scope of cache side-channel attacks is tremendous: "At the system level, cache state analysis is of concern in essentially any case where process separation is employed in the presence of malicious code. Beyond the demonstrated case of encrypted filesystems, this includes many multi-user systems, as well as web browsing and DRM applications. […] the leakage also occurs in non-cryptographic systems and may thus leak sensitive information directly.". "Predicting secret keys via Branch Prediction" (2006) is the paper containing the most critical warning in regards to current attacks, since it states that branch prediction is a new security risk and secure software mitigations should be taken into account: "[…] this paper has identified the branch prediction capability of modern microprocessors as a new security risk […] The practical results from our experiments should be encouraging to think about efficient and secure software mitigations for this kind of new side-channel attacks.". Flush+Reload (2013) is part of current most powerful attacks. The paper describing the attack stated that this will likely happen: "The technique is generic and can be used to monitor other software. It can be used to devise other types of attacks on cryptographic software.". Evict+Reload (2015) proved that cross-core and cross-VM attacks are practical on a wide variety of systems, and it states that it is a real threat, implicitly suggesting to immediately mitigate it to at least make cross-VM attacks more difficult: "[…] we believe that our attack is eminently practical, and as such presents a real threat against keys used by cloud-based services.". Meltdown (2018) finally states that we all have taken our part in the current disaster by accepting the security to performance trade off, often by not even investigating how those performance increases were achieved: "The fact that hardware optimizations can change the state of microarchitectural elements, and thereby imperil secure software implementations, is known since more than 20 years [20]. Both industry and the scientific community so far accepted this as a necessary evil for efficient computing. Today it is considered a bug when a cryptographic algorithm is not protected against the microarchitectural leakage introduced by the hardware optimizations. Finally, Spectre (2018) criticises that hardware developers left a lot undocumented: "As the attack involves currently undocumented hardware effects […] there is currently no way to know whether a particular code construction is, or is not, safe across today's processors – much less future designs. A great deal of work lies ahead". This will never change as long as hardware developers keep secrets to keep an advantage against other hardware manufacturers, which is a strong argument for the development of open-source hardware. After some research, the answer to the question "How did manufacturer of CPUs react to the publication of such attacks?" is the following: They took this problem serious and began to develop microcode updates for their products. They also worked with OS developers to find mitigations. So it is in their interest to close those leaks. Nevertheless they don't seem to care about the damages their customers have taken in the time of vulnerability, since no recall or other compensation was offered. [3] [4] In my research I was not able to find any major changes in the philosophy of CPU manufactures after the release of each attack presented in this paper. Quite the contrary, they have continued to develop features for the BPU (for example speculative execution) even after the authors of the paper "Predicting secret keys via Branch Prediction" (2006) have strongly signalled that the BPU contained critical weaknesses twelve years ago from now. The last question to be answered is "Have special communication canals next to papers been used to sensitize journalist or computer users to the problems?". The answer is yes and no. Most attacks have been presented in conferences, often even including an oral presentation. In addition to that organisations like IEEE or ACM provide journals containing the newest results in research, including the attacks described in this paper. Finally, news services (like heise.de in germany) often report about those discoveries. After all, the news seem to only reach computer scientists through all those portals described. Those are only a small percentage of the world population which implies that most people will not be informed about those security flaws in the early stages. It is important that computer scientists who understand what is going on sensibilize computer users about this - in private, through social media and so on. Only

[1] https://lore.kernel.org/lkml/151727414808.33451.187323713067278533l.stgit@dwillia2-desk3.amr.corp.intel.com/T/#u

[2] https://support.google.com/faqs/answer/7625886

[3] https://www.forbes.com/sites/thomasbrewster/2018/01/04/intel-arm-amd-no-recalls-for-meltdown-spectre-vulnerabilities/#7a5fa97f7d3a

[4] https://www.cnet.com/news/meltdown-spectre-intel-ceo-no-recall-chip-processor/

this way the whole group of computer users can form an opinion like ”don’t develop new features until current security debates have been finished” and support this for example via a petition.

6 CONCLUSION

In the years 2003-2006 many cryptoanalysts begun to set the path to current attacks by proving that side-channel attacks apply in environments containing a lot of noise. Different standard procedures like Prime+Probe have been published. Further, it was shown that the underlying hardware of common servers or computers is vulnerable and can be controlled to execute in the favour of the attacker. Those three years were the years where attacks were developed which already showed, in retrospective, that processors are vulnerable. These attacks already show everything required, side channel attacks, analysis of underlying hardware, indirect control of hardware and abuse of specific code patterns. To many of those who are experienced in this field and were up to date, who read the attacks and understood them, it should be already clear that this is not the end, but the beginning of critical attacks and detection of hardware vulnerabilities. In the following years, attacks like Flush+Reload and Evict+Reload were perfected to give the attackers the ability to measure side effects on the hardware incredibly accurate. At this point it was clear, if we can abuse the hardware somehow to leak information into the cache, we can also read it. When finally features like out-of-order execution or speculative execution were abused to leak information into the cache, the catastrophe was inevitable. Now everybody listens, but why had the catastrophe to happen first? We as computer scientists who see this in the early stages have to emphasize the dangers multiple time in the early stages. Publish more, tell your friends their data is insecure and let them spread the word. Talk in public about this, mention it in simple conversions. The next problem is that the giant processor designers and producers seem to don’t care. First, how do we know they didn’t know that these issues exist and were probably paid for it? We can’t. Second, where is the recall campain or another compensation? The big players implement features to gain an advance compared to the other big players at the cost of the users. Now that the users have to pay, the big players don’t care. They try to fix it in future products, but the damage that the users have taken in the time the vulnerabilities were not detected is not taken in account.

I conclude, we have to produce open-source hardware to:

(1) Have more experts to cross-check the hardware design and discuss it before it is released.
(2) Have more experts to find vulnerabilities faster after a release.

Further, I suggest every ISA to contain keywords to disable any feature in the processor like out-of-order execution, speculative execution, caching, etc. Compilers could offer a pragma like ”critical_section” to disable all features which are known to be vulnerable and ”end_critical_section” to enable them again. The compiler keeps a list containing a mapping between processors architectures and vulnerable features. It can then replace ”critical_section” with the instructions to disable the vulnerable features and replace ”end_critical_section” with the instructions to enable them again. This requires the application to be recompiled if new bugs appear, but if developers of security critical applications use this keywords, fast mitigation could be easier. To avoid the need to compile the programs again, the kernel could offer functions for the critical sections.

Using those two suggestions, the probability that new architectures contain security vulnerabilities is reduced. Additionally, if there are vulnerabilities in produced chips they can be detected faster since many experts can work on detecting them and in this case can be worked around fast by recompiling or updating the kernel.

REFERENCES

[1] Onur Acıiçmez, Çetin Kaya Koç, and Jean-Pierre Seifert. 2006. Predicting Secret Keys via Branch Prediction. In *Proceedings of the 7th Cryptographers’ Track at the RSA Conference on Topics in Cryptology (CT-RSA’07)*. Springer-Verlag, Berlin, Heidelberg, 225–242. https://doi.org/10.1007/11967668_15

[2] David Brumley and Dan Boneh. 2003. Remote Timing Attacks Are Practical. In *Proceedings of the 12th Conference on USENIX Security Symposium - Volume 12 (SSYM’03)*. USENIX Association, Berkeley, CA, USA, 1–1. http://dl.acm.org/citation.cfm?id=1251353.1251354

[3] Guoxing Chen, Sanchuan Chen, Yuan Xiao, Yinqian Zhang, Zhiqiang Lin, and Ten H. Lai. 2018. SgxPectre Attacks: Leaking Enclave Secrets via Speculative Execution. *CoRR* abs/1802.09085 (2018). arXiv:1802.09085 http://arxiv.org/abs/1802.09085

[4] Dmitry Evtyushkin, Ryan Riley, Nael CSE Abu-Ghazaleh, ECE, and Dmitry Ponomarev. 2018. BranchScope: A New Side-Channel Attack on Directional Branch Predictor. In *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS ’18)*. ACM, New York, NY, USA, 693–707. https://doi.org/10.1145/3173162.3173204

[5] Daniel Gruss, Moritz Lipp, Michael Schwarz, Richard Fellner, Clémentine Maurice, and Stefan Mangard. 2017. KASLR is Dead: Long Live KASLR. In *Engineering Secure Software and Systems*, Eric Bodden, Mathias Payer, and Elias Athanasopoulos (Eds.). Springer International Publishing, Cham, 161–176.

[6] David Gullasch, Endre Bangerter, and Stephan Krenn. 2011. Cache Games – Bringing Access-Based Cache Attacks on AES to Practice. In *Proceedings of the 2011 IEEE Symposium on Security and Privacy (SP ’11)*. IEEE Computer Society, Washington, DC, USA, 490–505. https://doi.org/10.1109/SP.2011.22

[7] Paul Kocher, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, and Yuval Yarom. 2018. Spectre Attacks: Exploiting Speculative Execution. *CoRR* abs/1801.01203 (2018). arXiv:1801.01203 http://arxiv.org/abs/1801.01203

[8] Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, and Mike Hamburg. 2018. Meltdown. *CoRR* abs/1801.01207 (2018). arXiv:1801.01207 http://arxiv.org/abs/1801.01207

[9] Fangfei Liu, Yuval Yarom, Qian Ge, Gernot Heiser, and Ruby B. Lee. 2015. Last-Level Cache Side-Channel Attacks Are Practical. In *Proceedings of the 2015 IEEE Symposium on Security and Privacy (SP ’15)*. IEEE Computer Society, Washington, DC, USA, 605–622. https://doi.org/10.1109/SP.2015.43

[10] Dag Arne Osvik, Adi Shamir, and Eran Tromer. 2006. Cache Attacks and Countermeasures: The Case of AES. In *Proceedings of the 2006 The Cryptographers’ Track at the RSA Conference on Topics in Cryptology (CT-RSA’06)*. Springer-Verlag, Berlin, Heidelberg, 1–20. https://doi.org/10.1007/11605805_1

[11] Yuval Yarom and Katrina Falkner. 2014. FLUSH+RELOAD: A High Resolution, Low Noise, L3 Cache Side-channel Attack. In *Proceedings of the 23rd USENIX Conference on Security Symposium (SEC’14)*. USENIX Association, Berkeley, CA, USA, 719–732. http://dl.acm.org/citation.cfm?id=2671225.2671271