Evolution of Defenses against Transient-Execution Attacks

Claudio Canella
Graz University of Technology, Austria
claudio.canella@iaik.tugraz.at

Sai Manoj Pudukotai Dinakarrao
George Mason University, USA
spudukot@gmu.edu

Daniel Gruss
Graz University of Technology, Austria
daniel.gruss@iaik.tugraz.at

Khaled N. Khasawneh
George Mason University, USA
kkhasawn@gmu.edu

ABSTRACT

Transient-execution attacks, such as Meltdown and Spectre, exploit performance optimizations in modern CPUs to enable unauthorized access to data across protection boundaries. Against these attacks, we have noticed a rapid growth of deployed and proposed countermeasures. In this paper, we show the evolution of countermeasures against transient-execution attacks by both industry and academia since the initial discoveries of the attacks. We show that despite the advances in the understanding and systematic view of the field, the proposed and deployed defenses are limited.

KEYWORDS

Transient-execution attacks, Meltdown, Spectre, LVI

ACM Reference Format:
Claudio Canella, Sai Manoj Pudukotai Dinakarrao, Daniel Gruss, and Khaled N. Khasawneh. 2020. Evolution of Defenses against Transient-Execution Attacks. In *Proceedings of the Great Lakes Symposium on VLSI 2020 (GLSVLSI '20), September 7–9, 2020, Virtual Event, China.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3386263.3407584

1 INTRODUCTION

Transient execution enables unauthorized access to data across security protection boundaries. Transient execution refers to the execution of instructions that will eventually get squashed, *i.e.*, their execution results will not be committed to the architectural state. Nonetheless, transient execution can leave a trace in the microarchitectural state, e.g., the cache state. Therefore, transient-execution attacks utilize the execution of transient instructions to access secret data, e.g., a password, and leave a secret-dependent trace in the microarchitectural state that can be recovered later using non-transient execution. These attacks can be classified into three main classes, namely Meltdown, Spectre, and Load Value Injection (LVI), based on the nature of the transient execution and the attack direction. Spectre is based on misprediction in the victim domain, Meltdown is based on faults and assists in the attacker domain, and LVI is based on faults and assists in the victim domain.

GLSVLSI '20, September 7–9, 2020, Virtual Event, China

ACM ISBN 978-1-4503-7944-1/20/09…$15.00
https://doi.org/10.1145/3386263.3407584

The initial discovery of transient-execution attacks, *i.e.*, Meltdown and Spectre, became one of the most complex and largest industry-wide embargos as processors from various manufacturers turned out to be affected. As a result, many attacks variants were discovered, but more noticeable is the proliferation of countermeasures from both industry and academia. Given the large number and the rapid growth of both adopted and proposed countermeasures, a systematic view is required to understand the scope of current defenses and facilitate the evaluation of future defenses.

In this paper, we show how the landscape of countermeasures against transient-execution attacks evolved since the initial discoveries of the attacks. We build our systematization based on a concurrent 6-phase generalization of transient-execution attacks [16]. We systematically describe hardware- and software-based countermeasure advances from both industry and academia. Beyond previous work [18, 99], our systematic view does not only cover Spectre and Meltdown defenses but also LVI defenses. We show that despite the advances in the understanding and systematic view of the field, the proposed and deployed defenses are limited.

Outline. First, we briefly discuss background in Section 2. The paper then gives a systematic overview of countermeasures for Spectre (Section 3), Meltdown (Section 4), and LVI (Section 5). We conclude in Section 6.

2 BACKGROUND

Out-of-order and speculative execution. To increase performance, modern CPUs rely on features like speculative and out-of-order execution. With speculative execution, CPUs try to predict the outcome of a potential control-flow change to start the execution of the likely path instead of stalling. For that, the CPU provides various predictors that together comprise the Branch Prediction Unit (BPU) [18]. Out-of-order execution allows executing later instructions that are ready to be executed due to the operands being available in advance but still requires to retire them in order. Recently, these optimizations have resulted in various transient-execution attacks [18, 55, 60].

Transient-execution attacks. Transient-execution attacks exploit modern CPUs performance optimizations to enable unauthorized access to data across protection boundaries. According to a concurrent generalization of transient-execution attacks, these attacks consist of 6 distinct phases [16]: *Phase 1 (preparation):* preparing the micro-architecture to enter transient execution, *Phase 2 (misspeculation):* triggering transient execution using a trigger instruction, *Phase 3 (access):* accessing data of interest, *Phase 4 (encoding):* encoding data of interest in the microarchitecture state, *Phase 5 (leakage):* end of transient window, i.e., the architectural changes are reverted

and the pipeline is flushed, and *Phase 6 (decoding):* decoding the microarchitectural state to the architectural state.

3 SPECTRE COUNTERMEASURES

A countermeasure can try to break any phase of a Spectre attack [16]: preparation, misspeculation, access, encoding, leakage, decoding. However, targeting different phases has different effects on security. As the following discussion also shows, mitigating all Spectre attacks in practice likely will remain an open problem in the foreseeable future [62].

3.1 Preparation Prevention (Phase 1)

Phase 1 prepares the microarchitecture, e.g., the cache or branch predictors, for the attack. Defenses targeting this phase usually do not prevent this step entirely but only eliminate the attacker's influence on the victim domain. However, some variants do not require any preparation or run in-place, making it hard to distinguish malicious training from benign execution.

3.1.1 Industry. To prevent mistraining, the industry, e.g., Intel and AMD, extended ISAs with a mechanism for controlling indirect branches [4, 44]. Indirect Branch Restricted Speculation (IBRS) prevents unprivileged code from influencing the prediction of privileged code. Single Thread Indirect Branch Prediction (STIBP) restricts the sharing of branch prediction mechanisms across hyperthreads. The Indirect Branch Predictor Barrier (IBPB) prevents code that executes before it from affecting the prediction of code following it. Some ARM CPUs implement specific controls that invalidate the branch predictor, which should be used during context switches [8]. Linux enabled those by default [52].

For Spectre-STL, ARM introduced new barrier instructions and control registers to prevent the re-ordering of loads and stores [8]. Likewise, Intel [44] and AMD [3] provide Speculative Store Bypass Disable (SSBD) microcode updates.

3.1.2 Academia. In contrast to industry, academia proposed more fundamental architecture and microarchitecture changes. Vougioukas et al. [93] use per-context buffers for branch predictor state to improve performance after branch predictor flushes. Instead of flushing, Zhao et al. [102] randomize the prediction based on the running context. Both proposals maintain performance within a process across context switches. However, in-place same-domain attacks are unaffected by these designs, and the approach by Zhao et al. [102] may allow cross-domain and out-of-place attacks by reverse-engineering the randomization.

3.2 Misspeculation Prevention (Phase 2)

Entirely disabling speculation seems easy, but the performance loss is prohibitive [55, 87]. Hence, more realistic solutions in this phase only selectively disable or stop speculative execution.

3.2.1 Industry. CPU vendors designed solutions using serializing instructions (*lfence*), stopping speculation at security-critical branches. Unfortunately, these branches have to be identified and essentially annotated on all layers.

Software-based defenses. Google proposed *retpoline* [89], a code sequence replacing indirect branches with return instructions, to prevent branch poisoning. Intel proposed *randpoline* [14] as a more efficient alternative. Due to its probabilistic nature, randpoline does not fully mitigate Spectre-BTB but only reduces success and leakage rates of attacks. Linux and Windows use retpoline on affected machines by default [24, 43].

Hardware-based defenses. Both Intel and AMD described fencing-based solutions [4, 47]. However, they also both introduced new architectural features to constrain speculative execution on the microarchitectural level including instructions for synchronization barriers for data (`DSB`) and instructions (`ISB`), broader speculation barriers (`CSDB`) [8], new registers to restrict speculative execution and instructions to restrain control-flow (`cfp`) and data value (`dvp`) prediction, and cache prefetches (`cpp`) [7]. Even more broadly, both Intel (with `serialize`) and ARMv8.5-A [7] (with `sb`) introduced generic speculative execution barriers.

On future CPUs with Control-flow Enforcement Technology (CET) capabilities, retpoline might trigger false positives in the CET defenses [43]. Therefore, these CPUs implement *enhanced IBRS*, a hardware defense for Spectre-BTB [43]. Intel [43] also provided a microcode update against Spectre-RSB to stop speculation. However, on Skylake and newer architectures, the RSB may fall back to the BTB, re-enabling Spectre-BTB attacks via return instructions. To prevent this, the RSB is stuffed with the address of a benign gadget when entering the kernel [43].

3.2.2 Academia. Academia helped identifying the limitations of the deployed serializing countermeasures [82]. Furthermore, they proposed techniques to reduce the overhead of such defenses.

Software-based defenses. Schwarz et al. [82] showed that *lfence* instructions only stop execution units from running subsequent operations. Thus, fetch and decode still work, potentially leaking data through the power-up of AVX functional units, the TLB, or the instruction cache. Furthermore, performance-wise, serializing every branch can be worse than using a processor without branch prediction in the first place [42]. Shen et al. [83] split code into small blocks and insert fences between the entry point and a potentially leaking memory access to mitigate Spectre-BTB and Spectre-RSB. However, an attacker could still jump unaligned into a code block, *i.e.*, directly to the memory access.

Instead of using *lfence*, Oleksenko et al. [68] propose to introduce data dependencies between branch condition operands and operations following the branch, stalling the execution of dependent instructions. Unfortunately, due to compiler re-ordering, this proposal is limited in its effectiveness.

As an alternative to retpoline and randpoline, Amit et al. [5] designed JumpSwitches, which add a shortcut path for indirect branches with a direct branch for the most likely target.

Hardware-based defenses. Vassena et al. [92] proposed to annotate variables and insert `lfences` in code paths where such variables may be leaked. To reduce the high cost of adding fences, Taram et al. [88] propose a hardware-based technique to dynamically insert fences before potentially leaking loads. Koruyeh et al. [57] argue that Spectre-BTB and Spectre-RSB usually leave the defined control-flow graph. Hence, they propose SpecCFI to repurpose control-flow integrity (CFI) to prevent speculative diversion from the control-flow graph. Capability systems may also contribute to Spectre mitigations [96].

Several designs introduce buffer flushes or hardware partitioning to isolate different domains (e.g., security enclaves) [13, 31, 69]. However, a limitation to the flushing of caches and buffers upon domain switches is that it is difficult to ensure no microarchitectural state persists. A similar argument was made for the Raspberry PI 3 [90]. However, speculative fetches may leave microarchitectural traces sufficient for an attack [9].

3.3 Data Access Prevention (Phase 3)

Preventing access to certain data during speculative execution is a promising approach to fully mitigate Spectre attacks. Solutions in this phase focus on secrets in memory. None of the solutions presented for this phase protect against Spectre attacks on data in registers.

3.3.1 Industry. Mainly, software-based defenses against data access were adopted by the industry. With process isolation, Google presented the first defense for this phase [22, 73]. Leaking secrets from other contexts is mitigated unless the attacker can utilize Meltdown to bypass process isolation permission checks.

Sanitizing values used in speculation can affect *phase 3* and *phase 4* as memory locations may be inaccessible. The idea of Speculative Load Hardening (SLH) [19] is to check loads using branchless code to ensure that it is executing along a valid control-flow path. One prerequisite for this approach is that the hardware enables branchless and unpredicted conditional updates of register values. Both LLVM and GCC support SLH today and provide a builtin function to either emit a speculation barrier or return a safe value if the instruction is transient [29].

WebKit employs two techniques to limit access to secret data [72]. First, bound checks are replaced with index masking, thus, only introducing a maximum range for the out-of-bounds violation. Second, a pseudo-random *poison value* protects pointers from misuse. Using this approach, an attacker would first have to learn the *poison value* to use it. Furthermore, mispredictions on type checks result in the wrong type being used for the pointer.

3.3.2 Academia. Academia proposed software and hardware defenses, including utilizing existing hardware technologies, e.g., Memory Protection Extensions (MPX) and Memory Protection Keys (MPK).

Software-based defenses. Narayan et al. [66] implemented a sandboxing framework for Firefox that supports process-based isolation. Furthermore, Ojogbo et al. [67] used bitmasks to arithmetically guarantee that any speculatively computed index is in bounds. Dong et al. [27] used Intel MPX for this purpose.

As a probabilistic countermeasure, Sianipar et al. [84] constantly move secret data around in virtual and physical memory. However, this only reduces the leakage rate. In contrast, many deterministic proposals also target this attack phase. Palit et al. [70] use a compiler extension that keeps annotated secret data encrypted in memory most of the time. The secret key is stored in a register. Hence, the attack surface is significantly reduced. Kiriansky and Waldspurger [54] propose to restrict access to sensitive data by using protection keys like Intel MPK technology [45]. However, as an attacker could use Spectre to disable MPK using the `wrpkru` instruction, they propose a microcode update for this instruction to include an `lfence`. Nonetheless, an attacker can still access the data if the system is susceptible to Meltdown-PK [18]. Jenkins et al. [48] propose to use ELFbac [10] or Intel MPK against Spectre attacks.

Hardware-based defenses. Schwarz et al. [79] propose multiple defenses against Spectre that all rely on the annotation of secrets. The compiler groups secret variables onto pages and marks these pages as secure. For commodity systems, they then suggest a technique called ConTExT-light [79], which uses uncacheable memory for secrets, making them inaccessible during speculative execution. Kiriansky et al. [53] propose to securely partition the cache across its ways, with protection domains that isolate on a cache hit, cache miss, and metadata level. However, this requires the correct management of these domains in software.

3.4 Data Encoding Prevention (Phase 4)

Kocher et al. [55] proposed to track data loaded during transient execution and prevent its use in subsequent operations. Several academic works propose new processor designs similar to this idea. There is still no industry solution that targets this phase.

3.4.1 Academia. NDA [97] identifies potentially leaking instructions and defers their execution if they depend on a previous, not yet retired, operation. Yu et al. [101] taint data that has not yet been committed and uses light-weight taint tracking to delay instructions that use such tainted inputs. Cabodi et al. [15] use a similar approach and verify it using model checking. Barber et al. [11] defer the wake up of dependent load instructions from when the load instruction it depends on is retired instead of when it is dispatched. Other works [32, 79] propose to annotate secrets and, thus, only track and protect secrets in registers and the cache.

3.5 Leakage Prevention (Phase 5)

Several solutions propose to speculate as usual but to either store results in new buffers or to completely remove the microarchitectural traces. Many of the proposals only focus on memory accesses and the cache. While effective against simple attacks, more sophisticated attacks may remain unaffected [12].

3.5.1 Academia. Several proposed defenses introduce shadow hardware structures for transient execution [1, 34, 51, 63, 77, 100] to squash microarchitectural state changes upon a wrong prediction. Lowe-Power et al. [61] and Saileshwar et al. [76] propose to undo modifications to the microarchitectural state after misspeculation. Li et al. [59] design a solution that targets specifically the Flush+Reload covert channel, which spreads different values to different pages and block speculative instructions that may lead to accesses to different pages. Rockicki [74] also explored a similar direction for in-order processors that use dynamic binary translation optimizations for performance. Sakalis et al. [78] propose to delay L1 misses until they are certain to be committed. Nonetheless, these proposals are vulnerable against side channels other than caches, e.g., DRAM buffers [71], or execution-unit congestion [2, 12].

3.6 Data Decoding Prevention (Phase 6)

Preventing covert channels is most likely infeasible as long as any shared resource remains. Still, several works propose to mitigate or detect Spectre by breaking or detecting the covert channel.

3.6.1 Industry. Accurate timers are a common, but not crucial, building block of covert channels to distinguish microarchitectural states. Hence, to mitigate browser-based attacks, many web browsers reduced the accuracy of timers in JavaScript [21, 65, 72, 94]. However, custom timers can always be constructed [81] and, thus, further mitigations are required [80]. After initially disabling `SharedArrayBuffers` in response to Meltdown and Spectre [21], they have been re-enabled with the introduction of site isolation [85]. This is in line with an older research direction of randomizing or reducing the resolution of timing measurements for security [39].

3.6.2 Academia. Several works propose to detect the cache covert channel used in Spectre attacks and stopping the corresponding process. Most solutions proposed so far use hardware performance counters for this purpose [26], while Sabbagh et al. [75] use memory access traces, and Austin et al. [38] use the cyclic interference property of contention-based cache leakage. However, several works show that it is trivial to evade detection [25, 49, 50, 58]. It is important to note that these proposals only consider cache covert channels.

Ge et al. [33] temporarily reduce the timer resolution whenever the cache flush interface is used. Wang et al. [95] explore varying the processor frequency to hinder native cache attacks. To alleviate the performance and energy impact, they introduce value prediction. However, value prediction is not inherently secure against Spectre attacks, and transiently diverting the control-flow of a victim by inducing a false value via value prediction effectively provides the attacker with the same capabilities. Chen et al. [20] propose to mitigate transient-execution attacks on SGX by preventing interruption of enclaves. However, an attacker does not necessarily have to interrupt an enclave to mount an attack.

4 MELTDOWN COUNTERMEASURES

Meltdown attacks exploit deliberate incorrect behavior of the hardware during transient execution. While this may have been assumed secure in the past, it must be considered a hardware bug today. We first discuss applying Spectre-focused defenses to Meltdown attacks followed by Meltdown-focused defenses.

The fundamental difference between Spectre and Meltdown type attacks is based on the transient execution trigger, prediction or fault based, respectively. Spectre-focused defenses that target *Phases 1*, *2*, and *3* cannot mitigate Meltdown attacks, as the attack runs entirely in the attacker domain. *Phase 4* defenses could be used with an additional performance cost [11, 15, 32, 79, 97, 101]. For these defenses, it is important to not just focus on cache accesses to guarantee mitigation of Meltdown attacks but more broadly prevent operations from using non-architectural and potentially secret data. *Phase 5* defenses could be used to prevent or unroll transient execution microarchitectural effects [1, 12, 34, 38, 51, 59, 61, 74, 76–78, 100]. However, mitigating the cache covert channel is not sufficient to mitigate Meltdown attacks. *Phase 6* defenses could be used to break or slow down [21, 33, 65, 72, 81, 94, 95], or detect [28, 38, 58, 75] the covert channel. However, currently, none of these defenses can guarantee the absence of Meltdown-usable covert channels.

4.1 Meltdown-Focused Defenses

4.1.1 Industry. As Meltdown attacks are considered to be a hardware bug, newer CPUs contain patches. For instance, newer Intel CPUs contain fixes for Meltdown-US, which have been reverse-engineered by Canella et al. [17]. They show that instead of returning data, access to privileged memory now returns 0.

For SGX, Intel proposes to either store secrets in uncacheable memory or to flush the L1 data cache when switching protection domains. Hypervisors similarly flush the L1 upon context switches between untrusted virtual machine threads. To prevent attacks from a VM running on a hyperthread, hypervisors implement variants of gang scheduling [41, 64]. SGX takes the hyperthreading status into account for attestation for the same reason. System Management Mode (SMM) rendezvous logical cores and flushes the L1 upon context switches.

Meltdown-GP, *i.e.*, transient reads of system registers, has been fixed via a microcode update [42]. Newer ARM CPUs are also not vulnerable to Meltdown-GP, whereas older ones can be protected via software workarounds [8]. Meltdown-NM (Lazy-FP) [86], which exploited the lazy switching of floating-point unit (FPU) registers, is mitigated by disabling lazy switching.

4.1.2 Academia. The first software-based defense against Meltdown type attacks was KAISER [36, 37], which removes the kernel mapping while running in user space. Unfortunately, on x86, some privileged memory locations must always be mapped in user space, and thus, can still be attacked [17]. KAISER was merged into Linux as kernel page-table isolation (KPTI) [23]. Other operating systems have received similar patches [35]. LAZARUS [6] pursues a similar idea but uses unmapping and re-mapping of pages upon context switching, which is problematic in multi-threaded applications.

Hua et al. [40] propose EPTI (Extended Page Table Isolation), a variant of KPTI relying on extended page tables. As there is hardware support for EPT switching and TLB entries from different EPTs are tagged, e.g., with VM process IDs (VPIDs), the performance loss is not as severe as with KPTI. However, as this approach uses extended page tables, it leaves the system vulnerable to Meltdown-P. MemoryRanger [56] isolates drivers, kernel and user space into separate address spaces using EPTs.

To mitigate Meltdown-P (Foreshadow) on commodity systems, operating systems now sanitize physical page-number fields of unmapped page-table entries [41, 98] by setting the physical page-number field to values that would refer to non-existent physical memory.

Finally, academic research shows how formal verification could more generically prevent Meltdown bugs [15, 30].

5 LVI COUNTERMEASURES

LVI attacks exploit deliberate incorrect behavior of the hardware during transient execution, similar to Meltdown. In contrast to Meltdown, LVI attacks run in the victim domain and turn the Meltdown-type leakage around into data injection. That is, the victim erroneously runs into transient execution with the injected data values, similar to Spectre. In contrast to Spectre, LVI attacks trigger the transient execution using illegal data flows instead of misprediction. In principle, any data flow can be attacked using

LVI, which, based on the attacker's capabilities, can be every load operation in the victim.

Although LVI has similarities to both Meltdown and Spectre attacks, unfortunately, the industry deployed countermeasures against Meltdown and Spectre (silicon-level and microcode), are orthogonal to LVI attacks. Specifically, Spectre defenses stop speculation around (branch) mispredictions, while LVI defenses should stop speculation around all possible illegal data flows (e.g., all loads in a program). Meltdown software and microcode defenses which flush the microarchitectural structures after victim execution [41] cannot mitigate LVI because LVI runs entirely within the victim domain. Even silicon hardening against Meltdown attacks by zeroing illegal data flows [44] do not fully eliminate the LVI threat [91].

Furthermore, applying defenses that target the covert channel that leaks the secret outside the transient execution, *Phases 4*, *5*, and *6*, could hinder LVI attacks. However, these defenses are limited, as we discussed in Section 4.

5.1 LVI-Focused Defenses

5.1.1 Industry. Intel argues that LVI is not practical in non-SGX environments because the attacker has limited ability to cause faults or assists in the victim process in such environments [46]. Therefore, Intel updated the SGX SDK (compiler and assembler-based mitigations) to enable LVI-resilient enclaves. In contrast, hardware-based mitigations could ultimately address LVI's root cause by ensuring that there are no illegal data flows from faulting or assisted loads to dependent instructions. In principle, mitigating specific Meltdown attacks would provide implicit mitigation of the corresponding LVI attacks.

5.1.2 Academia. To fully mitigate LVI attacks without hardware changes, serialization using `lfence` instructions after possibly every illegal data flow, e.g., memory load, is required [91]. However, the performance overheads of such mitigations are prohibitive, and future work has to find better security-performance trade-offs.

6 CONCLUSION

This paper shows a systematic evolution of defenses against transient-execution attacks in both industry and academia since the initial discoveries of the attacks. We show that despite the advances in the understanding and systematic view of the field, the proposed and deployed defenses are limited.

ACKNOWLEDGMENTS

This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (grant agreement No 681402). This work has been supported by the Austrian Research Promotion Agency (FFG) via the project ESPRESSO, which is funded by the province of Styria and the Business Promotion Agencies of Styria and Carinthia. Additional funding was provided by generous gifts from ARM. Any opinions, findings, and conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of the funding parties.

REFERENCES

[1] Sam Ainsworth and Timothy M Jones. 2019. MuonTrap: Preventing Cross-Domain Spectre-Like Attacks by Capturing Speculative State. *arXiv:1911.08384*.
[2] Alejandro Cabrera Aldaya, Billy Bob Brumley, Sohaib ul Hassan, Cesar Pereida García, and Nicola Tuveri. 2019. Port contention for fun and profit. In *S&P*.
[3] AMD. 2018. AMD64 Technology: Speculative Store Bypass Disable.
[4] AMD. 2018. Software Techniques for Managing Speculation on AMD Processor.
[5] Nadav Amit, Fred Jacobs, and Michael Wei. 2019. Jumpswitches: restoring the performance of indirect branches in the era of spectre. In *USENIX ATC*.
[6] Orlando Arias, David Gens, Yier Jin, Christopher Liebchen, Ahmad-Reza Sadeghi, and Dean Sullivan. 2017. LAZARUS: Practical Side-channel Resilient Kernel-Space Randomization. In *RAID*.
[7] ARM. 2013. ARM Architecture Reference Manual ARMv8.
[8] ARM. 2018. Cache Speculation Side-channels.
[9] Musard Balliu, Mads Dam, and Roberto Guanciale. 2019. InSpectre: Breaking and Fixing Microarchitectural Vulnerabilities by Formal Analysis. arXiv:1911.00868.
[10] Julian Bangert, Sergey Bratus, Rebecca Shapiro, Michael E Locasto, Jason Reeves, Sean W Smith, and Anna Shubina. 2013. *ELFbac: using the loader format for intent-level semantics and fine-grained protection.* . Dartmouth Technical Report.
[11] Kristin Barber, Anys Bacha, Li Zhou, Yinqian Zhang, and Radu Teodorescu. 2019. Specshield: Shielding speculative data from microarchitectural covert channels. In *PACT*.
[12] Atri Bhattacharyya, Alexandra Sandulescu, Matthias Neugschwandtner, Alessandro Sorniotti, Babak Falsafi, Mathias Payer, and Anil Kurmus. 2019. Smotherspectre: exploiting speculative execution through port contention. In *CCS*.
[13] Thomas Bourgeat, Ilia Lebedev, Andrew Wright, Sizhuo Zhang, and Srinivas Devadas. 2019. MI6: Secure enclaves in a speculative out-of-order processor. In *MICRO*.
[14] R Branco, K Hu, K Sun, and H Kawakami. 2019. Efficient mitigation of side-channel based attacks against speculative execution processing architectures.
[15] Gianpiero Cabodi, Paolo Camurati, Fabrizio Finocchiaro, and Danilo Vendraminetto. 2019. Model-Checking Speculation-Dependent Security Properties: Abstracting and Reducing Processor Models for Sound and Complete Verification. *Electronics* (2019).
[16] Claudio Canella, Khaled N. Khasawneh, and Daniel Gruss. 2020. The Evolution of Transient-Execution Attacks. In *GLSVLSI*.
[17] Claudio Canella, Michael Schwarz, Martin Haubenwallner, Martin Schwarzl, and Daniel Gruss. 2020. KASLR: Break It, Fix It, Repeat. In *AsiaCCS*.
[18] Claudio Canella, Jo Van Bulck, Michael Schwarz, Moritz Lipp, Benjamin von Berg, Philipp Ortner, Frank Piessens, Dmitry Evtyushkin, and Daniel Gruss. 2019. A Systematic Evaluation of Transient Execution Attacks and Defenses. In *USENIX Security*. Extended classification tree and PoCs at https://transient.fail/.
[19] Chandler Carruth. 2018. RFC: Speculative Load Hardening (a Spectre variant1 mitigation).
[20] Guoxing Chen, Mengyuan Li, Fengwei Zhang, and Yinqian Zhang. 2019. Defeating Speculative-Execution Attacks on SGX with HyperRace. In *DSC*.
[21] Chromium Projects. 2018. Actions required to mitigate Speculative Side-Channel Attack techniques.
[22] Chromium Projects. 2018. Site Isolation.
[23] Jonathan Corbet. 2017. The current state of kernel page-table isolation.
[24] Microsoft Corp. 2019. https://support.microsoft.com/en-us/help/4482887/windows-10-update-kb4482887
[25] Sai Manoj P D, Sairaj Amberkar, Sahil Bhat, Abhijitt Dhavlle, Hossein Sayadi, Avesta Sasan, Houman Homayoun, and Setareh Rafatirad. 2019. Adversarial attack on microarchitectural events based malware detectors. In *DAC*.
[26] Jonas Depoix and Philipp Altmeyer. 2018. Detecting Spectre Attacks by identifying Cache Side-Channel Attacks using Machine Learning. *WAMOS* (2018).
[27] Xiaowan Dong, Zhuojia Shen, John Criswell, Alan Cox, and Sandhya Dwarkadas. 2018. Spectres, virtual ghosts, and hardware support. In *HASP*.
[28] Swastika Dutta and Sayan Sinha. 2019. Performance statistics and learning based detection of exploitative speculative attacks. In *CF*.
[29] R Earnshaw. 2018. Mitigation against unsafe data speculation (CVE-2017-5753).
[30] Mohammad Rahmani Fadiheh, Dominik Stoffel, Clark Barrett, Subhasish Mitra, and Wolfgang Kunz. 2019. Processor hardware security vulnerabilities and their detection by unique program execution checking. In *DATE*.
[31] Andrew Ferraiuolo, Mark Zhao, Andrew C Myers, and G Edward Suh. 2018. HyperFlow: A processor architecture for nonmalleable, timing-safe information flow security. In *CCS*.
[32] Jacob Fustos, Farzad Farshchi, and Heechul Yun. 2019. SpectreGuard: An Efficient Data-centric Defense Mechanism against Spectre Attacks. In *DAC*.
[33] Jingquan Ge, Neng Gao, Chenyang Tu, Ji Xiang, and Zeyi Liu. 2019. AdapTimer: Hardware/Software Collaborative Timer Resistant to Flush-Based Cache Attacks on ARM-FPGA Embedded SoC. In *ICCD*.
[34] Abraham Gonzalez, Ben Korpan, Jerry Zhao, Ed Younis, and Krste Asanović. 2019. Replicating and Mitigating Spectre Attacks on an Open Source RISC-V Microarchitecture. In *CARRV*.
[35] Daniel Gruss, Dave Hansen, and Brendan Gregg. 2018. Kernel isolation: From an academic idea to an efficient patch for every computer. *; login: the USENIX Magazine* (2018).

[36] Daniel Gruss, Moritz Lipp, Michael Schwarz, Richard Fellner, Clémentine Maurice, and Stefan Mangard. 2017. Kaslr is dead: long live kaslr. In *ESSoS*.

[37] Daniel Gruss, Clémentine Maurice, Anders Fogh, Moritz Lipp, and Stefan Mangard. 2016. Prefetch side-channel attacks: Bypassing SMAP and kernel ASLR. In *CCS*.

[38] Austin Harris, Shijia Wei, Prateek Sahu, Pranav Kumar, Todd Austin, and Mohit Tiwari. 2019. Cyclone: Detecting Contention-Based Cache Information Leaks Through Cyclic Interference. In *MICRO*.

[39] Wei-Ming Hu. 1992. Reducing timing channels with fuzzy time. *Journal of computer security* (1992).

[40] Zhichao Hua, Dong Du, Yubin Xia, Haibo Chen, and Binyu Zang. 2018. EPTI: Efficient Defence against Meltdown Attack for Unpatched VMs. In *USENIX ATC*.

[41] Intel. 2018. Deep Dive: Intel Analysis of L1 Terminal Fault.

[42] Intel. 2018. Intel Analysis of Speculative Execution Side Channels.

[43] Intel. 2018. Retpoline: A Branch Target Injection Mitigation. Revision.

[44] Intel. 2018. Speculative Execution Side Channel Mitigations.

[45] Intel. 2019. Intel 64 and IA-32 architectures software developer's manual.

[46] Intel. 2020. Deep Dive: Load Value Injection.

[47] Intel. 2020. Side Channel Mitigation by Product CPU Model.

[48] Ira Ray Jenkins, Prashant Anantharaman, Rebecca Shapiro, J Peter Brady, Sergey Bratus, and Sean W Smith. 2020. Ghostbusting: Mitigating spectre with intraprocess memory isolation. In *HoTSoS*.

[49] Khaled N Khasawneh, Nael Abu-Ghazaleh, Dmitry Ponomarev, and Lei Yu. 2017. RHMD: Evasion-resilient hardware malware detectors. In *MICRO*.

[50] Khaled N Khasawneh, Nael B Abu-Ghazaleh, Dmitry Ponomarev, and Lei Yu. 2018. Adversarial Evasion-Resilient Hardware Malware Detectors. In *ICCAD*.

[51] Khaled N Khasawneh, Esmaeil Mohammadian Koruyeh, Chengyu Song, Dmitry Evtyushkin, Dmitry Ponomarev, and Nael Abu-Ghazaleh. 2019. Safespec: Banishing the spectre of a meltdown with leakage-free speculation. In *DAC*.

[52] Russel King. 2018. Spectre-v2: harden branch predictor on context switches.

[53] Vladimir Kiriansky, Ilia Lebedev, Saman Amarasinghe, Srinivas Devadas, and Joel Emer. 2018. DAWG: A defense against cache timing attacks in speculative execution processors. In *MICRO*.

[54] Vladimir Kiriansky and Carl Waldspurger. 2018. Speculative buffer overflows: Attacks and defenses. *arXiv:1807.03757* (2018).

[55] Paul Kocher, Jann Horn, Anders Fogh, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, et al. 2019. Spectre attacks: Exploiting speculative execution. In *S&P*.

[56] Igor Korkin. 2018. Divide et Impera: MemoryRanger Runs Drivers in Isolated Kernel Spaces. *arXiv:1812.09920* (2018).

[57] Esmaeil Mohammadian Koruyeh, Shirin Haji Amin Shirazi, Khaled N Khasawneh, Chengyu Song, and Nael Abu-Ghazaleh. 2019. SPECCFI: Mitigating Spectre Attacks using CFI Informed Speculation. *arXiv:1906.01345* (2019).

[58] Congmiao Li and Jean-Luc Gaudiot. 2020. Challenges in Detecting an "Evasive Spectre". *IEEE Computer Architecture Letters* (2020).

[59] Peinan Li, Lutan Zhao, Rui Hou, Lixin Zhang, and Dan Meng. 2019. Conditional Speculation: An effective approach to safeguard out-of-order execution against spectre attacks. In *HPCA*.

[60] Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Anders Fogh, Jann Horn, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, and Mike Hamburg. 2018. Meltdown: Reading Kernel Memory from User Space. In *USENIX Security*.

[61] Jason Lowe-Power, Venkatesh Akella, Matthew K Farrens, Samuel T King, and Christopher J Nitta. 2018. Position Paper: A case for exposing extra-architectural state in the ISA. In *HASP*.

[62] Ross Mcilroy, Jaroslav Sevcik, Tobias Tebbi, Ben L Titzer, and Toon Verwaest. 2019. Spectre is here to stay: An analysis of side-channels and speculative execution. *arXiv:1902.05178* (2019).

[63] Avi Mendelson. 2019. Secure Speculative Core. In *IEEE SOCC*.

[64] Microsoft. 2018. Microsoft Techcommunity. Hyper-V HyperClear Mitigation for L1 Terminal Fault.

[65] Microsoft. 2018. Mitigating speculative execution side-channel attacks in Microsoft Edge and Internet Explorer.

[66] Shravan Narayan, Craig Disselkoen, Tal Garfinkel, Nathan Froyd, Eric Rahm, Sorin Lerner, Hovav Shacham, and Deian Stefan. 2020. Retrofitting Fine Grain Isolation in the Firefox Renderer. In *USENIX Security*.

[67] Ejebagom John Ojogbo, Mithuna Thottethodi, and TN Vijaykumar. 2020. Secure automatic bounds checking: prevention is simpler than cure. In *CGO*.

[68] Oleksii Oleksenko, Bohdan Trach, Tobias Reiher, Mark Silberstein, and Christof Fetzer. 2018. You shall not bypass: Employing data dependencies to prevent bounds check bypass. *arXiv:1805.08506* (2018).

[69] Hamza Omar and Omer Khan. 2019. IRONHIDE: A Secure Multicore Architecture that Leverages Hardware Isolation Against Microarchitecture State Attacks. *arXiv:1904.12729* (2019).

[70] Tapti Palit, Fabian Monrose, and Michalis Polychronakis. 2019. Mitigating data leakage by protecting memory-resident sensitive data. In *ACSAC*.

[71] Peter Pessl, Daniel Gruss, Clémentine Maurice, Michael Schwarz, and Stefan Mangard. 2016. DRAMA: Exploiting DRAM addressing for cross-cpu attacks. In *USENIX Security*.

[72] Filip Pizlo. 2018. What Spectre and Meltdown mean for WebKit.

[73] Charles Reis, Alexander Moshchuk, and Nasko Oskov. 2019. Site isolation: process separation for web sites within the browser. In *USENIX Security*.

[74] Simon Rokicki. 2020. GhostBusters: Mitigating Spectre Attacks on a DBT-Based Processor. In *DATE*.

[75] Majid Sabbagh, Yunsi Fei, Thomas Wahl, and A Adam Ding. 2018. SCADET: a side-channel attack detection tool for tracking Prime+ Probe. In *ICCAD*.

[76] Gururaj Saileshwar and Moinuddin K Qureshi. 2019. CleanupSpec: An "Undo" Approach to Safe Speculation. In *MICRO*.

[77] Christos Sakalis, Mehdi Alipour, Alberto Ros, Alexandra Jimborean, Stefanos Kaxiras, and Magnus Själander. 2019. Ghost loads: what is the cost of invisible speculation?. In *CF*.

[78] Christos Sakalis, Stefanos Kaxiras, Alberto Ros, Alexandra Jimborean, and Magnus Själander. 2019. Efficient invisible speculative execution through selective delay and value prediction. In *ISCA*.

[79] Michael Schwarz, Moritz Lipp, Claudio Canella, Robert Schilling, Florian Kargl, and Daniel Gruss. 2020. ConTExT: A Generic Approach for Mitigating Spectre. In *NDSS*.

[80] Michael Schwarz, Moritz Lipp, and Daniel Gruss. 2018. JavaScript Zero: Real JavaScript and Zero Side-Channel Attacks.. In *NDSS*.

[81] Michael Schwarz, Clémentine Maurice, Daniel Gruss, and Stefan Mangard. 2017. Fantastic timers and where to find them: high-resolution microarchitectural attacks in JavaScript. In *FC*.

[82] Michael Schwarz, Martin Schwarzl, Moritz Lipp, Jon Masters, and Daniel Gruss. 2019. Netspectre: Read arbitrary memory over network. In *ESORICS*.

[83] Zhuojia Shen, Jie Zhou, Divya Ojha, and John Criswell. 2018. Restricting control flow during speculative execution. In *CCS*.

[84] Johannes Sianipar, Muhammad Sukmana, and Christoph Meinel. 2018. Moving Sensitive Data Against Live Memory Dumping, Spectre and Meltdown Attacks. In *2018 26th International Conference on Systems Engineering (ICSEng)*. IEEE.

[85] Ben Smith. 2018. Enable SharedArrayBuffer by default on non-android.

[86] Julian Stecklina and Thomas Prescher. 2018. Lazyfp: Leaking fpu register state using microarchitectural side-channels. *arXiv:1806.07480* (2018).

[87] SUSE. 2018. Security update for kernel-firmware. https://www.suse.com/support/update/announcement/2018/suse-su-20180008-1

[88] Mohammadkazem Taram, Ashish Venkat, and Dean Tullsen. 2019. Context-sensitive fencing: Securing speculative execution via microcode customization. In *ASPLOS*.

[89] Paul Turner. 2018. Retpoline: a software construct for preventing branch-target-injection.

[90] Eben Upton. 2018. Why Raspberry Pi isn't vulnerable to Spectre or Meltdown.

[91] Jo Van Bulck, Daniel Moghimi, Michael Schwarz, Moritz Lipp, Marina Minkin, Daniel Genkin, Yarom Yuval, Berk Sunar, Daniel Gruss, and Frank Piessens. 2020. LVI: Hijacking Transient Execution through Microarchitectural Load Value Injection. In *S&P*.

[92] Marco Vassena, Klaus V Gleissenthall, Rami Gökhan Kici, Deian Stefan, and Ranjit Jhala. 2020. Automatically eliminating speculative leaks with blade. *arXiv:2005.00294* (2020).

[93] Ilias Vougioukas, Nikos Nikoleris, Andreas Sandberg, Stephan Diestelhorst, Bashir M Al-Hashimi, and Geoff V Merrett. 2019. BRB: Mitigating Branch Predictor Side-Channels. In *HPCA*.

[94] Luke Wagner. 2018. Mitigations landing for new class of timing attack.

[95] Han Wang, Hossein Sayadi, Tinoosh Mohsenin, Liang Zhao, Avesta Sasan, Setareh Rafatirad, and Houman Homayoun. 2020. Mitigating Cache-Based Side-Channel Attacks through Randomization: A Comprehensive System and Architecture Level Analysis. DATE.

[96] Robert NM Watson, Jonathan Woodruff, Michael Roe, Simon W Moore, and Peter G Neumann. 2018. *Capability hardware enhanced RISC instructions (CHERI): Notes on the Meltdown and Spectre attacks*. Technical Report.

[97] Ofir Weisse, Ian Neal, Kevin Loughlin, Thomas F Wenisch, and Baris Kasikci. 2019. Nda: Preventing speculative execution attacks at their source. In *MICRO*.

[98] Ofir Weisse, Jo Van Bulck, Marina Minkin, Daniel Genkin, Baris Kasikci, Frank Piessens, Mark Silberstein, Raoul Strackx, Thomas F Wenisch, and Yuval Yarom. 2018. Foreshadow-NG: Breaking the virtual memory abstraction with transient out-of-order execution.

[99] Wenjie Xiong and Jakub Szefer. 2020. Survey of Transient Execution Attacks. *arXiv:2005.13435* (2020).

[100] Mengjia Yan, Jiho Choi, Dimitrios Skarlatos, Adam Morrison, Christopher Fletcher, and Josep Torrellas. 2018. Invisispec: Making speculative execution invisible in the cache hierarchy. In *MICRO*.

[101] Jiyong Yu, Mengjia Yan, Artem Khyzha, Adam Morrison, Josep Torrellas, and Christopher W Fletcher. 2019. Speculative Taint Tracking (STT) A Comprehensive Protection for Speculatively Accessed Data. In *MICRO*.

[102] Lutan Zhao, Peinan Li, Rui Hou, Jiazhen Li, Michael C Huang, Lixin Zhang, Xuehai Qian, and Dan Meng. 2020. A Lightweight Isolation Mechanism for Secure Branch Predictors. *arXiv:2005.08183* (2020).