Hardware-Software Codesign for Mitigating Spectre

Nicholas Mosier, Kate Eselius, Hamed Nemati*, John Mitchell, Caroline Trippel
Stanford University, *CISPA Helmholtz Center for Information Security
{nmosier,keselius,hnnemati,jcm,trippel}@stanford.edu

1 Introduction

Transient execution attacks break secure programming paradigms (e.g., constant-time, or CT, programming [2, 3, 4, 27, 33, 46, 14, 26, 37, 1, 12, 10, 13, 41], sandbox isolation [16, 28, 34, 20]) by steering secrets towards the sensitive operands of (transient) transmitters[1] that leak them [21, 23]. These attacks leverage two high-level mechanisms for creating *transient execution*[2] at runtime on modern processors: (i) *faulting* instructions, and (ii) control- or data-flow *mispredictions.* These mechanisms give rise to *Meltdown attacks* and *Spectre attacks*, respectively [6].

Spectre attacks [21, 22, 17, 15, 8] (our focus) are characterized according to distinct sources of control- and data-flow (mis)prediction on modern processors, called *speculation primitives* [25, 6]. Predictions introduce *speculative execution*, which may turn out to be *sequential* (when predictions are correct) or *transient* (when predictions are incorrect). On modern processors, there are five well-documented speculation primitives: conditional branch prediction (PHT) [21], indirect branch prediction (BTB) [21], return address prediction (RSB) [22], store-to-load forwarding prediction (STL) [17], and predictive store forwarding (PSF) [9, 15, 30].

The Spectre Mitigation Challenge. Comprehensively mitigating Spectre leakage of program secrets is hard, and doing so while preserving performance is even harder.

Prior work has proposed a variety of **software mitigations** targeting existing hardware [7, 38, 28, 18, 39, 40, 34, 20]; however, *none* of these are comprehensive for any threat model. To our knowledge, only one mitigation is known to be widely deployed in practice: retpoline [38, 35], for Spectre-BTB. However, retpoline has been shown to introduce other (non-BTB) Spectre vulnerabilities [43].

Existing **hardware mitigations** for Spectre fall into two categories: (i) comprehensive hardware mitigations and (ii) hardware speculation controls. Comprehensive hardware mitigations to Spectre attacks require little to no software cooperation [11, 45, 24, 31, 36, 44, 42]. Yet, they have not been deployed in practice, likely due to the complexity of the microarchitectural changes they require. Hardware speculation controls, exposed by hardware vendors like Intel and AMD, support per-process disabling of (a few) particular speculation primitives in software. For example, SSBD, PSFD, and IPRED_DIS [19] controls disable STL, PSF, BTB, respectively. Yet, they are rarely enabled, likely due to their high performance overhead; the Linux kernel leaves them disabled by default. Besides, they do not offer comprehensive Spectre protections.

Notably, even if hardware speculation controls are combined with software mitigations for speculation primitives which cannot be disabled (i.e., for PHT), a comprehensive Spectre attack mitigation is still not achieved (due to RSB).

Our Vision. Our view is that a comprehensive, efficient, and low-complexity mitigation for Spectre attacks requires engaging in *software-compiler-hardware co-design.* Our goal is to develop such a co-designed mitigation that can be widely deployed in security-critical applications like the Linux kernel or OpenSSL with little to no performance overhead.

2 Software-Compiler-Hardware Co-Design

In this talk, we will first discuss the inherent trade-offs that arise when mitigating Spectre attacks exclusively in software *or* hardware (summarized here).

2.1 Mitigating Spectre in Software

Advantages. Mitigating Spectre in software has two distinct advantages.

First, software mitigations are more easily tailored to an application-level threat model. For example, one software mitigation may be tailored to prevent an *untrusted sandboxed program* from transiently accessing and leaking secrets outside its sandbox. Another may be tailored to prevent a *trusted CT program* from transiently leaking any secrets it processes. Such application-specific threat model information is lost when programs are compiled to run on commodity hardware.

Second, software mitigations can *in theory* (with knowledge of hardware speculation primitives) perform *static program analyses* to identify values that may be secret *exclusively* during transient execution. They can further identify transmitters whose operands are dependent on (and thus may leak) transiently secret values. Thus, software has the opportunity to insert fewer, more targeted mitigations than hardware which cannot perform such analyses in advance.

Disadvantages. Unfortunately, the potential of software mitigations are limited by two major factors.

First, most processors exhibit unconstrained speculative control- and data-flow, which renders static software analyses intractable for all Spectre variants except Spectre-PHT. For example, unconstrained indirect branch and return address prediction may direct speculative control-flow to *any*

[1] *Transmitters* are *unsafe* instructions, whose execution creates operand-dependent hardware resource usage.
[2] I.e., execution of instructions that are never architecturally committed [6].

program instruction or even to the *middle* of an instruction [5]. Newer Intel processors have introduced hardware speculative control-flow restrictions via the *Control-flow Enforcement Technology (CET)* extension [32]; new work [28, 29] uses Intel CET to provide the first non-naive software mitigations for Spectre-BTB and -RSB. However, the only way to restrict speculative data-flow on current hardware is to disable data-flow speculation entirely via the SSBD and PSFD speculation controls [19], which incurs significant overhead for general purpose applications in our experience.

Second, only *coarse-grained* mitigations are made available to software by hardware vendors: (i) the LFENCE serialization instruction, which requires that *all prior instructions* commit before any subsequent instructions begin execution; and (ii) speculation controls [19] which disable a particular speculation primitive altogether. To achieve higher precision than what existing ISA mitigations afford, some software mitigations (e.g., SLH [7] for Spectre-PHT) resort to code transformations which are less expensive than using ISA mitigations but still incur unacceptable performance overhead.

2.2 Mitigating Spectre in Hardware

Advantages. Hardware-based Spectre mitigations have two key strengths.

First, hardware is fully aware of all aspects of a program's speculative execution behavior. Thus, it can selectively insert mitigations exactly when particular runtime conditions are satisfied. In contrast, software must *conservatively assume* that undesirable runtime conditions will manifest in *some* execution of a program due to limited precision of static program analyses (e.g., memory alias analysis).

Second, hardware-based approaches can insert *fine-grained* mitigations that stall the execution of individual instructions (e.g., secret-dependent transmitters) rather than the entire pipeline (like x86's LFENCE). As a result, each dynamically inserted hardware mitigation has a modest impact on the program's performance (unlike LFENCEs inserted by software).

Disadvantages. On the other hand, comprehensive hardware mitigations to Spectre are largely software-unaware. As a result, they must make conservative or unsound assumptions about the program that result in either high performance overhead due to over-mitigation or incomplete security guarantees, respectively. For example, due to the absence of software hints, most existing hardware mitigations conservatively assume *all* (speculatively or non-speculatively) accessed data is secret [44, 36, 31, 42, 24], and some unsoundly (for some threat models) assume only speculatively accessed data is secret [45]. Furthermore, since they are software-unaware, these mitigations must implement dynamic analyses like taint tracking that require complex design changes.

2.3 The Best of Both Worlds

Through studying existing Spectre attack mitigations, we identify two core opportunities for co-design.

Hardware-Software. We identify a software-hardware coordination bottleneck. First, hardware does not enable software to communicate its precise mitigation requirements. E.g., we find that speculative data-flow may only require mitigation at runtime if the relevant load and store access (logically) distinct stack frames. Second, hardware cannot offload program analyses to software, rendering it unable to leverage important program metadata (e.g., security types).

Software-Compiler. We also observe that the widely-deployed CT programming approach (in particular) permits code patterns that make efficient mitigations of Spectre in software impractical. This issue represents a lack of coordination between software and compilers. We will elaborate on the details of this programming contract problem in our talk and our proposed (more performant) solution.

3 A Comprehensive and Verified Spectre Attack Mitigation

Our talk will pitch a software-compiler-hardware co-designed Spectre mitigation that is comprehensive, proven-correct, and widely deployable at a low cost in security-critical applications.

Preliminary Work. As a first step, we have developed Serberus, a comprehensive and proven-correct Spectre mitigation for CT code that targets existing hardware. Serberus is based on the following.

First, we define an *operational semantics*, ASP, that encodes all (sequential and transient) control- and data-flow that a program may exhibit when it runs on a CET-enabled Intel microarchitecture. Second, we propose *static constant-time (CTS) programming*, a strengthening of CT programming that enables efficient software mitigations of Spectre. Third, with the support of ASP and CTS programming, we propose Serberus, the first (proven-correct) mitigation to harden CT code (satisfying CTS) against all Spectre attacks exploiting: PHT, BTB, RSB, STL, and/or PSF.

Next Steps. Serberus excels at mitigating constant-time code (e.g., crypto libraries), producing more performant and secure code than other state-of-the-art software mitigations. These performance benefits do not carry over to other application domains, however, like the Linux kernel. To improve Serberus's performance, we are exploring the following.

First, the overwhelming majority of Serberus's mitigation overhead comes from the coarse-grained ISA mitigations it inserts. We are exploring how to overcome this problem with fine-grained ISA mitigations that require *minimal* microarchitectural modifications. Second, while our hardware model ASP features low-cost control-flow restrictions, realizing our model in existing hardware requires disabling predictive store forwarding (PSF), which otherwise introduces unconstrained speculative data-flow into the program. We are currently exploring lower-cost alternatives for selectively restricting such data-flow in hardware.

References

[1] Adil Ahmad, Kyungtae Kim, Muhammad Ihsanulhaq Sarfaraz, and Byoungyoung Lee. 2018. Obliviate: a data oblivious filesystem for intel sgx. In *Network and Distributed System Security Symposium*.

[2] Gilles Barthe, Gustavo Betarte, Juan Campo, Carlos Luna, and David Pichardie. 2014. System-level non-interference for constant-time cryptography. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*.

[3] Daniel J. Bernstein. 2006. Curve25519: new diffie-hellman speed records. In *Public Key Cryptography - PKC 2006*. Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, and Tal Malkin, (Eds.)

[4] Daniel J. Bernstein. 2005. The poly1305-aes message-authentication code. In *Fast Software Encryption*. Henri Gilbert and Helena Handschuh, (Eds.)

[5] Atri Bhattacharyya, Andrés Sánchez, Esmaeil M. Koruyeh, Nael Abu-Ghazaleh, Chengyu Song, and Mathias Payer. 2020. SpecROP: speculative exploitation of ROP chains. In *23rd International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2020)*.

[6] 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 *28th USENIX Security Symposium (USENIX Security 19)*, 249–266.

[7] Chandler Carruth. 2020. Cryptographic software in a post-spectre world. talk at the real world crypto symposium. https://chandlerc.blog/talks/2020_post_spectre_crypto/post_spectre_crypto.html. Accessed October 2022. (2020).

[8] Sunjay Cauligi, Craig Disselkoen, Klaus v. Gleissenthall, Dean Tullsen, Deian Stefan, Tamara Rezk, and Gilles Barthe. 2020. Constant-time foundations for the new spectre era. In *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation*.

[9] Sunjay Cauligi, Craig Disselkoen, Daniel Moghimi, Gilles Barthe, and Deian Stefan. 2022. Sok: practical foundations for software spectre defenses. In *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 666–680.

[10] Sunjay Cauligi, Gary Soeller, Fraser Brown, Brian Johannesmeyer, Yunlu Huang, Ranjit Jhala, and Deian Stefan. 2017. Fact: a flexible, constant-time programming language. In *2017 IEEE Cybersecurity Development (SecDev)*. IEEE, 69–76.

[11] Rutvik Choudhary, Jiyong Yu, Christopher Fletcher, and Adam Morrison. 2021. Speculative privacy tracking (spt): leaking information from speculative execution without compromising privacy. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*.

[12] Bart Coppens, Ingrid Verbauwhede, Koen De Bosschere, and Bjorn De Sutter. 2009. Practical mitigations for timing-based side-channel attacks on modern x86 processors. In *2009 30th IEEE Symposium on Security and Privacy*.

[13] Sushant Dinesh, Grant Garrett-Grossman, and Christopher W. Fletcher. 2022. SynthCT: towards portable constant-time code. In *NDSS*.

[14] Saba Eskandarian and Matei Zaharia. 2019. Oblidb: oblivious query processing for secure databases. *Proc. VLDB Endow.*

[15] Roberto Guanciale, Musard Balliu, and Mads Dam. 2020. InSpectre: breaking and fixing microarchitectural vulnerabilities by formal analysis. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*.

[16] Marco Guarnieri, Boris Köpf, Jan Reineke, and Pepe Vila. 2021. Hardware-software contracts for secure speculation. In *2021 IEEE Symposium on Security and Privacy*.

[17] Jann Horn. 2018. Speculative execution, variant 4: speculative store bypass. https://bugs.chromium.org/p/project-zero/issues/detail?id=1528. (2018).

[18] Intel. 2018. Analysis of Speculative Execution Side Channels. https://newsroom.intel.com/wp-content/uploads/sites/11/2018/01/Intel-Analysis-of-Speculative-Execution-Side-Channels.pdf. (2018).

[19] Intel. 2022. CPUID Enumeration and Architectural MSRs. https://www.intel.com/content/www/us/en/developer/articles/technical/software-security-guidance/technical-documentation/cpuid-enumeration-and-architectural-msrs.html. (2022).

[20] 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 *Proceedings of the 7th Symposium on Hot Topics in the Science of Security*, 1–11.

[21] Paul Kocher et al. 2018. Spectre attacks: exploiting speculative execution. *CoRR*, abs/1801.01203. https://arxiv.org/abs/1801.01203.

[22] Esmaeil Mohammadian Koruyeh, Khaled N. Khasawneh, Chengyu Song, and Nael B. Abu-Ghazaleh. 2018. Spectre returns! Speculation attacks using the return stack buffer. *12th USENIX Workshop on Offensive Technologies (WOOT)*.

[23] Moritz Lipp et al. 2018. Meltdown. *CoRR*, abs/1801.01207. https://arxiv.org/abs/1801.01207.

[24] Kevin Loughlin, Ian Neal, Jiacheng Ma, Elisa Tsai, Ofir Weisse, Satish Narayanasamy, and Baris Kasikci. 2021. Dolma: securing speculation with the principle of transient non-observability. In *USENIX Security Symposium*, 1397–1414.

[25] Matt Miller. 2018. Mitigating speculative execution side channel hardware vulnerabilities. https://tinyurl.com/23xy63dt. (2018).

[26] Pratyush Mishra, Rishabh Poddar, Jerry Chen, Alessandro Chiesa, and Raluca Ada Popa. 2018. Oblix: an efficient oblivious search index. In *2018 IEEE Symposium on Security and Privacy (SP)*.

[27] David Molnar, Matt Piotrowski, David Schultz, and David Wagner. 2006. The program counter security model: automatic detection and removal of control-flow side channel attacks. In *Information Security and Cryptology - ICISC 2005*. Dong Ho Won and Seungjoo Kim, (Eds.)

[28] Shravan Narayan et al. 2021. Swivel: hardening WebAssembly against spectre. In *30th USENIX Security Symposium (USENIX Security 21)*.

[29] Oleksii Oleksenko, Christof Fetzer, Boris Köpf, and Mark Silberstein. 2022. Revizor: testing black-box cpus against speculation contracts. In *ASPLOS '22: 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Lausanne, Switzerland, 28 February 2022 - 4 March 2022*, 226–239. DOI: 10.1145/3503222.3507729.

[30] Hernán Ponce-de-León and Johannes Kinder. 2022. Cats vs. spectre: an axiomatic approach to modeling speculative execution attacks. In *Proceedings of the 43rd IEEE Symposium on Security and Privacy*.

[31] Michael Schwarz, Moritz Lipp, Claudio Canella, Robert Schilling, Florian Kargl, and Daniel Gruss. 2020. Context: a generic approach for mitigating spectre. In *Network and Distributed System Security Symposium*.

[32] Vedvyas Shanbhogue, Deepak Gupta, and Ravi Sahita. 2019. Security analysis of processor instruction set architecture for enforcing control-flow integrity. In *Proceedings of the 8th International Workshop on Hardware and Architectural Support for Security and Privacy* (HASP '19) Article 8. Association for Computing Machinery, Phoenix, AZ, USA, 11 pages. ISBN: 9781450372268. DOI: 10.1145/3337167.3337175.

[33] Fahad Shaon, Murat Kantarcioglu, Zhiqiang Lin, and Latifur Khan. 2017. Sgx-bigmatrix: a practical encrypted data analytic framework with trusted processors. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*.

[34] Zhuojia Shen, Jie Zhou, Divya Ojha, and John Criswell. 2019. Restricting control flow during speculative execution with venkman. *arXiv preprint arXiv:1903.10651*.

[35] Ben Treynor Sloss. 2018. Protecting our google cloud customers from new vulnerabilities without impacting performance. https://www.blog.google/products/google-cloud/protecting-our-google-cloud-

customers-new-vulnerabilities-without-impacting-performance/. (2018).

[36] Mohammadkazem Taram, Ashish Venkat, and Dean Tullsen. 2019. Context-sensitive fencing: securing speculative execution via microcode customization. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*.

[37] Shruti Tople and Prateek Saxena. 2017. On the trade-offs in oblivious execution techniques. In *Detection of Intrusions and Malware, and Vulnerability Assessment*. Michalis Polychronakis and Michael Meier, (Eds.)

[38] Paul Turner. 2018. Retpoline: a software construct for preventing branch-target-injection. https://support.google.com/faqs/answer/7625886. Accessed October 2022. (2018).

[39] Marco Vassena, Craig Disselkoen, Klaus von Gleissenthall, Sunjay Cauligi, Rami Gökhan Kıcı, Ranjit Jhala, Dean Tullsen, and Deian Stefan. 2021. Automatically eliminating speculative leaks from cryptographic code with blade. *Proc. ACM Program. Lang.*

[40] Guanhua Wang, Sudipta Chattopadhyay, Ivan Gotovchits, Tulika Mitra, and Abhik Roychoudhury. 2019. Oo7: low-overhead defense against spectre attacks via program analysis. *IEEE Transactions on Software Engineering*, 47, 11, 2504–2519.

[41] Conrad Watt, John Renner, Natalie Popescu, Sunjay Cauligi, and Deian Stefan. 2019. Ct-wasm: type-driven secure cryptography for the web ecosystem. *Proceedings of the ACM on Programming Languages*, 3, POPL, 1–29.

[42] Ofir Weisse, Ian Neal, Kevin Loughlin, Thomas F Wenisch, and Baris Kasikci. 2019. Nda: preventing speculative execution attacks at their source. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, 572–586.

[43] Johannes Wikner and Kaveh Razavi. 2022. RETBLEED: arbitrary speculative code execution with return instructions. In *31st USENIX Security Symposium (USENIX Security 22)*. USENIX Association, Boston, MA, (Aug. 2022), 3825–3842. ISBN: 978-1-939133-31-1. https://www.usenix.org/conference/usenixsecurity22/presentation/wikner.

[44] Mengjia Yan, Jiho Choi, Dimitrios Skarlatos, Adam Morrison, Christopher Fletcher, and Josep Torrellas. 2018. Invisispec: making speculative execution invisible in the cache hierarchy. In *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 428–441.

[45] 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 *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*.

[46] Wenting Zheng, Ankur Dave, Jethro G. Beekman, Raluca Ada Popa, Joseph E. Gonzalez, and Ion Stoica. 2017. Opaque: an oblivious and encrypted distributed analytics platform. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*.