Internships on Formally Secure Compilation Against Spectre Attacks

Advisor: Catalin Hritcu, MPI-SP, Bochum, Germany

November 12, 2025

Introduction

The Spectre speculative side-channel attacks are a formidable threat for the security of software systems [9, 22]. Without any defenses, attackers can easily construct "universal read gadgets" that leak a sensitive program's entire memory [22, 27]. Yet strong and efficient Spectre defenses are notoriously challenging to implement. Even the Linux kernel, for which many developers have spent years to implement effective Spectre defenses, still sometime falls pray to Spectre attacks, including universal read gadgets [37, 38], and the situation seems worse when developers have fewer resources and rely entirely on the insufficient defenses currently provided by the OSes and compilers. Some of these attacks exploit new Spectre variants that current defenses don't deal with [20, 21, 39], but other attacks simply exploit implementation mistakes, bad security-efficiency tradeoffs, and unclear and overly optimistic assumptions that developers often make when implementing Spectre defenses, which often involve a complex combination of software in compilers and OSes as well as hardware features and models, which are all very difficult to get right.

We believe that formally secure compilation can play a central role in building watertight Spectre defenses that are strong and fit well together. This will raise the security bar and go beyond increasing attacker effort to achieving an end-to-end security notion with respect to mathematically specified attacker models, which guarantees the elimination of entire classes of Spectre attacks. Yet formally secure compilation against Spectre attacks [2, 3, 5, 28, 36] is very challenging and has not been achieved for realistic compilers before. The goal of this project is to go beyond the current state of the art and to build the first realistic, formally secure compiler that protects arbitrary programs written in general-purpose programming languages like C/C++ and Rust against all main Spectre variants (i.e., Spectre PHT, BTB, RSB, STL, and PSF [9]).

To accomplish this, we will devise a secure LLVM variant, enforcing that any compiled program running with speculation on a modern x86 processor does not leak more than what the source program leaks sequentially (**RO1**). This relative security property provides guarantees for arbitrary source programs, irrespective of whether they satisfy the cryptographic constant-time discipline or not. To achieve this property efficiently, our compiler will use a combination of user annotations for secret inputs, static information-flow analysis, selective software defenses, careful compiler optimization, and hardware features (e.g., Intel CET [19]). To make sure that efficient enforcement does not compromise security, we will build an effective property-based testing (PBT) framework for relative security (**RO2**) and use it to systematically validate that our LLVM variant is secure with respect to precise hardware/software contracts [16] of modern x86 processors [30]. For the most interesting techniques of our compiler, we will also formalize simplified models in Rocq [35] and construct machine-checked proofs of relative security (**RO3**).

Previous Work Building watertight and efficient Spectre defenses is very challenging [10], but cryptographic engineering researchers have shown that this is possible [2, 3, 5, 28, 36]. They have proposed specialized defenses such as *selective* Speculative Load Hardening (SLH), which efficiently achieve speculative constant-time guarantees against Spectre PHT (v1) for crypto code with typical overheads under

1% [33, 34]. They have also proposed defenses efficiently protecting crypto code against multiple Spectre variants [3, 28], with typical overheads 2-7% [3] or around 21% [28]. This work is, however, very much specialized to only crypto code (e.g., code that does not use any indirect jumps [3] or that cannot be securely linked with non-cryptographic code [23] unless it runs in a different process [3, 28]) and often also to domain-specific languages for cryptography, such as Jasmin [1, 2, 3]. These ideas have been implemented in prototype compilers aimed at enforcing and preserving speculative constant time [2, 3, 28], yet even these domain-specific compilers for crypto code provide no *formal* secure compilation guarantees at scale.

The idea of asking developers to annotate secret inputs and using that information to provide selective Spectre defenses is, however, general and can be applied beyond cryptography. In recent work, we introduced Flexible SLH [5], which generalizes Selective SLH to arbitrary programs, and also proposes a suitable relative security definition for such transformations protecting arbitrary programs: any transformed program running with speculation does not leak more than what the source program leaks sequentially. We prove in Rocq, not only relative security for Flexible SLH against Spectre PHT attacks, but also that Flexible SLH is a generalization of both Selective SLH [33] and Ultimate SLH [44]. The connection to Selective SLH guarantees that code following the constant-time discipline pays no extra performance penalty under Flexible SLH compared to Selective SLH, which as mentioned above is very efficient. Flexible SLH [5] received a Distinguished Paper Award at CSF'25 and is a starting point for the efficient defenses of topics 1-1 and 1-2.

Internship topics

Research Objective 1 (RO1): Realistic Compiler Efficiently Enforcing Relative Security

Topic 1-1: Efficient Spectre PHT Defenses for Arbitrary LLVM Programs Our defenses will be based on Flexible SLH [5], yet implementing this for LLVM is technically challenging, even if we should be able to build on the implementation of Ultimate SLH [44]. Flexible SLH additionally requires adding a flow-sensitive IFC analysis to LLVM [18], keeping track of which program inputs are secret and which ones not throughout the compiler chain, and making sure that the defenses we add are not removed by subsequent compiler passes. Since some of this work is useful for both crypto and non-crypto code alike, we are teaming up with Zhang and Barthe, who are interested in implementing selective SLH in LLVM for use with crypto code, based on a static analysis for constant time they recently build for LLVM [43].

Achieving relative security for LLVM requires not only enforcing that speculation does not add extra leakage, but also that LLVM does not introduce *sequential* leakage. Unfortunately, LLVM can introduce sequential leakage [32, 43] and this problem seems to be getting worse, as the compiler is getting smarter at optimizing code [40]. Fortunately, only a small set of optimization passes are the root of most observed leakage [15, 32, 40, 43] (e.g., instcombine, inline), and these passes can be disabled via flags with minimal performance overhead [15], or maybe even *selectively* disabled, based on the secrecy annotations and information-flow analysis above. This is a best-effort mitigation though, so to gain confidence that it works, we will use the framework from RO2 to test relative security for the *sequential* execution model of x86 (which is easy to do with Revizor [30]). Finally, one way to reduce the overhead of our speculative defenses is using LLVM's analysis and optimization framework, for instance to remove unnecessary tracking and masking in SLH, but this needs to be done very carefully to not break relative security, which we will check using testing framework from RO2.

Topic 1-2: Complete Spectre Protection Spectre BTB and RSB attacks and are very powerful, because they speculatively hijack control flow and exploit arbitrary gadgets in the code to leak secrets [7], even in the presence of many mitigations [38]. To efficiently defend against these attacks we will combine hardware CFI protection (Intel CET [19]) and software defenses. In particular, we will introduce SpecIBT: a forward-edge CFI protection specifically designed against Spectre BTB attacks and that precisely detects mispeculation, as opposed to previous attempts at retrofitting fine-grained CFI to Spectre [14, 24], which seem to disregard that Spectre has a different attacker model and that fine-grained CFI is necessarily approximate, which still

allows for practical Spectre BTB attacks [38].

Spectre STL and PSF can be mitigated on x86 by turning off the corresponding speculation using hardware controls [3, 29]. For Serberus [28], this was the most efficient defense against PSF, while against STL they report a more efficient software solution.

We will experimentally evaluate our LLVM variant on realistic workloads, showing that it matches the efficiency of state-of-the-art solutions that are specialized for cryptographic code [3, 28], while being general enough to efficiently protect arbitrary programs handling confidential data (e.g., sshd, nginx, MySQL, Chrome). Our aim here is under 25% overhead for arbitrary programs and under 5% for cryptographic code.

Research Objective 2 (RO2): Property-Based Testing (PBT) Framework for Relative Security on x86

Topic 2-1: Generating LLVM IR programs We will test relative security for randomly generated LLVM IR programs. Yet generating programs that exercise all LLVM features and find bugs in Spectre defenses, while not triggering undefined behavior is nontrivial. We plan to build on GenLLVM [6], a recent QuickChick generator for LLVM IR that compared to previous work [11, 25, 26, 41] produces more diverse code, with more complex control flow (recursion, loops, calls), dynamic memory management, casts between integers and pointers, etc. GenLLVM is based on Vellvm, which, moreover, provides an executable semantics to LLVM IR, directly usable for testing [42]. We will extend GenLLVM to shrinking large randomly generated programs to easily understandable counterexamples [31].

Topic 2-2: Checking Speculative Indistinguishability with Revizor At the lowest level, we will use Revizor to check that the compiled program does not leak with respect to precise speculation contracts of x86 processors [30]. We will check this for many pairs of inputs that are indistinguishable by the sequential semantics of the source program. We will do this by first generating an input with the AFL++ software fuzzer [13] and then using dynamic taint tracking to determine which input parts are sequentially leaked and should thus be kept the same when generating an indistinguishable input. We will use the newest version of Revizor, based on DynamoRIO [8] which in our initial experiments proved more efficient and easier to use than previous versions, and even more so compared to LmSpec/LmTest [4].

Revizor provides speculation contracts for Spectre PHT and STL, and new contracts are required for the other Spectre variants we target: BTB, RSB, and PSF [9]. LmSpec/LmTest [4] showed how to build speculation contracts supporting more variants (e.g., RSB), still extending Revizor to support BTB is an open challenge, because misspeculated indirect jumps can go anywhere and testing against a model with unrestricted nondeterminism is very hard. We expect that testing software against new speculation contracts modeling the more restricted mispeculation allowed by Intel CET [19] would be more tractable.

We will evaluate the quality of our testing not only by gathering statistics, but also by systematically studying large numbers of realistic bugs that violate relative security in the enforcement mechanisms from RO1. We will improve the quality of our testing until all introduced bugs are found and no violations are found without the studied bugs, which will yield practical confidence that relative security indeed holds [17].

Research Objective 3 (RO3): Machine-Checked Proofs for Simplified Models

Topic 3-1: Formally Verified Protection Against All Main Spectre Variants We will prove in Rocq the security of the SpecIBT defense (Topic 1-2) against Spectre PHT, BTB, and RSB attacks. This requires extending our proof technique [5] to a lower-level language with jumps between basic blocks. We will then extend the formalization with data speculation attacks and defenses and devise a relative security proof in Rocq against all main Spectre variants. The only such mechanized proof was done recently for a defense very much specialized for crypto, which did not allow for indirect jumps [3]. Our proof will also be novel compared to the composition framework recently proposed by Fabian et al. [12], which is neither mechanized nor can it deal with SLH-based defenses. Before starting these proofs, we will use QuickChick testing to find bugs and build intuition [17].

References

[1] J. B. Almeida, M. Barbosa, G. Barthe, A. Blot, B. Grégoire, V. Laporte, T. Oliveira, H. Pacheco, B. Schmidt, and P. Strub. Jasmin: High-assurance and high-speed cryptography. *CCS*. 2017.

[2] S. Arranz Olmos, G. Barthe, L. Blatter, B. Grégoire, and V. Laporte. Preservation of speculative constant-time by compilation. *PACMPL*, 9(POPL), 2025.

[3] S. Arranz-Olmos, G. Barthe, C. Chuengsatiansup, B. Grégoire, V. Laporte, T. Oliveira, P. Schwabe, Y. Yarom, and Z. Zhang. Protecting cryptographic code against Spectre-RSB: (and, in fact, all known spectre variants). *ASPLOS*. 2025.

[4] G. Barthe, M. Böhme, S. Cauligi, C. Chuengsatiansup, D. Genkin, M. Guarnieri, D. M. Romero, P. Schwabe, D. Wu, and Y. Yarom. Testing side-channel security of cryptographic implementations against future microarchitectures. *CCS*. 2024.

[5] J. Baumann, R. Blanco, L. Ducruet, S. Harwig, and C. Hritcu. FSLH: flexible mechanized speculative load hardening. *CSF*. 2025.

[6] C. Beck, H. Chen, and S. Zdancewic. Vellvm: Formalizing the informal LLVM. *NFM*. 2025.

[7] A. Bhattacharyya, A. Sánchez, E. M. Koruyeh, N. B. Abu-Ghazaleh, C. Song, and M. Payer. SpecROP: Speculative exploitation of ROP chains. *RAID*. 2020.

[8] D. Bruening. *Efficient, Transparent, and Comprehensive Runtime Code Manipulation*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, USA, 2004.

[9] C. Canella, J. V. Bulck, M. Schwarz, M. Lipp, B. von Berg, P. Ortner, F. Piessens, D. Evtyushkin, and D. Gruss. A systematic evaluation of transient execution attacks and defenses. *USENIX Security*. 2019.

[10] S. Cauligi, C. Disselkoen, D. Moghimi, G. Barthe, and D. Stefan. SoK: Practical foundations for software Spectre defenses. *IEEE SP*. 2022.

[11] J. Chen, J. Patra, M. Pradel, Y. Xiong, H. Zhang, D. Hao, and L. Zhang. A survey of compiler testing. *ACM Comput. Surv.*, 53(1), 2021.

[12] X. Fabian, M. Patrignani, M. Guarnieri, and M. Backes. Do you even lift? strengthening compiler security guarantees against Spectre attacks. *Proc. ACM Program. Lang.*, 9(POPL), 2025.

[13] A. Fioraldi, D. C. Maier, H. Eißfeldt, and M. Heuse. AFL++ : Combining incremental steps of fuzzing research. *WOOT*. 2020.

[14] A. J. Gaidis, J. Moreira, K. Sun, A. Milburn, V. Atlidakis, and V. P. Kemerlis. FineIBT: Fine-grain control-flow enforcement with indirect branch tracking. *RAID*. 2023.

[15] A. Geimer and C. Maurice. Fun with flags: How compilers break and fix constant-time code. *CoRR*, abs/2507.06112, 2025.

[16] M. Guarnieri, B. Köpf, J. Reineke, and P. Vila. Hardware-software contracts for secure speculation. *IEEE SP*. 2021.

[17] C. Hriţcu, L. Lampropoulos, A. Spector-Zabusky, A. Azevedo de Amorim, M. Dénès, J. Hughes, B. C. Pierce, and D. Vytiniotis. Testing noninterference, quickly. *JFP*, 26, 2016.

[18] S. Hunt and D. Sands. On flow-sensitive security types. *POPL*. 2006.

[19] Intel Corporation. Intel® 64 and IA-32 architectures software developer's manual: Volume 1: Basic architecture: Chapter 18 Control-Flow Enforcement Technology (CET), 2025.

[20] J. Kim, J. Chuang, and D. Genkin. FLOP: Breaking the Apple M3 CPU via false load output predictions. To appear in USENIX Security, 2025.

[21] J. Kim, D. Genkin, and Y. Yarom. SLAP: data speculation attacks via load address prediction on Apple silicon. *IEEE SP*. 2025.

[22] P. Kocher, J. Horn, A. Fogh, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, and Y. Yarom. Spectre attacks: Exploiting speculative execution. *IEEE SP*. 2019.

[23] M. Kolosick, B. A. Shivakumar, S. Cauligi, M. Patrignani, M. Vassena, R. Jhala, and D. Stefan. Robust constant-time cryptography. *Proc. ACM Program. Lang.*, 9(PLDI), 2025.

[24] E. M. Koruyeh, S. H. A. Shirazi, K. N. Khasawneh, C. Song, and N. B. Abu-Ghazaleh. SpecCFI: Mitigating Spectre attacks using CFI informed speculation. *IEEE SP*. 2020.

[25] V. Le, M. Afshari, and Z. Su. Compiler validation via equivalence modulo inputs. *PLDI*. 2014.

[26] V. Livinskii, D. Babokin, and J. Regehr. Random testing for C and C++ compilers with YARPGen. *Proc. ACM Program. Lang.*, 4(OOPSLA), 2020.

[27] R. McIlroy, J. Sevcík, T. Tebbi, B. L. Titzer, and T. Verwaest. Spectre is here to stay: An analysis of side-channels and speculative execution. *CoRR*, abs/1902.05178, 2019.

[28] N. Mosier, H. Nemati, J. C. Mitchell, and C. Trippel. Serberus: Protecting cryptographic code from Spectres at compile-time. *IEEE SP*. 2024.

[29] S. Narayan, C. Disselkoen, D. Moghimi, S. Cauligi, E. Johnson, Z. Gang, A. Vahldiek-Oberwagner, R. Sahita, H. Shacham, D. M. Tullsen, and D. Stefan. Swivel: Hardening WebAssembly against Spectre. *USENIX Security*. 2021.

[30] O. Oleksenko, C. Fetzer, B. Köpf, and M. Silberstein. Revizor: testing black-box CPUs against speculation contracts. *ASPLOS*. 2022.

[31] J. Regehr, Y. Chen, P. Cuoq, E. Eide, C. Ellison, and X. Yang. Test-case reduction for C compiler bugs. *PLDI*. 2012.

[32] M. Schneider, D. Lain, I. Puddu, N. Dutly, and S. Capkun. Breaking bad: How compilers break constant-time implementations. To appear at AsiaCCS, 2025.

[33] B. A. Shivakumar, J. Barnes, G. Barthe, S. Cauligi, C. Chuengsatiansup, D. Genkin, S. O'Connell, P. Schwabe, R. Q. Sim, and Y. Yarom. Spectre declassified: Reading from the right place at the wrong time. *IEEE SP*. 2023.

[34] B. A. Shivakumar, G. Barthe, B. Grégoire, V. Laporte, T. Oliveira, S. Priya, P. Schwabe, and L. Tabary-Maujean. Typing high-speed cryptography against Spectre v1. *IEEE SP*. 2023.

[35] The Rocq Development Team. The Rocq prover, version 9.0. Zenodo.

[36] S. van der Wall and R. Meyer. SNIP: speculative execution and non-interference preservation for compiler transformations. *Proc. ACM Program. Lang.*, 9(POPL), 2025.

[37] S. Wiebing and C. Giuffrida. Training Solo: On the limitations of domain isolation against Spectre-v2 attacks. *IEEE SP*. 2025.

[38] S. Wiebing, A. de Faveri Tron, H. Bos, and C. Giuffrida. InSpectre Gadget: Inspecting the residual attack surface of cross-privilege Spectre v2. *USENIX Security*. 2024.

[39] J. Wikner, D. Trujillo, and K. Razavi. Phantom: Exploiting decoder-detectable mispredictions. *MICRO*. 2023.

[40] F. Willi. Identifying compiler optimizations that break constant time programming techniques. Master's thesis, ETH Zürich, 2025.

[41] X. Yang, Y. Chen, E. Eide, and J. Regehr. Finding and understanding bugs in C compilers. *PLDI 2011*. 2011.

[42] Y. Zakowski, C. Beck, I. Yoon, I. Zaichuk, V. Zaliva, and S. Zdancewic. Modular, compositional, and executable formal semantics for LLVM IR. *Proc. ACM Program. Lang.*, 5(ICFP), 2021.

[43] Z. Zhang and G. Barthe. CT-LLVM: Automatic large-scale constant-time analysis. Cryptology ePrint Archive, Paper 2025/338, 2025.

[44] Z. Zhang, G. Barthe, C. Chuengsatiansup, P. Schwabe, and Y. Yarom. Ultimate SLH: Taking speculative load hardening to the next level. *USENIX Security*. 2023.