KAWA: An Abstract Language for Scalable and Variable Detection of Spectre Vulnerabilities

Zheyuan Wu
Saarland University
Saarbrücken, Germany
zhwu00001@stud.uni-saarland.de

Haoyi Zeng
Saarland University
Saarbrücken, Germany
haze00001@stud.uni-saarland.de

Aaron Bies
Saarland University
Saarbrücken, Germany
bies@cs.uni-saarland.de

Abstract

Since the discovery of Spectre attacks, various detection methods for speculative vulnerabilities have been developed. Sound static analyses based on symbolic execution give precise results but lack scalability, while pattern-based analyses can accommodate large code bases but may be unsound and require manually crafted patterns for each microarchitecture.

We introduce KAWA, an abstract language designed to model control and data flows, allowing efficient analysis of Spectre vulnerabilities. KAWA's abstract nature also enables interpretation as schemata to capture entire classes of concrete programs with speculative leaks.

CCS Concepts: • **Security and privacy → Side-channel analysis and countermeasures**.

Keywords: symbolic execution, side-channel attacks, information flow, SMT solver

1 Introduction

Static analysis has the potential to play an important role in the task of securing computer systems. The difficulty of this task has fundamentally increased since the advent of Spectre attacks [1, 7] in 2018 and subsequent revelations [2, 13, 14].

Speculative execution is a hardware optimization that predicts a program's control flow and executes instructions speculatively. Spectre attacks exploit the fact that speculatively executed instructions may leave traces in the microarchitectural state, *e.g.*, in the cache state.

In the prototypical Spectre v1 example below, arbitrary out-of-bounds array accesses to array `B` can be triggered if the attacker controls the input `x` and trains the branch predictor to mispredict the outcome of the conditional statement. Once the processor determines the correct branch target, the accessed data is discarded. However, prior to that, the access to array `A` encodes the value of `B[x]` in the cache state. This allows the attacker to later extract the value of `B[x]` by measuring access latencies to the cache.

Example 1.1. The prototypical Spectre-v1 example [7]:

```
void victim_function_v1(int x) {
    if (x < size) {
        out = A[B[x] * 512];
    }
}
```

To address Spectre and related vulnerabilities, various countermeasures have been proposed, both at hardware [11, 12, 18–20] and at software level [4, 8–10, 15, 17, 21]. This work focuses on software-level defenses, where the main challenges are *precision*, *scalability*, and *variability*: *Precision* involves identifying vulnerabilities with few false positives, while *scalability* refers to the capability to handle large-scale programs effectively. The *variability challenge* arises from the large variety of speculative execution mechanisms, which result in a large spectrum of Spectre attacks across different microarchitectures.

Here, formal approaches based on symbolic execution such as Spectector [5] are precise and sound, but scale only to fairly small programs. This approach has later been extended to cover a variety of microarchitectural optimizations [3] addressing the variability challenge. Conversely, pattern-based methods such as oo7 [16] are highly scalable but may lack precision. These methods rely on manually crafted patterns that are not directly grounded in semantic notions of vulnerability, which may result in unsoundness and is also challenging to adapt to new hardware mechanisms.

The goal of this work is to develop a scalable and variable detection framework. To this end, we introduce KAWA, a simple abstract language for modelling the control and data flows in concrete programs. KAWA programs can be efficiently analyzed to detect Spectre vulnerabilities and can also be interpreted as schemata to capture entire classes of concrete programs that contain speculative leaks, elevating the analysis to a syntactic level. This dual application of KAWA not only enhances *scalability* but can also be adapted easily to various hardware configurations, thus addressing the *variability challenge*.

2 Our Approach: KAWA

We present KAWA, an abstract language for modelling information flow in programs. It features mutable variables and structures control flow using basic blocks. Thus, structurally, KAWA programs are similar to assembly programs and intermediate representations in compilers.

Crucially, as the language only serves to capture information flow, all concrete values and computation are abstracted away. Expressions, for example, only consist of two syntactic constructs, registers $\%x$ and n-ary uninterpreted functions $\psi(\dots)$ applied to expressions.

Example 2.1. The program from Ex. 1.1 can be represented in KAWA as follows:

```
start:
  %cond ← ψ(%x, %size)     // register assignment
  br %cond, then, end      // conditional branching
then:
  load %tmp, ψ(%A, %x)     // load from memory
  load %out, ψ(%B, %tmp)   // A[x] is leaked here
  br end                   // program termination
```

Here, the instruction `load <reg>, <addr>` reads the data at a given address into a specified register.

By design, KAWA programs can be seen as abstractions of programs in a concrete programming language. That is, for a given concrete language C, one can define a representation function $\beta : C \to \mathcal{K}$ that maps concrete programs to KAWA programs by abstracting away concrete operations, while keeping the control and data flow intact. We define the concretization function $\gamma : \mathcal{K} \to \wp(C)$ as follows:

$$\gamma(\pi) = \{\, p \in C \mid \beta(p) = \pi \,\}$$

Notably, this definition implies $p \in \gamma(\beta(p))$ for any concrete program $p \in C$.

2.1 Detecting Spectre Vulnerabilities

Inspired by Spectector [5], we have developed a method to execute KAWA programs symbolically to check whether they satisfy *speculative non-interference* [5], a semantic notion of security against speculative execution attacks, which identifies Spectre vulnerabilities by comparing leakage of a program *with* and *without* speculative execution.

For a well-chosen representation function β, one can show that if a KAWA program $\pi \in \mathcal{K}$ is secure, then all programs in $\gamma(\pi)$ are also secure from Spectre attacks. Inversely, if a concrete program $p \in C$ is vulnerable, then $\beta(p)$ is also deemed vulnerable. This means, searching for vulnerabilities in a program p by analyzing $\beta(p)$ may lead to false positives, but never to false negatives.

Complexity & Scalability. One of the major challenges with symbolic execution is scalability, as the SMT instances generated from the precise semantics of a programming language can quickly overwhelm the SMT solver. As concrete computations have been abstracted away in KAWA, this concern has been largely eliminated. One can show that all SMT instances we encounter are quantifier free, and only use a combination of the theory of equality and theory of arrays.

A Completeness Conjecture. As symbolic execution only considers a finite fragment of a program's behavior, it usually cannot prove safety properties of cyclic programs. Yet, since KAWA only models the information flow of a program, one can argue that all possible information flows of a finite program can be observed in a finite number of steps.

We conjecture that if a program with N instructions is vulnerable, our symbolic execution will find the vulnerability within N^2 steps[1], which implies that safety from Spectre attacks is decidable for KAWA. While a counterexample has eluded our randomized search, we emphasize that this statement is yet to be formally verified.

Support for Different Microarchitectures. To address the variability challenge, we designed and implemented the symbolic execution to adjust to different microarchitectures using *hardware-software contracts* [6]. From the software side, these contracts specify precisely what information the underlying hardware might leak and what kind of side channel attacks we need to detect and prevent in software.

2.2 KAWA as Schemata

As a result of KAWA's abstractness, KAWA programs can be interpreted as schemata, which can be matched syntactically in concrete programs. Given a concrete program $p \in C$ and a schema $\pi \in \mathcal{K}$, we say p *matches* π if $\beta(p)$ contains a subprogram that is isomorphic to π. Therefore, a given KAWA program, which has been determined to be vulnerable, can be used to identify parts of a larger concrete program that may share this vulnerability.

Example 2.2. The schema on the left was proven to be vulnerable, and the concrete program on the right matches the schema.

```
block0:
  load %x, ψ(%x)
  br %x, block0, end
```

```
start:
  load %a, 49 + %a
  %b ← 2 * %a
  br %a < 20, start, end
```

3 Ongoing Work

We are currently exploring further enhancements to the precision and automation of our approach.

3.1 Reducing False Positives

KAWA makes effective symbolic execution analysis feasible, yet applied naively, this approach leads to more false positives. While schemata matching offers higher scalability, it further decreases accuracy.

We are developing techniques to flexibly mix concrete and abstract code, which allows the analysis to quickly match a potentially vulnerable code fragment via schemata, and then confirm and localize vulnerabilities by semantic analysis.

3.2 Automated Schemata Mining

The analyzer discussed in Sec. 2.1 can be used to directly decide whether a given schema is vulnerable. Combining this analysis with a generator to enumerate KAWA up to a given size, we can automatically obtain a complete library of vulnerable KAWA programs under a given size bound.

[1] A tighter upper bound may be determined by inspecting the longest simple path in the dependency graph of a given KAWA program.

References

[1] 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)*. USENIX Association, Santa Clara, CA, 249–266. https://www.usenix.org/conference/usenixsecurity19/presentation/canella

[2] Claudio Canella, Daniel Genkin, Lukas Giner, Daniel Gruss, Moritz Lipp, Marina Minkin, Daniel Moghimi, Frank Piessens, Michael Schwarz, Berk Sunar, Jo Van Bulck, and Yuval Yarom. 2019. Fallout: Leaking Data on Meltdown-resistant CPUs. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019*, Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz (Eds.). ACM, 769–784. https://doi.org/10.1145/3319535.3363219

[3] Xaver Fabian, Marco Guarnieri, and Marco Patrignani. 2022. Automatic Detection of Speculative Execution Combinations. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS 2022, Los Angeles, CA, USA, November 7-11, 2022*, Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi (Eds.). ACM, 965–978. https://doi.org/10.1145/3548606.3560555

[4] Roberto Guanciale, Musard Balliu, and Mads Dam. 2020. InSpectre: Breaking and Fixing Microarchitectural Vulnerabilities by Formal Analysis. In *CCS '20: 2020 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, USA, November 9-13, 2020*, Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna (Eds.). ACM, 1853–1869. https://doi.org/10.1145/3372297.3417246

[5] Marco Guarnieri, Boris Köpf, José F. Morales, Jan Reineke, and Andrés Sánchez. 2020. Spectector: Principled Detection of Speculative Information Flows. In *2020 IEEE Symposium on Security and Privacy, SP 2020, San Francisco, CA, USA, May 18-21, 2020*. IEEE, 1–19. https://doi.org/10.1109/SP40000.2020.00011

[6] Marco Guarnieri, Boris Köpf, Jan Reineke, and Pepe Vila. 2021. Hardware-Software Contracts for Secure Speculation. In *42nd IEEE Symposium on Security and Privacy, SP 2021, San Francisco, CA, USA, 24-27 May 2021*. IEEE, 1868–1883. https://doi.org/10.1109/SP40001.2021.00036

[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. *meltdownattack.com* (2018). https://spectreattack.com/spectre.pdf

[8] Cong Ma, Dinghao Wu, Gang Tan, Mahmut Taylan Kandemir, and Danfeng Zhang. 2023. Quantifying and Mitigating Cache Side Channel Leakage with Differential Set. *Proc. ACM Program. Lang.* 7, OOPSLA2, Article 274 (oct 2023), 29 pages. https://doi.org/10.1145/3622850

[9] Shravan Narayan, Craig Disselkoen, Daniel Moghimi, Sunjay Cauligi, Evan Johnson, Zhao Gang, Anjo Vahldiek-Oberwagner, Ravi Sahita, Hovav Shacham, Dean M. Tullsen, and Deian Stefan. 2021. Swivel: Hardening WebAssembly against Spectre. In *30th USENIX Security Symposium, USENIX Security 2021, August 11-13, 2021*, Michael D. Bailey and Rachel Greenstadt (Eds.). USENIX Association, 1433–1450. https://www.usenix.org/conference/usenixsecurity21/presentation/narayan

[10] Oleksii Oleksenko, Bohdan Trach, Mark Silberstein, and Christof Fetzer. 2020. SpecFuzz: Bringing Spectre-type vulnerabilities to the surface. In *29th USENIX Security Symposium, USENIX Security 2020, August 12-14, 2020*, Srdjan Capkun and Franziska Roesner (Eds.). USENIX Association, 1481–1498. https://www.usenix.org/conference/usenixsecurity20/presentation/oleksenko

[11] Gururaj Saileshwar and Moinuddin K. Qureshi. 2019. CleanupSpec: An "Undo" Approach to Safe Speculation. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture* (Columbus, OH, USA) *(MICRO '52)*. Association for Computing Machinery, New York, NY, USA, 73–86. https://doi.org/10.1145/3352460.3358314

[12] Christos Sakalis, Stefanos Kaxiras, Alberto Ros, Alexandra Jimborean, and Magnus Själander. 2019. Efficient invisible speculative execution through selective delay and value prediction. In *Proceedings of the 46th International Symposium on Computer Architecture* (Phoenix, Arizona) *(ISCA '19)*. Association for Computing Machinery, New York, NY, USA, 723–735. https://doi.org/10.1145/3307650.3322216

[13] Michael Schwarz, Moritz Lipp, Daniel Moghimi, Jo Van Bulck, Julian Stecklina, Thomas Prescher, and Daniel Gruss. 2019. ZombieLoad: Cross-Privilege-Boundary Data Sampling. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019*, Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz (Eds.). ACM, 753–768. https://doi.org/10.1145/3319535.3354252

[14] Michael Schwarz, Moritz Lipp, Daniel Moghimi, Jo Van Bulck, Julian Stecklina, Thomas Prescher, and Daniel Gruss. 2019. ZombieLoad: Cross-Privilege-Boundary Data Sampling. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019*, Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz (Eds.). ACM, 753–768. https://doi.org/10.1145/3319535.3354252

[15] 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.* 5, POPL, Article 49 (jan 2021), 30 pages. https://doi.org/10.1145/3434330

[16] Guanhua Wang, Sudipta Chattopadhyay, Ivan Gotovchits, Tulika Mitra, and Abhik Roychoudhury. 2021. oo7: Low-Overhead Defense Against Spectre Attacks via Program Analysis. *IEEE Trans. Software Eng.* 47, 11 (2021), 2504–2519. https://doi.org/10.1109/TSE.2019.2953709

[17] Jingbo Wang, Chungha Sung, Mukund Raghothaman, and Chao Wang. 2021. Data-Driven Synthesis of Provably Sound Side Channel Analyses. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. 810–822. https://doi.org/10.1109/ICSE43902.2021.00079

[18] Zilong Wang, Gideon Mohr, Klaus von Gleissenthall, Jan Reineke, and Marco Guarnieri. 2023. Specification and Verification of Side-channel Security for Open-source Processors via Leakage Contracts. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security, CCS 2023, Copenhagen, Denmark, November 26-30, 2023*, Weizhi Meng, Christian Damsgaard Jensen, Cas Cremers, and Engin Kirda (Eds.). ACM, 2128–2142. https://doi.org/10.1145/3576915.3623192

[19] 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* (Columbus, OH, USA) *(MICRO '52)*. Association for Computing Machinery, New York, NY, USA, 572–586. https://doi.org/10.1145/3352460.3358306

[20] Yuheng Yang, Thomas Bourgeat, Stella Lau, and Mengjia Yan. 2023. Pensieve: Microarchitectural Modeling for Security Evaluation. In *Proceedings of the 50th Annual International Symposium on Computer Architecture, ISCA 2023, Orlando, FL, USA, June 17-21, 2023*, Yan Solihin and Mark A. Heinrich (Eds.). ACM, 59:1–59:15. https://doi.org/10.1145/3579371.3589094

[21] Zhiyuan Zhang, Gilles Barthe, Chitchanok Chuengsatiansup, Peter Schwabe, and Yuval Yarom. 2023. Ultimate SLH: Taking Speculative Load Hardening to the Next Level. In *32nd USENIX Security Symposium, USENIX Security 2023, Anaheim, CA, USA, August 9-11, 2023*, Joseph A. Calandrino and Carmela Troncoso (Eds.). USENIX Association, 7125–7142. https://www.usenix.org/conference/usenixsecurity23/presentation/zhang-zhiyuan-slh