Sébastien Michelland

Ph.D in computer science

sebastien.michelland@inria.fr • Nov. 1999
Personal website: https://silent-tower.net

Positions and education

Inria center at the University of Rennes — Post-doc Rennes, France • *Nov. 2025–ongoing*

Laboratoire de Conception et d'Intégration des Systèmes — Ph.D Valence, France • *2022–Oct. 2025*
Thesis titled "Compilation beyond semantics for hardware security".

École Normale Supérieure de Lyon — Computer Science MSc Lyon, France • *2017–2022*
Graduated BSc in 2018 and MSc in 2020 (both *summa cum laude*). The curriculum covers general computer science theory; I focused on languages, semantics and compilers. This includes internship projects:

- **University of Edinburgh** — Composable formal semantics for MLIR Edinburgh, U.K. • *2022*
  I outlined a modular semantics for MLIR, a modular framework for creating LLVM-like compiler IRs, based on monadic interpreters, up to basic reasoning on peephole rewrites.
- **Verimag** — Extensions of the congruence closure algorithm Grenoble, France • *2020*
  I developed extensions to the congruence closure algorithm and implemented the resulting decision procedure in OCaml, to be used in an updated version of the Coq tactic congruence [5, 4].
- **UQÀM** — Study of interactions between LLVM passes Montreal, Canada • *Summer 2019*
  I collected a database of LLVM optimization passes and dependencies, and sketched software engineering tools that study how they interact to guide empirical phase ordering [6].
- **Inria** — Coq formalization of the dancing links algorithm Paris, France • *Summer 2018*
  I proved the dancing links algorithm in Coq and partially proved an OCaml implementation [7].

Research projects

Most of my research work fits in the general category of program specification and compilation, which I like to approach across abstraction boundaries. This has led me to work both at the lowest levels of software abstraction, with my Ph.D relating to security against hardware vulnerabilities; and at the highest levels, with my second biggest project being certifying abstract interpreters in the context of monadic semantics.

Ph.D: Compilation beyond semantics for hardware security (2022–ongoing).

My Ph.D revolves around methodological problems with countermeasures for fault injection and side-channel attacks. These attacks that target hardware have complex impacts on software, the characterization of which *(fault and side-channel modeling)* is an art in and of itself. Crucially, these effects cannot be expressed accurately across abstraction boundaries, meaning that

1. countermeasures should ideally target the accurate lower-level models;
2. end-user security properties stemming from source code and countermeasures applied at various compilation stages must be connected to and preserved until these low levels are reached.

Neither of these is quite easy. Lower-level fault models are typically analyzed empirically as formal verification efforts usually start at the more amenable high-level languages (and these communities lack overlap). Thus it's not obvious whether one can capture architectural details in the study of a protected program. I showed in one paper [2] that it was possible to semantically model a RISC-V fetch-stage skip attack, which enabled the design of a fine software/hardware countermeasure whose validity is proven before it is tested.

This paper is significant in establishing a connection between two kinds of communities and methodologies: compilers with semantics and formal reasoning on one hand, and hardware security with accurate fault models and micro-architecture awareness on the other.

The compilation step is equally hairy; in order to allow the production of secure target code, a compiler needs to either take secure code as input or insert the security countermeasure itself; and then, it needs to preserve whatever desirable properties the "secure" code has until the target language is reached. As security properties cannot be expressed uniformly across abstraction boundaries, this requires modeling a *series* of security properties at each intermediate level, something that I argue for in my Ph.D but is otherwise almost never considered.

The main tool out of this project is *Tracing LLVM*, a lightweight LLVM mod that provides concrete tools for secure compilation[1]. Its recurring theme is to allow specific elements of the source program (variable accesses, computations, data-flow...) to be *traced* as they get lowered, while guaranteeing their integrity from compiler interference. This defines a perimeter in which a security engineer can assert manual control in the compilation, facilitating the implementation of security countermeasures.

Abstract semantics for monadic interpreters (2021–2024).

I designed and implemented in Coq/Rocq a basic framework for writing and proving abstract interpreters for simple languages in the context of the Vellvm project [8]. Vellvm provides denotational semantics for LLVM through a monadic model known as *interaction trees*, which happens to be executable. This opens the door to defining *abstract semantics* whose execution constitutes an abstract interpreter.

I showed that it was possible to certify an abstract interpreter defined through this process, which involves a series of so-called *monadic interpreters*. This means that one can define concrete and abstract semantics together in a mirrored fashion and derive a large part of the abstract interpreter's proof of correctness. The open-source implementation of this framework features a number of reusable analysis components that can be repurposed across languages[2].

This work started as an internship but was continued for about two years then published to ICFP'24 [1].

Other research projects.

In a 2018 COMPAS paper [3] derived from an architecture class, we experimented with an ISA design that keeps memory bandwidth to a minimum. I wrote an interactive emulator/debugger[3] for the fictional CPU, which was later used in multiple iterations of a compilers class at UCBL.

Tools and engineering work

Unikernel development on embedded calculators (2015–ongoing).

Since 2015 I've developed and maintained a unikernel and SDK for native programming across a dozen or so graphing calculator models, contributing to significant experience with low-level development. These machines already have an OS, so my kernel programmatically takes and yields control of hardware from it to go bare-metal in a hypervisor-like context switch. Features grew over time to include module drivers, memory management, interrupt-based async. I/O, a USB 2.0 driver, remote debugging, and a custom C99 libc among others. This project involved a hefty amount of reverse-engineering from flat OS binaries to reconstruct closed documentation. More details can be found online[4].

[1] https://gricad-gitlab.univ-grenoble-alpes.fr/tracing-llvm
[2] https://gitlab.inria.fr/sebmiche/itree-ai
[3] https://github.com/lephe/memory-light-isa
[4] https://silent-tower.net/projects/gint

Teaching

As a Ph.D student, I taught ~180 hours at Grenoble INP–Esisar and created quite a bit of material:

Algorithms and C programming in two different classes (CS221–L2, IN330–L3).

- Including 3 class sessions that I taught with original material mostly on memory management in C.
- I also built a RISC-V emulator project, complete with guiding exercises, hidden tests, grading processes...

Functional programming in Haskell (CS222–L2).

- There I heavily reworked all 7 TP subjects and contributed a robust semi-automatic test system.

Programming languages and compilers (CS444–M1).

Analysis of software and hardware security in operating systems (OS430–M1).

Qualifications/Skills

French: Native speaker
English: CEFR level C2 — *CAE 203/210 (2018), TOEFL 116/120 (2021)*
C and embedded development: Advanced — *User/kernel land, LLVM internals, Linking, Basic RE*
Functional programming and type theory: Very comfortable — *Coq, Lean 4, Haskell*
Also experienced with: C++, Linux administration, LaTeX

Bibliography

Journal articles

[1] Sébastien **Michelland**, Yannick Zakowski, and Laure Gonnord. "Abstract Interpreters: A Monadic Approach to Modular Verification". In: *Proceedings of the ACM on Programming Languages* 8.ICFP (Aug. 2024), pp. 1–28. URL: https://hal.science/hal-04628727.

Conference articles

[2] Sébastien **Michelland**, Christophe Deleuze, and Laure Gonnord. "From low-level fault modeling (of a pipeline attack) to a proven hardening scheme". In: *Compiler Construction*. Edinburgh (Scotland), United Kingdom, Mar. 2024. URL: https://hal.science/hal-04438994.

[3] Florent de Dinechin, Maxime Darrin, Antonin Dudermel, Sébastien **Michelland**, and Alban Reynaud. "Une architecture minimisant les échanges entre processeur et mémoire". In: *ComPAS 2018 - Conférence d'informatique en Parallélisme, Architecture et Système*. Toulouse, France, July 2018, pp. 1–8. URL: https://inria.hal.science/hal-01959855.

Workshops, reports, etc

[4] Sébastien **Michelland**. *Rapport de stage : Une procédure de décision pour relations d'équivalence*. https://silent-tower.net/static/internship_congruence_closure.pdf. June 2020.

[5] Sébastien **Michelland**, Pierre Corbineau, Lionel Rieg, and Karine Altisen. "A Decision Procedure for Equivalence Relations". In: *Coq Workshop 2020*. Aubervilliers, France, July 2020. URL: https://hal.science/hal-04880486.

[6] Sébastien **Michelland**. *Rapport de stage : Exploration et cartographie des passes de LLVM*. https://silent-tower.net/static/internship_llvm_passes.pdf. July 2019.

[7] Sébastien **Michelland**. *Rapport de stage : Permutations et liens dansants vérifiés en CFML*. https://silent-tower.net/static/internship_dancing_links.pdf. July 2018.

Other publications cited for context

[8] Yannick Zakowski, Calvin Beck, Irene Yoon, Ilia Zaichuk, Vadim Zaliva, and Steve Zdancewic. "Modular, Compositional, and Executable Formal Semantics for LLVM IR". In: *Proc. ACM Program. Lang.* 5.ICFP (Aug. 2021). URL: https://doi.org/10.1145/3473572.

Built on November 6, 2025