Building and Verifying a Simple Secure Compiler

General Information

Advisor: Cătălin Hriţcu ⟨catalin.hritcu@inria.fr⟩

Institition: Inria Paris, Prosecco Team

Location: 2 rue Simone Iff, 75012 Paris, France

Language: English

Keywords:

- state-of-the-art research in programming languages;
- security foundations and/or building secure systems;
- secure compilation, full abstraction, components;
- dynamic enforcement, tag-based monitoring;
- formal verification in the Coq proof assistant;
- property-based testing (e.g., with QuickChick).

Context

Severe low-level vulnerabilities abound in today's computer systems, allowing cyber-attackers to remotely gain full control. This happens in big part because our programming languages, compilers, and architectures were designed in an era of scarce hardware resources and too often trade off security for efficiency. The semantics of mainstream low-level languages like C is inherently insecure, and even for safer languages, establishing security with respect to a high-level semantics does not guarantee the absence of low-level attacks. *Secure compilation* using the coarse-grained protection mechanisms provided by mainstream hardware architectures would be too inefficient for most practical scenarios.

SECOMP[1] is a new ERC-funded project aimed at leveraging emerging hardware capabilities for fine-grained protection to build the first, efficient secure compilers for realistic low-level programming languages (the C language, and Low* [5] a safe subset of C embedded in F* [1] for verification). These compilers will provide a more secure semantics for source programs and will ensure that high-level abstractions cannot be violated even when interacting with untrusted low-level code. To achieve this level of security without sacrificing efficiency, our secure compilers target a *tagged architecture* [2, 3], which associates a metadata tag to each word and efficiently propagates and checks tags according to software-defined rules. We are using property-based testing and formal verification to provide high confidence that our compilers are indeed secure. Formally, we are constructing machine-checked proofs in Coq of fully abstract compilation and of a new property we call robust compilation, which implies the preservation of trace properties even against an adversarial context [4]. These strong properties complement compiler correctness and ensure that no machine-code attacker can do more harm to securely compiled components than a component already could with respect to a secure source-level semantics.

[1] `http://secure-compilation.github.io/`

Objective of the internship

We are looking for highly motivated MSc students interested in getting an initial contact with research. Successful internships with us normally result in a research publication at a good international conference, and most often the MSc students we advise continue with a PhD.

The objective of this internship is to build a simple secure compiler and to prove its security formally. This compiler will translate programs from a core imperative language with procedures and components to an idealized RISC machine, so that each component is protected from the others. Formally, we will investigate two properties capturing secure compilation: The first one is a variant of *full abstraction* that ensures the preservation of confidentiality and integrity properties from the source to the target. The second one, is a new property we call *robust compilation* that captures only the preservation of trace properties (and thus certain forms of integrity, for instance data invariants) in an adversarial low-level context. While weaker than full abstraction, robust compilation is easier to enforce efficiently and also is unaffected by side-channel attacks. We will explore different ways for proving secure compilation and we will build machine-checked proofs in Coq. For details please get in contact using the information above or check out `https://secure-compilation.github.io/`

References

[1] D. Ahman, C. Hriţcu, K. Maillard, G. Martínez, G. Plotkin, J. Protzenko, A. Rastogi, and N. Swamy. Dijkstra monads for free. *POPL*. 2017.

[2] A. Azevedo de Amorim, M. Dénès, N. Giannarakis, C. Hriţcu, B. C. Pierce, A. Spector-Zabusky, and A. Tolmach. Micro-Policies: Formally verified, tag-based security monitors. *Oakland S&P*. 2015.

[3] U. Dhawan, C. Hriţcu, R. Rubin, N. Vasilakis, S. Chiricescu, J. M. Smith, T. F. Knight, Jr., B. C. Pierce, and A. DeHon. Architectural support for software-defined metadata processing. *ASPLOS*. 2015.

[4] Y. Juglaret, C. Hriţcu, A. Azevedo de Amorim, B. Eng, and B. C. Pierce. Beyond good and evil: Formalizing the security guarantees of compartmentalizing compilation. *CSF*. 2016.

[5] J. Protzenko, J.-K. Zinzindohoué, A. Rastogi, T. Ramananandro, P. Wang, S. Zanella-Béguelin, A. Delignat-Lavaud, C. Hriţcu, K. Bhargavan, C. Fournet, and N. Swamy. Verified low-level programming embedded in F*. *ICFP*, 2017. To appear.