Title:
Formally Secure Compilation

Abstract:

Severe low-level vulnerabilities abound in today's computer systems, allowing cyber-attackers to remotely gain full control. The semantics of mainstream low-level languages like C provide no security against devastating vulnerabilities like buffer overflows and control-flow hijacking. Even for safer languages, establishing security with respect to the language's semantics does not prevent low-level attacks. All the abstraction and security guarantees of the source language are currently lost when interacting with low-level code, e.g., when using libraries.

Secure compilation is an emerging field that puts together advances in security, programming languages, compilers, verification, systems, and hardware architectures in order to devise secure compilation chains that eliminate many of today's low-level vulnerabilities. Secure compilation aims to protect high-level language abstractions in compiled code, even against adversarial low-level contexts, and to allow sound reasoning about security in the source language.

However, what does it mean that a compiler chain is secure? How does one define such secure compilation formally? And to what attacker model does it correspond? In this talk I will argue that a secure compilation chain should preserve some well-specified class of security properties of source programs even against adversarial low-level contexts. Particularly interesting classes include safety properties, hyperproperties (e.g. non-interference), and relational hyperproperties (e.g. observational equivalence).

Such strong secure compilation criteria complement compiler correctness and ensure that no adversarial low-level context can do more harm to a securely compiled program than a source-level context already could with respect to a *safe* source language semantics. We further extend these criteria to also provide end-to-end security guarantees even for software components written in *unsafe*, low-level languages with C-style undefined behavior. These extended criteria are the first to model dynamic compromise in a system of mutually distrustful components running with clearly specified privileges. Each component is protected from all the others---in particular, from components that have encountered undefined behavior and become compromised. Each component receives secure compilation guarantees up to the point when it becomes compromised, after which an attacker can take complete control over the component and use any of its privileges to attack the remaining uncompromised components.

To illustrate this model, we build a secure compilation chain for an unsafe language with buffers, procedures, and components. We compile it to a simple RISC abstract machine with built-in compartmentalization and provide thorough proofs, many of them machine-checked in Coq, showing that the compiler satisfies our secure compilation criterion. Finally, we show that the protection guarantees offered by the compartmentalized abstract machine can be achieved at the machine-code level using either software fault isolation or a tag-based reference monitor.

Bio: Catalin is a researcher at Inria Paris where he develops rigorous formal techniques for solving security problems. He is particularly interested in formal methods for security (secure compilation, memory safety, compartmentalization, dynamic monitoring, integrity, security protocols, information flow), programming languages (type systems, verification, proof assistants, property-based testing, semantics, formal metatheory, certified tools), and the design and verification of security-critical systems (reference monitors, secure compilation chains, secure hardware). He is actively involved in the design of the F* verification system and was recently awarded an ERC Starting Grant on secure compilation. Catalin was also a PhD student at Saarland University, a Research Associate at University of Pennsylvania, and a visiting researcher at Microsoft Research Redmond.