Abstract

Scalable Proof Techniques for Formally Secure Compilation of Compartmentalized Programs

Jérémy Thibault

Current software systems are distressingly insecure, and despite constant attempts at patching systems and implementing protection mechanisms, many programs, especially those written in fundamentally unsafe languages such as C, admit vulnerabilities that are sometimes exploited to devastating effect.

A typical mitigation is compartmentalization: software system designers split their applications in several compartments that can only interact via trusted interfaces, in the hope a vulnerability in one compartment doesn't jeopardize the security of the entire application. Unfortunately, compartmentalization mechanisms are still fallible, and and regularly broken, allowing an attacker to exploit vulnerabilities to attack other compartments. Indeed, these mechanisms are rarely designed with formal guarantees in mind, and even when they are, the sheer complexity of the implementation makes it difficult to guarantee security bugs do not slip through.

In this thesis, we work towards improving the situation, from the perspective of secure compilation. Secure compilation is a research area that aims at giving programmers tools to reason about the security of their programs in their source languages, putting together techniques from programming languages, compiler design, systems, and security in order to build secure compilation chains. Often, in the formal security community, designers of simple secure compilation chains try to prove that they satisfy formal guarantees; however, these proofs are often extremely lengthy and complex, even for these very simple compilation chains. Given the very high stakes for security and the challenge of scaling up formally secure compilation to realistic compilers, secure compilation proofs are the prime target for mechanization in a proof assistant, yet such mechanizations are still few and far between.

We study the formal verification of secure compilers, and in the process, develop new proof techniques that make it easier to formally verify a secure compiler. In particular, we improve upon an existing proof technique by introducing *recomposition*, a way to avoid reasoning about partial semantics, which are known to be tricky to get right and lead to difficult proofs.

Following this, we build SECOMP, a secure compiler for a compartmentalized variant of C, as an extension of the CompCert compiler. To do so, we extend all of CompCert's languages with a notion of compartments that can only interact with each other in protected ways, and we adapt the compilation passes of CompCert to this setting. We prove not only the correctness of SECOMP, but also that it is secure, in the sense that undefined behaviors are restricted to the compartment that triggers them. In the process of proving security, we introduce several proof engineering novelties that were necessary to scale up to C. SECOMP is the first formally verified secure compiler for a realistic programming language with compartments.

We conclude this thesis by presenting two interesting extensions of the proof technique we use: one that allows handling memory sharing between compartments, a common programming pattern that is not yet supported by SECOMP; and a back-translation technique that allows one to prove a much stronger criterion with little extra effort.

Kurzfassung

Skalierbare Beweistechniken für die formal sichere Kompilierung von kompartimentierten Programmen

Jérémy Thibault

Note This is a machine translation of the abstract. I do not speak German, so I cannot guarantee that it's accurate. Please refer to the English version of the abstract.

Aktuelle Softwaresysteme sind erschreckend unsicher. Trotz ständiger Versuche, Systeme zu patchen und Schutzmechanismen zu implementieren, weisen viele Programme, insbesondere solche, die in grundsätzlich unsicheren Sprachen wie C geschrieben sind, Schwachstellen auf, die mitunter mit verheerender Wirkung ausgenutzt werden.

Eine typische Abschwächungsmaßnahme ist die Kompartimentierung: Softwareentwickler teilen ihre Anwendungen in mehrere Bereiche auf, die nur über vertrauenswürdige Schnittstellen interagieren können, in der Hoffnung, dass eine Schwachstelle in einem Bereich die Sicherheit der gesamten Anwendung nicht gefährdet.

Leider sind Kompartimentierungsmechanismen immer noch fehleranfällig und werden regelmäßig beschädigt, sodass Angreifer Schwachstellen ausnutzen und andere Bereiche angreifen können.

Tatsächlich werden diese Mechanismen selten mit formalen Garantien entwickelt, und selbst wenn dies der Fall ist, erschwert die Komplexität der Implementierung die Gewährleistung, dass keine Sicherheitslücken durchrutschen.

In dieser Arbeit arbeiten wir daran, die Situation aus der Perspektive der sicheren Kompilierung zu verbessern. Sichere Kompilierung ist ein Forschungsgebiet, das Programmierern Werkzeuge an die Hand gibt, um die Sicherheit ihrer Programme in ihren Quellsprachen zu beurteilen. Dazu werden Techniken aus Programmiersprachen, Compilerdesign, Systemen und Sicherheit kombiniert, um sichere Kompilierungsketten zu erstellen.

In der formalen Sicherheitscommunity versuchen Entwickler einfacher sicherer Kompilierungsketten oft nachzuweisen, dass sie formale Garantien erfüllen. Diese Beweise sind jedoch oft extrem langwierig und komplex, selbst für diese sehr einfachen Kompilierungsketten.

Angesichts der hohen Sicherheitsanforderungen und der Herausforderung, formal sichere Kompilierung auf realistische Compiler zu skalieren, sind sichere Kompilierungsbeweise das wichtigste Ziel für die Mechanisierung in einem Beweisassistenten. Solche Mechanisierungen sind jedoch noch selten.

Wir untersuchen die formale Verifikation sicherer Compiler und entwickeln dabei neue Beweistechniken, die die formale Verifikation sicherer Compiler erleichtern. Insbesondere verbessern wir eine bestehende Beweistechnik durch die Einführung von *Rekomposition*. Diese Methode vermeidet das Denken über partielle Semantik, die bekanntermaßen schwierig zu verstehen ist und zu schwierigen Beweisen führt.

Anschließend entwickeln wir SECOMP, einen sicheren Compiler für eine kompartimentierte Variante von C, als Erweiterung des CompCert-Compilers. Dazu erweitern wir alle CompCert-Sprachen um das Konzept von Kompartimenten, die nur geschützt miteinander interagieren können, und passen die Kompilierdurchläufe von CompCert an diese Einstellung an. Wir beweisen nicht nur die Korrektheit von SECOMP, sondern auch dessen Sicherheit, da undefiniertes Verhalten auf das auslösende Kompartiment beschränkt ist.

Im Zuge des Sicherheitsnachweises führen wir mehrere Neuerungen im Proof-Engineering ein, die für die Skalierung auf C notwendig waren. SECOMP ist der erste formal verifizierte sichere Compiler für eine realistische Programmiersprache mit Kompartimenten.

Wir schließen diese Arbeit mit der Präsentation zweier interessanter Erweiterungen der von uns verwendeten Beweistechnik ab: Eine ermöglicht die gemeinsame Nutzung von Speicher zwischen Kompartimenten, ein gängiges Programmiermuster, das von SECOMP noch nicht unterstützt wird, und eine Rückübersetzungstechnik, mit der sich mit geringem Mehraufwand ein deutlich stärkeres Kriterium beweisen lässt.