Flexible Scheduler-Independent Security

Heiko Mantel and Henning Sudbrock

Computer Science, TU Darmstadt, Germany
{mantel,sudbrock}@cs.tu-darmstadt.de

Abstract. We propose an approach to certify the information flow security of multi-threaded programs independently from the scheduling algorithm. A scheduler-independent verification is desirable because the scheduler is part of the runtime environment and, hence, usually not known when a program is analyzed. Unlike for other system properties, it is not straightforward to achieve scheduler independence when verifying information flow security, and the existing independence results are very restrictive. In this article, we show how some of these restrictions can be overcome. The key insight in our development of a novel scheduler-independent information flow property was the identification of a suitable class of schedulers that covers the most relevant schedulers. The contributions of this article include a novel security property, a scheduler independence result, and a provably sound program analysis.

1 Introduction

Whether a program can be entrusted secrets depends on the flow of information caused by running this program. Noninterference is a security property that characterizes secure information flow by the requirement that a program’s output to untrusted sinks does not depend on secrets [1]. This requirement ensures that an attacker cannot conclude any information about secrets from the output that he can possibly observe, even if he has access to the full code of the program.

In order to obtain reliable analysis results, a noninterference analysis needs to properly respect the semantics of the given language. This raises the question of how to deal with aspects that influence a program’s behavior, but that are outside the definition of the programming language’s semantics. Examples are elements of the language whose behavior is not specified in the language’s definition (e.g. native methods in Java) or elements of the runtime environment.

In this article, we focus on how to deal with a particular element of the runtime environment, namely the scheduler. Unlike for other properties, it is not sufficient to assume a possibilistic scheduler in a noninterference analysis, i.e. the scheduler that admits all possible scheduling choices. Secure information flow under a possibilistic scheduler need not imply that a program is secure for other schedulers because refining some part of a secure system’s specification (such as the scheduler) may result in a system that violates security [2].

Many information flow analyses are scheduler dependent in the sense that they assume a particular scheduler, such that the analysis results are only valid if the
program is executed under this scheduler. For instance, a uniform scheduler is assumed in [3], a Round-Robin scheduler in [4], and a possibilistic scheduler in [5].

There are also a few approaches that support a scheduler-independent analysis. So far, there are two main approaches to defining information flow properties that are scheduler independent, firstly, requiring that a program’s public output is deterministically determined by the program’s public input and, secondly, requiring that a program’s possible behaviors for any two inputs, which comprise identical public inputs, are stepwise indistinguishable to an observer of the program’s public outputs. The first approach was introduced by Zdancewic and Myers, adapting the idea of defining secure information flow based on observational determinism to language-based security [6]. The second approach was used by Sabelfeld and Sands to define the so-called strong security property [7]. While both approaches provide a semantic basis for program analyses that are sound independently of the scheduling algorithm, they are far from satisfactory. The resulting security properties are very restrictive because they are violated by many intuitively secure programs. The main deficiency of security properties based on observational determinism is that they forbid nondeterminism in the publicly observable behavior of a program, albeit intuitively secure programs can have nondeterministic public output. Strong security suffers from a different problem. It requires a restrictive lock-step indistinguishability, which implies, for instance, that a program’s execution time must not depend on secrets, even if such differences in the timing do not cause differences in the public output.

In this article, we propose a scheduler-independent security property that permits nondeterminism in a program’s publicly observable behavior without requiring a restrictive lock-step indistinguishability. Our solution does not require non-standard modifications to the interface of schedulers (as in other approaches, e.g., [8,9]). In fact our approach is the first that is suitable for programs with nondeterministic publicly observable behavior whose runtime depends on secrets, while providing scheduler independence for common schedulers like Round-Robin and uniform schedulers (see Section 6 for a more detailed comparison). The existence of a scheduler-independent security property with these features is somewhat surprising given that Sabelfeld proved in [10] that strong security is the weakest property that implies information flow security for a natural class of schedulers. The key step in our development was the identification of a different class of schedulers, the robust schedulers, that also contains the most relevant schedulers.

In summary, our contributions include (1) the definition of a novel security property for multi-threaded programs, (2) the novel notion of robust schedulers, (3) a theorem showing that our security property is scheduler independent for robust schedulers, and (4) a provably sound, scheduler-independent program analysis for enforcing our security property. We illustrate the progress made by the security analysis of a small, but realistic example program. The proofs of all theorems in this article are made available on the authors’ website. We expect that our improvements constitute a significant step towards more widely applicable information flow analyses for concurrent programs.