Computer Languages, Systems & Structures 50 (2017) 82–107

Contents lists available at ScienceDirect

Computer Languages, Systems & Structures

journal homepage: www.elsevier.com/locate/cl

Atomistic Galois insertions for flow sensitive integrity

Flemming Nielson*, Hanne Riis Nielson

DTU Compute, Kongens Lyngby, Denmark

ARTICLE INFO

Article history:
Received 12 January 2017
Revised 8 May 2017
Accepted 13 June 2017
Available online 23 June 2017

Keywords:
Abstract interpretation
Security policy
Information flow

ABSTRACT

Several program verification techniques assist in showing that software adheres to the required security policies. Such policies may be sensitive to the flow of execution and the verification may be supported by combinations of type systems and Hoare logics. However, this requires user assistance and to obtain full automation we shall explore the over-approximating nature of static analysis.

We demonstrate that the use of atomistic Galois insertions constitutes a stable framework in which to obtain sound and fully automatic enforcement of flow sensitive integrity. The framework is illustrated on a concurrent language with local storage and polyadic synchronous communication.

1. Introduction

Trustworthy software requires adherence to advanced security policies. Policies are often formulated in terms of *principals* (or users) and the *variables* (or data) that they access; accesses are often divided into *uses* (or reads) and *influences* (or writes) [19].

In *access control* an access is either granted or denied. Restrictions may be expressed in terms of the *names* of the principals and variables or in terms of the *security classifications* of the principals and variables [6,8].

In *information flow* [11,17,18,22,28,43] it is further required that even if the use of a variable is granted it may not later be used in a way that violates the policy; similarly, even if the influence of a variable is granted this may not depend on previous data that violates the policy. Enforcement of information flow policies often takes the form of type systems [40], and a well-known example is that of the Decentralised Label Model [28,29].

Sometimes the applicable policy is sensitive to the *flow* of execution: for influences we are concerned about how we reached the point of interest, and for uses we are concerned about how we continue the execution. To express these *flow sensitive* policies requires the ability to express conditions on the current values of variables. Indeed, the need for content dependent security policies is argued in [27] to be a main challenge in the context of ensuring the security of avionics software [13,26,35].

To check that software adheres to flow sensitive security policies requires methods and techniques from program verification. Generally one would need to characterise values of variables at various program points in order to determine adherence to the policy. Program logics such as *Hoare Logic* [1,3,4,21,37,41] are fully capable of this even in the presence of concurrent and communicating programs. It is also possible to extend program logics to take security policies into account and to express that programs conform to the policies [32,33].

* Corresponding author.
E-mail addresses: fnie@dtu.dk (F. Nielson), hrni@dtu.dk (H. Riis Nielson).

http://dx.doi.org/10.1016/j.cl.2017.06.004
1477-8424/

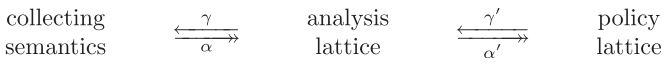

Fig. 1. Atomistic Galois insertions in our development.

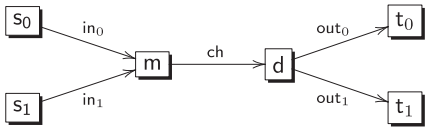

Fig. 2. The processes and channels of the multiplexer example.

However, it is likely that software development already employs powerful static analyses [31] for ensuring the functionality of the code. A substantial use of abstract interpretation for avionics software is presented in [7] and an example of the use of abstract interpretation for large scale analysis of mobile code is given in [42]. Static analysis computes sound approximations to the values that variables may have at various program points. This is often expressed as over-approximating fixed-points in a complete lattice of properties of states and many techniques exist for developing sound and reasonably precise analyses with a tractable complexity [14–16], including the use of widening.

What is needed is to find a way to express the security policies of interest so that they exploit the static analyses. In [25] it is shown that abstract interpretation based approaches are valuable also when dealing with declassification according to a number of dimensions (who, what, where, when) [36]. In [12] abstract interpretation of sets of pairs of program runs is used to deal with declassification according to the dimensions of [36]. An interesting use of statistical learning to understanding the security implications of the massive amounts of output produced by static analysis is given in [38] and the use of static analysis for enforcing context-dependent security policies is developed in [9]. A first attempt at dealing with time-dependent information flow policies is considered in [34] where the policies are translated to Timed Automata that are model checked.

In our approach the main challenge therefore is to find a way to allow elements of complete lattices to be used in flow sensitive security policies in such a way that it becomes algorithmically decidable whether or not a program adheres to the security policy. In order to pose as few conditions as possible on the static analyses used for analysing programs we shall be using two (likely) different complete lattices: an *analysis lattice* for the construction of sound approximations to the values that variables may have at program points, and a *policy lattice* for the properties used to determine the applicability of the flow sensitive properties.

Our development, illustrated in Fig. 1, then hinges on ensuring that:

- the analysis lattice is an atomistic complete lattice and there is an *atomistic Galois insertion* from the (collecting) semantics to the analysis lattice,
- the policy lattice is an atomistic complete lattice and there is an *atomistic Galois insertion* from the analysis lattice to the policy lattice, and
- the policy lattice has only *finitely* many atoms (whereas the analysis lattice is allowed to be infinite and even contain infinite chains).

In this paper we perform the development for a programming language with *concurrent processes, local storage* and *polyadic synchronous communication*. However, the framework developed is much more widely applicable and implementations will be fully automatic.

We use an illustrative example taken from our work with Airbus: a component of an avionics gateway [26] that essentially amounts to a multiplexer as shown in Fig. 2. The multiplexer merges data from different sources and forwards it over a joint channel to the demultiplexer that will split the data so as to reach different targets. The sources and targets have different security policies and the challenge is to express and enforce the policy of the merged data as it will depend on information within the data itself.

Overview. The language and its representation as a family of program graphs (one for each process) is covered in Section 2. We develop the necessary notion of atomistic Galois insertion in Section 3 and apply it to our program graphs. The use of atomistic Galois insertions facilitates a precise treatment of the content dependent information flow policies introduced in Section 4. The enforcement mechanism developed in Section 5 uses the results of the abstract interpretation and amounts to requiring the satisfaction of a system of constraints. We conclude with a soundness theorem. In Appendix A we provide further details on how one might be building the abstract domains in an independent attribute approach; however, we would like to stress that our theoretical results are valid in general.