Generating Information-Flow Control Mechanisms from Programming Language Specifications

Andrew Bedford (andrew.bedford.1@ulaval.ca)

Laval University, Canada

Motivation

Modern operating systems rely mostly on access-control mechanisms to protect users information. However, access control mechanisms are insufficient as they cannot regulate the propagation of information once it has been released for processing. To address this issue, a new research trend called *language-based information-flow security* [**?**] has emerged. The idea is to use techniques from programming languages, such as program analysis, monitoring, rewriting and type checking, to enforce information-flow policies (e.g., information from a private file should not saved in a public file). Mechanisms that enforce such policies (e.g., [**?**, **?**, **?**, **?**]) are called *information-flow control mechanisms*.

Problem

Developping sound information-flow control mechanisms can be a laborious and error-prone task, particularly when dealing with complex programming languages, due to the numerous ways through which information may flow in a program.

Background

Most information-flow control mechanisms seek to enforce a policy called *non-interference* [**?**], which essentially states that private information may not interfere with the publicly observable behavior of a program. To enforce non-interference, a mechanism must take into account two types of information flows: *explicit flows* and *implicit flows* [**?**].
An insecure explicit information flow (Listing 1) occurs when private information flows directly into public information.

```
public := private
```

Listing 1: Insecure explicit flow

Explicit flows can be prevented by associating labels to sensitive information and propagating them

Ott-IFC

Information-flow control mechanisms are usually designed and implemented from the ground-up by a human. In order to make this process less laborious and reduce the risk of errors, we have created a tool called that takes as input a programming language's specification (i.e., syntax and semantics) and produces a mechanism's specification (e.g., instrumented semantics).
As the name implies, the specifications that takes as input (and outputs) are written in Ott [**?**]. Ott is a tool that can generate LaTeX, Coq or Isabelle/HOL versions of a programming language's specification. The specification is written in a concise and readable ASCII notation that ressembles what one would write in informal mathematics (see following Listings).

Definition

Example

```
read value from privateFile;
w := 0;
x := value + 1;
x' := value + 2;
y := x mod 3;
z := y * 4;
write z to publicFile
```

Listing 3:

images/derivations-graph-large.png

Future Work

- Language Support
- Parametrization
- Generating Formal Proofs
- Verifying Existing Mechanisms

Acknowledgements

We would like to thank Josée Desharnais, Nadia Tawbi and the anonymous reviewers for their helpful comments.