# Ott-IFC: Automatically Generating Information-Flow Control Mechanisms

Andrew Bedford
Laval University
Quebec, Canada
andrew.bedford.1@ulaval.ca

## Abstract

We present Ott-IFC  a tool that can automatically generate information-flow control mechanisms from programming language specifications (i.e., syntax and semantics).

## 1 Problem and 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* [3] has emerged. The idea being 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 information-flow policies (e.g., [2]) are called *information-flow control mechanisms*.

Due to the numerous ways through which information may flow in a program, developping sound information-flow control mechanisms can be a challenging and error-prone task, particularly when dealing with complex programming languages. To help with this task, we have developped a tool called Ott-IFC that can automatically generate information-flow control mechanisms from programming language specifications (i.e., syntax and semantics).

## 2 Background and Related Work

One of the most studied information-flow policy is called *non-interference*. It essentially states that private information may not interfere with the publicly observable behavior of a program. In other words, that private information may not flow to public information. For example, Listings 1 and 2 both violate non-interference because someone observing the contents of `public` could learn something about `private`.

```
public := private
```

**Listing 1.** Insecure explicit flow

```
if (private > 0) then
  public := 0
else
  public := 1
end
```

**Listing 2.** Insecure implicit flow

Mechanisms that enforce non-interference are called *information-flow control mechanisms*, as they track and control where information may flow during the execution of a program.

In order to help language-based security researchers develop sound

## 3 Approach and Uniqueness

We chose to use Ott [4] as our input language. Ott is a programming language specification tool which

```
arith_expr, a ::= x | n | a1 + a2 | a1 * a2
bool_expr, b ::= true | false | a1 < a2
commands, c ::= skip | x := a | c1 ; c2 |
                if b then c1 else c2 end |
                while b do c end
```

**Listing 3.** Ott syntax of a simple imperative language

```
<a, s> || <n, s>
---------------------------------
<x := a, s> || <skip, s[x |-> n]>
```

**Listing 4.** Ott big-step semantics of the assign command

```
<b, s> || <true, s>
<c1, s> || <skip, s1>
-------------------------------------------
<if b then c1 else c2 end, s> || <skip, s1>

<b, s> || <false, s>
<c2, s> || <skip, s2>
-------------------------------------------
<if b then c1 else c2 end, s> || <skip, s2>
```

**Listing 5.** Ott big-step semantics of the if command

## 4 Current Status and Future Work

We have implemented a prototype of our algorithm [1] and validated that it works on two simple imperative languages: one defined using small-step semantics and the other using big-step semantics. We have also begun to draft a soundness proof, that is, a proof showing that the generated mechanisms enforce non-interference.

Before our tool can be of real use to most researchers, much work remains to be done.

*POPL 2018, January 07–13, 2018, Los Angeles, CA, USA*
2017. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM...$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

***Language Support*** Ott-IFC currently makes two assumptions about the language: (1) that the syntax be composed of expressions, which may only read the memory, and commands, which may read or write the memory; and (2) that the program configurations be of the form $\langle command, memory \rangle$. While these assumptions simplify the implementation, it also restricts the types of languages that can be used in Ott-IFC. For example, most functional languages would not be supported because, in those languages, functions can be expressions. We are currently in the process of building a repository of formalized languages so that we can test our approach on additional languages.

***Parametrization*** For the moment, Ott-IFC only generates one type of information-flow control mechanism: a type system which enforces termination-insensitive noninterference. We plan on parametrizing our tool so that users can choose the type of mechanism to generate (e.g., type system, monitor) and choose some of its features (e.g., flow-sensitivity, termination-sensitivity, progress-sensitivity).

***Generating Formal Proofs*** We expect that some users will use the mechanisms generated by Ott-IFC as a foundation to build better and more precise mechanisms. One of the most grueling task when building an information-flow control mechanism is to prove its soundness. In order to help those users, we plan on generating a skeleton of the proof in Coq or Isabelle/HOL (both languages are supported by Ott).

***Verifying Existing Mechanisms*** The same rules that Ott-IFC uses to generate sound mechanisms could be used to verify the soundness of existing mechanisms and identify potential errors.

## Acknowledgments

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

## References

[1] Andrew Bedford. 2017. Ott-IFC's Repository. https://github.com/andrew-bedford/ott-ifc. (2017).

[2] Andrew Bedford, Stephen Chong, Josée Desharnais, Elisavet Kozyri, and Nadia Tawbi. 2017. A progress-sensitive flow-sensitive inlined information-flow control monitor (extended version). *Computers & Security* 71 (2017), 114–131. https://doi.org/10.1016/j.cose.2017.04.001

[3] Andrei Sabelfeld and Andrew C. Myers. 2003. Language-based information-flow security. *IEEE Journal on Selected Areas in Communications* 21, 1 (2003), 5–19. https://doi.org/10.1109/JSAC.2002.806121

[4] Peter Sewell, Francesco Zappa Nardelli, Scott Owens, Gilles Peskine, Thomas Ridge, Susmit Sarkar, and Rok Strnisa. 2010. Ott: Effective tool support for the working semanticist. *J. Funct. Program.* 20, 1 (2010), 71–122. https://doi.org/10.1017/S0956796809990293