Limitations of Non-Interference

Flemming Nielson Hanne Riis Nielson Ximeng Li

DTU Compute, Technical University of Denmark, Denmark

{fnie,hrni,ximl}@dtu.dk

Submitted to NWPT 2015

Abstract

We show that non-interference falls short of providing a convincing semantic characterisation of information flow policies for confidentiality and integrity and motivate an approach based on instrumented semantics.

Introduction. We have been working with Airbus on developing security policies for dealing with the challenges of communication between security domains subject to strict safety concerns, and in the course of this work we have uncovered a limitation of non-interference in establishing convincing semantic characterisations of the required security policies. In this paper we illustrate this limitation on an utterly simple example and discuss ways of providing alternate semantic characterisations more acceptable to our industrial partners.

An illustrative example. Let us consider a simple process `D` that takes inputs `a`, `b`, and `c`, and produces outputs `d1=a+b` and `d2=b*c`.

```
1   process D
2   begin
3     input(a,b,c);
4     d1:=0; d2:=1;
5     d1:=d1+a; d2:=d2*b;
4     d1:=d1+b; d2:=d2*c;
7     output(d1,d2)
8   end
```

In the full development, the inputs would be received from other parallel processes and the outputs would be delivered to other parallel processes. Here we simply assume that the variables `a`, `b`, `c` belong to the processes `A`, `B`, `C`, respectively, and that the variables `d1` and `d2` both belong to the process `D`.

Security policies. Motivated by the Decentralized Label Model [3] the security policies of interest have two components. One component, R, tracks where the values of variables are allowed to flow and is useful for dealing with confidentiality; we shall say that it tracks the *readers* of variables. The other component, I, tracks what might have have influenced the values of variables and is useful for dealing with integrity; we shall say that it tracks the *influencers* (or writers) of variables. It is natural to require that $X \in R(x)$ and $X \in I(x)$ whenever the variable x belongs to the process X, and an example security policy might be given by the following definition of R and I:

	a	b	c	d1	d2
R	A,D	B,D	C,D	D	D
I	A	B	C	A,B,D	B,C,D

Typing the example. To analyse the example we need to define $R(x)$ and $I(x)$ for all variables in such a way that they satisfy conditions imposed by a type system that are intended to ensure that the annotations are correct. In our extremely simple program there are two principles for ensuring this.

One concerns assignments of the form $x\texttt{:=}y_1 \# y_2$ where $\#$ is one of the operators + or *. For confidentiality is is natural to impose that $R(x) \subseteq R(y_1) \cap R(y_2)$, or equivalently $R(x) \subseteq R(y_1) \wedge R(x) \subseteq R(y_2)$, because one should not allow any readers beyond those allowed by both y_1 and y_2. For integrity it is natural to impose that $I(x) \supseteq I(y_1) \cup I(y_2)$, or equivalently $I(x) \supseteq I(y_1) \wedge I(x) \supseteq I(y_2)$, because one should not forget any of the influencers of y_1 and y_2.

The other principle concerns assignments of the form $x\texttt{:=}c$ where c is a constant. These are always acceptable. To fit the model of the previous case we might say that $R(c) = \mathcal{U}$ and $I(c) = \emptyset$ where $\mathcal{U} = \{\texttt{A}, \texttt{B}, \texttt{C}, \texttt{D}\}$ is the universe of all processes and $\emptyset$ is the empty set.

The definition of R and I expressed in the Table above satisfies the constraints imposed by our example program and is in line with the Decentralized Label Model [3].

Non-Interference. We would imagine that our parallel language is equipped with an operational semantics. Configurations might take the form $\langle S, \sigma \rangle$ indicating that the system S of parallel processes is currently executing from the store σ. There would then be a transition relation $\langle S, \sigma \rangle \rightarrow \langle S', \sigma' \rangle$ indicating that one step of evaluation transforms $\langle S, \sigma \rangle$ into $\langle S', \sigma' \rangle$. We do not have the space to present the details of this transition relation nor to discuss the possibility of labelling it (as is often done to deal with communication).

The non-interference approach to the semantic characterisation of a security policy then introduces the following notions to be defined below: $S_1 \simeq S_2$ for when one system (S_1) is similar to another (S_2), $\sigma_1 \cong \sigma_2$ for when one store (σ_1) is close to another (σ_2), and a notion of when a variable is low.

Unlike the case of bisimulations, the simulation relation $\simeq$ is not necessarily reflexive, and the correctness of a type system amounts to ensuring that $S \simeq S$ whenever the system S is admitted by the type system.

We now provide the definitions of the three notions introduced. The system S_1 is similar to another S_2, written $S_1 \simeq S_2$, whenever $\langle S_1, \sigma_1\rangle \to \langle S_1', \sigma_1'\rangle$, $\sigma_1 \cong \sigma_2$, and $\langle S_2, \sigma_2\rangle \to \langle S_2', \sigma_2'\rangle$ ensure that $S_1' \simeq S_2'$ and $\sigma_1' \cong \sigma_2'$, and vice versa. This definition is recursive and needs to be interpreted co-inductively in the usual manner of bisimulations.

The store σ_1 is close to another σ_2, written $\sigma_1 \cong \sigma_2$, when they agree on all low variables: $\sigma_1(x) = \sigma_2(x)$ whenever x is low. A variable x is said to be low if its security level $L(x)$ (either $R(x)$ or $I(x)$ in our case) is dominated by some security value ℓ (a subset of $\mathcal{U}$ in our case) according to some partial order $\sqsubseteq$ (being $\supseteq$ for R and $\subseteq$ for I). The key property is that the set of low variables is closed under reducing the security classification under the partial order $\sqsubseteq$.

Most proofs of correctness [5] of a type system with respect to non-interference then rely on the type system ensuring that whenever y is somehow used in defining x (either explicitly or implicitly) then $L(y) \sqsubseteq L(x)$. This is fully in line with our explanation of typing the example above.

Limitations of Non-Interference. The above explanation uses lattice duality in sometimes choosing $\sqsubseteq$ to be $\subseteq$ and sometimes $\supseteq$. Let us rephrase the confidentiality component so that we can always use $\subseteq$ for $\sqsubseteq$.

This amounts to replacing $R(x)$ with its complement $\overline{R}(x) = \mathcal{U} \setminus R(x)$. The conditions imposed by typing are then changed to $\overline{R}(c) = \emptyset$ whenever c is a constant and to demanding $\overline{R}(x) \supseteq \overline{R}(y_1) \cup \overline{R}(y_2)$ for an assignment $x := y_1 \# y_2$. Intuitively, $\overline{R}(x)$ lists those processes *not* allowed to read x. Our typing becomes:

	a	b	c	d	e
$\overline{R}$	B,C	A,C	A,B	A,B,C	A,B,C
I	A	B	C	A,B,D	B,C,D

In other words we have explicitly replaced the lattice for R with a dual lattice for $\overline{R}$ which is unsurprising from a lattice theoretical point of view and quite in line with the usual statements of information flow that integrity is the dual of confidentiality (and used to motivate that technical developments often only focus on confidentiality).

So what is the point?

The point is that with this change the formal definition of non-interference is the same, symbol for symbol, for integrity as for confidentiality, and that there is not even the need to perform dual choices of the partial order.

What does this mean?

It means that while the non-interference result produces some validation of the type system against errors, it has no way of expressing whether or not $I(x)$ denotes the set of influencers of x, or rather the set of processes not allowed to read x; similarly for $\overline{R}(x)$. In other words:

> Non-interference is unable to express the correctness of the intuitive explanations of what the security policies for influencers and readers are supposed to capture.

Instrumented Semantics. In our work with Airbus we are using ideas from program analysis in order to overcome the limitations of non-interference. In particular the use of an instrumented operational semantics where each transition is labelled with the *flow* taking place. In the case of an assignment statement performed by process Z we would have the following transition:

$$\langle x := y1 \# y2 ; S, \sigma \rangle \rightarrow_{(y1,x),(y2,x),(y1,Z),(y2,Z),(Z,x)} \langle S, \sigma[x \mapsto \sigma(y1) \# \sigma(y2)] \rangle$$

Here the subscript on the arrow indicates that both $y1$ and $y2$ are involved in producing x. Additionally we record that the process Z is reading $y1$ and $y2$ and is influencing (writing) the variable x. The full semantics would extend this to the other constructs in our parallel programming language and deal with both explicit (as illustrated) and implicit flow (not illustrated here).

In the full development we will allow policies to be influenced by the values of variables, so as to model content-dependent security policies. Much as in a Hoare logic [1] there would then be a policy (I, R) pertaining to the program point *before* the action and a possibly different policy (I', R') pertaining to the program point *after* the action.

The semantic correctness of the security policies with respect to a system S is then expressed by requiring that whenever

$$\langle S, \sigma \rangle \rightarrow^*_{\ldots} \langle S', \sigma' \rangle \rightarrow_F \langle S'', \sigma'' \rangle$$

then we insist for the security policy (R, I) *before* the action and the security policy (R', I') *after* the action, that the following property

$$(R, I) \triangleright F \triangleleft (R', I')$$

holds. It is defined as follows:

- whenever $(Z, x) \in F$ we have $Z \in I'(x)$,
- whenever $(y, Z) \in F$ we have $Z \in R(y)$, and
- whenever $(y, x) \in F$ we have $I(y) \subseteq I'(x)$ and $R'(x) \subseteq R(y)$.

This formulation makes it clear that constraints regarding influencers flow in the *forward* direction whereas constraints regarding readers flow in the *backward* direction. In this way we would be thinking of integrity as a forward analysis problem (like reaching definitions [4]) and confidentiality as a backward analysis problem (like live variables). This formulation clearly indicates the different directions of flow needed for formalizing integrity and confidentiality.

Conclusion. We have shown that non-interference falls short of providing convincing semantic explanations of the correctness of security policies for confidentiality and integrity as found in information flow frameworks like the Decentralized Label Model [3].

This contradicts conventional wisdom in the area of security policies for information flow. To quote an anonymous reviewer on a paper lacking a non-interference result: "My main complaint is the independence of the annotations from the actual semantics of the program and the non-interference properties it may have."

This may be contrasted with the approach of static analysis where hardly any non-interference results are proved. To quote an international reviewer on a project attempting to establish such results: "Non-interference is a rather restrictive property so I am not totally convinced that one should start with it as a requirement."

Our proposal therefore is to provide convincing semantic explanations of the correctness of security policies for confidentiality and integrity using suitably instrumented versions of an operational semantics.

Acknowledgement. We are supported by IDEA4CPS [2] and benefitted from discussions with Michael Paulitsch and Kevin Müller from Airbus.

References

[1] Krzysztof R. Apt. Ten years of Hoare's logic: A survey - part 1. *ACM Transactions on Programming Languages and Systems*, 3(4):431–483, 1981.

[2] IDEA4CPS: Foundations for Cyber-Physical Systems. Danish Research Foundations for Basic Research (Project DNRF86-10). Webpage: http://idea4cps.dk.

[3] Andrew C. Myers and Barbara Liskov. A decentralized model for information flow control. In *16th ACM Symposium on Operating Systems Principles*, pages 129–142, 1997.

[4] F. Nielson, H. Riis Nielson, and C. L. Hankin. *Principles of Program Analysis*. Springer, 1999. Second printing, 2005.

[5] Dennis M. Volpano, Cynthia E. Irvine, and Geoffrey Smith. A sound type system for secure flow analysis. *Journal of Computer Security*, 4(2/3):167–188, 1996.