Implementations that Preserve Confidentiality (Extended Abstract)

Jan Cederquist Pablo Giambiagi
Swedish Institute of Computer Science*

1. Introduction

The problem we address here is proving that a program respects the confidentiality properties of a specification. In particular we are interested in programs implementing, for instance security protocols that rely on cryptographic primitives to establish secrecy.

As definition of secrecy we use admissibility, which lets us control the amount of secret information that is leaked by a process. Admissibility was introduced in [2]. In section 2 a more thorough presentation of admissibility is given in terms of an operational semantics corresponding to a language where the programs we are interested in can be expressed (such as Abadi-Gordon's spi calculus [1]).

In section 3 we show how to tailor admissibility with respect to a specification, in such a way that, if an admissible process p outputs some information about secrets then so does the specification.

In this paper we only work on the semantical level, but in our research we are also working on a simple type system and a type checking algorithm to verify admissibility of programs with respect to specifications of security protocols.

2. Admissibility

Admissibility correlates changes in observable behavior with changes in input. These changes are modeled using a *relabeling*, which is a function from actions to actions performed by a process. Given a process p and a relabeling f, we syntactically construct $p[f]$. The relabeling is then given the following transition semantics: $p[f]$ performs $f(\alpha)$ and becomes $p'[f]$ iff p performs α and becomes p'. If $\sim$ is some equivalence relation, $p[f] \sim p$ means that the changes f performs on p's input are canceled out by the changes it performs on p's output. Given a set F of relabeling functions, admissibility is defined as

$$\forall f \in F. p[f] \sim p.$$

By choosing an appropriate set F we can correlate changes in inputs with changes in behavior. If we can identify the inputs that introduce secret information to the system, we can also control how much of it is leaked.

Now we give the semantics by defining the actions in the language and how values are formed:

Definition 1 (Actions)

$$\begin{array}{llll} (\textit{Action}) & \alpha & ::= & \tau \mid v?k \mid v!v \\ (\textit{Val}) & v & ::= & k \mid v??k \mid \textit{pf}(v_1, \ldots, v_n) \end{array}$$

where τ is the unobservable action, k ranges over constants, v ranges over values, and pf ranges over primitive functions. Note the distinction between $v?k$ and $v??k$; the action $v?k$ reads the value $v??k$ from channel v.

The set of secrets is defined from their entry points:

Definition 2 (Entries and Secrets)

1. *A set of entries $E \subseteq$ Val identifies the entry points of secrets.*
2. *The set of secrets $S \subseteq$ Val consists of all values that depend on inputs from entry points in E:*

$$\frac{v \in E \cup S}{v??k \in S} \qquad \frac{v_i \in S}{\textit{pf}(v_1, \ldots, v_n) \in S}$$

The entry points can for instance be the names of all local channels. Then, if p is admissible (definition 4), p can only leak information about locally stored values if the specification allows that.

The relabeling functions are defined using functions that transform secret values. These secret permuters must respect the structure of how values are formed (otherwise there is no possibility of correlating changes in input with changes in output).

Definition 3 (Secret permuter) *A secret permuter g should satisfy $g^{-1} = g$ and*

$$\begin{array}{ll} g(k) & = k \\ g(\textit{pf}(\bar{v})) & = \textit{pf}(g(v_1), \ldots, g(v_n)) \\ g(v??k) & = \begin{cases} g(v)??k', & \textit{for some } k', \textit{ if } v??k \in S \\ v??k, & \textit{otherwise} \end{cases} \end{array}$$

*Authors' address: Swedish Institute of Computer Science, Box 1263, S-164 29 Kista, Sweden, {janc,pablo}@sics.se

We denote the set of all secret permuting functions by X_S.

The relabeling functions are then defined as

$$\begin{aligned} f(\tau) &= \tau \\ f(v?k) &= g(v??k) \\ f(v_1!v_2) &= g(v_1)!g(v_2), \end{aligned}$$

where g ranges over X_S. The set of all such relabelings are denoted F_E. Admissibility is then defined by quantification over this set.

Definition 4 (Admissibility [2]) *Given secret entries E and $F \subseteq F_E$, process p is F-admissible if $\forall f \in F.p[f] \sim p$.*

3. Confidential implementation

We are interested in verifying implementation of security protocols. As such, our programs are not completely restricted by the specification: they can do much more, provided that they still respect confidentiality. An specification is a pair (A, E) where A is a process and E is a set of entries. Intuitively, process p implements (A, E) if it manipulates secrets (as derived from E) only in the way stated by A.

Secrets are manipulated in critical actions (*Crit* $\in$ *Action*), as defined by:

$$\frac{v \in S}{v?k \in \mathit{Crit}} \qquad \frac{v_i \in S \text{ for some } i}{v_1!v_2 \in \mathit{Crit}}$$

If the semantics of process p_A is given by the rules:

$$\frac{p \xrightarrow{\alpha} p' \quad A \xrightarrow{\alpha} A'}{p_A \xrightarrow{\alpha} p'_{A'}} \qquad \frac{p \xrightarrow{\alpha} p' \quad A \not\xrightarrow{\alpha} \quad \alpha \notin \mathit{Crit}}{p_A \xrightarrow{\alpha} p'_A},$$

then we have:

Definition 5 (Implementation) *A process p implements a specification (A, E) if and only if $p \sim p_A$.*

An implementation of (A, E) does not necessarily leak information about secrets as A. The idea here is to define the set F of Definition 4, such that if p is F-admissible, then it does not leak more information about secrets in S than what A does.

A problem we encounter with non data independent programs (i.e. programs that perform tests on secrets) is that a relabeling f may make the tests performed by p and $p[f]$ yield different results, so that $p[f] \sim p$ does not hold. This is solved here by letting the specification A force restrictions on the set of relabelings, in such a way that we only consider "data independent equivalence classes" of A. In other words, f changes a behavior of p into another yielding the same results on the tests performed by A. Since this is easily obtained by requiring $A[f] \sim A$, we can now restrict the relabeling set F in the desired way:

Definition 6 (Confidential implementation) *Process p confidentially implements specification (A, E) if:*

1. *p implements (A, E).*
2. *$\forall f \in F_E.A[f] \sim A \Rightarrow p[f] \sim p$*

We write this as $A \sqsubseteq_E p$.

4. Conclusions

In this paper we have addressed the problem of describing what it means for a program to preserve the confidentiality properties of a specification. We have also produced a detailed description of admissibility in the context of an interesting operational semantics, and have applied it to our problem in a way that easily accommodates the kind of information downgrading that is inherent to, say, cryptographic protocols.

Most program analyses for security consist on the verification of noninterference properties, which basically disallow any downgrading. In [3], Ryan and Schneider presented a generalisation of noninterference based on the same idea of measuring information leakage by defining equivalence classes over high level behaviors. However, our presentation does neither require a distinction between security levels, nor depends on some variation of trace equivalence. Besides, our approach is targeted to a more concrete context: that of defining confidentiality with respect to a specification.

We are currently working on the definition of a type system for a simple imperative programming language parameterised by a specification (A, E), such that if the program type checks then it confidentially implements (A, E).

References

[1] M. Abadi and A. D. Gordon. A calculus for cryptographic protocols: The Spi Calculus. In *Proc. 4th ACM Conference on Computer and Communications Security*, pages 36–47, 1997. Full version available as tech. rep. 414, Univ. Cambridge Computer Lab.

[2] M. Dam and P. Giambiagi. Confidentiality for mobile code: The case of a simple payment protocol. To appear in Proceedings of the 13th IEEE Computer Security Foundations Workshop, Feb. 2000.

[3] P. Y. A. Ryan and S. A. Schneider. Process algebra and noninterference. In *Proceedings of CSFW-12*, pages 214–227, Mordano, Italy, June 1999. IEEE.