Implementations that Preserve Confidentiality (Extended Abstract)

Jan Cederquist
Swedish Institute of Computer Science
Box 1263, S-164 29 Kista, Sweden
janc@sics.se

Pablo Giambiagi
Dept. of Teleinformatics, KTH/IT
Electrum 204, S-164 40 Kista, Sweden
pgiamb@it.kth.se

1. Introduction

The problem we address here is showing that a program satisfies certain confidentiality properties. We are in particular interested in programs implementing 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 [1]. In section 2 a more thorough presentation of admissibility is given in terms of an operational semantics.

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.

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: $p[f]$ can perform $f(\alpha)$ and become $p'[f]$ iff p can perform α and become p'. For some equivalence relation $\sim$, $p[f] \sim p$ 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

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

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, v and pf range over constants, values, and primitive functions, respectively. 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 (Relabeling functions) *A secret permuter g satisfies $g^{-1} = g$ and*

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

The set of all secret permuting functions is denoted X_S.

For every $g \in X_S$ there is a corresponding relabeling function:

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

The set of all such relabelings is denoted F_E.

Admissibility is then defined by quantification over subsets of F_E:

Definition 4 (Admissibility [1]) *Given secret entries E and $F \subseteq F_E$, a process p is F-admissible if*

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

If the inputs that introduce secret information can be identified, then by choosing an appropriate set F we can control the leakage of secrets.

3. Confidential implementation

A 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* $\subseteq$ *Action*), as defined by:

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

Let p_A be the restriction of p to those behaviors that A allows:

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

We say that a process p implements the specification (A, E) if $p \sim p_A$. This implies that the (critical) actions p performs are the correct ones, and they occur in the correct order. However, confidentiality properties are in general not preserved by refinement. For instance, there is an information leakage if the program branches on secret input and the branches perform different non critical actions. To control these implicit leaks we use admissibility.

Below we let the specification A force restrictions on the set of relabelings, in such a way that only "data independent equivalence classes" of A are considered. This is obtained by requiring $A[f] \sim A$.

Definition 5 (Confidential implementation) *Process p confidentially implements specification (A, E) if:*

1. $p \sim p_A$.
2. $\forall f \in F_E. A[f] \sim A \Rightarrow p[f] \sim p$

We write this as $A \sqsubseteq_E p$.

Definition 5 can be justified in terms of observations (properties of secret input). Let a trace be a sequence of actions $\alpha = \alpha_1\alpha_2 \ldots$ a process can perform; then for $f \in F_E$, we define $f(\alpha) = (f\alpha_1)(f\alpha_2) \ldots$. If p implements (A, E), a trace α in p can be projected into a corresponding trace $\alpha_{|A}$ in A by removing certain non critical actions. Properties of secret input are here identified with equivalence classes of *indistinguishable traces*. Observers are identified with sets of relabeling functions $F \subseteq F_E$ such that F contains the identity function and is closed under composition. The observation of secret input of the system$X$ by an observer F can then be formalised as

$$O^F_X(\alpha) = \{f(\alpha) \mid f \in F \wedge X[f] \sim X\}.$$

Definition 5 then says that, if p is a confidential implementation of A, then an observation in A is as least as good as the corresponding observation in p:

$$A \sqsubseteq_E p \;\Rightarrow\; O^F_A(\alpha_{|A}) \subseteq O^F_p(\alpha)_{|A}.$$

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 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 [2], 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 variations of trace equivalence. Besides, our approach is targeted to a more concrete context: that of defining confidentiality with respect to a specification.

References

[1] 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.

[2] 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.