



HAL
open science

Formal Specification and Validation of Security Policies

Tony Bourdier, Horatiu Cirstea, Mathieu Jaume, Hélène Kirchner

► **To cite this version:**

Tony Bourdier, Horatiu Cirstea, Mathieu Jaume, Hélène Kirchner. Formal Specification and Validation of Security Policies. FPS - 4th Canada-France MITACS Workshop on Foundations and Practice of Security - 2011, May 2011, Paris, France. pp.148-163, 10.1007/978-3-642-27901-0_12 . inria-00507300v2

HAL Id: inria-00507300

<https://inria.hal.science/inria-00507300v2>

Submitted on 22 Feb 2011

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Formal Specification and Validation of Security Policies

Tony Bourdier¹, Horatiu Cirstea¹, Mathieu Jaume², and H el ene Kirchner³

¹ INRIA Nancy - GRAND-EST Research Center & Nancy University & LORIA

² SPI LIP6, Universit e Paris 6

³ INRIA Bordeaux - SUD-OUEST Research Center

Abstract. We propose a formal framework for the specification and validation of security policies. To model a secured system, the evolution of security information in the system is described by transitions triggered by authorization requests and the policy is given by a set of rules describing the way the corresponding decisions are taken. Policy rules are constrained rewrite rules whose constraints are first-order formulas on finite domains, which provides enhanced expressive power compared to classical security policy specification approaches like the ones using Datalog, for example. Our specifications have an operational semantics based on transition and rewriting systems and are thus executable. This framework also provides a common formalism to define, compare and compose security systems and policies. We define transformations over secured systems in order to perform validation of classical security properties.

1 Introduction

When addressing the field of security policies in computer science, we are faced to multiple definitions of this concept, most often based on their purpose rather than on their behavior. For instance, in a very generic way, one can say that the purpose of a security policy is to define what it means to be secure for a system, an organization or another entity. With this point of view, security policies can be seen as special procedures that deliver authorizations to perform specific actions: for instance, they decide whether or not an access is granted, whether or not a transaction may be approved, possibly taking into account the history of transactions (e.g., on a bank account, the total amount of cash withdrawal during the month should not exceed a fixed amount), or priority considerations (e.g., an emergency call is always given priority).

The additional specificity of security policies is their reactive behaviour with respect to their execution environment: on one hand, a target system may query the policy for an authorization before performing specific accesses or transactions; on the other hand, the answers of the policy not only determine the way the corresponding action is handled in the system but can also modify the (security) information of the system and consequently subsequent executions. For example, a negative authorization from an ATM machine security policy due to

an incorrect PIN not only prevents immediate money withdrawal but can also induce a (bad PIN) counter incrementation and lead to a permanent blocking of the corresponding account after a certain number of unsuccessful attempts. So, the security information could be seen as part of the target system but it is also intrinsic to the corresponding policy whose decisions strongly depend on it.

Our first contribution is to provide a framework which can be used to formalise separately the security system that manipulates all the security information used for producing the authorization decisions and the policy rules that compute the decisions. This separation is relevant not only for a conceptually clear specification and design, but also for the verification, comparison and composition of policies. In particular, this allows one to analyse separately properties related to the management of the security information (expressed as invariants of the security system) and properties related to the policy rules (consistency or completeness for example). A security system is formalised as a transition system whose states are generated by syntactic environments, and whose transitions are described by transition rules on environments. Each transition is triggered by an event which corresponds to an authorization given by the security policy. The policy is given as a set of rules describing the way the decisions are taken. Policy rules are constrained rewrite rules, whose constraints are first-order formulas solved in the current state of the transition system. According to the authorization, the transition rule may or may not apply. So, conceptually, the security policy restricts the possible transitions of the security system. Such specifications of security systems and policies have a well-understood operational semantics based on transition and rewriting systems and are thus executable.

Our second contribution is a transformation based approach which can be used to check generic security properties for security systems which have been independently designed. A security policy is often expected to fulfill a certain security property expressed on some entities, while it is dealing with a different set of entities. A typical example is given by access control policies designed for ensuring flow properties: such policies do not deal with information flow but only with objects containing information to be traced. Indeed, access control policies allow to grant or to revoke the rights for active entities (the subjects) to access some passive entities (the objects), but cannot always control how the information is used once it has been accessed (there is no control on its propagation). Intuitively, a link is needed between “what you do” (the policy) and “what you want” (the goal for which the policy is designed). We formalize this link through a transformation of environments, whose aim is to translate an environment into another one dealing with the entities we are interested in. We can use, for example, such a link to define a flow-based interpretation of access control policies. This kind of transformations allow thus the validation of a property over a sys-

tem even if the property is expressed in a different specification. In practice, this approach provides a way to reuse the same specification of a security property in order to analyse or to verify several policies and systems, thus showing the benefits of a library of generic security properties, dedicated to particular domains (like information flows) and that can be considered in several contexts. For example, it becomes possible to check the same information flow property expressed by a specification for several access control policies.

We first introduce some useful notions and notations in Section 2. Section 3 presents the different components of our specification framework: security signatures, environments, transition rules as well as security systems, policy rules, and secured systems. Section 4 addresses the validation point of view by defining environment transformations and illustrating the verification of security properties. In Section 5, we compare our approach with other works. Conclusion and future work are presented in Section 6.

2 Preliminaries

We assume the reader familiar with the standard notions of term rewriting, first order logic and Datalog. This section briefly recalls basic notions used in this paper; more details can be found in [16] for logic considerations, in [2] for rewriting considerations and in [24] for Datalog related notions.

A many-sorted signature $\Sigma = (\mathcal{S}, \mathcal{F}, \mathcal{P})$ is given by a set of sorts \mathcal{S} , a set of function symbols \mathcal{F} and a set of predicate symbols \mathcal{P} . A function symbol f with arity $\mathfrak{s}_1, \dots, \mathfrak{s}_n \in \mathcal{S}$ and co-arity \mathfrak{s} is written $f:\mathfrak{s}_1, \dots, \mathfrak{s}_n \mapsto \mathfrak{s}$. A predicate symbol p with arity $\mathfrak{s}_1, \dots, \mathfrak{s}_n \in \mathcal{S}$ is written $p:\mathfrak{s}_1, \dots, \mathfrak{s}_n$. Variables are also sorted and the notation $x:\mathfrak{s}$ specifies that the variable x has sort \mathfrak{s} . We assume in this paper that all variables are ranging over finite sets. This condition can be relaxed under some conditions [24], especially for allowing built-in sorts such as integers. Given a set ζ extending a set of variables \mathcal{X} (possibly empty) with constants sorted by \mathcal{S} , the set of Σ -terms over ζ denoted by $\mathcal{T}_{\Sigma, \zeta}^{\mathfrak{s}}$ is the smallest set containing elements of ζ of sort \mathfrak{s} and all the $f(t_1, \dots, t_n)$ such that $f:\mathfrak{s}_1, \dots, \mathfrak{s}_n \mapsto \mathfrak{s} \in \Sigma$ and $t_i \in \mathcal{T}_{\Sigma, \zeta}^{\mathfrak{s}_i}$ for $i \in [1..n]$. We write $\mathcal{T}_{\Sigma}^{\mathfrak{s}}$ instead of $\mathcal{T}_{\Sigma, \emptyset}^{\mathfrak{s}}$ and the sort is omitted when not important in the context. We also consider a partial ordering $<$ on the set \mathcal{S} of sorts of a signature Σ and we write $\mathfrak{s}_1 < \mathfrak{s}_2$ if $\mathcal{T}_{\Sigma, \zeta}^{\mathfrak{s}_1} \subseteq \mathcal{T}_{\Sigma, \zeta}^{\mathfrak{s}_2}$. $\mathcal{Pos}(t)$ denotes the set of positions of a term t , $t|_{\omega}$ denotes the subterm of t at position ω , and $t[u]_{\omega}$ the term t with the subterm at position ω replaced by u . The set of variables occurring in a term t is denoted by $\mathcal{Var}(t)$. If $\mathcal{Var}(t)$ is empty, t is called a ground term. All the following definitions are given *w.r.t.* to a set ζ whose subset of variables is denoted by \mathcal{X} . A substitution is a mapping from \mathcal{X} to $\mathcal{T}_{\Sigma, \zeta}$ which is the identity except over a finite set of

variables called domain of σ and denoted by $Dom(\sigma)$. σ naturally extends to an endomorphism of $\mathcal{T}_{\Sigma, \zeta}$. If any variable in the domain is mapped to a ground term then, the corresponding substitution is called ground. A Σ -atom is of the form $p(t_1, \dots, t_n)$ or $t_1 = t_2$ with $p \in \mathcal{P}$ and $t_1, \dots, t_n \in \mathcal{T}_{\Sigma, \zeta}$. A Σ -literal is either a Σ -atom or a negated (with \neg) Σ -atom and the set of Σ -formulae built out of Σ -literals is denoted by $\mathcal{F}or_{\Sigma, \zeta}$. The set of free variables of a formula ϕ (i.e. variables not in the scope of a quantifier) is denoted by $\mathcal{F}Var(\phi)$. A logical rule over Σ , denoted by $a \leftarrow l_1 \wedge \dots \wedge l_n$, consists of a conjunction of Σ -literals l_i called the body and a Σ -atom a called the goal.

A constrained rewrite rule over a signature Σ is a 3-tuple $(l, \varphi, r) \in \mathcal{T}_{\Sigma, \mathcal{X}} \times \mathcal{F}or_{\Sigma, \mathcal{X}} \times \mathcal{T}_{\Sigma, \mathcal{X}}$, denoted by $l \xrightarrow{\varphi} r$, such that $Var(r) \subseteq Var(l) \cup \mathcal{F}Var(\varphi)$. A constrained term rewrite system (CTRS) \mathcal{R} is a set of constrained rewrite rules. We say that $t \in \mathcal{T}_{\Sigma}$ rewrites into a term $t' \in \mathcal{T}_{\Sigma}$ with respect to \mathcal{R} and a Σ -theory ϑ , which is denoted by $t \xrightarrow{\vartheta}_{\mathcal{R}} t'$ iff there exist a position $p \in Pos(t)$, a rewrite rule $l \xrightarrow{\varphi} r \in \mathcal{R}$, and a ground substitution σ with $Dom(\sigma) = Var(l) \cup \mathcal{F}Var(\varphi)$ such that $\vartheta \models \{t|_p = \sigma(l) ; t' = t[\sigma(r)]_p ; \sigma(\varphi)\}$.

3 Secured systems

A security policy responds to the authorization requests of a system according to a certain number of rules and to the configuration of the system at the moment of the request. We consider thus that a system constrained by a security policy consists of two parts: on one hand, the set of rules describing the way the decisions are taken and on the other hand, the information used by the rules and the way these evolve in the system. We call the former the policy rules and the latter the security system. In our framework all objects manipulated by the security system and the policy rules are described as first order terms over a common signature called the security signature. We define the security system using transition rules and the policy rules as a constrained rewrite system.

3.1 Security signature

A transition of the security system is triggered when an authorization request occurs and the result of the respective transition depends on the corresponding decision. We thus call *events* the pairs consisting of an authorization request and the associated decision and the security signature always defines the sorts *Query* and *Decision* corresponding to the sorts of the first and respectively second element of such a pair.

Definition 1. A **security signature** is a signature $\Sigma_{Sys} \cup \Sigma_{Ev}$ such that Σ_{Ev} contains two sorts *Query* and *Decision* with $Decision < Query$ and a set of function symbols whose co-arity belongs to \mathcal{S}_{Ev} .

Example 1. Along the lines of this paper, we consider an access control system on which we define a confidentiality policy (which can be viewed as a variant of the mandatory part of the Bell and LaPadula policy [6]). This policy constrains accesses done by subjects (S) over objects (O) according to access modes (A) by considering levels of security belonging to a finite lattice (L, \preceq) associated with subjects and objects. Hence, we introduce the security signature $\Sigma_{Sys} \cup \Sigma_{Ev}$ as follows. First, $\Sigma_{Sys} = (\mathcal{S}_{Sys}, \mathcal{F}_{Sys}, \mathcal{P}_{Sys})$ consists of $\mathcal{S}_{Sys} = \{S, O, A, L\}$ and

$$\mathcal{P}_{Sys} = \left\{ \begin{array}{l} \preceq : L, L \\ m : S, O, A \\ sudo : S \\ redlist, blacklist : S \end{array} \right\} \quad \mathcal{F}_{Sys} = \left\{ \begin{array}{ll} read : \mapsto A, & f_s : S \mapsto L \\ write : \mapsto A, & f_o : O \mapsto L \\ erase : \mapsto A, & root : \mapsto S \\ topSecret : \mapsto L \end{array} \right\}$$

The functions f_s and f_o describe security levels associated with subjects and objects; $root$ (resp. $topSecret$) is a particular subject (resp. security level). The predicate m describes current accesses over objects by subjects: $m(s, o, a)$ means that the subject s performs an access of type a over an object o . The predicate $sudo$ describes “sudoers”, *i.e.* users with root privileges. $redlist$ and $blacklist$ respectively indicate if a subject has requested one or more successive denied accesses. The signature Σ_{Ev} is based on the following function symbols:

$$\mathcal{F}_{Ev} = \left\{ \begin{array}{ll} ask : S, O, A \mapsto Query, & release : S, O, A \mapsto Query, \\ deny : \mapsto Decision, & permit : \mapsto Decision \end{array} \right\}$$

$ask(s, o, a)$ (resp. $release(s, o, a)$) means that the subject s asks to get (resp. to release) an access over an object o according to the access mode a .

3.2 Environments and transition rules

A security system is a transition system that describes the way security information evolve. The states of the system are defined intensionally by a set of kernel information (that can be modified by the transition rules of the system) and an immutable set of closure rules used to compute the complete security information. The result of such a computation represents the extensional description of the corresponding state.

Definition 2. An environment η over a signature $\Sigma = \langle \mathcal{S}, \mathcal{F}, \mathcal{P} \rangle$ consists of:

- (i) a **domain**: a finite set $|\eta|$ of sorted constants which contains all the constants of Σ ;
- (ii) a **base of facts**: a finite set \mathcal{B}_η of atoms of the form $p(t_1, \dots, t_n)$ with $p \in \mathcal{P}$, $n > 0$ and $t_1, \dots, t_n \in |\eta|$.
- (iii) a **base of equalities**: a finite set \mathcal{E}_η of equalities of the form $f(t_1, \dots, t_n) = t$ with $f \in \mathcal{F}$, $n > 0$ and $t_1, \dots, t_n, t \in |\eta|$ which does not contain two equalities with the same left-hand side;
- (iv) **closure rules**: a set \mathcal{R}_η of safe and stratified logical rules over Σ .

Safety and stratification of logical rules are well-known notions [24] imposing that (i) any variable occurring in a rule has a bounded domain and (ii) negations wrapped inside recursion are forbidden.

The base of equalities gives the interpretation into the domain of the environment for any term of the signature. We denote by $t \downarrow_\eta$ the interpretation of the term t in $|\eta|$, i.e. $f(t_1, \dots, t_n) \downarrow_\eta = u$ iff $f(u_1, \dots, u_n) = u \in \mathcal{E}_\eta$ and $u_i = t_i \downarrow_\eta$ for all $i \in [1, n]$. If $t = t \downarrow_\eta$ we say that t is η -normalized.

Example 2. If we consider the security signature Σ introduced in Example 1, we can define the environment η as follows. The domain $|\eta|$ contains the constants *Alice* and *Charlie* of sort S , the constant *PwdFile* of sort O , and the constants *Secret*, L_1 , L_2 , *Public*, of sort L . The base of facts \mathcal{B}_η (partially) defines the partial order \preceq and states that *Charlie* is a *sudoer*:

$$\mathcal{B}_\eta = \left\{ \begin{array}{l} L_1 \preceq \textit{Secret}, L_2 \preceq \textit{Secret}, \textit{Public} \preceq L_1, \textit{Public} \preceq L_2, \\ \textit{Secret} \preceq \textit{topSecret}, \textit{sudo}(\textit{Charlie}) \end{array} \right\}$$

The base of equalities \mathcal{E}_η provides a definition for the security levels associated with the subjects defined in the domain:

$$\mathcal{E}_\eta = \left\{ \begin{array}{l} f_s(\textit{root}) = \textit{topSecret}, f_s(\textit{Alice}) = L_2, f_s(\textit{Charlie}) = \textit{Public}, \\ f_o(\textit{PwdFile}) = \textit{Secret}, \end{array} \right\}$$

The set of closure rules completes the definition of \preceq :

$$\mathcal{R}_\eta = \{ x \preceq x \leftarrow \quad ; \quad x \preceq z \leftarrow x \preceq y \wedge y \preceq z \}$$

Due to the restrictions imposed on the domain and on the formulas, we have:

Proposition 1. *For any environment η over Σ , there exists a unique and computable least fixpoint of the logic program consisting of \mathcal{B}_η , \mathcal{E}_η and \mathcal{R}_η . This fixpoint is denoted by $\llbracket \eta \rrbracket$ and is called the **semantics** of η .*

Since $|\eta|$ is finite, the validity of any first-order formula in $\llbracket \eta \rrbracket$ is decidable.

The transition rules of a security system describe the evolution of environments. Several approaches like, for example, action languages [13] or updating rules [1], have been already used for the specification of logical models change. We follow in this paper the main ideas of the latter approach.

Definition 3. *An **update** u over $\Sigma = (\mathcal{S}, \mathcal{F}, \mathcal{P})$ is a logical rule ($\textit{goal} \leftarrow \textit{body}$) of one of the following form:*

$$\begin{array}{ll} (i) & p(x_1, \dots, x_n) \leftarrow \varphi \quad \textit{or} \\ (ii) & \neg p(x_1, \dots, x_n) \leftarrow \varphi \quad \textit{or} \\ (iii) & f(x_1, \dots, x_n) = y \leftarrow \varphi \wedge y = t \end{array}$$

where x_i and y are variables, $p \in \mathcal{P}$, $f \in \mathcal{F}$ and φ is a formula. u is applicable iff (in the case (iii)) $\text{Var}(t) \subseteq \{x_1, \dots, x_n\}$. In this case, the application of u over the environment η is the environment $\eta' = \eta \oplus u$ such that:

$$\begin{aligned}
\text{case (i)} \quad \mathcal{B}_{\eta'} &= \mathcal{B}_{\eta} \cup \left(\bigcup_{\llbracket \eta \rrbracket \models \mu(\varphi)} p(\mu(x_1), \dots, \mu(x_n)) \right) \\
\text{case (ii)} \quad \mathcal{B}_{\eta'} &= \mathcal{B}_{\eta} \setminus \left(\bigcup_{\llbracket \eta \rrbracket \models \mu(\varphi)} p(\mu(x_1), \dots, \mu(x_n)) \right) \\
\text{case (iii)} \quad \mathcal{E}_{\eta'} &= \mathcal{E}_{\eta} \setminus \{f(\mu(x_1), \dots, \mu(x_n)) = t' \in \eta \mid \llbracket \eta \rrbracket \models \mu(\varphi \wedge y = t)\} \\
&\quad \cup \{f(\mu(x_1), \dots, \mu(x_n)) = \mu(y) \mid \llbracket \eta \rrbracket \models \mu(\varphi \wedge y = t)\}
\end{aligned}$$

where μ are maps from $\{x_1, \dots, x_n, y\}$ to $|\eta|$.

The result of applying a sequence of updates $U = (u_1, \dots, u_n)$ over η is the environment $\eta \oplus U = (\dots (e \oplus u_1) \oplus u_2) \dots) \oplus u_n$.

Applying an (applicable) update over an environment consists thus in adding (or removing) the fact or changing the equality corresponding to its goal for any substitution making its body true in the environment.

The transitions of the security system describing the evolution of the states are sequences of updates labelled by the events that trigger them:

Definition 4. A Σ -transition rule is a pair of terms $\text{event} \in \mathcal{T}_{\Sigma, \mathcal{X}}^{\text{Query}} \times \mathcal{T}_{\Sigma, \mathcal{X}}^{\text{Decision}}$ whose subterms are constants or variables of Σ_{Sys} together with a sequence of updates U over Σ such that $\sigma(U)$ ⁴ is applicable for any ground substitution σ of domain $\text{Var}(\text{event})$. A transition rule is usually written $(\text{event}) : U$. For any transition rule $r = (\text{event}) : U$, $\eta \xrightarrow{\text{evt}}_r \eta'$ iff $\text{evt} = \sigma(\text{event})$ and $\eta' = \eta \oplus \sigma(U)$. Moreover, for any set of rules δ , $\eta \xrightarrow{\text{evt}}_{\delta} \eta'$ iff there is a rule $r \in \delta$ such that $\eta \xrightarrow{\text{evt}}_r \eta'$.

Example 3. If we consider the security signature Σ introduced in Example 1, we can define the following set δ of

$$\begin{aligned}
(i) \quad (\text{ask}(s, o, a), \text{permit}) &: \begin{cases} m(s, o, a) \leftarrow \\ \neg \text{redlist}(s) \leftarrow \\ \neg \text{blacklist}(s) \leftarrow \end{cases} \\
(ii) \quad (\text{ask}(s, o, a), \text{deny}) &: \begin{cases} \text{blacklist}(s) \leftarrow \text{redlist}(s) \\ \neg m(s, o', a') \leftarrow \text{blacklist}(s) \wedge m(s, o', a') \\ \text{redlist}(s) \leftarrow \end{cases} \\
(iii) \quad (\text{release}(s, o, a), \text{permit}) &: \{ \neg m(s, o, a) \leftarrow \}
\end{aligned}$$

The rules (i) and (iii) indicate that when an access request is permitted, the corresponding fact is added and when the respective access is released, the fact is removed. The rule (ii) expresses that a subject is registered on a red-list when it attempts a denied access and that it is black-listed when it attempts twice in a row. When it is black-listed, all its current accesses are removed.

The expressive power of the formalism goes well beyond the updates of Example 3. For instance, if we consider a predicate *delegate*, the following rule

⁴ For any substitution σ , $\sigma(U)$ is the update sequence obtained by replacing in the goal and body of all its updates any $x \in \text{Dom}(\sigma)$ by $\sigma(x)$.

indicates how user s can give his rights to user s' ; if s is *root* then s' becomes a *sudoer* unless it had been red- or black-listed:

$$(delegate(s, s'), permit) : \begin{cases} f_s(s') = l \leftarrow l = f_s(s) \\ sudo(s') \leftarrow s = root \wedge \neg redlist(s') \wedge \neg blacklist(s') \end{cases}$$

Provided the rules events are not unifiable, we get:

Proposition 2. *For any set δ of disjoint transition rules (i.e such that rules events do not overlap), the relation \rightarrow_δ is deterministic.*

3.3 Security systems, policy rules and secured systems

A security system is defined by a set of transition rules and an initial environment.

Definition 5. *Given a security signature $\Sigma = \Sigma_{Sys} \cup \Sigma_{Ev}$, a **security system**, is the labelled transition system \mathfrak{S} whose states are environments over Σ_{Sys} , with an initial state η_{init} and whose transitions are $\eta \xrightarrow{evt}_\delta \eta'$ for some $evt \in \mathcal{T}_{\Sigma, |\eta|}^{Query} \times \mathcal{T}_{\Sigma, |\eta|}^{Decision}$.*

Example 4. The security system \mathfrak{S} over the security signature Σ defined in Example 1 consists of the initial environment η defined in Example 2 and the set δ of transition rules defined in Example 3.

Definition 6. *A set of **policy rules** over a security signature $\Sigma = \Sigma_{Sys} \cup \Sigma_{Ev}$ is an ordered constrained term rewrite system \mathfrak{R} over Σ with all the rules of the form $l \xrightarrow{\varphi} r$ with l, r terms sorted by sorts in \mathcal{S}_{Ev} and φ a Σ_{Sys} -formula.*

We write $q \xrightarrow{\eta}_{\mathfrak{R}} d$ when q is rewritten in one step w.r.t. the policy rules \mathfrak{R} and the environment η into d and we write $q \xrightarrow{*}_{\mathfrak{R}}^\eta d$ for multiple-step rewriting.

Example 5. The following ordered CTRS

$$\left\{ \begin{array}{l} ask(s, o, a) \xrightarrow{blacklist(s)} deny \\ ask(s, o, a) \xrightarrow{sudo(s) \wedge s \neq root} ask(root, o, a) \\ ask(s, o, read) \xrightarrow{f_o(o) \preceq f_s(s) \wedge (\forall o'. m(s, o', write) \Rightarrow f_o(o) \preceq f_o(o'))} permit \\ ask(s, o, write) \xrightarrow{\forall o'. m(s, o', read) \Rightarrow f_o(o') \preceq f_o(o)} permit \\ ask(s, o, erase) \xrightarrow{f_o(o) \preceq f_s(s)} permit \\ ask(s, o, a) \rightarrow deny \\ release(s, o, a) \rightarrow permit \end{array} \right\}$$

defines a policy specifying that:

- a black-listed subject is denied any access;
- a *sudoer* has the same access rights as *root*;

- a subject can read an object whose level of security is smaller than its level of security if it is not writing an object of a lower security level;
- a subject can write an object if it is not reading an object of a higher level;
- a subject can erase an object whose security level is smaller than its level;
- a subject can release any of its accesses;
- in all other cases, the request is denied.

Notice that since the rules are ordered, the constraints do not need to impose explicitly the negation of the constraints of previous overlapping rules and, in particular, no constraint is needed for the “default” rules.

Definition 7. A set of policy rules \mathfrak{R} over a security signature $\Sigma = \Sigma_{Sys} \cup \Sigma_{Ev}$ is η -consistent (resp. η -complete) for an environment η over Σ_{Sys} iff for any query $q \in \mathcal{T}_{\Sigma, |\eta|}^{Query}$, there exists at most (resp. at least) one decision $d \in \mathcal{T}_{\Sigma, |\eta|}^{Decision}$ such that $q \xrightarrow{\mathfrak{R}}^{\eta} d$.

These properties can be proved for a large class of policy rules.

Proposition 3. A set \mathfrak{R} of policy rules is η -consistent if (1) for each rule, its left-hand side contains only one occurrence of each variable and its constraint does not involve terms of sort *Query*; (2) for any two rules $l \xrightarrow{\varphi} r$ and $l' \xrightarrow{\varphi'} r'$ there exists no position ω and no substitution σ such that $\llbracket \eta \rrbracket \models \{\sigma(l|_{\omega}) = \sigma(l') \wedge \sigma(\varphi) \wedge \sigma(\varphi')\}$.

Proof. The proof is obtained by adapting the proof for orthogonal TRS [2].

Proposition 4. A set \mathfrak{R} of policy rules over a security signature $\Sigma = \Sigma_{Sys} \cup \Sigma_{Ev}$ is η -complete if (1) the reduction $\xrightarrow{\mathfrak{R}}^{\eta}$ terminates, (2) for any symbol $f: \mathfrak{s}_1, \dots, \mathfrak{s}_n \mapsto \mathfrak{s} \in \mathcal{F}_{Ev}$ with $\mathfrak{s} > Decision$ there exists a default rule for f , i.e. a rule containing no constraint and whose left-hand side is a term $f(x_1, \dots, x_n)$ with $x_i \in \mathcal{X}$, (3) each rule of \mathfrak{R} is sort-preserving or sort-decreasing (i.e. the sort of its left-hand side is equal or greater than the sort of its right-hand side).

Proof. We prove that $q \in \mathcal{T}_{\Sigma, |\eta|}^{Query}$ has a normal form of sort *Decision* by induction on the well-founded rewrite relation induced by \mathfrak{R} . If q is of sort *Decision* then its normal form is necessarily of the same sort by (3). If not, then q is reducible by \mathfrak{R} (because of the default rule) into a ground term q' which, by induction hypothesis, has a normal form of sort *Decision*.

The classical methods for proving termination of TRS can be adapted for CTRS. For example, the policy rules introduced in Example 5 can be shown terminating using an obvious polynomial interpretation [2] connected to the corresponding constraints. There is also a default rule for each symbol of sort

Query and the rules are sort-preserving or sort-decreasing. Consequently, the corresponding normal forms are clearly in this case *permit* or *deny*. The policy rules obviously satisfy condition (1) of Proposition 3 and, because of the order, condition (2) as well. The policy rules of Example 5 are thus η -complete and η -consistent for any environment η .

Definition 8. Given a security signature $\Sigma = \Sigma_{Sys} \cup \Sigma_{Ev}$, a security system \mathfrak{S} over Σ and a set \mathfrak{R} of policy rules over Σ η -complete and η -consistent for any environment η over Σ_{Sys} , a **secured system** is the labelled transition system \wp whose states are environments over Σ_{Sys} , whose initial state is the initial state of \mathfrak{S} and whose transitions are $\eta \xrightarrow{\langle q, d \rangle}_{\wp} \eta'$ for some $\langle q, d \rangle \in \mathcal{T}_{\Sigma, |\eta|}^{Query} \times \mathcal{T}_{\Sigma, |\eta|}^{Decision}$ such that $\eta \xrightarrow{\langle q, d \rangle}_{\delta} \eta'$ and $q \xrightarrow{*}_{\mathfrak{R}} d$.

Since the set of security rules of a secured system is η -complete and η -consistent for any environment η , the corresponding relation is computable. Moreover, the underlying relation of the security system is computable by construction and thus, so is the underlying relation of the corresponding secured system.

Proposition 5. The relation \rightarrow_{\wp} is computable for any secured system \wp .

4 Checking security properties

In this section we propose a methodology based on environment transformations for the validation of security properties enforced by a policy over a system. This is particularly relevant when the security property is expressed on some entities like, for instance, when considering information flow properties, while the policy is an access control policy that can only manipulate objects containing the information to be traced. To solve this problem, we introduce a transformation whose aim is to translate an environment into another one dealing with the entities we are interested in. Such a transformation is defined below by a signature morphism that allows the translation of the domain and of the base of equalities and by a set of transformation rules on the base of facts.

Definition 9. A **signature morphism** θ from $(\mathcal{S}_1, \mathcal{F}_1, \mathcal{P}_1)$ to $(\mathcal{S}_2, \mathcal{F}_2, \mathcal{P}_2)$ is a pair $(\theta_{\mathcal{S}}, \theta_{\mathcal{F}})$ such that $\theta_{\mathcal{S}}: \mathcal{S}_1 \rightarrow \mathcal{S}_2$ and $\theta_{\mathcal{F}}: \mathcal{F}_1 \rightarrow \mathcal{F}_2$ are (partial or total) functions such that $\forall f: \mathfrak{s}_1, \dots, \mathfrak{s}_n \mapsto \mathfrak{s} \in \text{Dom}(\theta_{\mathcal{F}})$ where $\mathfrak{s}_1, \dots, \mathfrak{s}_n, \mathfrak{s} \in \text{Dom}(\theta_{\mathcal{S}})$, $\theta_{\mathcal{F}}(f): \theta_{\mathcal{S}}(\mathfrak{s}_1), \dots, \theta_{\mathcal{S}}(\mathfrak{s}_n) \mapsto \theta_{\mathcal{S}}(\mathfrak{s}) \in \mathcal{F}_2$. We extend θ to a morphism $\hat{\theta}$ (which is simply denoted by θ) over terms as follows:

- $\forall x: \mathfrak{s} \in \mathcal{X}, \hat{\theta}(x: \mathfrak{s}) = x: \theta_{\mathcal{S}}(\mathfrak{s})$
- $\forall f \in \text{Dom}(\theta_{\mathcal{F}}), \hat{\theta}(f(t_1, \dots, t_n): \mathfrak{s}) = \theta_{\mathcal{F}}(f)(\hat{\theta}(t_1), \dots, \hat{\theta}(t_n))$

Definition 10. Given two signatures $\Sigma_1 = (\mathcal{S}_1, \mathcal{F}_1, \mathcal{P}_1)$ and $\Sigma_2 = (\mathcal{S}_2, \mathcal{F}_2, \mathcal{P}_2)$, an **environment transformation** Θ is a tuple $(\theta, \delta, \mathcal{R})$ where:

- θ is a signature morphism from Σ_1 to Σ_2 ;
- δ is a set of pairs $\langle \text{condition}, \text{conclusion} \rangle$ with condition a Σ_1 -formula and conclusion a conjunction of Σ_2 -atoms such that $x:\mathfrak{s} \in \mathcal{FVar}(\text{condition})$ iff $\theta(x:\mathfrak{s}) \in \mathcal{FVar}(\text{conclusion})$; thus, variables of $\mathcal{FVar}(\text{conclusion})$ (resp. $\mathcal{FVar}(\text{condition})$) are sorted by $\text{Dom}(\theta_S)$ (resp. $\text{Im}(\theta_S)$);
- \mathcal{R} is a set of safe and stratified logical rules over Σ_2 .

Applying Θ on an environment η over Σ_1 produces an environment $\Theta(\eta)$ over Σ_2 defined as follows:

- $|\Theta(\eta)| = \{c:\theta(\mathfrak{s}) \mid c:\mathfrak{s} \in |\eta| \wedge \mathfrak{s} \in \text{Dom}(\theta)\}$;
- $\mathcal{E}_{\Theta(\eta)}$ contains an equality $\theta(f(t_1, \dots, t_n)) = \theta(t)$ for each $f(t_1, \dots, t_n) = t$ in \mathcal{E}_η whose image by θ is defined;
- $\mathcal{B}_{\Theta(\eta)}$ contains all the Σ_2 -atoms $p(\mu(t_1), \dots, \mu(t_m)) \downarrow_{\Theta(\eta)}$ for which there exists a pair $\langle \text{condition}, \text{conclusion} \rangle \in \delta$ where $p(t_1, \dots, t_n)$ occurs in conclusion, and a mapping μ from $\text{Var}(\text{condition})$ to $|\eta|$ such that $\llbracket \eta \rrbracket \models \mu(\text{condition})$;
- $\mathcal{R}_{\Theta(\eta)} = \mathcal{R}$.

We say that η is transformed by Θ into $\Theta(\eta)$.

Any environment transformation $\Theta = (\theta, \delta, \mathcal{R})$ from Σ_1 to Σ_2 induces a total mapping $\eta \mapsto \Theta(\eta)$ from Σ_1 -environments into Σ_2 -environments; this operational view justifies to call Θ a *transformation operator*. We can use this notion of environment transformation to check a security property expressed as a Σ_2 -formula ψ over reachable environments of a secured system \wp over Σ_1 . Indeed, this amounts to check that for every reachable environment η of \wp , we have $\llbracket \Theta(\eta) \rrbracket \models \psi$, which is decidable for any ψ , as a consequence of Proposition 1 and the computability of $\Theta(\eta)$ for any η .

Example 6. We consider now environment transformations that can be used to deal with information flow properties of access control policies. We introduce the “generic” signature $\Sigma_{FLOW} = (\{\text{Actor}, \text{Information}\}, \mathcal{F}_{FLOW}, \mathcal{P}_{FLOW})$ where \mathcal{P}_{FLOW} is:

$$\left\{ \begin{array}{ll} \text{Get} : \text{Actor}, \text{Information}; & \text{MoveTo} : \text{Information}, \text{Information}; \\ \text{Put} : \text{Actor}, \text{Information}; & \text{Trustworthy} : \text{Actor}, \text{Information}; \\ \text{Eligible} : \text{Actor}, \text{Information}; & \text{Gflow} : \text{Information}, \text{Information} \end{array} \right\}$$

and where \mathcal{F}_{FLOW} is an arbitrary set of function symbols. $\text{Get}(a, i)$ means that the actor a knows the information i , $\text{Put}(a, i)$ means that the actor a modifies the information i (by using the information it knows), $\text{MoveTo}(i_1, i_2)$ means that the information i_2 is enriched with information i_1 , $\text{Eligible}(a, i)$ means that the actor a is granted to know the information i , $\text{Trustworthy}(a, i)$ means that the actor a is granted to modify the information i and $\text{Gflow}(i_1, i_2)$ means that

the information i_1 is authorized to flow into i_2 . The predicates *Get*, *Put* and *MoveTo* are useful for describing existing flows while the predicates *Eligible*, *Trustworthy*, and *Gflow* are used to specify flow policies (respectively a confidentiality policy, an integrity policy and a confinement policy). Now, it is possible to define, in a generic way, confidentiality, integrity and confinement security properties as follows:

$$\begin{aligned} \text{Confidentiality } \psi_{conf} & \forall a, i. \text{Get}(a, i) \Rightarrow \text{Eligible}(a, i) \\ \text{Integrity } \psi_{int} & \forall a, i. \text{Put}(a, i) \Rightarrow \text{Trustworthy}(a, i) \\ \text{Confinment } \psi_{info} & \forall i, i'. \text{MoveTo}(i, i') \Rightarrow \text{Gflow}(i, i') \end{aligned}$$

Let us consider the environment transformation defined from the signature Σ_{Sys} of Example 1 and the signature Σ_{FLOW} and consisting of the partial function $\theta_S : \mathcal{S}_{Sys} \rightarrow \mathcal{S}_{FLOW}$ such that $\text{Dom}(\theta_S) = \{S, O\}$ with $\theta_S(S) = \text{Actor}$ and $\theta_S(O) = \text{Information}$ together with the identity function $\theta_{\mathcal{F}}$, the following logical rules over Σ_{FLOW}

$$\mathcal{R}_{FLOW} = \left\{ \begin{array}{l} \text{MoveTo}(i, i) \leftarrow \\ \text{MoveTo}(i, i') \leftarrow \text{Get}(a, i) \wedge \text{Put}(a, i') \\ \text{Get}(a, i) \leftarrow \text{MoveTo}(i, i') \wedge \text{Get}(a, i') \\ \text{Put}(a, i') \leftarrow \text{MoveTo}(i, i') \wedge \text{Put}(a, i) \\ \text{MoveTo}(i, i'') \leftarrow \text{MoveTo}(i, i') \wedge \text{MoveTo}(i', i'') \end{array} \right.$$

and δ defined by

$$\frac{m(x, y, \text{read}) \wedge \forall x'. \neg m(x', y, \text{erase})}{\text{Get}(x, y)} \quad \frac{f_o(y) \preceq f_s(x)}{\text{Eligible}(x, y)}$$

$$\frac{m(x, y, \text{write}) \wedge \forall x'. \neg m(x', y, \text{erase})}{\text{Put}(x, y)} \quad \frac{f_o(y) \preceq f_o(y')}{\text{Gflow}(y, y')}$$

The rules introducing *Get* and *Put* allow the translation of the accesses expressed in the source environment using the predicate m into accesses expressed in the target environment using the predicates *Get* and *Put*. The rules introducing *Eligible* and *Gflow* can be viewed as the definition of the flow interpretation of a (security level-based) access control policy.

Note that this transformation allows to handle transitive information flows generated by accesses performed simultaneously in a given environment but does not take into account the past (accesses) of the system. Indirect flows can be nevertheless dealt with, by adding a new predicate which keeps track, in the source environments, of the origins of the information contained into each object of the system (for example, see [14, 19] for an intrusion detection based approach dealing with indirect information flows).

The above environment transformation provides the means for checking that our policy ensures confinement. This can be done by checking that each reachable environment η of the secured system \wp is such that $\llbracket \Theta(\eta) \rrbracket \models \psi_{info}$. However, the existence of *sudoers* may generate reachable environments that do not satisfy

the confidentiality property *w.r.t.* Θ , *i.e.* it is possible to obtain a reachable environment in \wp which is transformed into an environment which does not satisfy ψ_{conf} . This is for example the case for the environment obtained by considering the initial environment introduced in Example 2 and a transition labelled by the event $ask(Charlie, PwdFile, read)$. Indeed, since *Charlie* is a *sudoer*, the policy defined in Example 5 allows *Charlie* to have a read access over *PwdFile* even if its security level is not greater than *Secret* and hence, we have $Get(Charlie, PwdFile)$. However, the meaning of *Eligible* specified by δ does not take into account *sudoers* and we have thus $\neg Eligible(Charlie, PwdFile)$.

Of course, when adding the rule $\frac{sudo(s)}{Eligible(s, o)}$ to δ and give thus a different semantics to the confidentiality property, one can check that any reachable environment η of the system \wp is such that $\llbracket \Theta(\eta) \rrbracket \models \psi_{info} \wedge \psi_{conf}$. Note also that if we add the possibility for a user to assign its security level to another user (as illustrated on page 8) then the security level of a user can decrease and thus, only the property ψ_{info} is preserved. By constraining the delegation mechanism with the following policy rules:

$$\left\{ \begin{array}{l} delegate(s, s') \xrightarrow{f_s(s') \preceq f_s(s)} permit \\ delegate(s, s') \xrightarrow{\neg(f_s(s') \preceq f_s(s))} deny \end{array} \right\}$$

to ensure that security levels of subjects can only increase, the property ψ_{conf} can be preserved.

The transformation approach can be also useful when one wants to enforce policies by directly using the desired security properties to constrain the transitions of a security system. Indeed, suppose we want to constrain a security system $\mathfrak{S} = (\eta_{init}, \delta)$ over a signature Σ_1 in order to ensure a security property expressed as a formula φ over a different signature Σ_2 . The corresponding secured system can be obtained by using an environment transformation $\Theta = (\theta, \delta_\Theta, \mathcal{R}_2)$ from Σ_1 to Σ_2 and by considering the transition relation $\rightarrow_{\delta_\varphi}$ such that $\eta \xrightarrow{evt}_{\delta_\varphi} \eta'$ iff $\eta \xrightarrow{evt}_\delta \eta' \wedge \llbracket \Theta(\eta') \rrbracket \models \varphi$. Of course, such an approach leads to a system whose reachable states satisfy φ iff $\llbracket \Theta(\eta_{init}) \rrbracket \models \varphi$. The notion of environment transformation makes thus possible the application of a security policy expressed as a (required) property to several systems.

5 Related work

Among a rich literature on security policies (see for instance [11] for policy specification languages, and [18] for the rule-based and property-based approaches to define policies), our approach is in the line of logic-based languages providing a well-understood formalism, which is amenable to analysis and verification.

Our formalism borrows inspiration from various sources. Horn clause logic has been used extensively for RBAC models [22]. Since negation and recursion are often needed, the concept of stratified theories has been used for instance in the authorization specification language ASL [17] for access control. Integrity rules specify application dependent conditions that limit the range of acceptable policies. Stratified logic for RBAC policies is also developed in [3]. In our work, we use similar concepts but do not restrict to RBAC models.

Constraint logic programming for designing RBAC and temporal RBAC policies is considered in [4]. Their constraints are conjunctions of equational constraints over sets of constants, and arithmetic constraints over non-negative integers. While keeping a declarative approach, CLP adds the expressive power and efficiency of constraint solving and database querying. A security administrator has then analysis capability thanks to the computation of sets of constraints as answers. Formalisation of security analysis in an abstract setting is done in [21] and exemplified for RBAC. In comparison, we allow a different class of constraints that we keep decidable by restricting to safe theories, and we use constraints in a rewriting context. Note that it is also possible to apply constraint narrowing to get analysis power as in [20].

Whereas most existing works on reasoning about security policies model the environment only lightly, if at all, there are some exceptions. One of the closest works is [12] who represents the behavior of access control policies in a dynamic environment. Policies are written in Datalog and can refer to facts in the authorization state. Events, such as access requests, can change the authorization state, and the changes are specified as a state machine whose transition labels are guarded by the policy. Security properties can then be analyzed by model checking formulas in first-order temporal logic. In [5], the authors introduce a logic for specifying policies where access requests can have effects on the authorization state. The effects are explicitly specified in the language, an extension of Datalog backed on transaction logic. They also propose a proof system for reasoning about sequences of user actions. In comparison, thanks to constraint rewriting, we provide a more expressive formalism, while keeping it operational and decidable. The full expressive power of constraint rewriting is explored in [8].

Comparing the expressive power of access control models is a fundamental problem in computer security, already addressed in several works. In [7], different access control models are represented in C-Datalog (an object-oriented extension of Datalog) and compared using results from logic programming. In [23], the authors express access control systems as state transitions systems as we do and introduce security-preserving mappings, called reductions, to compare security analysis based on accessibility relations in two different models. In [9, 15], the comparison mechanism is based on a notion of simulation. Thanks to the no-

tion of environment transformation, we address this problem with an operational transition rules based approach.

6 Conclusion and future work

We proposed a framework which provides a common formalism for defining security signatures, environments, systems, and policy rules. We have shown that secured systems specified in this formalism have an operational semantics based on transition and rewriting systems and are thus executable.

Such a formalism can be viewed as the definition of a mechanism allowing to constrain the executions of a transition system by applying a policy. As shown by our examples, this can be useful in the context of security, but such a mechanism can also be considered when supervising a system for other purposes. Our framework also allows the definition of transformations of security signatures and environments and consequently, of secured systems. We defined a transformation operator and showed how it can be used to check security properties over the reachable environments of a secured system. This approach based on a transformation operator allows us to check some properties over a system even if these properties are expressed on a different signature (and/or specification). Our framework facilitates thus the reusability since the same specification of a security property can be used to check several policies and systems. It encourages also the specification of generic security properties dedicated to particular domains like, for example, information flows, and that can be used in different contexts. The operational nature of our formalism should lead to relatively direct implementations using rewrite based tools like *Maude* [10] for example. The rule engine of *Maude* can then be used to simulate secured systems and their transformations and the integrated model-checker could be used to verify the corresponding properties.

As future work, we aim to focus on the extension of the proposed transformation in order to define policies and systems in a completely independent way and to provide thus an enhanced modularity in formal developments. We also want to study how the transformation operators could be used for comparing and composing security policies and systems. Indeed, the comparison between two policies expressed as policy rules \mathfrak{R}_1 and \mathfrak{R}_2 , respectively based on the signatures Σ_1 and Σ_2 , is often based on an embedding of Σ_1 -formulas into Σ_2 -formulas. Such an approach can also be considered for systems, using transformations between environments to define a comparison mechanism. Similarly, for composition, transformation operators could be used to translate policies and systems into policies and systems sharing the same security signature and specification, thus easing the definition of a composition relation.

References

1. Alferes, J., Pereira, L., Przymusinska, H., Przymusinski, T.: LUPS – A language for updating logic programs. *Logic Programming and Nonmonotonic Reasoning* pp. 162–176 (1999)
2. Baader, F., Nipkow, T.: *Term Rewriting and All That*. Cambridge University Press (1998)
3. Barker, S.: Access control for deductive databases by logic programming. In: 18th International Conference of Logic Programming. LNCS, vol. 2401, pp. 54–69. Springer (2002)
4. Barker, S., Stuckey, P.J.: Flexible access control policy specification with constraint logic programming. *ACM Transactions on Information and System Security* 6(4), 501–546 (2003)
5. Becker, M.Y., Nanz, S.: A logic for state-modifying authorization policies. In: ESORICS 2007. LNCS, vol. 4734, pp. 203–218. Springer (2007)
6. Bell, D., LaPadula, L.: *Secure Computer Systems: a Mathematical Model*. Tech. Rep. MTR-2547 (Vol. II), MITRE Corp., Bedford, MA (May 1973)
7. Bertino, E., et al.: A logical framework for reasoning about access control model. In: *ACM Transactions on Information and System Security*. vol. 6, pp. 71–127 (2003)
8. Bourdier, T., Cirstea, H.: Constrained rewriting in recognizable theories. Tech. rep., INRIA (2010), <http://hal.archives-ouvertes.fr/inria-00456848/en/>
9. Chander, A., Mitchell, J., Dean, D.: A state-transition model of trust management and access control. In: *Proceedings of the 14th IEEE Computer Security Foundation Workshop CSFW*. pp. 27–43. IEEE Comp. Society Press (2001)
10. Clavel, M., Durán, F., Eker, S., Lincoln, P., Martí-Oliet, N., Meseguer, J., Talcott, C.: *All About Maude - A High-Performance Logical Framework, How to Specify, Program and Verify Systems in Rewriting Logic*, vol. 4350. Springer-Verlag (2007)
11. Damianou, N., et al.: A survey of policy specification approaches. Tech. rep., Department of Computing, Imperial College of Science Technology and Medicine, London, UK (2002), <http://www.doc.ic.ac.uk/~mss/Papers/PolicySurvey.pdf>
12. Dougherty, D.J., Fisler, K., Krishnamurthi, S.: Specifying and reasoning about dynamic access-control policies. In: 3rd International Joint Conference on Automated Reasoning (IJ-CAR). LNCS, vol. 4130, pp. 632–646 (2006)
13. Gelfond, M., Lifschitz, V.: Action languages. *Electron. Trans. Artif. Intell.* 2, 193–210 (1998)
14. George, L., Tong, V.V.T., Mé, L.: Blare tools: A policy-based intrusion detection system automatically set by the security policy. In: *Recent Advances in Intrusion Detection, 12th International Symposium, RAID. Proceedings. Lecture Notes in Computer Science*, vol. 5758, pp. 355–356. Springer (2009)
15. Habib, L., Jaume, M., Morisset, C.: Formal definition and comparison of access control models. *Journal of Information Assurance and Security* 4(4), 372–381 (2009)
16. Hinman, P.: *Fundamentals of mathematical logic*. A.K. Peters, Ltd. (2005)
17. Jajodia, S., et al.: A unified framework for enforcing multiple access control policies. In: *ACM SIGMOD international conference on Management of data*. pp. 474–485. ACM (1997)
18. Jaume, M.: Security rules versus security properties. In: Jha, S., Mathuria, A. (eds.) *Information Systems Security - 6th International Conference, ICISS 2010, Proceedings. Lecture Notes in Computer Science*, vol. 6503, pp. 231–245. Springer (2010)
19. Jaume, M., Tong, V.V.T., Mé, L.: Contrôle d'accès versus contrôle de flots. In: *Approches Formelles dans l'Assistance au Développement de Logiciels, AFADL2010*. pp. 27–41 (2010)
20. Kirchner, C., Kirchner, H., Santana de Oliveira, A.: Analysis of rewrite-based access control policies. In: *Proceedings of the Third International Workshop on Security and Rewriting Techniques (SecReT 2008)*. ENTCS, vol. 234, pp. 55–75. Elsevier (2009)
21. Li, N., Tripunitara, M.V.: Security analysis in role-based access control. *ACM Transactions on Information and System Security (TISSEC)* 9(4), 391–420 (2006)

22. Sandhu, R.S., Coyne, E.J., Feinstein, H.L., Youman, C.E.: Role-based access control models. *Computer* 29, 38–47 (1996)
23. Tripunitara, M.V., Li, N.: A theory for comparing the expressive power of access control models. *Journal of Computer Security* 15(2), 231–272 (2007)
24. Ullman, J.: *Database and Knowledge - Base Systems*, vol. 1: *Classical Database Systems*. Computer Science Press (1988)