Dealing with Multi-Policy Security in Large Open Distributed Systems

Christophe Bidan[1] and Valérie Issarny[2]

[1] Distributed Software Engineering Section,
Department of Computing,
Imperial College, London SW7 2BZ, UK,
c.bidan@doc.ic.ac.uk
[2] IRISA / INRIA Rennes,
Campus Universitaire de Beaulieu,
35042 Rennes Cedex, France,
Valerie.Issarny@irisa.fr

Abstract. From the security point of view, one challenge for today's distributed architectures is to support interoperation between applications relying on different possibly inconsistent security policies.
This paper proposes a practical solution for dealing with the coexistence of different security policies in distributed architectures. We introduce a model for specifying security policies in terms of security domains, access control and information flow rules. Then, we identify the set of operators for combining the specifications of sub-policies and we address the validity of the resulting policy according to the security properties of the sub-policies.

1 Introduction

Object-based distributed computing architectures[1] like CORBA (Common Object Request Broker Architecture) defined by the Object Management Group (OMG) [24], and the Telecommunication Intelligent Network Architecture proposed by the TINA consortium [26] are promising approaches to support large open distributed systems. Goals of these architectures include interoperability between software components in heterogeneous distributed environments, where components may appear and disappear dynamically, as the result of individual and autonomous actions. From the security point of view, the interoperability promoted by the above architectures stresses the complexity of security policies which have to be implemented [15, 10], as well as the necessity to cope with the coexistence of multiple security policies. This coexistence results from the interoperation between systems (or software components) having different security requirements, possibly at different granularity levels.

For evaluating the security of open distributed systems, *security officers* should be able to reason about the *composition* of the interoperating systems'

[1] From the security point of view, an object-based distributed computing architecture is viewed as multiple interconnected systems [23].

security policies. We identify two different approaches for the composition of security policies: *policy interoperation* and *policy combination*. Using policy interoperation, the composed policy should not violate the security of the sub-policies, and should guarantee their autonomy [14]. On the other hand, with policy combination, the composed policy may be inconsistent with both sub-policies, but must be secure in the given context. Hence, security combination allows to compose conflicting policies in a secure and controlled manner [3]. In the context of large open distributed architectures, we assert that policy combination is better suited than policy interoperation. In this paper, we introduce a model that forms the basis for dealing with security in heterogeneous distributed architectures, allowing to specify and to combine a wide variety of security policies. In particular, the proposed model is useful to security officers by supplying a practical set of combination operators.

This paper is structured as follows: the next section presents related work and gives our standpoint concerning policy composition. Section 3 presents our model for the specification and combination of *access control policies*[2]. In particular, it gives the notion of *complete* and *sound* combination. Section 4 extends this model so as to deal with combination of information flow policies. Finally, we draw some conclusions in section 5 and present current status and future work.

2 Topics for Policy Combination

Various models have been proposed in order to reason about security policies (e.g., see an overview of security models by Landwehr [18], the Bell-LaPadula model [4], the information flow model from Bieber [8], the Clark-Wilson model for commercial security constraints [9], and the McLean model [20]). These models are introduced to check whether a policy verifies given security properties like the *nondeducibility property* [25] or the *noninterference property* [13]. However, these models do not deal with the composition of policies.

Concerning the definition of a security policy resulting from the composition of policies, we identify two different approaches:

- *policy interoperation* which infers the composed policy based on the security properties of the sub-policies, and
- *policy combination* which specifies the composed policy based on the specifications of the sub-policies.

The former approach relies on the principles of *autonomy* and of *security* of sub-policies [14]: during the interoperation of two different security policies, any access authorized within an individual policy must also be authorized within the composed policy; and dually, any access denied within an individual policy must also be denied within the composed policy. Abadi et al. [1] propose a general

[2] In this paper, an *access control policy* is defined as a security policy that only checks the accesses between entities, without verifying the information flow resulting from access operations.

solution to the inference of execution properties of interoperating components from the components' specification, which can be adapted to security policies. More recently, McLean [21] has presented an approach which allows to reason about security properties of composed information flow policies. Finally, Gong et al. discuss computational issues concerning the general secure interoperation problem [14].

The above proposals allow to verify that interoperation preserves the sub-policies' security properties under certain conditions. However, the undertaken approach to the definition of policy composition is restrictive; the security properties of the composed policy must be the same as those of the sub-policies (the principles of autonomy and security must be guaranteed). In consequence, the sub-policies have to be compatible. In large open distributed system, this is not always the case, since the properties of a policy resulting from the composition of two incompatible sub-policies can be an extension or a restriction of those sub-policies. We illustrate this through two examples.

In both example, we consider a file system where we have two sets of users, A and C, and two sets of files, B and D. We say that an user u is authorized to access the file f if and only if u is authorized to access f's data. We specify only authorized accesses, and all accesses that are not specifically authorized, are denied. We define the security policy $\mathcal{P}_1$ (resp. $\mathcal{P}_2$) managing A users and B files (resp. C users and D files): $\mathcal{P}_1$ (resp. $\mathcal{P}_2$) authorizes users in A (resp. in C) to access files in B (resp. in D).

First, let us combine $\mathcal{P}_1$ with $\mathcal{P}_2$ to build the following composed policy: the users belonging to both sets A and C (denoted as $A \cap C$) are authorized to access the files in $B \cap D$, users in A but not in C (denoted as $A - C$) are authorized to access the files in $B - D$, and finally, users in $C - A$ are authorized to access the files in $D - B$.

As a second example, we combine $\mathcal{P}_1$ and $\mathcal{P}_2$ so as to obtain the following policy: all the users belonging to sets A or C (denoted as $A \cup C$) are authorized to access all the files in $B \cup D$. Notice that this policy may be viewed as the federation of the sub-policies.

The composed policy of the first example is a restriction of the participating sub-policies, whereas the composed policy in the second example is an extension of the participating sub-policies (for instance, A users are authorized to access D files). However, these two composed policies are secure in the context in which they are specified, nevertheless they cannot be obtained by policy interoperation.

The specification of a policy based on the specifications of its sub-policies allows to combine possibly inconsistent policies. In particular, a combined policy does not necessarily guarantee all the properties of the sub-policies. However, notice that the combined policy may keep the principles of autonomy and secrecy of the sub-policies.

McLean [19] seems to be the first to propose a formal approach including combination operators: he introduces an algebra of security which enables to reason about the problem of policy conflict. However, even though this approach permits to detect conflicts between policies, it does not propose a method to resolve the conflicts and to construct a security policy from inconsistent sub-policies. Hosmer [16] has introduced the notion of metapolicies, or "policies about policies", an informal framework for combining security policies. Following Hosmer's work, Bell formalizes the combination of two sub-policies with a function (*policy combiner*), and introduces the notion of *policy attenuation* to allow the composition of conflicting security policies [3].

Our work is placed in the framework of policy combination, and it is close to Bell's approach. More specifically, our model is a practical alternative solution to Bell's proposal: we clearly identify the set of combination operators enabling security officers to combine security policies in a controlled and secure way. Briefly, our model allows to specify the general behavior of sub-policies. Based on these specifications, we introduce combination operators for combining policy specifications. Finally, we verify that the behavior of the composed policy is secure and conforms with sub-policies specifications. The following sections introduce our model to deal with access control policies and flow policies.

3 Specifying Access Control Policies

In this section, we concentrate on the following type of access control: is an entity ϵ_1 authorized to access an entity ϵ_2 in the context of a given policy, regardless of the information flow that is implied by this access ?

Our solution to policy composition relies on the definition of a set of combination operators. A policy is specified either as an *elementary policy* or a *combined policy* built from existing sub-policies through the use of combination operators. The next sub-section addresses the specification of access control policies and is followed by the definition of the operators allowing to combine policy specifications. Finally, we address the completeness and soundness of policy specifications.

3.1 Elementary access control policy

An access control policy defines a set of rules that specify for each pair of object and subject, whether the subject is allowed to access the object's state or not[3]. Since in an open distributed architecture, an access control policy cannot be defined for the whole set of the system's entities, we take an approach which allows to specify the sensitive entities for each policy. In other words, an access control policy is defined not solely in terms of its *access control rules* but also in terms of its *access control domain* [22], that determines the system's objects and subjects to which the policy applies.

The access control domain of a policy is subdivided into the set of the system's entities called *object domain*, that contains sensitive information for this policy,

[3] We define the *subjects* as active entities, and the *objects* as passive or active entities [11].

and the set of the system's entities called *subject domain*, that gives the subjects belonging to the policy domain. These two sets are defined formally in terms of security classifications[4]. Given a security property P (i.e. a predicate) defined on the set $\mathcal{E}$ of the system's entities, the set of entities verifying this property is termed the P *classification*, noted Cl_P:

$Cl_P = \{\varepsilon \in \mathcal{E} \mid P(\varepsilon) \equiv true\} = \{\varepsilon \in \mathcal{E} \mid P\}$

Given the definition of classification, an access control domain is defined as a set of classifications:

$Dom = \{Cl_i\}_{i=1}^{n}$,

where, for each $i = 1, ..., n$, Cl_i is the classification defined by a security property P_i.

Each access control rule λ of a given policy defines the set of subject and object couples that verify a given security property P_λ (called *access control predicate*), and an *access control operator*. The access control operator is noted $\sim\!\!>$ for authorization, $\not\sim\!\!>$ for denial, and the $\div\!\!>$ operator will be used to denote $\sim\!\!>$ or $\not\sim\!\!>$[5]. Formally, an access control rule λ takes the form:

$\lambda = \{(s \in Cl_1), (o \in Cl_2) \mid P_\lambda(s, o) \ ; s \div\!\!> o\}$.

In other words, we say that a pair (s, o) belongs to λ if the access control predicate $P_\lambda(s, o)$ holds ; then, the access control operator of λ define whether s is authorized or not to access o.

In the remainder, for $\mathcal{P}$ being an access control policy, its domain is noted $Dom_{\mathcal{P}}$, the notation $Dom^o_{\mathcal{P}}$ (resp. $Dom^s_{\mathcal{P}}$) is used to designate the sub-domain of $Dom_{\mathcal{P}}$ that defines $\mathcal{P}$'s objects (resp. subjects). Finally, $R_{\mathcal{P}}$ is the set of access control rules of the policy $\mathcal{P}$.

Example. We now illustrate our model for policy specification with the example of a file system expressed as a distributed architecture. Let $\mathcal{FS}$ be the set of software components accessing files. Let further $READ \subset \mathcal{FS}$ and $WRITE \subset \mathcal{FS}$ be respectively the set of software components allowed to read and to write files. We also consider two sets of users, A and C, and two sets of files, B and D. We introduce the predicate $Owner(sc, U)$ which, given a software component sc, holds if the owner of sc (i.e. the user executing sc) belongs to the set U of users (i.e. A or C). In the following, we do not specify reflexive accesses. Given the above definitions, we define the $\mathcal{P}_1$ and $\mathcal{P}_2$ access control policies introduced in §2, as follows:

- The subject domain of the policy $\mathcal{P}_1$ is the set of components $READ$ ($Dom^s_1 = READ$), and $\mathcal{P}_1$'s object domain is the set of files B ($Dom^o_1 = B$). The policy $\mathcal{P}_1$ has a single access control rule:

  $\lambda_1 = \{(s \in READ), (o \in B) \mid Owner(s, A); s \sim\!\!> o\}$,

  that authorizes subjects of A to access objects of B.

[4] We use the term *classification* for both object classification and subject clearance.

[5] We consider that an access corresponds to the execution of an operation (e.g. read access corresponds read method). This enables us to use a single access operator.

– The subject domain of the policy $\mathcal{P}_2$ is the set of components $WRITE$ ($Dom_2^s = WRITE$), and $\mathcal{P}_2$'s object domain is the set of files D ($Dom_2^o = D$). The policy $\mathcal{P}_2$ has a single access control rule:
  $\lambda_2 = \{(s \in WRITE), (o \in D) \mid Owner(s, C); s \rightsquigarrow\!\!> o\}$,
  that authorizes subjects of C to access objects of D.

3.2 Combining access control policies

A combined access control policy is built by combining two *sub-policies*. The domain and rules of the combined policy are defined in terms of the combination of its sub-policies' domains and rules.

Combining access control domains Given two classifications, four types of combination can be implemented: the union, intersection, product of classifications, and the denial of classification combination.

Let $Cl_1 = \{\varepsilon \in \mathcal{E} \mid P_1\}$ and $Cl_2 = \{\varepsilon \in \mathcal{E} \mid P_2\}$ be two classifications. We formally define their union and intersection as: $Cl_1 \cup Cl_2 = \{\varepsilon \in \mathcal{E} \mid P_1 \vee P_2\}$ and $Cl_1 \cap Cl_2 = \{\varepsilon \in \mathcal{E} \mid P_1 \wedge P_2\}$ respectively. Concerning the product of Cl_1 and Cl_2 (noted $Cl_1 \uplus Cl_2$), three separate classifications are computed: the classifications defined by the access control predicates $P_1 \wedge \neg P_2$, $\neg P_1 \wedge P_2$ and $P_1 \wedge P_2$ respectively. Finally, to forbid the combination of two classifications, we introduce the operator $\not\cup$ defined by: $Cl_1 \not\cup Cl_2 = \emptyset$.

The $\cup$ operator allows to extend the sub-classifications. On the other hand, the classification obtained by $\cap$ operator is a restriction of the sub-classifications. Finally, the $\uplus$ operator permits to distinguish the entities belonging to a single sub-classification from the entities belonging to both sub-classifications.

The formal definition for combining domains directly follows: it amounts to specifying the type of combination for each pair of classifications of the sub-policies' domains. Let $Dom_1 = \{Cl_i\}_{i=1}^{n}$ and $Dom_2 = \{Cl_j\}_{j=1}^{m}$ be two access control domains. Let $\Delta = \{\delta_{i,j} \in \{\cup, \cap, \uplus, \not\cup\}, i = 1, ..., n, j = 1, ..., m\}$ be the set specifying the operator for combining the classifications Cl_i of Dom_1 and Cl_j of Dom_2. The combined access control domain Dom resulting from the combination of Dom_1 with Dom_2 with respect to Δ is given by:

$Dom = \{\forall\ i = 1, ..., n, \forall\ j = 1, ..., m, Cl_{i,j} = Cl_i\ \delta_{i,j}\ Cl_j\}$.

We bring to the reader's attention that a combined domain can itself be combined with another domain.

Combining access control rules. Because both $\rightsquigarrow\!\!>$ and $\not\rightsquigarrow\!\!>$ access operators exist, the combination of access control rules raises the problem of conflicts among rules. Thus, we distinguish between combination of non-conflicting rules and combination of conflicting rules.

Combination of non-conflicting rules. Let us first consider the combination of non-conflicting rules. Given two non-conflicting access control rules, their combination results from the combination of their classifications and of their access control predicate.

Let λ_1 and λ_2 be two access control rules specifying either authorization for both or denial of access for both:
$\lambda_1 = \{(s \in Cl_1), (o \in Cl'_1) \mid P_1(s,o) \ ; s{\div}{>}o\}$ and
$\lambda_2 = \{(s \in Cl_2), (o \in Cl'_2) \mid P_2(s,o) \ ; s{\div}{>}o\}$.

Let us first suppose that $Cl_1 = Cl_2$ and $Cl'_1 = Cl'_2$. Then, combining λ_1 and λ_2 consists of combining the access control predicates P_1 and P_2. With respect to existing work (e.g. [2]), we identify two types of combination: the *logical and* ($\wedge$) and the *logical or* ($\vee$) between P_1 and P_2. Then, the λ access control rule resulting from the combination of λ_1 and λ_2 takes one of the following forms:
$\lambda = \{(s \in Cl_1), (o \in Cl'_1) \mid (P_1(s,o) \vee P_2(s,o)) \ ; s{\div}{>}o\}$, or
$\lambda = \{(s \in Cl_1), (o \in Cl'_1) \mid (P_1(s,o) \wedge P_2(s,o)) \ ; s{\div}{>}o\}$.
Informally, the *logical or* operator allows to combine two access control rules in such a way that the resulting access control rule preserves the authorized accesses of the sub-rules (principle of autonomy). On the other hand, the access control rule resulting from the *logical and* operator authorizes (resp. denies) access if both sub-rules authorize (resp. deny) the access.

Let us now suppose that $Cl_1 \neq Cl_2$ and $Cl'_1 \neq Cl'_2$. Then, the set of access control rules resulting from the combination of λ_1 and λ_2 depends not only on the operator for combining access control predicates, but also on the operators for combining classifications. Let $\delta_{1,2}$ (resp. $\delta'_{1,2}$) be the operator defined for the combination of the Cl_1 and Cl_2 (resp. Cl'_1 and Cl'_2) classifications. Then, the set Λ of access control rules takes one of the following forms, depending on the operators used for combining classifications:

- if $\delta_{1,2} = \not\cup$ or $\delta'_{1,2} = \not\cup$, then $\Lambda = \emptyset$ (the empty set), i.e. the combination is forbidden.
- if $\delta_{1,2} \in \{\cup, \cap\}$ and $\delta'_{1,2} \in \{\cup, \cap\}$, then Λ consists of a single access control rule:
  $\lambda = \{(s \in Cl_1 \ \delta_{1,2} \ Cl_2), (o \in Cl'_1 \ \delta'_{1,2} \ Cl'_2) \mid (P_1(s,o) \ \mathbf{op} \ P_2(s,o)) \ ; s{\div}{>}o\}$,
  where $\mathbf{op} \in \{\vee, \wedge\}$.
- if $\delta_{1,2} = \uplus$, we only combine the access control predicates for subjects belonging to $Cl_1 \cap Cl_2$, and we keep unchanged the rules for the other entities. In other words, Λ consists of three separate access control rules:
  $\lambda = \{(s \in (Cl_1 - Cl_2)), (o \in (Cl'_1 \ \delta'_{1,2} \ Cl'_2)) \mid P_1(s,o) \ ; s{\div}{>}o\}$,
  $\lambda' = \{(s \in (Cl_2 - Cl_1)), (o \in (Cl'_1 \ \delta'_{1,2} \ Cl'_2)) \mid P_2(s,o) \ ; s{\div}{>}o\}$, and
  $\lambda'' = \{(s \in (Cl_1 \cap Cl_2)), (o \in (Cl'_1 \ \delta'_{1,2} \ Cl'_2)) \mid (P_1(s,o) \ \mathbf{op} \ P_2(s,o)) \ ; s{\div}{>}o\}$,
  where $\mathbf{op} \in \{\vee, \wedge\}$. If $\delta'_{1,2} = \uplus$ or $\delta_{1,2} = \delta'_{1,2} = \uplus$, the set Λ of access control rules is equivalent to the one computed for $\delta_{1,2} = \uplus$.

Coupling the operators for combining classifications and the ones for combining access control predicates, allows either to restrict or to extend the access control rules of the sub-policies. In particular, using $\delta_{1,2} = \delta'_{1,2} = \cap$ and the $\wedge$ operator for combining access control predicates, the access control rule λ resulting from the composition of λ_1 and λ_2, verifies that: $(s,o) \in \lambda \iff (s,o) \in \lambda_1$

and $(s,o) \in \lambda_2$, i.e. the access is authorized (resp. denied) if and only if it is authorized (resp. denied) by both λ_1 and λ_2 (λ is more specialized than λ_1 and λ_2 in the sense that it provides finest classifications). In this case, the composed policy keeps the principles of autonomy and secrecy of the sub-policies [14]: the combined policy is identical to the one obtained with policy interoperation.

Combination of conflicting rules. Let us now examine the combination of conflicting rules. Let λ_1 and λ_2 be two access control rules specifying respectively access authorization and access denial:

$\lambda_1 = \{(s \in Cl_1), (o \in Cl'_1) \mid P_1(s,o)\ ; s {\rightsquigarrow} o\}$ and

$\lambda_2 = \{(s \in Cl_2), (o \in Cl'_2) \mid P_2(s,o)\ ; s \not\rightsquigarrow o\}$.

Let $\delta_{1,2}$ (resp. $\delta'_{1,2}$) be the operator defined for the combination of the classifications Cl_1 and Cl_2 (resp. Cl'_1 and Cl'_2). Let further $Cl_{1,2} = Cl_1\ \delta_{1,2}\ Cl_2$ and $Cl'_{1,2} = Cl'_1\ \delta'_{1,2}\ Cl'_2$. The $\wedge^+$ and $\wedge^-$ operators allow to combine two conflicting rules: $\lambda_1 \wedge^+ \lambda_2$ and $\lambda_1 \wedge^- \lambda_2$. The result of the combination of λ_1 and λ_2 consists of three separate rules λ, λ' and λ'' defined by:

$\lambda = \{(s \in Cl_{1,2}), (o \in Cl'_{1,2}) \mid (P_1(s,o) \wedge \neg P_2(s,o))\ ; s \rightsquigarrow o\}$

$\lambda' = \{(s \in Cl_{1,2}), (o \in Cl'_{1,2}) \mid (\neg P_1(s,o) \wedge P_2(s,o))\ ; s \not\rightsquigarrow o\}$

$\lambda'' = \{(s \in Cl_{1,2}), (o \in Cl'_{1,2}) \mid (P_1(s,o) \wedge P_2(s,o))\ ; s\ \mathsf{ac}\ o\}$

with ($\mathsf{ac} \equiv \rightsquigarrow$) for the operator $\wedge^+$, and ($\mathsf{ac} \equiv \not\rightsquigarrow$) for the operator $\wedge^-$.

The λ and λ' rules are coherent with both sub-rules. On the other hand, the λ'' rule is clearly inconsistent with either λ_1 or λ_2. The $\wedge^+$ operator favors access authorization whereas the $\wedge^-$ operator favors access denial. Notice that the $\wedge^+$ and $\wedge^-$ operators are close to the notion of strong and weak authorizations introduced in database systems to support multiple access control policies [5].

As with the composition of non-conflicting rules, coupling the operators for combining classifications and the ones for combining access control rules allows either to restrict or to extend the sub-rules.

Finally, the combination of two sets of access control rules $R_1 = \{\lambda_1^i\}_{i=1}^n$ and $R_2 = \{\lambda_2^j\}_{j=1}^m$ amounts to specifying, for each pair of rules $(\lambda_1^i, \lambda_2^j)$, the operators for combining their classifications and their access control predicate. Let us remark that the proposed combination model applies recursively: a combined policy can be combined with another policy.

Example. We now illustrate the use of combination operators with the example of the file system given in §3.1. Given the component classifications $READ$ and $WRITE$, the two sets of users A and C, and the file classifications B and D, we define the following access control policy $\mathcal{P}$: the $(A - C)$ users are authorized to read files in $(B - D)$, $(A \cap C)$ users can read and write files in $(B \cap D)$, and $(C - A)$ users are authorized to write files in $(D - B)$.

Using the $\uplus$ operator for combining classifications, we are able to differentiate components that only allow to read files ($R\_ONLY = (READ - WRITE)$), components that only allow to write files ($W\_ONLY = (WRITE - READ)$), and components that allow to read and to write files ($R\_and\_W = (READ \cap$

$WRITE$)). The operator $\uplus$ further allows to differentiate the following sets of files: $B - D$, $B \cap D$ and $D - B$. Finally, using the operator $\wedge$ for combining access control rules, we are able to get the specifications of $\mathcal{P}$:

$\lambda = \{(s \in R\_ONLY), (o \in B - D) \mid Owner(s, A) \ ; s \rightsquigarrow o\}$,

$\lambda' = \{(s \in W\_ONLY), (o \in D - B) \mid Owner(s, C) \ ; s \rightsquigarrow o\}$, and

$\lambda'' = \{(s \in R\_and\_W), (o \in (B \cap D)) \mid (Owner(s, A) \wedge Owner(s, C)) \ ; s \rightsquigarrow o\}$.

3.3 Complete and sound access control policy

We have proposed a model for the specification of access control policies as well as a set of combination operators allowing to compose these specifications. Now, we define the notion of secure access control policy in the context of our model: the policy is *secure* if and only if its specification is *complete* and *sound*. Let us precisely define the completeness and soundness of policy specifications.

Definition 1. (Complete policy) *An (access control) policy $\mathcal{P}$ is* complete *(from the standpoint of its specification) if and only if: (1) there exists an access control rule for every object and subject of $\mathcal{P}$'s domain, and (2) in the case that $\mathcal{P}$ is a combined (access control) policy, its sub-policies are also complete.*

In an open distributed architecture, an access control policy cannot be defined for the whole set of the system's entities. The above definition states that a complete access control policy ensures access control for any entity belonging to the policy's object domain, and for entity belonging to the policy's subject domain[6].

Definition 2. (Sound policy) *An (access control) policy $\mathcal{P}$ is sound (from the standpoint of its specification) if and only if: (1) given a subject s and an object o belonging to the policy's domains, all the access control rules specifying whether s can access o or not, have the same access control operator, and (2) in the case that $\mathcal{P}$ is a combined (access control) policy, its sub-policies are sound.*

Given two sound sub-policies, if both specify only access authorization (or only denial) then for all combination operators, the combined policy is sound. On the opposite, when at least one of them specifies both authorized and denied accesses, the soundness of the combined policy may not be guaranteed. However, given two complete and sound access control policies, we have the following result which guarantees that there exists a secure policy resulting from the combination of these policies.

Theorem 1. (Existence of secure combined access control policy) *Let $\mathcal{P}_1$ and $\mathcal{P}_2$ be two secure access control policies. There exists a set of operators for combining classifications and for combining access control rules such that, if $\mathcal{P}$ is the policy resulting from the combination of $\mathcal{P}_1$ and $\mathcal{P}_2$ according to these sets, then $\mathcal{P}$ is secure.*

[6] In general, the completeness of a given policy can be guaranteed by specifying default access control rules.

Proof: Let $Dom_1 = \{Cl_i\}_{i=1}^{n}$ (resp. $Dom_2 = \{Cl_j\}_{j=1}^{m}$) be $\mathcal{P}_1$'s (resp. $\mathcal{P}_2$'s) access control domains. Let $\Delta = \{\delta_{i,j} = \cap, i = 1, ..., n, j = 1, ..., m\}$ be the set of operator for combining classifications. Let $R_1 = \{\lambda_1^i\}_{i=1}^{n}$ (resp. $R_2 = \{\lambda_2^j\}_{j=1}^{m}$) be $\mathcal{P}_1$'s (resp. $\mathcal{P}_2$'s) sets of access control rules. Let $\Psi = \{\alpha_{i,j} \in \{\vee, \wedge^+, \wedge^-\}, 1, ..., n, j = 1, ..., m\}$ be the set of operator for combining access control predicates, with: *(1)* $\alpha_{i,j} = \vee$ if λ_1^i and λ_2^j are non-conflicting rules; *(2)* $\alpha_{i,j} = \wedge^+$ if λ_1^i and λ_2^j are conflicting rules and λ_1^i is an access authorization rule; and *(3)* $\alpha_{i,j} = \wedge^-$ if λ_1^i and λ_2^j are conflicting rules and λ_1^i is an access denial rule. It is trivial to show that the policy $\mathcal{P}$ resulting from the composition of $\mathcal{P}_1$ and $\mathcal{P}_2$ according to the sets Δ and Ψ is a complete and sound policy (i.e. $\mathcal{P}$ is secure). $\square$

Example. Let us now address the completeness and soundness of policies specified in the file system example. The policies $\mathcal{P}_1$ and $\mathcal{P}_2$ are not complete. For instance, there are not access control rules concerning software components not belonging to subject's classifications and which require to access B or D files. If we specify default rules which deny all accesses between entities in $\mathcal{P}_1$ (resp. $\mathcal{P}_2$) security domain and the other entities, $\mathcal{P}_1$ (resp. $\mathcal{P}_2$) becomes complete. By giving priority to the access control rules over the default rules, the soundness of the policies $\mathcal{P}_1$ and $\mathcal{P}_2$ is also verified. Finally, by using the same default rules, the combined policy $\mathcal{P}$ is secure.

4 Extension for Information Flow policies

In this section, we extend the proposed model for specifying and combining access control policies to introduce the notion of information flow. Then, we address completeness and soundness of the resulting specifications, hence defining secure information flow policy according to our model.

4.1 Information flow policy

The previous model allows to describe access control policies but does not integrate any notion of information flow. Therefore, it does not meet the specification of information flow. In order to deal with flow policies, we extend the previous model as follows: (1) the specifications of access control rules are enriched with the notion of information flow, and (2) rules that control the transitive information flow (by denying some information flows) are introduced.

Input and output flows. For specifying the information flow that is associated to an access control rule, let us distinguish the access control rules for input flow (e.g. the read accesses) from the access control rules with output flow (e.g. the write accesses). We extend our definition of access control rules given in §3.1 by specifying the flow associated to the access operation thanks to the operators $<\!\sim$ and $\sim\!>$.

Definition 3. (Rules for input and output flow) *An access control rule for input flow takes the form:* $\varphi = \{(s \in Cl), (o \in Cl') \mid P(s,o) \; ; s \lessdot\!\!\sim o\}$. *An access control rule for output flow takes the form:* $\varphi = \{(s \in Cl), (o \in Cl') \mid P(s,o) \; ; s \sim\!\!> o\}$.

Given two entities ϵ_1 and ϵ_2, the expression $\epsilon_1 <\!\!\sim \epsilon_2$ specifies that ϵ_1 is authorized to access ϵ_2, and this access produce a flow from ϵ_1 to ϵ_2. Dually, the expression $\epsilon_1 \sim\!\!> \epsilon_2$ specifies that ϵ_1 is authorized to access ϵ_2, and this access produce a flow from ϵ_2 to ϵ_1. Note that we only specify the information flow for the authorized accesses.

Information flow rules. The information flow resulting to accesses is transitive: let ϵ_1, ϵ_2 and ϵ_3 be system entities, if $\epsilon_1 \sim\!\!> \epsilon_2$ and $\epsilon_2 \sim\!\!> \epsilon_3$ then $\epsilon_1 \sim\!\!> \epsilon_3$. In order to specify information flow policies, we specify the information flows which are denied, i.e. the information flow rules.

Definition 4. (Information flow rule) *An information flow rule is defined as* $\phi = \{(\epsilon_1 \in Cl), (\epsilon_2 \in Cl') \mid P(\epsilon_1, \epsilon_2) \; ; \epsilon_1 \not\leadsto \epsilon_2\}$.

The rule ϕ specifies that the information flow from Cl entities to Cl' entities verifying the predicate P is denied. Then, an information flow policy is defined as follows:

Definition 5. (Information flow policy) *An information flow policy is an access control policy for which the access control rules are enriched according to the definition 3, and which defines a set of information flow rules.*

Example. In order to illustrate our model, we describe a simple multilevel security policy [4], called $\mathcal{F}_S$. We consider a distributed military system consisting of a set of software components (i.e. programs representing users), and a set of files that the components manipulate. Suppose that we have two security levels for both files and components: $Secret$ and $Unclassified$[7] such that $Secret > Unclassified$ (i.e. the $Secret$ information is more sensible than the $Unclassified$ information).

A software component can use a set of methods (e.g. read, write, execute, etc) to access files. In the following, we only consider the read and write methods. We define the $READ$ (resp. $WRITE$) classification to be the set of software components allowed to read (resp. write) files. We introduce the predicate $(\epsilon_1 \geq \epsilon_2)$ which, given two entities ϵ_1 and ϵ_2, holds if and only if the security level of ϵ_1 dominates, i.e. is greater than or equal to the security level of ϵ_2. We define the following information flow policy $\mathcal{F}_S$:

– The subject domain consists of the $READ$ and $WRITE$ classifications, and the object domain consists of files belonging to $Secret$ or $Unclassified$ classifications.

[7] The $Secret$ and $Unclassified$ terms are used for denoting both the security level, the classifications and the predicate.

– A component belonging to $READ$ is authorized to access a file if and only if its security level dominates the one of the file, and the flow resulting to such access is an input flow:
  $\lambda_1 = \{(s \in READ), (o \in (Secret \ \cup \ Unclassified)) \mid s \ \geq \ o \,; s {<\!\sim} o\}$.
– A component belonging to $WRITE$ is authorized to access an file if and only if the file security level dominates the one of the component, and the flow resulting to such access is an output flow:
  $\lambda_2 = \{(s \in WRITE), (o \in (Secret \cup Unclassified)) \mid s \ \geq o \,; s {\sim\!>} o\}$.
– All the accesses not authorized previously are denied through a default rule.
– $Unclassified$ entities are not authorized to access $Secret$ information:
  $\phi_S = \{(\epsilon_1 \in READ), (\epsilon_2 \in (Secret \cup Unclassified)) \mid (\epsilon_2 \ \geq \ \epsilon_1) \,; \epsilon_1 \not\leftrightarrow \epsilon_2\}$.

4.2 Combined information flow policy

Let us now address the combination of information flow policies. Given two information flow policies, we distinguish three steps for the combination of these policies: (1) the combination of these policies according to the operators for combining access control policies (see §3.2), (2) the combination of the $<\!\sim$ and $\sim\!>$ operators in order to specify the information flow of each combined access control rule, and (3) the combination of information flow rules.

The first step allows to obtain the access control rules of the combined flow policy. During this step, the information flow sub-policies are viewed as access control policies, and the combination is carried out from the standpoint of combination of access control policies. The second step permits to specify the information flow within the combined access control policy obtained at the first step. Informally, the combination of two rules for input flow (resp. rules with output flow) is a rule for input flow (resp. rule with output flow). In the other case, the combined rule is a rule for both input and output flow ($<\!\sim\!>$ operator).

Finally, we compute the combined information flow rules by composing the information flow rules of the sub-policies, in a way similar to the combination of non-conflicting rules (see §3.2). Informally, the *logical or* operator allows to combine two information flow rules so as to generate a unique rule that provides the information flow control of the sub-policies. On the other hand, the information flow rule resulting from the *logical and* operator denies the information flow if and only if both sub-rules deny the flow.

Example. Let us combine the information flow policy $\mathcal{F}_S$ introduced in §4.1 with another policy having different classifications. Suppose that we have two classifications in terms of military domains for both files and software components (e.g. users): *Nuclear* and *Conventional* classifications such as $Nuclear >$ *Conventional*. Similarly, we define the information flow policy $\mathcal{F}_D$, except for the set of information flow rules that we define as an empty set.

By using the $\cap$ operator for combining classifications, and the $\wedge$ operator for combining access control predicates, we obtain the policy $\mathcal{F}$ depicted in figure 1. For simplifying the figure, we do not distinguish components and files, and we do not show neither the denial accesses nor the implied transitive accesses. The specification of the information flow in the combined access control rules are immediate according to the sub-policies' specifications (i.e. the read access produces input flow whereas the write access generates output flow). The information flow rule of the combined policy is the one of the policy $\mathcal{F}_S$:

$$\phi = \{(\epsilon_1 \in READ), (\epsilon_2 \in (Secret \cup Unclassified)) \mid (\epsilon_2 \geq \epsilon_1) \,; \epsilon_1 \not\rightarrow \epsilon_2\}.$$

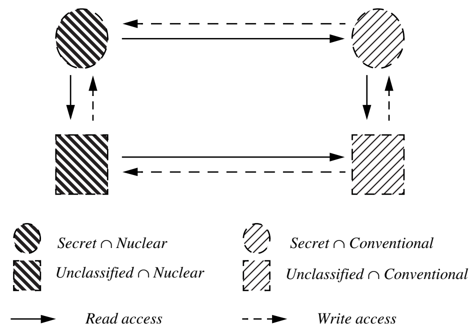

Fig. 1. Combined multilevel policies

4.3 Complete and sound information flow policy

Given the specifications of information flow policies in our model, we now focus on the notion of secure information flow policy by addressing the completeness and soundness properties.

In our model, we only specify the information flows which are denied. In other words, any information flows that are not denied are authorized, and the problem of conflicts among information flow rules does not exists. Based on this remark, an information flow policy $\mathcal{F}$ is *complete* (from the standpoint of its specification) if and only if $\mathcal{F}$ is complete as an access control policy (see definition 1). In the same way, an information flow policy $\mathcal{F}$ is sound (from the standpoint of its specification) if and only if $\mathcal{F}$ is sound as an access control policy (see definition 2).

A secure information flow policy is a policy for which the specifications in our model are complete and sound. In the case of a combined policy, the secure property implies that the policy's specifications are coherent with the specifications of the sub-policies according to the combination operators that are used. Like in §3.3, given two information flow policies, there exists operators for combining classifications and rules such that the combined policy is secure.

Notice that a policy being secure according to our model does not imply that the policy have no covert channels. For example, the write access is an access having output flow, but, when the file does not exist, the information flow induced by the write method is an input flow: after the operation, the user have information concerning the file's existence. Such covert channels do not invalidate the secure property. However, our secure property ensure that the policy's specifications are defined *without ambiguity* for any entity belonging to the policy's domain.

Example. Let us address the completeness and the soundness of the multilevel policies $\mathcal{F}_S$ and $\mathcal{F}_D$. Let us consider that the classifications in terms of security level or in terms of military domains are both complete classifications, i.e. any entity of the system has a (unique) security level and belongs to a (unique) military domains. Then the flow policies $\mathcal{F}_S$ and $\mathcal{F}_D$ becomes complete and sound access control policies. In the same way, the combined flow policy $\mathcal{F}$ is complete and sound as an access control policy. Finally, the soundness of the policy $\mathcal{F}$ is trivial, and $\mathcal{F}$ is a secure flow policy.

5 Conclusion

In this paper, we have proposed a practical solution for reasoning about the coexistence of different security policies. Our approach relies on the specification of security policies and on the definition of operators for combining these specification. We have also addressed the completeness and the soundness of the (combined) security policy according to its specification.

Our model can be viewed as an implementation design of the Bell's model. We have introduced a set of combination operators enabling security officers to specify and to combine security policies in a controlled and secure manner. In addition, we assert that the dissociation between the combination domains and the combination rules permits to get a simplified and growing vision of security policies in open distributed systems. Finally, the distinction between access operations and the generated information flows gives to our model a useful approach, information flow policies being considered as an extension of access control policies.

We have illustrated our approach through two different examples of file systems in an distributed architecture. In particular, the multilevel security policy example enables us to demonstrate the strength for specifying security policies and the ease for combining security policies.

Current and future work

In open distributed systems, the coexistence of applications which have different security constraints results from rich and complex interactions among software components [15]. Integrating our model in the software architecture paradigm provides an appealing solution to the sound development of applications with security requirements [7]. More specifically, such a solution allows to specify the security requirements, and to build systematically, from existing components, the reference monitor that meet these requirements. One application area for our results can be the World-Wide-Web (Www): we are currently examining the automatic selection of the components for ensuring information exchanges in a secure way. We also consider the use of this solution for reasoning about Java-applets, in order to customize the Java Virtual Machine in a controlled and secure way.

From the implementation standpoint, we are currently integrating our model in the Aster *configuration-based* distributed programming environment [6, 17]. The resulting framework allows the easy development of applications consisting of heterogeneous software components, running on heterogeneous hardware platforms, and having multiple security requirements. From the theoretical perspective, we are interested with the extension of our model in order to specify security policies with dynamic relabeling [12].

References

1. M. Abadi and L. Lamport. Composing Specification. Technical Report 66, Digital Systems Research Center, Oct. 1990.
2. M. Abrams, L. LaPadula, K. Eggers, and I. Olson. A Generalized Framework for Access Control: an Informal Description. In *Proceedings of the 13th National Computer Security Conference*, pages 134–143, Oct. 1990.
3. D. E. Bell. Modeling the Multipolicy Machine. In *Proceedings of the New Security Paradigm Workshop*, pages 2–9, Aug. 1994.
4. D. E. Bell and L. J. LaPadula. Secure Computer Systems: Unified Exposition and Multics Interpretation. Technical Report MTR-2997 Rev. 1, MITRE Corporation, Bedford, Mass, 1976.
5. E. Bertino, S. Jajodia, and P. Samarati. Supporting Multiple Access Control Policies in Database Systems. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 94–107, May 1996.
6. C. Bidan and V. Issarny. A Configuration-based Environment for Dealing with Multiple Security Policies in Open Distributed Systems. In *Proceedings of the 2nd European Research Seminar on Advances in Distributed Systems*, Mar. 1997. URL: `http://www.irisa.fr/solidor/work/aster`.
7. C. Bidan and V. Issarny. Security Benefits from Software Architecture. In *Proceedings 2nd International Conference on Coordination Models and Languages*, pages 64–80, Sept. 1997. URL: `http://www.irisa.fr/solidor/work/aster`.
8. P. Bieber and F. Cuppens. A logical view of secure dependencies. *Journal of Computer Security*, 1(1):99–129, 1992.
9. D. Clark and D. Wilson. A Comparison of Commercial and Military Computer Security Policies. In I. C. Society, editor, *Proceedings of the IEEE Symposium on Security and Privacy*, 1987.
10. R. Deng, S. Bhonsle, W. Wang, and A. Lazar. Integrating Security in CORBA Based Object Architectures. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 50–61, May 1995.
11. Department of Defense Standard. Trusted Computer System Evaluation Criteria. Technical Report DoD 5200.28-STD, Dec. 1985.
12. S. Foley, L. Gong, and X. Qian. A Security Model of Dynamic Labeling Providing a Tiered Approach to Verification. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 142–153, May 1996.
13. J. Goguen and J. Meseguer. Security Policies and Security Models. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 11–20, May 1982.

14. L. Gong and X. Qian. Computational issue in secure interoperation. *IEEE Transactions on Software Engineering*, 22(1):43–52, Jan. 1996.
15. OMG Security Working Group. White Paper on Security. TC Document 94.4.16, OMG, Apr. 1994. Available by ftp at `ftp.omg.org:/pub/docs`.
16. H. Hosmer. Metapolicies II. In *Proceedings of the 15th National Computer Security Conference*, pages 369–378, 1992.
17. V. Issarny, C. Bidan, and T. Saridakis. Achieving Midleware Customization in a Configuration-Based Development Evironment: Experience with the Aster Prototype. In *Proceedings of the 4th International Conference on Configurable Distributed Systems*, 1998. URL: `http://www.irisa.fr/solidor/work/aster`.
18. C. E. Landwehr. Formal models for computer security. *ACM Computing Surveys*, 13(3):247–278, Nov. 1981.
19. J. McLean. The Algebra of Security. In *Proceedings of the 1988 IEEE Computer Society Symposium on Security and Privacy*, pages 2–7, Apr. 1988.
20. J. McLean. Security Models and Information Flow. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 180–187, May 1990.
21. J. McLean. A general theory of composition for a class of possibilistic properties. *IEEE Transactions on Software Engineering*, 22(1):53–67, Jan. 1996.
22. J. D. Moffett, M. D. Sloman, and K. Twidle. Specifying Discretionary Access Control Policy for Distributed Systems. *Computer Communications*, 13(9):571–580, Nov. 1990.
23. National Computer Security Center. Trusted Network Interpretation of the TCSEC. Technical Report NCSC-TG-005, July 1987.
24. OMG. The Common Object Request Broker: Architecture and Specification – Revision 2.0. Technical report, OMG Document, 1995.
25. D. Sutherland. A Model of Information. In *Proceedings of the 9th National Computer Security Conference*, pages 2–12, Sept. 1986.
26. TINA-C. TINA Object Definition Language (TINA-ODL) Manual – Version 1.3. Technical Report TR_NM.002_1.3_95, TINA-C Document, 1995.