Nomad: A Security Model with Non Atomic Actions and Deadlines

Frédéric Cuppens  Nora Cuppens-Boulahia  Thierry Sans

\{frederic.cuppens,nora.cuppens,thierry.sans\}@enst-bretagne.fr

GET/ENST-Bretagne
2, rue de la Châtaigneraie
35576 Cesson Sévigné - France

Abstract

Modelling security policies requires means to specify permissions and prohibitions. However, this is generally not sufficient to express security properties such as availability and obligations must be also considered. By contrast to permissions and prohibitions, obligations are often associated with deadlines to specify bounded time availability requirements. In this case, a violation only occurs if the obliged action is not performed before the deadline. On the other hand, when specifying high level security policies, it is convenient to consider abstract non atomic actions. Since most access control mechanisms only deal with atomic actions such as read or write, these non atomic actions must be decomposed into more basic ones. In this paper, we define a formal security model called Nomad to express privileges on non atomic actions. This model combines deontic and temporal logics. In Nomad, we model conditional privileges and obligations with deadlines. We also formally analyze how privileges on non atomic actions can be decomposed into more basic privileges on elementary actions.

1. Introduction

Classical access control models such as DAC [13], MAC [3] or RBAC [20] only consider authorization on atomic actions such as read, write, etc. An important limitation of these models is that they do not provide means to control execution of complex and structured activities such as those used in workflow systems, for instance.

More recent models such as TBAC [21] or workflow models suggested by Atluri et al. [1, 5] attempt to circumvent this limitation. These models provide access control that depends on the history of events. The objective is to define requirements for just in time authorization, that is a subject will obtain privileges just when it is necessary for performing a given task scheduled in the workflow. This subject will then loose the privileges immediately after executing the task.

However, these models do not provide a formal semantics to privileges granted to non elementary actions. Notice that when dealing with privileges that apply to atomic actions, it is possible to use classical first order logic to model such privileges [4, 14]. However, when non atomic actions are used, first order logic is no longer appropriate. For instance, a permission to execute action \( \alpha \) and a permission to execute action \( \beta \) do not imply a permission to execute action \( \alpha \) followed by action \( \beta \) [18].

The solution suggested in this paper to deal with this problem is based on deontic modal logic [9]. Using deontic logic, it is possible to formally define privileges that apply to non atomic actions and then analyze how these privileges may be decomposed into more elementary privileges on atomic actions. This is useful to define a general approach to refine security policies, starting with high level security requirements on abstract non elementary actions, and then decompose these requirements to obtain concrete policies that could be used to configure various access control components such as firewalls, operating systems or database management systems.

On the other hand, classical models only consider positive authorizations also called permissions. Some of them include negative authorizations also called prohibitions. However, recently, several authors argued that security models should also include obligations [6]. Compared with permission and prohibition, there are two major differences when managing obligations:

1. When a permission or prohibition is granted, it applies immediately. By contrast, a subject will have generally some time to fulfill an obligation. Take for example an obligation to change its password before the end of the month. So, in this case, it is necessary to associate the obligation with a deadline [8].
2. Characterization of situation of violation is more complex when we deal with obligations. This is a consequence of the first item. If we consider permission or prohibition, a violation occurs when a subject executes an action whereas he is prohibited (or not permitted) to do so. If we consider an obligation associated with a deadline, a violation occurs when the deadline is reached and the obligation is not fulfilled.

The main contribution of this paper is to define a new security model with Non Atomic Actions and Deadlines, called Nomad, that combines deontic and temporal modal logic. This model provides means to specify:

- Privileges (that is permission, prohibition or obligation) associated with non atomic actions.
- Conditional privileges, that is privileges that are only triggered when specific conditions are satisfied.
- Privileges that must be fulfilled before some specific deadlines.

Nomad also includes formal definition of violation. In this paper, we focus on these problems. In particular, to simplify the notation, we only consider a policy that applies to a single subject (who is omitted in the notation). The model should be further refined to be able to specify policies that apply to set of subjects or roles.

Using this model, we shall show how to specify several security properties, including access control and availability requirements. We also investigate the problem of decomposing obligations, permissions and prohibitions that apply to non atomic actions into privileges on atomic actions.

The remainder of this paper is organized as follows. Section 2 presents a logic of temporized actions that provides means to reason about non atomic actions (sequence and parallelism) and action duration. This logic is extended in section 3 to include the possibility to request for the execution of an action. Based on the logic of temporized actions with request, we then define in section 5 our logic to represent privileges with deadlines. Section 6 shows how to use this logic to specify security policies that include conditional privileges, privileges that apply on non atomic actions and privileges associated with deadlines. Section 7 presents an example. Section 8 investigates how to decompose privileges on non atomic actions into privileges on more elementary actions. Section 9 is a comparison with related works and section 10 concludes the paper.

2. Logic of temporized actions

We first introduce a logical model to express non atomic actions. This is a simple model since we only consider two different operators: $\alpha$ followed by $\beta$ denoted $\alpha \& \beta$ and $\alpha$ in parallel with $\beta$ denoted $\alpha \oplus \beta$.

The logic we suggest in this section also includes means to reason about action duration.

2.1. Syntax

We consider an enumerable set $Var$ of propositional variables and an enumerable set $Act$ of relational variables. Propositional variables represent atomic sentences and relational variables represent atomic actions. The syntax of the logic of temporized actions is defined as follows:

- A relational variable is an action.
- If $\alpha$ and $\beta$ are actions then $(\alpha; \beta)$ ("$\alpha$ followed by $\beta$") and $(\alpha \& \beta)$ ("$\alpha$ in parallel with $\beta$") are actions.
- A propositional variable is a formula.
- If $A$ and $B$ are formulae then $(A \& B), (A \lor B), (A \rightarrow B)$ and $(A \leftrightarrow B)$ are formulae.
- If $A$ is a formula then $\neg A, \oplus A$ ("in the next time, $A$") and $\ominus A$ ("in the previous time, $A$") are formulae.
- If $\alpha$ is an action then $start(\alpha)$ ("starting $\alpha$"), $doing(\alpha)$ ("doing $\alpha$") and $done(\alpha)$ ("finishing $\alpha$") are formulae.

Using model operator $\oplus$ and $\ominus$, we then define the modalities $\ominus^d A$ and $\ominus^{\leq d} A$.

If $d \geq 0$, $\ominus^d A$ is to be read "$A$ will be true after $d$ units of time" and $\ominus^{\leq d} A$ is to be read "$A$ is eventually true within a delay of $d$ units of time".

Thus, for every $d \geq 0$, $\ominus^d A$ and $\ominus^{\leq d} A$ are respectively defined as follows:

- $\ominus^0 A = \ominus^{\leq 0} A = A$
- $\ominus^{d+1} A = \ominus^d \oplus A$
- $\ominus^{\leq (d+1)} A = \ominus^{\leq d} A \lor \ominus^{d+1} A$

If $d \leq 0$, $\ominus^d A$ is to be read "$A$ was true $d$ units of time ago" and $\ominus^{\leq d} A$ is to be read "$A$ was true within the last $d$ units of time".

Thus, for every $d \leq 0$, $\ominus^d A$ and $\ominus^{\leq d} A$ are respectively defined as follows:

- $\ominus^0 A = \ominus^{\leq 0} A = A$
- $\ominus^{d-1} A = \ominus^d \ominus A$
- $\ominus^{\leq (d-1)} A = \ominus^{\leq d} A \lor \ominus^{d-1} A$

We finally define $\ominus^{\leq d} A$ as follows:

- $\ominus^{\leq d} A = \ominus A$
- $\ominus^{\leq d} A$ is to be read "$A$ will be true (if $d \geq 0$) or was true (if $d \leq 0$) during $d$ units of time."

2.2. Semantics

A model of the logic of temporized actions corresponds to an history $h$. An history $h$ is a pair $h = (A_h, V_h)$ where:

- $A_h$ is a total function that associates every integer $x$ with a set of pairs $(\alpha, d)$ where $\alpha$ is an action and $d$ is a positive integer.
- $V_h$ is a total function that associates every integer $x$ with a set of propositional variables.

Intuitively, $(\alpha, d) \in A_h(x)$ means that an execution of action $\alpha$ starts at time $x$ and finished at time $x + d$. $P \in V_h(x)$ means that proposition $P$ is true at time $x$. 
We assume that, if \((a, d) \in A_h(x)\) and \((a', d') \in A_h(x)\) then \(d = d'\). The positive integer \(d\) represents the duration of the execution of action \(a\).

We also assume that, once action \(a\) is being done, one cannot start action \(a\) again until \(a\) is finished\(^1\). This is modelled as follows: if \((a, d) \in A_h(x)\), then there is no integer \(x'\) and \(d'\) such that \(x < x'\) and \(x' \leq x + d\) and \((a, d') \in A_h(x')\).

In the following, we shall use \(\|a\|\) to represent the duration of the execution of action \(a\).

We then extend definitions of functions \(A_h\) and \(V_h\) to every action and formula as follows:

- \(((a, \beta, d) \in A_h(x)\) if and only if there are two positive integers \(d_1\) and \(d_2\) such that \((a, d_1) \in A_h(x)\) and \(\beta, d_2 \in A_h(x + d_1)\) and \(d = d_1 + d_2\)
- \(((a, \beta, d) \in A_h(x)\) if and only if there are two positive integers \(d_1\) and \(d_2\) such that \((a, d_1) \in A_h(x)\) and \(\beta, d_2 \in A_h(x)\) and \(d = \max(d_1, d_2)\)
- \((A \land B) \in V_h(x)\) if and only if \(A \in V_h(x)\) and \(B \in V_h(x)\)
- \((A \lor B) \in V_h(x)\) if and only if \(A \in V_h(x)\) or \(B \in V_h(x)\)
- \(\neg A \in V_h(x)\) if and only if \(A \not\in V_h(x)\)
- \(\ominus A \in V_h(x)\) if and only if \(A \not\in V_h(x + 1)\)
- \(\ominus A \in V_h(x)\) if and only if \(A \not\in V_h(x - 1)\)
- \(\text{start}(a) \in V_h(x)\) if and only if there is a positive integer \(d\) such that \((a, d) \in A_h(x)\)
- \(\text{done}(\alpha) \in V_h(x)\) if and only if there is a positive integer \(d\) such that \((a, d) \in A_h(x)\)
- \(\text{doing}(a) \in V_h(x)\) if and only if there are two positive integers \(d_1\) and \(d_2\) such that \((a, d_1) \in A_h(d_1)\) and \(d_1 \leq x\) and \(x \leq d_1 + d_2\)

We say that a formula \(A\) is satisfiable in the logic of temporized actions if there is an history \(h\) and an integer \(x\) such that \(A \in V_h(x)\).

We say that a formula \(A\) is valid, denoted \(\models A\), in the logic of temporized actions if for every history \(h\) and every integer \(x\), we have \(A \in V_h(x)\).

One can check that \(A\) is valid if and only if \(\neg A\) is satisfiable.

2.3. Axiomatization

The axiomatisation of the logic of temporized actions is defined by the following set of elements:

- The axioms of classical propositional logic
- \(\ominus (A \rightarrow B) \rightarrow (\ominus A \rightarrow \ominus B)\)
- \(\ominus (A \rightarrow B) \rightarrow (\ominus A \rightarrow \ominus B)\)
- \(\neg \ominus \neg A \leftrightarrow \ominus A\)
- \(\neg \ominus \neg A \leftrightarrow \ominus A\)
- \(A \rightarrow \ominus A\)
- \(A \rightarrow \ominus A\)
- \(\text{start}(a) \rightarrow \ominus \text{done}(a)\)
- \(\text{start}(a; \beta) \rightarrow (\text{start}(a) \land \ominus \text{done}(\beta))\)
- \(\text{start}(a \& \beta) \rightarrow (\text{start}(a) \land \text{start}(\beta))\)
- \(\text{start}(a \& \beta) \rightarrow \ominus \text{done}(\alpha \& \beta)\) if \(|\alpha| \geq |\beta|\)
- \(\text{doing}(a) \rightarrow (\text{start}(a) \lor \ominus (\text{doing}(a) \land \neg \text{done}(a)))\)
- \((\text{done}(\alpha) \land \neg \text{done}(\beta)) \rightarrow \ominus \text{start}(\alpha)\)
- \(\ominus \text{start}(\alpha) \rightarrow (\text{start}(\alpha) \lor \ominus (\text{doing}(a) \land \neg \text{done}(a)))\)
- \(\ominus \text{start}(\alpha) \rightarrow (\text{start}(\alpha) \lor \ominus (\text{doing}(a) \land \neg \text{done}(a)))\)

\(^1\) We introduce this assumption to avoid logical ambiguity. Indeed, if we consider \(\text{start}(a)\) at time \(x\) and \(\text{done}(a)\) at time \(x'\), then it is impossible to decide without this assumption if it is the same execution of \(a\) that starts at time \(x\) and ends at time \(x'\).
ting \(\alpha\), action \(\alpha\) will be eventually executed. This actually corresponds to an availability requirement. We shall show in section 4 how to specify such availability requirements.

3.1. Syntax

The language of the logic of temporized actions is extended as follows:
- If \(\alpha\) is an action then \(req(\alpha)\) (“request for executing \(\alpha\)”) and \(waiting(\alpha)\) (“waiting for an execution of \(\alpha\)”) are formulae.

3.2. Semantics

A model of the logic of temporized actions with request is an history. However, an history is now a triple \(h = (A, \mathcal{V}, R)\) where:
- Definitions of \(A\) and \(\mathcal{V}\) are similar to the definitions given in the logic of temporized actions.
- \(R\) is a total function that associates every integer \(x\) with a set of actions.

Intuitively, if \(\alpha \in R_h(x)\), then the execution of action \(\alpha\) is requested in the history \(h\) at time \(x\).

We then simply extend the definition of function \(\mathcal{V}\) as follows:
- \(req(\alpha) \in \mathcal{V}_h(x)\) if and only if \(\alpha \in R_h(x)\)
- \(waiting(\alpha) \in \mathcal{V}_h(x)\) if and only if there is a positive integer \(d_1\) such that \(d_1 \leq x\) and \(req(\alpha) \in \mathcal{V}_h(d_1)\) and for every integer \(d_2\), if \(d_1 \leq d_2\) and \(d_2 < x\) then \(start(\alpha) \notin \mathcal{V}_h(d_2)\).

Definitions of satisfiability and validity are similar to the definitions given for the logic of temporized actions.

3.3. Axiomatisation

The axiomatisation of the logic of temporized actions with request is similar to the axiomatisation of the logic of temporized actions except we have the following additional axiom:
- \(waiting(\alpha) \leftrightarrow (req(\alpha) \lor \Box(waiting(\alpha) \land \lnot start(\alpha)))\)

Definitions of a theorem and a consistent formula are similar to the definitions given for the logic of temporized actions.

**Theorem 2**: The axiomatisation of the temporized logic of actions is sound and complete with respect to the semantics given in section 3.2.

**Sketch of Proof**: Proof is similar to theorem 1.

4. Logic of privileges with deadlines

4.1. Syntax

The objective is to provide means to express a security policy. For this purpose, the language is now extended by considering alethic modalities \(\Box\) and \(\Diamond\) and deontic modalities \(O\), \(F\) and \(P\). Differences between alethic modalities and deontic modalities are discussed in [2]. Thus, we insert the following rules:
- If \(A\) is a formula, then \(\Box A\) (“\(A\) is necessary”) and \(\Diamond A\) (“\(A\) is possible”) are formulae.
- If \(A\) is a formula then \(OA\) (“\(A\) is obligatory”), \(FA\) (“\(A\) is forbidden”) and \(PA\) (“\(A\) is permitted”) are formulae.
- \(\Box A\) represents what is necessary in the policy. \(\Box\) means that \(A\) must be true in the sense that \(A\) cannot be violated. Thus we shall accept \(\Box A \rightarrow A\) as an axiom of our logic.
- \(OA\) represents what is obligatory in the policy. \(OA\) means that \(A\) must be true but in the sense that \(A\) may be violated. Thus, we consider that \(OA \land \lnot A\) is a satisfiable formula in our logic.

In some senses, \(\Box A\) may be viewed as a strong obligation that is an obligation that cannot be violated.

4.2. Semantics

A model of the logic of privileges with deadlines is a triple \((H, Nec, Ob)\) defined as follows:
- \(H\) is a set of histories, that is a set of triples \((A, \mathcal{V}, R)\).
- \(Nec\) is a total function that associates every integer \(x\) with a relation on \(H\).
- \(Ob\) is a total function that associates every integer \(x\) with a relation on \(H\).

Intuitively, if \(h\) and \(h'\) are two histories, then \((h, h') \in Nec(x)\) means that \(h'\) is a possible alternative of \(h\). And \((h, h') \in Ob(x)\) means that \(h'\) is an ideal alternative of \(h\) (an ideal alternative is an history where the regulation cannot be violated).

We assume that, for every integer \(x\), \(Nec(x)\) defines an equivalence relation over \(H\), \(Ob(x)\) is a serial relation over \(H\) (\(\forall h \in H, \exists h' \in H, (h, h') \in Ob(x)\) and \(Ob(x) \subseteq Nec(x)\)).

Definition of function \(\mathcal{V}\) is then extended to every formula of the language as follows:
- \(\Box A \in \mathcal{V}_h(x)\) if and only if for every history \(h',\) if \((h, h') \in Nec(x)\) then \(A \in \mathcal{V}_{h'}(x)\).
- \(OA \in \mathcal{V}_h(x)\) if and only if for every history \(h',\) if \((h, h') \in Ob(x)\) then \(A \in \mathcal{V}_{h'}(x)\).

We define the following abbreviation:
- \(\Diamond A = \lnot \Box \lnot A\)
4.3. Axiomatisation

The axiomatisation of the logic of privileges with deadlines is defined by the set of following elements:

- Axioms of the logic of temporized actions with request.
- \( \Box (A \rightarrow B) \rightarrow (\Box A \rightarrow \Box B) \)
- \( \Diamond (A \rightarrow B) \rightarrow (\Diamond A \rightarrow \Diamond B) \)
- \( \Box A \rightarrow A \)
- \( \Box A \rightarrow \Box \Box A \)
- \( \Diamond A \rightarrow \Box \Diamond A \)
- \( \Box A \rightarrow \Diamond A \)
- \( \Diamond A \rightarrow \Box A \)
- \( \Diamond A \rightarrow \Diamond A \)
- \( \Box A \rightarrow \Box A \)

- The following inference rules:
  - Inference rules of the logic of temporized actions.
  - If \( A \) is a theorem then \( \Box A \) is a theorem.
  - If \( A \) is a theorem then \( \Diamond A \) is a theorem.

Theorem 3 : The axiomatisation of the logic of privileges with deadlines is sound and complete with respect to the semantics given in section 4.2.

Sketch of Proof : It is straightforward to check that the axiomatisation of the logic of temporized actions is sound with respect to the semantics.

Regarding completeness, the approach is similar to the proof of theorem 1. Thus, let \( (H, \text{Nec}, \text{Ob}) \) be a structure defined as follows:

- \( H \) is the set of all histories defined in the proof of theorem 1.
- \( \text{Nec} \) is a total function that associates every integer \( x \) with a relation \( \text{Nec}(x) \) on \( H \) defined as follows:
  - \( (h_T, h'_T) \in \text{Nec}(x) \) if and only if, for every formula \( A \), if \( \bigcirc^x \Box A \in T \) then \( \bigcirc^x A \in T' \).
- \( \text{Ob} \) is a total function that associates every integer \( x \) with a relation \( \text{Ob}(x) \) on \( H \) defined as follows:
  - \( (h_T, h'_T) \in \text{Ob}(x) \) if and only if, for every formula \( A \), if \( \bigcirc^x \Box A \in T \) then \( \bigcirc^x A \in T' \).

One can verify that \( (H, \text{Nec}, \text{Ob}) \) is a model of the logic of privileges with deadlines and that for every formula \( A \), every integer \( x \) and every history \( h_T \) we have:

- \( A \in V_{h_T}(x) \) if and only if \( \bigcirc^x A \in T \).

We can then apply Lindenbaum’s lemma to the logic of privileges with deadlines and derive that if \( A \) is valid then \( A \) is a theorem.

5. Expression of a security policy with the logic of privileges with deadlines

In this section, we show how to use the logic of privileges with deadlines to express security policies. We first define obligations with deadlines and conditional privileges. We then formally show how to derive actual privileges from conditional privileges.

These different concepts are used to define a new security model called Nomad. We provide an example to illustrate how to specify a security policies using Nomad in section 7.

5.1. Obligation with deadline

As mentioned in the introduction, most obligation does not apply immediately and a violation occurs only after a deadline elapses.

Thus, we define the modality \( O^{\leq d} A \) for \( d \geq 0 \) as follows:

- \( O^{\leq d} A = O \bigcirc^{\leq d} A \)
- \( O^{\leq d} A \) is to be read “it is obligatory that \( A \) within a delay of \( d \) units of time”.

One can notice that we have \( O^{\leq 0} A = O A \) that is \( O A \) corresponds to an immediate obligation of doing \( A \).

5.2. Conditional privileges

Most privileges do not apply unconditionally. Thus, we need to model privileges that are only active in specific contexts. For this purpose, we define diadic operators \( O(A|C) \) and \( O^{\leq d}(A|C) \) as follows:

- \( O(A|C) = \Box (C \rightarrow O A) \)
- \( O^{\leq d}(A|C) = \Box (C \rightarrow O^{\leq d} A) \)

\( O(A|C) \) is to be read “in context \( C \), it is obligatory that \( A \)”. This definition says it is necessary (so this cannot be violated) if condition \( C \) holds, then \( A \) is obligatory.

\( O^{\leq d}(A|C) \) is to be read “in context \( C \), it is obligatory that \( A \) within a delay \( d \)”.

One can notice that we have \( O^{\leq 0}(A|C) = O(A|C) \)

We also define two other diadic operators \( F(A|C) \) and \( P(A|C) \) as follows:

- \( F(A|C) = \Box (C \rightarrow F A) \)
- \( P(A|C) = \Box (C \rightarrow P A) \)

\( F(A|C) \) is to be read “in context \( C \), it is permitted that \( A \)”.

One can notice that we have \( F(A|C) \leftrightarrow O(\neg A|C) \)

By contrast, we have not \( P(A|C) \leftrightarrow \neg F(A|C) \)

But we have \( (P(A|C) \land \Diamond C) \rightarrow \neg F(A|C) \)
5.3. Effective privileges

When a conditional privilege applies in a given context $C$, one can derive an effective privilege when context $C$ holds. To make a distinction between conditional privileges and effective privileges, we define modalities $\mathcal{F}_eA$ and $\mathcal{P}_eA$ as follows :
- $\mathcal{F}_eA = (\mathcal{F}(A|C) \wedge C)$
- $\mathcal{P}_eA = (\mathcal{P}(A|C) \wedge C)$

$\mathcal{F}_eA$ is to be read “there is an effective prohibition that $A$”. This definition says that if there is a conditional prohibition that $A$ in context $C$, then when this context holds, there is an explicit prohibition that $A$.

$\mathcal{P}_eA$ is to be read “there is an effective permission that $A$”. We also define $O^{\leq d}_eA$ and $O_eA$ as follows :
- $O^{\leq d}_eA = (O^{\leq d}(A|C) \wedge C) \vee (O^{(d+1)}_eA \wedge \neg A)$
- $O_eA$ is to be read “there is an effective obligation that $A$ within a delay $d$”. The definition says that when there is a conditional obligation $O^{\leq d}(A|C)$ and context $C$ holds, then an effective obligation associated with a timer $d$ is activated. The timer of this effective obligation is then decremented while $A$ is not true.
- $O_eA = O^{\leq 0}_eA$

$O_eA$ is to be read “there is an effective immediate obligation that $A$”. It holds when the associated delay is equal to 0.

5.4. Expression of a security policy

A security policy specified in the Nomad security model is simply a set of conditional privileges of the form $\mathcal{F}(A|C)$, $\mathcal{P}(A|C)$, $O(A|C)$ and $O^{\leq d}(A|C)$ expressed in the language of privileges with deadline.

5.5. Modelling the tranquility principle

The tranquility principle is an optional assumption that guarantees that the security policy does not change over time. In this section, we investigate how to specify the tranquility principle in the logic of privileges with deadlines.

For this purpose, we add the following assumption in the semantics :
- For every integer $x$, $Ob(x) = Ob(x + 1)$
- For every integer $x$, $Nec(x) = Nec(x + 1)$

The counter-part in the axiomatisation corresponds to the two following axioms :
- $O \oplus A \leftrightarrow \oplus O A$
- $\Box A \leftrightarrow \Box \Box A$

One can notice that, with the tranquility principle, we can derive the following theorems :
- $O(\oplus A|C) \leftrightarrow \oplus O(A|C)$
- $\mathcal{F}(\oplus A|C) \leftrightarrow \oplus \mathcal{F}(A|C) \wedge C$
- $\mathcal{P}(\oplus A|C) \leftrightarrow \oplus \mathcal{P}(A|C) \wedge C$

We also have :
- $O^{\leq d}(\oplus A|C) \leftrightarrow \oplus O^{\leq d}(A|C) \wedge C$
- $O^{\leq d}(\oplus A|C) \rightarrow O^{\leq d+1}(A|C)$

Notice that there is no equivalence for the last theorem. In particular, doing $A$ immediately will fulfill obligation $O^{\leq d+1}(A|C)$ but not obligation $O^{\leq d}(\oplus A|C)$.

6. Expression of security properties in Nomad

6.1. Conflict

A first security property to be checked is the absence of conflict. We shall actually consider two different situations of conflict : conflict between permission and prohibition and conflict between obligation and prohibition. Notice that conflicts cannot derive from combining permissions with obligations.

There is no explicit conflict between permission and prohibition if, for every time $d$, we can derive the following theorem :
- $O^d$(\mathcal{F}_eA \land \mathcal{P}_eA)$

There is no explicit conflict between obligation with deadline and prohibition if, for every times $d$ and $d'$, we can derive the following theorem :
- $O^d$(\mathcal{F}_eA \land O^{\leq d'}_eA)$

Notice that in both theorems, $\mathcal{F}_eA$ may correspond to the interdiction to simultaneously execute some conflicting actions. For instance, we may have a prohibition to simultaneously execute action open_account (opening an account) and close_account (closing an account). In this case, the non conflict theorems enforce that it should not be possible to derive that it is permitted or obligatory to simultaneously execute actions open_account and close_account.

These theorems guarantee absence of conflict between explicit permission and prohibition. Next step will be to define other conditions that provide means to manage conflict between conditional privileges of the form $\mathcal{F}(A|C)$ and $\mathcal{P}(A|C)$. This is not the purpose of this paper to define such an approach to manage conflict but see [10] for the case of atomic actions without obligations. We plan to refine [10] to also deal with non atomic actions and obligations.

6.2. Access control properties

Access control properties define situations in which starting an action should be accepted. There are traditionally two different cases called closed policy and open policy.

In the case of a closed policy, the access control property is satisfied if, for every time $d$ and every action $\alpha$, we can derive the following theorem :
- $O^d(start(\alpha) \rightarrow \mathcal{P}_e start(\alpha))$ (AC1)
In the case of an open policy, the access control property is satisfied if, for every time \( d \) and every action \( \alpha \), we can derive the following theorem:
\[
\bigcirc^d(\text{start}(\alpha) \rightarrow \neg F_c\text{start}(\alpha)) \quad \text{(AC2)}
\]
Notice that if there is no conflict between permission and prohibition, it is easy to check that the closed policy provides a stronger requirement than the open policy in the sense that theorem (AC1) implies theorem (AC2).

### 6.3. Abiding by prohibitions

Access control properties only control execution of actions. We can actually consider generalization of these properties to every formula.

In the case of a closed policy, this would lead to the following requirement for every time \( d \):
\[
\bigcirc^d(A \rightarrow P_e.A)
\]
This is a too strong requirement because in this case every change in the system should be permitted.

On the other hand, in the case of an open policy, this would lead to the following requirement for every time \( d \):
\[
\bigcirc^d(A \rightarrow \neg F_e.A)
\]
This requirement seems perfectly acceptable and we call it the abiding by prohibitions requirement.

### 6.4. Obligation fulfilment and violation condition

In this section, we define requirements associated with obligation fulfilment. To specify these requirements, we first define the two modalities \( \text{fulfill}(A) \) and \( \text{violation}(A) \) as follows:
\[
\begin{align*}
\text{fulfill}(A) &= \bigcirc_e^dA \land A \\
\text{violation}(A) &= \bigcirc_e^0A \land \neg A
\end{align*}
\]
These modalities respectively define that \( A \) is fulfilled when \( A \) is obligatory within a delay \( d \) and \( A \) is true; \( A \) is violated when there is an obligation of doing \( A \) and the deadline elapses without \( A \) being done.

The security property associated with obligation fulfilment simply states that, for every \( d \), there is no violation:
\[
\begin{align*}
\bigcirc^d(\neg \text{violation}(A)) \\
\text{which is equivalent to} \\
\bigcirc^d(\bigcirc_e^0A \rightarrow A)
\end{align*}
\]
that is, when the delay elapses, then the only way to avoid a violation is to immediately fulfill the obligation.

### 7. Example

We present a short example to illustrate how to express a security policy using the Nomad security model. This example only contains atomic privileges, that is privileges that apply to atomic actions. We shall deal with non atomic privileges in the next section.

In this example, we use the following atomic actions: \( \text{open\_account} \) (opening a user account), \( \text{block\_account} \) (blocking a user account), \( \text{change\_pwd} \) (changing a password), \( \text{notify\_violation} \) (notifying that a violation occurs).

We also use the following propositions: \( \text{exist\_account} \) and repeated\_violation.

If \( n \) is an integer, then \( nD \) represents a delay of \( n \) days whereas \( nH \) represents a delay of \( n \) hours.

The policy is defined by the following rules:
\[
\begin{align*}
\&\quad \bigcirc_e^0(\text{start}(\text{open\_account}) \rightarrow \neg \text{exist\_account}) \\
\&\quad \text{If an account does not exist, then it is permitted to open it.} \\
\&\quad \bigcirc_e^0(\text{start}(\text{change\_pwd}) \land \text{exist\_account}) \\
\&\quad \text{When an account exists, it is permitted to change its password.} \\
\&\quad \bigcirc_e^{\leq 1D}(\text{start}(\text{open\_account}) \land \neg \text{exist\_account} \land \neg \text{req}(\text{open\_account})) \\
\&\quad \text{When there is a request to open an account whereas the account does not exist, then there is an obligation to open it within a delay of one day. This corresponds to an availability requirement: subject responsible for opening the account must be available so that the account will be open before the delay of one day.} \\
\&\quad \bigcirc_e^{\leq 1H}(\text{done}(\text{open\_account}) \land \text{start}(\text{open\_account})) \\
\&\quad \text{When one starts opening an account, it must be open within a delay of one hour. This requirement is an example of user contract [22]. This contract provides a guarantee that a given action (here opening an account) will be finished within a given delay once this action is initiated.} \\
\&\quad \bigcirc_e^{\leq 1D}(\text{start}(\text{change\_pwd}) \land \text{done}(\text{open\_account})) \\
\&\quad \text{Once the account is open, there is an obligation to change its password within a delay of one day.} \\
\&\quad \bigcirc_e^{\leq 30D}(\text{start}(\text{change\_pwd}) \land \text{done}(\text{change\_pwd})) \\
\&\quad \text{There is an obligation to change its password every thirty days. That is, when one changes its password, there is an obligation to change its password again within the next thirty days.} \\
\&\quad \bigcirc_e^{\leq 2D}(\text{start}(\text{change\_pwd}) \land \neg \text{violation}(\text{start}(\text{change\_pwd}))) \\
\&\quad \text{If the obligation to change its password is violated, there is an immediate obligation to notify the violation.} \\
\&\quad \bigcirc_e^{\leq 2D}(\text{start}(\text{change\_pwd}) \land \text{violation}(\text{start}(\text{change\_pwd}))) \\
\&\quad \text{If the obligation to change its password is violated, there is still an obligation to change its password within a delay of two days. This requirement and the previous one are called contrary to duty obligations (see [19] for instance), that is obligations that apply when other obligations or prohibitions are violated.}
\end{align*}
\]
8. Decomposition of non atomic privileges

Our model provides means to specify non atomic privileges. For instance, we may have:

- \( \mathcal{O}(\text{start(block\_account)} \mid \text{repeated\_violation}) \)
- \( \exists \text{\_account} \iff (\text{done(open\_account)} \lor \neg \text{\_account}) \)

An account exists if and only if the action of opening the account is finished or the action of blocking the account is not finished and the account was existing in the previous state.

- \( \text{repeated\_violation} \iff (\text{violation(start(change\_pwd))} \land \square^{\neg \text{fail}} \text{violation(start(change\_pwd))}) \)

There is a repeated violation if and only if there is a violation to change its password and this violation already occurred two days ago (which corresponds to the extension granted when a first violation occurs).

8.2. Decomposition of non atomic permissions

Regarding permissions, we have the following theorems:

- \( \mathcal{P}(A \land B) \rightarrow (\mathcal{P}(A) \land \mathcal{P}(B)) \) (Th2)
- \( (\mathcal{P}(A) \land \mathcal{O}(B)) \rightarrow \mathcal{P}(A \land B) \) (Th3)

Notice that, in both cases, there is no equivalence. In particular, since there is no equivalence in (Th2), we can show that the formula \((\mathcal{P}(A/C) \land \mathcal{P}(B/C)) \land \mathcal{F}(A \land B/C)\) is satisfiable. This is illustrated by examples of Chinese Wall security policy [7] that can be expressed in our model as follows:

- \( \mathcal{P}(\text{start(access\_coca)} \mid \text{create\_account}) \)
- \( \mathcal{P}(\text{start(access\_pepsi)} \mid \text{create\_account}) \)
- \( \mathcal{P}(\text{start(access\_coca)} \land \text{access\_pepsi}) \mid \text{create\_account}) \)

Let us now try to apply (Th2) to decompose non atomic action \(\alpha;\beta\). We shall obtain:

- \( \mathcal{P}(\text{start}(\alpha;\beta)) \rightarrow (\mathcal{P}(\text{start}(\alpha)) \land \mathcal{P}(\square^{\alpha}\text{start}(\beta))) \)

This is a too weak decomposition because permission \(\square^{\alpha}\text{start}(\beta))\) provides means to execute \(\beta\) without having executed \(\alpha\). However, we have:

- \( \mathcal{P}(\square^{\alpha}\text{start}(\beta)) \rightarrow \mathcal{P}(\square^{\alpha}\text{start}(\beta) \mid (C \land \text{start}(\alpha))) \)

Thus, we can suggest decomposing the non atomic permission \(\mathcal{P}(\text{start}(\alpha;\beta))\) by the following atomic permissions:

- \( \mathcal{P}(\text{start}(\alpha)) \)
- \( \mathcal{P}(\square^{\alpha}\text{start}(\beta) \mid (C \land \text{start}(\alpha))) \)

Now, the second permission specifies that there is a permission to execute \(\beta\) once \(\alpha\) has been executed.

In the case of a non atomic action \(\alpha\&\beta\), we can decompose the non atomic permission \(\mathcal{P}(\text{start}(\alpha;\beta))\) by the following atomic permissions:

- \( \mathcal{P}((\text{start}(\alpha)) \mid C) \)
- \( \mathcal{P}(\text{start}(\beta)) \)

However, we may have still some problems with these decompositions. This is because, in both cases, one can execute action \(\alpha\) without executing \(\beta\). This is sometimes problematic when action \(\alpha\) should not be executed separately from action \(\beta\).

To distinguish these cases from the previous ones, we shall use the following notations: \(\alpha;\beta\) and \(\alpha\&\beta\). The semantics of \(\alpha;\beta\) and \(\alpha\&\beta\) are respectively similar to \(\alpha;\beta\) and \(\alpha\&\beta\). The only difference is in the way permissions are decomposed.
Decomposition of permissions associated with $\alpha \land \beta$ and $\alpha \land \beta$ is based on theorem (Th3). Thus, $P(start(\alpha; \beta)|C)$ is decomposed as follows:
- $P(start(\alpha)|C)$
- $O((\land_{\alpha \land \beta} start(\beta)|(C \land start(\alpha)))$

and $P(start(\alpha; \beta)|C)$ is decomposed as follows:
- $P(start(\alpha)|C)$
- $P(start(\beta)|C)$
- $O(start(\beta)|(C \land start(\alpha)))$
- $O(start(\alpha)|(C \land start(\beta)))$

For instance, the following non-atomic permission:
- $P(start(open\_account & notify\_open\_account)|$ $\neg exist\_account)$
is decomposed into the two following atomic permissions:
- $P(start(open\_account)|\neg exist\_account)$
- $P(start(notify\_open\_account)|\neg exist\_account)$

On the other hand, the non-atomic permission:
- $P(start(open\_account; change\_pwd) |$ $\neg exist\_account)$
is decomposed into the two following atomic permissions:
- $P(start(open\_account)|\neg exist\_account)$
- $O((open\_account)|\neg exist\_account)$
- $O((change\_pwd)|\neg exist\_account)$

8.3. Decomposition of prohibitions

Regarding prohibition, we have the following theorem:
- $(F(A|C) \lor F(B|C)) \rightarrow F(A \land B|C)$

Notice that the inverse is not true. As mentioned before, since the formula $(P(A|C) \land P(B|C) \land F(A \land B|C))$ is satisfiable, this theorem cannot be used to decompose non-atomic prohibitions into atomic ones.

Thus, we follow another approach. We actually suggest decomposing prohibition of the form $F(A \land B|C)$ into prohibition $F((A \land B|C)$ this means that if the initial prohibition specifies that, if in context $C$, $A$ and $B$ are both prohibited then we can decompose it in a prohibition that says that, in context $C$ and $B$, $A$ is prohibited.

When applied to prohibition having the form $F(start(\alpha; \beta)|C)$, we obtain the following prohibitions:
- $F(start(\alpha)|(C \land start(\beta)))$
- $F(start(\beta)|(C \land start(\alpha)))$

Notice that this decomposition may lead to inconsistency if, on the other hand, we have also: $P(start(\alpha)|C)$ and $P(start(\beta)|C)$ (which corresponds to the case of a Chinese Wall security policy). This is because we have:
- $P(start(\alpha)|C) \rightarrow P(start(\beta)|(C \land start(\beta)))$

We plan to solve this inconsistency by considering that prohibition $F(start(\beta)|(C \land start(\alpha)))$ has higher priority than permission $P(start(\alpha)|C)$. Thus this prohibition takes precedence over the permission $P(start(\alpha)|C)$ when execution of action $\alpha$ starts. As mentioned earlier, this is not the purpose of this paper to discuss management of such conflicting privileges, but see [10] for further details.

Similarly, when applied to prohibition having the form $F(start(\alpha; \beta)|C)$, we obtain the following prohibition:
- $F((\land_{\alpha \land \beta} start(\beta)|(C \land start(\alpha)))$

For instance, let us consider the following prohibition:
- $F(start(block\_account; open\_account)|$ $repeated\_violation)$

When a repeated violation occurs, it is prohibited to block the account and then immediately reopen it.

This non-atomic prohibition is decomposed into the following atomic prohibition:
- $F((block\_account)| start(open\_account) |$ $repeated\_violation) \land start(block\_account)$

8.4. Decomposition of obligations with deadlines

Finally, decomposition of obligations with deadlines is based on the following theorem:
- $O^{\leq d}(A \land B|C) \rightarrow (O^{\leq d}(A|C) \land O^{\leq d}(B|C))$

However, since the inverse is not true, this is a weak decomposition. More precisely, the decomposition does not guarantee that the obligations of $A$ and $B$ will be fulfilled at the same time.

Thus, we add to the decomposition the two following additional obligations:
- $O(B|fulfill(A|C))$
- $O(A|fulfill(B|C))$

that is there is an immediate obligation to do $B$ once obligation to do $A$ in context $C$ is fulfilled (and similarly for $A$ when obligation to do $B$ in context $C$ is fulfilled).

Applying this approach, the obligation $O^{\leq d}(start(\alpha; \beta)|C)$ is decomposed into the four following obligations:
- $O^{\leq d}(start(\alpha)|C)$
- $O^{\leq d}(start(\beta)|C)$
- $O((open\_account)|fulfill(start(\beta)|C))$
- $O(start(\beta)|fulfill(start(\alpha)|C))$

Similarly, the obligation $O^{\leq d}(start(\alpha; \beta)|C)$ is decomposed into the three following obligations:
- $O^{\leq d}(start(\alpha)|C)$
- $O^{\leq d}(start(\beta)|C)$
- $O((open\_account)|fulfill(start(\alpha)|C))$

For instance, the following obligation:
- $O^{\leq 1d}(start(open\_account; notify\_open)|$ $\neg exist\_account \land req(open\_account))$
is decomposed into the three following obligations:

2 Notice that we slightly change the definition of modality $fulfill$ to keep track of the context in which the obligation is fulfilled:
- $fulfill(A|C) = O^{\leq d}(A|C) \land A$
- $O^{\leq d}(A|C)$ is defined as follow:
- $O^{\leq d}(A|C) = (O^{\leq d}(A|C) \land A) \lor (O^{\leq d}(d+1)(A|C) \land \neg A)$
\[\begin{align*}
\mathcal{O}^{\leq 1D}(\text{start}(\text{open\_account})) & \land \neg \text{exist\_account} \\
\mathcal{O}^{\leq 1D}(\text{notify\_open}) & \land \neg \text{exist\_account} \\
\mathcal{O}^{\leq 1D}(\text{notify\_open}) & \land \text{req}(\text{open\_account})
\end{align*}\]

9. Comparison with related work

In [17], J. Meyer defines a deontic logic that applies to actions. He models formulae of the form \(O(\alpha)\) (obligation to do \(\alpha\)), \(F(\alpha)\) (prohibition to do \(\alpha\)) and \(P(\alpha)\) (permission to do \(\alpha\)). His logic is defined as a variant of dynamic logic. He considers non atomic actions \(\alpha; \beta\) (\(\alpha\) followed by \(\beta\)), \(\alpha \lor \beta\) (\(\alpha\) or \(\beta\)) and \(\bar{\alpha}\) (abstention of doing \(\alpha\)). He also uses a special proposition \(V\) that represents a situation of violation. The obligation, prohibition and permission to do \(\alpha\) are then defined as follows:

- \(O(\alpha) = [\alpha]V\)
- \(F(\alpha) = [\alpha]V\)
- \(P(\alpha) = \langle \alpha \rangle \neg V\)

Thus, deontic concepts are reduced to the notion of violation. Such an approach is called reductionist. This is a first major difference with the logic presented in this paper since we first define the deontic concepts using a Krypke model and then specify situations of violation. There are also many differences with the axiomatisation suggested in [17]. For instance, regarding \(O(\alpha; \beta)\), we have the following theorem:

- \(O(\alpha; \beta) \iff O(\alpha) \land [\alpha]O(\beta)\)

Thus, the obligation of doing \(\beta\) is only effective in the state resulting from the execution of \(\alpha\). In our approach, if there is now an obligation of doing \(\alpha\) followed by \(\beta\), we can derive that there is now an obligation of doing \(\beta\) after a delay equal to the execution of \(\alpha\).

[8] suggests an extension of [17] to model a deontic logic of deadlines. They consider formulae of the form \(O_a(A \leq D)\) to be read agent \(a\) is obliged to do \(A\) before deadline \(D\). The suggested semantics is defined using a reductionist approach based on the branching time temporal logic CTL (Computation Tree Logic) [11]. Notice that a deadline is represented as a logical formula instead of a concrete duration as we suggested in this paper. Modelling deadline by formula is interesting and may be viewed as an abstraction of our model.

In the logic of privileges with deadlines, we only consider simple constraints between actions, namely that a given action is executed before another action (sequence) or that two actions are executed in parallel. In [5], Bertino et al. consider more complex constraints that would be useful to manage task executions in a workflow system. Extended our logic to consider such more complex constraints represent further work that remains to be done.

In [1], Atluri and Huang suggests an implementation of workflow system using Petri Nets. Execution of the different tasks that are part of a workflow system is constrained by temporal intervals. Enforcement of these temporal constraint is controlled be a temporalized colored Petri Net. Specifying these constraints may be easily integrated in our model so that Petri Net is a candidate for an implementation of our approach.

10. Conclusion

In this paper, we have formally defined Nomad, a new security model that provides means to specify permissions, prohibitions and also obligations that apply to non atomic actions. We analyze how these complex privileges can be decomposed into more elementary privileges. Regarding obligations, we notice that they generally do not apply immediately but one has some time to fulfill them. Thus, we model obligations with deadlines and define situations of violation in this case. We also show how to specify further obligations that could apply when primary obligations are violated. For instance, when an obligation to change a password is violated, then we could specify that there is an obligation to block the account if the password remains unchanged. These kinds of requirements are called contrary to duty [19] and we guess they are important when the security policy includes obligations.

Continuation of this work will consist in integrating the approach in a global methodology to be used to specify abstract security policies and decompose them to obtain concrete security requirements. Our objective is actually to integrate this work in the Organization Based Access Control (Or-BAC) model [15]. For this purpose, we need to extend the logic of privileges with deadlines suggested in this paper to be able to consider various concepts of Or-BAC, in particular the concepts of organization, role, activity and view. We also plan to extend the approach suggested in [10] so that it will be possible to manage conflicts between privileges (including obligations) on non atomic actions.

Another direction for future work will be to develop a framework to prove security properties specified in the Nomad model. For this purpose, we plan to use Uppaal [16], a tool for model-checking in a subset of CTL. We are especially interesting in specifying and analyzing availability properties. This will be presented in a forthcoming paper.

Acknowledgement. This work was supported by funding from the French ministry for research under “ACI Sécurité Informatique 2003-2006. Projet DISPO”
Références


