Security Policy Compliance with Violation Management

Julien Brunel
Institut de Recherche en Informatique de Toulouse
Toulouse, France

Frédéric Cuppens
Ecole Nationale Supérieure des Télécommunications de Bretagne
Rennes, France

Nora Cuppens-Boulahia
Ecole Nationale Supérieure des Télécommunications de Bretagne
Rennes, France

Thierry Sans
Ecole Nationale Supérieure des Télécommunications de Bretagne
Rennes, France

Jean-Paul Bodeveix
Institut de Recherche en Informatique de Toulouse
Toulouse, France

ABSTRACT

A security policy of an information system is a set of security requirements that correspond to permissions, prohibitions and obligations to execute some actions when some contextual conditions are satisfied. Traditional approaches consider that the information system enforces its associated security policy if and only if actions executed in this system are permitted by the policy (if the policy is closed) or not prohibited (if the policy is open) and every obligatory actions are actually executed in the system (no violation of obligations). In this paper, we investigate a more sophisticated approach in which an information system specification is compliant with its security policy even though some security requirements may be violated. Our proposal is to consider that this is acceptable when the security policy specifies additional requirements that apply in case of violation of other security requirements. In this case, we formally define conditions to be satisfied by an information system to comply with its security policy. We then present a proof-based approach to check if these conditions are enforced.

Categories and Subject Descriptors

D.2.4 [**Software Engineering**]: Software/Program Verification—*Formal methods, Model checking*; K.6.5 [**Management of computing and Information Systems**]: Security and Protection—*Unauthorized access*

General Terms

Algorithms, Theory, Verification

Keywords

deontic logic, labeled kripke structure, security policy, temporal logic, violation

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
FMSE'07, November 2, 2007, Fairfax, Virginia, USA.
Copyright 2007 ACM 978-1-59593-887-9/07/0011 ...$5.00.

1. INTRODUCTION

Modelling access control policies has been extensively investigated in the literature [12, 20, 1, 13, 14]. An access control policy corresponds to a set of permission rules which specifies which actions subjects are authorized to perform in the information system controlled by this policy. A permission may only apply when some contextual conditions are satisfied [17, 11, 22]. For instance, in a bank, the access control policy may specify that a clerk is permitted to grant a loan to a customer, *only if the amount of the loan is less than 50,000 euros*. The access control policy can also include prohibitions that are especially useful to specify exceptions to permissions.

More recently, it has also been suggested to consider other requirements in the security policy that correspond to obligations [3, 10]. For instance, the policy may specify that any user in the bank information system is obliged to change his or her password if this password has not been changed for more than 30 days. Models that consider obligations go beyond access control and are used to specify *usage* control requirements [18]. As an example of usage control requirement, the bank can specify in its policy that it is obligatory to stop the internet transaction of a bank customer if this user has been idle for more than one minute. In [15] the security rules and the system specification are expressed in the same formalism, Deontic Interpreted Systems. In this paper, we consider a security policy as a set of rules which apply to a separate system specification.

Once the security policy is specified, the next steps consist in (1) enforcing this security policy in the target system, this can be achieved by the deployment of the relevant security components and mechanisms in this system, and then (2) checking its compliance with this policy. For this purpose, traditional approaches consider that the system is compliant with permission rules if a transition can occur in the system only if the policy specifies that it is permitted to execute the action corresponding to the transition (closed policy principle) [21]. Another possibility would be to consider that a transition can only occur if the corresponding action is not prohibited by the policy (open policy principle). Regarding obligations, a simple approach would be to consider that if the policy specifies a requirement having the form $C \rightarrow O(e)$ (where O represents the obligation modality, C is the condition that triggers the obligation, and e is an event that corresponds to the occurrence of a transition) then one can prove that the implemented system satisfies the requirement $C \rightarrow e$. Notice that in this case, using obliga-

tion is useless since the policy may directly specify the requirement $C \rightarrow e$.

In this paper, we suggest a more sophisticated approach. We characterize situations of *violation* of some security requirements and define conditions where the implemented system may be considered compliant with the policy even if some violations occur. Let us briefly illustrate the approach through an example. In the bank, let us consider a security rule which specifies that a customer is prohibited to perform an account withdrawal if the account balance is negative after the withdrawal. According to the business model of this bank, there may be several possible implementations of this security rule. A first compliant implementation would be, for some customers, to block any attempt to withdraw an amount higher than the account balance. This corresponds to an implementation which prevents violation.

For other customers, another implementation of this rule would be to accept violation of the rule and implement a sanction, for instance, the customer pays bank charges when his or her account balance is negative. In our approach, this second implementation is also compliant with the policy if the policy specifies that the customer is obliged to pay bank charges when the account balance is negative. The system implementation may in turn accept violation of this obligation and still remain compliant with the policy if it implements another sanction in case the customer does not pay the bank charges.

Notice that it may be sometimes difficult to enforce some obligation requirements without violation because they require an external intervention. For instance, a user will have to choose his or her new password. As a consequence, it is somewhat unavoidable to consider violation when implementing the obligation for the user to change his or her password if this password has not been changed for more than 30 days.

Even if aforementioned examples deal with violation of obligations or prohibitions, our approach applies to permissions as well. For instance, in a closed policy, one generally considers that *all* actions executed in the system must be permitted by the policy. In our approach, a system implementation will be considered compliant with its policy even if some actions are executed without being permitted but some sanctions apply in this case.

Our objective in this paper is first to define what it means for a system to be compliant with its security policy when some violations are possible. We also define an approach to design an information system which is compliant with its security policy. There are two different approaches for this purpose. A first approach is a proof-based approach. In this case, the security policy specification and the information system specification are provided as inputs and the objective is to prove that the system specification is compliant with the policy specification [8]. Another approach would be to get a functional specification of the information system as input and to weave the security requirements in the function system specification using, for instance, Aspect Oriented Programming (AOP) [23, 9]. In this paper, we follow a proof-based approach.

The remainder of the paper is organized as follows. Section 2 further motivates the objective of our work and details the different situations in which a system specification is considered compliant with its security policy. Section 3 presents a model for the system specification based on finite state automata. Section 4 defines a language to specify the security policy. Section 5 presents a formal definition of the compliance of a system with a security policy which allows certain kinds of violations. An algorithm that checks the compliance is provided in section 6. Section 7 illustrates the approach through an example and section 8 concludes the paper.

2. MOTIVATIONS

We want to consider independently a system, a policy, and then to determine whether the system is compliant with the policy. The first way to deal with this problematics is to consider the system as a set of possible behaviours, the policy as a set of authorized/correct behaviours, and then to check whether the system is included in the policy. Two main approaches deal with this. A first way consists in representing the policy by a security automaton [21] and to compute the system in tandem with a simulation of the security automaton. Each step of the system generates an input symbol sent to the security automaton. If the security automaton is able to perform a transition on this input symbol, then the system is allowed to perform the step, else, the system is terminated. A second approach corresponds to model-checking techniques [8]. If the policy is modeled by a temporal formula and the system by an automaton, then we can check that the system enforces the policy. If it is not the case, some works (see, e.g. [16]) study the synthesis of a controller so that the conjunction of the initial system and the controller meets the policy, considered here as the system specification. Thus the role of the controller is to guide the system so that the specification is satisfied. In brief, in the first approach, the policy is viewed as a set of permitted behaviours, in the second approach it is viewed as a property the system has to satisfy (generally by using model checking techniques) or as the specification of the system (controller synthesis approach).

We aim to develop a framework that allows a richer analysis. Indeed, we want to be able to express in the policy that there is a sanction when some violation occurs. The sanction may be a new weak obligation which can also be violated, or a strong obligation which cannot be violated. More precisely, we deal with two kinds of obligations[2]: (1) a violable weak obligation that may trigger other obligations in case it is violated and (2) an unviolable obligation called strong obligation (a system which violates a strong obligation is not considered as *compliant* with its policy). Then, checking that the system respects the policy is refined: the system is said to be *compliant* with the policy if when a violation occurs, the sanction described in the policy is applied. However, it would be interesting to distinguish the case where no violation occurs, and the one where there is some, and the sanction is enforced. Indeed, both cases correspond to a "compliant" behaviour, but the first one can be considered as a "perfect" behaviour, whereas the second one is viewed as a contrary to duty behaviour[6] in which some violation has been made up for. So, instead of a yes/no answer, we would like to have a more subtle analysis, which aims at answering several questions:

- Is there any violation?
- If there is some violation, is a sanction specified?
- Is the sanction enforced?

Actually, the notion of enforcement has to be stated more precisely. During our study, we will find out that when a sanction is not enforced, it can be due to three different reasons: "some strong obligation is not fulfilled", "there is no specified sanction for some violation", or "there is an infinite sequence of unfulfilled weak obligations". So we need to specify five diagnostic cases. The first two cases correspond to a compliant behaviour ("there is no violation", and "every sanction is enforced"), and the three other cases correspond to different situations of non compliance.

Since violations occur with respect to prohibitions and obligations, we use deontic concepts to enrich the classical approach. So we naturally base our study on deontic logic whose aim is to ex-

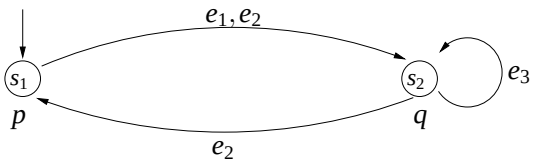

Figure 1: Labeled Kripke Structure

press and reason on permission, obligation, prohibition, and violation. In this paper, in order to simplify the discussion, we avoid the detection of internal conflicts in the policy by only considering permissions and obligations on positive events[1]. Prohibitions are not explicitly taken into account. Thus, any permission/prohibition or obligation/prohibition conflict cannot appear. However, we assume that the absence of permission corresponds to an implicit prohibition: an event that occurs without permission corresponds to a violation, i.e. we consider that *the policy is closed*. Moreover, we consider that an event which is explicitly obligatory in the policy is implicitly permitted, which is a usual hypothesis of deontic reasoning[24].

3. BEHAVIOUR MODEL

In this section, we define the model used to describe a system. It corresponds to usual automata with both labels on states and transitions. These automata are called Labeled Kripke Structures (*LKS*) and are used as models of the state/event extension of *LTL* [5]. It allows us to reason on both propositions and action-like atoms called events. Unlike actions in dynamic logic, events are at the same level as propositions in the syntax of the logic. Formula are actually interpreted on a trace of an *LKS*. If p is a proposition, and e an event, then $p \wedge e$ is true if p is true in the current state of the trace, and e is going to be performed next. Moreover, events have no duration, and are atomic. There are no new combinators as for actions in dynamic logic (sequence, choice, iteration). However, we can combine them using the usual logical operators, e.g., $e_1 \wedge e_2 \wedge Xe_3$ means that e_1 and e_2 occur simultaneously during the next transition, and e_3 will occur during the following one.

Such an automaton over the sets $(P, \mathcal{E})$ of propositions and events is defined as a tuple (S, S_0, Δ, V) where

- S is the set of states,
- $S_0 \subseteq S$ is the set of initial states,
- $\Delta \subseteq S \times 2^{\mathcal{E}} \times S$ is a transition relation, where $(s, E, s') \in \Delta$ if there is a transition from s to s' labelled by E, which means that all the events in E occur simultaneously during this transition
- $V : S \rightarrow 2^P$ is the valuation function which associates each state with a set of atomic propositions

Notice that the definition of the transitions here slightly differs from [5], where two given states cannot be related by several distinct transitions. A state s may have several successors by E because several distinct outgoing transitions may be labeled by E.

Figure 1 shows an illustration of an *LKS* where the atomic propositions are p and q, the events are e_1, e_2, and e_3, and the initial state is s_1.

[1]For a discussion of conflict detection and management, see [7]

DEFINITION 1 (SYNTAX OF *SE-LTL*). *Given a set P of atomic propositions, and a set $\mathcal{E}$ of events, the language $\mathcal{L}_{SELTL}$ of* State/Event Linear Temporal Logic *(SE-LTL) is defined as follows*

$$\begin{aligned} \mathcal{L}_{SELTL} ::= \quad & P \mid \mathcal{E} \mid \bot \mid \mathcal{L}_{SELTL} \Rightarrow \mathcal{L}_{SELTL} \\ & \mid X\mathcal{L}_{SELTL} \mid \mathcal{L}_{SELTL} U \mathcal{L}_{SELTL} \end{aligned}$$

The informal meaning of the temporal operators are as follows:
$X\varphi$: "*at the next moment, φ will hold.*"
$\varphi_1 U \varphi_2$: "*φ_2 will eventually hold at some moment m, while φ_1 holds from now until the moment before m*"

The boolean operators are defined as usual :

$$\neg\varphi \stackrel{def}{=} \varphi \Rightarrow \bot \qquad \top \stackrel{def}{=} \neg\bot$$

$$\varphi_1 \vee \varphi_2 \stackrel{def}{=} \neg\varphi_1 \Rightarrow \varphi_2 \qquad \varphi_1 \wedge \varphi_2 \stackrel{def}{=} \neg((\neg\varphi_1) \vee (\neg\varphi_2))$$

The temporal operators F (finally, or eventually) and G (globally, or always) are defined as the following abbreviations:

$$F\varphi \stackrel{def}{=} \top U \varphi \qquad G\varphi \stackrel{def}{=} \neg F \neg\varphi$$

Given an *LKS* $\mathcal{A} = (S, S_0, \Delta, V)$, an *SE-LTL* formula is interpreted on a state/event trace $\sigma = (s_0, E_0, s_1, E_1, \ldots)$, which is an alternating sequence of states and event sets such that $s_0 \in S_0$, and $\forall i \in \mathbb{N}\ (s_i, E_i, s_{i+1}) \in \Delta$. We note σ_i (resp. σ^i) the i^{th} state (resp. event set) of σ. A trace is either infinite, or ends with a state which has no successor by Δ.

DEFINITION 2 (STATE/EVENT SEMANTICS). *Given an LKS $\mathcal{A} = (S, S_0, \Delta, V)$ over the sets $(P, \mathcal{E})$ of propositions and events, and a state/event trace σ, we define the satisfaction relation $\models$ for state/event propositional formulas as follows:*

$\sigma, i \models p$	*iff*	$p \in V(\sigma_i)$	*where* $p \in P$
$\sigma, i \models e$	*iff*	$e \in \sigma^i$	*where* $e \in \mathcal{E}$
$\sigma, i \not\models \bot$			
$\sigma, i \models \varphi_1 \Rightarrow \varphi_2$	*iff*	$\sigma, i \models \varphi_1$ *implies* $\sigma, i \models \varphi_2$	
$\sigma, i \models X\varphi$	*iff*	$\sigma, i+1 \models \varphi$	
$\sigma, i \models \varphi_1 U \varphi_2$	*iff*	$\exists i' \geqslant i$ *such that* $\sigma, i' \models \varphi_2$ *and* $\forall\, i \leqslant i'' < i' \quad \sigma, i'' \models \varphi_1$	

A trace σ satisfies φ iff its first state satisfies it : $\sigma \models \varphi$ *iff* $\sigma, 0 \models \varphi$.
An LKS satisfies φ iff all its traces satisfy it.

For instance, considering the *LKS* illustrated by Figure 1, the following holds:
$(s_1, \{e_1, e_2\}, s_2, \{e_3\}, s_2, \{e_2\}, \ldots) \models p \wedge e_1 \wedge e_2 \wedge Xe_3$.

4. SECURITY POLICY LANGUAGE

A policy is defined as a set of rules of the form $C \rightsquigarrow \varphi$, where C is a condition under which the rule is "triggered". φ is a deontic sentence, i.e., a permission ("*it is permitted that ...*"), or an obligation ("*it is obligatory that ...*"). Actually, we distinguish two kinds of obligations: strong obligations and weak obligations. As said in section 2, a violation of a weak obligation may trigger a sanction, which can also be violated if it is a weak obligation. All this violations may be performed by a system which is still considered as compliant with the policy, if the sanctions are eventually enforced. The policy language also allows to specify strong obligations: a system which violates such a strong obligation is not compliant with the policy.

The condition C can be either a propositional formula whose atoms are propositions (not events), or a violation formula $Viol(e)$ that expresses that there is a violation concerning the event e, or

the conjunction of both forms. Actually, two kinds of violations may occur: a (weakly) obligatory event that does not happen, or a non-permitted event that happens. So we have in fact two violation formulas $Viol_O(e)$, which means *"e was (weakly) obligatory in the previous state, but it did not occur"*, and $Viol_P(e)$, which means *"e has just happened without being permitted"*. We only consider in this paper immediate obligations, but we plan to take into account obligations with deadline, as suggested in [10, 4].

The right hand side φ of a rule is either of the form $\boldsymbol{P}(\varphi_{ev})$ (φ_{ev} is permitted), or $\boldsymbol{O}(\varphi_{ev})$ (φ_{ev} is weakly obligatory), or $\hat{\boldsymbol{O}}$ (φ_{ev}) (φ_{ev} is strongly obligatory) where φ_{ev} is a positive event formula, i.e., a propositional formula (without any negation) whose atoms are events.

Here are some examples of rules that could be specified in the security policy, with $P \stackrel{def}{=} \{p,q\}$ and $\mathcal{E} \stackrel{def}{=} \{e_1,e_2\}$:

- $p \wedge q \rightsquigarrow \boldsymbol{O}(e_1)$
  If p and q are true in the current state then there is a weak obligation that e_1 occurs next.
- $p \rightsquigarrow \boldsymbol{P}(e_1 \wedge e_2)$
  If p is true in the current state then it is permitted that e_1 and e_2 occur simultaneously next.
- $p \wedge Viol_O(e_1) \rightsquigarrow \boldsymbol{O}(e_2 \vee e_3)$
  If p is true and the obligation to perform e_1 was violated by the previous transition, then there is a weak obligation to perform e_2 or e_3.
- $p \wedge Viol_O(e_2 \vee e_3) \rightsquigarrow \hat{\boldsymbol{O}}(e_4)$
  If p is true and the obligation to perform e_2 or e_3 was violated by the previous transition, then the event e_4 must occur (and this cannot be violated).

Let us define more formally the rule language. We first define the language $\mathcal{L}_{\mathcal{E}}$ of (positive) event formulas (in the rest of the paper, φ_{ev} will denote an event formula):

$$\mathcal{L}_{\mathcal{E}} ::= \mathcal{E} \mid \mathcal{L}_{\mathcal{E}} \wedge \mathcal{L}_{\mathcal{E}} \mid \mathcal{L}_{\mathcal{E}} \vee \mathcal{L}_{\mathcal{E}}$$

The limitation to positive events (without any negation) is due to the choice not to reason on explicit prohibitions, as explained in section 2. With the negation in the language of events, we would be able to express the obligation not to perform an event, which is equivalent to the prohibition to perform this event.

The condition of a rule (left hand side of a rule) can be a propositional formula, a violation formula, or a conjunction of a propositional and a violation formula. We define the languages $\mathcal{L}_P$ and $\mathcal{L}_{viol}$ of the propositional and the violation formulas as follows (in the rest of the paper, φ_p will denote a propositional formula):

$$\mathcal{L}_P ::= P \mid \mathcal{L}_P \wedge \mathcal{L}_P \mid \mathcal{L}_P \vee \mathcal{L}_P \mid \neg \mathcal{L}_P$$
$$\mathcal{L}_{viol} ::= Viol_O(\mathcal{L}_{\mathcal{E}}) \mid Viol_P(\mathcal{L}_{\mathcal{E}})$$

The left hand side of a rule is then a condition, and the right hand side is a positive deontic formula (permission, weak obligation, or strong obligation).

In our framework, the obligation of a given event can be both violable if it appears as a weak sanction in some rule, and non violable if it appears as a strong sanction in some other rule. For instance, if $p \rightsquigarrow \boldsymbol{O}(e_2)$ and $Viol_O(e_1) \rightsquigarrow \hat{\boldsymbol{O}}(e_2)$ are two rules of the policy, then in the context p there is an obligation to perform e_2, which can be violated, but if there is a violation of an obligation to perform e_1, then the obligation to perform e_2 is not violable.

In case there is a violation formula in the condition, then the right hand side is called *sanction*. Notice that a sanction is necessarily an (either weak or strong) obligation. Indeed, if the system violates a part of the policy, it does not make sense to give some permission as a sanction. We define the language $\mathcal{L}_{rule}$ of the rules as:

$$\begin{array}{rcl} \mathcal{L}_{rule} ::= & \mathcal{L}_P & \rightsquigarrow (\boldsymbol{O}(\mathcal{L}_{\mathcal{E}}) \mid \boldsymbol{P}(\mathcal{L}_{\mathcal{E}}) \mid \hat{\boldsymbol{O}}(\mathcal{L}_{\mathcal{E}})) \\ \mid & \mathcal{L}_P \wedge \mathcal{L}_{viol} & \rightsquigarrow (\boldsymbol{O}(\mathcal{L}_{\mathcal{E}}) \mid \hat{\boldsymbol{O}}(\mathcal{L}_{\mathcal{E}})) \end{array}$$

A policy is defined as a set of rules.

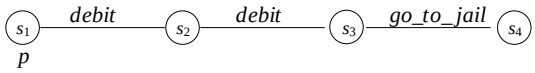

Figure 2: State/Event trace

5. COMPLIANCE OF A SYSTEM WITH ITS SECURITY POLICY

Given a model $\mathcal{A} = (S, S_0, \Delta, V)$ of a security system, and a policy, we focus on the meaning of the compliance of the system with the policy, in our context of violation management. We actually reason on a single trace of the system (the compliance of the whole system is then defined by the compliance of all its traces). We first give an intuition of the definition, and then propose a semantics based on traces, to formally specify these aspects.

5.1 Informal view

Roughly speaking, we say that a system is compliant with a policy if

- either there is no violation,
- or each time some violation occurs, the associated sanction is enforced.

To state more precisely the notion of enforcement, we introduce some vocabulary and a support example.

5.1.1 Example

As an example, we present a part of the behaviour of a bank customer. Consider the state/event trace illustrated by Figure 2 together with the policy consisting in the rules (1), (2), and (3).

1. $\neg p \rightsquigarrow \boldsymbol{P}(debit)$
2. $Viol_P(debit) \rightsquigarrow \boldsymbol{O}(pay\_charges)$
3. $Viol_O(pay\_charges) \rightsquigarrow \hat{\boldsymbol{O}}(go\_to\_jail)$

The first event (*debit*) is performed without permission since $\boldsymbol{P}(debit)$ cannot be deduced from the rules in the context of the state s_1 (recall we only work with closed policies). Therefore, in the state s_2, there is a violation $Viol_P(debit)$. So, according to rule (2) there is an obligation to perform the event *pay_charges* as a sanction. During the second transition, the event *pay_charges* is not performed, so the sanction is also violated. According to rule (3), there is then a strong obligation to perform *go_to_jail* as a second sanction, in the state s_3. It is effectively performed by the specified bank customer behaviour. Thus, according to our approach, this system is compliant with its security policy, even if security rules (1) and (2) are violated.

5.1.2 Vocabulary

We say that an obligation $\boldsymbol{O}(\varphi_{ev})$ is *fulfilled* if φ_{ev} holds in the current state, i.e., if the event formula φ_{ev} is going to be performed during the next transition. In the example of Figure 2, the (strong) obligation to perform *go_to_jail* is *fulfilled* in the state s_3.

A violation may *induce* several obligations. In the example, $Viol_P(debit)$ holds in the state s_2 (because of the first transition). Then rule (2) *triggers* the obligation $\boldsymbol{O}(pay\_charges)$. (Recall that an obligation which is *triggered* by a violation is also named *sanction*). Then, in s_3, because of the violation $Viol_O(pay\_charges)$, rule (3) *triggers* the *sanction* $\hat{\boldsymbol{O}}(go\_to\_jail)$. We say that the initial violation $Viol_P(debit)$ *triggers* the *sequence of sanctions* $[\boldsymbol{O}(pay\_charges),\ \hat{\boldsymbol{O}}(go\_to\_jail)]$.

A violation is called *managed* if every *sequence of sanctions* which is *triggered* by this violation ends with a *fulfilled* obligation. In the example, $Viol_P(debit)$ in s_2 is then *managed*.

When a violation is not *managed*, i.e., when it *triggers* some *sequence* of sanctions which does not end with a *fulfilled* obligation, we can distinguish three situations:

- Some *triggered sequence* of sanctions ends with a strong obligation which is not fulfilled, then the violation is called *ultimately strong*
- Some *triggered sequence* of sanctions ends with a weak obligation which is not fulfilled, and no sanction is specified by the policy in case this weak obligation is violated, then the (initial) violation is called *ultimately unexpected*
- There is an infinite *triggered sequence* of sanctions, then the violation is called *never caught*.

This last situation is illustrated by the system described in Figure 3, together with the following policy:

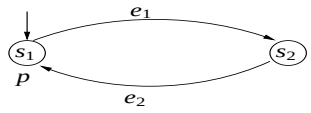

Figure 3: Infinite sequence of weak sanctions

(1) $p \rightsquigarrow \boldsymbol{O}(e_2)$

(2) $Viol_O(e_2) \rightsquigarrow \boldsymbol{O}(e_1)$

(3) $Viol_O(e_1) \rightsquigarrow \boldsymbol{O}(e_2)$

In the state s_1, according to rule (1), it is obligatory to perform e_2 since p is true. So, performing the event e_1 violates this obligation, and rule (2) triggers $\boldsymbol{O}(e_1)$ in the destination state s_2. Then, doing the event e_2 violates this obligation. The system is then back to the state s_1, where rule (3) triggers $\boldsymbol{O}(e_2)$. If we consider the infinite state/event trace generated by this automaton, then in every state of this trace, there is a violation which triggers some rule. Therefore, the initial violation $Viol_O(e_2)$ triggers the infinite sequence of sanctions $[\boldsymbol{O}(e_1),\ \boldsymbol{O}(e_2),\ \boldsymbol{O}(e_1),\ \boldsymbol{O}(e_2),\ \ldots]$.

Notice that this system also illustrates an *unexpected* violation. Indeed, in the state s_2, the violation $Viol_P(e_1)$ holds and no rule specifies a sanction.

Each obligation of a triggered sequence could be related to the previous one in a more complex way. At each step, several rules can be involved in the same time so that the next obligation is triggered. For instance, let us consider a policy which contains the following rules:

$$Viol_P(e_1) \rightsquigarrow \boldsymbol{O}(e_3) \quad Viol_O(e_2) \rightsquigarrow \boldsymbol{O}(e_4) \quad Viol_O(e_3 \wedge e_4) \rightsquigarrow \boldsymbol{O}(e_5)$$

Then, in a state which satisfies $Viol_P(e_1) \wedge Viol_O(e_2)$, the first two rules are triggered, and the corresponding sanction is $\boldsymbol{O}(e_3) \wedge \boldsymbol{O}(e_4)$, which is equivalent to $\boldsymbol{O}(e_3 \wedge e_4)$. If the next transition does not perform $e_3 \wedge e_4$, then the third rule triggers the sanction $\boldsymbol{O}(e_5)$ in the next state. So we see that the *conjunction* $Viol_P(e_1) \wedge Viol_O(e_2)$ of violations can trigger the sequence $[\boldsymbol{O}(e_3 \wedge e_4),\ \boldsymbol{O}(e_5)]$, whereas the single violation $Viol_P(e_1)$, for instance, cannot. So we have to take into account in the remainder that the different characterizations (*managed*, *ultimately strong*, *ultimately unexpected*, and *never caught*) do not only concern a single violation but a *conjunction* of violations.

To sum up, there are five different diagnostic cases to consider:

1. no violation occurs
2. every conjunction of violations is *managed*
3. some *ultimately strong* conjunction of violations occurs
4. some *ultimately unexpected* conjunction of violations occurs
5. some *never caught* conjunction of violations occurs

It is clear that the cases 1 and 2 correspond to a system which is compliant with its security policy and that the cases 3 and 5 correspond to a system which is not. The case 4 is less clear: one can consider that there is a lack in the policy specification, and the system is still compliant with the (specified part of the) policy. On the other hand, we cannot consider that the sanction is enforced (because no sanction is specified), so it is also reasonable to conclude that the system is not compliant with the security policy. In this paper, we will adopt the latter point of view.

DEFINITION 3 (COMPLIANCE). *A state/event trace is said to be compliant with a policy iff*

- *there is no violation*
- *or every conjunction of violation is managed*

A system is compliant iff all its traces are compliant.

5.2 Formal definitions

In this section, we enrich the trace semantics given in section 3 to interpret obligations, permissions, and violations in each state. This allows to formally define our diagnostic cases, and thus the compliance. Recall that φ_{ev} always denotes an event formula.

DEFINITION 4 (EXTENSION OF THE TRACE SEMANTICS). *Given a state/event trace σ, and a policy pol, we define the semantics of permission, weak and strong obligations, and violation.*

φ_{ev} is permitted in the i^{th} state of σ if there is a rule in the policy whose hypothesis is satisfied by this state, which states that φ'_{ev} is permitted, where φ_{ev} is a logical consequence (in propositional logic, considering events as propositions) of φ'_{ev}. We also consider that if φ_{ev} is explicitly obligatory, then it is implicitly permitted. (This is a usual assumption in deontic logic.) The formal semantics is then:

$$\begin{array}{l} \sigma, i \models_{pol} \mathbf{P}\varphi_{ev} \qquad iff \\ \exists (C \rightsquigarrow \mathbf{P}(\varphi'_{ev})) \in pol \ \ s.\ t.\ \ \sigma, i \models_{pol} C \ and\ \models \varphi'_{ev} \Rightarrow \varphi_{ev} \\ or \quad \sigma, i \models_{pol} \mathbf{O}\varphi_{ev} \quad or \quad \sigma, i \models_{pol} \hat{\mathbf{O}}\varphi_{ev} \end{array}$$

where $\models$ stands for validity in propositional logic.
φ_{ev} is weakly obligatory if the conjunction of all the weak obligations we can infer from the rules implies φ_{ev}:

$$\begin{array}{l} \sigma, i \models_{pol} \mathbf{O}\varphi_{ev} \ iff \\ \models \left(\bigwedge \ \{\varphi'_{ev} \,/\, \exists (C \rightsquigarrow \mathbf{O}(\varphi'_{ev})) \in pol \ \ s.\ t.\ \sigma, i \models_{pol} C\} \right) \quad \Rightarrow \quad \varphi_{ev} \end{array}$$

Similarly, we give the formal semantics of strong obligations:

[2] we define $\wedge S$, where S is a set, as $\bigwedge_{\phi \in S} \phi$

$\sigma, i \models_{pol} \hat{\mathbf{O}}\varphi_{ev}$ *iff*
$\models \big(\bigwedge \{\varphi'_{ev} \,/\, \exists (C \rightsquigarrow \hat{\mathbf{O}}(\varphi'_{ev})) \in pol \text{ s. t. } \sigma, i \models_{pol} C\}\big) \quad \Rightarrow \quad \varphi_{ev}$

The obligation to perform φ_{ev} is violated ($Viol_O(\varphi_{ev})$) if φ_{ev} was obligatory in the previous state and has not occurred during the previous transition. The prohibition (viewed as a lack of permission) to perform φ_{ev} is violated ($Viol_P(\varphi_{ev})$) if φ_{ev} has just occurred without being permitted. (Recall that, since we only consider closed policies, an event formula which is not explicitly permitted is considered prohibited.)

$\sigma, i \models_{pol} Viol_O(\varphi_{ev})$ *iff*
$i > 0$ *and* $\sigma, i-1 \models_{pol} \mathbf{O}(\varphi_{ev}) \wedge \neg\varphi_{ev}$
$\sigma, i \models_{pol} Viol_P(\varphi_{ev})$ *iff*
$i > 0$ *and* $\sigma, i-1 \models_{pol} \neg\mathbf{P}(\varphi_{ev}) \wedge \varphi_{ev}$

For propositions, events, boolean and temporal operators, we define the satisfaction relation as for SE-LTL (cf section 3).
We write $\sigma \models_{pol} \varphi$ *iff* $\sigma, 0 \models_{pol} \varphi$.

Before defining a whole sequence of sanctions which is triggered after a given conjunction of violations, we first consider one step. The set $Wsanction_*(i,\varphi)$ represents the formulas which are weakly obligatory in state i of a state/event trace σ because of the violation $Viol_*(\varphi)$ (where $* \in \{P, O\}$). In other words, obligations in state $i-1$ have not been fulfilled by the last transition, or non-permitted events have occur, and we consider the sanction which stands in state i. (σ is implicit in the following predicates for ease of reading.)

$Wsanction_*(i,\varphi) \stackrel{def}{=}$
$\{\phi \;/\; \exists(\varphi_p \wedge Viol_*(\psi) \rightsquigarrow \boldsymbol{O}(\phi)) \in pol$ such that
$\models \varphi \Rightarrow \psi$
and $\sigma, i \models_{pol} \varphi_p \wedge Viol_*(\psi)\}$
where $* \in \{\boldsymbol{O}, \boldsymbol{P}\}$.

Similarly, we define $Ssanction(i, \boldsymbol{O}(\varphi))$ as the set of the formulas which are strongly obligatory in state i because of the violation $Viol_*(\varphi)$:

$Ssanction_*(i,\varphi) \stackrel{def}{=}$
$\{\phi \;/\; \exists(\varphi_p \wedge Viol_*(\psi) \rightsquigarrow \hat{\boldsymbol{O}}(\phi)) \in pol$ such that
$\models \varphi \Rightarrow \psi$
and $\sigma, i \models_{pol} \varphi_p \wedge Viol_*(\psi)\}$
where $* \in \{\boldsymbol{O}, \boldsymbol{P}\}$.

DEFINITION 5 (SEQUENCE OF TRIGGERED SANCTIONS). *We consider finite and infinite sequences, indexed by I. In the former case, I has the form $0..N$ for some non-negative integer $N \in \mathbb{N}$. In the latter case, $I = \mathbb{N}$. The notation for a sequence of sanctions is $(\mathbf{O}_k)_{k\in I}$, where the last obligation (in case the sequence is finite) can be either a weak or a strong obligation, and all the other obligations are weak. A sequence $(\mathbf{O}_k)_{k\in I}$ of sanctions is said to be triggered by a conjunction $\bigwedge_{l\in 0..L} v_l$ of violation formulas in the i^{th} state of a state/event trace σ if*

(*i*) *obligations of the sequence hold in the successive states of σ:*

$$\forall k \in I \quad \sigma, i+k \models_{pol} \mathbf{O}_k$$

where each $\mathbf{O}_k$ is of the form $\mathbf{O}(\varphi_k)$, and the last obligation $\mathbf{O}_N$ (in case the sequence is finite) is of the form $\mathbf{O}(\varphi_N)$ or $\hat{\mathbf{O}}(\varphi_N)$

(*ii*) *the conjunction $\bigwedge_{l\in 0..L} v_l$ triggers the first sanction of the sequence:*

$$\models \Big(\bigwedge_{\phi \in sanctions(\bigwedge_{l\in 0..L} v_l)} \varphi\Big) \;\Leftrightarrow\; \varphi_0$$

where $\mathbf{O}_0 = \mathbf{O}(\varphi_0)$ *and*
$sanction(\bigwedge_{l\in 0..L} v_l) = \bigcup_{l\in 0..L} Wsanction_{*_l}(i, \phi_l)$
with $*_l = P$ *if* $v_l = Viol_P(\phi_l)$ *and* $*_l = O$ *if* $v_l = Viol_O(\phi_l)$

(*iii*) *every sanction $\mathbf{O}_k = \mathbf{O}(\varphi_k)$ in the sequence is triggered by the violation of the previous sanction $\mathbf{O}_{k-1} = \mathbf{O}(\varphi_{k-1})$:*

$$\forall k \in I\backslash\{0\} \quad \models \varphi_k \Leftrightarrow \bigwedge_{\varphi \in Wsanction_O(i+k, \varphi_{k-1})} \varphi$$
$$\text{or} \qquad k = N \text{ and } \models \varphi_k \Leftrightarrow \bigwedge_{\varphi \in Ssanction_O(i+k, \varphi_{k-1})} \varphi$$

(*iv*) *if the sequence is finite and the last sanction is weak, then,*

– *either the last sanction $\mathbf{O}_N = \mathbf{O}(\varphi_N)$ is fulfilled*

$$\sigma, i+N \models \varphi_N$$

– *or the policy specifies no sanction for $Viol_O(\varphi_N)$*

$$Wsanction_O(i+N+1, \varphi_N) = \emptyset$$

In the example of section 5.1, we can see that the sequence of sanctions $[\boldsymbol{O}(pay\_charges), \hat{\boldsymbol{O}}(go\_to\_jail)]$ is triggered by the violation $Viol_P(debit)$ in the state s_2, according to this definition.

We can now define the semantics of the different kinds of violation: managed violation, ultimately strong violation, ultimately unexpected violation, and never caught violation, through the predicates *managed*, *strong*, *unexpected*, and *never_caught*.

DEFINITION 6 (MANAGED VIOLATION). *A conjunction φ of violations is managed if every sequence of triggered sanctions is finite and ends with a fulfilled sanction.*

$\sigma, i \models_{pol} managed(\varphi)$ *iff*
every sequence of sanctions triggered by φ is finite and
$\forall (\mathbf{O}_k)_{k\in 0..N}$ *sequence of sanctions triggered by φ*
$\sigma, i+N \models_{pol} \varphi_N$ *where* $\mathbf{O}_N = \mathbf{O}(\varphi_N)$ *or* $\mathbf{O}_N = \hat{\mathbf{O}}(\varphi_N)$

DEFINITION 7 (ULTIMATELY STRONG VIOLATION). *A conjunction of violations is ultimately strong if there is a sequence of triggered sanctions which ends with an unfulfilled strong violation.*

$\sigma, i \models_{pol} strong(\varphi)$ *iff*
$\exists (\mathbf{O}_k)_{k\in 0..N}$ *sequence of sanctions triggered by φ*
such that $\mathbf{O}_N = \hat{\mathbf{O}}(\varphi_N)$ *and* $\sigma, i+N \models \neg\varphi_N$

DEFINITION 8 (ULTIMATELY UNEXPECTED VIOLATION). *A conjunction φ of violations is ultimately unexpected if the policy specifies no sanction for φ, or for some induced violation.*

$\sigma, i \models_{pol} unexpected(\varphi)$ *where φ has the form* $\bigwedge_{l\in 0..L} v_l$ *iff*
$\forall l \in 0..L \;\; \nexists (\varphi_{p_l} \wedge v_l \rightsquigarrow \mathbf{O}_l) \in pol$ *such that* $\sigma, i \models \varphi_{p_l} \wedge v_l$
or $\exists (\mathbf{O}_k)_{k\in 0..N}$ *sequence of sanctions triggered by φ such that*
$\sigma, i+N \models \neg\varphi_N$ *with* $\mathbf{O}_N = \mathbf{O}(\varphi_N)$ *and*
$Wsanction_O(i+N+1, \varphi_N) = Ssanction_O(i+N+1, \varphi_N) = \emptyset$

DEFINITION 9 (NEVER CAUGHT VIOLATION). *A conjunction of violations is never caught if it triggers an infinite sequence of sanctions.*

$\sigma, i \models_{pol} never\_caught(\varphi)$ *iff*
there is an infinite sequence of sanctions triggered by φ

PROPERTY 1. *A conjunction of violations which occurs without being* managed *is either* ultimately strong*, or* ultimately unexpected*, or* never caught.

Given a policy pol, a state/event trace σ*, a natural i, and a conjunction* φ *of violation formulas, then the following property holds*

$$\sigma, i \models_{pol} (\varphi \wedge \neg managed(\varphi)) \Leftrightarrow (strong(\varphi) \vee unexpected(\varphi) \vee never\_caught(\varphi))$$

The **proof** is omitted for the sake of brevity.

The specification of the five diagnostic cases is straightforward with the predicates defined above. Given a state/event trace σ and a policy *pol*, we reason on a conjunction φ of violation formulas:

1. There is no occurrence of φ
   $\sigma \models_{pol} G\ \neg\varphi$
2. Every occurrence of the violation φ is managed
   $\sigma \models_{pol} G\ (\varphi \Rightarrow managed(\varphi))$
3. There is an occurrence of φ which is a ultimately strong
   $\sigma \models_{pol} F\ strong(\varphi)$
4. There is an occurrence of φ which is unexpected
   $\sigma \models_{pol} F\ unexpected(\varphi)$
5. There is an occurrence of φ which is never caught
   $\sigma \models_{pol} F\ never\_caught(\varphi)$

So, a trace is compliant with its policy iff it satisfies $G\ \varphi \Rightarrow managed(\varphi)$ for every conjunction φ of violation formulas. (Notice that a trace which satisfies no violation is compliant, since $G\ \neg\varphi$ implies $G\ (\varphi \Rightarrow managed(\varphi))$.) From property 1 we deduce that a trace σ is not compliant iff there is some conjunction φ of violation formulas such that $\sigma \models F\ (strong(\varphi) \vee unexpected(\varphi) \vee never\_caught(\varphi))$. Recall that a system is compliant with its policy iff all its traces are such. Notice that although there is an infinite number of syntactically different event formulas, there is a finite number of semantically different event formulas, for a given set $\mathcal{E}$ of events.

In the next section, we provide an algorithm which checks the compliance of a system with respect to a policy, i.e. whether all its traces satisfy $G\ \varphi \Rightarrow managed(\varphi)$ for every conjunction φ of violation formulas. In case it is compliant, it indicates whether there is some violation, i.e. whether there is a trace which satisfies $F\ \varphi$. In case it is not compliant, for each of the three corresponding properties ($F\ strong(\varphi)$, $F\ unexpected(\varphi)$, and $F\ never\_caught(\varphi)$), it checks the existence of some trace and some conjunction φ of violation formulas that satisfies it.

6. DIAGNOSTIC ALGORITHM

We present an algorithm that analyses both the system and the policy to check whether the five following properties hold:

1. There is no violation
2. Every conjunction of violations is managed
3. There is some ultimately strong conjunction of violations
4. There is some ultimately unexpected conjunction of violations
5. There is some never caught conjunction of violations

Our algorithm starts from a given state considered as initial. In case there are several initial states, we have to call this algorithm for each one in order to analyse the whole system.

We refer to propositional logic, and suppose we have access to a decision procedure for the validity of any propositional formula. Given an *LKS* $\mathcal{A} = (S, S_0, \Delta, V)$ over the sets P and $\mathcal{E}$ of propositions and events, we call $\bar{s}$ the propositional formula that characterizes the atomic propositions that are true in $s \in S$, $\hat{E}$ the conjunction of all the events in $E \subseteq \mathcal{E}$, $succ(s,E)$ the set of the possible successors of s after a transition labelled by E, and $Out(s)$ the set of all the event sets that label an outgoing transition from s:

$$\bar{s} \stackrel{def}{=} \bigwedge_{p \in V(s)} p \wedge \bigwedge_{p \notin V(s)} \neg p \qquad \hat{E} \stackrel{def}{=} \bigwedge_{e \in E} e$$

$$succ(s,E) \stackrel{def}{=} \{s' \in S \ / \ (s,E,s') \in \Delta\}$$

$$Out(s) \stackrel{def}{=} \{E \in 2^{\mathcal{E}} \ / \ \exists s' \in S \text{ such that } (s,E,s') \in \Delta\}$$

We use some variables to "diagnose" the situation. *no_viol* is true until some violation occurs, *strong_violation* is false until some strong sanction is not fulfilled, *unexpected_violation* is false until some violation occurs for which no sanction is specified, and *never_caught* is false until some never caught violation is detected. The detection uses a variable *po* which records the triples (current state, current transition, parsed obligation). In order to have more precise information, we could provide the trace of the different rules that are triggered, but we do not consider such trace here for the sake of brevity.

We need to introduce some additional notations that make the algorithm more readable when it computes the obligations which stand in the current state according to the policy.

$WObl(s)$ (resp. $SObl(s)$) is the set of the formulas which are weakly (resp. strongly) obligatory in state s, and which are not sanctions (they are not triggered by any sanction).

$$WObl(s) \stackrel{def}{=} \{\phi \ / \ \exists(\varphi_p \rightsquigarrow \boldsymbol{O}(\phi) \in pol \text{ such that } \bar{s} \Rightarrow \varphi_p\}$$
$$SObl(s) \stackrel{def}{=} \{\phi \ / \ \exists(\varphi_p \rightsquigarrow \hat{\boldsymbol{O}}(\phi) \in pol \text{ such that } \bar{s} \Rightarrow \varphi_p\}$$

$Perm(s)$ is the sets of all the event sets which are permitted in state s. Notice that it may be the case that $\{e_1\}$ and $\{e_2\}$ are permitted whereas $\{e_1, e_2\}$, i.e., the simultaneous occurrence of e_1 and e_2, is not. Therefore, $Perm(s)$ is necessarily a set of event sets and not a set of events. We denote $\hat{Perm}(s)$ the event formula which characterizes permitted events.

$$Perm(s) \stackrel{def}{=} \{E' \subseteq E \ / \ \exists(\varphi_p \rightsquigarrow \boldsymbol{P}(\varphi_{ev})) \in pol \text{ s.t. } \models \varphi_{ev} \Rightarrow \hat{E} \text{ and } \models \bar{s} \Rightarrow \varphi_p\}$$
$$\hat{Perm}(s) \stackrel{def}{=} \bigvee_{E \in Perm(s)} \hat{E}$$

$WSanc_P(s,E)$ (resp. $SSanc_P(s,E)$) is the set of the weakly (resp. strongly) obligatory formulas which are triggered by the occurrence of the non-permitted set E of events.

$$WSanc_P(s,E) \stackrel{def}{=} \{\phi \ / \ \exists(\varphi_p \wedge Viol_P(\varphi_{ev}) \rightsquigarrow \boldsymbol{O}(\phi)) \in pol \text{ such that } \bar{s} \Rightarrow \varphi_p \text{ and } \hat{E} \Rightarrow \varphi_{ev} \text{ and } \hat{Perm}(s) \not\Rightarrow \varphi_{ev}\}$$
$$SSanc_P(s,E) \stackrel{def}{=} \{\phi \ / \ \exists(\varphi_p \wedge Viol_P(\varphi_{ev}) \rightsquigarrow \hat{\boldsymbol{O}}(\phi)) \in pol \text{ such that } \bar{s} \Rightarrow \varphi_p \text{ and } \hat{E} \Rightarrow \varphi_{ev} \text{ and } \hat{Perm}(s) \not\Rightarrow \varphi_{ev}\}$$

$WSanc_O(s,E,\varphi_{ev})$ (resp. $SSanc_O(s,E,\varphi_{ev})$) is the set of the weakly (resp. strongly) obligatory formulas which are triggered when a set E of events occurs and violates the obligation to perform φ_{ev}.

$$WSanc_O(s,E,\varphi_{ev}) \stackrel{def}{=} \{\phi \ / \ \exists(\varphi_p \wedge Viol_O(\varphi'_{ev}) \rightsquigarrow \boldsymbol{O}(\phi)$$

such that $\varphi_{ev} \Rightarrow \varphi'_{ev}$ and $\hat{E} \not\Rightarrow \varphi'_{ev}$ and $\bar{s} \Rightarrow \varphi_p\}$

$SSanc_O(s,E,\varphi_{ev}) \stackrel{def}{=} \{\phi \,/\, \exists(\varphi_p \wedge Viol_O(\varphi'_{ev}) \rightsquigarrow \hat{\boldsymbol{O}}(\phi)$
such that $\varphi_{ev} \Rightarrow \varphi'_{ev}$ and $\hat{E} \Rightarrow \varphi'_{ev}$ and $\bar{s} \Rightarrow \varphi_p\}$

The variables *wo* and *so* are used to record the formulas which are weakly and strongly obligatory in the current state. In case weak obligations are violated, they are assigned to the set of the weak and strong sanctions that should be fulfilled in the next states. We note $\hat{wo}$ (resp. $\hat{so}$) the conjunction of the weakly (resp. strongly) obligatory formulas [3].

$$\hat{wo} \stackrel{def}{=} \bigwedge_{\phi \in wo} \phi \qquad \hat{so} \stackrel{def}{=} \bigwedge_{\phi \in so} \phi$$

Given a state s, considered as initial, $CheckCompliance(s)$ checks whether all the traces which start from s are compliant. It first initialises the four diagnostic variables. Then, for each outgoing transition, it calls *CheckTransition* with no inherited obligations. *CheckCompliance* then returns the four variables that provide a diagnostic.

CheckCompliance (s)
$strong\_violation := false$; $unexpected\_violation := false$;
$never\_caught := false$; $po := \emptyset$; $no\_viol := true$;
For each $E \in Out(s)$ **Do**
  CheckTransition(s,E,∅,∅);
Return $no\_viol, strong\_violation, unexpected\_violation, never\_caught$

$CheckTransition(s,E,wo,so)$ aims at checking that a transition performs events which are permitted, and that obligatory events effectively occur. *wo* (resp. *so*) is the set of the weakly (resp. strongly) obligatory formulas which are inherited from previous violations. These two sets are updated with the obligations which stand in the context of the current state s. If the current transition does not perform strongly obligatory events, then the variable *strong* is set to true. If the the set of weak obligations *wo* is already parsed in the current state/transition pair, then the variable *never_caught* is set to true and the algorithm terminates. If *wo* is not already parsed, then the set *po* of the parsed obligations is increased with the triple (s,E,wo). If there is some violation, i.e. if some obligatory formula does not holds or some non-permitted event set is performed, then the variable *no_viol* is set to false, and the sets *wo* and *so* of weak and strong sanctions are computed. If they are empty, i.e., if no sanction is specified, then the variable *unexpected* is set to true, else, the algorithm checks whether the sanctions are enforced in the next states calling itself recursively on every possible successor.

CheckTransition(s,E,wo,so)
$wo := wo \cup WObl(s)$; $so := so \cup SObl(s)$;
If $\hat{E} \not\Rightarrow \hat{so}$ **Then**
  $strong := true$;
If $(s,E,wo) \in po$ **Then**
  $never\_caught := true$;
Else
  $po := po \cup \{(s,E,wo)\}$;
  If $\hat{E} \not\Rightarrow \hat{wo}$ or $\hat{E} \not\Rightarrow \hat{so}$ or $\hat{Perm}(s) \not\Rightarrow \hat{E}$ **Then**
    $no\_viol := false$;
    $wo := WSanc_O(s,E,\hat{wo}) \cup WSanc_P(s,E)$;
    $so := SSanc_O(s,E,\hat{wo}) \cup SSanc_P(s,E)$;
    If $wo = so = \emptyset$ **Then**
      $unexpected := true$
  Else
    For each $s' \in succ(s,E)$ and $E' \in Out(s')$ **Do**
      CheckTransition(s',E',wo,so)

[3] We consider that $\hat{\emptyset} = \top$

PROPERTY 2 (TERMINATION).
Algorithm CheckCompliance terminates.

Sketch of the proof There is no loop, and the only recursive call is in *CheckTransition*. The termination of *CheckTransition* is straightforward, because the set of triples (state, transition, obligation) is finite (finiteness of the set of rules implies finiteness of the set of possible obligations), and the set of triples for which the obligation is not parsed (the complementary of the set *po*) is strictly decreasing at each recursive call, which guarantees the termination. ■

We now establish the soundness of *CheckCompliance* with respect to the semantics defined in section 5.2.

PROPERTY 3 (SOUNDNESS).
Given a system $\mathcal{A} = (S, S_0, \Delta, V)$, and a state $s \in S$, after the call of $CheckCompliance(s)$, the following holds

- *no_viol = true iff for every state/event trace $\sigma = (s,\ldots)$ of $\mathcal{A}$ starting with s, and for every conjunction φ of violation formulas, $\sigma \models_{pol} G\,\neg\varphi$*
- *strong_violation, unexpected_violation, and never_caught all equal false iff for every state/event trace $\sigma = (s,\ldots)$ of $\mathcal{A}$ starting with s, and every conjunction φ of violation formulas, $\sigma \models_{pol} G\,(\varphi \Rightarrow managed(\varphi))$*
- *strong_violation = true iff there is some state/event trace $\sigma = (s,\ldots)$ of $\mathcal{A}$ starting with s, and some conjunction φ of violation formulas, such that $\sigma \models_{pol} strong(\varphi)$*
- *unexpected_violation = true iff there is some state/event trace $\sigma = (s,\ldots)$ of $\mathcal{A}$ starting with s, and some conjunction φ of violation formulas, such that $\sigma \models_{pol} unexpected(\varphi)$*
- *never_caught = true iff there is some state/event trace $\sigma = (s,\ldots)$ of $\mathcal{A}$ starting with s, and some conjunction φ of violation formulas, such that $\sigma \models_{pol} never\_caught(\varphi)$*

The **proof** presents no particular difficulties because the algorithm deals with concepts that are close to the semantics of the formal definitions. We omit to develop the proof here because it is quite long.

7. EXAMPLE

In this section, we develop the bank example aforementioned in the introduction. The algorithm given in section 6 checks the compliance of a system with respect to its policy and provides a diagnostic as explained above. Figure 7 shows a partial output of this algorithm for the three following instances of a bank model.

The bank system is modeled as an automaton that models the behaviour of a customer together with the state of his/her bank account. The sets P and $\mathcal{E}$ of atomic propositions and events are $P = \{positive\}$ and
$\mathcal{E} = \{credit, debit, get\_pos\_balance, get\_neg\_balance, pay\_charges, go\_to\_jail\}$. *positive* is true when the balance is positive. *credit* (resp. *debit*) labels any transition that credits (resp. debits) the account. *get_neg_balance* (resp. *get_pos_balance*) labels the transitions that go from a negative balance to a positive one (resp. from

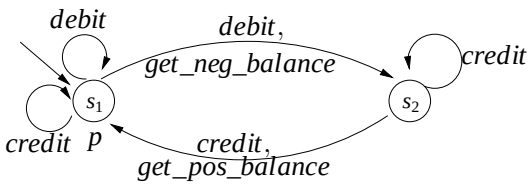

Figure 4: First example of a system

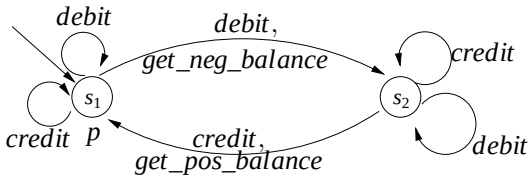

Figure 5: Second example of a system

a positive balance to a negative one). *pay_charges* models the payment of charges by the customer, and *go_to_jail* is an event by which the customer has no longer access to his/her account.

Recall that when several events label some transition, it means that these events occur simultaneously when this transition is performed. For instance, in the first example (Figure 4), when the transition from s_1 to s_2 is performed, *debit* and *get_neg_balance* both occur. Also recall that $\boldsymbol{P}(e_1 \wedge e_2)$ means that there is the permission to perform e_1 and e_2 simultaneously. According to our semantics (cf section 5.2), $\boldsymbol{P}(e_1 \wedge e_2)$ implies $\boldsymbol{P}(e_1) \wedge \boldsymbol{P}(e_2)$ but the converse does not hold.

We consider the following policy:

- *credit, pay_charges, get_pos_balance, and go_to_jail operations are always permitted.*
  $\top \rightsquigarrow \boldsymbol{P}(credit \wedge pay\_charges \wedge get\_pos\_balance)$
- *When the balance is positive, it is permitted to perform a debit operation, even if it leads to a negative balance.*
  $positive \rightsquigarrow \boldsymbol{P}(debit \wedge get\_neg\_balance)$
- *If the balance is negative, then it is obligatory to credit the account.*
  $\neg positive \rightsquigarrow \boldsymbol{O}(credit)$
- *If a debit operation is performed without permission, then it is obligatory to pay charges.*
  $Viol_P(debit) \rightsquigarrow \boldsymbol{O}(pay\_charges)$

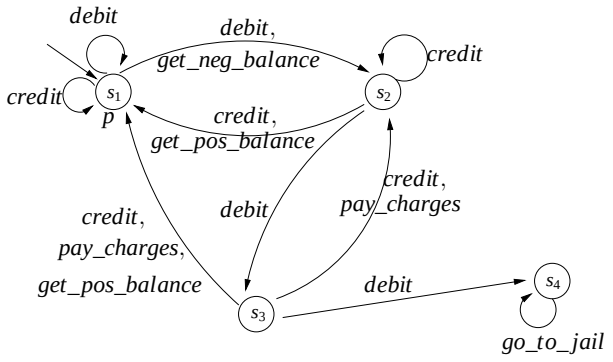

Figure 6: Third example of a system

system \ variable	*no_viol*	*strong_violation*
example1	true	false
example2	false	true
example3	false	false

Figure 7: Output of $CheckCompliance(s_1)$

- *If an obligation to credit the account is violated, then it is obligatory to pay charges.*
  $Viol_O(credit) \rightsquigarrow \boldsymbol{O}(pay\_charges)$
- *If an obligation to pay the charges is violated, the customer has a strong obligation to go to jail.*
  $Viol_O(pay\_charges) \rightsquigarrow \hat{\boldsymbol{O}}(go\_to\_jail)$

The first system (Figure 4) is compliant with the policy, and never violates any obligation or prohibition (all its traces satisfy the property $G\,\neg\varphi$ for every violation formula φ). The second one (Figure 5) may clearly violate the obligation to credit the account in the state s_2, if it performs the event *debit*. Then, there is the obligation to pay charges, which can also be violated by performing again the event *debit*. There is then a strong obligation to go to jail, which may be violated by performing *debit*, or also *credit*. So the second system is not compliant with the policy (a trace, for instance, which starts with $(s_1, \{debit\}, s_2, \{debit\}, s_2, \{debit\}, s_2, \{debit\})$ satisfies the property $F\ strong(Viol_O(credit))$. The third one (Figure 6) is compliant but may violate some obligations: all its traces satisfy the property $G\ (\varphi \Rightarrow managed(\varphi))$ for every violation formula. For instance, a trace which starts with $(s_1, \{debit\}, s_2, \{debit\})$ satisfies $Viol_O(credit)$ in its second state, but also satisfies $managed(Viol_O(credit))$.

Figure 7 shows the value of the variables *no_viol* and *strong_violation* after the call of our algorithm for state s_1. The two other diagnostic variables *unexpected_violation* and *never_caught* both equal false for the three examples.

8. CONCLUSION

The main contribution of this paper is the definition of a formal framework to specify and manage security requirements that apply when other security requirements are violated, and the formal definition of compliance in this context. In deontic logic, these requirements are generally called "contrary to duty" rules [6, 19]. To the best of our knowledge, this is the first time such kind of rules are considered to specify a security policy.

In this context, we have presented a model based on deontic logic to specify a security policy. We also defined an approach to check if some system specification complies with its associated security policy. We refined the definition of compliance into five diagnostic cases: (1) there is no violation, (2) there are violations but the sanctions described in the policy apply, (3) there is a violation for which the sanction is not enforced, (4) there is a violation for which no sanction is specified, and (5) there is a violation which is never caught. The algorithm presented in section 6 provides this diagnostic for every transition. Its termination and soundness are established.

In this paper, we deliberately simplify the security policy specification by only considering permission and obligation requirements. We assumed that prohibitions implicitly correspond to the absence of permission. As a first perspective, we plan to extend our approach by also including explicit prohibitions. This extension requires to detect and manage inconsistencies within the policy due to permission/prohibition and obligation/prohibition conflicts [7].

Another extension of the approach would be to consider obligation requirements with deadline as suggested in [10, 4]. We need to refine the definition of compliance given in this paper to consider these more complex requirements. Dedicated decision procedures which provide diagnostics for the enriched logical framework should be studied. Model-checking techniques for timed logic as implemented for instance in the UPPAAL tool may be useful to check the compliance of a system with respect to these security requirements.

9. REFERENCES

[1] E. Bertino, B. Catania, E. Ferrari, and P. Perlasca. A Logical Framework for Reasoning about Access Control Models. *ACM Transactions on Information and System Security*, 6(1), February 2003.

[2] E. Bertino, S. Jajodia, and P. Samarati. Supporting Multiple Access Control Policies in Database Systems. In *IEEE Symposium on Security and Privacy*, Oakland, USA, 1996.

[3] C. Bettini, S. Jajodia, X. S. Wang, and D. Wijesekera. Obligation Monitoring in Policy Management. In *International Workshop, Policies for Distributed Systems and Neworks (Policy 2002)*, Monterey CA, June 5–7 2002.

[4] J. Brunel, J.-P. Bodeveix, and M. Filali. A State/Event Temporal Deontic Logic. In L. Goble and J.-J. C. Meyer, editors, *8th International Workshop on Deontic Logic in Computer Science (DEON'06)*, pages 85–100, Utrecht, The Netherlands, 2006.

[5] S. Chaki, E. M. Clarke, J. Ouaknine, N. Sharygina, and N. Sinha. State/event-based software model checking. In E. A. Boiten, J. Derrick, and G. Smith, editors, *4th International Conference on Integrated Formal Methods (IFM '04)*, volume 2999 of *Lecture Notes in Computer Science*, pages 128–147. Springer-Verlag, 2004.

[6] R. M. Chisholm. Contrary-to-duty imperatives and deontic logic. *Analysis*, 24(2):33–36, December 1963.

[7] L. Cholvy and F. Cuppens. Reasoning about norms provided by conflicting regulations. In P. McNamara and H. Prakken, editors, *Fourth International Workshop on Deontic Logic in Computer Science (DEON)*, Bologna, Italy, 1998.

[8] E. Clarke, O. Grumberg, and P. D.A. *Model Checking*. MIT Press, 1999.

[9] F. Cuppens, N. Cuppens-Boulahia, and T. Ramard. Availability enforcement by obligations and aspects identification. In *The First International Conference on Availability, Reliability and Securit (ARES)*, pages 229–239, Vienna, Austria, 2006.

[10] F. Cuppens, N. Cuppens-Boulahia, and T. Sans. Nomad : A Security Model with Non Atomic Actions and Deadlines. In *The computer security foundations workshop (CSFW)*, Aix en Provence, France, 2005.

[11] F. Cuppens and A. Miège. Modelling Contexts in the Or-BAC Model. In *19th Annual Computer Security Applications Conference (ACSAC '03)*, 2003.

[12] M. Harrison, W. Ruzzo, and J. Ullman. Protection in operating systems. *CACM*, 19(8):461–471, August 1976.

[13] S. Jajodia, S. Samarati, and V. S. Subrahmanian. A logical Language for Expressing Authorizations. In *IEEE Symposium on Security and Privacy*, Oakland, CA, May 1997.

[14] A. A. E. Kalam, R. E. Baida, P. Balbiani, S. Benferhat, F. Cuppens, Y. Deswarte, A. Miège, C. Saurel, and G. Trouessin. Organization Based Access Control. In *Proceedings of IEEE 4th International Workshop on Policies for Distributed Systems and Networks (POLICY 2003)*, Lake Come, Italy, June 2003.

[15] A. Lomuscio and M. Sergot. A formalisation of violation, error recovery, and enforcement in the bit transmission problem. *Journal of Applied Logic*, 2:93–116, 2004.

[16] F. Martinelli and I. Matteucci. An approach for the specification, verification and synthesis of secure systems. *Electr. Notes Theor. Comput. Sci.*, 168:29–43, 2007.

[17] P. McDaniel. On Context in Authorization Policy. In *Proceedings of the 8th ACM Symposium On Access Control Models and Technologies (SACMAT 2003)*, Como, Italy, June 2003.

[18] J. Park and R. Sandhu. The UCON-ABC Usage Control Model. *ACM Transactions on Information and System Security*, 7(1):128–174, 2004.

[19] H. Prakken and M. Sergot. Contrary-to-Duty Imperatives, Defeasibility and Violability. In A. J. I. Jones and M. Sergot, editors, *Second International Workshop on Deontic Logic in Computer Science*, Oslo, Norway, 1994.

[20] R. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. Role-based access control models. *IEEE Computer*, 29(2):38–47, 1996.

[21] F. B. Schneider. Enforceable security policies. *Information and System Security*, 3(1):30–50, 2000.

[22] M. Strembeck and G. Neumann. An Integrated Approach to Engineer and Enforce Context Constraints in RBAC Environements. *ACM Transactions on Information and System Security*, 7(3):392–427, 2004.

[23] B. D. Win, B. Vanhaute, and B. Decker. Security Through Aspect-Oriented Programming. In *Advances in Network and Distributed Systems Security. IFIP TC11 WG11.4 First Working Conference on Network Security*, Leuven, Belgium, 2001. Kluwer Academic Publishers.

[24] G. H. V. Wright. Deontic logic. *Mind*, 1951.