Analyzing Consistency of Security Policies

Laurence Cholvy and Frédéric Cuppens
ONERA-CERT
2 Av. E. Belin
31055, Toulouse Cedex
France
email: {cholvy,cuppens}@cert.fr
fax: +(33) 62 25 25 93

Abstract

This paper discusses the development of a methodology for reasoning about properties of security policies.

We view a security policy as a special case of regulation which specifies what actions some agents are permitted, obliged or forbidden to perform and we formalize a policy by a set of deontic formulae.

We first address the problem of checking policy consistency and describe a method for solving it. The second point we are interesting in is how to query a policy to know the actual norms which apply to a given situation. In order to provide the user with consistent answers, the normative conflicts which may appear in the policy must be solved. For doing so, we suggest using the notion of roles and define priorities between roles.

1. Introduction

The primary goal of a security policy is to specify means for facing a given environment of threats. All organizations generally have designed security policies that apply to all systems within the organization and define the security relationship between the organization and the outside world. In our approach, we view a security policy as a specific case of regulation. The systems to be regulated are composed of agents which can perform some actions on some objects. A regulation on such a system aims at defining what actions the agents are permitted, obliged or forbidden to perform. This represents a set of constraints to be enforced by the agents. We can actually divide these constraints into two classes:

1. The constraints to be enforced by the agents when they perform actions on the system objects.
2. The constraints to be enforced by agents when they interact with other agents. In this case, the regulation may use various concepts such as the concepts of responsibility, delegation, hierarchical authority and so on...

This paper mainly focus on the first class of constraints. In this context, our first goal is to provide a precise and non ambiguous specification of a security policy. For doing so, we use a deontic logic, also called logic of norms, for representing the concepts of permission, obligation and prohibition. We then extend this formalism with the notion of role. Intuitively, each individual is associated with a set of roles, each of them representing the ideal behaviour he should have in a given situation. Each role defines the permissions, obligations and prohibitions laid upon the role-holder.

The advantage of a representation based on formal logic is that it is then possible to precisely define the axioms to reason about a regulation. This enables one to develop tools to analize the consequences of the norms used to define a given regulation. In this paper, we especially focus on the following functionalities:

1. Check the regulation consistency, i.e. check if the regulation does not create conflicting situations, for instance situations in which an agent is permitted to perform a given action and, at the same time, forbidden to perform this action.

2. Query a regulation to know which norms apply to a given situation. Since some conflicting situations may be created by the regulation and since our objective is to provide consistent answers, we first need to solve these conflicts. Our solution is the following. We consider that there is no normative conflict within a given role. Therefore, a conflict can only exist when an individual is playing two different roles and a conflict exists between these two roles. In this case, the central idea

is to consider that it is possible to make a judgment of priority between these two roles in order to evaluate the *actual* norm which applies in the examined situation.

Analyzing these two functionalities is not completely new. Abadi et al. [1] proposed a language for specifying security policies based on access control lists. The authors also include the possibility to deal with roles and delegation and provide theories for deciding whether requests should be accepted or not. In [10], Gong and Qian analyze the problem of complexity and composability of secure interoperation. They consider a context in which there are several systems, each system having its own security policy. Two general principles are then stated. (a) *Principle of Autonomy:* Any access permitted within an individual system must be also permitted under secure interoperation. (b) *Principle of Security:* Any access not permitted within an individual system must be also denied under secure interoperation. In this context, the authors show that most problems are NP-complete even for systems with very simple access control structures. Another approach for combining components and policies was proposed in [8]. The authors provide a means for showing whether the combinations of components will satisfy specified policies. In [3], Bertino et al. propose an authorization mechanism that enables multiple access control policies to be supported. The mechanism enforces a general authorization model which distinguishes between positive and negative authorizations. A positive authorization corresponds to permission whereas a negative authorization corresponds to prohibition. It also distinguishes between weak and strong authorizations. A strong authorization overrides a weak authorization whereas strong authorization cannot be overridden. In this model, only conflicts between weak authorizations are manageable and, in this case, the authors propose an approach to resolve conflicts.

Since our approach in based on deontic logic, we can directly reuse the formal and precise definition of the deontic notions proposed in this kind of logic. It is an attractive candidate for expressing security policies which was first used by Glasgow and McEwen to specify confidentiality policies [9]. It enables one to consider security policies specifying norms with obligation whereas other approaches only consider permission and prohibition. Our model also includes the possibility of dealing with conditional norms, for example: "If it is during the day and if an individual is located in room S155, this individual is permitted to access the system. Otherwise, he is forbidden to do so". As far as we know, the only model dealing with conditional norms was proposed in [16] but the problem of consistency checking is not addressed in this paper.

Our final goal in this work would be to define a global security model to deal with MAC (Mandatory Access Control), DAC (Discretionary Access Control) or RBAC (Role Based Access Control) and which enables confidentiality, integrity and availability requirements to be specified. Of course, this is an ambitious goal and our studies may be viewed as a first step in this direction.

This paper is organized as follows. In section 1, we describe an example of security policy we shall use along this paper to illustrate our approach. In section 2, a first logic based on SDL (Standard Deontic Logic) is defined and it is shown how our example is formalized within it. In section 3, the problem of regulation consistency is addressed. In section 4, we refine the previous logic and describe our approach to solve normative conflicts. We apply our solution to the example. Finally, in section 5, we discuss additional functionalities that may be developed to assist security administrators, in their attempt to specify and formalize security policies. The study of these functionalities remains to be done.

2. Example of regulation

The logical approach we propose to deal with a security policy is based on the concept of role. Each role is associated with a set of norms (i.e. permissions, obligations and prohibitions). Depending on circumstances, agents play roles. When playing a role, an agent inherits the set of norms associated with this role. An agent can play several roles simultaneously. In this case, he inherits the union of the norms associated with each role he plays.

Therefore, specifying a regulation by using the concept of role leads to defining the set of permissions, obligations and prohibitions associated with each role. This part of the specification is what we call the *normative* specification of a regulation.

Notice that is also necessary to specify the general conditions which must be satisfied by an an agent in order to say that he is playing a given role. These general conditions must be included in the regulation specification. We shall call this part the *descriptive* specification of a regulation.

Let us now informally specify the example security policy we shall use through the remainder of this paper: we consider a security policy which defines obligations, permissions and prohibitions associated with the four following roles: User, Secret_User, SSO (System Security Officer) and Bad_User.

The normative part of our security policy is as follows:

- Any agent playing the role of User is:
  - Permitted to read the public files.
  - Permitted to write his own public files.
  - Forbidden to downgrade a file.
  - Obliged to change his password monthly.
- Any agent playing the role Secret_User is:

  - Permitted to read the secret files.
  - Permitted to write his own secret file.

- Any agent playing the role SSO is:
  - Permitted to downgrade a file.

- Any agent playing the role Bad_User is:
  - Forbidden to access the system.

The descriptive part of this security policy is as follows:

- Role "User" characterizes any agent who is using the system.
- Role "Secret_User" characterizes any agent using the system and cleared at the secret level.
- Role "SSO" characterizes any secret user who is, additionally cleared to be a system security officer.
- Role "Bad_User" characterizes users who do not perform what they should do, according to the norms associated with the role of User. In our example, this is characterized by an agent who has not changed his password for more than one month.

3. Logical formalization in SDL

In this section, we briefly present the logic SDL (Standard Deontic Logic) which is the simplest logic designed for reasoning about deontic notions. In order to model the relationship between an agent and a role he is playing, we introduce a binary predicate *Play*. We shall then show in section 3 that SDL is adequate to analyze the problem of regulation consistency.

3.1. The language of SDL

Let us denote the language of SDL by L. This is a first order language without function symbols, but with one modality denoted O. Therefore, the language L is defined by the following rules:

- If P is a n-ary predicate and if a_1, ..., a_n are constants or variables, then $P(a_1, ..., a_n)$ is a (atomic) formula.
- If p is a formula of L then $\neg p$ is a formula of L.
- If p and q are formulae of L then $p \wedge q$ is a formula of L.
- If p is a formula and x a variable, then $\forall x\ p$ is a formula.
- If p is a formula then Op is a formula. Op is to be read "p is obligatory".
- Nothing else is a formula of L.

Other deontic modalities are introduced by the following definitions:

- $Pp \stackrel{\text{def}}{=} \neg O \neg p$
- $Wp \stackrel{\text{def}}{=} \neg Op$
- $Fp \stackrel{\text{def}}{=} O \neg p$

Pp, Wp and Fp are respectively to be read "p is permitted, waived, forbidden".

3.2. Axiomatics

The axiomatics of SDL is similar to the one of KD logic [11, 4]. Axioms are:

- All axioms of first order logic.
- $O \neg p \rightarrow \neg Op$
- $Op \wedge O(p \rightarrow q) \rightarrow Oq$

and inference rules are:

- The inference rules of first order logic.
- O-Necessitation: If $\vdash p$ then $\vdash Op$

Due to space limitation, we shall not recall the complete semantics of SDL. Let us just say that the interpretations of SDL are standard Kripke interpretations of KD logic, i.e. interpretations related by one accessibility relation which has the property to be serial.

3.3. Specifying our example in SDL

We now propose a specification in SDL of the example we informally presented in section 1. For this purpose, we introduce the following predicates:

- Unary Predicates: *File, Public, Secret, Old_Passwd, Access_System, Change_Passwd.*
- Binary Predicates: *Play, Owner, Password, Cleared, Login, Read, Write, Downgrade.*

The intuitive meaning of these predicates will become clear through the following formulae used to specify our security policy.

The normative part of the policy is as follows:

- (R1) Any agent playing the role User is permitted to read any public file.

$$\forall f, \forall A, File(f) \wedge Public(f) \wedge Play(A, User) \longrightarrow P\ Read(A, f)$$

- (R2) Any agent playing the role User is permitted to write his own public files.

  $\forall f, \forall A,$
  $File(f) \wedge Public(f) \wedge Owner(f, A) \wedge Play(A, User)$
  $\longrightarrow P\ Write(A, f)$

- (R3) Any agent playing the role User is forbidden to downgrade a file.

  $\forall f, \forall A, File(f) \wedge Play(A, User)$
  $\longrightarrow F\ Downgrade(A, f)$

- (R4) Any agent playing the role User is obliged to change his password if this password is more than one month old.

  $\forall A, \forall pass,$
  $Password(A, pass) \wedge Old\_Passwd(pass) \wedge$
  $Play(A, User)$
  $\longrightarrow O\ Change\_Passwd(A)$

- (R5) Any agent playing the role Secret_User is permitted to read the secret files.

  $\forall f, \forall A,$
  $File(f) \wedge Secret(f) \wedge Play(A, Secret\_User)$
  $\longrightarrow P\ Read(A, f)$

- (R6) Any agent playing the role Secret_User is permitted to write his own secret files.

  $\forall f, \forall A,$
  $File(f) \wedge Secret(f) \wedge Owner(f, A) \wedge$
  $Play(A, Secret\_User)$
  $\longrightarrow P\ Write(A, f)$

- (R7) Any agent playing the role SSO is permitted to downgrade a file.

  $\forall f, \forall A, File(f) \wedge Play(A, SSO)$
  $\longrightarrow P\ Downgrade(A, f)$

- (R8) Any agent playing the role Bad_User is forbidden to access the system.

  $\forall A, Play(A, Bad\_User) \longrightarrow F\ Access\_System(A)$

The classical approaches used to specify security policies only consider norms with permission or prohibition to do something. In our approach, we also want to consider norms with obligation. Rule (R4) provides an example of such a norm. One may argue that it would not be possible to implement access controls in a system so that it is a certainty that a user will change his password monthly. This is the reason why the regulation makes provision for the case when rule (R4) is violated, i.e. a user has not changed his password when he should have. In this case, rule (R8) applies and the user is no longer permitted to access the system[1].

The descriptive part of this security policy is as follows:

- (C1) Role "User" characterizes any agent who is using the system.

  $\forall A, Play(A, User) \equiv \exists level, Login(A, level)$

- (C2) Role "Secret_User" characterizes any agent using the system and cleared at the secret level.

  $\forall A, Play(A, Secret\_User) \equiv$
  $Cleared(A, secret) \wedge Login(A, secret)$

- (C3) Role "SSO" characterizes any secret user who is, additionally cleared to be a system security officer.

  $\forall A, Play(A, SSO) \equiv$
  $Play(A, Secret\_User) \wedge Cleared(A, SSO)$

- (C4) Role "Bad_User" characterizes any agent who has not changed his password for more than one month.

  $\forall A, Play(A, Bad\_User) \equiv$
  $Play(A, User) \wedge$
  $\exists pass,$
  $(Password(A, pass) \wedge Old\_Passwd(pass)) \wedge$
  $\neg Change\_Passwd(A)$

Finally, there are general rules which are not directly related to the regulation specification but to the domain of application handled by this regulation. These general rules include a specification of how the predicates are connected together. In particular, in order to completely analyze the consistency of our example, we need to add the three following general rules:

- (G1) An agent cannot read a file without accessing the system.

  $\forall f, \forall A, Read(A, f) \rightarrow Access\_System(A)$

- (G2) An agent cannot write a file without accessing the system.

  $\forall f, \forall A, Write(A, f) \rightarrow Access\_System(A)$

- (G3) An agent cannot downgrade a file without accessing the system.

  $\forall f, \forall A, Downgrade(A, f) \rightarrow Access\_System(A)$

In the remainder of this paper, rules (G1),(G2),(G3) are called the domain constraints.

Let us notice that (G1),(G2),(G3) do not give a complete description of the worlds ruled by the regulation. For instance, we could add the following rule:

- (G4) The effect of downgrading a file is that the file classification becomes public.

  $\forall f, \forall A, Downgrade(A, f) \rightarrow Public(f)$

However, rules (G1),(G2),(G3) are sufficient to illustrate how to verify regulation consistency.

[1] This kind of rule is generally called a *contrary to duty* norm

4. Verifying regulation consistency

We restrict our study of regulation consistency to regulations which are sets of SDL formulae of the following form:

$$QX \bigwedge_{\text{i=1..n}} \bigvee_{j=1..m} l_{ij} \rightarrow \bigvee_{r=1..k} \mathbf{M}\, l_r$$

where QX are universally or existentially quantified variables, the l_{ij}'s and the l_r's are literals[2] and $\mathbf{M} \in \{O, \neg O\, P, \neg P, F, \neg F, W, \neg W\}$. One can verify that rules (R1)-(R8) previously defined are of this form.

These rules express what is obligatory or not, permitted or not, forbidden or not, waived or not, and according to which conditions. The main restriction is that we only consider literals within the range of a deontic modality.

Intuitively, we want a regulation $\mathcal{R}$ to be consistent if and only if there exists no world, ruled by $\mathcal{R}$, in which $\mathcal{R}$ leads to a contradiction (for instance, it is both permitted and forbidden for someone to do something) or in which $\mathcal{R}$ requires that someone face a moral dilemma (i.e, one is obliged to do something and also obliged to do its contrary). In other words, as soon as we can detect such a world, we say that $\mathcal{R}$ is not consistent.

For instance, let us assume that one rule of a regulation permits a user to read a secret-file and another one prohibits him from accessing the file system. Since we know that reading a file implies accessing the file system, we shall consider that the regulation is not consistent. Indeed, as soon as a user is logged in the system, he is faced with a contradiction: he is permitted to read a secret-file and therefore, to access the file system but he is also prohibited from doing so.

This example shows us that, for defining the notion of regulation consistency, we need to consider not only the rules in the regulation (which express what is permitted, obligatory or forbidden) but also the rules which constrain the worlds ruled by the regulation, the so-called domain constraints. As a matter of fact, if in the previous example, we do not assume that reading a file implies accessing the file system, then we do not have a contradiction.

4.1. Definition of regulation consistency

Let us denote Dom the set of domain constraints and let us denote SDL_{Dom} the logical system derived from SDL by adding formulae of Dom as proper axioms. $\models_{Dom}$ will denote the logical consequence in SDL_{Dom}.

Intuitively, regulation $\mathcal{R}$ is consistent if there is no world in which it would not be possible to apply $\mathcal{R}$. So, a trivial case of regulation consistency is when there is a world, satisfying the domain constraints Dom, in which $\mathcal{R} \rightarrow False$ is satisfied. However, we also want to consider conditional norms. For instance, we may have:

$$p_1 \rightarrow O\ q \quad \text{and} \quad p_2 \rightarrow F\ q$$

where p_1 and p_2 are two conditions. In this case, the inconsistency only appears in a situation where $p_1 \wedge p_2$ is satisfied. This is the reason why we proposed, in [6], the following more complete definition of regulation consistency:

Definition 1 Let Dom be a set of domain constraints. Let $\mathcal{R}$ be a regulation.

$\mathcal{R}$ is consistent (according to domain Dom) iff there is no first order formula f, such that: $\models_{Dom} (\mathcal{R} \wedge f) \rightarrow False$ and $(f \wedge Dom)$ satisfiable.

This definition says that, if there is a situation *f* (corresponding to $p_1 \wedge p_2$ in our previous example) compatible with the domain constraints Dom, in which $\mathcal{R}$ leads to a contradiction, then we shall consider that $\mathcal{R}$ is not consistent.

4.2. Translating the problem into first order logic

In this section, we define a method for deciding if a given regulation is consistent or not[3]. This method consists in translating a regulation and a set of domain constraints into a set of first order formulae. Let us denote t this translation. We do not have enough place here to give the definition of t but let us illustrate this translation through our example:

- (R1) *a User is permitted to read public file* is expressed by the SDL formula:

  $$\forall f, \forall A, File(f) \wedge Public(f) \wedge Play(A, User) \longrightarrow P\ Read(A, f)$$

  It is translated into the following first-order formula:

  $$\forall f, \forall A, File(f) \wedge Public(f) \wedge Play(A, User) \longrightarrow \neg obligatory(not(read(A, f)))$$

  where $obligatory$ is now a predicate, not and $read$ are two functions.

- (R8) *a Bad_User is forbidden to access the system* is expressed by the SDL formula:

  $$\forall A, Play(A, Bad\_User) \longrightarrow F\ Access\_System(A)$$

  It is translated into the first order formula:

  $$\forall A, Play(A, Bad\_User) \longrightarrow obligatory(not(access\_system(A)))$$

The translation t has the following property:

[2] i.e. a positive or negative atomic formulae

[3] The complete description of this method is described in [6]. This technical report may be provided to the interested reader on request.

Proposition 1 Let us denote **D** the following axiom schema:

$$\mathbf{D} = \neg obligatory(a(X)) \vee \neg obligatory(not(a(X))$$

A regulation $\mathcal{R}$ is consistent iff there is no formula f such that:

1. f belongs to the language $t(L) \setminus \{obligatory\}$
2. $\models t(\mathcal{R}) \wedge t(Dom) \wedge \mathbf{D} \rightarrow \neg f$
3. $f \wedge t(Dom)$ is satisfiable

4.3. Verifying consistency vs consequence generation

According to the previous proposition, we notice that verifying the consistency of $\mathcal{R}$ comes down to ensuring that $t(R) \wedge t(Dom) \wedge \mathbf{D}$ has no consequence $\neg f$ belonging to $t(L) \setminus \{obligatory\}$, such that $f \wedge t(Dom)$ is satisfiable. This comes to a problem of *consequence finding* [13, 12].

Since the consequences $\neg f$ we want to generate belong to the language $t(L) \setminus \{obligatory\}$ (condition 1 of proposition 1) and satisfy a given condition (condition 3), we think that it is adequate to apply the inference rule called SOL-deduction, introduced by Inoue [12]. As a matter of fact, this rule has especially been designed to generate consequences of a given set of clauses which belong to a given language and which satisfy a given condition.

We do not detail here the SOL deduction because describing an inference rule for consequence finding is out of the scope of our paper. But we shall briefly present the SOL deduction in the appendix.

Let us also notice that, in [14] and [15], Ong and Lee have already shown that the problem of checking regulation consistency reduces to a problem of abduction or equivalently, a problem of consequence finding. In their papers, Ong and Lee defined an ad-hoc algorithm when the regulation and domain constraints are specified with Horn clauses. Our approach is more general since it is not restricted to Horn clauses and can be applied to general clauses.

This kind of algorithm which can be applied to general clauses is of course time-consuming. But we argue that the regulation consistency is not frequently checked; actually, it is only checked when the regulation is defined or updated. Thus, we are developing an off-line tool which does not need real-time performance.

4.4. Application to our example

Running the algorithm which implements the SOL-deduction, on our toy example provides us with the following answers:

- $\exists f \exists A \; File(f) \wedge Play(A, SSO)$

  This means that, as soon as there is a security officer A and a file f in the file-system, there is a contradiction. Indeed, in this case, we can derive that A is both permitted and forbidden to downgrade f.

- $\exists f \exists A \; File(f) \wedge Public(f) \wedge Play(A, Bad\_User)$

  This means that, as soon as there is a bad user A and a public file f, there is a contradiction. Indeed, in this case, we can derive that A is both permitted and forbidden to access the system.

- $\exists f \exists A \; File(f) \wedge Secret(f) \wedge Play(A, Bad\_User) \wedge Play(A, Secret\_User)$

  This means that, as soon as there is a secret file f and a secret user A who is a bad user, there is a contradiction. Indeed, in this case, we can also derive that A is both permitted and forbidden to access the system.

5. Solving normative conflicts

In this section, we address the problem of solving normative conflicts.

As said previously, our approach is to consider that each individual is associated with a set of roles, each of them representing the ideal behaviour the individual can play in a given situation. Each role is associated with a set of permissions, obligations, prohibitions, laid upon the role-holder. Roles are defined separately from the individuals. An individual inherits the set of norms associated with a role when he plays this role. For instance, in our example, we identified four roles: User, Secret_User, SSO and Bad_User.

We assume that the set of norms associated with a role is conflict free. Therefore, a conflict can only exist when an individual is playing different roles. For instance, the set of norms associated with User, prohibits a user from downgrading a file. However, a security officer is a particular user and he is permitted to downgrade a file. So, playing these two roles leads to a conflict: it is both permitted and forbidden to downgrade a file.

So the problem is the following: given several sets of norms corresponding to the different roles an individual can play, what is the set of norms corresponding to the set of roles he plays at a given time. For instance, what are the obligations, permissions and prohibitions applying to an individual who is both a user and a security officer? In particular, is he permitted to downgrade a file or not?

In this section, we first refine the formalism we propose in section 2 for checking regulation consistency. The deontic operators are now indexed by roles and we also slightly change the definition of permission. We then present the axioms of the logic used for reasoning with composite roles, i.e. roles obtained by merging several roles. For solving

the problem of conflicts between the different roles which constitute a composite role, we suggest using an order for merging the roles. This order represents a priority between them. We assume here, that this order is total. The extension to partial orders is discussed in [5].

In our example, the order imposed on the two roles User and SSO comes from the structure of the two roles: a security officer is a kind of user. So the preference is given to SSO and the individual who is a security officer is permitted to downgrade a file.

5.1. The language

Let us consider a finite set $Role = \{R_1, R_2, ..., R_n\}$ of roles. We shall use the following notation:

- If the roles to be merged are denoted $R_{i_1}, ..., R_{i_k}$, then the role obtained by merging them using the order $R_{i_1} > ... > R_{i_k}$ will also be denoted $R_{i_1} > ... > R_{i_k}$.
- If $o = R_{i_1} > ... > R_{i_k}$ is a composite role obtained by merging k roles, we shall denote $o > R_{i_{k+1}}$ the composite role obtained by merging $R_{i_1}, ..., R_{i_k}, R_{i_{k+1}}$ with the order $R_{i_1} > ... > R_{i_k} > R_{i_{k+1}}$.

The language L' we use is a refinement of language L we defined in section 2. It is defined as follows:

- If P is a n-ary predicate and if a_1, ..., a_n are constants or variables, then $P(a_1, ..., a_n)$ is a formula.
- If p is a formula of L' then $\neg p$ is a formula of L'.
- If p and q are formulae of L' then $p \wedge q$ is a formula of L'.
- If p is a formula and o is any primitive or composite role obtained by merging several roles, $O_o p$, $P_o p$ and $F_o p$ are formulae of L'. They will mean that, according to the primitive or composite role o, p is respectively obligatory, permitted or forbidden.
- Nothing else is a formula of L'.

5.2. Axiomatics

In the following axioms, l represents a literal, p and q represent formulae, R_i represents a primitive role and o represents a primitive or composite role.

- (A0) All axioms of first order logic.
- (A1) $O_o p \wedge O_o(p \rightarrow q) \rightarrow O_o q$
- (A2) $P_o(p \wedge q) \rightarrow P_o p \wedge P_o q$
- (A3) $O_o p \rightarrow P_o p$
- (A4) $O_o p \rightarrow \neg P_o \neg p$
- (A5) $F_o p \equiv O_o \neg p$
- (A6) $O_o l \rightarrow O_{o>R_i} l$
- (A7) $O_{R_i} l \wedge \neg P_o \neg l \rightarrow O_{o>R_i} l$
- (A8) $O_{o>R_i} l \rightarrow O_o l \vee O_{R_i} l$
- (A9) $P_o l \rightarrow P_{o>R_i} l$
- (A10) $P_{R_i} l \wedge \neg O_o \neg l \rightarrow P_{o>R_i} l$
- (A11) $P_{o>R_i} l \rightarrow P_o l \vee P_{R_i} l$

Axioms (A1)–(A5) are the axioms which characterize the way of reasoning with obligations, permissions and prohibitions inside a (primitive or composite) role.

The axiomatics for each modality O_{R_i} is similar to SDL. In particular, notice that from axioms (A3) and (A4) we can derive: $\vdash O_o p \rightarrow \neg O_o \neg p$.

On the other hand, we break with the tradition in deontic logic which generally views obligation as dual of permission, i.e. $\vdash O_o p \leftrightarrow \neg P_o \neg p$. We only accept the implication from the left to the right (A4) but not the converse. This is because we consider that the set of norms associated with a role generally does not represent a complete regulation of the world, i.e. within a role, there may be sentences which are neither permitted nor obligatory.

This assumption is important when we merge the regulations associated with two different roles R_i and R_j. As a matter of fact, by merging the two roles, we want to consider that the roles are in some sense complementary and this is not possible if each role R_i and R_j represents two different but *complete* regulations of the world.

Axiom (A5) says that "it is forbidden that p" is defined as "it is obligatory that $\neg p$".

Axiom (A6) expresses that if l is obligatory in role o, then it is also obligatory according to the composite role $o > R_i$.

Axiom (A7) expresses that if l is obligatory in a primitive role R_i and if $\neg l$ is not permitted in role o, then l is obligatory according to the role $o > R_i$.

Axiom (A8) expresses that if l is not obligatory in both roles o and R_i, then it is not obligatory in role $o > R_i$.

Axiom (A9) expresses that if l is permitted in role o, then it is also permitted in role $o > R_i$.

Axiom (A10) expresses that if l is permitted in a role R_i and if $\neg l$ is not obligatory in role o, then l is permitted in role $o > R_i$.

Axiom (A11) expresses that if l is not permitted in both roles o and R_i, then it is not permitted in role $o > R_i$.

The inference rules are:

- (I1) The inference rules of first order logic.
- (I2) $\frac{\vdash p}{\vdash O_o p}$

- (I3) $\frac{\vdash p}{\vdash P_o p}$

5.3. Deriving actual norms

Merging several roles together allows us to derive the actual norms which apply to an individual in a given situation (For the sake of simplicity, we consider only one individual called "the individual"). For this purpose we first need a means for specifying which roles the individual is playing in the situation we want to consider and the order of priority among these roles applies. Therefore, we add to the language the following propositions:

- $All\_Roles(R_1, ..., R_k)$ for each subset $\{R_1, ..., R_k\}$ of the set of roles.

Intuitively, $All\_Roles(R_1, ..., R_k)$ is to be read "$R_1, ..., R_k$ are all and only all the roles played by the individual and these roles are prioritized in the order induced by $R_1 > ... > R_k$".

We can now define which actual norms apply to the individual in a given situation. For this purpose, we also add to the language simple modalities O, P and F. Intuitively, Op, Pp and Fp are respectively to be read: "p is an *actual* obligation, permission or prohibition for the individual". The axioms defining O, P and F are the following:

- (A12) $All\_Roles(R_1, ..., R_k) \rightarrow (Op \leftrightarrow O_{R_1>...>R_k} p)$
- (A13) $All\_Roles(R_1, ..., R_k) \rightarrow (Pp \leftrightarrow P_{R_1>...>R_k} p)$
- (A14) $Fp \leftrightarrow O\neg p$

Axiom (A12) says that if $R_1, ..., R_k$ are all and only all the roles played by the individual and if $R_1 > ... > R_k$ represents the order in which these roles are prioritized, then the actual obligations of the individual are derived by merging all the roles according to this order of priority.

Axiom (A13) is a similar definition for the actual permissions. Axiom (A14) simply says that "p is an actual prohibition" if and only if "$\neg p$ is an actual obligation".

5.4. Application to our example

To specify our example in language L', we have simply to index the deontic modalities by role $User$ in rules (R1)...(R4), $Secret\_User$ in rules (R5)-(R6), SSO in rule (R7) and Bad_User in rule (R8).

Let us now consider an agent A whose password is more than one month old, i.e,

$$\exists pass, Password(A, pass) \wedge Old\_Passwd(pass)$$

Let us also assume that there exists a file f_1 whose classification is secret, i.e.,

$$File(f_1) \wedge Secret(f_1)$$

Finally, let us assume that the four following roles we considered in the previous section are ordered as follows:

$$Bad\_User > SSO > Secret\_User > User$$

Figure 1 shows through several examples the answers we obtain when our approach to query a regulation is applied.

For instance, let us assume that one wants to know if A is permitted to downgrade f_1.

In a situation where A is playing the role SSO but not the role Bad_User, the answer is $True$. This is because, we can derive from the regulation that $P_{SSO}\ Downgrade(A, f_1)$ and, since SSO is the role having the highest priority among the roles played by A in this situation, we can also derive that $P\ Downgrade(A, f_1)$.

On the other hand, in a situation where A is playing both roles SSO and Bad_User, the answer is $False$. This is because, we can now derive that $F_{Bad\_User}\ Access\_System(A)$. And, by (G3), we can also derive that:

$$\neg Access\_System(A) \rightarrow \neg Downgrade(A, f_1)$$

Therefore, from the prohibition to access the system, we can derive the prohibition to downgrade f_1:

$$F_{Bad\_User}\ Downgrade(A, f_1)$$

This implies that downgrading f_1 is not permitted:

$$\neg P_{Bad\_User}\ Downgrade(A, f_1)$$

Since Bad_User is the role with the highest priority, we can now derive that $\neg P\ Downgrade(A, f_1)$ and the answer is $False$.

6. Conclusion

This paper investigates new directions for analyzing some properties of security policies. We have found the framework of deontic logic useful for this purpose especially when dealing with security policies which include situations where some norms may be violated. We focused on :

- How to check the security policy consistency.
- How to consistently query a security policy.

Queries	Roles played by A		
	$Secret\_User \wedge \neg SSO \wedge \neg Bad\_User$	$SSO \wedge \neg Bad\_User$	$SSO \wedge Bad\_User$
$O\ Change\_Passwd(A)$	$True$	$True$	$True$
$P\ Access\_System(A)$	$True$	$True$	$False$
$P\ Downgrade(A, f_1)$	$False$	$True$	$False$

Figure 1. Examples of queries

For these two aspects, support tools in Prolog have been implemented. We have developped a consistency checker based on the SOL-deduction as well as an algorithm for solving normative conflicts based on the logic described in section 4. Furthermore, we have applied this formalism to analyse a regulation, used in the context of the National Defense, which define means to protect secret data [7]. This analyse revealed many subtleties and ambiguities of this security policy.

This experiment also enables us to raise up several interesting real-world problems which seem to require more theoretical development. In particular, the analysis of this regulation shows that the concepts of responsibility and delegation must be modelled. It also show that our formalism must be extended in order to express temporal notions (see [7]). For this purpose, it may be interesting to adapt the approach proposed in [2].

There are several other functionalities that we plan to investigate in the future. For instance:

- Functionality for checking if a given situation does not violate the security policy. For instance, let us denote s the following situation:

  $File(f) \wedge Play(A, User) \wedge \neg Play(A, SSO) \wedge Downgrade(A, f)$

  This is a problem of checking conformity because situation s violates the prohibition for a user which is not a system security officer to downgrade a file. In this case, the formula to be checked to detect a conformity problem is the following:

  $$\models_{Dom} (s \wedge \mathcal{R}) \rightarrow F\ s$$

  where s is the situation to be checked. It would be also interesting to derive which sanction applies when a policy violation occurs.

- Functionality for designing new security policies resulting from the interoperability between several systems, each of them being associated with its own security policy.

7. Acknowledgements

This work was carried out with the support of the DRET (Contract No. 94002.012). The authors also wish to thank Lee Benzinger for his helpful criticism.

8. Appendix: The SOL-deduction: a short overview

Definition 2
A production field P *is defined by a language* L_P *and a condition* $Cond_P$.
A clause belongs to P *iff it belongs to language* L_P *and it satisfies* $Cond_P$.
A structured clause *is a pair* $< P, Q >$ *where* P *and* Q *are clauses.*

Let Σ be a set of clauses, C a clause and **P** a production field. A SOL deduction of a clause S from $\Sigma + C$ and **P** is a sequence of structured clauses $D_0, D_1 \ldots D_n$, such that:

1. $D_0 =< \Box, C >$
2. $D_n =< S, \Box >$
3. For each $D_i =< Pi, Qi >, P_i \cup Q_i$ is not a tautology.
4. For each $D_i =< Pi, Qi >, P_i \cup Q_i$ is not subsumed by a clause $P_j \cup Q_j$, where $D_j =< P_j, Q_j >$ is a structured clause such that $j < i$.
   This rule is not applied if D_i is generated form D_{i-1}, by application of the rule 5(a)i.
5. $D_{i+1} =< P_{i+1}, Q_{i+1} >$ is generated from D_i according to the following steps:
   (a) Let *l* be the left-most literal of Q_i. P_{i+1} and R_{i+1} are obtained by applying one of the following rules :
      i. **(Skip)** If $P_i \cup \{l\}$ belongs to **P**, then $P_{i+1} = P_i \cup \{l\}$ and R_{i+1} is the new ordered clause obtained by removing *l* from Q_i.

ii. (**Resolve**) If there is a clause B_i in Σ such that $\neg k \in B_i$ and l and k are unifiable with most general unifier θ, then $P_{i+1} = P_i\theta$ and R_{i+1} is an ordered clause obtaining by concatenating $B_i\theta$ and $Q_i\theta$, framing $l\theta$ and removing $\neg k\theta$.

iii. (**Reduce**) If either,

A. P_i or Q_i contains an unframed literal k different from l (**factoring**), or

B. Q_i contains a framed literal $\neg k$ (**ancestry**)

such that l and k are unifiable with most general unifier θ, then $P_{i+1} = P_i\theta$ and R_{i+1} is obtained from $Q_i\theta$ by deleting $l\theta$.

(b) Q_{i+1} is obtained from R_{i+1} by deleting every framed literal not preceded by an unframed literal in the remainder (**truncation**).

The production field considered in the problem of testing of regulation consistency is the following:

$\mathbf{P_{L,Dom}} = \{L \setminus \{obligatory\}\ ,\ \{c : \neg c \wedge Dom$ is satisfiable$\}\}$

References

[1] M. Abadi, M. Burrows, B. Lampson, and G. Plotkin. A Calculus for Access Control in Distributed Systems. *ACM Transactions on Programming Languages and Systems*, 15(4), September 1993.

[2] E. Bertino, C. Bettini, E. Ferrari, and P. Samarati. Supporting Periodic Authorizations and Temporal Reasoning in Database Access Control. In *Proceedings of the 22nd International Conference on Very Large Data Bases*, Bombay, India, 1996.

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

[4] B. F. Chellas. *Modal logic, an introduction*. Cambridge University Press, 1980.

[5] L. Cholvy and F. Cuppens. Solving normative conflicts by merging roles. In *Fifth International Conference on Artificial Intelligence and Law*, University of Maryland, USA, 1995.

[6] F. Cuppens and L. Cholvy. Etude des r glements de s curit : formalisation et d veloppement d'un outil d'interrogation. Rapport final 1/3521.00/DERI, Centre d' tudes et de recherches de Toulouse, 1995. Convention DRET n° 94002.012.

[7] F. Cuppens and C. Saurel. Specifying a Security Policy: A Case Study. In *Proc. of the computer security foundations workshop*, Kenmare, Co. Kerry, Ireland, 1996.

[8] G. Dinollt, L. Benzinger, and M. Yatabe. Combining Components and Policies. In *Proc. of the Computer Security Foundations Workshop VII*, Franconia, 1994.

[9] J. Glasgow and G. McEwen. Reasoning about knowledge and permission in secure distributed systems. In *Proc. of the computer security foundations workshop*, Franconia, 1988.

[10] L. Gong and X. Qian. The Complexity and Composability of Secure Interoperation. In *IEEE Symposium on Security and Privacy*, Oakland, 1994.

[11] G. E. Hughes and M. J. Cresswell. *An introduction to modal logic*. Methren London and New York, 1972.

[12] K. Inoue. Consequence-finding based on ordered linear resolution. In *proc of IJCAI*, 1991.

[13] R. Lee. *A completeness theorem and computer program for finding theorems derivable from given axioms*. PhD thesis, University of California, Berkeley, 1967.

[14] K. Ong and R. M. Lee. Detecting deontic dilemnas in bureaucratic rules : a first-order implementation using abduction. In A. Jones and M. Sergot, editors, *Proc of DEON'94, Oslo*, 1994.

[15] K. Ong and R. M. Lee. A decision support system for bureaucratic policy administration : an abductive logic programming approach. *Decision Support Systems*, (16), 1996.

[16] V. Varadharajan and C. Calvelli. Extending the Schematic Protection Model - I Conditional Tickets and Authentication. In *IEEE Symposium on Security and Privacy*, Oakland, 1994.