

Specifying Intrusion Detection and Reaction Policies: An Application of Deontic Logic

Nora Cuppens-Boulahia, Frédéric Cuppens

TELECOM Bretagne, 2 rue de la châtaigneraie, 35512 Cesson Sévigné Cedex, France

Abstract. The security policy of an information system may include a wide range of different requirements. The literature has primarily focused on access and information flow control requirements and more recently on authentication and usage control requirements. Specifying administration and delegation policies is also an important issue, especially in the context of pervasive distributed systems. In this paper, we are investigating the new issue of modelling intrusion detection and reaction policies and study the appropriateness of using deontic logic for this purpose. We analyze how intrusion detection requirements may be specified to face known intrusions but also new intrusions. In the case of new intrusions, we suggest using the *bring it about* modality and specifying requirements as prohibitions to bring it about that some security objectives are violated. When some intrusions occur, the security policy to be complete should specify what happens in this case. This is what we call a reaction policy. The paper shows that this part of the policy corresponds to *contrary to duty* requirements and suggests an approach based on assigning priority to activation contexts of security requirements.

1 Introduction

Current information systems have to face many threats that attempt to exploit their vulnerabilities. Moreover, since information systems tend to be increasingly complex, specifying their security policy is a tedious and error-prone task. In this context, specifying consistent, relevant and complete security policies of information systems is a major challenge for researchers.

There are many advantages of using a formal approach to specify the policy: (1) It provides non ambiguous specification of security requirements, (2) It is possible to develop support tools to formally analyse these requirements, (3) It is also possible to develop support tools to assist the security administrator in the task of automatically deploying these requirements over a security architecture.

A security policy may actually specify very different security requirements. We first suggest a classification of these various requirements a security policy may contain. We then focus on two sub-parts of the security policy that specify (1) intrusion detection requirements (IDR) and (2) reaction requirements (RR). We investigate the relevance of deontic logic to specify such requirements.

Intrusion detection has been an active research field for more than twenty years and many intrusion detection systems (IDS) have been developed and are

now available. However, intrusion detection requirements enforced by IDSs are generally considered independently of the remainder of the security policy. A first contribution of this paper is to consider that IDRs are actually a sub-part of the access control policy. This approach has several advantages. A first advantage is that it is then possible to formally analyse whether IDRs are consistent with other security requirements. Another advantage is that we can integrate IDRs into a deploying process in order to automatically configure IDSs. Finally and as shown in this paper, this approach provides means to formally specify in a reaction policy what should happen in case of violation of some IDRs.

Traditionally, access control requirements are modelled using permissions and prohibitions. We show how IDRs can be specified using prohibitions. However, we consider two different types of IDRs depending on the fact these requirements apply to known or unknown attacks. In the case of a known attack, an intrusion detection requirement specifies that it is prohibited to execute the action corresponding to this attack. Notice that the attack may actually correspond to an elementary action or to a composition of elementary actions corresponding to an attack scenario. In the case of unknown attacks, the specification of IDRs is more complex. Our approach in this case is based on the specification of security objectives. An IDR then corresponds to a prohibition for any subject (or group of subjects in the case of a distributed intrusion) to *bring it about* [29] that some security objective is violated. To our best knowledge, this is the first time the bring it about operator is used in this context. We then show how different IDRs may be deployed on different Intrusion Detection Systems (IDSs) including misuse based detection systems, anomaly based detection systems or correlation based detection systems.

However, the security policy must also specify what happens when an intrusion is detected. This is what we call a reaction policy. A reaction policy is a set of deontic requirements specifying obligations, prohibitions and possibly permissions that are triggered when an intrusion is detected. We show that these requirements may be actually viewed as *contrary to duty norms* (see [30, 33]).

In this paper, we do not actually develop a new deontic formalism to specify intrusion detection and reaction policies. Instead, we analyse which problems addressed in this paper may be solved using the current state of the art in deontic logic and which problems still require further investigation.

The remainder of this paper is organized as follows. In section 2, we informally introduce the concept of security policy and suggest a classification of the different requirements that may be specified in a security policy. We then focus in section 3 on a part of the security policy that corresponds to the access control policy. The access control policy should include an intrusion detection policy as explained in section 4. We show in section 4 how to express various requirements of such an intrusion detection policy. When an intrusion occurs in the information system, the security policy is violated. Thus, another part of the security policy consists in specifying a reaction policy. This is presented in section 5. In section 6, we discuss how to implement the approach suggested in this paper and list some open issues. Finally, section 7 concludes the paper.

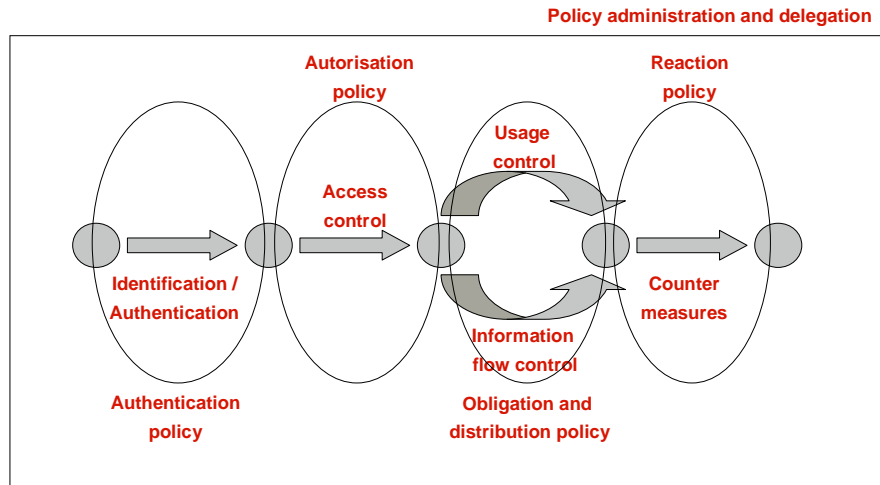


Fig. 1. General structure of a security policy

2 What is a security policy?

In the following, we shall view a security policy as a set of norms corresponding to permissions, prohibitions, obligations and dispensations. In the security literature, it is generally considered that these norms apply to users or processes (called subjects) when they access to resources (called objects) in order to execute services or programs (called actions).

A security policy may be structured into several sub policies (see figure 1):

- Authentication policy. Authentication is the first step to get an access to the information system and is used to securely associate a subject with its identity. The authentication policy specifies which authentication protocol a subject is permitted, prohibited or obliged to use to get an access to the system. Many authentication protocols, using password, smart cards or biometric traits, have been defined in the literature. It is also possible to specify that mutual authentication protocols such as Kerberos must be used so that each party involved in the interaction authenticates each other. More recently, authentication has included Single Sign On (SSO) functionalities. The authentication policy may influence other downstream policies. For example, the access control policy may condition its decisions on the kind of authentication that has been performed by the user. The use of stronger authentication protocols (loss of availability) may allow access to more sensitive resources (increase of confidentiality).
- Access control policy. This part of the policy is also called authorization policy in the literature and applies once subjects are authenticated. It corresponds to a set of permissions and prohibitions that specify which action

a subject may or may not execute on objects. The access control policy is further discussed in section 3.

- Usage control policy. It is a set of requirements that apply once a subject gets an authorized access to a resource. The objective is to control how this subject uses the resource. A usage control policy corresponds to a set of obligations to be enforced before the access (pre usage control), during the access (on going usage control) or after the access (post usage control). Even if there are many interesting issues to investigate, this is not the purpose of this paper to further address usage control. The interesting reader may consider the following references [28, 24].
- Information flow control policy. The objective here is to control how information flows in the information system, i.e. how information is transferred from subject to subject. A malicious subject, being permitted to access some information, may attempt to illegally transfer this information to another unauthorized subject. This problem is not appropriately managed by access control requirements and specific models have been defined for this purpose (see [3, 21]). However, there are currently no security models that integrate both access control requirements and the different forms of information flow control requirements. Even if this is out of the scope of this paper, the interesting reader may have a look at [2] for a preliminary work in this direction.
- Reaction policy. Since some requirements of the security may be violated, a security policy would not be complete if it does not include requirements that specify what happens in case of violation. This set of requirements is what we call a reaction policy. How to specify a reaction policy is further investigated in section 5.
- Administration and delegation policy. The administration policy specifies who is permitted to define new security requirements or update existing security requirements. Delegation also corresponds to the creation of permissions and obligations but generally (1) the delegator must own the permissions or obligations he or she delegates and (2) there is a transfer of these permissions or obligations from the delegator to the delegatee (see [20] for a more detailed discussion of the differences between administration and delegation). As suggested in the literature [19], administration and delegation may be modelled using special speech acts. Another possibility suggested in [15] would be to manage administration through licenses. A license is a special object that generally represents a permission for the subject who owns this license. In that case, the administration and delegation policies are specified by permissions to create or revoke licenses. A possible extension would be to define another special object called *duty* to represent an obligation for the subject who owns a duty and specify delegation of obligations through the creation of new duties.

In this paper, we shall actually focus on intrusion detection and reaction policies. However, since we view the intrusion detection policy as a sub part of the access control policy, we shall first briefly discuss how to model an access control policy.

3 Access control policy

3.1 Principles of access control

Specifying an access control policy has been investigated for more than thirty years [23]. In many models, an access control policy is modelled as a set of permissions. There are no obligation and dispensation in an access control policy but it is possible to also specify explicit prohibitions. The issue of using prohibition in an access control policy is further investigated in section 3.2 below.

Traditionally, access control policies apply to queries corresponding to subjects that ask to execute actions on objects. When a subject formulates such a query, the access controller analyses the query to check if it is permitted, in which case the query is accepted, else this query is rejected.

Generally, it is assumed that actions to be controlled corresponds to “elementary” actions. As a consequence, access control policies are specified using first order logic using a predicate like *is_permitted(subject, action, object)*.

More recently, it has been suggested to use the concept of role as in the RBAC (Role Based Access Control) model [32]. In this case, permissions are not directly assigned to subject but to role which may be modelled using a predicate *permission(role, action, object)*. Subjects are assigned to role using a predicate *empower(subject, role)* and we have the following derivation rule¹:

$$\begin{aligned} & \textit{permission}(\textit{Role}, \textit{Action}, \textit{Object}) \wedge \textit{empower}(\textit{Subject}, \textit{Role}) \\ & \rightarrow \textit{is_permitted}(\textit{Subject}, \textit{Action}, \textit{Object}) \end{aligned}$$

Actually, the RBAC model does not give any formal semantics to the concept of role. This lack of formalization leads to consider in many approaches that the concept of role is a panacea to solve every access control problems. This leads to consider “strange” roles such as location dependent role [6] or temporal dependent role [5].

There were several works that attempt to use deontic logic to provide formal semantics to the role concept [10, 27, 18]. The central idea is that a role is an organisation dependent concept. Basically, security policies are defined for moral authorities, called organisations. In this context, most of security requirements do not directly apply to concrete and implementation dependent entities such as subject, action and object. Instead, it is more appropriate to use abstract organisation dependent concepts. As suggested by RBAC, a role is one of such concept to create organization dependent abstraction of subjects. When an organization defines roles and assigns these roles to subjects, these subjects are no longer acting as individuals but as subjects empowered in some roles.

The OrBAC model [1] suggests to proceed similarly for actions and objects. For this purpose, we respectively suggest the concepts of activity and view as abstraction of action and object (see [1] for further explanation about these concepts). The predicate *use(object, view)* specifies that a given object is used in the

¹ In the following, we shall assume that terms starting with a capital letter represent variables and that all free variables in formula are implicitly universally quantified.

organization in a given view and the predicate $consider(action, activity)$ specifies that a given action is considered in the organization as an implementation of a given activity².

We also need to specify access control requirements that depend on *contextual* conditions for instance:

- R1: A nurse is permitted to consult medical records *in a context of urgency*,
- R2: A physician is permitted to consult medical records *in his or her office during work hours*.

Contexts are modelled using the predicate $hold(subject, action, object, context)$ that specifies conditions to be satisfied to consider that a given subject executes a given action on a given object in some context. For instance, a context *in_office* is specified by the following rule:

$$subject_of_office(Subject, Office) \wedge location(Subject, Office) \\ \rightarrow hold(Subject, Action, Object, in_office)$$

Using these different concepts, an access control policy is simply defined by a set of facts having the form: $permission(Role, Activity, View, Context)$. For instance, rules R1 and R2 are respectively modelled by the two following facts:

R1: $permission(nurse, consult, med_record, urgency)$

R2: $permission(physician, consult, med_record, in_office \& working_hours)$

Notice the possibility in the second rule to combine contexts using conjunction (negation or disjunction would be also possible).

Given an access control policy (which possibly depends on contextual conditions), the objective of the access controller consists in deciding if a given subject is actually permitted to execute some action on a given object. This is modelled by the following rule:

$$permission(Role, Activity, View, Context) \wedge \\ empower(Subject, Role) \wedge consider(Action, Activity) \wedge \\ use(Object, View) \wedge hold(Subject, Action, Object, Context) \\ \rightarrow is_permitted(Subject, Action, Object)$$

3.2 Prohibition and management of conflicts

In recent models, it is generally accepted that access control policies may not only specify permissions but also prohibitions [12]. For instance, in the OrBAC model, it is possible to specify organizational prohibitions using the predicate $prohibition(role, activity, view, context)$ and derive concrete prohibitions using the predicate $is_prohibited(subject, action, object)$.

When the access control policy contains both permissions and prohibitions, a conflict occurs when it is possible to derive both $is_permitted(s, a, o)$ and $is_prohibited(s, a, o)$ for the same subject, action and object.

Several papers have investigated the problem of conflict detection and management (see for instance [4, 12]). The solution is generally based on assigning

² In OrBAC, the organization is made explicit in every predicate but here, to simplify, the organization is left implicit since we consider always only one organization.

priorities to security requirements so that when a conflict occurs between two requirements, the requirement with the higher priority takes precedence.

This is basically the approach suggested in the OrBAC model [12]. It consists in detecting and managing *potential* conflicts. A potential conflict exists between a permission and a prohibition if these two requirements may possibly apply to the same subject, action and object. There is no such potential conflict between two requirements if these requirements are *separated*. In OrBAC, separation between requirements is defined as follows: Two requirements are separated if their respective roles are separated, or their views are separated, or their activities are separated, or their contexts are separated. Two roles are separated if it is impossible to simultaneously assign a subject to these two roles. Separations of activities, views and contexts are similarly defined.

Since no conflict can occur between separated requirements, it is sufficient to assign priorities between every pair of non separated permission/prohibition requirements. This guarantees that all actual and potential conflicts are solved.

Notice that when using both permissions and prohibitions to specify an access control policy, it is actually possible to define two different decision procedure called *open* policy and *closed* policy. In the open policy, an access is accepted only if it is explicitly permitted by the policy, else it is rejected. In the closed policy, an access is rejected only if it is explicitly prohibited, else it is accepted.

Since an open policy is generally not equivalent to a closed policy, this means that when specifying an access control policy, it is generally assumed that a prohibition is not equivalent to a non permission³. Thus formalism based on Standard Deontic Logic (SDL) would not be appropriate. We need a more complex deontic formalism such as the one suggested in [9].

4 Intrusion policy

The primary objective of computer security is actually to protect the information system against intrusions. An intrusion is an action or a sequence of actions (also called an intrusion scenario) that exploits a *vulnerability*.

Many intrusions actually correspond to known intrusions that exploit known vulnerabilities. To detect known intrusions, most intrusion detection systems (IDS) implement techniques called *misuse* detection to recognize a *signature* of the intrusion. A signature specifies evidences that actions corresponding to the intrusion have been executed. Current implementations work quite well when the intrusion corresponds to an elementary action but do not provide so good results in case of intrusion scenarios.

Detection of unknown (or new) intrusions is much more complex. Current techniques, called anomaly detection, attempt to detect abnormal behavior that would reveal an intrusion. However, the implemented techniques are far from being perfect and generate many false positives (an alert is launched whereas there is no intrusion) and also false negatives (no alert is launched whereas an intrusion actually occurs).

³ However, conflict resolutions guarantees that prohibition *implies* not permission.

Even if an intrusion is generally defined as a violation of the security policy, there is no approach that attempts to include intrusion detection requirements in the security policy specification.

This is the purpose of this section to investigate this issue. We start investigating the case of known intrusions and then move to new intrusions.

4.1 Security policy for known intrusions

Since an intrusion corresponds to a malicious behavior, it seems appropriate to specify that such malicious behaviors are prohibited. Thus an intrusion detection policy corresponds to a set of prohibition requirements. Regarding known intrusions, the action used to exploit the vulnerability can be explicitly specified and prohibited. The following example illustrates the approach.

- **Example:** The Land Attack is a known intrusion that consists in forging illegal IP packets where the IP addresses of the source is equal to the destination. The impact of this intrusion is that it may cause a denial or service on the server which receives such packets. This intrusion is taken into account in the intrusion detection policy by the following prohibition:
R3: *prohibition(any_host, send_IP_packet, same_source_destination, default)*
In this requirement, a subject is a network host and *any_host* is a role assigned to every network host, *send_IP_packet* corresponds to the activity of sending packets using the IP protocol, *same_source_destination* is a view that contains any IP packet with a source IP address equal to its destination IP address and *default* is the default context which is always active.

As mentioned in the introduction, one advantage of formally specifying such prohibitions in the security policy is that it is then possible to analyze possible conflicts between other security requirements. For instance, the access control policy may include a filtering requirement specifying that hosts assigned to the role *private host* are permitted to open HTTP connection with the Internet:

R4: *permission(private_host, open_HTTP, to_Internet, default)*

Since it is possible to use an HTTP connection to send a Land Attack, requirements R3 and R4 are conflicting. It is important to detect and solve such conflicts and in our example, it is likely that requirement R3 should have higher priority than R4 so that hosts from the private zone are prohibited to launch the Land Attack using HTTP connection. The approach defined in [12] provides means to detect and solve this kind of conflicts.

Another advantage is that it is possible to use a formal specification of the intrusion detection policy to automatically configure IDSs, for instance we have defined a process to automatically deploy intrusion detection requirements such as R3 onto the Snort IDS⁴ [31].

Requirement R3 actually corresponds to an elementary intrusion that can be executed by a single action. Unfortunately, many intrusions require several

⁴ Snort is a network intrusion detection system that uses a signature based approach.

actions in sequence or in parallel to be executed. This is called an intrusion scenario and many examples of such scenarios could be given such as worms like Nimda or distributed denial of service intrusions like Trinoo [22].

It is possible to take into account known intrusion scenarios in the intrusion detection policy by specifying prohibitions. The language presented in section 3, restricted to permissions and prohibitions that apply to elementary actions, cannot express these intrusion scenarios. However, it can be extended, and norms that apply to non elementary actions (sequence, parallelism, ...) have already been extensively investigated in the deontic logic literature (see [25] for instance).

The analysis of conflicts is more complex when the policy includes security requirements on non elementary actions corresponding to intrusion scenarios. This problem is further discussed in section 6.

Regarding the deployment of these requirements, notice that classical IDSs only manage intrusions corresponding to elementary actions. However, there are research prototypes that could be used for this purpose. For instance, the approach suggested in [26] is based on specification of chronicles to represent intrusion scenarios so that prohibitions could be translated into chronicles to automatically configure this prototype.

Notice that the enumeration of every known intrusion scenarios is a complex and fastidious task. Another problem is that it is difficult to find the appropriate level of description of the intrusion scenario. If the description is not precise enough, then false alerts could be launched (false positive). But if the description is too precise, then variants of the intrusion scenario could not be detected (false negative). This is one of the issue we attempt to address in section 4.2.

4.2 Security policy for new intrusions

Detection of new intrusions is still a major issue of computer security. Current approaches based on anomaly detection attempts to recognize abnormal behavior of subjects but are far from giving perfect results. In this section, we suggest an approach to specify security requirements associated with new intrusions.

This approach is based on the specification of security objectives. A security objective is a condition on the state of the information system that should be enforced. Of course, these objectives depend on the information system to be protected but it is generally considered that they can be classified into confidentiality, integrity and availability requirements. From the point of view of the defender, an attacker is a subject that attempts to violate a security objective. Thus, we call an intrusion objective the negation of a security objective. We provide examples of security objectives:

- $server(h, DNS) \wedge denial_of_service(h)$
i.e. the DNS server is in denial of service.
- $get_access(s, root, h) \wedge \neg(authorized_access(s, root, h))$
i.e. s illegally gets a root access on a given host h .

If S is a state formula that represents an intrusion objective, then the associated security requirement specifies that it is forbidden for any subject s to

bring it about that S is true: $forbidden(E(s, S))$

where E represents the *bring it about* modality (see for instance [29])⁵.

The bring it about modality makes implicit the action which is executed to get the effect S . This action may actually correspond to a new intrusion. The interest of this approach is twofold:

1. If there are sensors that could detect that some intrusion objectives are achieved, it is possible to infer that some (possibly new) intrusion occurs.
2. If there is a library of elementary actions modelled through their pre and post conditions⁶, then the approach can be also used to detect new intrusion scenario. This is the approach suggested in [11] in which a mechanism is defined to correlate actions and detect when an intrusion objective can be achieved by these correlated actions. The specification of the library of elementary actions is generally easier to manage than the explicit description of entire intrusion scenarios suggested in section 4.1 with the advantage that new intrusion scenarios can be detected.

5 Reaction policy

As its name points it out, this policy is activated to react against an intrusion. It is a set of rules that specify what happens in case of violation (or attempt of violation) of some requirements of the security policy. According to these (attempts of) violations and their impacts on the target information system, new permissions, prohibitions or obligations are activated and pushed in the appropriate security components. For instance, if an intrusion occurs, and the alert diagnosis identifies the path of the attack or the equipments targeted by this attack and used to reach the intrusion objectives, (1) some packet flows have to be rejected or at least redirected or (2) some of the vulnerable equipments used by the attack have to be stopped or at least isolated typically to contain its spread in the whole system. As suggested in [16], a first form of reaction would be to update the access control policy by activating and deploying new permissions or prohibitions. For instance, a rule:

– R5: $permission(private_host, open_TCP, to_hostObelix, default)$,

might be replaced by a new one such as⁷:

– R6: $prohibition(any_host, open_TCP, to_hostObelix, syn_flooding)$.

In the second case, a reaction requirement may be specified by means of obligations. We actually consider two different kinds of obligations called server-side obligation and client-side obligation. A server-side obligation must be enforced

⁵ The *forbidden* and *prohibition* operators clearly refer to the same concept. In the following and to avoid confusion, we shall use *prohibition* when we refer to an explicit action and *forbidden* when we refer to an implicit action through the bring it about operator.

⁶ The pre condition represents the condition that must be true before executing the action and the post condition represents the effect of executing the action.

⁷ Syn flooding is a denial of service attack against the TCP protocol.

by the security components controlled by the security server and generally corresponds to immediate obligations. R7 is an example of such rules expressed in the OrBAC model:

– R7: *obligation(mail_daemon, stop, mailserver, imap_threat)*

Client side obligations generally correspond to obligations that might be enforced after some delay. Several papers have already investigated this problem and suggested models to specify obligation with deadlines [7, 13, 8, 17]. For instance, if there is an intrusion that attempts to corrupt an application server by a Trojan Horse intrusion, then this server must be quarantined by the administrator within a deadline of 10s. R8 provides a specification of this requirement:

– R8: *deadline_obligation(administrator, quarantine, application_server, trojan_horse_threat, before(10))*

where *deadline_obligation* can be used to specify one more attribute that corresponds to the deadline condition *before(10)*.

As intrusions correspond to some implicit prohibited behaviours and actions, the security requirements inferred by the need to react correspond to contrary to duty requirements. Management of contrary to duty is known to cause trouble (see “pragmatic oddity” [30]). In our approach, management of conflicts is based on classification with respect to the context of activation. In fact, we consider three different types of activation contexts: *threat*, *operational* and *minimal*.

The *operational contexts* aim at describing traditional operational policy [14]. They may correspond to temporal, geographical or provisional contexts (i.e. contexts that depend on the history of previous executed actions).

Intrusion classes are associated with *threat contexts* and for each threat a security rule is defined in the (reaction) policy. *Threat contexts* are activated when a violation of the security policy is detected and are used to specify the reaction policy. The activation of these contexts (*hold* facts, see section 3), leads to the instantiation of the policy rules in response to the considered threat. For instance, a Syn-flooding attack is reported by an alert with a classification reference equal to CVE-1999-0116 (corresponding to the CVE reference of a Syn-flooding attack), the target corresponds to some network host *Host* and some service *Service*. Then the synflooding context is specified as follows [16]:

– IC: *hold(corp, -, Service, Host, syn_flooding) ←
alert(Time, Source, Target, Classification),
reference(Classification, 'CVE - 1999 - 0116'),
service(Target, Service), hostname(Target, Host).*

Notice that, since the intruder is spoofing (masquerading) its source address in a Syn-flooding attack, the subject corresponding to the threat origin is not instantiated in the hold predicate. When an attack occurs and a new alert is launched by the intrusion detection system, new facts *hold* are derived for threat context *Ctx*. So, *Ctx* is then active and the security rules associated with this context are triggered to react to the intrusion.

Most of reaction requirements are in conflict with other access control requirements, i.e. the access control policy may specify a permission whereas the reaction policy specifies a conflicting prohibition that applies when an intrusion

is detected. For instance, HTTP is permitted when there is no intrusion but prohibited if an intrusion on the HTTP protocol is detected.

We suggest to solve these conflicts by assigning higher priority to the reaction requirement than the access control requirement. Since access control requirements are associated with operational contexts whereas reaction requirements are associated with threat contexts, we actually consider that threat contexts have higher priority than operational contexts.

However, there are some security requirements such as availability requirements that must be preserved even if an intrusion occurs. For instance, the access to the email server must be preserved even if some intrusions occur. This is modelled as a minimal requirement. *Minimal contexts* then define high priority exceptions in the policy, describing minimal operational requirements that must apply even in case of characterized threat.

So, using an algebra of contexts and priority assignment to security rules, we consider two parameters to manage conflicting situations called *criticity* and *specificity*. A criticity parameter is used to assess context priority between the three defined categories of contexts *operational*, *threats* and *minimal*. We define an operator L_c to assess the level of criticity of contexts, so that if Ctx is a set of well formed contexts: $L_c: \text{Ctx} \rightarrow \{\text{ope}, \text{threat}, \text{min}\}$ with $\text{ope} < \text{threat} < \text{min}$. We define the criticity relation as follows: $c_1 <_c c_2 \iff L_{c1} < L_{c2}$. We consider also a specificity parameter that deals with inheritance and context composition, hierarchical specificity context inheritance. For instance, we say that c_2 is more specific than c_1 if $\text{sub_context}(c_2, c_1)$. We define specificity for contexts as follows: $c_1 \leq_s c_2 \iff \text{sub_context}(c_2, c_1)$ and $c_1 <_s c_2 \iff c_1 \leq_s c_2 \wedge \neg(c_1 = c_2)$. We have then defined two strategies to assess rule priorities in case of potential conflicts and prove that they are not conflicting strategies, that is we never obtain conflicting decisions when applying them (see [16]).

6 Discussion and open issues

As mentioned in the introduction, one of the interest of a formal specification is that it provides means to analyze conflicts. This is not an easy task because a security policy is an heterogeneous set of requirements. It corresponds to permissions, prohibitions and obligations and some requirements apply to explicit actions whereas others correspond to unknown actions. We suggest modelling these last requirements using the bring it about modality. This is especially useful to specify security requirements associated to new intrusions.

The next problem is then to analyze conflicts when the policy combines such heterogeneous requirements. Our suggested solution consists in reformulating security requirements that apply to explicit actions into requirements that use the bring it about modality.

For this purpose, we need a formal specification of various actions used to specify the policy through their pre and post conditions. Then, let us assume that a given subject s is prohibited to execute a given action a on some object o in a given context c and let $\text{pre}(a, o)$ and $\text{post}(a, o)$ be respectively the pre and

post conditions associated with the execution of action a on object o . Then this requirement is translated into the following security requirement:

$forbidden(E(s, a, post(a, o)), c \& pre(a, o))$

where $E(s, a, p)$ means subject s brings it about p by executing action s and $forbidden(p, c)$ is a diadic modality to specify that p is forbidden in context c .

We now obtain an homogeneous set of security requirements that we can analyze to detect conflicts. Defining a complete analysis still represents further work. However, we can already state the following principles:

- Let $permitted(E(s, a, p_1), c_1)$ and $forbidden(E(s, a, p_2), c_2)$ be two security requirements. These requirements conflict if p_1 implies p_2 and c_1 and c_2 are consistent.
- Let $permitted(E(s, a, p), c)$ and $forbidden(E(s, S))$ be two security requirements where S is an intrusion objective. These requirements are conflicting if p implies S .
- Let $permitted(E(s, a_1, p), c)$ and $forbidden(E(s, a_2, p), c)$ be two security requirements where a_1 and a_2 are different actions. These requirements are not necessarily conflicting. For instance, let a_1 be the action “Authentication using a credit card” and a_2 be “Authentication using a password” and p the proposition “ s is authenticated”. Then it is *not* conflicting to state that s is permitted to bring it about p by executing a_1 but prohibited to bring it about p by executing a_2 .
- Let $permitted(E(s, a_1, p), c_1)$ and $forbidden(E(s, a_2, p), c_2)$ be two security requirements where a_1 is a non elementary scenario. These requirements are conflicting if action a_2 is part of scenario a_1 and c_1 and c_2 are consistent.

We plan to develop a complete analysis as an extension of the approach suggested in [9].

7 Conclusion

Specifying a security is a central issue when developing a secure information system. Since a security policy may include very different requirements, it is essential to use an homogeneous formal model to specify these different requirements and deontic logic provides such an adequate formalism.

Traditional security policy models only consider norms that apply to explicit elementary actions. In this paper, we first focus on intrusion detection policies and show that these traditional models are not expressive enough to specify security requirements corresponding to the detection of known non elementary intrusion scenarios. There are also not appropriate to specify security requirements that correspond to new unknown intrusions. For this last purpose, we suggest using the bring it about modality and security requirements correspond to prohibition to bring it about that some security objectives are violated.

The security policy would then not be complete if it does not include a reaction policy that specifies what happens in case of violation of some security requirement. We show that these requirements correspond to contrary to duty norms. Contrary to duty has been extensively investigated in the deontic logic

literature and several proposals have been suggested to solve problems such as the Chisholm paradox or the pragmatic oddity. However, to our best knowledge, it is the first time practical applications of these works are investigated in the context of security policies. We suggest an approach to manage conflicts between reaction requirements and other security requirements based on the definition of priorities between contexts associated with the activation of these different requirements. We consider three different types of context called operational, threat and minimal contexts. Threat contexts are associated with reaction requirements and have higher priority than operational contexts. However, since reacting may have some negative side effects on the information system, we also consider minimal contexts associated with security requirements that must be preserve even when reactions are activated. Minimal contexts have higher priority than other contexts including threat contexts.

We finally address the issue of analyzing consistency of these different requirements. Since the security policy may include heterogeneous requirements corresponding to norms that apply to explicit actions and others to implicit actions, our proposal consists in translating all the requirements into norms specified using the bring it about modality. It is then possible to analyze possible conflicts between the different requirements. This approach requires further work to be validated and we plan to take our inspiration into [9] to define a complete algorithm to detect and solve conflicts in this case.

Acknowledgement: This work is supported by the European CELTIC RED (Reaction After Detection) and ANR Polux (Policy Unified Expression) projects.

References

1. A. Abou El Kalam, R. El Baida, P. Balbiani, S. Benferhat, F. Cuppens, Y. Deswarte, A. Miège, C. Saurel, and G. Trouessin. Organization Based Access Control. In *4th IEEE Policy*, June 2003.
2. S. Ayed, N. Cuppens-Bouahia, and F. Cuppens. An Integrated Model for Access Control and Information Flow Requirements. In *11th ASIAN*, pages 111–125, 2007.
3. D. Bell and L. LaPadula. Secure Computer Systems: Unified Exposition and Multics Interpretation. Technical Report ESD-TR-75-306, MTR-2997, MITRE, Bedford, Mass, 1975.
4. S. Benferhat, R. El Baida, and F. Cuppens. A Stratification-Based Approach for Handling Conflicts in Access Control. In *8th ACM Symposium on Access Control Models and Technologies (SACMAT'03)*, Lake Como, Italy, June 2003.
5. E. Bertino, P. A. Bonatti, and E. Ferrari. TRBAC: A temporal role-based access control model. *ACM TISSEC*, 4(3):191–233, 2001.
6. E. Bertino, B. Catania, M. L. Damiani, and P. Perlasca. Geo-rbac: a spatially aware rbac. In *10th ACM SACMAT*, June 1-3 2005.
7. J. Broersen, F. Dignum, and J.-J. Meyer V. Dignum. Designing a Deontic Logic of Deadlines. In *DEON*, Madeira, Portugal, May 2004.
8. J. Brunel, J.-P. Bodeveix, and M. Filali. A State/Event Temporal Deontic Logic. In *DEON*, Utrecht, The Netherlands, July 2006.
9. 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*, Bologna, Italy, 1998.

10. F. Cuppens. Roles and Deontic Logic. In *Second International Workshop on Deontic Logic in Computer Science*, Oslo, Norway, 1994.
11. F. Cuppens, F. Autrel, A. Miège, and S. Benferhat. Recognizing Malicious Intention in an Intrusion Detection Process. In *HIS*, Santiago, Chili, 2002.
12. F. Cuppens, N. Cuppens-Boulahia, and M. Ben Ghorbel. High Level Conflict Management Strategies in Advanced Access Control Models. *Electronic Notes in Theoretical Computer Science*, 186:3–26, 2007.
13. F. Cuppens, N. Cuppens-Boulahia, and T. Sans. Nomad: A Security Model with Non Atomic Actions and Deadlines. In *18th IEEE CSFW*, pages 186–196, 2005.
14. F. Cuppens and A. Miège. Modelling Contexts in the Or-BAC Model. *ACSAC*, 2003.
15. F. Cuppens and A. Miège. Administration Model for Or-BAC. In *Computer Systems Science and Engineering (CSSE'04)*, volume 19, May 2004.
16. H. Debar, Y. Thomas, N. Boulahia-Cuppens, and F. Cuppens. Using Contextual Security Policies for Threat Response. In *DIMVA*, Berlin, Germany, July 2006.
17. R. Demolombe, P. Bretier, and V. Louis. Norms with Deadlines in Dynamic Deontic Logic. In *ECAI*, Riva del Garda, Italy, September 2006.
18. R. Demolombe and V. Louis. Norms, Institutional Power and Roles: Towards a Logical Framework. In *ISMIS*, pages 514–523, Bari, Italy, 2006.
19. R. Demolombe and V. Louis. Speech Acts with Institutional Effects in Agent Societies. In *DEON*, pages 101–114, 2006.
20. M. Ben Ghorbel, F. Cuppens, N. Cuppens-Boulahia, and A. Bouhoula. Managing Delegation in Access Control Models. In *15th ADCOM*, 2007.
21. J. Goguen and J. Meseguer. Unwinding and Inference Control. In *IEEE Symposium on Security and Privacy*, Oakland, 1984.
22. J. Harrington. *Network Security: A Practical Approach*. The Morgan Kaufmann Series in Networking, 2005.
23. M. Harrison, W. Ruzzo, and J. Ullman. Protection in operating systems. *CACM*, 19(8):461–471, August 1976.
24. M. Hilty, A. Pretschner, D. A. Basin, C. Schaefer, and T. Walter. A Policy Language for Distributed Usage Control. In *ESORICS*, pages 531–546, 2007.
25. J.-J. Meyer. A different approach to deontic logic: deontic logic viewed as a variant of dynamic logic. *Notre Dame Journal of Formal Logic*, 29(1):109–136, 1988.
26. B. Morin and H. Debar. Correlation of Intrusion Symptoms: An Application of Chronicles. In *RAID*, Pittsburgh, PA, September 2003.
27. O. Pacheco and J. Carmo. A Role Based Model for the Normative Specification of Organized Collective Agency and Agents Interaction. *Autonomous Agents and Multi-Agent Systems*, 6(3):145–184, 2003.
28. J. Park and R. S. Sandhu. The UCONABC usage control model. *ACM Trans. Information and System Security*, 7(1), 2004.
29. I. Pörn. *Action Theory and Social Science; Some Formal Models*, volume 120 of *Synthese Library*. D. Reidel, Dordrecht, 1977.
30. H. Prakken and M. Sergot. Contrary-to-duty obligations. *Studia Logica*, 57(1):91–115, 1996.
31. S. Preda, N. Cuppens-Boulahia, F. Cuppens, J. Garcia-Alfaro, and L. Toutain. Reliable Process for Security Policy Deployment. In *International Conference on Security and Cryptography (Secrypt 2007)*, Barcelona, Spain, July 2007.
32. R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. Role-Based Access Control Models. *Computer*, 29(2):38–47, 1996.
33. L. W. N. van der Torre. Violated Obligations in a Defeasible Deontic Logic. In *ECAI*, Amsterdam, The Netherlands, 1994.