Reaction policy model based on dynamic organizations and threat context

Fabien Autrel, Nora Cuppens-Boulahia, Frédéric Cuppens

Telecom-Bretagne, 35576 Cesson Sévigné (France)

Abstract. The tasks a system administrator must fulfill become more and more complex as information systems increase in complexity and connectivity. More specifically, the problem of the expression and update of security requirements is central. Formal models designed to express security policies have proved to be necessary since they provide non ambiguous semantics to analyze them. However, such models as RBAC or OrBAC are not used to express reaction requirements which specify the reaction policy to enforce when intrusions are detected. We present in this article an extension of the OrBAC model by defining dynamic organizations and threat contexts to enable the expression and enforcement of reaction requirements.

1 Introduction

Information systems are becoming more and more complex and system administrators have to face several problems. In this context, specifying a security policy becomes a very tedious and error prone task. Using a formal approach brings several benefits: the policy expression is non-ambiguous and tools can be used to analyze a security policy[CCBG07] and deploy it[PCBC+07].

A security policy may express very different requirements and can contain different types of policies: authentication, access control, flow control and usage control requirements. However, the reaction policy, which expresses the security requirements related to the detection of attacks, is generally not considered as part of the security policy. This reaction policy expresses which reaction requirements (RR) should be enforced when the security policy is violated.

Specifying and deploying RR is actually not an easy task. [DTCCB07] is a preliminary work in this direction but we shall explain why this approach is not fully satisfactory. In this paper, we shall define a model based on the concept of dynamic organization created to manage intrusions which significantly enhances the approach suggested in [DTCCB07].

In this paper we suggest to express the RR using the OrBAC model. Section 2 presents the problems related to the expression of the RR. Section 3 reviews some related work. Section 4 outlines the OrBAC model and section 5 explains how we use it to model the RR. Section 6 presents three examples of RR related to an elementary attack and two multi-step intrusions. Section 7 presents the implementation in the MotOrBAC 2[1] support tool. We conclude in section 8.

[1] http://motorbac.sourceforge.net

2 Problematic

In this paper, the task of expressing the RR takes place into a supervision loop (figure 1) where the processes of intrusion detection and policy specification/enforcement interact with each other. In this approach we consider that the reaction module implements a policy-based reaction process, i.e, upon the detection of an attack, the reaction process consists in enforcing new security rules.

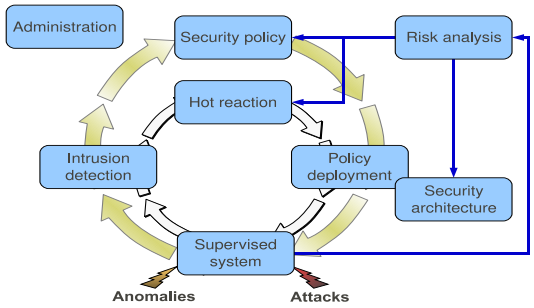

Fig. 1. Supervision loop

The RR expressed in a reaction policy may include prohibitions, obligations and permissions. Those requirements can be expressed as a set of reaction rules which are specified for each attack that the intrusion detection systems may detect. For example the informal specification of the RR for a brute force attack executed against a UNIX account of a SSH server could be the following requirements:

1. Block the attacker
2. Prevent the attacker from getting a user access to the target computer
3. Suspend the victim account
4. Warn the user who owns the attacked account and tell him he must change his/her password

We shall show that some requirements can be formalized as obligations (requirements 1, 3, 4) and prohibitions (requirement 2). Additionally, those requirements should be only enforced in the context of a brute force attack. Note that the fourth requirement can be divided into two requirements: the user must be informed that his account has been compromised and once he has been warned, he must change his password. In the following section, we show that the policy model which should be used to express the RR must satisfy several properties.

Formalizing the reaction policy Let us consider a set of Attacks A. Writing the RR for an attack A_i in A consists in specifying a set of security rules R_i which are activated when the attack A_i is detected.

The notion of role introduced in the RBAC model is mandatory to specify RR since we cannot know in advance the ip addresses and/or identity of the attackers and victims. However, RBAC is not sufficiently expressive to express RR for several reasons:

- The Attacker and Victims RBAC roles names must be unique for each A_i. Actually if two rules $r \in R_1$ and $r' \in R_2$ specified for attacks A_1 and A_2 use the same role *attacker*, a problem appears when the set of rule R_1 must be activated because attack A_1 has been detected. Indeed if the attacker of A_1 instance is assigned to role *attacker*, r will be activated and r' too despite the fact that no occurrence of A_2 has been detected.
- Since the original RBAC model specifies that user-role assignment is handled statically [FSG+01], the rules R_i for each attack A_i cannot be dynamically activated. Even if considering a mechanism which dynamically assigns alert subjects to roles when attacks are detected, the model lacks expressiveness to associate each R_i to a given A_i.
- The concept of session in the RBAC model could be used to assign subjects to roles when an attack is detected, but only one subject can be assigned to a session. This raises a problem for distributed attacks involving several attackers and victims.
- The dynamic nature of a reaction policy cannot be modeled with RBAC. For example the set of rules R_i associated with attack A_i should be activated when the attack is detected, but at least a subset of those rules should be deactivated after the end of the attack. This could possibly be modeled through the use of several roles, but this adds even more artificial roles.

An extended RBAC model including contexts [MF03] can express the link between an alert classification and a rule through the use of security context if this context is activated when a given intrusion is detected. However, as said above, it is impossible to associate activated rules with the corresponding occurrence of an attack. Moreover the meaning of the context in this extension is that of a condition evaluated when an access is made to the system.

We propose to use the concept of dynamic organization to address the problem of multiple occurrences of an attack. Intuitively, a new dynamic organization is created to manage a given intrusion and different subjects will be assigned locally to roles (like *victim*, *attacker*) within this dynamic organization. Once the intrusion is processed, this dynamic organization is deactivated.

3 Related work

A majority of research works has focused on the detection of intrusions but few works exist in the field of response to intrusions. Incidentally, very few articles deals with the expression of RR and its enforcement. Some taxonomies of intrusion responses have been defined ([Fis96,CP00,SBW07]). Fisch's taxonomies [Fis96] are system-oriented, they distinguish intrusion response systems by degree of automation and activity of triggered response. Carver and Posh's [CP00]

taxonomy deals with response, but from the attack side. Their taxonomy includes 6 dimensions and does not classifies responses. In [Bra98], Brackney says that detection is the first step leading to a response process and explains that taking action after detection is not a trivial task. Actually reacting quickly and efficiently is difficult considering the time needed to analyze an intrusion, hence an automated and autonomous response system is desirable.

In [CCBB+08,DTCCB07], the OrBAC model is used to express the RR. A new type of OrBAC context is introduced to manage intrusion detection alerts expressed in IDMEF (Intrusion Detection Message Exchange Format)[HCF07]. Such a context specifies the alert classification, an identifier that defines which attack is detected, that triggers its activation and the mapping between alert attributes and concrete entities (subjects, actions and objects). However, the approach lacks a mechanism to specify the mapping between concrete entities mapped with the alert and abstract entities (roles, activities and views) specified in the OrBAC security rules. Moreover the mapping between alert attributes and concrete entities is specified for each context although a more generic mapping that could apply to every threat context would be more convenient and scalable.

In this paper, we propose to use the concept of threat contexts, but the definition we give is different from the one proposed in the aforementioned articles. We separately specify the alert classification which triggers the context activation and the mapping between alert attributes and concrete entities. Abstract entity definition are used to specify this mapping and assign concrete entities to abstract entities. Moreover, we introduce the concept of threat organisations. Hence, we can define intrusion-dependent roles like *attacker* and *victim* and consider that subjects are assigned to these roles, but locally to this threat organization. Thus, it is possible to consider that two different subjects are both assigned to the role *victim* but in two different threat organizations. This will be typically the case if these two subjects are victims of two different intrusions.

4 Introduction to OrBAC

OrBAC [KBB+03] aims at modelling a security policy centered on the concept of organization. An organization defines and manages a security policy. An OrBAC policy specification is done at the organizational level, also called the abstract level, and is implementation-independent. The enforced policy, called the concrete policy, is inferred from the abstract policy.

This approach makes all the policies expressed in the OrBAC model reproducible and scalable. Actually once the concrete policy is inferred, no modification or tuning has to be done on the inferred policy since it would possibly introduce inconsistencies. The inferred concrete policy expresses security rules using subjects, actions and objects. The abstract policy, specified at the organizational level, is specified using *roles*, *activities* and *views* which abstract the traditional *subjects*, *actions* and *objects*.

The OrBAC model uses a first order logic formalism with negation. However, since first order logic is generally undecidable, we have restricted our

model in order to be compatible with a stratified Datalog program [Ull89]. A stratified Datalog program can be evaluated in polynomial time. In the rest of this article the security rules must correspond to a stratified Datalog program. We use a Prolog-like notation where terms beginning with an upper case are variables and terms beginning with a lower case are constants. The fact $parent(john, jessica)$. says that *john* is a parent of *jessica*. A rule such as $grandparent(X, Z) : -parent(X, Y), parent(Y, Z)$. means that X is a grandparent of Z if Y exists such that X is a parent of Y and Y is a parent of Z.

Using this formalism, each organization specifies its own security rules. Some *role* may have the permission, prohibition or obligation to do some *activity* on some *view* given an associated *context* is true. The *context* concept [CCB08] has been introduced in OrBAC in order to express dynamic rules. Those security rules are represented using 5-ary predicates:

- $permission(org, role, activity, view, context)$ means that in organization org, role *role* is authorized to perform activity *activity* on view *view* if context *context* is true.
- the *prohibition* and *obligation* predicates are similarly defined but express different security requirements. The *prohibition* predicate states that a role is not authorized to perform some activity on some view when a given context is true. The *obligation* predicate means that some role *must* do some activity on some view when the associated context is true.

For example, the expression:
permission(hospital, nurse, consult, medical_record, emergency)
means that nurses can access the patients medical records in the context of an emergency. Security rules can be hierarchically structured so that they are inherited in the organization, role, activity and view hierarchies (see [CCBM04]). Since a security policy can be inconsistent because of conflicting security rules (for example a permission can be in conflict with a prohibition), it is necessary to define strategies to solve those conflicts [CCBG07].

Once the security policy has been specified at the organizational level, it is possible to instantiate it by assigning concrete entities to abstract entities. To do so, three ternary predicates have been defined to assign a subject to a role, an action to an activity and an object to a view:

- $empower(org, subject, role)$: specifies that in organization org, subject *subject* is empowered in role *role*.
- $consider(org, action, activity)$: specifies that in organization org, action *action* implements activity *activity*.
- $use(org, object, view)$: specifies that in organization org, object *object* is used in view *view*.

For example, the fact $empower(hospital, john, surgeon)$ states that *john* is empowered in the role *surgeon* in the *hospital* organization.

Contexts are defined through logical rules which express the condition that must be true in order for the context to be active. In the OrBAC model such rules have the predicate *hold* in their conclusion:

– $hold(org, subject, action, object, context)$: specifies that in organization org, subject $subject$ does action $action$ on object $object$ in context $context$.

As suggested in [CCB08], contexts can be combined in order to express conjunctive contexts (denoted &), disjunctive contexts (denoted $\oplus$) and context negation (denoted $\overline{ctx}$, ctx being a context name).

Using this model, concrete security rules applying to subject, actions and objects can be inferred as specified by the following derivation rule:

$is\_permitted(Subject, Action, Object) :-$
$permission(Org, Role, Activity, View, Context)$,
$empower(Org, Subject, Role)$,
$consider(Org, Action, Activity)$,
$use(Org, Object, View), hold(Org, Subject, Action, Object, Context)$.

Similar rules are defined to infer the *is_prohibited* and *is_obliged* predicates which represent concrete prohibitions and concrete obligations.

5 Reaction policy specification

As said in section 2, when an intrusion A_i is detected, a set of security rules R_i should be activated. We showed that modeling a reaction policy in RBAC requires to introduce at least as many roles as attacks and that the link between roles and intrusion detection alerts cannot be established. The concept of organization in the OrBAC model is well-suited to address those problems.

Actually in OrBAC a subject is empowered into a role within an organization. It is possible to define generic intrusion related roles such as *attacker* and *victim* and activate a subset of rules R_i into an organization O_i thanks to the contexts associated with the rules. Those contexts are activated upon the detection of an occurrence of A_i. This way the set of concrete rules inferred from R_i are linked with the occurrence of A_i which activates the contexts used in R_i through the organization O_i. In this approach the dynamic organizations associated with the detection of attack occurrences are dynamically and automatically created instead of being handled manually by an administrator. In the context of intrusion detection, we call *threat organizations* these dynamic organizations. In some sense, there is some analogy with the activation of a session in the RBAC model. However, there are two main differences with a session: an organization threat is automatically created when an alert is received to process the associated attack and there are several subjects involved in an organization threat, typically one or several attackers and one or several victims.

In the remaining of this article we consider that the alerts are expressed in the IDMEF format. The IDMEF format is an object-oriented representation of the data generated by Intrusion Detection Systems (IDSs). It defines several classes such as *Source* and $Target$ to represent respectively the attacker and the victim of an event detected by an IDS and the $Classification$ class which uniquely identifies the event.

5.1 Threat organization and threat context

We assume that the rules in R making up the RR are defined in an organization called *supervision*. The threat organizations are created as sub-organizations of *supervision* so that the reaction policy is inherited. When a new IDMEF alert is generated, a new threat organization is created to manage it. Let us consider an alert $alert_{ij}$ which is the alert generated upon the detection of the j^{th} occurrence of attack A_i. The following predicate becomes true and binds the alert and the new organization $threat\_org_{ij}$:

$threat\_context\_management(alert_{ij}, threat\_org_{ij})$

We defined in [CBCdV+08], a complete ontological mapping from IDMEF schema onto abstract OrBAC roles, activities and views managed by the supervision organization. Due to space limitation, we do not present this mapping here but simply introduce those abstract entities we use in the remainder of this paper:

- roles *attacker* and *victim*
- activity *attack*
- views *to_victim* and *to_attacker* to additionally describe various information related to the victims and attackers of an intrusion.

We then introduce a new context type (see [CCB08] for other types of contexts) called threat context. A threat context is activated in *threat_org* to manage a given alert (modelled using the predicate *threat_context_management(Alert, Theat_org)*) if its definition matches the classification of the alert (for this purpose we use the predicate $mapping\_classification(Alert, Threat\_context)$). This context is active for every triple $\{subject, action, object\}$ and defined as follows:

$hold(Threat\_org, \_, \_, \_, Threat\_context) :-$
$threat\_context\_management(Alert, Threat\_org),$
$mapping\_classification(Alert, Threat\_context).$

This rule says that, if a given threat organization *threat_org* is associated with the management of an IDMEF alert *alert* and if this alert classification maps onto a threat context *threat_context*, then this threat context is activated in the threat organization *threat_org* for every subject, action and object (denoted by the do not care symbol "_").

Using this context in the specification of rules R_i, or a composition of context including it, allows the activation of the rules in R_i or a subset of R_i in *threat_org*. The activated rules for alert *Alert* are deleted when the threat is no more active by deleting organization *threat_org*.

Notice that in case of conflicts, we consider that security rules that depend on threat contexts have higher priority than other security rules (see [DTCCB07] for more explanation).

5.2 Mapping alert to abstract entities

By contrast to [DTCCB07], The mapping from an alert to the threat abstract entities in *threat_org* is done using role, activity and view definitions. Here is

an example of generic mapping between the source of an alert and the *attacker* role:

$$empower(Threat\_org, Subject, victim) :- \\ threat\_context\_management(Alert, Threat\_org), \\ mapping\_target(Alert, Subject).$$

The mapping from the alert to the abstract entities is independent from the specification of the threat context. This way a reaction policy can be updated by adding new rules corresponding to new threats generally without having to specify the mapping for each new threat.

5.3 Expression of security requirements

The reaction policy may express various security requirements as introduced in the example of section 2. Those requirements may include permissions activated in the context of an attack $attack_1$, for example the communication between two servers located in two different networks, which are normally independent, to backup critical data:

$$permission(supervision, data\_server, open\_ssh\_connection, \\ to\_backup\_server, attack_1\_ctx)$$

Here hosts assigned to the *data_server* role are authorized to backup their critical data on hosts assigned to the *to_backup_server* view. This may be implemented by adding rules in some firewalls and routers.

Prohibitions can also be part of the RR. Consider the case of an attacker trying to connect to a telnet server running on a router from outside a company's network. A possible reaction might be to block the traffic coming from the attacker's machine and going to the victim's machine:

$$prohibition(supervision, attacker, all\_traffic, victim, telnet\_attack\_ctx)$$

In this abstract rule, the $all\_traffic$ activity abstracts the network protocols so that there is no need to express the prohibition for each network protocol.

A reaction requirement may be expressed by means of obligations. For instance a web server may be vulnerable to a newly discovered vulnerability and it should be stopped when an attacker tries to exploit this new vulnerability:

$$obligation(supervision, web\_server\_daemon, stop, web\_server, new\_threat)$$

Note that we consider that this obligation is an immediate obligation, i.e. it must be fulfilled as soon as the associated context is true and is no more active when the context becomes false. Generally some obligations might be enforced after some delay, typically if the subject of the obligation is a human operator. Those obligations are called obligations with deadlines [GF05,CCBB+08]. Enforcing obligations with deadline is more complex than immediate obligations. In order to simplify both the implementation and expression of obligations, we consider only immediate obligation in the remainder of this paper.

Other examples of reaction policies which demonstrate the need for permissions, obligations and prohibitions as part of the reaction policy are presented in section 6.

5.4 Reaction process

The reaction process when an alert is received is the following:

1. creation of a threat organization org_{threat} associated with the new alert. org_{threat} is created as a sub-organization of the *supervision* organization.
2. activation of threat contexts in org_{threat}. Yet no concrete rules are inferred since no concrete entities are assigned to abstract entities in org_{threat}.
3. creation of concrete entities from the alert mapping. Role, activity and view definitions are evaluated to extract the data necessary to create the concrete entities from the alert.
4. assignment of subjects, actions and objects created from the alert to intrusion roles, activities and views in org_{threat}. This step and the previous one are specified through the abstract entity definitions.
5. activation of abstract security rules associated with the threat context into org_{threat}.
6. derivation of concrete security rules from activated abstract rules.
7. deployment of concrete security rules to configure security components.

The last step in this process is not covered by this article but an approach to deploy dynamic contextual security policies is proposed in [CCBB+08].

6 Reaction policy examples

In this section we present three examples of reaction policy. The first example specifies how to react when a buffer overflow is detected. The second and third examples demonstrate how to react to multi-step attacks: a brute force attack and a distributed denial of service (DDOS). The multi-step attacks examples assume that the information system on which the attack is detected uses a correlation engine to correlate the elementary attacks generated during the execution of the multi-step attacks (such as [CAB+06]). This correlation engine is part of the supervision loop presented in section 2. The reaction policy instantiation process is detailed in section 6.2.

6.1 Reacting against a buffer overflow

A possible counter-measure against a buffer overflow (BOF) is to isolate the machine from the network to correct the exploited vulnerability. We take the example of a BOF vulnerability in the WebDAV-enabled IIS 5.0 servers from Microsoft. The default threat abstract entities (see section 5) are used to write the abstract rules. Additionally, we define the *all_protocol* activity which abstracts the network protocols used by the machines involved in the attack, the *backup_web_server* view which abstracts the web server(s) started to replace a stopped web server and the *to_any_address* view which abstracts any ip address. $webdav\_bof\_ctx$ is the threat context defined for this attack. We define the following obligations and prohibitions:

– the victim must be isolated from the rest of the network, i.e. all traffic is prohibited from and to the victim.:

$prohibition(supervision, victim, all\_protocol, to\_any\_address, webdav\_bof\_ctx)$
$prohibition(supervision, any\_address, all\_protocol, to\_victim, webdav\_bof\_ctx)$

– obligation to start a backup web server while the attacked server is unavailable:

$obligation(supervision, administrator, start, backup\_web\_server, webdav\_bof\_ctx)$

– the vulnerable web server must be patched to remove the vulnerability:
$obligation(supervision, administrator, update, to\_victim, webdav\_bof\_ctx)$

Notice that, in that reaction scenario, all reaction rules may be activated in parallel.

6.2 Reacting against multi-step attacks

Brute force attack An example of attack requiring an alert correlation engine is a brute force attack. As said at the beginning of section 6, we assume an alert correlation engine has generated a global alert by fusing the elementary alerts corresponding to several failed logins on the same account using *ssh*.

As for the previous example, we introduce some additional abstract entities to express the reaction policy. The *rdp* role represents the components which deploy the reaction policy. The *victim_user* role abstracts the user being targeted by the attack. The *send_reset* activity abstracts the action(s) that disconnect the attacker from the victim. The *suspend_account* activity abstracts the action(s) taken by the administrator or a process to suspend a user account (which can be implemented by a script for different OSs like Linux/Unix, Microsoft windows, etc...). The *change_password* activity abstracts the actions that implement a password change. The *send_warning* activity abstracts the way a message is sent to a user to warn him/her that his/her account has been compromised. Finally we define a *to_victim_account* view which abstracts the users accounts. The mapping between an IDMEF alert and the *victim_user* role and *to_victim_account* view is done through the following role and view definitions (cf section 5.2):

$empower(Threat\_org, Subject, victim\_user) :-$
$threat\_context\_management(Alert, Threat\_org),$
$mapping\_target\_user\_name(Alert, Subject).$
$use(Threat\_org, Object, to\_victim\_account) :-$
$threat\_context\_management(Alert, Threat\_org),$
$mapping\_target\_user(Alert, Object).$

The following rules correspond to the RR example given in section 2 ($brute\_force\_ctx$ is the threat context defined for this attack):

– all traffic coming from the attacker and going to the victim is prohibited:
$prohibition(supervision, attacker, all\_protocol, to\_victim, brute\_force\_ctx)$

– the connexion between the attacker and victim must be interrupted:
$obligation(supervision, rdp, send\_reset, to\_attacker, brute\_force\_ctx)$

– the victim account must be suspended:
obligation(supervision, rdp, suspend_account, to_victim_account, brute_force_ctx)

– the victim must be warned that his/her account has been attacked:
$obligation(supervision, rdp, send\_warning, to\_victim\_user, brute\_force\_ctx)$

– once he/she has been warned that his/her account has been attacked, the victim must change his/her password:
$obligation(supervision, victim\_user, change\_password, to\_victim\_account, brute\_force\_ctx \ \& \ received\_warning\_ctx)$
The $received\_warning\_ctx$ context is true if the subject has received a warning through an email for example. Note the use of the context conjunction operator &.

Reaction policy instantiation example: Consider a brute force attack performed by a machine having ip address ip_A on a computer having for ip address ip_V against the account ac_V of user $user_V$. Suppose that an Intrusion Detection System (IDS) detects the attack and generates an alert m. The reaction policy for this attack occurrence is instantiated as follows:

1. When alert m is received by the reaction module, a threat organization $threat\_org_1$ is created as a sub-organization of organization $supervision$.
2. The threat context associated with the alert classification of m, namely the $brute\_force\_ctx$ context, is activated.
3. The relevant role and view definitions are evaluated so that the following statements become true:
   $empower(threat\_org_1, ip_A, attacker)$, $empower(threat\_org_1, ip_V, victim)$,
   $empower(threat\_org_1, user_V, victim\_user)$,
   $use(threat\_org_1, ac_V, to\_victim\_account)$.
   On the other hand we suppose the following statements are already true before the attack takes place and have been introduced in the policy by an administrator:
   – $empower(supervision, rdp\_host, rdp)$
   – $consider(supervision, tcp\_reset, send\_reset)$
   – $consider(supervision, tcp, all\_protocol)$
   – $consider(supervision, udp, all\_protocol)$
   – $consider(supervision, suspendacct, suspend\_account)$
   – $consider(supervision, send\_warning\_email, send\_warning)$
   – $consider(supervision, passwd, change\_password)$
4. Since the $brute\_force\_ctx$ context is true and given the aforementioned assignments of concrete entities to abstract entities, the following concrete security rules are derived:
   – $is\_obliged(rdp\_host, tcp\_reset, ip_A)$
   – $is\_prohibited(ip_A, tcp, ip_V)$
   – $is\_obliged(rdp\_host, suspendacct, ac_V)$
   – $is\_obliged(rdp\_host, send\_warning\_email, user_V)$

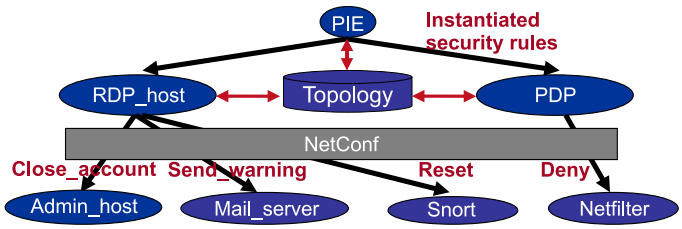

Fig. 2. The deployment architecture. The netconf [NWGN] protocol is used to configure some components of the information system. The PIE (Policy Instantiation Engine) is responsible for managing the global security policy. A PDP (Policy Decision Point) dispatches and translates concrete permissions and prohibitions. The RDP_host is responsible for the deployment of obligations

The *suspendacct* is the name of a script which can be used under POSIX compatible OSs to suspend a user account. Note that when the user receives the warning, the $received\_warning\_ctx$ is activated, thus the conjunction $brute\_force\_ctx$ & $received\_warning\_ctx$ becomes true and the following obligation is inferred:

- $is\_obliged(user_V, passwd, ac_V)$

  Here *passwd* is the command line tool used to change a user password.

5. The concrete inferred security rules are deployed (figure 2). Please refer to [CCBB+08] for an example of reaction policy deployment architecture.

Distributed denial of service We take the example of a DDOS attack implemented with the trinoo attack tool [Dit99]. An attacker uses the trinoo tool by connecting his/her computer to computers running the trinoo master program. Those master computers send commands to slave computers which generate attack traffic to specified target computers. The slave computers run the trinoo slave program. As in the previous example, we assume an alert correlation engine has detected the full DDOS scenario, hence an alert containing all the information about the master(s), slave(s) and target(s) of the attack has been generated.

Since the trinoo attack involves master and slave computers, we refine the generic abstract entity *attacker* by creating two sub-roles *master* and *slave*. We can still use the *attacker* role if for example the same rules must be applied to the master(s) and slave(s) machines (section 4, hierarchies). Figure 3 shows the ports and protocols used by the tool.

The number of machines composing the attack network may vary a lot. Hence formalizing the reaction policy as an abstract OrBAC policy is interesting as the number of abstract rules is independent from the attack network size. Moreover a trinoo network is composed of different types of attackers: the main attacker who controls the trinoo network, the master machine(s) and the slave machine(s). We can abstract all those types of attackers as different roles and manage them into the same dynamic threat organization. We define the following rules:

Fig. 3. Ports and protocols used by Trinoo

– master and slave machine(s):
$prohibition(supervision, attacker, all\_protocols, to\_any\_address, trinoo\_ddos\_ctx)$
$prohibition(supervision, any\_address, all\_protocols, to\_attacker, trinoo\_ddos\_ctx)$
All traffic is prohibited from and to the master(s) and slave(s). Enforcing the concrete rules derived from those abstract rules may be impossible if the computers are outside of the information system. In such a case it is still possible to filter the incoming traffic. Note that since we want to block the outgoing and incoming traffic of the master(s) and slave(s), we use the *attacker* role and the *to_attacker* view in the rule so that the rules are inherited through the hierarchies.
– master machine(s):
The master machine(s) can possibly be inside the information system monitored by the IDSs (a malicious employee can install them), so an alternative way of disabling them is to kill the process making them master(s):
$obligation(supervision, administrator, kill\_master\_process, to\_master, trinoo\_ddos\_ctx \ \& \ is\_ip)$
– slave machine(s):
The same applies for the slave(s):
$obligation(supervision, administrator, kill\_slave\_process, to\_slave, trinoo\_ddos\_ctx \ \& \ is\_ip)$
– victim machine(s):
$prohibition(supervision, slave, udp\_protocol, to\_victim, trinoo\_ddos\_ctx)$
UDP traffic between the slave(s) and victim(s) is prohibited.

The *is_ip* context is true when a given ip address is part of the information system network:

$hold(Org, \_, \_, O, is\_ip) : -ip\_belong\_to\_information\_system(O)$

Note that we use a context conjunction operator & in the two obligations so that those two rules apply only to computers inside the information system. The following role definitions are used to map the alert onto the *slave* and *master* sub-roles:

$empower(Threat\_org, Subject, master) : -$
$threat\_context\_management(Alert, Threat\_org),$
$mapping\_source\_master(Alert, Subject).$

$empower(Threat\_org, Subject, slave) : -$
$threat\_context\_management(Alert, Threat\_org),$
$mapping\_source\_slave(Alert, Subject).$

7 Implementation

Our approach is implemented as part of the supervision platform presented in section 2.

The MotOrBAC support tool [ACCBC08] is used to specify the reaction policy (as well as the entire information system policy). The user can write the abstract reaction policy and use the simulation window to test his/her policy. Actually the user can load IDMEF alerts in the simulation window and see the concrete security rules inferred by MotOrBAC. MotOrBAC is able to assist the user during steps 1 to 6 in the list presented in section 5.4. The 7^{th} step consists in translating the concrete security rules inferred from the abstract policy into a set of languages used by the components implementing the security mechanisms as illustrated in figure 2 and further explained in [CCBB+08].

Rule name	Organization	Role	Activity	View	Context
oblig3	supervision	rdp	send_warning	to_victim_user	brute_force
oblig4	supervision	victim_user	change_password	to_victim_account	composed_ctx
oblig1	supervision	rdp	send_reset	to_attacker	brute_force
oblig2	supervision	rdp	suspend_account	to_victim_account	brute_force

Type	Derives from	Subject	Action	Object
prohibition	prohib1	IP10_0_0_50	udp	IP10_0_0_60
prohibition	prohib1	IP10_0_0_50	tcp	IP10_0_0_60
obligation	oblig2	rdp_host	suspendacct	victim_user
obligation	oblig3	rdp_host	email_warning	victim_user
obligation	oblig1	rdp_host	tcp_reset	IP10_0_0_50
obligation	oblig4	victim_user	passwd	victim_user

Fig. 4. from top to bottom: the list of abstract obligations defined in motorbac for the brute force attack example and the concrete security rules derived from a sample IDMEF alert. The prohibitions are in red and the obligations in blue. The last obligation is not activated since the conjunction of contexts specified in the abstract rule from which it derives is false.

Figure 4 shows the abstract obligations defined with MotOrBAC to specify the reaction policy for the brute force attack example. One can see that this list of rules contains the rules presented in section 6.2 to react against a brute force attack. The same figure also shows an example of concrete security rules inferred when a new alert is received.

Abstract entity definitions for dynamic organizations are specified differently from other entity definitions because the organization in which the definition is given does not exist at the time of specification. Instead of selecting the organization in which the abstract entity definition is specified, the user chooses the type *Threat organization* and specify explicitly the mapping between the IDMEF alerts and the roles, activities and views using XPath expressions. The IDMEF alert generated by the correlation engine for the trinoo scenario can contain multiple instances of the *Source* IDMEF class. In section 6.2, the proposed reaction policy is different for the slave and master computers, hence their corresponding *Source* class instances in the IDMEF alert must be identified. The correlation engine segregates the master and slave computers in the alert by using the IDMEF *Process* class of the *Source* class. The *name* attribute of the *Process* class

is used to hold the names segregating the master computer(s) (*master* process) and slave computer(s) (*ns* process).

8 Conclusion

Managing the security of an information system involves defining a security policy and enforcing it. The supervision loop introduced in section 2 mentions the role of intrusion detection and the processing of its results, which involves reacting against attackers. The way the security components behave when an attack is detected can be formalized as a reaction policy. In this article, we showed that the OrBAC model is sufficiently expressive to model such policies. Although the RBAC model lacks expressiveness to specify such type of policy, we have shown that the OrBAC concept of dynamic organization is well suited to manage the activation of security rules upon the detection of an attack. Threat contexts can be combined with other types of contexts to express complex conditions. Since the reaction policy is expressed in the OrBAC model, it can be analyzed so that conflicts between rules can be detected and solved.

We have defined default intrusion roles, activities and views to manage most intrusions which involve one attacker and one victim. For more complex attacks with multiple attackers and victims, such as a DDOS, the default abstract entities can be refined. We have implemented the approach in the OrBAC API library, as a result the MotOrBAC support tool can be used to edit a reaction policy and help the policy administrators analyze and simulate it. We have integrated the OrBAC API in the implementation of a PIE to enable the automatic instantiation of reaction policies.

Some aspects have not yet been covered in this article, such as the lifetime of dynamic organizations, i.e. the lifetime of the concrete security rules deployed for each detected attack.

Acknowledgements: This work is partially funded by the Polux project (ANR 06-SETIN-012).

References

[ACCBC08] F. Autrel, F. Cuppens, N. Cuppens-Boulahia, and C. Coma. Motorbac 2: a security policy tool. In *Third Joint Conference on Security in Networks Architectures and Security of Information Systems (SARSSI)*, 2008.

[Bra98] Richard Brackney. Cyber-intrusion response. In *Proceedings of the 17th IEEE Symposium on Reliable Distributed Systems*, 1998.

[CAB+06] F. Cuppens, F. Autrel, Y. Bouzida, J. Garcia, S. Gombault, , and T. Sans. Anti-correlation as a criterion to select appropriate counter-measures in an intrusion detection framework, 2006.

[CBCdV+08] N. Cuppens-Boulahia, F. Cuppens, J. E. Lopez de Vergara, E. Vazquez, J. Guerra, and H. Debar. An ontology-based approach to react to network attacks. In *Third International Conference on Risk and Security of Internet and Systems (CRiSIS 2008)*, 2008.

[CCB08] F. Cuppens and N. Cuppens-Boulahia. Modeling contextual security policies. In *International Journal of Information Security (IJIS). Vol. 7, no. 4. August*, 2008.

[CCBB+08] F. Cuppens, N. Cuppens-Boulahia, Y. Bouzida, W. Kanoun, and A. Croissant. Expression and deployment of reaction policies. In *SITIS Workshop Web-Based Information Technologies and Distributed Systems (WITDS). Bali, Indonesia*, 2008.

[CCBG07] F. Cuppens, N. Cuppens-Boulahia, and M. Ben Ghorbel. High-level conflict management strategies in advanced access control models. In *Electronic Notes in Theoretical Computer Science (ENTCS)*, 2007.

[CCBM04] F. Cuppens, N. Cuppens-Boulahia, and A. Miège. Inheritance hierarchies in the Or-BAC model and application in a network environment. In *Second Foundations of Computer Security Workshop (FCS'04)*, 2004.

[CP00] Curtis A. Carver and Udo W. Pooch. An intrusion response taxonomy and its role in automatic intrusion response. In *IEEE Systems, Man, and Cybernetics Information Assurance and Security Workshop*, 2000.

[Dit99] D. Dittrich. The DoS project's trinoo distributed denial of service attack tool. http://staff.washington.edu/dittrich/misc/trinoo.analysis, 1999.

[DTCCB07] Hervé Debar, Yohann Thomas, Frédéric Cuppens, and Nora Cuppens-Boulahia. Enabling automated threat response through the use of a dynamic security policy. *Journal in Computer Virology 3(3)*, 2007.

[Fis96] Eric A. Fisch. A taxonomy and implementation of automated responses to intrusive behavior. In *PhD thesis, Texas A and M University*, 1996.

[FSG+01] David F.Ferrailo, Ravi Sandhu, Serban Gavrila, D.Richard Kuhn, and Ramaswamy Chandramouli. Proposed NIST standard for rbac. In *ACM Transactions on Information and System Security*, 2001.

[GF05] P. Gama and P. Ferreira. Obligation policies: An enforcement platform. In *Policies for Distributed Systems and Networks, IEEE International Workshop*, 2005.

[HCF07] H.Debar, D. Curry, and B. Feinstein. The intrusion detection message exchange format (idmef), 2007.

[KBB+03] A. Abou El Kalam, R. El Baida, P. Balbiani, S. Benferhat, F. Cuppens, Y. Deswarteand A. Miège, C. Saurel, and G. Trouessin. Organization based access control. In *IEEE 4th International Workshop on Policies for Distributed Systems and Networks (Policy 2003)*, 2003.

[MF03] G.H.M.B. Motta and S.S. Furuie. A contextual role-based access control authorization model for electronic patient record. In *IEEE Transactions on information technology in biomedecine vol.7, No3*, 2003.

[NWGN] http://tools.ietf.org/wg/netconf/trac/wiki NETCONF Working Group. Netconf.

[PCBC+07] 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 07)*, 2007.

[SBW07] Natalia Stakhanova, Samik Basu, and Johnny Wong. A taxonomy of intrusion response systems. In *International Journal of Information and Computer Security, 1(1/2):169-184*, 2007.

[Ull89] Jeffrey D. Ullman. Principles of database and knowledge-base systems. In *Computer Science Press*, 1989.