An ontology-based approach to react
to network attacks

Nora Cuppens-Boulahia¹, Frédéric Cuppens¹, Fabien Autrel¹, Hervé Debar²
¹ TELECOM Bretagne, Cesson Sévigné, France
² France Télécom Caen, France

Abstract

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 from the remainder of the security policy. Our approach is to consider that intrusion detection requirements are actually a part of the access control policy. This provides means to formally specify in a reaction policy what should happen in case of intrusions, intrusion attempts or violation of requirements. It is then possible to integrate these requirements into a deploying process in order to automatically configure security components and thus develop support tools to assist administrator in this task. In this paper, we propose a contextual and ontology-based approach to express and instantiate this reaction policy. We use ontologies to describe IDMEF (Intrusion Detection Message Exchange Format) alerts and OrBAC (Organization based Acess Control) policies. We then define a reaction process based on the concepts of dynamic threat organization and threat contexts and a set of rules used to map alerts onto threat contexts to perform the instantiation of the policy-based reaction in response to the detected intrusion.

1 Introduction

Intrusion Detection Systems (IDSs) are widely used to secure information systems, and became a primary component in modern security architecture solutions. Different intrusion detection techniques have been introduced and implemented in the governmental, academic and commercial information systems. Moreover, Intrusion Prevention Systems (IPSs) are highly used along with the IDSs to counter the detected threats. However, current intrusion prevention devices act only as conventional firewalls with the ability to block, terminate or redirect the traffic when the corresponding intrusion event is triggered. In other words, the intrusion response is statically associated with one (or several) intrusion event(s). By contrast, in [1], a reaction formalism based on the definition of a contextual security policy, was introduced. This reaction is performed globally allowing a global access control modification in an organization. However, the scalability remains an open issue that was not addressed in [1]. The threat context mechanism was implemented as a set of contextual rules that are triggered when the corresponding threat contexts become active. Only access control rules, i.e. permissions and prohibitions, were considered. We note that prohibitions and permissions are not appropriate to launch some actions, for instance shutting down a server, or redirecting undesirable traffic (e.g. syn-flooding packets).

In this paper, we aim at defining and designing solutions to enhance the detection/reaction process, improving the overall resilience of IP networks to attacks and help telecommunication and service providers to maintain sufficient quality of service and respect service level agreements. A main component of our architecture, called policy instantiation engine (PIE), is devoted to the instantiation of new security policies that counteract the network attacks. This component has to deal with the mapping of the alerts about attacks in the network into these security policies, providing the most suitable policy to reduce or even eliminate the problems caused in the network. We call this approach policy-based reaction.

This paper proposes an ontology-based approach using OWL-DL [2] and Jena [3] to express and instantiate the reaction policies. This technology provides a way to map alerts into attack contexts, which are used to identify the policies to be applied in the network to solve the threat. For this, ontologies to describe alerts and policies are defined, using inference rules to perform such mappings.

The use of ontologies for defining policies is not new. For instance, KAoS [4] and Rei [5] are well known policy frameworks based on ontologies. In this paper, we suggest an ontological approach based on the OrBAC security model [6]. A security policy in OrBAC corresponds to sets of contextual security rules corresponding to per-
missions, prohibitions and obligations. These security rules apply when their associated contexts are activated. As presented in [7], we suggest using OrBAC to define reaction security rules associated with the activation of special contexts called threat contexts. Notice also that, in OrBAC, one may consider several sets of security rules, each set of rules being associated to some organization. In this paper, we use this possibility to dynamically create organizations associated with the management of some intrusions. Such an organization, called threat organizations, is created to apply some reaction policy to circumvent the intrusion. We shall show how this concept of threat organization provides means to address scalability issues not solved in [7].

On the other side, attack ontologies have also been described before. Examples of them can be found in [8] and [9], providing a formal modeling of network attacks. However, although we also use ontologies, the approach proposed in this paper is focused on the mapping from attack alerts to threat contexts. In particular, these mappings are used to determine the reaction focus, i.e. on which components the reaction applies, and the reaction strategy, i.e. how to react on the selected components. One interesting property of ontologies is the ability to deal with several information models, allowing an easy integration of them. The work presented here exploits this capacity to integrate in a same view alert and policy instances, making the mapping process feasible.

The rest of this paper is structured as follows. Section 2 presents the issues related to the expression and deployment of a reaction policy and reviews related works. In section 3, we present a global overview of our supervision architecture and explain the different mapping steps to perform the instantiation of the policy-based reaction. In section 4, the ontology-based representation of OrBAC and IDMEF are provided, as well as the set of rules that allow the ontological mapping from IDMEF to OrBAC in order to activate the reaction policy. Section 6 presents three examples of policy-based reactions related to an elementary attack and two multi-step intrusions. Section 7 presents the implementation in the MotOrBAC 2 [10] support tool. We conclude in section 8.

2 Motivation

2.1 Problematic

In this article, the task of expressing a reaction policy takes place into the context of 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.

The security requirements 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 type of attack the intrusion detection systems may detect. For example the informal specification of a reaction policy 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 her she must change her password.

Some requirements in this specification 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 her account has been compromised and once she has been warned, she must change her password. In the following section, we show that the policy model which should be used to express the reaction policy example must satisfy several requirements.

2.2 Formalizing the reaction policy

Let us consider a set of Attacks $A$. Writing a reaction policy for an attack $A_i$ in $A$ consists in specifying a set of security rules $R_i$ which are activated when attack $A_i$ is detected.
The notion of role introduced in the RBAC model [11] is mandatory to specify a reaction policy 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 the reaction policy 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 [11], 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 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 [12] can express the link between an alert classification and a rule through the use of security context. Such a 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.

In this paper, 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.

2.3 Related works

A majority of research works have focused on the detection of intrusions but few works exist in the field of response to intrusions. Incidentally, very few articles deal with the expression of a reaction policy and its enforcement. Some taxonomies of intrusion responses have been defined ([13, 14, 15]). Fisch’s taxonomies [13] are system-oriented, they distinguish intrusion response systems by degree of automation and activity of triggered response. Carver and Posh’s [14] taxonomy deals with response, but from the attack side. Their taxonomy includes 6 dimensions and does not classify responses since it is attack-oriented. In [16], 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 [17, 1], the OrBAC model is used to express a reaction policy. A new type of OrBAC context, called threat context, is introduced to manage intrusion detection alerts expressed in IDMEF (Intrusion Detection Message Exchange Format) [18]. 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 article we propose to use the concept of threat contexts, but the definition we give is different from the one proposed in the aforementioned article. 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 further extend the OrBAC model by adding the concept of dynamic organizations.

3 Overview

3.1 Supervision architecture

We propose an auto-adaptive architecture that starts from the security policy management of the monitored information system. The low level tools including intrusion detection and access control mechanisms that are implemented locally to monitor the information system, are configured according to the high level security specifications. Then, whenever it is necessary, some of the generated alerts are forwarded to the upper level, by crossing different levels of reaction. At the upper level, and accordingly to the detected threat, an evaluation of the current system state takes place. Consequently, either direct responses will be launched or the whole security policy will be changed. We define three re-
action levels [17]: (1) low level reaction, (2) intermediate level reaction, and (3) high level reaction. Each level considers particular security requirements and deploys appropriate security components and mechanisms to react against the detected threats.

To provide the different detection and reaction functionalities, we propose a supervision architecture containing a set of components, depicted in figure 2:

- **ACE (Alert Correlation Engine):** this component is in charge of receiving alerts from network nodes, and enhances the detection of attacks by preforming alert correlation and other diagnosis combinations.

- **PIE (Policy Instantiation Engine):** this component receives the information about attacks from the ACE and instantiates new security policies to react to the attack in a high level reaction loop.

- **PDP (Policy Decision Point):** this component receives the new security policies defined by the PIE and deploy them in the enforcement points.

- **RDP (Reaction Decision Point):** this component receives the information about attacks from the ACE and decides a reaction in a mid level reaction loop.

- **PEP/REP (Policy Enforcement Point/Reaction Enforcement Point):** This component enforces the security policies provided by the PDP and reaction provided by the RDP. It may also perform an immediate low level reaction.

One may notice that three type of reactions have been defined, based on level of diagnosis required to apply them: low level reactions are directly decided by the PEP/REP, mid level reactions are decided by the RDP based on the information provided by the ACE and without instantiating new security policies, and finally, high level reactions are decided by the PIE, generating new security policies based on the ACE alerts that are passed to the PDP to deploy them in the PEPs.

This paper focuses on high level reactions, also called policy-based reactions in the following.

### 3.2 Alerts and policies representation

To propagate alerts from the PEP to the ACE, and from the ACE to the REP and PIE, messages are sent using the Intrusion Detection Message Exchange Format (IDMEF) [18] using the publish-subscribe protocol as suggested in [19]. The IDMEF format is based on XML and defined to report alerts about events in an Intrusion Detection System. It is composed of a set of tags that describe the different properties that an alert can contain, such as timestamps, source and target of the attack, and its classification. At this point, it is important to remark that the vulnerability exploited is commonly used to classify the attacks, using CVE (Common Vulnerabilities and Exposures) identifiers [20].

Several languages can be used to define the security policy, including Policy Core Information Model (PCIM) [21], Ponder [22], or Organization Based Access Control (OrBAC) [6]. After an analysis of existing policy languages, OrBAC was selected as the policy language to be used in our approach because it is expressive enough to specify a large variety of security requirements. In fact, it has been successfully applied to specify network access control policies and translation mechanisms have been defined to automatically generate firewall configuration rules that are free of inconsistency and redundancy [23].

In OrBAC, security policies are specified as sets of security requirements that correspond to permissions, prohibitions or obligations. Moreover, in OrBAC, these requirements may depend on contexts that express temporal, location or provisional conditions [24], which is very useful to specify new security requirements to be triggered in reaction to an intrusion. As suggested in [7], contexts are used in this paper to express also an attack condition, namely when an attack happens, a context about that attack, called threat context, is activated. Then, given a threat context and information provided in the IDMEF alert especially the source, target and classification, an OrBAC condition is held, activating new security rules.

However, one major difference in comparison with [7] is that we introduce the notion of threat organization and we suggest combining this notion with the one of threat context. When a given intrusion is detected, a new threat organization is created to deal with this intrusion. Then, we can define intrusion-dependent roles like attacker and victim and consider that subjects are assigned to these roles.
of attacker and victim, 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 victim of two different intrusions. Similarly, the corresponding threat contexts are activated locally to a given threat organization, so that the specified reaction policy also applies locally to this intrusion.

### 3.3 Policy instantiation engine process

Once the main concepts have been introduced, the goal of this section is to describe how the PIE maps IDMEF alerts into OrBAC contexts. For this purpose, the following steps have been defined:

- **Ontological representation of the IDMEF alert.** When the PIE receives a new IDMEF alerts, this alert is modeled using an ontological representation of the IDMEF alert in OWL-DL (AO, Alert Ontology). This is further explained in section 4.3.

- **Ontological representation of the reaction policy.** The reaction policy expressed in OrBAC is also modeled in OWL-DL (OO, OrBAC Ontology). This is further explained in section 4.2.

- **Threat organization creation.** When the PIE receives an IDMEF alert, it dynamically creates a new threat organization to react to the intrusion described in the alert.

- **Threat context activation.** Based on the alert content, especially the alert classification attribute, the PIE determines which threat contexts are activated by the alert. This is modeled by relationships between AO and OO with OWL-DL properties and Jena rules to link an attack to a context.

- **Determination of reaction focuses.** Based on the alert content, especially the source and target attributes, the PIE determines which subjects are the attackers and victims of the intrusion. This is modeled by relationships between AO and OO with OWL-DL properties and Jena rules that respectively map the source to one or several subjects assigned to the role **attacker** and the target to one or several subjects assigned to the role **victim**.

- **Choice of reaction strategy.** Based on the reaction policy specified in OrBAC, the PIE determines on which components to react and how to react on these components. The reaction policy decides of the granularity of reaction. For instance, even if a single host is a victim of the detected intrusion, the reaction policy may decide to react on every host involved in the same local network as the victim. The reaction policy also specifies that the reaction applies at different levels; at the network level, at the system level or at the application level.

- **Derivation of policy instances.** The PIE derives concrete security rules that apply to subject, action and object to be sent to the PDP for deployment. This is modeled by Jena rules.

Figure 3 shows the information involved in the policy instantiation if ontologies are used. An inference engine needs the OrBAC and the Alerts ontology, new alert instances as well as Jena rules to map those alerts into OrBAC hold instances, which are selected thanks to the reasoning process (left side of the figure). These OrBAC hold instances are then used to obtain the security policy rules to activate (right side of the figure). It should be noted that these operations are obtained directly using MotOrBAC and its plug-ins [10] (the OrBAC engine implemented by the SERES team of TELECOM Bretagne).

### 4 Representing information with ontologies

#### 4.1 Ontologies: OWL-DL and Jena

An ontology is defined in [25] as “a formal specification of a shared conceptualization”. In practical terms, an ontology is a hierarchy of concepts with attributes and relations that defines a terminology to define in consensus semantic networks of inter-related information units. An ontology provides a vocabulary of classes and relations to describe a domain, stressing knowledge sharing and knowledge representation. With the advent of the Semantic Web, OWL [26], the Web Ontology Language has gained relevance. It is a general purpose ontology language defined for the Semantic Web that contains all the necessary constructors to formally describe classes and properties, with hierarchies, and range and domain restrictions. An OWL ontology contains a sequence of axioms and facts. It includes several types of axioms, such as subclass axioms, equivalent Class axioms and property constraints.

When we study ontological languages, we noted that the inference and query languages used with OWL are SWRL [27] and SPARQL [28]. Although SWRL is often used, it does not really correspond to our requirements. Actually, SWRL imposes some syntactic restrictions on the language. For instance, it is not possible to use predicates with an arity higher than 2 and the symbols of function are rejected. These restrictions are not introduced to ensure computability properties of the language; the problem of satisfiability in SWRL remains undecidable. Moreover, most implementations of SWRL do not take into account all OWL’s con-
constraints. This gap makes that contradictory rules are possible and consistency must be assured by the security administrator. The consistency of the knowledge model is one of the properties on which our security policy rests. In addition, when a user requests an access, the security controller must imperatively decide if this access request is authorized or not. We should not thus allow indecision at this level. That is why SWRL has been discarded.

As our ontologies are defined using OWL-DL, the best candidate inference language is SPARQL but, for practical reasons and implementation effectiveness, we chose Jena. Jena [3] is a library of open Java classes source, developed by HP, which facilitates the development of applications for the semantic Web. It provides an environment of programming for RDF [29], RDFS [30], OWL [2], SPARQL [28] and includes an inference engine. Jena implements the RETE forward-chaining algorithm [31] and its inferences satisfy the transitivity, reflexivity, antisymmetry and hierarchy properties of OWL.

Thus, Jena makes it possible to manage ontologies and to make inferences and reasoning on these ontologies. It provides classes to represent ontology’s entities such as RDF graphs (Model class) and their triplets: subject (class Resource), predicate (Property class) and object (Literal class). Requesting these various classes are carried out using ARQ which is a SPARQL implementation [28] for Jena. Thus, Jena is well suited to create our reaction security policies and to derive the facts related to these policies.

So, in this work, OWL-DL is used as the ontology language to describe both IDMEF and OrBAC concepts, and Jena is used to map information from IDMEF to OrBAC threat contexts. For this purpose, a first step in the policy instantiation process is to define ontologies for both OrBAC and IDMEF alerts. Next sections provide such definitions.

### 4.2 OrBAC ontology representation

With respect to the OrBAC ontology, the following classes have been defined (see Figure 4):

- **Object, Subject and Action**: these classes respectively specifies the objects, subjects and actions contained in OrBAC.
- **View, Role and Activity**: these classes respectively model the views, roles and activities contained in OrBAC. A view has a relationship with an object, and a view can be derived from another view. Similar relationship exists for a role with a subject and for an activity for an action.
- **Organization**: this class is central in OrBAC and models the views contained in OrBAC.
- **Hold**: this class specifies when OrBAC contexts are active. A hold will have a subject, an object, an action, a context and an organization to be asserted (see Figure 5).
- **Context**: this class specifies the contexts contained in OrBAC. A context will have a name. Two auxiliary properties have been defined: the first one provides which alerts activate this context, based on the CVE classification of the alert. The second one is used to know if the context is active or not.
- **Rule**: this class models the rules contained in OrBAC. A rule will have an identification number, a type (permission, prohibition, obligation), and relationships with a context, a view, a role, an activity and an organization. An auxiliary property has been included to know if a rule is active or not (see Figure 6).
Figure 4. OrBAC ontology. Role-Subject, View-Object, Activity-Action

Figure 5. OrBAC ontology. Hold

Figure 6. OrBAC ontology. Security Rule
4.3 IDMEF ontology representation

With respect to the classes of the alerts ontology, the followed approach tries to reduce as much as possible the translation problems from real IDMEF alerts to the alert ontology. Thus, an alert ontology has been defined that maps the IDMEF alerts structure. This means that the alerts ontology is much more complex than the OrBAC ontology, with many more classes and relationships. For instance, the Alert class, as depicted in Figure 7, has relationships with Create_time, Analyzer_time, Detect_time Source, Target, AdditionalData, Assessment, Analyzer, and Classification. Then, Source and Target classes are related to a User, a Process, a Service and a Node, which has an Address (see Figure 8 and Figure 9).

5 Reaction policy specification

As we said in section 2.2, 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 these problems.

Actually in OrBAC, an object 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 organizations associated with the detection of attack occurrences are dynamically and automatically created instead of being handled manually by an administrator, we call them dynamic organizations. More specifically in the context of intrusion detection, we call them threat 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.

5.1 Threat organization and threat context

The rules in \( R \) making up the reaction policy are defined in an organization called supervision in the following. 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 \( threat\_org_{ij} \):

\[
\text{threat\_context\_management?(alert}_{ij},?\text{threat\_org}_{ij})
\]

To achieve the ontological mapping from IDMEF schema onto OrBAC entities, we introduce some abstract entities managed by the supervision organization.

- 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 [32] for other types of contexts) called threat context. A threat context is activated in \( threat\_org \) to manage a given alert if its definition matches the classification of the alert (for this purpose we use the predicate mapping\_classification?(alert,?\text{threat\_context}). This context is active for every triple \( \{\text{subject, action, object}\} \) and defined as follows:

\[
\text{hold?(threat\_org,?\text{threat\_context}(\_\_\_))} \leftarrow \\
\text{threat\_context\_management?(threat\_org,?\text{alert})} \\
\wedge \text{mapping\_classification?(alert,?\text{threat\_context})}.
\]

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 \( alert \) can be simply deleted when the threat is no more active by deleting organization \( threat\_org \).

5.2 Mapping alert to abstract entities

The mapping from an alert to the threat abstract entities in \( threat\_org \) is done using role, activity and view definitions. Here are two examples of generic mapping between the source and target of an alert and the roles attacker and victim:
Figure 7. IDMEF ontology (Alert relationships)

Figure 8. IDMEF ontology (Source relationships)

Figure 9. IDMEF ontology (Target relationships)
By contrast to [1], 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\textsubscript{1}, for example the communication between two servers located in two different networks, which are normally independent, to backup critical data:

$$\text{securityRule(}\text{supervision, permission(data_server, open_ssh_connection, to_backup_server, attack}_{1}\text{ctx)})}$$

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 a reaction policy. Consider the case of an attacker trying to login into 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:

$$\text{securityRule(}\text{supervision, prohibition(}\text{attacker, all_traffic, victim, telnet_attack_ctx)\text{)}\text{)}\text{)}$$

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:

$$\text{securityRule(}\text{supervision, obligation(web_server_daemon, stop, web_server, new_threat)\text{)}\text{)}\text{)}$$

Note that we consider that this obligation is an immediate obligation, i.e the obligation 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 [33, 17]. 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\textsubscript{threat} associated with the new alert. org\textsubscript{threat} is created as a suborganization of the supervision organization.

2. activation of threat contexts in org\textsubscript{threat}. Yet no concrete rules are infered since no concrete entities are assigned to abstract entities in org\textsubscript{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\textsubscript{threat}. This step and the previous one are specified through the abstract entity definitions.

5. abstract security rules associated with the threat context into org\textsubscript{threat} are activated.

6. since concrete entities have been created and assigned to abstract entities in org\textsubscript{threat}, some concrete security rules are infered from the activated abstract rules.

7. concrete security rules are deployed 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 [17].

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 [34]). This correlation
engine is part of the supervision loop presented in section 2. The reaction policy instantiation process is detailed in section 6.2.1.

6.1 Reacting against a buffer overflow

Since a buffer overflow (BOF) can be used by an attacker to execute arbitrary code on a machine and hence take control of it, the typical counter-measure against such an attack is to isolate the machine from the network to correct the exploited vulnerability. We only give here the reaction policy specification, the detailed instantiation process of a reaction policy is given for the brute force attack example in section 6.2.1.

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 machine 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:

\[ \text{securityRule}(\text{supervision, prohibition(victim, all_protocol, to_any_address, webdav_bof_ctx)}) \]

\[ \text{securityRule}(\text{supervision, prohibition(any_address, all_protocol, to_victim, webdav_bof_ctx)}) \]

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

\[ \text{securityRule}(\text{supervision, obligation(admin, start, backup_web_server, webdav_bof_ctx)}) \]

Since the attacked web server must be isolated, the web service must be implemented by another machine. Here the admin role is used in the rules but one might have created a web_admin role to manage the sub-part of the information system security policy related to the web server.

- the vulnerable web server must be patched to remove the vulnerability:

\[ \text{securityRule}(\text{supervision, obligation(admin, update, to_victim, webdav_bof_ctx)}) \]

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

6.2 Reacting against multi-step attacks

6.2.1 Brute force attack

An example of attack requiring an alert correlation engine is a brute force attack. As said at the begining 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 all_protocol activity is used and has the same meaning that in the previous example. The send_reset activity abstracts the actions that disconnect the attacker from the victim. The suspend_account activity abstracts the actions taken by the administrator or a process to suspend a user account (the account can be a linux/unix account, a microsoft windows account, 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):

\[ \text{hasProperty(?thread_org, empower(?subject, victim_user))} \]
\[ \text{← thread_context_management(?alert, ?thread_org),} \]
\[ \text{mapping_target_user_name(?alert, ?subject)} \]

\[ \text{hasProperty(?thread_org, use(?object, to_victim_account))} \]
\[ \text{← thread_context_management(?alert, ?thread_org),} \]
\[ \text{mapping_target_user(?alert, ?object)} \]

The following rules correspond to the reaction policy 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:

\[ \text{securityRule}(\text{supervision, prohibition(attacker, all_protocol, to_victim, brute_force_ctx)}) \]

- the connection between the attacker and victim must be interrupted:

\[ \text{securityRule}(\text{supervision, obligation(rdp, send_reset, to_attacker, brute_force_ctx)}) \]
• the victim account must be suspended:
  \[
  \text{securityRule}(\text{supervision}, \text{obligation}(\text{rdp}, \text{suspend_account}, \text{to_victim_account}, \text{brute_force_ctx}))
  \]

• the victim must be warned that his/her account has been attacked:
  \[
  \text{securityRule}(\text{supervision}, \text{obligation}(\text{rdp}, \text{send_warning}, \text{to_victim_user}, \text{brute_force_ctx}))
  \]

• once he/she has been warned that his/her account has been attacked, the victim must change his/her password:
  \[
  \text{securityRule}(\text{supervision}, \text{obligation}(\text{victim_user}, \text{change_password}, \text{to_victim_account}, \text{brute_force_ctx} & \text{received_warning_ctx}))
  \]

The \text{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 \text{user}_V\). Suppose that an Intrusion Detection System (SDI) 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 \text{threat_org}\(_1\) is created as a sub-organization of organization \text{supervision}.

2. The threat context associated with the alert classification of \(m\), namely the \text{brute_force_ctx} context, is activated.

3. The relevant role and view definitions are evaluated so that the following statements become true:
  \[
  \text{hasProperty}(\text{threat_org}_1, \text{empower}(ip_A, \text{attacker}))
  \]
  \[
  \text{hasProperty}(\text{threat_org}_1, \text{empower}(ip_V, \text{victim}))
  \]
  \[
  \text{hasProperty}(\text{threat_org}_1, \text{empower}(\text{user}_V, \text{victim_user}))
  \]
  \[
  \text{hasProperty}(\text{threat_org}_1, \text{use}(ac_V, \text{to_victim_account}))
  \]

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:

\[
\text{hasProperty}(\text{company_org}, \text{empower}(\text{rdp_host}, \text{rdp}))
\]
\[
\text{hasProperty}(\text{company_org}, \text{consider}(\text{tcp_reset}, \text{send_reset}))
\]

\[
\text{hasProperty}(\text{company_org}, \text{consider}(\text{tcp, all_protocol}))
\]
\[
\text{hasProperty}(\text{company_org}, \text{consider}(\text{udp, all_protocol}))
\]
\[
\text{hasProperty}(\text{company_org}, \text{consider}(\text{suspendacct, suspend_account}))
\]
\[
\text{hasProperty}(\text{company_org}, \text{consider}(\text{send_warning_email, send_warning}))
\]
\[
\text{hasProperty}(\text{company_org}, \text{consider}(\text{passwd, change_password}))
\]

Here \text{company_org} is a super-organization of the \text{supervision} organization.

4. Since the \text{brute_force_ctx} context is true and given the aforementioned assignments of concrete entities to abstract entities, the following concrete security rules are derived:

- \(\text{is_obliged}(\text{rdp_host}, \text{tcp_reset}, \text{ip}_A)\)
- \(\text{is_prohibited}(\text{ip}_A, \text{tcp}, \text{ip}_V)\)
- \(\text{is_obliged}(\text{rdp_host}, \text{suspendacct}, \text{ac}_V)\)
- \(\text{is_obliged}(\text{rdp_host}, \text{send_warning_email}, \text{user}_V)\)

The \text{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 \text{received_warning_ctx} is activated, thus the conjunction \text{brute_force_ctx} & \text{received_warning_ctx} becomes true and the following obligation is inferred:

- \(\text{is_obliged}(\text{user}_V, \text{passwd}, \text{ac}_V)\)

Here \text{passwd} is the command line tool used to change a user password.

5. The concrete inferred security rules are deployed (figure 10). Please refer to [17] for an example of reaction policy deployment architecture.

**6.2.2 Distributed denial of service**

We take the example of a DDOS attack implemented with the trinoo attack tool [36]. 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
Figure 10. The deployment architecture. The netconf [35] protocol is used to configure some components of the information system. The PIE (Policy Instanciation 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.

Figure 11. Ports and protocols used by Trinoo

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 are able to abstract all those types of attacker as different roles and manage them into the same dynamic threat organization when an alert is received. We define the following rules:

- master and slave machine(s):
  \[
  \text{securityRule}(\text{supervision}, \text{prohibition}(\text{attacker}, \text{all protocol}, \text{to any address}, \text{trinoo ddos ctx}))
  \]
  \[
  \text{securityRule}(\text{supervision}, \text{prohibition}(\text{any address}, \text{all protocol}, \text{to attacker}, \text{trinoo ddos ctx}))
  \]
  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):
  \[
  \text{securityRule}(\text{supervision}, \text{obligation}(\text{admin}, \text{kill master process}, \text{to master}, \text{trinoo ddos ctx} & \text{is ip}))
  \]

- slave machine(s):
  The same applies for the slave(s):
  \[
  \text{securityRule}(\text{supervision}, \text{obligation}(\text{admin}, \text{kill slave process}, \text{to slave}, \text{trinoo ddos ctx} & \text{is ip}))
  \]

- victim machine(s):
  \[
  \text{securityRule}(\text{supervision}, \text{prohibition}(\text{supervision}, \text{slave, udp protocol, to victim, trinoo ddos ctx}))
  \]
  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:

\[
\text{hold}(\text{?org, \ldots ?o, is_ip}) \leftarrow \text{ip belong to information system(?o)}
\]

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:

```plaintext

```
hasProperty(?threat_org, empower(?subject, master)) ← threat_context_management(?alert, ?threat_org) ∧ mapping_source_master(?alert, ?subject).

hasProperty(?threat_org, empower(?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 [10] is used to specify the reaction policy (as well as the entire information system policy)\(^1\). 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 from MotOrBAC. MotOrBAC is able to assist the user during steps 1 to 6 in the list presented in section 5.2. 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 10 and further explained in [17].

![Figure 12. Abstract Obligations defined with MotOrBAC, from top to bottom: the list of abstract obligations defined in motorabc for the brute force attack example, the list of entity definitions, the list of contexts 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.](image)

The IDMEF alert generated by the correlation engine for the trinoo scenario can contain multiple instances of the source IDMEF class. In section 6.2.2 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

This paper has presented an approach to react to network attacks in which a set of alerts are used to generate new security policies, called reaction policies. Although the RBAC model lacks expressiveness to specify such type of reaction policy, we suggest an approach based on the OrBAC model and define the concept of dynamic threat organization 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 [37].

In this paper, an ontology approach has been used, in which the alert and the policy information models can be combined to help in the instantiation of the policies. The use of OWL-DL and JENA provides some advantages: there are existing generic tools for parsing and reasoning. The nature of OWL as a Semantic Web component allows merging distributed ontologies. Moreover, there are many works on mapping different knowledge bases that can be leveraged for our purpose.

In this framework, 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, for example by

---

\(^1\)MotOrBAC is available on http://motorbac.sourceforge.net
considering master and slave roles.

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.

Acknowledgement: This paper has been partially funded by the European CELTIC Project ReD (CP3-011).

References


