A FORMAL APPROACH TO SPECIFY AND DEPLOY A NETWORK SECURITY POLICY

F. Cuppens[1], N. Cuppens-Boulahia[1], T. Sans[1], A. Miège[1,2]
[1]*GET/ENST Bretagne, 2 rue de la Châtaigneraie, 35512 Cesson Sévigné Cedex, France*
[2]*GET/ENST, 46 rue Barrault, 75634 Paris Cedex 13, France*

Abstract

Current firewall configuration languages have no well founded semantics. Each firewall implements its own algorithm that parses specific proprietary languages. The main consequence is that network access control policies are difficult to manage and most firewalls are actually wrongly configured. In this paper, we present an access control language based on XML syntax whose semantics is interpreted in the access control model Or-BAC (Organization Based Access Control). We show how to use this language to specify high-level network access control policies and then to automatically derive concrete access control rules to configure specific firewalls through a translation process. Our approach provides clear semantics to network security policy specification, makes management of such policy easier for the administrator and guarantees portability between firewalls.

1. Introduction

It is well known in the computer security community that specifying and managing access control rules is a hard task whatever the level of abstraction considered. These access control rules are actually part of a more global set of rules called an organizational policy. We argue that this organizational policy has to be unfolded to obtain packages of access control rules. Each rule package is handled by a security component. For instance, environmental security package, physical security package, operating system security package, staff package and network security package. Firewalls are those components that deal with network security packages. They are used to block to some extent any suspicious communication from Internet to the private local area network (LAN) and to deny the members of the private LAN access the all harmful Internet temptations. One of the problems encountered with firewalls is the difficulty the administrators have to well configure them. There is really a lack of methodology and corresponding supporting tools to help them in setting the network security policy part, and generating and deploying the rules derived

from this policy. There is actually no intermediary levels between the policy requirement formulated as an English sentence and its equivalent set of firewall rules, say the code.

Even if the firewall administrator is proficient in many configuration languages and tools, this expertise does not avoid from making mistakes. Without a clear methodology and some corresponding supporting tools, this may lead to the generation of configuration rules that are not consistent with the intended network security policy. We claim that the use of a high level language to specify a network security policy will avoid such mistakes and will help to consistently modify the firewall rules when necessary. Moreover, this high level language must allow administrators to specify security requirements and have to be expressive enough to specify any network security policy.

We also notice that there is not a global security policy specification so that an underlying hypothesis is always done: a single security component is used, say a single firewall. Now, it is sometimes more convenient to deploy security rules on several security components. In particular, access security rules can be separated into relevant packages and enforced by more than one firewall on the same LAN.

Furthermore, in most of firewalls, administrators use dual security policy. That is they specify both permission and prohibition rules. In this case, the selection by the firewall of the appropriate rule is based on a first matching or a last matching procedure. In both cases, the decision depends on how the security rules are sorted. Hence, administrators have to find out the correct and efficient order of the rules, order that is dependent on the filtering procedure. This is a complex task to manage especially when the security policy has to be updated. Moreover, in some cases, it is even not always possible to sort the rules. So, a closed access control policy that only includes permissions may be an alternative.

In this paper, we present an access control language based on XML syntax. This language is supported by the access control model Or-BAC [Kalam et al., 2003] to specify access control meta-rules. The concepts introduced by Or-BAC are used all top-down specification long to properly generate firewall configuration rules. There are other attempts to suggest such a top-down approach. For instance, [Hassan and Hudec, 2003] applies the RBAC model [Sandhu et al., 1996] but fails to fit the model semantics to low level implementation. The reason is largely due to the fact that RBAC is less expressive than Or-BAC and hence network level security rules are not naturally derived (see section 5).

To handle a network security policy, some topology of the organization's local area network must be enforced. Hence, the LAN is parcelled out into *zones*. The access control consists in securely managing communications between these *zones*. We show in this paper that view and role definitions of Or-

BAC allow a fine grained specification of zones. The hierarchy frameworks of extended Or-BAC [Cuppens et al., 2004a] avoid the use of artifices like *open* and *closed* groups – a specialization of zones – as suggested in [Bartal et al., 1999] (see section 5). In this connection, we also investigate if it is possible to specify a network security policy by making use of permissions only. The great contribution of this closed policy is to avoid having to sort firewall rules that are derived to enforce this policy. Sorting the rules is actually complex to manage and is a major source of errors. It is one of the main drawbacks of many firewall configuration languages.

There are some tools that help administrators to build their security policy and to translate it to the actual configuration language (for instance Cisco PIX [Degu and Bastien, 2003], Ipfilter [Russell, 2002], ...) but these tools, for example firewall builder [Kurland, 2003], are bottom-up approaches. That is, they deal with the particular problem of producing the code in the configuration language of the target firewall. The reasoning process on the access control policy is not considered. Hence, there is a lack of accurate semantics that allows the security administrator to avoid firewall mis-configuration (see section 5). We also investigate the automatic generation of the target firewall rules *from the formal specification of a network security policy.*

The remainder of this paper is organized as follows. Section 2 presents the main concepts of Or-BAC using an XML syntax [W3C, 2004] in expectation of its translation into a given target platform. We explain in section 3 how to specify a network security policy in Or-BAC and its counterpart in XML. Section 4 presents the compilation process of the abstract policy into concrete firewall rules through a real application. A comparison with other similar works is done in section 5 and finally section 6 concludes this paper.

2. Modelling Or-BAC in XML

2.1 Basic model

The Or-BAC model was first presented in [Kalam et al., 2003] using first order logic formalization. In this paper, we present an interpretation of Or-BAC in XML. The complete XML schema corresponding to the basic Or-BAC model is available in [Cuppens et al., 2004b].

The Or-BAC model enables `organizations` to define their access control `policy`. An organization corresponds to any entity in charge of managing a set of `security rules`. In the basic Or-BAC model, security rules are restricted to `permissions`, but they may be extended to also include `prohibitions` and `obligations` (see section 2.2 below). For instance, a given hospital is an organization. A concrete security component, such as a firewall, may be also viewed as an organization since it manages a set of security rules. In the organization, `subjects` will request to perform `actions` on `objects` and

the final objective of an access control policy is to decide if these requests are permitted or not. However, permissions in Or-BAC model do not directly apply to subject, action and object. Instead, subject, action and object are respectively abstracted into `role`, `activity` and `view`.

Thus, subjects obtain permission based on the role they play in the organization. We use a similar approach for actions and objects. An action is permitted based on the role this action plays in the organization. In Or-BAC, an action role is called an *activity*. For instance, a given organization may specify that *consult* is an activity and that a possible role of action *acroread* is to *consult* medical record. Similarly, permissions to have an access to an object are based on the role this object plays in the organization. In Or-BAC, an object role is called a *view*. For instance, a given organization may specify that *medical record* is a view and that a possible role of object *fich27.pdf* is to be used as a *medical record*.

Each organization respectively specifies the roles, activities and views that are `relevant` in this organization.

For each relevant role, the organization specifies the subjects that are assigned to this role using the XML element `empower`. Similarly, for each relevant activity, actions are assigned to this activity using the XML element `consider` and, for each relevant view, objects are assigned to this view using the XML element `use`.

The XML schema is interpreted by the set of predicates suggested in [Cuppens et al., 2004a] to define a logical model for Or-BAC: (1) Predicate $relevant\_role(org, r)$ where org is an organization and r a role to define roles that are relevant in a given organization, (2) Predicate $relevant\_activity(org, a)$ where org is an organization and a an activity to define activities that are relevant in a given organization, (3) Predicate $relevant\_view(org, v)$ where org is an organization and v a view to define views that are relevant in a given organization, (4) Predicate $empower(org, s, r)$ where org is an organization, s a subject and r a role to define subjects that are empowered in a given role in a given organization, (5) Predicate $consider(org, \alpha, a)$ where org is an organization, α an action and a an activity to define actions that implement a given activity in a given organization, (6) Predicate $use(org, o, v)$ where org is an organization, o an object and v a view to define objects that are used in a given view in a given organization, (7) Predicate $permission(org, r, a, v)$ where org is an organization, r a role, a an activity and v a view to define that in a given organization, some roles are permitted to perform some activities over some views.

The Or-BAC model also provides means to automatically derive concrete permissions between subjects, actions and objects. For this purpose, the following predicate (not represented in the XML schema) is used:

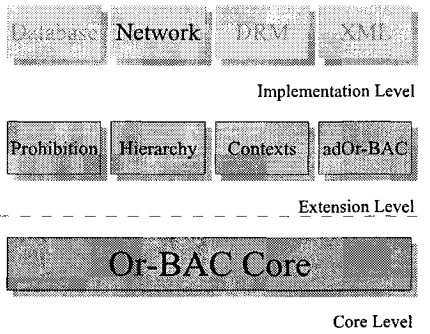

Figure 1. Or-BAC architecture

$Is\_permitted(s, \alpha, o)$ where s is a subject, α an action and o an object to define that some subjects are permitted to perform some actions on some objects.

In Or-BAC, triples that are instances of the predicate $Is\_permitted$ are derived from permissions granted to roles, views and activities by the predicate $permission$ using a logical general rule to specify that: if an organization org grants role r permission to perform activity a on view v, and if org empowers subject s in role r, and if org uses object o in view v, and if org considers that action α implements activity a then s is permitted to perform α on o.

2.2 Or-BAC extensions

There are several possible extensions to the basic model presented in the previous section (called Or-BAC core in figure 1). In particular, security rules may include prohibitions and obligations in addition to permissions. Considering both permissions, prohibitions and obligations may lead to conflicts. Managing conflicts in Or-BAC is discussed in [Cuppens and Miège, 2003a]. It is also possible to consider contextual security rules. This problem is further addressed in [Cuppens and Miège, 2003b] where we show how to manage various types of context, such as temporal, spatial, prerequesite, user-declared and provisional contexts (see [Cuppens and Miège, 2003a] for further details). Another possible extension consists in activating AdOr-BAC, the administration model of Or-BAC [Cuppens and Miège, 2004].

However, in the following, we shall suggest defining a network security policy using only non contextual permissions so that the context, prohibition and obligation extensions are not activated. The only extension we shall actually consider is the hierarchy extension.

In Or-BAC, it is possible to consider role, activity, view and organization hierarchies (see [Cuppens et al., 2004a] for a more complete presentation and formalization in first order logic). All these hierarchies are partial order relations,

i.e. reflexive and transitive relations. In the XML schema, role, activity and view hierarchies are respectively specified using the `subRole`, `subActivity` and `subView` elements. These XML elements are interpreted in a logical formalism by the following three predicates: (1) $sub\_role(org, r_1, r_2)$: in organization org, role r_1 is a sub-role of role r_2, (2) $sub\_activity(org, a_1, a_2)$: in org, activity a_1 is a sub-activity of activity a_2, (3) $sub\_view(org, v_1, v_2)$: in org, view v_1 is a sub-view of view v_2.

The hierarchy on roles has the special meaning that, in organization org, role r_1 inherits from r_2 all the permissions associated with r_2. The hierarchy on activities and views are associated with similar inheritance mechanisms.

In the hierarchy extension, we also consider hierarchies on organization that may be specified using the `subOrganization` element in the XML schema. This is interpreted by the following logical predicate: $sub\_organization(org_1, org_2)$ meaning that organization org_1 is a sub-organization of organization org_2.

For those roles of org_2 that are relevant in org_1, we consider that the role, activity and view hierarchies defined in org_2 also applies in org_1. We also accept a similar principle for inheritance of permissions through the organization hierarchy provided that the role, activity and view in the scope of the permission are relevant in the sub-organization.

3. Using Or-BAC to specify a network security policy

3.1 Principles

In this section, we show how to use Or-BAC in the context of network security. Our final objective will be to derive security rules to configure specific firewalls (see section 4).

A firewall may be viewed as a security component that filters IP packets with respect to a given security policy. How can we interpret the concepts of subject, action and object in this case? Our proposal is the following. A subject is any host machine. A host machine is modelled by two elements: IP address and network mask. An action is any implementation of a network service such as http, snmp or ping. In our model, a service has three elements: a protocol, a source port and a destination port. Finally, an object is a message sent to another host machine. A message is represented by two elements: a content and a receiver (a host machine).

Thus, triples ⟨subject, action, object⟩ are interpreted as host machines that use services to send messages to other host machines. Notice that messages have content so that we can define security rules to make filtering decision based on the IP packet content. However, this possibility is not used in the remainder of this section but is further discussed in the conclusion.

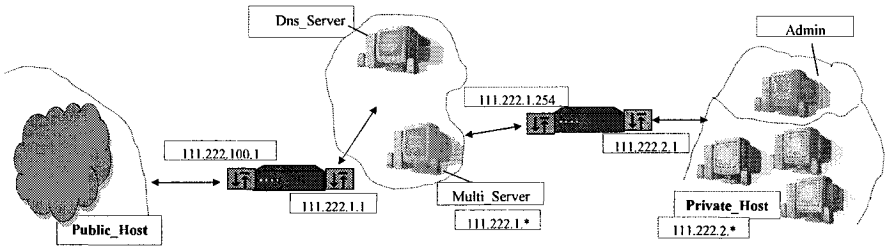

Figure 2. Application example

We shall now give interpretation to the Or-BAC notions of organization, role, activity and view. To illustrate our approach, we reuse the example used in Firmato [Bartal et al., 1999]. The objective is to model the access control policy of a corporate network used in an organization H. H has a two-firewall network configuration, as shown in figure 2. As presented in [Bartal et al., 1999], the external firewall guards the corporation's Internet connection. Behind is the DMZ, which contains the corporation's externally visible servers. In our case these servers provide HTTP/HTTPS (web), FTP, SMTP (e-mail), and DNS services. The corporation actually only uses two hosts to provide these services, one for dns and the other (called Multi_server) for all the other services. Behind the DMZ is the internal firewall which guards the corporation's intranet. This firewall actually has two interfaces: one for the DMZ and another one for the private network zone. Within the private network zone, there is one distinguished host, *Admin*, which provides the administration for the servers in the DMZ and firewalls.

3.2 Organization

In Or-BAC, a security policy will be generally modelled using several organizations. Of course, the organization H which is defining its access control policy is an organization for Or-BAC. The network security policy of H will be managed by a sub-organization of H. Let us call H_LAN this sub-organization. A first objective in this section is to show how to use Or-BAC to define the network security policy of H_LAN.

Since the network security policy is actually managed by two firewalls, we shall consider that H_LAN has two sub-organizations denoted H_fwi and H_fwe that respectively correspond to the internal and external firewalls. Another objective of our approach is to show how to derive specifications of policies managed by H_fwi and H_fwe from the one defined for H_LAN using inheritance rules presented in section 2.2.

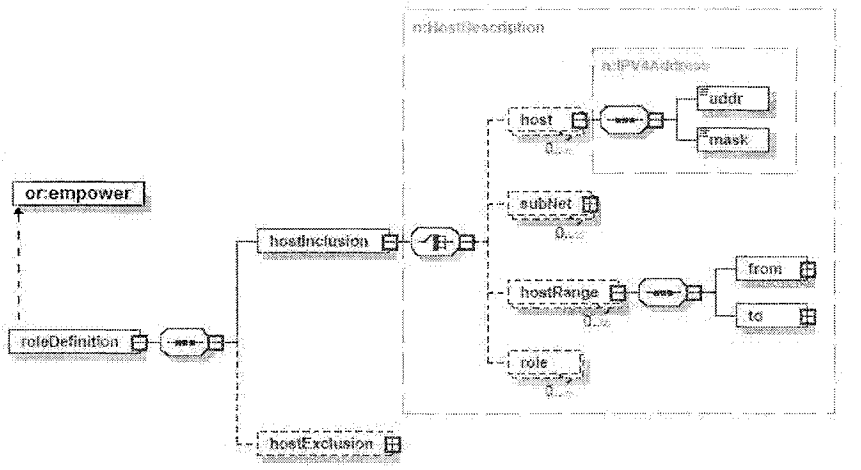

Figure 3. Role Definition

3.3 Role

As mentioned in section 3.1, subjects correspond to host machines. So, we consider roles that may be assigned to hosts. Examples of such roles may be dns_server or $firewall$. Using the Or-BAC core model, the only way to assign roles to hosts is by using the `empower` element. This means that the security administrator must enumerate every host assigned to each role. This would never be practical nor efficient since security configuration rules of firewalls would be derived for each host.

This is why we suggest using `roleDefinition` to define assignment conditions of hosts to roles (see figure 3). Role definition has two parts: `hostInclusion` that corresponds to the positive condition a given host must satisfy to be assigned to the role and `hostExclusion` that corresponds to the negative condition the host must not satisfy. Each of these two conditions has four sub-parts: `host` to enumerate hosts, `subnet` to specify network zones identified by their mask, `hostRange` to specify intervals of IP addresses and `role` to refer to other roles.

Role definition of a role R in a given organization org can be interpreted by a logical rule as follows:

- $\forall s, (host(s) \wedge host\_inclusion(s) \wedge \neg host\_exclusion(s)) \rightarrow empower(org, s, R)$

where $host\_inclusion(s)$ and $host\_exclusion(s)$ respectively represents the disjunction of conditions specified by elements `host`, `subnet`, `hostRange` or `role`. If the `hostInclusion` element is actually empty, then we assume that

$host\_inclusion(s)$ is equivalent to true (meaning that any host is included); If the `hostExclusion` element is empty, then $host\_exclusion(s)$ is equivalent to false (meaning that no host is excluded).

For instance, the following XML structure [W3C, 2004] represents the `Private` role definition:

```
<relevantRole name="Private">
   <n:roleDefinition>
      <n:hostInclusion>
         <n:subNet>
           <n:addr>111.222.2.0</n:addr>
           <n:mask>24</n:mask>
         </n:subNet>
      </n:hostInclusion>
      <n:hostExclusion>
         <n:role roleName="FW_intern"/>
         <n:role roleName="Admin"/>
      </n:hostExclusion>
  </n:roleDefinition>
</relevantRole>
```

This structure says that a given host is empowered in the `Private` role if it belongs to the subnet 111.222.2.0/24 (host inclusion condition) and if it is not empowered in role `FW_intern` or in role `Admin` (host exclusion condition).

3.4 Activities

Activities are abstractions of network services. Our approach is similar to the one suggested for role definition. Instead of enumerating actions assigned to activities, we suggest using `activityDefinition` (see figure 4).

To specify an activity definition corresponding to network services, one has first to choose a protocol `tcp`, `udp` or `icmp`. If the protocol is TCP, two elements can be used to specify the activity: `scrPort` and `destPort` to respectively express conditions on source and destination port numbers. Both elements have similar structure. It is then possible to use the `portRange` element to specify an interval of port numbers or `singlePort` to enumerate a set of port numbers.

The structure is similar to define activities corresponding to UDP protocols. For ICMP protocol, the structure is different. It must be defined using two elements: `type` and `code`.

For instance, the following XML structure represents the activity `Web_HTTP`:

```
<relevantActivity name="Web_HTTP">
  <n:activityDefinition>
    <n:tcp>
      <n:destPort>
        <n:singlePort>80</n:singlePort>
      </n:destPort>
```

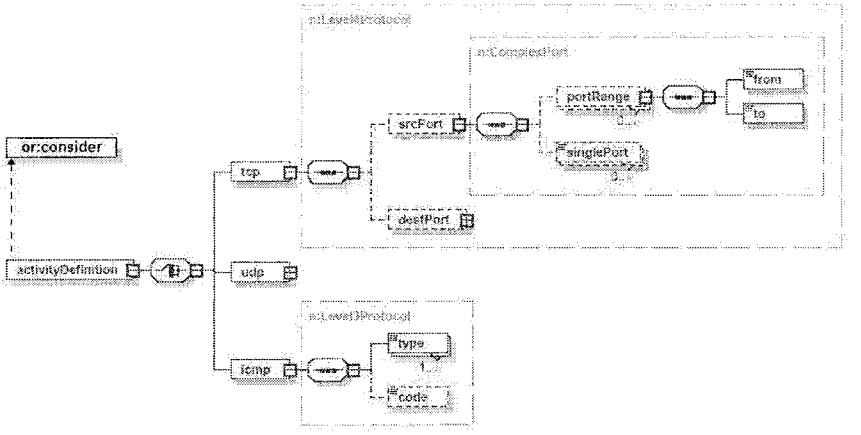

Figure 4. Activity Definition

```
    </n:tcp>
  </n:activityDefinition>
</relevantActivity>
```

This structure says that a given action corresponds to activity `Web_HTTP` if protocol is equal to TCP and destination port number is equal to 80. Notice that the `srcPort` element is not used. In this case, we assume that any source port number are acceptable.

3.5 Views

Views are abstraction of messages sent to destination hosts. A view definition can be defined by using the element `toTarget` that must be assigned to a role name. A view definition of a given view V defined by organization org and having `toTarget` element equal to role R may be interpreted by the following logical rule:

- $\forall m, (message(m) \wedge dest(m, h) \wedge empower(org, h, R)) \rightarrow use(org, m, V)$

This rule says that a message m is used in a given view V if the destination host h of this message is empowered in the role R specified in the view definition.

For instance, the following XML structure defines a view corresponding to the messages whose destination hosts are empowered in the `Internet` role:

```
<relevantView name="To_Internet">
  <n:viewDefinition>
    <n:toTarget roleName="Internet"/>
  </n:viewDefinition>
```

```
</relevantView>
```

In the following, we shall call such a view "target role" for short. Notice that, since we do not consider filtering rules based on the message content in the following, the view definition does not include possibility to specify condition on the message content attribute. However, we plan to provide this extension in the near future.

3.6 Permission

To specify the access control policy using our approach, we have simply to express permissions between role, activity and view. For instance, the following XML structure specifies that role `Private` is permitted to perform activity `Web_HTTP` on view `To_Internet`:

```
<permission roleName="Private"
            activityName="Web_HTTP" viewName="To_Internet"/>
```

Notice that our approach enables the administrator to precisely specify which hosts are empowered in a given role. This is why it is not necessary to include prohibitions in the network security policy specification. Our methodology is based on a closed policy that only includes permissions, even though deny rules may be generated from these permissions to configure a given target firewall. This point is addressed in the following section.

4. Derivation of concrete firewall rules

We aim at deriving network components configuration from a network security policy. This policy is expressed in the XML syntax based on the Or-BAC semantics. In this section, we illustrate with the help of the running example introduced in section 4, how our approach is used to derive firewall rules. There are two steps in this derivation process. In the first step, we generate, from the Or-BAC policy, rules expressed in an intermediary multi-target[1] firewall language that also uses an XML syntax (see section 4.1 below). In the second step, we derive, from this intermediary language, concrete configuration rules expressed in the specific target firewall language. In the next two sub-sections, we give details about these two-steps derivation process.

4.1 From abstract policy to generic firewall rules

The aim of this first XSL transformation is to derive generic firewall rules from the abstract network security policy expressed with the Or-BAC formalism. This process uses hierarchy mechanisms to derive concrete rules relevant for each firewall involved in the security architecture. For this purpose, once

[1] That is to say a language that is independent from the the target firewall.

the network security policy of a given organization is specified (organization H_LAN in our example), we have simply to specify which entities (roles, activities and views) are relevant in the sub-organizations (firewalls H_fwi and H_fwe in our example).

The derivation process is then able to automatically distribute every permission that each firewall has to manage. More precisely, if there exists a permission that bounds a role R_1 and a "target role" view corresponding to role R_2, and if both roles R_1 and R_2 are relevant in a sub-organization, it means that this permission will be managed by this sub-organization. For example, the permission that bounds the role `Private` and a target role `To_DNS_server` is relevant for the internal firewall H_fwi because both roles `Private` and `DNS_Server` are relevant in this sub-organization. When a role is relevant in a sub-organization and the target role is relevant in another organization, we cannot distribute the permission on only one organization. So, in this case we conclude that this rule must be managed by both firewalls. For instance, with this mechanism the permission that bounds `Private` and `To_Internet` presented in section 3.6 is relevant to the two firewalls and must be duplicated on both of them.

Notice that the derivation process must check that there is no loops in role definitions. That is we construct a graph between roles where an edge from role R_1 to role R_2 means that role R_2 appears in the `inclusionRole` or `exclusionRole` elements of R_1 role definition. Then, we have to check that this graph is free of loops else the security policy is rejected by the derivation process.

To generate the security rules in the multi-target firewall language, the derivation process parses abstract security rules specified in the Or-BAC model to unfold role definition, activity definition and view definition.

4.2 From generic rules to specific firewall rules

In order to validate our approach, we have chosen to derive concrete rules for NetFilter[2]. So we have designed XSL transformations to derive NetFilter rules from the multi-target firewall language. Using the same example of permission, the following configuration script for NetFilter shows the rules we obtain when applying the specific XSL transformations for NetFilter:

```
iptables -N Intranet-Web_HTTP-To_Internet
iptables -A FORWARD -s 111.222.2.0/24 -p tcp --dport 80
         -j Intranet-Web_HTTP-To_Internet
iptables -A Intranet-Web_HTTP-To_Internet -s \$Admin -j RETURN
iptables -A Intranet-Web_HTTP-To_Internet -s 111.222.2.1/32 -j RETURN
iptables -A Intranet-Web_HTTP-To_Internet -s 111.222.1.254/32 -j RETURN
```

[2]NetFilter is embedded in all Linux kernels upper than 2.4 version.

```
iptables -A Intranet-Web_HTTP-To_Internet -d 111.222.0.0/16 -j RETURN
iptables -A Intranet-Web_HTTP-To_Internet -j ACCEPT
```

The first rule creates a chain called *Intranet-Web_HTTP-To_Internet*. The second rule corresponds to the positive inclusion condition. When a given packet matches this rule, the decision is to jump to the chain *Intranet-Web_HTTP-To_Internet* to check the negative exclusion conditions. If this packet matches a condition in the sub chain, then the decision is to return to the main chain and the next rule in this chain will be checked. Else, if all exclusion conditions expressed in the chain do not match, the packet is accepted. In our example, the first exclusion condition corresponds to the `Admin` that is excluded from the `Private` role. The next two conditions exclude the two internal firewall interfaces. The third condition excludes every destination address that does not belong to the H_LAN network. Notice that it is possible to optimize the rule generation to remove the condition on the firewall interface with address 111.222.1.254 because this firewall interface (corresponding to the DMZ interface) is not included in the subnet 111.222.2.0/24.

Notice also that the order of the generated rules does not matter, as for the abstract security rules, except for the last rule in our example (the rule that takes the decision to accept the packet) that must be at the end of the chain. Thanks to the chain mechanism of NetFilter, we can derive rules without ordering exclusion conditions. But for firewall languages that do not provide such mechanism, exclusion conditions will correspond to *deny* rules. These *deny* rules have to be interleaved with *accept* rules to obtain the same result as with NetFilter chains. Ordering the rules has to be done by the XSL transformation process.

5. Comparison with related works

Most of firewall products are supplied with configuration tools with friendly user interfaces. But these tools discard the main problem of managing security. They do not give a way to think and specify an access control policy before deriving firewall configuration rules that enforce this policy. When the security administrator chooses a particular firewall, may be a commercial product like Cisco PIX [Degu and Bastien, 2003] or Checkpoint Firewall-1 [Checkpoint, 2004] or an open source product like FireHOL [Tsaousis, 2004] or Firewall builder [Kurland, 2003], it is quite easy to set filtering rules using the configuration tool included in the offering. There is however a snag: These rules can be inconsistent with each other or/and with the global security policy leading to security holes. Our approach avoids the administrator pondering on access security using filtering rules. The specification of the access control policy is done at a more abstract level and problems like inconsistency are solved before generating the concrete filtering rules [Cuppens and Miège, 2003a].

There are some works coming under the same topic as ours, that is: (1) specifying a network security policy which is not topology-dependent but rather inspire it, (2) specifying a network security policy which is independent from a particular firewall product and (3) generating automatically the configuration rules from the high level network security policy. Hence, firewall management toolkit Firmato [Bartal et al., 1999] uses an entity-relationship model to specify both the access security policy and the network topology and makes use of the concept of *roles* to define network capabilities. In this approach there is some mixing between the net topology – a particular concrete level – and the access security policy to be enforced so that the role concept becomes ambiguous. Indeed, the authors are bounded to introduce the "*group*" concept with an unclear semantics ; sometimes *group* is used to design a set of hosts and sometimes it stands for a role. This can lead to some difficulties to assign network entities to the model entities. In this connection, Firmato's authors use privileges inheritance through hierarchy of groups to derive automatically permissions. They also make use of artifices to avoid permission leakage. Hence, they introduce notions of "*open group*" to authorize inheritance of permissions and "*closed group*" to prohibit it. The reason is the fact that concept of group is not well defined and we claim that this concept is not needed at the access control policy specification level.

Another work whose motivations are close to ours is the RBNS model [Hassan and Hudec, 2003]. Although authors claim that their work is based on the RBAC model [Sandhu et al., 1996], it seems that they keep from this model only the concept of *role*. Indeed, the specification of network entities and role and permission assignments are not rigorous and does not fit any reality. In particular, (1) all RBNS relations are binary even though an access control security goal and its equivalent filtering rule are always a triple ⟨source, service, target⟩. This leads to a loss of information: *permissions* are missing in RBNS model although authors consider the assignment of a service to an IP address as a permission which is semantically weak. (2) Hosts are of two kinds, *client* or *server* and roles are assigned to hosts thanks to the pre-declared type of hosts. This is a wrong assignment as at the abstract level role of a given host is service-dependent. (3) the approach makes an excessive use of the concept of *role*, hence this leads authors to introduce a role-to-role assignments which is a "*limping*" use of a role based access control model as it means assigning a permission package to another permission package.

Using Or-BAC to model network security policy allows security administrator to make a clear separation between network entities and abstract model entities like roles, services, groups of hosts having the same role, hosts concerned by the source host query, and so on. Indeed, Or-BAC gives an accurate semantics to permissions assigned to hosts (roles), services (activities) and hosts targeted by the client host queries (views). Hence, some one is never

surprised when he/she checks or analyzes configuration rules of the LAN firewalls as they are derived from well specified access control policy. If a security problem occurs, it is due to a wrong security policy.

6. Conclusion

We have presented a formal approach to specify network security policies based on the semantics of the Or-BAC model. We suggest an XML syntax to specify an abstract level network security policy independently from the implementation of this policy in a given firewall. Our proposal uses high level concepts such as inheritance hierarchies between organizations, roles, activities and views. We show how to use these inheritance hierarchies to distribute the abstract level network policy specification over several security components. We illustrate this approach by an example of security architecture based on two firewalls. We then design a translation process in XSLT to generate filtering configuration rules of specific firewalls. This approach has been implemented for the NetFilter firewall.

The originality of our proposal is that it provides a clear semantics link between an abstract access control model, namely Or-BAC, and its implementation into specific security components, namely firewalls. Our approach provides a high level of abstraction compared to the final security rules used to configure a firewall. This should simplify management of such security rules and guarantee portability between firewalls.

There are several perspectives to this work. First, we plan to apply our approach to derive security configuration rules of other network security components, in particular other firewall configuration languages (Cisco PIX, Check-Point Firewall-1, ...) but also Network Intrusion Detection System (for instance Snort). In this later case, we plan to use the `content` element associated with the message structure suggested in section 3.1 to specify high level IDS signatures. A similar approach may also apply to configure other security components such as access control modules of operating systems or database management systems. Our final objective would be to actually use Or-BAC to specify the global security policy of a given organization, and then using a decomposition mechanism, derive configuration rules of various security components involved in a given security architecture.

Finally, in the case of already configured security components, another application of our approach would be to specify an abstract security policy and then develop mechanisms to check if the concrete security rules are consistent with this abstract security policy. Some proposals for such an approach are already suggested in [Mayer et al., 2000].

Acknowledgement

The work presented in this paper is supported by the ACI DESIRS of the French ministry of Research.

References

Bartal, Y., Mayer, A., Nissim, K., and Wool, A. (1999). Firmato: A novel firewall management toolkit. In *20th IEEE Symposium on Security and Privacy*, pages 17–31, Oakland, California.

Checkpoint (2004). Firewall-1. In http://www.checkpoint.com/.

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

Cuppens, F., Cuppens-Boulahia, N., Sans, T., and Miège, A. (2004b). A Formal Approach to Specify and Deploy a Network Security Policy. In http://www.rennes.enst-bretagne.fr/ fcuppens/articles/fast2004.pdf (full version of the paper presented at FAST 2004).

Cuppens, F. and Miège, A. (2003a). Conflict management in the Or-BAC model.

Cuppens, F. and Miège, A. (2003b). Modelling contexts in the Or-BAC model. In *19th Annual Computer Security Applications Conference*, Las Vegas.

Cuppens, F. and Miège, A. (2004). Administration Model for Or-BAC. *Journal of Computer Systems Science and Engineering (CSSE)*. To appear.

Degu, C. and Bastien, G. (2003). CCP Cisco Secure PIX firewall Advanced Exam Certification Guide.

Hassan, A. and Hudec, L. (2003). Role Based Network Security Model: A Forward Step towards Firewall Management. In *Workshop On Security of Information Technologies*, Algiers.

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

Kurland, V. (2003). Firewall Builder. *White paper*.

Mayer, A., Wool, A., and Ziskind, E. (2000). Fang: A Firewall Analysis Engine. In *21th IEEE Symposium on Security and Privacy*, pages 177–187, Oakland, California.

Russell, R. (2002). Linux 2.4 Packet Filtering. In http://www.netfilter.org/documentation/HOWTO//packet-filtering-HOWTO.html.

Sandhu, R., Coyne, E. J., Feinstein, H. L., and Youman, C. E. (1996). Role-Based Access Control Models. *IEEE Computer*, 29(2):38Ű47.

Tsaousis, C. (2004). FireHOL, R5 V1.159. In http://firehol.sourceforge.net/.

W3C (2004). Extensible Markup Language (XML) 1.0 (Third Edition). In http://www.w3.org/TR/REC-xml/.