RELIABLE PROCESS FOR SECURITY POLICY DEPLOYMENT

Stere Preda†, Nora Cuppens-Boulahia†, Frédéric Cuppens†, Joaquin G. Alfaro†,‡ and Laurent Toutain†

†GET/ENST Bretagne, 2 rue de la Châtaigneraie, 35512 Cesson Sévigné, France

‡Universitat Oberta de Catalunya, Rambla Poble Nou 156, 08018 Barcelona, Spain

Keywords: Network Security, Security Devices, Security Rules, Deployment of Policies, Policy Anomalies.

Abstract: We focus in this paper on the problem of configuring and managing network security devices, such as Firewalls, Virtual Private Network (VPN) tunnels, and Intrusion Detection Systems (IDSs). Our proposal is the following. First, we formally specify the security requirements of a given system by using an expressive access control model. As a result, we obtain an abstract security policy, which is free of ambiguities, redundancies or unnecessary details. Second, we deploy such an abstract policy through a set of automatic compilations into the security devices of the system. This proposed deployment process not only simplifies the security administrator's job, but also guarantees a resulting configuration free of anomalies and/or inconsistencies.

1 INTRODUCTION

Specifying, deploying and managing access control rules for a network architecture is one of the main tasks of a security administrator. These rules are usually implemented by different security devices, such as firewalls, virtual private network (VPN) tunnels and intrusion detection systems (IDSs). The configuration of these devices must be compatible with an established security policy. In order to ensure this compatibility in the case of a simple network architecture, the configuration of the security devices may be obtained directly by translating the security requirements into packages of specific rules for each of these devices. When the architecture is more complex and involves several security devices, this procedure may lead to anomalies in the configuration of these devices and become an important source of errors exploited by potential attackers.

These anomalies can be classified as follows. *Firewall anomalies*, also defined in the literature as *intra-* and *inter-firewall* anomalies (Al-Shaer et al., 2005), and that refer to those conflicts that might exist within the local set of rules of a given firewall (*intra*) or between the configuration rules of different firewalls that match the same traffic (*inter*); *tunneling anomalies*, which refer to those conflicts that might exist when both firewalls and VPN tunnels match the same traffic; and *intrusion detection system anomalies*, which refer to those conflicts that might exist when both firewalls and IDSs match the same traffic.

There actually exist several proposals that address the problem of managing security policies free of anomalies. In (Bartal et al., 1999), for example, the authors propose a refinement mechanism based on a high level language and that performs an automatic firewall deployment through a refinement. However, its approach is not fully satisfactory since it does not apply a complete separation between the abstract security policy and the security device features and technology. The authors in (Cuppens et al., 2004b) propose a more complete proposal by using the *Organization Based Access Control* (OrBAC) model (Kalam et al., 2003) as a high level policy language and an ulterior set of compilations that derive the OrBAC specifications into specific device configurations. Unfortunately, only firewall management is addressed in such an approach. Other approaches, such as (Al-Shaer et al., 2005; Fu et al., 2001; Alfaro et al., 2006b), on the other hand, present audit solutions for the analysis of more complex security setups, where not only firewalls, but also VPN devices and IDSs, are in charge of the whole network's security. However, the main drawback of these audit approaches relies on their lack of knowledge about a global security policy, which is very helpful for maintenance and troubleshooting tasks.

In this paper, we extend the refinement approach

Preda S., Cuppens-Boulahia N., Cuppens F., G. Alfaro J. and Toutain L. (2007).
RELIABLE PROCESS FOR SECURITY POLICY DEPLOYMENT.
In *Proceedings of the Second International Conference on Security and Cryptography*, pages 5-15
DOI: 10.5220/0002119200050015

presented in (Cuppens et al., 2004b), and propose a more complete refinement process to derive not only firewall configurations, but also VPN/IPSec (Internet Security Protocol Suite) and scenario-based IDSs configurations. We propose a 2-steps process to (1) formally specify the global set of security requirements by using an expressive access control model based on OrBAC; and (2) a set of ulterior compilations to automatically transform such an abstract security policy into the specific configuration of each security device deployed over the system (e.g., firewalls, VPN/IPSec tunnels and IDSs). This strategy not only simplifies the administrator's job, but also guarantees that the management of policies at both high and specific level is completely free of anomalies, i.e., ambiguities, redundancies or unnecessary details.

The rest of this paper is structured as follows. Section 2 presents some related works. Sections 3 and 4 overview our strategy and introduce the main aspects of our expressive access control model. Section 5 presents our deployment algorithms. Finally, Section 6 closes the paper with some conclusions and work in progress not covered in this paper.

2 RELATED WORK

There exist in the literature several proposals to manage and deploy access control policies on security devices free of anomalies. We overview in this section those works that we consider close to ours.

A first approach presented in (Bartal et al., 1999) proposes a refinement mechanism based on a high level language that allows administrators to perform automatic firewall deployments. It uses the concept of roles to define network capabilities, and propose the use of an inheritance mechanism through a hierarchy of entities to automatically generate permissions. However, this approach is not fully satisfactory since it does not apply a complete separation between the abstract security policy and the security device features and technology. More specifically, it does not fix clear semantics, and its concept of role becomes ambiguous. A similar refinement approach is also presented in (Hassan and Hudec, 2003). However, and although the authors claim that their work is based on the RBAC model (Sandhu et al., 1996), it also presents a lack of semantics — it seems that they only keep from the RBAC model the concept of role. Indeed, the specification of network entities, roles, and permission assignments are not rigorous and does not seem to fit any reality.

The authors in (Kalam et al., 2005) present an intrusion detection approach to enforce a security policy. They propose the use of a ”neutral language” to define a global policy which is further deployed into a heterogeneous system. As their work is focused on a Linux protection language, the rules that cannot be translated into file access rules are to be translated into IDS or firewall rules. However, although the distribution of the global policy into the system is done in a manual fashion on different hosts/nodes, there is no algorithm explaining the choice of these hosts/nodes that optimally respond to global security requirements. Although some verification processes try to guarantee anomaly-free policies, only local configurations are considered. Some drawbacks when managing those anomalies are moreover pointed out in (Blanc et al., 2004) and no solution has been yet presented. Furthermore, no IPSec devices are taken into account.

The work presented in (Cuppens et al., 2004b) successfully applies a set of refinement transformation to derive from an abstract security policy based on the OrBAC model (Kalam et al., 2003) into the network's firewalls that might be enforced. However, the network administrator has to assist the deployment of the access control rules by indicating which firewall implements a specific rule. For instance, concerning a given rule stating a certain traffic is allowed (e.g., the *ftp* service) between two hosts, the administrator has to indicate which firewalls should implement an accept rule to fulfill this requirement. We extend in this paper this later approach by introducing new security devices (VPN/IPSec-based tunnels and IDSs), improving some of the previous limitations, and guaranteeing that neither VPN nor IDS anomalies may apply over resulting setup.

Some other approaches propose to directly analyze existing configurations in order to warn and fix inconsistencies. The work presented in (Fu et al., 2001), for example, concerns the analysis of VPN overlapping tunnels in order to detect *tunneling anomalies*. In their approach, if an access rule concerning a protected traffic between two points is implemented by configuring more than one IPSec overlapping tunnels, the risk is that in some network zones the IP packets circulate without any protection. The authors in (Fu et al., 2001) present a discovery process to detect such situations and propose a high-level language to deal with VPN policies. However, a significant aspect is ignored in their approach: the whole security policy cannot be seen as two independent aspects — VPN tunnels and the firewall issues. They should not be separately modeled. Otherwise, there is a risk of conflicts at the end of their process. The use of a single access model, as our approach does, solves this limitation and allows us to deal with security aspects as a whole.

In (Hamed and Al-Shaer, 2006), a complete taxonomy of conflicts in security policies is presented, and two main categories are proposed: (1) *intra-policy* anomalies, which refer to those conflicts that might exist within the local configuration of security devices; and (2) *inter-policy* anomalies, which refer to those conflicts that might exist between the configuration rules of different security devices that match the same traffic. The authors in (Al-Shaer et al., 2005) propose, moreover, an audit mechanism in order to discover and warn about these anomalies. In (Alfaro et al., 2006a; Alfaro et al., 2006b), some existing limitations in (Hamed and Al-Shaer, 2006) are pointed out, and an alternative set of anomalies and audit algorithms to deal with these anomalies are proposed. However, as noted in (Alfaro et al., 2007), the main drawback of these solutions relies on the lack of knowledge about the security policy as a whole — from a global point of view — which is very useful for maintenance and troubleshooting tasks. The managing of anomalies during our refinement process, not only guarantees equivalent results, but also keeps with such a knowledge.

Support tools can also be used to assist administrators in their task of configuring security devices. The Cisco Security Manager (Cisco Systems, Inc.), for example, is designed to support the security policy deployment on a heterogeneous network involving a large diversity of cisco-based devices. However, we observe the following problems when using such a tool. First, it does not offer a semantic model rich enough to express a global security policy. Although there is the possibility of defining variables, and thus defining access rules involving such variables, the administrator tasks are not much simplified. The administrator always needs a global view of the topology in order to correctly specify each rule to network devices; there is no automatic discovery of security devices that optimally implement an access rule involving an IP source and a destination, as our approach does. Furthermore, the lack of a real top-down approach as ours (cf. Section 3.1) is partially replaced by other tools — e.g., conflict discovery tools that need the administrator's assistance and that unfortunately only guarantee conflict resolution for local configurations.

3 SECURITY POLICY EXPRESSION

3.1 Downward Approach

Let us start by showing in Figure 1 the strategy of our approach. The informal security requirements specified in current language (*informal layer*) are first translated into a high level language based on the OrBAC model (Kalam et al., 2003). Although this translation to the OrBAC-based policy expression can not be wholly automatic, the abstract concepts in the OrBAC model (cf. Section 3.2) facilitate this translation for an administrator. Based upon this abstract security policy that is detached from any specific security device technology (e.g., NetFilter (Welte et al.)), we defined a set of deployment algorithms marked in Figure 1. These compilers are further detailed in Section 5. The first compilation is iterated every time a *sub-organization* (e.g., a firewall) is revealed (in Section 3.3 we will explain the element that determines these iterations). The result is a package of rules written in a generic expression (*multi-target*) and not for a specific technology. The second compilation takes into account the specific technology and grammar (syntax & semantics) of the security devices. For example, different transformations have to be conceived when dealing with NetFilter, Netasq or Cisco PIX firewalls.

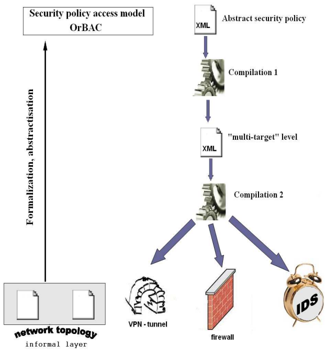


Figure 1: Downward approach.

3.2 Security Policy Specification

OrBAC is the access control model we used to express the abstract security policy (Kalam et al., 2003). This model involves two levels of abstraction: (1) an organizational level (”role”, ”activity”, ”view” and ”context” concepts); and (2) a concrete level (”subject”, ”action”, ”object”) — that are entirely compatible with our downward approach. The OrBAC model uses first order logic to write access control rules in the form of permissions (*Is_permited*), prohibitions (*Is_prohibited*) and obligations (*Is_Obliged*). For example, a permission is derived as follows:

$$\begin{array}{l} \forall\, org, \forall\, s, \forall\, o, \forall\, \alpha, \forall\, r, \forall\, \nu, \forall\, a, \forall\, c \\ permission(org, r, a, \nu, c) \wedge \\ empower(org, s, r) \wedge use(org, o, \nu) \wedge \\ consider(org, \alpha, a) \wedge hold(org, s, a, o, c) \\ \rightarrow Is\_permitted(s, \alpha, o) \end{array}$$

If the organization *org* grants role *r* the permission to perform activity *a* in view ν in context *c* and if the role *r* is assigned to subject *s* (*empower*), the object *o* is used in ν (*use*) and α is considered the action implementing activity *a* (*consider*), *s* is granted permission to perform α on *o*. Let us note that the new concepts introduced by OrBAC are the following: (1) *Activity*, regrouping *actions* having common properties; (2) *View*, several *objects* having the same properties on which the same rules are applied; and (3) *Context*, a concept defining the circumstances in which some security rules can be applied.

OrBAC is based on the *organization* concept assigned to each network entity that deals with a part of the security policy. If the (virtual) LAN the security policy is designed for, constitutes an organization then a firewall, an IDS or an IPSec device become *sub-organizations* (organization hierarchy) of this LAN organization. Roles are assigned to *subjects*, i.e., active entities in the network (e.g., a host, a server, a firewall interface). A subject is assigned one or several roles and will therefore obtain certain permissions. The notion of *role* facilitates the handling of subjects and permissions. Permissions are obtained for each of the subjects according to their role. The *activities* are an abstraction of the network services. For example, the action defined as ”ALL_TCP” includes all tcp network services; ”WEB” refers to https (port 443) and http (port 80).

A view regroups the *objects*. As we have seen, at the concrete level of the OrBAC model, the rules appear as *Is_permitted(s,* α*, o)* meaning that an entity/subject *s* has the permission to perform the action α on the object *o*. Hence, the object is either a network entity (e.g., a web server) identified by its IP address or an IP packet with a given data payload. The *context* allows the definition of specific security requirements directly at the OrBAC level. Some of the permissions occur in a ”protected” context; this leads to the configuration of an IPSec tunnel. On the other hand, a scenario-based IDS alert is triggered in a ”vulnerability” context associated with an attack with a known signature; a specific IP payload may also be specified as a part of the attack signature.

Finally, OrBAC, as also RBAC does (Sandhu et al., 1996), defines role hierarchies, and also views, activities and context hierarchies (Cuppens et al., 2004a). In the specialization/generalization hierarchy, permissions and prohibitions are inherited *downward*. These hierarchies facilitate the administrator’s task by attributing privileges and also simplify the formalization of the security policy.

3.3 OrBAC Security Policy

The authors in (Cuppens et al., 2004b) describe an XML-based OrBAC security policy implementation. We chose to keep the same XML environment, but we slightly modified and proposed new XML data structures to handle the IDS and IPSec devices. The network architecture shown in Figure 2 will be used to explain our methodology. It illustrates a private network (the ”Corp” network: 111.222.0.0/16) including even geographically different sites.

Accordingly to the OrBAC model, the scheme describing the high level policy includes an organization. It is composed of the following parts: (1) The organization’s name; (2) An element describing the organization structure; (3) A set of rules (permissions); and, if necessary, (4) a reference to a higher level organization (organization hierarchy).

In the ”structure” element, we distinguish the entities relevant for the security policy (the *subjects*) that compose the network, the *roles* assigned to these entities and finally, the network services. The entities can be ”host”, ”subnet” or ”address_interval” types. Also, the entities exclusion is used to simplify the structure representation. As an example, the Internet entity is defined as 0.0.0.0/0 excluding the corporate ”Corp” network 111.222.0.0/16:

```
<entity>
    <entityName>Net</entityName>
    <subNet>
        <addr>0.0.0.0</addr>
        <mask>0</mask>
    </subNet>
    <exclusionEntity>
        <entityName>Corp</entityName>
    </exclusionEntity>
</entity>
```

The *roles* are assigned to the entities (”entityName”). The specialization/generalization role hierarchy is used to simplify the OrBAC rule expression. This hierarchy is indicated by an XML child element of the ”role” element: ”seniorRole”. For instance, the role ”R_FW” or ”R_VPN” is inherited by all subjects having firewall functionalities (e.g., ”FW_Extern”, in Figure 2), respectively IPSec functionalities (e.g., ”FW_Intern”). In the following ex-

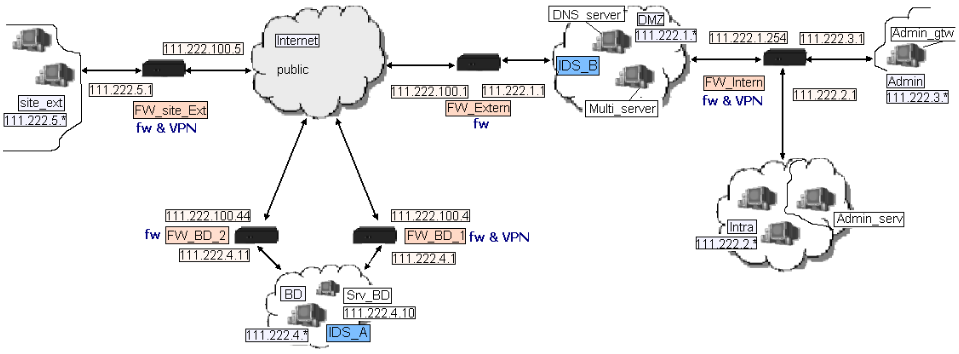


Figure 2: Topology example.

ample, the role ”R_DNS_srv” (DNS server) is assigned to the entity/subject ”DNS_server” that inherits the role of a server - ”R_Srv”. Thus, an access rule implying the role ”R_Srv” will automatically be propagated to DNS server and other servers:

```
<role>
    <roleName>R_DNS_srv</roleName>
    <seniorRole>
        <roleName>R_Srv</roleName>
    </seniorRole>
    <entityName>DNS_server</entityName>
</role>
```

The permissions at the abstract level respect the data structure shown in Figure 3. This XML schema is compliant with the OrBAC specification. The role ”roleName” performs the activity ”serviceName” on the object ”target” with the role ”target/roleName”. The ”context” element is optional. If the security policy does not specify any particular conditions in which this permission is attributed to the role ”roleName”, the context is ”default”. Otherwise, the security policy may announce a ”protected” context or a ”vulnerability” one. In the first case, an IPSec-based tunnel must be created according to the ”child elements” of the ”protected” context: the type of the encryption algorithm (e.g., AES), the entities which have to negotiate the tunnel, a time interval during which the IPSec tunnel is enabled, etc. The second case will correspond to an IDS alert; a possible (XML) attribute of the ”vulnerability” context element may be the CVE vulnerability code if known (MITRE Corp.). Moreover, a ”content” child element of the ”context” will contain a specific data pattern as part of an attack signature.

An important element of the ”permission” is the ”securityRole”. According to the OrBAC terminology, ”securityRole” identifies a sub organization. A ”securityRole” is also responsible for the activation of certain contexts, thus the activation of an access rule. It designates the role attributed to the security device(s) which implement(s) the corresponding access rules. For example, a rule stating that the access from the ”Internet” to the DNS server is allowed will be implemented by the firewall ”FW_Extern”; thus, the ”securityRole” is the role ”R_FW_Extern”. Furthermore, a permission that bounds the role ”R_Intra” and the target ”Internet” will be duplicated on both firewalls ”FW_Intern” and ”FW_Extern”. In this case, the ”securityRole” regroups both ”R_FW_Intern” and ”R_FW_Extern”.

In the case of less complex network architectures, the ”securityRole” is given by the network administrator. Concerning more complex architectures (and a great number of access rules), this security role assignment is difficult to elaborate and can lead to errors. That is why we propose some algorithms to automatically designate the right ”securityRole” under the following two hypothesis:

(1) We consider that the formal policy at the OrBAC level is *correct*[1]: the ”structure” (cf. Figure 3) is well defined, without ambiguities (e.g., a firewall is not attributed the role of a server) and the access rules (cf. the ”permissions” in Figure 3) are well specified (e.g., there is no OrBAC rule shadowed by any other). However, we admit that certain rules cannot be implemented because the adequate security device (with the appropriate functionalities) is missing. Our algorithms can detect this kind of mismatch when deploying the policy.

(2) Inside the (virtual) network the security policy is designed for, the IP packets flow according to the *shortest path* principle (as a routing protocol guarantees). The shortest path principle is used to identify the device(s) on which a given security

[1] We prove in (Cuppens et al., 2006) that such an assumption is feasible.

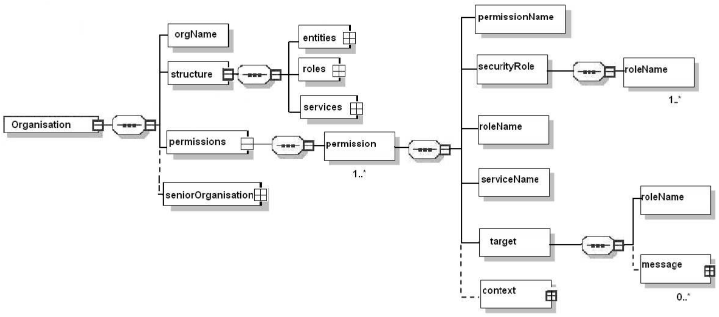


Figure 3: OrBAC organization structure.

rule must be deployed. However, notice that the shortest path is not always unique. For instance, in Figure 2, there are two possible shortest paths from site_ext to Srv_BD. In this case, our algorithms attempt to deploy the security rule on each candidate shortest path.

To achieve this, the OrBAC structure shown in Figure 3 is parsed and relevant information about the network topology is collected. Practically we will obtain a graph and the *shortest path* principle will be applied to it. We describe the methodology in the following section.

4 MODELING THE TARGET ARCHITECTURE

4.1 Modeling the Topology

At the OrBAC level, the security officer identifies the relevant active entities (i.e., *subjects*) and roles assigned to these entities with respect to the network topology and the security requirements. A role can be assigned to more than one entity (e.g., all firewalls have the firewall role ”R_FW”) and an entity can have more than one role (e.g., a firewall can have IPSec functionalities). The hierarchy of roles is defined too (e.g., a multi-server that has the DNS_server and Web_server roles inherits inevitably the server role - a less specialized role). An entity can be either a host, a subnet or an address interval type.

As mentioned in Section 3.3, the entity exclusion is used to achieve a better structuring and management of the entities. The DMZ zone is considered to be the 111.222.1.0/24 subnet excluding the two interfaces of the adjacent firewalls ”FW_Extern” and ”FW_Intern” (cf. Figure 2). Multi-level exclusions may also exist. For example, the ”CorpLessIntra” is the ”Corporate” entity (111.222.0.0/16) which excludes the ”Site_ext” entity (111.222.5.0/16) and the ”Intra” entity (111.222.2.0/24) which in turn excludes the internal interface of the firewall ”FW_Intern”. ”CorpLessIntra” defines briefly all hosts unused by general corporate employees and managed by ”Admin” zone (111.222.3.0/24 - the network administration zone).

Regarding complex network architectures, one of the most difficult tasks for a security officer is to indicate each of the security devices (i.e., ”securityRole”) which optimally implement the access rules. In our approach, the security officer does not need to give such an indication because our algorithms find the right set of ”securityRole” for each ”permission” if ”securityRole” exists. For this purpose, the OrBAC ”structure” is initially parsed and we construct a graph where every node is a *zone*[2]. In order to do so, the following information is automatically extracted during the initial phase of our process:

- The functionalities of each security devices. For example, in Figure 2, the firewall ”FW_Intern” has fw & VPN functionalities (firewall and IPSec capabilities).
- The set of neighbors of each zone (based on their IP addresses and masks[3]). For example, after parsing the ”structure” corresponding to Figure 3, the neighbors of the zone ”FW_Intern” are the ”Intra”, ”DMZ” and ”Admin” zones.

As a result of this first parsing phase, we obtain the following two outputs:

(1) A list of security devices, *Ss*, defined as follows: $Ss = \cup_j \{device_j, functionalities_j, [\cup_n \{neighbors_{jn}\}]\}$.

(2) A list of zones, *Zones*, define as follows: $Zones = \cup_i \{zone_i[neighbors_{ik}\}]\}$, and where $neighbors_{ik}$ is the neighbor k zones of the ith zone.

[2] A *zone* is either a subnet with no security device interfacing any other subnet or the set of interfaces of a security device

[3] The establishing neighbors algorithm is based on the longest prefix matching scheme.

4.2 Modeling Paths

Let us consider a rule at the OrBAC level; it implies a role (”roleName”) and an object (”target/roleName”) that corresponds to respectively a source ”Src” and a destination ”Dest” entities. This information is mandatory at the OrBAC level (cf. Figure 3). As already mentioned, we developed an algorithm that outputs the optimal set of ”securityRole” based on the following three assumptions:

- **source_zone**: $\cup_j\{zone_j\} = Src \cap Zones$;
- **dest_zone**: $\cup_i\{zone_i\} = Dest \cap Zones$;
- **shortest_path** : *Zones* x *Zones* $\rightarrow$ *Ss*, such that shortest_path($zone_j$,$zone_i$)$\leftarrow \cup_k\{device_k\}$.

Once identified the source and the destination zones for an access rule, the shortest paths between a source zone and a destination zone are computed and the security devices on this path are revealed. Some of these security devices will be designated as ”securityRole”. Moreover, the security devices on the shortest path must have the functionalities:

- if the access rule is in a ”default” context then firewall functionalities are necessary;
- if the access rule is in a ”protected” context then IPSec functionalities are required;
- if the access rule is in a ”vulnerability” context, then IDS functionalities are required.

In a ”default” context, a *permission* rule will be implemented on all firewalls found on the path; a *prohibition* rule might simply be implemented on the security device which is the closest to the IP flow source (our *shortest_path* algorithm was designed so as to choose the most *up-stream* — the closest to the source — or the most *down-stream* — the closest to the destination — security device). The fact that an access rule is either a *permission* or a *prohibition* filtering rule is not relevant for our discussion. That is why for didactical reasons, we considered only *permission* rules in the ”default” context.

5 DEPLOYMENT ALGORITHMS

The main part of the security policy deployment is included in the first compilation phase (cf. Figure 1) as a set of four processes:

- The ”structure_parsing” with the main results, i.e., the list of security devices Ss[4] and Zones;
- the ”hierarchies_treatement” including the role hierarchy treatment and the exclusion entities treatment;
- the ”securityRole” phase including the shortest path computation;
- ”multi-target” extracts all relevant information about the set of access rules the ”securityRole” must implement, in a generic format.

The compilation process is schematized in Figure 4. The input is the OrBAC security policy; the intermediary results are represented by dotted lines. The final compilation result is a set of files consisting of the part of the security policy assigned to each ”securityRole” (the ”multi-target” level). ”call” stands for function callings; ”input” means that the intermediary results serve as input for other processes.

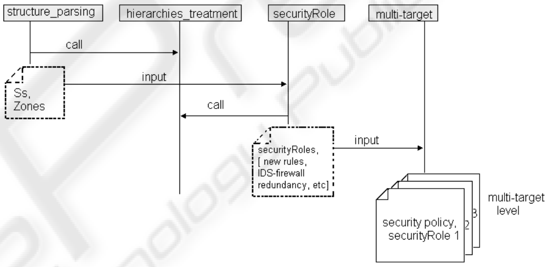


Figure 4: First compilation phases.

Concerning the first compilation, the main algorithms are the ”SecurityRoleDiscovery”, ”exclusion entities treatment” and ”IDS - Firewall_Redundancy”. From the ”multi-target” level, a second compilation is applied to obtain the packages of concrete rules according to the specific syntax of each relevant device.

5.1 The *SecurityRoleDiscovery* Algorithm

The ”securityRole discovery” takes into account, as an input, the ”structure” (cf. Figure 3) parsing results: Ss and Zones. It also uses the *shortest_path* function. Let us assume the ”permissions” = $\cup_i\{permission_i\}$ at the OrBAC level:

A ”permission” rule in a ”default” context is implemented by all security devices with firewall functionalities on the shortest path. For example, given the topology in Figure 2, both firewalls ”FW_site_Ext” and ”FW_Extern” implement an access rule stating that ”ftp” is allowed from the ”site_ext” zone to the ”DMZ” zone.

In the ”protected” context, we formulated an extension to the shortest path function - *passing_by()*:

[4] We recall, from Section 4.1, that Ss is the list of security devices, such that $Ss = \cup_j\{device_j, functionalities_j, [\cup_n\{neighbors_{jn}\}]\}$.

Algorithm 1: SecurityRoleDiscovery

```
foreach $permission_k$ do
    call hierarchies_treatement;
    $\cup_i\{zoneS_i\} \leftarrow (Src \cap Zones)$;
    $\cup_j\{zoneD_j\} \leftarrow (Dest \cap Zones)$;
    foreach $zoneS_i$ do
        foreach $zoneD_j$ do
            if $permission_k[context] = default$ then
                $S_{s\_list} \leftarrow$ shortest_path $(zoneS_i,zoneD_j)$;
                if $S_{s\_list}[functionalities] = fw$ then
                    $securityRole \leftarrow S_{s\_list}$;
                    break;

            if $permission_k[context] = protected$ then
                $Src\_VPN\_list \leftarrow$ find_VPN_components_in $\cup_k\{zoneS_i[neighbors_{ik}]$;
                $Dest\_VPN\_list \leftarrow$ find_VPN_components_in $\cup_k\{zoneD_j[neighbors_{jk}]$;
                if shortest_path $(zoneS_i,zoneD_j)$
                passing_by $(Src\_VPN\_list,Dest\_VPN\_list)$ then
                    call configure_VPN_IPsec_tunnel;
                else
                    warning (“$VPN\_tunnel\_impossible$”);
                    break
                if $permission_k[context] = vulnerability$ then
                    call IDS-Firewall_Redundancy;
```

- it identifies the security devices - neighbors of the source and the destination zones having IPSec functionalities and it is supposed to compute a path between two of them;
- if a path exists, the algorithm finds the right interfaces negotiating the IPSec tunnel (longest prefix matching scheme);
- new filtering access rules are to be implemented on the firewalls discovered on the tunnel path (for example, to enable the IPSec tunnel negotiation - *isakmp* with the above interfaces). This way, we avoid the conflict firewall ↔ IPSec tunnel.

Given the same topology (cf. Figure 2), let us consider an access rule stating that all TCP traffic from the zone ”Intra” to the ”site_BD” zone must be secured: *Is_permited(R_Intra, ALL_TCP, target-R_site_BD, context-protected)*. With the previous algorithm, the ”protected” context leads to the configuration of an IPSec-based tunnel. The IPSec tunnel will be implemented by ”FW_Intern” and ”FW_BD_1” security devices because: (1) a path exists from the ”Intra” zone to the ”site_BD” zone and (2) the path crosses these two security devices with IPSec functionalities in the immediate neighborhood of respectively the source IP traffic zone (”Intra”) and the destination (”site_BD”) zone. The interfaces in charge of the IPSec tunnel negotiation (*isakmp*) are the ”FW_Intern” interface adjacent to the DMZ zone and respectively the ”FW_BD_1” interface adjacent to the Internet zone. A filtering rule permitting the corresponding *isakmp* traffic is automatically deduced and finally implemented in all firewalls on the tunnel path (”FW_Intern”, ”FW_Extern” and also ”FW_BD_1”).

If the access rule is in a ”vulnerability” context, *IDS-Firewall_Redundancy* is called (cf. Section 5.3). Instead of deploying an IDS rule, we chose to exploit an eventual IDS-firewall redundancy. We do not consider any IDS-IPSec tunnel interaction because IDS generally works on an unencrypted IP traffic.

5.2 The *Exclusion Entities Treatment* Algorithm

During the first compilation, we treat the hierarchies of roles and activities (network services). Consequently, a permission involving the firewall role ”R_FW” will engage all entities/subjects playing a ”R_FW” role. These entities may contain other entities which are excluded. Deploying an access rule implying subjects/entities which exclude other entities is solved as follows:

- a permission at the OrBAC level involving entity E1, E1 excluding E2 will be translated in a generic rule which will include E1, E1 excluding E2;
- a permission involving E1, E1 excluding E2, E2 excluding E3 will be translated in two generic rules: the first will involve E1, E1 excluding E2 and the second will involve E3;
- a permission involving E1, E1 excluding E2, E2 excluding E3, E3 excluding E4, will be translated in two generic rules: the first will include E1, E1 excluding E2 and the second will include E3, E3 excluding E4;
- a permission involving E1, E1 excluding E2 and E3, E3 excluding E4 and E5, E5 excluding E6 will be translated in three generic rules: the first will include E1, E1 excluding E2 and E3, the second will include E4 and the third will include E5, E5 excluding E6.

This reasoning derives from a simple mathematical logic; for the last example, the entity E1 is defined as follows:

$$E1 \wedge \overline{(E2 \vee E3) \wedge \overline{E4 \vee (E5 \wedge \overline{E6})}} =$$

$$= (E1 \wedge \overline{E2 \vee E3}) \vee E4 \vee (E5 \wedge \overline{E6})$$

where ”∨” stands for the addition of a new generic permission at the multi target level (cf. Figure 4); and ”∧” denotes an exclusion.

5.3 The *IDS-Firewall_Redundancy* Algorithm

An access rule in the ”vulnerability” context is deployed on a single IDS device on the shortest path binding the source and the destination of an IP flow. We chose the most down-stream IDS because it is more efficient against spoofing attacks than the most up-stream IDS. Moreover, we do not eliminate the IDS-firewall redundancy: IDS alert sets off for IP packets that should have been blocked by an up-stream firewall. We take advantage of this redundancy in order to obtain relevant information regarding a malfunctioning firewall. The ”IDS-Firewall_Redundancy” algorithm is based on the shortest path principle with some extensions. To illustrate our approach, let us consider the topology shown in Figure 2.

A rule in the ”vulnerability” context for an IP flow with the ”Intra” source zone and the ”BD” destination zone will be implemented by IDS_A. An analysis of the firewall configurations located on the *shortest path* connecting the ”Intra” source and the ”BD” destination is launched. We choose to ”analyze” only the firewalls located up-stream; IDS_A cannot give any information regarding its malfunctioning down-stream firewalls.

Let us consider that the up-stream firewalls (”FW_BD_1”, ”FW_BD_2”, ”FW_Extern”, ”FW_Intern”) apply a default *deny all* filtering policy. In this case, with our security policy deployment, they implement only permission rules (*accept* or *pass*). Each rule generally involves an IP source and destination address and a network service (*ports*). Let (S,D,P) be the triplet including this information. The entire policy of a firewall (e.g., ”FW_Intern”) may be resumed as follows: (1) **pass**” for F = $(S_1 \wedge D_1 \wedge P_1) \vee (S_2 \wedge D_2 \wedge P_2) \vee \ldots \vee (S_n \wedge D_n \wedge P_n)$, where n = the filtering rules number; (2) ”**deny**” for non(F); (3) F and non(F) are included in the ”3D” space [*source*, *destination*, *service*] where (*source*, *destination*) ∈ [0.0.0.0 ; 255.255.255.255] and *service* ∈ [0 ; 65536].

We refer to the IDS-firewall redundancy only if the set of parameters forming the firewall and the IDS rules are the same; thus, the (S,D,P) triplet is taken into account alongside the IDS ”alert” rules and the firewall ”deny” rule. For example, there is an IDS-firewall redundancy if an IDS alerts for the IP packets including W = (111.222.2.0/24, 111.222.4.10, all_tcp) (regardless of their payload) and an up-stream firewall performs *deny* for the (111.222.2.32/27, 111.222.4.10, all_tcp) triplet. Our ”IDS-Firewall_Redundancy” algorithm finds the intersection between each triplet W (corresponding to an *alert*) and non(F_i) for each up-stream firewall FW_i.

Since Wr = W ∩ non(Fi) and Wa = W-Wr, then:

- A first IDS rule will be implemented in the immediately down-stream IDS_i against the firewall FW_i the previous intersection was computed for. The IDS alert message will not only be the message corresponding to the attack but will also include ”beware, malfunctioning FW_i”. For example, in the case of IDS_A↔”FW_Intern” redundancy, involving Wr = W ∩ non(F), an IDS rule involving Wr will be implemented in IDS_B and thus the ”FW_Intern” malfunction can be detected.

Algorithm 2: `IDS_Firewall_Redundancy`

```
// Let $IDS_N$ be the most down-stream IDS of the
// $sortest\_path(source, destination)$.
//
// Let $FW_N$ be the last firewall of the
// $sortest\_path(source, destination)$ placed just before $IDS_N$.
//
foreach $FW_i$ ∈ shortest_path (source,destination) do
    if exists ($IDS_i$ ∈ shortest_path (source,destination) then
        $W_r \leftarrow W \cap non(F_i)$;
        $W_a \leftarrow W - W_r$;
        if $W_r \neq \emptyset$ then
            alert_with_Wr_in ($IDS_i$) & “malfunction $FW_i$”;
            $W \leftarrow W_a$;
    else if $IDS_i = IDS_N$ then
        $W_r \leftarrow W \cap non(F_N)$;
        $W_a \leftarrow W - W_r$;
        if $W_r \neq \emptyset$ then
            alert_with_Wr_in ($IDS_N$) & “malfunction $FW_N$”;
        alert_with_Wa_in ($IDS_N$);
        break;
```

- The last IDS rule will be implemented in the most down-stream IDS and include Wa. For the previous example, an IDS rule with Wa and the unmodified message corresponding to the initial vulnerability will be implemented in IDS_A.

5.4 Final Phase

The second compilation phase (cf. Figure 1) translates the generic rules to a specific security device technology. We deal with a library of transformations. Each transformation is conditioned by the security device features.

The intrinsic matching rule (*first-matching*, or *last-matching*) and the rule order are taken into account when designing a transformation for a firewall. Let us consider that an entity excludes another entity in one generic *permission*; we have the choice to design either a transformation that outputs two rules (one of them being the *negative* rule and corresponding to the excluded entity) or a transformation in which the excluded entity is actually left out (the result is a single *pass*/*accept* rule). In the first case, the negative rule must be placed before (first-matching) or after (last-matching) the positive rule.

NetFilter offers a mean to skip the rule order importance. The authors in (Cuppens et al., 2004b) use the *jump* and *new chain* functionalities each time exclusion entities are involved. However, not all firewalls we dealt with had these functionalities. The order of rules will not be important if we succeed in writing *pass* rules and only the last rule is *deny all*. We designed such a transformation in XSLT language for Netasq F200 IPS. On the other hand, and in order to obtain the IPSec-based tunnel configurations, another transformation is required. Therefore, we conceived one for the Netasq F200 family which also includes the IPSec functionalities. As scenario-based IDS, we worked with SNORT-based IDS, for which we considered only the alert IDS rules and a specific transformation.

5.5 Performance Evaluation

The complete set of algorithms and processes overviewed in this section have been implemented and evaluated in a first software prototype. Let us briefly present in this section some of the results we obtained. The implementation has been done by using PHP, a general-purpose scripting language that is especially suited for web services development. In this way, the complete refinement process can be locally or remotely executed by using a HTTP server (e.g., Apache server over UNIX or Windows setups) and a web browser. On the other hand, the evaluation was carried out on an Intel-Pentium M 1.4 GHz processor with 512 MB RAM, running Ubuntu 6.0.6 with GNU/Linux 2.6.15 (32 bits), and using Apache/2.0.55 with PHP/5.1.2 interpreter configured.

For our evaluations, we specified three different IPv4 simulated networks. The topology for the first network consisted of four subnetworks, one SNORT-based IDS and two firewalls the access rules were deployed on. The topology for the second network included five subnetworks, one SNORT-based IDS, three firewalls - two of them having IPSec capabilities. The topology for the third network consisted of six subnetworks, two SNORT-based IDS, five firewalls - three of them with IPSec capabilities. For each topology we considered several security policies with an incremental number of OrBAC rules.

During the evaluation, we measured the memory space and the processing time needed to perform the whole refinement process. The results of these measurements are plotted in Figure 5(a) and Figure 5(b). We can first notice in Figure 5(a) that an important part of memory consumption is due to the structure parsing phase (cf. Section 4.1) and then the memory increases linearly with the OrBAC rules number. On the other hand, we notice in Figure 5(b) that the processing time is not due to the parsing structure phase but to the OrBAC rules number and complexity.

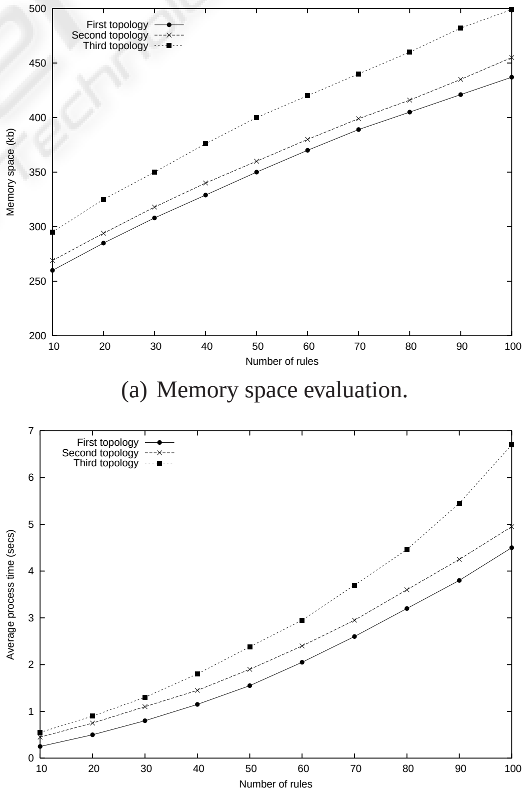
However, although both memory space and processing time results are pointing out to strong requirements, we consider they are reasonable since: (1) the implementation of our approach has been done by using a high level scripting language, and we expect that the use of a more efficient language will clearly improve these results; (2) our approach relies on an off-line process which does not affect the performance of the security policy enforcement.

We want finally to note that the implementation of our proposal in a software prototype demonstrates the practicability of our work; and the obtained results allow us to be very optimistic about its use in more complex security policy scenarios.

6 CONCLUSIONS

The configuration of security devices is a complex and cumbersome task. A wrong configuration of those devices may lead to weak security polices – easily to be bypassed by unauthorized parties. In order to help security administrators, we have presented in this paper a refinement mechanism to properly configure and manage the following security devices: firewalls, VPN/IPSec-based tunnels, and scenario-based Intrusion Detection Systems (IDSs).

Our proposal allows the administrator to formally specify security requirements by using an expressive



(a) Memory space evaluation.

(b) Processing time evaluation.

Figure 5: Memory and processing time evaluations.

access control model based on OrBAC (Kalam et al., 2003). As a result, an abstract security policy, which is free of ambiguities, redundancies or unnecessary details, is automatically transformed into specific security devices configurations. This strategy not only simplifies the security administrator's job, but also guarantees that the resulting configuration is free of anomalies and/or inconsistencies. The complete set of algorithms and processes presented in this paper have been implemented in a first software prototype, and the results of a first evaluation have been overviewed. Such implementation demonstrates the practicability of our work and its performance results allow us to be very optimistic about its use in more complex security policy scenarios.

As work in progress, we are actually studying how to extend our approach in the case where the security architecture includes IPv6 devices. More specifically, the construction of new VPN tunnels (e.g., IPv6-over-IPv4) for IPv6 networks must be revised, and more investigation has to be done in order to extend the approach presented in this paper. In parallel to this work, we are also extending our approach to make cooperate routing and tunneling policies.

REFERENCES

Abou el Kalam, A., Baida, R. E., Balbiani, P., Benferhat, S., Cuppens, F., Deswarte, Y., Miège, A., Saurel, C., and Trouessin, G. Organization Based Access Control. In *IEEE 4th Intl. Workshop on Policies for Distributed Systems and Networks*, pp. 120–131, Lake Come, Italy, 2003.

Abou el Kalam, A., Briffaut, J., Toinard, C., and Blanc, M. Intrusion detection and security policy framework for distributed environments. In *Collaborative Technologies and Systems*, pp.100-106, Missouri, USA, 2005.

Alfaro, J. G., Cuppens, F., and Cuppens-Boulahia, N. Towards Filtering and Alerting Rule Rewriting on Single-Component Policies. In *Intl. Conference on Computer Safety, Reliability, and Security*, pp. 182–194, Poland, 2006.

Alfaro, J. G., Cuppens, F., and Cuppens-Boulahia, N. Analysis of Policy Anomalies on Distributed Network Security Setups. In *11th European Symposium On Research In Computer Security*, pp. 496–511, Germany, 2006.

Alfaro, J. G., Cuppens, F., and Cuppens-Boulahia, N. Aggregating and Deploying Network Access Control Policies. In *1rst Symposium on Frontiers in Availability, Reliability and Security (FARES), 2nd International Conference on Availability, Reliability and Security (ARES2007)*, Vienna, Austria, 2007.

Al-Shaer, E. S., Hamed, H. H., and Masum, H. Conflict Classification and Analysis of Distributed Firewall Policies. In *IEEE Journal on Selected Areas in Communications*, 23(10):2069–2084, 2005.

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

Blanc, M., Clemente, P., Courtieu, P., Franche, S., Oudot, L., Toinard, C. and Vessiller, L. Hardening large-scale networks security through a meta-policy framework. In *Third Workshop on the Internet, Telecommunications and Signal Processing*, Adelaide, Australia, 2004.

Cisco Systems, Inc. Cisco Security Manager Product Information. [Online]. Available from: `http://cisco.com/go/csmanager`

Cuppens, F., Cuppens-Boulahia, N., and Miege, A. Inheritance hierarchies in the OrBAC Model and application in a network environment. In *2nd Foundations of Computer Security Workshop (FCS'04)*, Turku, Finlande, 2004.

Cuppens, F., Cuppens-Boulahia, N., and Ben Ghorbel, M. High-level conflict management strategies in advanced access control models. In *Workshop on Information and Computer Security (ICS 2006)*, Timisoara, Roumania, 2006.

Cuppens, F., Cuppens-Boulahia, N., Sans, T. and Miege, A. A formal approach to specify and deploy a network security policy. In *2nd Workshop on Formal Aspects in Security and Trust*, pp. 203–218, Toulouse, France, 2004.

Fu, Z., Wu, S. F., Huang, H., Loh, K., Gong, F., Baldine, I., Xu, C. IPSec/VPN Security Policy: Correctness, Conflict Detection and Resolution. In *Policy 2001 Workshop*, pp. 39–56, 2001.

Hamed, H. H. and Al-Shaer, E. S. Taxonomy of conflicts in network security policies. In *IEEE Communications*, 44(3):134-141, 2006.

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

MITRE Corp. Common Vulnerabilities and Exposures. [Online]. Available from: `http://cve.mitre.org/`

Sandhu, R., Coyne, E. J., Feinstein, H. L., and Youman, C. E. Role-Based Access Control Models. *IEEE Computer*, 29(2):38–47, 1996.

Welte, H., Kadlecsik, J., Josefsson, M., McHardy, P., and et al. The netfilter project: firewalling, nat and packet mangling for linux 2.4x and 2.6.x. [Online]. Available from: `http://www.netfilter.org/`