Thèse de Doctorat de l'université Paris VI
Pierre et Marie CURIE

Spécialité

SYSTÈMES INFORMATIQUES

présentée par

M. Cássio Ditzel Kropiwiec

pour obtenir le grade de

DOCTEUR de l'université Pierre et Marie CURIE

Framework for Distributed Firewall Administration in a Multi-Constraint Security Policies Context

Sountenance prevue le 10 juillet 2009 devant le jury composé de

Rapporteur	**M. Ahmed SERHROUCHNI**	**Professeur à Telecom ParisTech**
Rapporteur	**Mme. Solange GHERNAOUTI-HÉLIE**	**Professeur à l'Université de Lausanne**
Examinateur	**M. Harry PERROS**	**Professeur à NC State University**
Examinateur	**M. Pascal URIEN**	**Professeur à Telecom ParisTech**
Examinateur	**M. Mauro Sergio Pereira FONSECA**	**Professeur à PUC-PR**
Examinateur	**M. Manoel Camillo PENNA**	**Professeur à PUC-PR**
Président	**M. Guy PUJOLLE**	**Professeur à l'Université PMC**

Thèse de Doctorat de l'université Paris VI
Pierre et Marie CURIE

Spécialité

SYSTÈMES INFORMATIQUES

présentée par

M. Cássio Ditzel Kropiwiec

pour obtenir le grade de

DOCTEUR de l'université Pierre et Marie CURIE

Framework for Distributed Firewall Administration in a Multi-Constraint Security Policies Context

Sountenance prevue le 10 juillet 2009 devant le jury composé de

Rapporteur	**M. Ahmed SERHROUCHNI**	**Professeur à Telecom ParisTech**
Rapporteur	**Mme. Solange GHERNAOUTI-HÉLIE**	**Professeur à l'Université de Lausanne**
Examinateur	**M. Harry PERROS**	**Professeur à NC State University**
Examinateur	**M. Pascal URIEN**	**Professeur à Telecom ParisTech**
Examinateur	**M. Mauro Sergio Pereira FONSECA**	**Professeur à PUC-PR**
Examinateur	**M. Manoel Camillo PENNA**	**Professeur à PUC-PR**
Président	**M. Guy PUJOLLE**	**Professeur à l'Université PMC**

10/07/2009

To my Wife, Edelyse, who supported and encouraged me to accomplish this work.

Acknowledgements

Several people were important during the development of this work. I am very grateful to Edgard Jamhour, Manoel Camillo de Oliveira Penna Neto, Mauro Sérgio Pereira Fonseca and Guy Pujolle for inspiring, motivating and encouraging me.

I am thankful to everyone who directly or indirectly collaborated in the execution of this work.

I am especially thankful to my wife, Edelyse, to whom I dedicate this work, for her constant support, encouragement and comprehension during the period that I dedicated to this work.

Abstract

The management of security policies is an important issue for networks of any size. The policy must be designed to protect the internal resources from external users and also from internal users. In networks with one or only a few firewalls, defining the configuration of each device is easier. However, in larger networks, the administrator must consider the configuration of each firewall isolated and the effects of this configuration in the whole network. This thesis proposes a framework for representing and managing global network security policies for distributed firewall administration. The proposed framework defines a high-level policy language, which allows the specification of policies in mandatory, discretionary and security property models. This framework is able to handle simultaneously the three dimensions and coherently describes the resulting permissions in an abstract representation that is independent of how they will be enforced, without violating the global security goal. The framework also includes a mechanism responsible for translating the abstract representation of permissions into low-level configuration scripts/rules for firewalls of different models and vendors, allowing its use for configuration of heterogeneous networks. Each dimension can be defined by people of different roles, allowing the cooperation in definition of global policy. The framework is formalized in Z to demonstrate its completeness and correctness, and a scalability study is presented to demonstrate the behavior of the framework in larger networks.

Key Words:

Policy, Network Security, Security Management, Firewall Configuration, Formal Validation, High-level Language

Table of Contents

Chapter 1

Introduction

1.1 Motivation

Computer networks are present everywhere around the world. Small networks normally have just one security device that connects the internal network to the Internet. But it is very common the existence of several security devices in larger networks, in order to split the network in several sub networks, simplifying the management and increasing the whole security.

The most vulnerable security device normally is the responsible for the connection of the internal network to the Internet, since the attacks can be originated from virtually anywhere in the world. However, many threats are originated from inside the network. Therefore, it is necessary to control the access from external users and external resources as well to control the access from internal users to internal resources, reducing the risks of attacks originated from the inside of the network [26].

The main equipment used to enforce the network security is the firewall. However, many difficulties arise when configuring network with large number of firewalls, since it is a complex and error-prone task [48]. A generally accepted way to reduce security configuration efforts consists in employing policy based tools. High level policies are specified in terms of which resources the users can access, and should be converted into firewall specific language configuration scripts, in terms of source hosts where the users are located and destinations hosts where the resources are available. Moreover, the security policy must represent the security plan, which contains the organization's security goals and declare responsibilities and other security-relevant organizational issues [46].

In larger networks, it is possible that more than one firewall to be present between the source and the destination. Each firewall must be configured according to its position and the security policy that it must enforce. The conversion mechanism must take the policy and the topology as input to determine the rules that must be applied to each firewall, in a way that the overall security policy is satisfied [2]. When more than one firewall is present between a user and a resource, the rule set can be combined in order to better explore the distinct functionalities offered by the firewalls. This process of configuring several firewalls must be carried carefully, because although each firewall individually can be configured correctly, the connections between them can cause a security violation [25]. In addition, a typical large-scale enterprise network might involve hundreds of rules that might be written by different administrators in different times. This significantly increases the potential of anomaly occurrence in the firewall policy, increasing the risk of security violations in the protected network [35]. Due to complexity of firewall configuration, several works proposes approaches for analyzing the firewall configuration, such as [35][36][37][38][39][40].

Also, as firewalls of diverse models and vendors can be present, it is necessary to consider the specific language used for configuration and the set of rules that can be interpreted and enforced by each firewall in the network before applying the configuration. Ideally, the process of defining security policies should be decoupled from the mechanisms that will actually enforce them over the network. In most organizations, security policies are related to business goals, and are not anymore a purely technical issue [47]. Leaving the responsibility of configuring the entire network policy to one network administrator can be risky, since any mistake can lead to a security hole.

1.2 Objectives and Contribution

In order to address the aforementioned issues, this thesis proposes a Framework for Distributed Firewall Administration in a Multi-Constraint Security Policies Context that introduces a new approach related to the security policy definition and the generation of firewall configuration in a distributed environment.

The main objectives of this framework are:

- Specification of multi-constraint policies, using mandatory, discretionary and security property models;

- Definition of the a centralized global security policy, yet allowing the definition of each policy model by different groups of people;
- Independence of topology;
- Independence of firewall models and vendors;
- Mechanism for translating high-level security policy to low-level script/rule configuration files for each firewall in the network.

The framework adopts a policy model with three dimensions of security policies: mandatory, discretionary and security property. Mandatory policies are coarse grained and reflect the inviolable security restrictions in the organization. The security property policies are restrictions imposed to the sources where the users are located and the paths connecting users to resources. In our framework a path must satisfy some security requirements in order to be allowed. This permits to create policies which are independent on the user or resource location. Finally, the discretionary policies are fine grained, and are subjected to the mandatory and security property policies. The motivation for this division is to support the cooperation of multiple security staff in the security policy definition. For example, the right to define mandatory and security property policies could be assigned to an organizational-level risk management staff while the discretionary policies could be delegated to the local network administrators in several departments in the organization.

After determining the resulting set of permissions allowed by the three policy models, the algorithm performs the distribution of permissions among the firewalls within the network, according to their locations and functionalities. The firewalls are configured with the minimum set of rules that correctly and completely implements the high level security policy. The rules generated by our framework consider the functionalities and syntax of each firewall model through the named service concept, an abstract representation for firewall protected services. A named service encapsulates some firewall capabilities by pointing to a library that contains a vendor dependent implementation for the service.

The process of translating the three-dimensional high level security policy into firewall configuration is highly complex. To assure that the firewall configuration respects the high level definition, we have formalized both the policy model and the translating algorithm using the Z-language notation. The proof of some important theorems permits to demonstrate the coherence

of our approach with respect to the proposed multi-policy model. We also present a complexity analysis, which demonstrates the time necessary to process the rules according to the policy size.

The main contributions of this thesis are:

- Specification of three-dimensional framework for network security policy definition, which allows different security staff to be responsible for the description of each dimension;
- Specification of a high-level security policy language, able to handle mandatory, discretionary and security property policies, independently of topology and firewall models/vendors;
- Definition of an information model that comprises the network topology;
- Definition of the Named Service concept, that is a high-level representation of the firewall protected services;
- Description of the algorithm that process the three policies specifications, responsible for translating the high-level security policy to low-level scripts/rules for firewall configuration;
- Demonstration of the formal validation of the algorithm using a mathematical language;
- Scalability Study of the proposed framework.

The following characteristics of the framework are also innovative and can be viewed as contributions of this work. Because the topology is represented in the information model, the proposed refinement algorithm is able to determine the permissions that must be configured in each firewall along an access path. Moreover, by considering the named services available in each firewall and a mechanism to convert the permissions into firewall rules/scripts at the topmost level, the policy model is both topology and vendor independent.

1.3 Organization of the thesis

This thesis is organized as follows: Chapter 2 presents the traditional access control models and the works that are focused in representation and manipulation of security policies. Chapter 3 presents a description of the common types of firewalls available and their characteristics.

Chapter 4 describes our proposed framework, presenting both the security policy language and the translation algorithm. Chapter 5 presents the Z-language representation of the framework and the theorems proofs. The theorems are intended to demonstrate that the algorithm is complete and consistent, i.e., it correctly creates low-level permissions that are defined in high-level policy without creating any security violation. Chapter 6 presents a scalability study, which demonstrates the time complexity of the algorithm theoretically and the simulations performed with the prototype in real scenarios.

Chapter 7 presents a complete case study, illustrating the specification of the three policies models, the steps of the processing and the resulting scripts created for firewall configuration. Finally, Chapter 8 concludes the thesis and points to future developments.

Chapter 2

Basic Concepts and Related Work

2.1 Introduction

Information systems security refers to protection of information systems against unauthorized access to or modification of information, whether in storage, processing or transit, and against denial of service to authorized users, including measures necessary to detect, document, and counter such threats.

In network scenarios, the security is enforced mainly by firewalls, which have been widely deployed to secure private networks in businesses and institutions. The firewalls are very common in the connection between the private network and the Internet, but are also employed inside the private network, protecting and separating the internal subnets and resources.

The firewalls enforce the protection by analyzing the packets that transverse them. By examining their contents, the firewalls decide if each packet must be accepted or discarded. This decision is normally made based on a sequence of rules, which are the configuration or policy of the firewall.

The major problem in defining the network policy is the correct definition of the rules that must be configured in each firewall in the network. It is relatively easy have conflicting rules in one firewall, and more easy when multiple firewalls need to be configured in order to implement the policy. Due to this, several works proposes mechanisms for management and configuration of firewall policies.

This chapter presents the traditional access control models that are used for policy definition and the frameworks and languages used for policy representation.

2.2 Access Control Models

Access control is the ability to permit or deny the use of a particular resource by a particular entity. Access control is the problem of determining the operations (e.g., read and write) that subjects (e.g., users and services) can perform on objects (e.g., files and network connections). A particular access control specification instance (or policy) is called a configuration [22].

Several models have been proposed to address the access control requirements of distributed applications. Traditional access control models are broadly categorized as discretionary access control (DAC) and mandatory access control (MAC) models. New models such as role-based access control (RBAC) or task based access control (TBAC) models have been proposed to address the security requirements of a wider range of applications.

The focus of the framework proposed in this framework is to allow the administrator to define the security policy using the discretionary and mandatory access control models. This section presents a description of these control models. For the most common access control models, please read [29] and [30].

2.2.1 Mandatory Access Control

The Trusted Computer System Evaluation Criteria [13], the seminal work on the subject which is often referred to as the "Orange Book", defines MAC as "a means of restricting access to objects based on the sensitivity (as represented by a label) of the information contained in the objects and the formal authorization (i.e., clearance) of subjects to access information of such sensitivity".

Mandatory Access Control (MAC) is the strictest of all levels of control. The design of MAC was defined, and is primarily used by the government. MAC takes a hierarchical approach to controlling access to resources. Under a MAC enforced environment, access to all resource objects (such as data files) is controlled by settings defined by the system administrator. As such, all access to resource objects is strictly controlled by the operating system based on system administrator configured settings. It is not possible under MAC enforcement for users to change the access control of a resource. Mandatory Access Control begins with security labels assigned to all resource objects on the system. These security labels contain two pieces of information - a classification (top secret, confidential etc) and a category (which is essentially an indication of the management level, department or project to which the object is available). Similarly, each user account on the system also has classification and category properties from the same set of

properties applied to the resource objects. When a user attempts to access a resource under Mandatory Access Control the operating system checks the user's classification and categories and compares them to the properties of the object's security label. If the user's credentials match the MAC security label properties of the object access is allowed. It is important to note that both the classification and categories must match. A user with top secret classification, for example, cannot access a resource if they are not also a member of one of the required categories for that object. Historically and traditionally, MAC has been closely associated with multi-level secure (MLS) systems.

The Bell-LaPadula (BLP) model describes a generic multilevel confidentiality policy [31]. The subjects of the model have security clearances, whereas the objects have classifications. Labels may indicate the different security levels, corresponding to military classifications. The system is secure if the set of state transitions preserves the following two rules:

(1) The simple security condition (also know as read-down), which states that a subject can read an object iff confidentiality $\text{level}_{\text{subject}} \geq$ confidentiality $\text{level}_{\text{object}}$, and the subject has a discretionary read access to the object.

(2) The *-property (also known as write-up), which states that a subject can write an object iff confidentiality $\text{level}_{\text{subject}} \leq$ confidentiality $\text{level}_{\text{object}}$, and the subject has a discretionary write access to the object.

Therefore, the BLP policy allows information flow from low-confidentiality level to higher levels and disallows flow in the opposite direction. The BLP model may be extended with compartments (also named categories or need-to-know), which are specified areas of interest. Examples are the set of departments of an organization or the subset of information two nations agree to exchange (see Figure 2.1 for an example). Thus, compartments reflect a need-to-know-policy and restrict the subjects' access to information at levels for which they are cleared. The set {UNCLASSIFIED, CONFIDENTIAL, SECRET, TOP SECRET} forms a linear ordering, whereas compartments lead to a lattice including the set of all subsets of the set of compartments, also called the power set of the set of compartments. The combination of the security levels and the compartments forms a partially ordered set (a partially ordered set differs from a total order in that some pairs of elements may not be related to each other). Solaris TE is an example of current implementation of the extended BLP model [56].

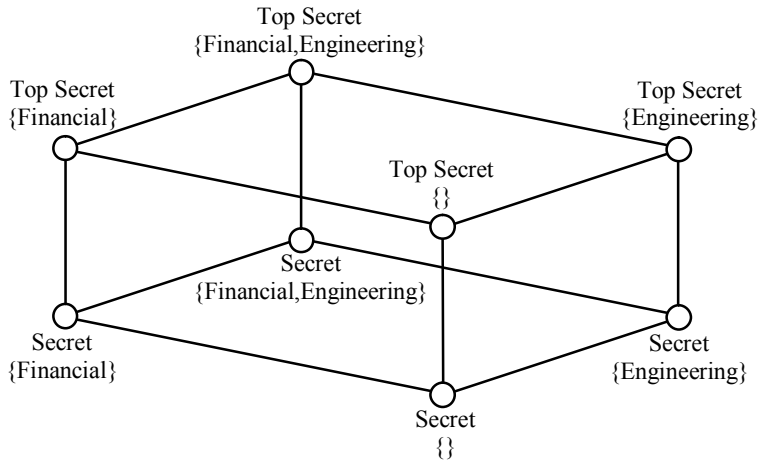

Figure 2.1: Example of Lattice

Denning [32] and Biba [33] also used lattices for information flow and integrity policies, respectively. The paper [34] gives detailed information about lattices and their applications.

2.2.2 Discretionary Access Control

In computer security, discretionary access control (DAC) is a kind of access control defined by the Trusted Computer System Evaluation Criteria [13] as "a means of restricting access to objects based on the identity of subjects and/or groups to which they belong. The controls are discretionary in the sense that a subject with certain access permission is capable of passing that permission (perhaps indirectly) on to any other subject (unless restrained by mandatory access control)".

Discretionary access control policies control the access of users to the objects on the basis of the user's identity and authorizations (or rules) that specify, for each user (or group of users) and for each object in the system, the access models (e.g., read, write or execute) the user is allowed on the object. Each request of a user to access an object is checked against the specified authorizations. If there is an authorization stating that the user can access the object in the specific mode, the access is granted, otherwise, it is denied.

The flexibility of discretionary policies makes them appropriate for a variety of systems and applications. For these reasons, they have been widely used in a variety of implementations, especially in the commercial and industrial environments [29].

Discretionary Access Control is most common access control mechanism for desktop operating systems. Instead of a security label in the case of MAC, each resource object on a DAC based system has an Access Control List (ACL) associated with it. An ACL contains a list of users and groups to which the user has permitted access together with the level of access for each user or group. For example, User A may provide read-only access on one of her files to User B, read and write access on the same file to User C and full control to any user belonging to Group 1. In operational systems and several other applications, each user can control the access to their own data. It is important to note that under DAC a user can only set access permissions for resources which they already own. A hypothetical User A cannot, therefore, change the access control for a file that is owned by User B. User A can, however, set access permissions on a file that she owns. Under some operating systems it is also possible for the system or network administrator to dictate which permissions users are allowed to set in the ACLs of their resources.

Discretionary access control provides a much more flexible environment than mandatory access control but also increases the risk that data will be made accessible to users that should not necessarily be given access.

Discretionary access controls policies based on explicitly specified authorizations are said to be closed, in that the default action is denial [29]. Similar policies, called open policies, could also be applied by specifying denials instead of permissions. In this case, for each user and each object of the system, the access modes the user is forbidden on the object are specified. Each access request by a user is checked against the specified (negative) authorizations and granted only if no authorizations denying the access exist. The use of positive and negative authorizations can be combined, allowing the specification of both the access to be authorized as well the access to be denied to the users. However, the interaction of positive and negative authorizations can become extremely complicated [57].

2.2.3 Role Based Access Control (RBAC)

The central notion of RBAC is that permissions are associated with roles, and users are assigned to appropriate roles. Roles permit the grouping of a set of permissions related to a position in an organization such as finance director or network operator. This allows permissions to be defined in terms of the position rather than the person assigned to the permission, so policies do not have to be changed when people are reassigned to different positions within the organization. Users

can be easily reassigned from one role to another and roles can be granted new permissions as new applications and systems are incorporated, and permissions can be revoked from roles as needed. Another motivation for RBAC has been to reuse role specification by a form of inheritance whereby one role (often a superior in the organization) can inherit the rights of another role and thus avoid the need to repeat the specification of permissions. One example of application of RBAC to firewall configuration is presented in [68].

Four conceptual models have been specified in an effort to standardize RBAC [41]. $RBAC_0$ contains users, roles, permissions and sessions. Permissions are attached to roles and users can be assigned to roles to assume those permissions. A user can establish a session to activate a subset of the roles to which the user is assigned. $RBAC_1$ includes $RBAC_0$ and introduces role hierarchies [42]. Hierarchies are a means of structuring roles to reflect an organization's lines of authority and responsibility, and are specified using inheritance between roles. Role inheritance enables reuse of permissions by allowing a senior role to inherit permissions from a junior role. For example the finance director of a company inherits the permissions of the accounts manager, as the latter plays the junior role. However, defining role hierarchies can be complicated and not represent correctly organizational hierarchy [67] (For example a managing director would not usually be able to perform the functions of a systems administrator much lower down in the organizational hierarchy). $RBAC_2$ includes $RBAC_0$ and introduces constraints to restrict the assignment of users or permissions to roles, or the activation of roles in sessions. Constraints are used to specify application-dependent conditions, and satisfy well-defined control principles such as the principles of least-privilege and separation of duties.

Finally, $RBAC_3$ combines both $RBAC_1$ and $RBAC_2$, and provides both role hierarchies and constraints. In the work [43] (adopted as an ANSI/INCITS standard in 2004), they propose an updated set of RBAC models in an effort to formalize RBAC. The models are called: flat RBAC, hierarchical RBAC, constrained RBAC and symmetrical RBAC, and correspond to the $RBAC_0$ – $RBAC_3$ models. Although the updated models define more precisely the basic features that must be implemented by an RBAC system, their description remains informal. A number of variations of RBAC models have been developed, and several proposals have been presented to extend the model with the notion of relationships between the roles [44], as well as with the idea of a team, to allow for team-based access control where a set of related roles belonging to a team are activated simultaneously [45].

The RBAC can be configured in order to enforce mandatory and/or discretionary access control policies. The works presented in [53], [54] and [55] demonstrates some approaches for this configuration.

2.3 Frameworks and Languages for Policy Representation

Several works discuss the representation, manipulation and enforcement of security policies. Some of them are very generic and can be applied to any domain of study, while others are specifically designed for network security.

The main objective of network security languages is to generate a set of rules that must be configured in a single firewall or a set of firewalls. Firewall rules normally contain a set of conditions and an action. The conditions are based on the a set of filtering fields such as protocol type, source and destination IP addresses and ports, and sometimes other packet information. The common actions are accept, which permits the forwarding of the packet, or deny, which drops the packet. The paper [49] provides a description of firewall rules.

This section presents the generic approaches for policy representation, and classifies the approaches that are specific for representation of network security policies.

2.3.1 Generic Approaches

Several works propose frameworks and languages designed to represent security policies generically. Although they aren't specifically designed for network security policies, they have important characteristics that must be considered.

XACML (eXtensible Access Control Mark-up Language) [28] is the OASIS standard language for the specification of access control policies. It is a general purpose access control policy language which defines a request/response language and framework to enforce authorization decisions. It is based on the XML language and was designed to express a large variety of policies, taking into account properties of subjects and protected objects as well as context information. The complete policy applicable to a particular decision request may be composed of a number of individual rules or policies. In general, a subject can request an action to be executed on a resource and the policy decides whether to deny or allow the execution of that action.

In a typical XACML framework, there is a policy enforcement point (PEP) and a policy decision point (PDP). The PEP is responsible for issuing requests and enforcing the access control decisions. The PDP receives requests from the PEP and evaluates policies applicable to the requests and sends a decision back to the PEP.

XACML policies include three main components: a Target, a Rule set and a Rule combining algorithm. The Target identifies the set of subjects, resources, actions and environments to which the policy is applicable. Each Rule in turn consists of another optional Target, a Condition and an Effect element. The Condition specifies restrictions on the attribute values in a request that must hold in order for the request to be permitted or denied as specified by the Effect. The Effect specifies whether the requested actions should be allowed (Permit) or denied (Deny). The Rule combining algorithm is used to resolve conflicts among applicable rules with different effects. An XACML policy may also contain one or more obligations, which represent functions to be executed in conjunction with the enforcement of an authorization decision. An XACML request consists of a list of attributes characterizing a subject and its environment along with the attributes of the action and resource.

The policy-based access control model described in [1] allows the decomposition of the policies for collaborative access control. It extends the XACML reference architecture by including the notion of collaborative access control, which means that several parties participate to make access control decisions.

The basic idea is that the global policy is obtained from the combination of policies defined by several participating parties, in a way that each party does not need to have any sensitive information belonging to other parties to make an access control decision. To make this possible, this system is compounded by one central policy enforcement point (PEP) and multiply policy decision points (PDP).

Although this access control model focus on the possibility of definition of security policy by more than one person or department, it is not designed for configuration of scenarios where several PEPs are present, considering that each firewall in a network is an enforcement point.

A multilevel security (MLS) is proposed in [8] to simplify traditional Lattices. It defines a structure with more security dimensions where each dimension is a simple linear ordered lattice. MLS describes an information system which is trusted to contain information classified into different security levels and to maintain separation between the levels. Concurrent users may have different permissions with respect to the security levels. The dimensions (named security requirements) used in this work are confidentiality, integrity and availability, which are handled

independently. This work doesn't provide the richness and granularity of traditional MLS-systems, since it is designed to adapt better to practical lightweight applications.

The proposal in [22] introduces the concept of access control spaces for the management of access control policies. An access control space represents the permission assignment state of a particular subject or role. The permissions are categorized into subspaces that have meaningful semantics. For example, the set of permissions explicitly assigned to a subject defines its specified subspace, and constraints define the prohibited subspace. An unknown subspace consists of all elements that are neither in the permissible nor in the prohibited spaces. This approach enables the administrator to find and resolve conflicts, since any overlapping subspace represents a conflict.

2.3.2 Network Specific Approaches

2.3.2.1 Vendor Dependency

The network specific approaches can be split in two main groups: vendor specific or vendor independent. The vendor specific languages are normally defined by firewall vendors, such as Cisco PIX [3] and Cisco IOS [4]. Although the INSPECT language is patented by CheckPoint [5], it is possible that different firewalls support it, since any vendor can create a firewall supporting the INSPECT standard by implementing a compiler that translates the INSPECT language into the firewall's native configuration instructions. However, it cannot be considered vendor independent.

These languages are designed to explore the complete range of functionalities of the firewalls. However, in large networks, probably there will be firewalls of different vendors and even of same vendor but different models. For these scenarios, it is more interesting to represent the security policies in a vendor independent language and compile these specifications for each model present in the network. The languages presented in the following are vendor independent, if not explicitly stated the opposite.

2.3.2.2 Topology Dependency

The vendor independent approach can be classified as topology dependent or independent. Topology dependence means that the location of users, resources and firewalls affects the policy, i.e., any change in the topology implies in changing the policy.

One example of topology dependence is presented in [6]. The language used in [6] represents firewall configuration as high-level policies based on the Ponder specification [7]. It employs a two-level optimization approach that allows software and hardware optimizations to proceed independently. This work includes a mechanism for conversion of high-level firewall description to low-level firewall rule representation using standard compilation techniques, and allows the implementation of the policy on reconfigurable hardware. Even though the high-level language provides the use of symbols for masquerading host and network addresses, it is still topology dependent, because there is no automated strategy for selecting a sub-set of rules that applies for a specific firewall. Another drawback is that, although the representation syntax for policies is the powerful Ponder language, which supports the generic definition of policies, the special policy types defined for this particular application are filtering rules-like. The proposal in [23] is another example of a topology dependent approach, since it bases the security in the location of the firewalls and servers and in the correct configuration of a DMZ network.

Topology independent approaches are more interesting for the administrators, since it is easy to describe the policy, and topology changes doesn't result in policy changes. In this case, a mechanism or algorithm is responsible for evaluating the network topology (i.e., the location of users and resources with respect to the firewalls) and translating the security policies into localized firewall configuration. The framework described in [21] and [25] are examples of topology independent firewall configuration. In [21], it is presented a policy-based framework for the automatic management of firewall rules in large networks. The access control policies are defined in three levels of abstraction: organizational, global and local. Policies at the organizational level are described in natural language, and define security goals such as blocking offensive content and scanning actions. Organizational policies are transformed into global filtering rules at the global level. The subset of the global rules that concerns each firewall is separated and distributed at the local level. The rule syntax used in all the three layers is a 3-tuple (source, destination, action), where source and destination are IP addresses (or ranges) and action is accept/deny. Consequently, the three levels correspond only to different topology levels, in which one is more general than the others but they do not have really different abstraction levels.

The project presented in [25] aims to automate the management of security policy in dynamic networks. The project focuses in simplifying the administrators' task by separating the policy from the topology description. One of the specific goals of this work is management of security configurations in networks that span multiple administrative domains. A paradigmatic

example is the situation of two connected firewalls, each of which has a local security policy (administered by a human perhaps). Even if each firewall correctly implements the local policy, the interconnection of the two firewalls may violate a global security policy that no firewall can detect by itself. The security policy, network topology, mechanism configuration and behavior are specified the proposed language named SPL (Security Policy Language), which is prolog-based. The central component is a policy engine with templates of the network elements and services that validates the policy and generates the new security configurations for the network elements when the security is violated. The important aspect of this project is the validation of the policies applied to the firewalls when multiple administrative domains are connected.

2.3.2.3 Abstraction Level

Policy languages can be classified according to their abstraction level. Low-level languages represent the policy as conditions-actions tuples (if conditions are satisfied, then enforce actions). Several languages are low-level, such as the language employed by the Firmato toolkit [12] and the language described in [24].

Firmato is a firewall management toolkit, based on entity-relation model. This model can be viewed as the application of a role-based model to the firewall policy area. Firmato support the separation of policy definition from network topology to increase the language modularity and reusability. It includes: (1) an entity relationship model containing, in a unified form, global knowledge of the security policy and topology, (2) a model definition language, which is used as an interface to define an instance of the entity-relationship model, (3) a model compiler, translating global knowledge of the model into firewall-specific configuration files, and (4) a graphical firewall rules illustrator. In Firmato, the user can only define allow rules; deny rules are supported only by the default rule: all traffic not explicit allowed will be denied.

The security policy modeling is executed in two phases. In the first phase, the modeler should define roles and describe which services the roles are permitted to use, regardless of its location on the network (i.e. a topology independent phase). In the second phase, the administrator specifies the assignments of roles to actual hosts. Named entities directly correspond to IP address ranges and hence physical and logical topologies are mutually intertwined.

In Firmato, the policy is defined as a list of rules, which are basically a set of condition-action's, and the concept of role is limited to a service or set of services that are initiated or accepted by the hosts (for instance, mail_client, mail_server, dns_server, etc.). However,

Firmato presents two aspects that must be pointed out: it considers the differences between distinct firewall models, through the representation of the features of each model; and the model compiler, which translates a model instance into firewall specific configuration files.

The language presented in [24] represents network policies based on paths. A path-based policy is defined as a policy were all attributes associated with the policy (traffic service type, conditions used to trigger the policy and the actions executed when the policy is triggered) are bound to a predefined path. The representation of a path, which includes every node from source to destination, is used to create virtual channels where resources are reserved to support real-time applications. This paper, however, doesn't describe how the policies are translated to firewall configuration files.

High-level languages uses more abstract concepts, wherein the security policies says "what must be done" instead of saying "how must be done". Some examples of policy based languages can be found in [9], [10] and [11]. The framework presented in [9] permits to represent high-level policies in the form of a list of data access rules (DACL), which declares permissions for executing simple operations (read or write) on objects. The framework translates the high-level policies into low-level policies suitable to be configured into the firewall devices. There is a subtle difference between the semantics of DACL and that of a low-level access-control policy. A DACL rule cannot simply be translated into some enforcement mechanism that controls access to the data. The security property specified by DACL is enforced collectively by the configurations of all those hosts. Thus, to verify that the high-level DACL policy is upheld, all configuration parameters in the network must be taken into account. The algorithm for checking the fidelity of the low-level policies with respect to the high-level policies is also presented. The FLIP language [10] aims automate the management of security policy in dynamic networks. The policies are represented as high-level service-oriented goals, which can be translated automatically into access control rules to be distributed to the firewalls. It generates rules that are conflict-free, both on individual firewalls and between firewalls, by restricting some conflicts at the high-level and handling the remaining conflicts during the processing.

The work presented in [11] introduces a Lisp-based language for representing global access control policies for configuring multiple distributed firewalls. The language abstracts hosts and area addresses by using names, which permits to easily determine which firewalls are traversed by the communication flows. It defines an algorithm that, given a specific topology, creates the filter set for each firewall or router. It also defines a second algorithm that verifies if the resulting configuration violates any of access policies. The authors use the expression "filtering

posture" to represent an assignment of filter functional behavior to each router interface in a network. It does not specify the router configuration files that will implement this functional behavior; it stipulates only the logical effects that those configuration files should achieve. An interesting aspect addressed in this work is defining how individual packet filters cooperate in order to achieve the global security policy. The authors called this problem "localization".

2.4 Security Properties

Most of the works presented earlier in this chapter are focused in the representation of permissions of users to access resources, explicitly specifying source and destination locations, but without any reference to the path used to perform the access. Some of these works allow the administrator to specify if some access is permitted or denied based on the protocol being used, but there isn't a classification of the security level provided by each protocol.

The concept of security properties are directly connected to the path that packet traverse from the source to destination and the security provided by the protocol used to perform the access.

The purpose of defining a policy based on security properties is to ensure that the packets cannot traverse some paths, or that just specific protocols be used when traversing some paths. The reason for blocking these packets traversing these paths can be because the packets have been originated from untrusted hosts (the framework consider that the source is the first node of the path) or the path contains untrusted networks or devices. Thus, the security property policy will guarantee that only acceptable paths will be used to perform an access.

The protocol used to perform the access affects the security. For example, consider the http and the https protocols. The http represents the content in plain text, and anyone in the path is able to intercept and read its content. On the other hand, https provides confidentiality, since the content is encrypted. Therefore, the protocol must be considered when evaluating the security of the accesses.

The work described in [14] adopts a graphic representation of security rules. It also defines the concept of security goals (e.g., top secret, mission critic, etc), which impose additional security properties that are required in order to access an object or perform a given access mode. At a lower level, a security goal is expressed in terms of a security requirement vector, which defines the minimum levels of properties such as confidentiality, integrity, availability and accountability. A security assumption vector defines the same properties assigned for the

principal and elements along the path between the principal and the resource. In order of an access to be granted, it is necessary that all properties of the security assumption vector satisfy the corresponding properties in the security requirement vector.

The works in [11] and [20] adopts a similar concept for security goals and security requirements, deciding if an access is permitted or denied analyzing the packets and the path that the packets traversed. These works focus on distributed packet filtering and IPsec gateways.

We have borrowed some concepts related to the security property model from [14].

2.5 Conclusion

There are several works that focus on management of firewall security policies. However, these works normally focus in specific issues concerning the policies, therefore they aren't generically.

In order to situate the framework that is being proposed, this chapter presented the traditional access control models and presented some related works classified by vendor dependency, topology dependency and abstraction level. It also presented some works that are related to the security property model used for policy definition in our framework.

The framework presented in this thesis can be classified as vendor and topology independent. The vendor independence is attained with the named service concept (please, see section 4.2 for details). All the computation is performed using an abstract representation, which is independent of devices. The framework includes a refinement algorithm responsible for generating the rules or scripts used for firewall configuration. There is a specific library for each different firewall, which receives abstract permissions and uses the corresponding named services as input to generate the rules/scripts. The topology independence is achieved isolating the security policies from the network topology description. The refinement algorithm is capable to generate and distribute firewall configuration, from the registered topology information. If there is any change in the topology, it is just necessary to reprocess the policies with the new topology description.

The policy language introduced can be classified as a high-level one. Although the discretionary policy is structured as conditions-actions tuples, the conditions are specified independently of the firewalls models and locations, and also independently of how they must be configured. Also, the mandatory policy and the security property policy allow the administrator to specify the security policy based on the declaration of security levels. The

policies represented by the language proposed in this work can be classified as closed, since they adopt the “anything not explicitly allowed is forbidden” strategy.

Chapter 3

Firewall Technology

3.1 Introduction

A firewall is a system or group of systems that enforces an access control policy between two networks. The actual means by which this is accomplished varies widely, but in principle, the firewall can be thought of as a pair of mechanisms: one which exists to block traffic, and the other which exists to permit traffic.

The purpose of a firewall is to protect the inside network and keep "bad" things outside a protected environment. To accomplish that, firewalls implement a security policy that is specifically designed to address what bad things might happen. For example, the policy might be to prevent any access from outside (while still allowing traffic to pass from the inside to the outside). Alternatively, the policy might permit accesses only from certain places, from certain users, or for certain activities. Part of the challenge of protecting a network with a firewall is determining which security policy meets the needs of the installation.

This section presents the main types of firewalls and describes how it works and the protection the can provide.

3.2 Packet Filter

A packet filter (also known as screening router or stateless inspection firewall) is the most basic type of firewall. Firewalls with packet filters normally operate at the network layer (see Figure 3.1), filtering the source and destination addresses and protocol type (such as TCP, UDP and ICMP), sometimes consulting fields of the transport layer, such as session source and destination

ports (e.g., TCP 80 for the destination port belonging to a Web server, TCP 1320 for the source port belonging to a personal computer accessing the server).

Before forwarding a packet, the firewall compares the IP header and TCP header against a user-defined table – rule base – which contains the rules that dictate whether the firewall should deny or permit packets to pass. The rules are scanned in sequential order until the packet filter finds a specific rule that matches the criteria specified in the packet-filtering rule. If the packet filter does not find a rule that matches the packet, then the default rule is executed. The default action can be accept or drop the packet, depending on what is the default behavior defined in security policy.

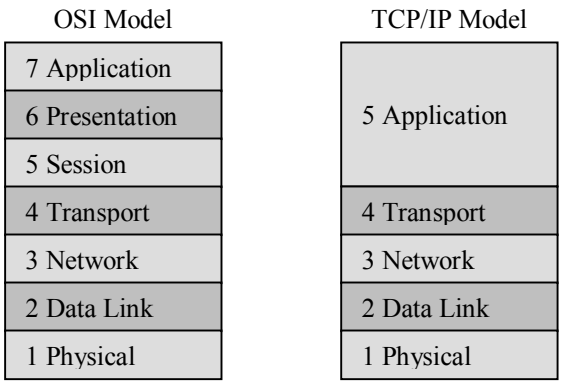

Figure 3.1: OSI and TCP/IP Models

Packet filters are not able to analyze the content of the packet (doesn't inspect higher layers). Thus, any detail in the packet's data field (for example, allowing certain Telnet commands while blocking other services) is beyond the capability of a packet filter.

The advantages of packet filter are speed and flexibility. Since packets seldom examine data above the network layer (with the possible exception of limited transport layer information), they are very fast. Likewise, since most modern network protocols can be accommodated using layer 3 or below, the packet filter can be used to secure nearly any type of network communication or protocol. On the other side, it has built-in limitations. Since packet filters don't examine upper layers, they are unable to prevent attacks that employ application-specific vulnerabilities or functions. They are also unable to detect spoofing attacks.

3.3 Stateful Inspection Filter

The stateful inspection (also known as dynamic packet filter) is the next step in the evolution of the static packet filter. It has the main characteristics and limitations of the static packet filter, but has an important difference: the state awareness, i.e., it keeps track of all packets associated with a specific communication session. Similar to packet filtering, stateful inspection intercepts packets at the network layer and inspects them to see if they are permitted by an existing firewall rule, but unlike packet filtering, stateful inspection keeps track of each connection in a state table. For this operation, the stateful inspection operates in OSI layer 4 (transport layer).

In simplest terms, a typical dynamic packet filter is able to differentiate packets of new connections from packets of established connections. Once a connection is established, the information about this connection is stored into a table that typically resides in RAM. Subsequent packets are compared to this table. When the packet is found to belong to an existing connection, it is allowed to pass without any further inspection. This also provides for the ability to create virtual sessions in order to track connectionless protocols such as UDP-based applications as well as RPC-based applications.

The evolution of stateful inspection is the deep packet inspection, which describes the capabilities of a firewall or an Intrusion Detection System (IDS) [58][59] to look within the application payload of a packet or traffic stream and make decisions on the significance of that data based on the content of that data. The engine that drives deep packet inspection typically includes a combination of signature-matching technology along with heuristic analysis of the data in order to determine the impact of that communication stream. This enables the identification of unexpected sequences of commands, such as issuing the same command repeatedly or issuing a command that was not preceded by another command on which it is dependent.

Deep Packet Inspection capable firewalls must not only maintain the state of the underlying network connection but also the state of the application utilizing that communication channel. For example, suppose the SMTP protocol [60]. The client establishes the SMTP connection by following the RFC defined protocol steps of issuing a *HELO*, waiting for the response by the mail server. The client may then issue a variety of commands include sending e-mail by specifying the SMTP command *MAIL FROM:*. The firewall monitoring the communication between the client and the mail server may raise an alarm or respond to the *VRFY* command by

disallowing it. The client may also try to exploit the sendmail [66] address token overflow (discussed in the CERT bulletin CA-2003-12 [74]) in order to gain shell access to the server. The firewall, because it is capable of Deep Packet Inspection, is able to identify the exploit attempt and deny the connection. Additionally, it may deny the connection from the client altogether.

3.4 Circuit Level Gateway

The circuit level gateway operates at OSI layer 5 (the session layer). In most situations, a circuit level gateway, also named relay host, is an extension of a packet filter in that it typically performs basic packet filter operations and then monitors TCP handshaking [62] between packets to determine whether a requested session is legitimate, analyzing the sequence numbers used in establishing the connection.

In this scheme, when a client wishes to connect to a server, it connects to a relay host (possibly supplying an authentication and/or other connection information). The connection to the server is done by the relay. Each successful connection attempt results in the creation of two separate connections – one between the client and the relay host, and another between the proxy server and the true destination. The name and IP address of the client normally is hidden from the server.

The circuit level gateway can also compare the IP and TCP headers against a rule table (as the packet filter does), to determine if the packet is accepted or denied. If none of the rule is applicable, the default action is executed (normally to drop the packet). If the packet is accepted by the rule table, then it checks if the session is legitimate verifying the SYN flags, ACK flags and sequence numbers involved in the TCP handshaking. The circuit level gateway can also perform logging and/or caching of the content.

3.5 Application-Proxy Gateway

An application proxy gateway (also know as Proxy Firewall) intercepts incoming and outgoing packets, working similarly to a circuit level gateway, preventing any direct connection between the client and the server. The main differences are: the application level gateways are application specific (i.e., they are only able to handle the services for which they were designed) and they examine the entire packet and can filter packets at the application layer of the OSI model (also examining the content of the packets).

For example, the HTTP traffic needs an HTTP proxy in order to work. If the application level gateway is the only connection device, incoming and outgoing packets cannot access services for which there is no proxy. For example, if an application level gateway ran FTP and HTTP proxies, only packets generated by these services could pass through the firewall. All other services would be blocked.

The application level gateway runs proxies that examine and filter individual packets, rather than simply copying and forwarding them across the gateway. Application proxies inspect each packet that passes through the gateway, verifying the contents of the packet up through the application layer (layer 7) of the OSI model. These proxies can filter on particular information or specific individual commands in the application protocols the proxies are designed to copy, forward and filter. As an example, an FTP application level gateway can filter on dozens of commands to allow a high degree of granularity on the permissions of specific users of the protected FTP service [63]. Other example is filtering electronic mail; it is possible to check for dirty words or blocking certain type of attachment (such as an executable file).

The advantages of application proxy gateways are many. As they prevent direct connections between two hosts and inspect the content of the communication, they offer a higher level of security. They usually have better logging capabilities, since they examine the entire packet. Another advantage is that the some application proxy gateways have the ability to decrypt packets (such as SSL – Secure Sockets Layer), inspect them and re-encrypt them before sending them to the destination. On the other side, the have disadvantages too. Performance is a critical issue, since the firewall spends more time reading and interpreting each packet. And, since it is necessary an application specific proxy for each type of network traffic, the application proxy gateways tend to be limited in terms of support for new networks applications and protocols.

3.6 Additional Functions

Besides the types of firewalls discussed in this chapter, the firewalls can also implements other functions, such as user authentication, logging and auditing, VPN, and NAT, among others.

Authentication services attempt to prove identity, to ensure that you know what person you are dealing with. Authentication services can be relatively simple when they're used on internal, trusted networks but are quite complex when they can't assume a basic level of trust. The firewall that implements user authentication enables the control of which users are allowed to have access.

Packets that are matched can be audited in a log or used to generate alerts. Normally, the log contains source and destination addresses, the protocol used, the service attempted, time and date, and the action carried out. Alerts can be used to run user-defined scripts to perform actions, such as triggering alarms and pager alerts, opening windows, and sending e-mail. Alerts can also be sent using SNMP to a SNMP Manager.

A virtual private network (VPN) [50] is a way of employing encryption and integrity protection (for example, using IPsec protocol [61]) in a way a public network can be used (for instance, the Internet) as if it were a private network (a piece of cabling that you control). Making a private, high-speed, long-distance connection between two sites is much more expensive than connecting the same two sites to a public high-speed network, but it's also much more secure. A virtual private network is an attempt to combine the advantages of a public network (it's cheap and widely available) with some of the advantages of a private network (it's secure). Fundamentally, all virtual private networks that run over the Internet employ the same principle: traffic is encrypted, integrity protected, and encapsulated into new packets, which are sent across the Internet to something that undoes the encapsulation, checks the integrity, and decrypts the traffic.

Network address translation (NAT) [51] allows a network to use one set of network addresses internally and a different set when dealing with external networks. It provides a mapping between internal IP addresses and officially assigned external addresses. NAT takes the IP address of an outgoing packet and translates it to an officially assigned global address. For incoming packets it translates the assigned address to an internal address. Network address translation does not, by itself, provide any security, but it helps to conceal the internal network layout and to force connections to go through a choke point (because connections to untranslated addresses will not work, and the choke point does the translation).

3.7 What Firewalls Cannot Do

It is important to understand that a firewall is a tool designed for enforcing security policy. If any access between the networks is not mediated by the firewall or the policy is ineffective, the firewall is not going to provide the adequate protection for the network. However, even a properly designed network with a properly configured firewall cannot protect the network from the following dangers.

A firewall, for instance, cannot decide if the communication is legitimate or not. Someone can use an authenticated Telnet session to perform an attack or create a tunnel to use an unauthorized protocol through another authorized protocol. Also, firewalls cannot protect connections that don't go through it. To perform the restrictions, it is necessary the existence of at least one firewall between the source and destination hosts.

Another problem is the social engineering. If someone obtains passwords they are not authorized to have or otherwise compromise authentication mechanisms trough social engineering mechanisms, the firewall won't stop them. Besides, a firewall is only as secure as the operating system on which it is installed. If the operational system has any bug, the protection enforce by the firewall can be compromised. This is why it is important to properly secure the operating system where the firewall are installed and keep it updated with the security patches.

3.8 Conclusion

This chapter presented the most common firewall types – packet filters, stateful inspection filters, circuit level gateways and application-proxy gateways – and their main characteristics.

Just installing a firewall in the network is not enough to protect the network. It is important to define the security policy and then correctly configure the firewall to reflect the security policy.

The framework proposed in this thesis allows the use of any model of firewall in the network that can be configured through the use of scripts or sequence of rules, since the named service can represent these scripts or rules independently of the firewall model/vendor. At this moment, there are library prototypes for INSPECT, IPTables and IPTables with L7-Filter [18].

For a more detailed description of firewalls technologies, please refer to [49] and [52].

Chapter 4

The Proposed Framework

4.1 Introduction

This chapter presents the proposal of a new policy based security framework for specification of network security policies, capable to handle simultaneously and coherently mandatory, discretionary and security property policies. The framework supports the definition of network security configuration for systems formed by a set of users willing to access a set of protected resources.

The framework has three main components: an information model, a high-level policy language, and a refinement algorithm. The information model and the language support the representation of the security information, whereas the refinement algorithm allows security policy formulated by high-level statements to be consistently translated to firewall security rules.

The main objectives of this framework are:

- Specification of policies using mandatory, discretionary and security property model;
- Definition of the a centralized global security policy, yet allowing the definition of each policy model by different groups of people;
- Independence of topology;
- Independence of firewall models and vendors;
- Specification of a mechanism for translating high-level security policy to low-level script/rule configuration files for each firewall in the network.

In order to achieve the independence of firewall model/vendor, it is used an abstract representation of the firewall protected services, denominated Named Service, which is explained in details in the following. The named service can also be relaxed, if the firewall doesn't implement it. This process is done without violation of the security, and will be explained in the description of the refinement algorithm.

4.2 Named Services

An important characteristic of the proposed model is that it isolates the low level security configuration from the high level policy model. The security of network services is actually implemented as a set of vendor dependent firewall rules (or scripts) configured along the firewalls in the network. On networks with different firewall models and vendors, it is necessary to use different languages for configuring each firewall. In order to isolate the firewall policy from the network elements, this vendor dependent configuration is encapsulated to the Named Service class.

A named service models a "high-level" secured network service that is supposed to be configured in the firewalls along the network, i.e., it is an abstract representation for the firewall protection services. It encapsulates the service configuration details that would impact the security permissions, modeling the firewall ability to perform some processing over the network packets. A named service can represent a broad range of secured services, from simple services like e-mail and web, to more complex services like file sharing and voice communications. The named service implicitly includes the protocols, client and server ports and any other necessary information to provide the secured service. In one network, each firewall can have its own set of named services, according to its characteristics or features. This concept is similar to the concept of feature introduced in Firmato [12], but allows the representation of any characteristic of underlying firewalls.

The use of named services is an important characteristic of this model, because it isolates the low level security configuration from the high level policy model. The inventory includes a set of configuration Libraries that are responsible for converting the high-level named service to the specific scripts/rules that must be used to configure the firewall. The corresponding configuration library implementing the named services in the corresponding vendor language is pointed by the Library attribute in the Firewall class. Observe that each firewall model will have its own library, allowing the configuration of any firewall model. The security of network

services is actually implemented as a set of vendor dependent firewall rules (or scripts) configured along the firewalls in the network. In this way, it is possible to configure any imagined named service, given that it can be supported by the corresponding firewall, and given that the configuration library exists. This allows splitting the high-level vendor independent policy configuration from the technical vendor dependent firewall configuration. Policy security administrators can define the high-level security policy, independently of vendors and specific firewalls models, leaving to network security administrators the task to configure the named services scripts in the appropriate libraries. Based on the information registered in the Named Service class, the refinement algorithm is able to configure firewall of different models and characteristics.

It is important to observe that not all firewalls have the same ability to protect the network resources. To address this issue, we introduce the concept of relaxation. For each firewall, we assume that it can implement at least the Generic Named Service, which is a rule or script that configures the firewall just to enforce the source and destination addresses of the packets (i.e. the firewall is at least a packet filter). A firewall that doesn't implement the named service related to some required access must be configured with this Generic named service. Note that this process doesn't cause violation of global security policy. The relaxation procedure is possible because the packets of a particular service access will cross all the firewalls along the corresponding path. If at least one of the firewalls blocks the offending packets, the access is protected. As the refinement algorithm will select only the feasible service accesses (i.e. at least one firewall in the path is enforcing the named service), it is assured that only connections permitted by the high-level policy will transverse the path. Therefore, the firewalls that don't implement any of the named service must permit the packet flow, validating only the source and destination.

4.3 The Information Model

The Information Model contains the objects and relationships that are used to represent the network and the security policy. It is organized in four main blocks: the Inventory, the Mandatory Model, the Discretionary Model, and the Security Property Model.

Although users interact with services, permissions are granted to resources. A Resource consists of one or more services (represented by named service class) delivered from one or more locations. For example, the E-mail resource could be defined to represent the SMTP, POP3 and IMAP named services, while the File Transfer resource could be defined as FTP named

service. A Location is the place from where users access resources and also the place where a resource is located. Physically it corresponds to a host or subnet, and is represented by an IP address and a mask. When assigned to users, a location plays the source role, and the destination role when assigned to resources. The User Located At and Resource Located At classes indicate, respectively, the locations from where a user can initiate an access or from where a resource can deliver a service. Users, locations and resources can be organized in groups, what is modeled by a corresponding abstract class. For example, an Abstract User can be a User or a User Group, which in its turn can contains many abstract users, that is, many users or user groups.

4.3.1 The Inventory Information Model

The Inventory information model contains the information necessary to define the policies that must be applied to the network. Figure 4.1 presents the class diagram of the inventory. It contains the users and resources that are present in the network, the locations where users and resources are located and the connections between them, the named services assigned to the corresponding resources and the firewalls present in the network. Users, resources, locations and named services can be grouped, in order to simplify the policy definition. The class User Located At connects a user or group of users (through the base class Abstract User) to a location or group of locations (through the Abstract Location class). The resources are compounded by one or more named services. Similarly to users, the resources are connected to the locations thought the Resource Located At class. This model allows the high granularity in specifying the location of users and resources, without preventing the use of groups to facilitate de configuration.

The Inventory also includes the representation of the named services implemented by the firewalls and the connections between locations and firewalls or between two firewalls. All named services available in the network are registered in the Named Service class, and each firewall is associated to the Named Services it implements. Even when there are firewalls that doesn't implement one or more required named services, it is possible to safely configure the set of firewalls involved in a secured access through the relaxation procedure. Such procedure is responsible to select the firewall rules (or scripts) for each firewall along a service access according to its capabilities, without violating the high-level security policy. The Connection, Location and Firewall classes model the network graph: the locations (subnets) and firewalls are the vertices and the connections are the edges. The topology is fundamental for computing the

permissions granted in the Security Property model. It is used to compute the paths from source to destination locations. Path computing is important because it affects the security property, as will be described in the following.

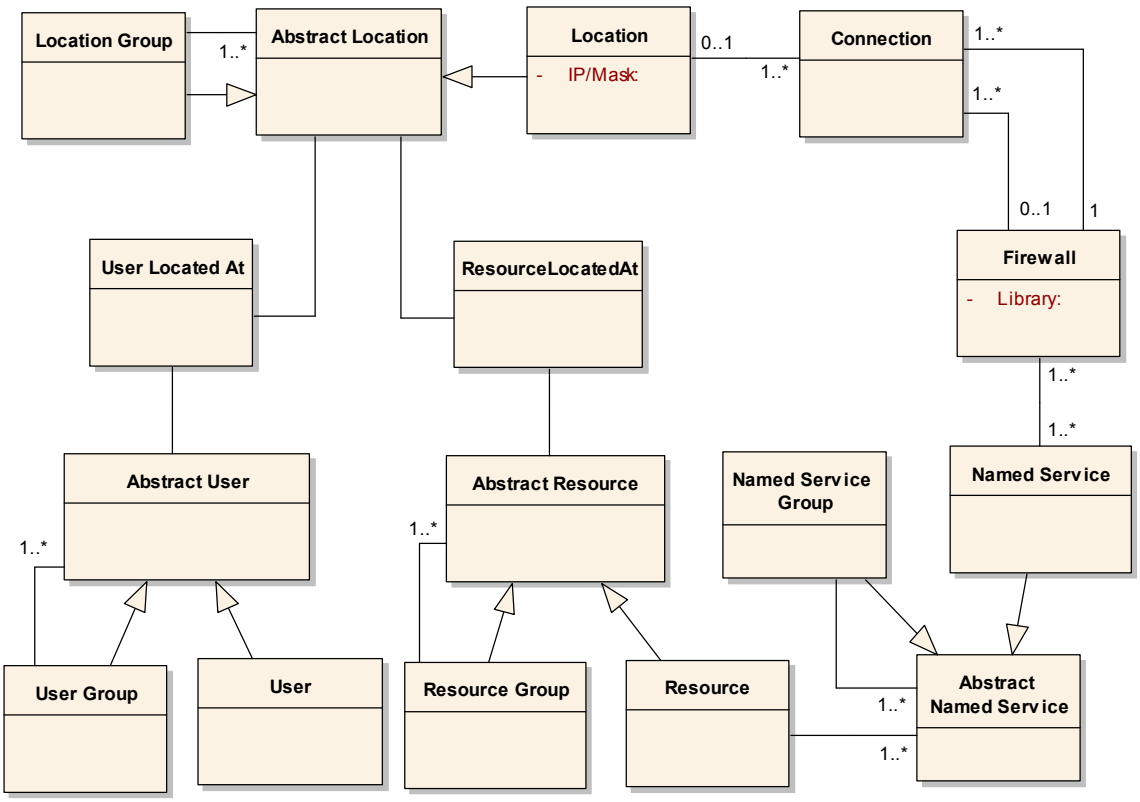

Figure 4.1: The Inventory Information Model

4.3.2 The Mandatory Information Model

The Mandatory Model (presented in Figure 4.2) establishes the mandatory access control policy by determining resource access according to a clearance versus classification and need-to-know schema. Clearance levels are assigned to users while classification levels are assigned to resources and compartments (need-to-know) are assigned to both users and resources.

The security levels UNCLASSIFIED, CONFIDENTIAL, SECRET, TOP SECRET are represented as integer numbers from 1 to 4, 1 corresponding to UNCLASSIFIED (the lowest level) and 4 corresponding to TOP SECRET (the highest level). The compartments can specify, for example, areas of interest, such as departments of an organization or projects which users are

on. For example, a resource can be associated to Financial compartment, meaning that only users associated to this compartment are allowed to access this resource.

A user is allowed to access a resource only if its clearance is greater than or equal to the corresponding resource classification and if the set of users' compartments is a superset of resources' compartments.

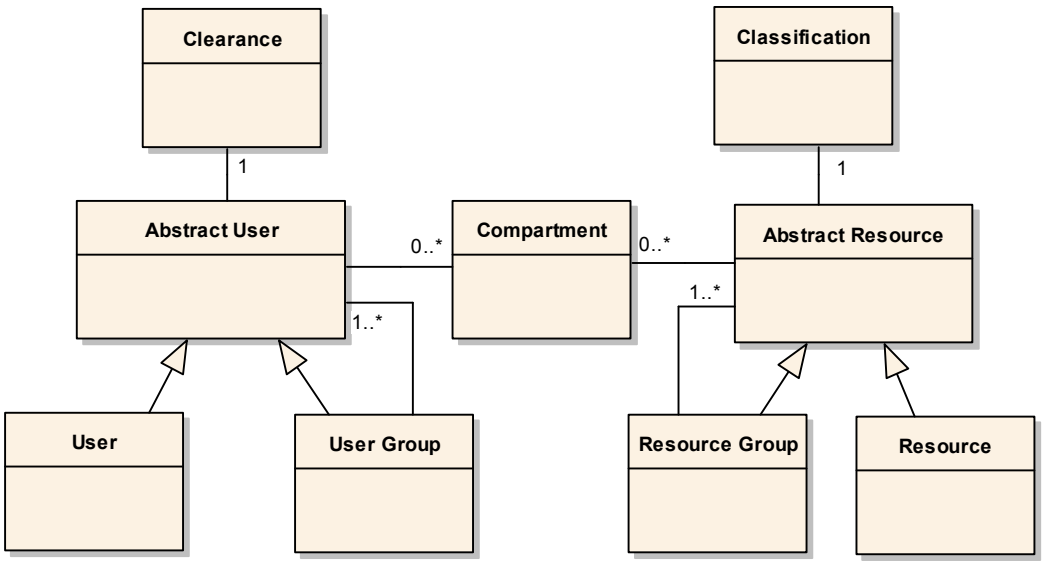

Figure 4.2: Mandatory Information Model

Figure 4.3 illustrates the concept of the mandatory model. Suppose three different users, User 1, User 2 and User 3 and a resource. User 1 has clearance 3 and is associated to Financial and Engineering compartments, User 2 has clearance 1 and is associated to Financial and Engineering compartments, and User 3 has clearance 4 and is associated to Engineering compartment. The resource has classification 2 and is associated to Financial compartment. In this scenario, only User 1 is allowed to access the resource, since it has the clearance greater than the resource's classification and has the necessary compartment (Financial). User 2 isn't allowed to access the resource, since its clearance is lower than resource's classification. User 3 isn't allowed to access the resource, since it isn't associated to the Financial compartment, which is necessary to access the resource.

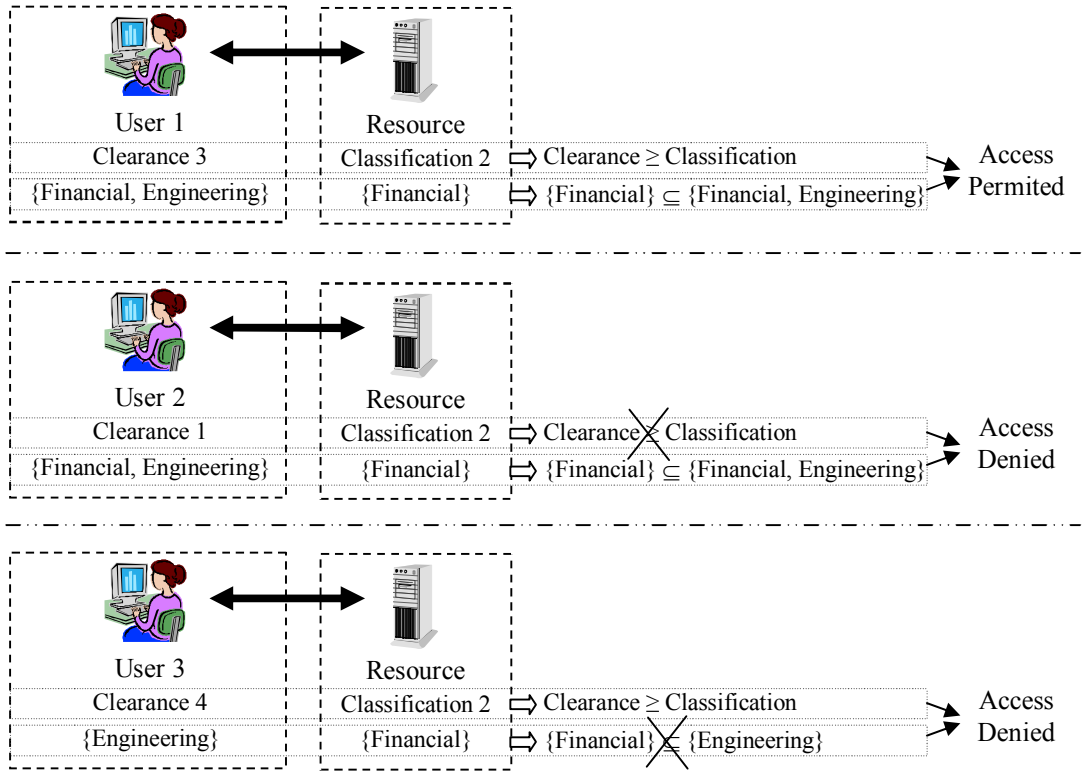

Figure 4.3: Example of Mandatory Levels and Compartments

4.3.3 The Discretionary Information Model

The Discretionary Model (Figure 4.4) is constructed by a set of discretionary rules that state the security actions to be enforced for specific service accesses. Each rule specify a set of users and a set of locations from where they can initiate an access; a set or resources and a set of destinations where they can be located; and an action. Conceptually, a (user, source) pair is considered the client and a (resource, destination) pair is considered the server for client-server services, and both sides are considered client and server for P2P services. Therefore, one discretionary rule represents the communication flow in both directions, and firewalls will be configured to permit packet flow from client to server and from server to client or between two peers. Examples of security actions are: accept, deny, log, and forward. In this thesis we just consider the accept action, and adopt the "anything not explicitly allowed is forbidden" strategy.

The discretionary rule allows the user of *any* keyword, allowing the representation of rules like "Users included in Engineering group from any location are allowed to access the E-mail resource at DMZ location", which means that the users can be located at any host. Another

example is: "Users included in the Internal User group are allowed to access any resource at the Internet from the Internal Network".

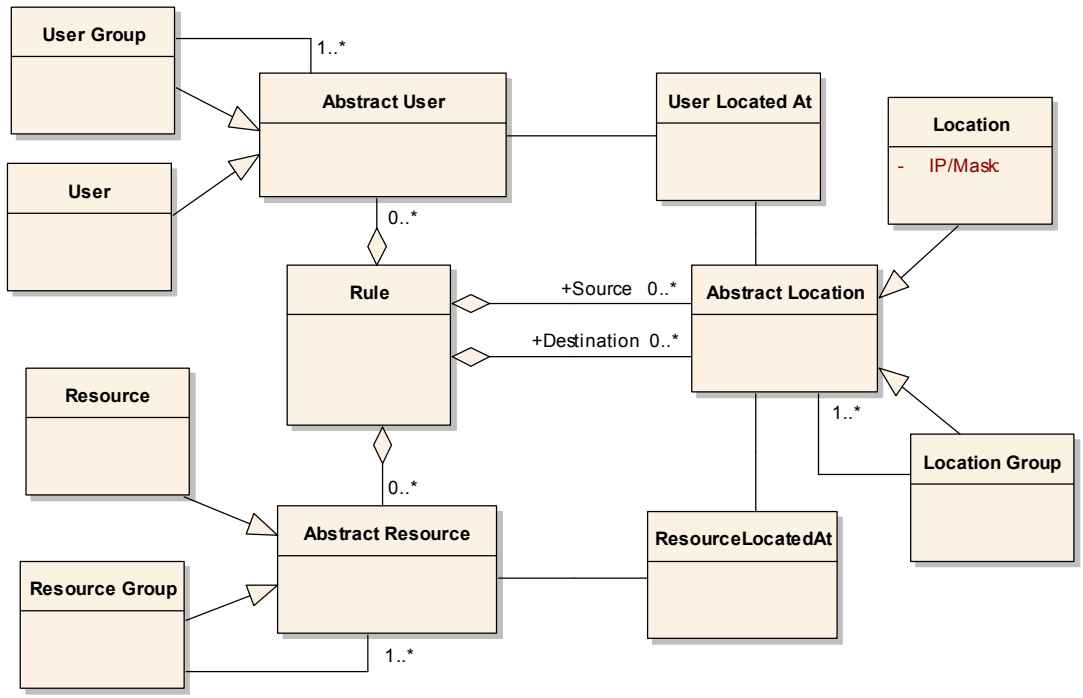

Figure 4.4: Discretionary Information Model

4.3.4 The Security Property Information Model

The Security Property Model provides fine-grained security information by including configuration and location dependent constraints. Security property rules enforce access control based on two properties, the security requirement (SR), which is defined for resources, and security assumption (SA), which is defined for the named services and the elements along the path including source location, intermediate locations and firewalls.

The security property concept is directly related to the path that packet traverse from source to destination. The purpose of defining a policy based on security properties is to ensure that the packets cannot traverse some paths, which are not safe enough. The reason for blocking the packets traversing these paths can be because the packets have been originated from untrusted hosts (the framework consider that the source is the first node of the path) or the path contains untrusted networks or devices. Thus, the security property policy will guarantee that only trustable paths will be used to perform an access.

As an example, suppose the network presented at Figure 4.5. In this diagram, DMZ, Finance and Engineering represent internal networks and External represents the Internet. The two firewalls, FW1 and FW2 connect the internal networks and external network. For this example, suppose that the Engineering and Finance networks must have access to Internet and to DMZ, and the External network is only allowed to access DMZ. In order to implement this policy using Security Properties, both External and DMZ networks should be associated to low security assumptions – since External is an untrusted and unsafe network and DMZ is unsafe, because it can be attacked from Internet. Finance and Engineering networks can have higher security assumptions – since they are inside the company and their users are known. It is also necessary associate high security requirements to resources located at Finance and Engineering and low security requirements to resources located at DMZ and Internet. With this configuration, FW1 will be configured to permit access to DMZ from External, Finance and Engineering networks, permit access to Internet from Finance, Engineering and DMZ and block any access to Finance and Engineering from External and DMZ networks. FW2 will be configured to permit access to Engineering from Finance and vice-versa, but will block any connections to Engineering from External and DMZ.

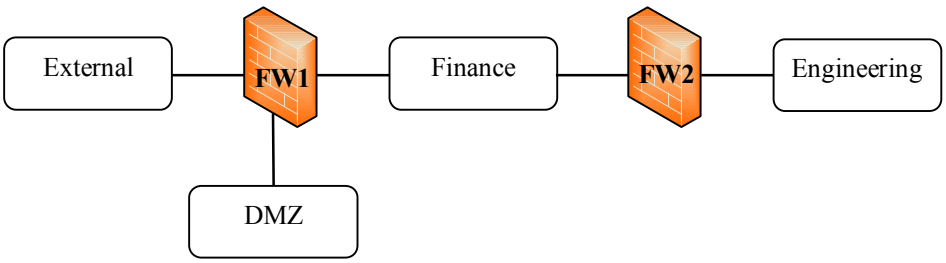

Figure 4.5: Example Network

Figure 4.6 presents the class diagram for Security Property Information Model. Security requirements and security assumptions are specified by security levels within a security class. The security level is a natural number ranging from one to four, expressing the "strength" of one of the following security properties: confidentiality, integrity, availability and accountability. The security level establishes a total order over the security property set: the greater is the corresponding number (1, 2, 3 or 4) the stronger is security level. A security class is defined by an array of size four. If *sc* is a security class, *sc*[1], *sc*[2], *sc*[3] and *sc*[4] correspond, respectively, to its confidentiality, integrity, availability and accountability security levels. These categories comprise a representative subset of the most common functional security

requirements as stated in standards like [64] and [65], but are not intended to be exhaustive. Any entity that can be involved in accessing a resource (i.e., locations, firewalls, and named services) has a security class.

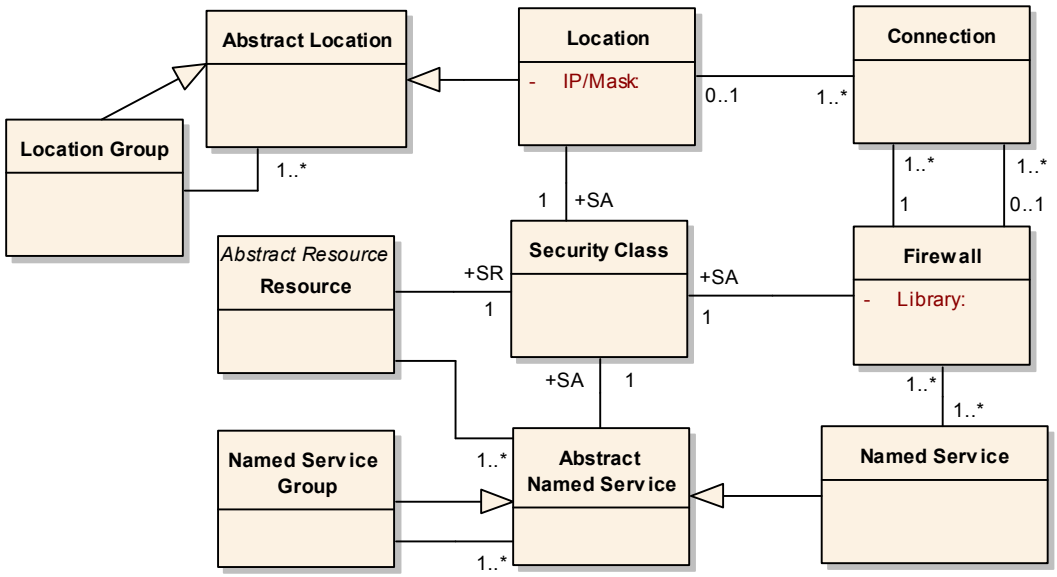

Figure 4.6: Security Property Information Model

Security requirements and security assumptions can change when the related objects are combined. For example, John Doe trying to access a resource from the Engineering subnet would probably have a different SA than when he is trying to access the same resource from JD-Home host. Assuming that the corporation has much more control over the Engineering subnet, the first combination should result in a stronger SA. The involved named service also changes the SA. For example, John Doe at Engineering subnet accessing a resource through a named service that implements HTTPS protocol introduces a lower risk than when he is trying to access the same resource from the same subnet through a named service that implements HTTP. Consequently a stronger SA should be assigned to the first. The modified security assumption is referred as effective security assumption (ESA). The upper effective class operation ($\cup$) over the security class set is defined to compute it. Let sc_1, sc_2, …, sc_n, to be security classes such that $sc_i = [x_{1i}, x_{2i}, x_{3i}, x_{4i}]$.

$$\bigcup_{i=1}^{n} sc_i = [\sup_i(x_{1i}), \sup_i(x_{2i}), \sup_i(x_{3i}), \sup_i(x_{4i})] \quad (1)$$

Security assumptions along an end-to-end path are combined together to form the overall security assumption (OSA). The permission to a user (from a source location) willing access a

resource (at a destination location) is granted only if the service access OSA is at least as "strong" as the resource SR. This involves the comparison of security classes. Because there is a partial order over the security class set, we are able to define its "strength". When sc_1 and sc_2 are security classes such that $sc_1 = [x_1, x_2, x_3, x_4]$ and $sc_2 = [y_1, y_2, y_3, y_4]$, the security class sc_2 is stronger than sc_1 if $y_i \geq x_i$, for $i = 1..4$.

The OSA is calculated as follows: First, compute the ESA for the (source location, named service) pair and for each (firewall, named service) and (location, named service) pairs along the path. Note that the destination is not included in this computation, since it is where the resource is located. Then, the resulting ESAs are combined along the end-to-end path. In this case the calculation should retain the set of weakest security levels. For this, the lower effective class operation ($\cap$) is defined over the security class set as follows. Let $sc_1, sc_2, \ldots, sc_n$, to be security classes such that $sc_i = [x_{1i}, x_{2i}, x_{3i}, x_{4i}]$.

$$\bigcap_{i=1}^{n} sc_i = \left[\inf_i(x_{1i}), \inf_i(x_{2i}), \inf_i(x_{3i}), \inf_i(x_{4i})\right] \tag{2}$$

When the OSA is at least as "strong" as the resource SR, the access is permitted.

Figure 4.7 illustrates the concept of security properties for two named services: http and https. For sake of simplicity, consider the security class as a vector of two elements: confidentiality and accountability. For the http named service, the ESA for the pair (H1, http) results in [1,3], and for the pair (FW, http) results in [4,1]. The computation of OSA for this situation results in [1,3], and, since it is not greater or equal to SR [3,3], the access is denied. For the https named service, the access is permitted: ESA(H1, https) results in [4,3] and ESA(FW, https) results in [4,4]; so the OSA for this situation is [4,3], which is greater than the SR of the resource.

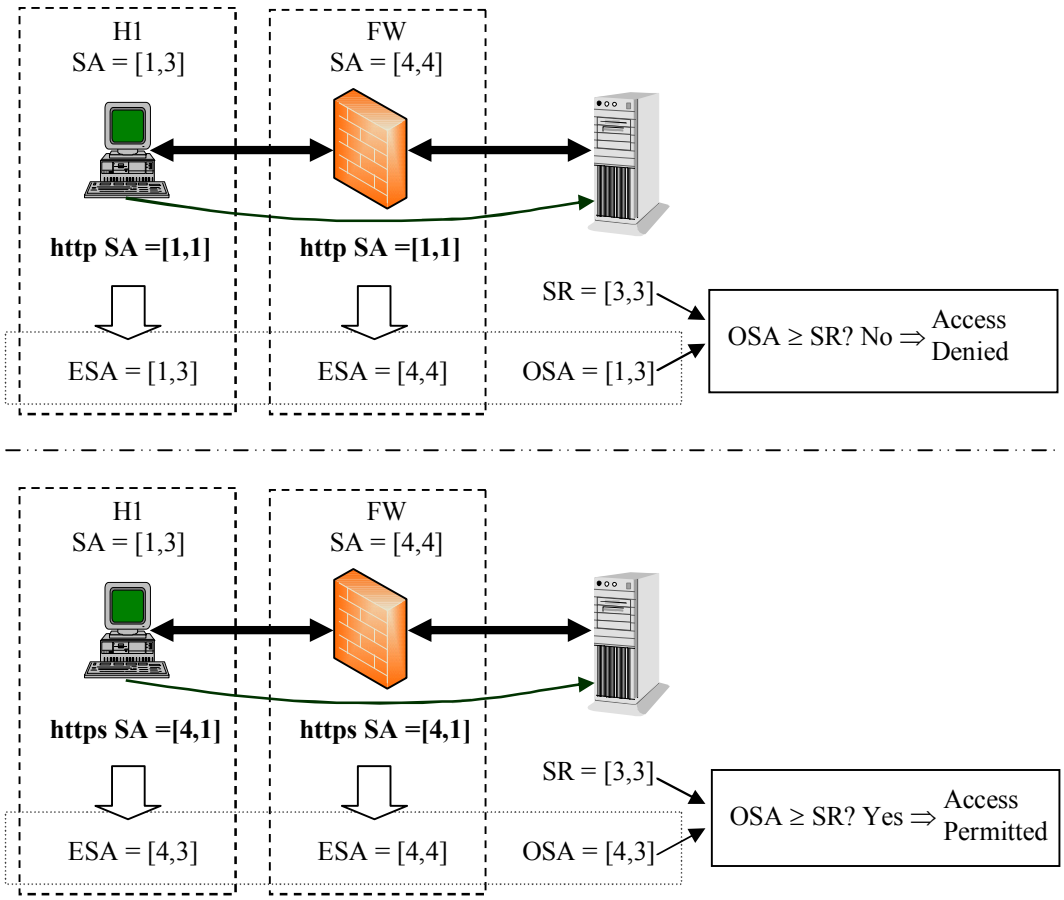

Figure 4.7: Example of ESA and OSA for http and https named services

4.4 The Proposed Language for Policy Representation

We propose a language for the representation of the three security policy models. The language is represented in text style, and is based on the syntax of Prolog Language.

Figure 4.8 presents the syntax for the elements of the inventory. The text that is emphasized in bold corresponds to the language primitives. The user john doe will be represented as *user(JohnDoe)*. A resource will be represented as *resource(email, [smtp, pop3, imap])*, where *email* is the resource name and *[smtp, pop3, imap]* are the associated named services. Please, see Chapter 7 for a complete example of the language use.

<inventory-item>: **user(**<user >**).** ||
user_located_at(<user>, **[**<location>$^{+}$**]).** ||
user_group(<user-group>, **[**<user>$^{+}$**]).** ||
resource(<resource >, **[**<named-service>$^{+}$**]).** ||
resource_located_at(<resource>, **[**<location>$^{+}$**]).** ||
resource_group(<resource-group >, **[**<resource>$^{+}$**]).** ||
location(<location >, <IP address/mask>**).** ||
location_group(<location-group>, <IP address/mask>, <location>$^{+}$**).** ||
firewall(<firewall >, **[**<interface>$^{+}$**]).** ||
connected(<location>**,** <location>**,** cost**).** ||
connected(<location>**,** <firewall>**,** <interface>, cost**).** ||
named_service(<named-service>**).** ||
named_service_group(<named-service-group >, **[**<named-service>$^{+}$**])** ||
firewall_library(<firewall>, <library>**).** ||
named_service_library(<firewall>, **[**<named-service>$^{+}$**]).**

Figure 4.8: Inventory Syntax

Figure 4.9 presents the syntax for mandatory model. For example, the user John Doe has clearance level 3 is represented as *clearance(John Doe, 3)*, the Engineering Group has clearance level 4 is represented as *clearance(Engineering, 4)*; on the other side, the resource Curitiba Server has the classification level 2 is represented as *classification(Curitiba Server, 2)*.

<mandatory-rule>: **clearance(**<user> || <user-group>**,** <clearance-level>**, [<**compartment>$^{+}$**]).** ||
classification(<resource> || <resource-group>**,** <classification-level>**,**
[<compartment>$^{+}$**]).**
<clearance-level>: <security-level>
<classification-level>: <security-level>
<security-level>: “1” || “2” || “3” || “4”

Figure 4.9: Mandatory Syntax

Figure 4.10 presents the syntax for the discretionary model. Note that it is possible to include in each rule set of users, resources, sources and destinations, or a set of groups of them. For example, the rule “Users included in Engineering group from any location are allowed to access the E-mail resource at DMZ location” is represented by *rule([Engineering], [any], [Email], [DMZ], permit)*. Another example is: “Users included in the Internal User group from the Internal Network are allowed to access anything at the Internet”, which is represented by *rule([Internal User], [Internal Network], [any], [Internet], permit)*.

<discretionary-rule>: **rule([** <user>$^{+}$ || <user-group>$^{+}$ || **any],**
[<source-location>$^{+}$ || <source-group>$^{+}$ || **any],**
[<resource>$^{+}$ || <resource-group>$^{+}$ || **any],**
[<destination-location>$^{+}$ || <destination-group>$^{+}$ || **any],**
<action>**).**
<action>: **permit**

Figure 4.10: Discretionary Syntax

Figure 4.11 presents the syntax for security property model. Both security assumption and security requirement use the Security Class definition, which is a vector of security levels where each security level represents the confidentiality, integrity, availability and accountability levels. For example, *security_requirement(Email, [4,2,3,1])* represents that the requirements are 4 for confidentiality, 2 for integrity, 3 for availability and 1 for accountability, and *security_assumption(host1, [1,2,3,4])* represents that host 1 provides 1 for confidentiality, 2 for integrity, 3 for availability and 4 to accountability.

<security-requirement>: **security_requirement(**<resource>, <security-class>**).**
<security-assumption>: **security_assumption(**<access-related-object>, <security-class>**).**
<access-related-object>: <location> || <firewall> || <named-service>
<security-class>: **[** <confidentiality-security-level>**,**
<integrity-security-level>**,**
< availability-security-level>**,**
<accountability-security-level> **]**
<confidentiality-security-level>: <security-level>
<integrity-security-level>: <security-level>
<availability-security-level>: <security-level>
<accountability security-level>: <security-level>
<security-level>: “1” || “2” || “3” || “4”

Figure 4.11: Security Property Syntax

4.5 The Refinement Algorithm

The translation process is decomposed in five refinement layers: location dependent, path dependent, device dependent and firewall dependent layer, as depicted in Figure 4.12. The box at the left represents the Inventory, and shows the information that is used by refinement at each layer. The topmost layer represents the security policies. Each layer below represents a transformation performed over the rules and each translation correspond to one or more steps in the algorithm.

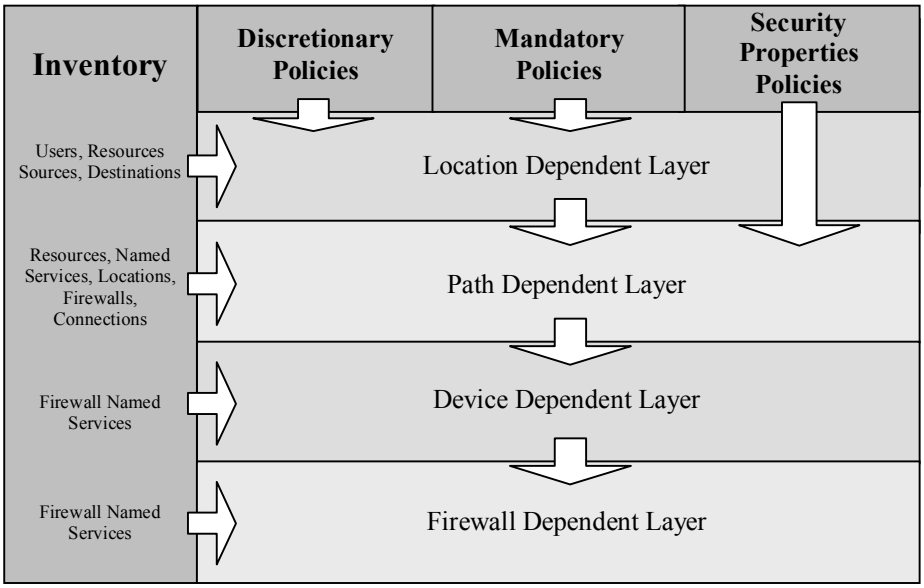

Figure 4.12: Layer Representation

For each rule, the algorithm performs the following steps:

1. Identify the set of users and sources: this step is the first part of the computing of the discretionary model. The result of this step is a set of users, and locations where these users can be located, in accordance to both the rule and the locations of the users specified in the inventory. The sub-steps are:

   i. Expand user groups and source groups to individual users and sources.
   ii. Select the users specified by the rule that can be located at the sources specified by the rule.
   iii. Select the sources that are locations where the users specified by the rule can be located.

2. Identify the sets of resources and destinations: this step is the second part of the computing of the discretionary model. The result of this step is a set of resources, and locations where these resources can be located, in accordance to both the rule and the locations of the resources specified in the inventory. The sub-steps are:

i. Expand resource groups and destination groups to individual resources and destinations.
ii. Select the resources that can be located at the destinations specified by the rule.
iii. Select the destinations that are locations where the resources specified by the rule can be located.

3. Considering the users and resources obtained in steps 1 and 2, select the pairs (user, resource), for which the user clearance is greater than or equal to the resource classification and compartments of the resource are a subset of the compartments of the user.

   This step is responsible to enforce the Mandatory Model. The pairs (user, resource) resulting from this step holds the mandatory model, and, due to steps 1 and 2, also holds the discretionary model. At this point, if there is no pair (user, resource) resulting from this step, the computation of the actual rule is stopped, and the next rule will be processed.
   The steps 1, 2 and 3 correspond to the Location Dependent Layer.

4. Compute the set of (source, destination, path) triples referred by this rule: this step is a preparation for the computation of the Security Property Model, which needs the source, the destination and all the network elements between them (path). Note that this step uses only the sources and destinations that were obtained in steps 1 and 2, therefore, the discretionary model is already enforced in the resulting tuples. The sub-step performed are:

   i. Compute the Cartesian product between the set of sources and the set of destinations selected in steps 1 and 2.
   ii. For each obtained (source, destination) pair, compute the lowest cost path between source and destination using the Dijkstra's shortest path algorithm [17].

5. Compute the set of possible service access. This set of possible service access includes the service access that holds the three policy models. From each (user,

resource) pair obtained in step 3 and from each (source, destination, path) triple obtained in step 4, perform the following sub-steps:

i. Identify the set of possible named services for the path, that is, the set of named services delivered at the corresponding destination.
ii. For each possible named service, use the path from the triple to compute the corresponding OSA. If it is at least as stronger as the resource SR, include the service access in the set of possible service access.

If the set of possible service access is empty, the processing of the actual rule is stopped, and the next rule will be processed. The steps 4 and 5 correspond to Path Dependent Layer.

6. Compute the set of feasible service access from the set of possible service access computed in step 5. This step verifies if there is one or more firewalls present in the attached path capable to implement the required named services. This verification uses the information stored in Named Service class. It is necessary that at least one firewall in the path implements the named service.
   The collection of possible service accesses that respect this condition will form the feasible service access set. Note that the feasible service accesses still refer to paths and not to individual firewalls.
   The following sub-step is performed:

   i. For each possible service access, verify if at least one firewall in the attached path implements the named service being referred. If yes, include the possible service access in the set of feasible service access.
   ii. Verify if the firewalls that don't implement the named service implement the Generic named service. If at least one firewall implements neither the required named service nor the Generic named service, then this path can't be used and will not be present in the resulting set of this step.

   If the set of feasible service access is empty, terminate the processing of this rule.
   This step corresponds to the Device Dependent Layer.

7. Determine the collection of firewall permissions for each firewall included in the feasible service accesses obtained in step 7. A firewall permission is an abstract representation of one rule or script that must be configured in the firewall, and has the format (firewall, source, destination, named service). The sub-steps performed at this step are:

   i. For each feasible service access, take the path associated to it.
   ii. For each firewall in this path, include or update optimized firewall permission. An optimized firewall permission is a firewall permission that references one or more sources, one or more destinations and one or more named services and has the following format (firewall, source^{+}, destination^{+}, named service^{+}). If the firewall implements the named service, then the optimized firewall permission will reference this named service; otherwise, the Generic named service will be referenced.

   At this step, the relaxation procedure is applied if necessary. A firewall that doesn't implement any of the named services related to the feasible access must configure a special service for the source destination pair, the Generic named service. The firewall script implemented in the Generic named service just enforces the source and destination of the packets. The relaxation procedure is possible because the packets of a particular service access will cross all the firewalls along the corresponding path. If at least one of the firewalls blocks the offending packets, the access is protected. As the refinement algorithm will select only the feasible service accesses, it is assured that the required named service will be present in path. Therefore, the firewalls that don't implement any of the named service must permit the packet flow, validating only the source and destination. This step also includes an optimization to prevent duplicate permissions. It checks if the refinement procedure has already produced equivalent firewall permission, for example, the same named service and destination with a different source, or the same named service and source with a different destination, or the same source and destination with a different named service.

8. For each firewall, execute its library passing as argument the set of optimized firewall permissions.

   The last step of the refinement algorithm is the execution of the specific script library for each firewall. Each one has its own script library, allowing the configuration of firewalls from different vendors. The Library attribute in the Firewall class points to the appropriate script. The optimized firewall permissions are passed as argument to the library.
   In order to configure a named service in a firewall, it is necessary to provide to the firewall library the parameters necessary for the configuration. The required parameters are the named service; the source; and the destination (please see an example in section 6.2). The parameters are organized in a set of tuples (firewall, source, destination, named service) named firewall permissions. For each firewall, from the set of feasible service access, we should build a collection of firewall permissions, that is, each library will receive as the configuration parameter, a list of named services and the corresponding source/destination.
   The steps 7 and 8 correspond to the Firewall Dependent Layer.

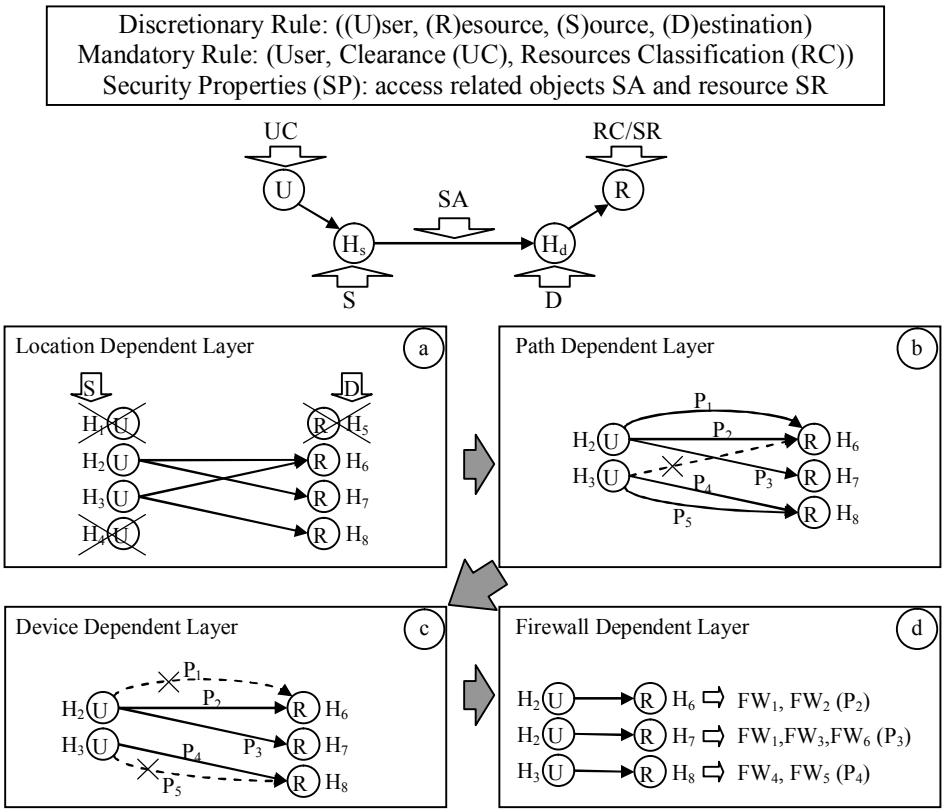

Figure 4.13: A Refinement Example

Figure 4.13 presents a generic refinement example. At the top, the network and the policies are graphically represented. U represents a user that initiates an access from source host H_s to the resource R located at host H_d. UC and RC are, respectively, the user's clearance and resource classification. SA represents the security assumptions of locations, firewalls and services, and SR represents the security requirement of the resource. In the solid arrows represent the resulting permissions of the corresponding layer, and dashed lines represent the permissions that have been removed at the layer.

The Location Dependent Layer corresponds to the steps 1, 2 and 3 and is illustrated in Figure 6a. Here, the algorithm expands user, source (step 1.i), resource and destination groups (step 2.i), but eliminates users, sources (step 1.ii), resources and destinations (step 2.ii) according to their locations. This is done because the rule is only relevant for users and resources in locations specified by the rule, and also because the rule is only relevant for locations (sources and destinations) where the users specified by the rule can be located. Step 3 eliminates users and resources according to the mandatory policy. For example, in Figure 6a, H_1 and H_4 are removed because they aren't valid locations for user U and H_5 is removed because it isn't a valid

location for resource R (according to the located at classes of inventory). The final result of the Location Dependent Layer is the set of (user, resource) pairs that hold both mandatory and discretionary models.

The Path Dependent Layer corresponds to steps 4 and 5 and is illustrated in Figure 6b. The lowest cost path connecting all (source, destination) pair is computed in step 4 using the Dijkstra's shortest path algorithm [17]. The link costs necessary for the shortest path determination are registered in the *Connection* inventory class. Only the service accesses whose paths hold the security property policy are maintained. For example, the service access from H_3 to H_6 is removed because, in the example, its OSA is lower than the SR of the resource.

The Device Dependent Layer corresponds to step 6 and is illustrated in Figure 6c. Here, only the service accesses whose corresponding paths have firewalls implementing the required named services are maintained. This step is necessary to ensure that the service access can be safely enforced, i.e., at least one firewall must implement the required named services.

The Firewall Dependent Layer corresponds to steps 7 and 8 and is illustrated in Figure 6d. This step computes the optimized firewall permission for each firewall and executes the library. It uses the feasible service access computed in step 6 and Library attribute from Firewall inventory class.

4.6 Named Services Libraries

The Named Service Library is responsible for translating optimized firewall permission to rules or scripts used to configure the physical firewalls. Each firewall has its own library, thus it is possible to have firewalls from different vendors. Every library must have at least the Generic Named Service, which is responsible for enforcing the source and the destination of the packets.

The idea behind using a library to perform the low level translation comes from the tremendous variability found in firewall technology with respect to the available functionality and the syntax and semantics of the configuration languages. The concept is simple: network administrators design a named service and carefully implement it using the available firewall configuration resources. The advantage of this approach is that a named service library will be implemented by experts that can fully use the corresponding firewall technology.

The design and implementation of a library named service may vary depending on the syntax of the configuration rules/script and the functionalities of the firewall. For example, some firewalls may need that each rule specifies one source and one destination, while others may

support rules with multiple sources and destinations. The configuration rules/script can include instructions for the firewall that aren't directly dependent of the service accesses, for example, for IPTables [27], it is necessary to include in the script the default policy, in this case, the default is to DROP the packets that doesn't match any rule.

Often, scripts that implements different named services are very similar. For example, considering the scripts for implementing the http, https and ipp (Internet Print Protocol) named services for iptables, the difference between them (for packet filters) are the server port number parameter, which are 80, 443 and 631, respectively. Due to this fact, it is possible to create a template for the similar named services, and fill the templates according to the parameters. For example, the template script line "*iptables -A FORWARD -s {0} -d {1} -p tcp --dport {2} -m state --state NEW,ESTABLISHED -j ACCEPT*" corresponds to a template for tcp packets from client to server. Parameters *{0}* and *{1}* corresponds to client and server IP addresses, and parameter *{2}* correspond to the server port number. Naturally, there are exceptions. Some named services will have different implementations, such as ftp, which needs an additional connection for data transfers. For these services, there are templates specific for each case. Other exceptions are the situations where the named service is implemented in better firewalls, such as a stateful firewall with deep packet inspection. In this case, the named services can have completely different implementations.

Figure 4.14 presents the example of named service library templates for IPTables, IPTables with L7-filter and CheckPoint INSPECT, corresponding to the HTTP named service. Note that a single optimized firewall permission can generate more than one instruction: for IPTables, the first instruction accepts packets of new and established connections from the client to the server, whereas the second accepts packets of established connections from server to client. For INSPECT, the same two instructions are generated. Due to the characteristics of L7-filter, its library needs to generate four instructions. This is necessary because the L7-filter needs to permit the transit of some packets until it is able to determine the type of the traffic. For more details about L7-filter, please see section 7.3 and the documentation available at [18].

IPTables

```
iptables -A FORWARD -s <source> -d <destination> -p tcp
        --dport 80 -m state --state NEW, ESTABLISHED -j accept
iptables -A FORWARD -s <destination> -d <source> -p tcp
        --sport 80 -m state --state ESTABLISHED -j  accept
```

IPTables with L7-filter

```
iptables -A FORWARD -s <source> -d <destination> -p tcp --dport 80
        -m layer7 --l7proto http -m state --state NEW,ESTABLISHED -j ACCEPT
iptables -A FORWARD -s <destination> -d <source> -p tcp --sport 80
        -m layer7 --l7proto http -m state --state ESTABLISHED -j ACCEPT
iptables -A FORWARD -s <source> -d <destination> -p tcp --dport 80
        -m layer7 --l7proto unset -m state --state NEW,ESTABLISHED -j ACCEPT
iptables -A FORWARD -s <destination> -d <source> -p tcp --sport 80
        -m layer7 --l7proto unset -m state --state ESTABLISHED -j ACCEPT
```

INSPECT

```
accept (tcp, http)
        (ip_src = <source>)
        (ip_dst = <destination>)
accept (tcp, http)
        (ip_src = <destination>)
        (ip_dst = <source>)
```

Figure 4.14: Example of Named Service Library Templates

As an example, consider the following optimized firewall permission obtained as result of the processing of the step 7 of the algorithm: *permission(fw3, [HostE3], [Srv3], [http])*. The corresponding script generated for IPTables is presented in Figure 4.15. For http named service, consider the template presented in Figure 4.14. The Inventory contains the information of IP/Mask for the locations *HostE3* and *Srv3*, which are 10.1.4.3/32 and 10.1.3.254/32, respectively.

```
iptables -A FORWARD -s 10.1.4.3/32 -d 10.1.3.254/32 -p tcp --dport 80 -m state
        --state NEW, ESTABLISHED -j accept
iptables -A FORWARD -s 10.1.3.254/32 -d 10.1.4.3/32 -p tcp --sport 80 -m state
        --state ESTABLISHED -j  accept
```

Figure 4.15: IPTables rules example

Note that the optimized firewall permission has generated 2 instructions: the first one accepts packets of new and established connections from the client (10.1.4.3/32) to the server

(10.1.3.254/32), whereas the second accepts packets of established connections from server to client.

At this moment, there are library prototypes for IPTables, IPTables with L7-filter and CheckPoint Firewall-1 (INSPECT language). For these three models, it was observed that the library implementation is very similar. The main differences are in the templates used for script generation, which must be represented using the specific syntax of each firewall language configuration.

4.7 Conclusion

In this chapter, the proposed framework for multi-constraint network security policy was presented. The language for security policy representation allows the representation of mandatory, discretionary and security property policies. The framework includes an algorithm that is responsible for computing the three policy models and generating the low-level configuration scripts/rules.

The proposed framework uses the concept of named service, a high-level representation of a firewall protected service in order to achieve the independency of firewall vendors/models. A specific library for each firewall model/vendor is used to translate the low level permission into configuration scripts/rules.

Note that the resulting policy in low level is the most restrictive, i.e., there will be a low level permission only if this permission is defined in the three high-level policies. The language is also topology independent. When there are changes in the network topology, there is no need to change the policies; the only necessary procedure is to recompile the policy using the new topology specification, and then reconfigure the firewalls with the resulting scripts/rules.

The main contributions of the proposed framework are:

- Specification of three-dimensional framework for network security policy definition, which allows different security staff to be responsible for the description of each dimension;
- Specification of a high-level security policy language, able to handle mandatory, discretionary and security property policies, independently of topology and firewall models/vendors;
- Specification of an information model that comprises the network topology;

- Definition of the Named Service concept, that is a high-level representation of the firewall protection services;
- Specification of the algorithm that process the three policies specifications, responsible for translating the high-level security policy to low-level scrips/rules for firewall configuration;

Chapter 5

Formal Specification and Validation

5.1 Introduction

Formal specifications use mathematical notation to describe in a precise way the properties that an information system must have, without unduly constraining the way in which these properties are achieved. They describe what the system must do without saying how it is to be done.

The refinement algorithm must guarantee that the translation process doesn't cause the violation of the policies. High-level and low-level policies must represent the same set of permissions; otherwise, the whole system can be compromised. Two main theorems must be demonstrated to validate the algorithm:

- **Theorem I**: Every access allowed by the high level policy should be supported by the low level policy (if they can be correctly enforced), and
- **Theorem II**: No action allowed by the low level policy should be forbidden by the high level policy.

These two theorems correspond to the Completeness and Consistency properties. Completeness means that the desired behavior specified in the high level (policy) is completely implemented at low level (firewall scripts). Consistency means that all actions enabled at the low level do not contradict the high level with undesired behavior specification.

The formalism used in this work for formal validation and analysis is based on Z notation [15][70][71]. The Z notation is a mathematical language used for describing and modeling

computing systems. The mathematical roots of Z are first-order logic and set theory. The notation uses standard logical connectives ($\wedge$, $\vee$, $\Rightarrow$, etc.) and set-theoretic operations ($\in$, $\rightarrow$, $\downarrow$, etc.) with their standard semantics. Using the Z language it is possible to provide a model of a mathematical object. Theses objects bear a resemblance to computational objects reflect the intention that Z to be used as a specification language for software engineering.

A Z specification consists of sections of mathematical text interspersed with prose. The mathematical text is a collection of types together with some predicates that must hold for the values of each type. Types in Z are sets of values. Z provides some fundamental types in its basic toolkit that are primitive, such as $\mathbb{N}$ for natural numbers and $\mathbb{Z}$ for integers. In addition, we can introduce further primitive types, called given types, by writing them in square brackets.

Note that the Z specification represents how the algorithm must behave mathematically. The real algorithm can include optimizations and other improvements that don't change the mathematical relations. Thus, its specification doesn't necessarily reflect the algorithm directly.

The Z/EVES tool [16] is used to support the specification and manipulation of Z notation. It is an interactive system for composing, checking, and analyzing Z specifications. The validation approach used in this work consists of representing the algorithm in Z notation; specifying Z/Eves theorems that represents the properties the system must hold; and using the Z/Eves engine to prove the theorems. Z/Eves proofs work on a predicate called the *goal*; each step transforms the goal into a new equivalent goal. Transforming a goal to *true* thus completes a proof. For more details about Z/Eves usage, please see [69].

5.2 Definitions

In order to represent the algorithm mathematically, some sets were defined. The sets corresponding to the inventory are the following:

- *Users:* $\mathbb{P}$ *USER*: network registered users;
- *Resources:* $\mathbb{P}$ *RESOURCE*: network registered resources;
- *NamedServices:* $\mathbb{P}$ *NAMEDSERVICE*: network registered named services;
- *Locations:* $\mathbb{P}$ *LOCATION*: network registered locations;
- *Actions:* $\mathbb{P}$ *ACTION*: the actions included in a discretionary rule. For this work, *Actions = { accept }*;

- *Firewalls:* $\mathbb{P}$ *FIREWALL*: network registered firewalls;
- *ResourceNamedServices: RESOURCE* $\rightarrow$ $\mathbb{P}$ *NAMEDSERVICE*: relation between the resource and the named services associated to the resource;
- *UserLocatedAt: USER* $\leftrightarrow$ *LOCATION*: represents the locations of the users in the network;
- *ResourceLocatedAt: RESOURCE* $\leftrightarrow$ *LOCATION*: represents the locations where the resources are available in the network;
- *FirewallNamedServices: FIREWALL* $\leftrightarrow$ *NAMEDSERVICE*: relation between the firewalls and the named services that they implement;
- *NetworkPaths:* $\mathbb{P}$ (seq *NODE*): set of all network paths. In this Z specification, we consider that the are already calculated and stored in this relation;

The following sets corresponds to the three policies models:

- *Rule:* $\mathbb{P}$ *USER* $\times$ $\mathbb{P}$ *RESOURCE* $\times$ $\mathbb{P}$ *LOCATION* $\times$ $\mathbb{P}$ *LOCATION* $\times$ *ACTION*: represents a rule of the discretionary policy. The rule contains a set of users, resources, sources and destinations, and one action;
- *Clearance: USER* $\rightarrow$ $\mathbb{N}$: maps the users to their clearances (mandatory policy);
- *Classification: RESOURCE* $\rightarrow$ $\mathbb{N}$: maps the resources to theirs classifications (mandatory policy);
- *UserCompartments: USER* $\rightarrow$ $\mathbb{P}$ *COMPARTMENT*: maps the users to a set of compartments (mandatory policy);
- *ResourceCompartments: RESOURCE* $\rightarrow$ $\mathbb{P}$ *COMPARTMENT*: maps the resources to a set of compartments (mandatory policy);
- *SR: RESOURCE* $\rightarrow$ *SC*: maps the resources to their security requirements (security property policy);
- *SANamedService: NAMEDSERVICE* $\rightarrow$ *SC* : maps the named services to their security assumptions (security property policy);
- *SANode: NODE* $\rightarrow$ *SC*: maps nodes to their security assumptions (security property policy);

Finally, the following sets are produced:

- *RulePermissions:* $\mathbb{P}$ ((*USER* $\times$ *RESOURCE*) $\times$ (*LOCATION* $\times$ *LOCATION*)): set of tupes ((*user*, *resource*), (*source*, *destination*)) that hold the discretionary policy and that the locations of the user and the resource are in conformance with Inventory, i.e., the *user* belongs to the set of users specified by the rule, the *resource* belongs to the set of resources specified by the rule, the *source* belongs to the set of sources specified by the rule, the *destination* belongs to the set of destinations specified by the rule, the *(user, source)* pair belongs to the *UserLocatedAt* set (meaning that the user can initiate an access from the source) and that the *(resource, destination)* pair belongs to the *ResourceLocatedAt* (meaning the resource is delivered at destination).
- *Accesses:* $\mathbb{P}$ (*USER* $\times$ *RESOURCE*): contains pair of *(user, resource)*, where the user is permitted to access the resource by the mandatory policy;
- *SUP:* $SC \times SC \rightarrow SC$: function that returns a security class with the greater values of security levels of the two security classes;
- *INF:* $\mathbb{P}\ SC \rightarrow SC$: function that returns a security class with the lower values of each security level from a set of security classes;
- *ESA: NAMEDSERVICE* $\times$ *NODE* $\rightarrow$ *SC*: function that returns the effective security assumption for a named service and a node;
- *OSA: NAMEDSERVICE* $\times$ seq *NODE* $\rightarrow$ *SC*: function that returns the overall security assumption for a named service and a sequence of nodes (path) ;
- *GOSA:* $SC \leftrightarrow SC$: relation between two security classes, where the first element of the tuple is greater than or equal to the second element (for a tuple (SC_1, SC_2), SC_1 is greater than or equal to SC_2).
- *ResourcesPaths:* $\mathbb{P}$ (*RESOURCE* $\times$ seq *NODE*): contains pairs *(resource, path)* such that the resource is located at the last node of the path (destination) and the path is a registered network path;
- *PossibleVias:* $\mathbb{P}$ (*RESOURCE* $\times$ *NAMEDSERVICE* $\times$ seq *NODE*): contains tuples *(resource, namedservice, path)* that holds the security property policy, i.e., the named service associated to the resource is delivered at the last node of the path, and the OSA

of the named service and path is equal to or greater than the resource security requirement.

- *ServiceAccesses:* $\mathbb{P}$ ((*USER* $\times$ *RESOURCE*) $\times$ (*LOCATION* $\times$ *LOCATION*) $\times$ (*NAMEDSERVICE* $\times$ seq *NODE*)): corresponds to the tuple *((user, resource), (source, destination), (named service, path))*, which means that the user from the source location is allowed to access the resource at the destination location through the path using the named service.
- *PossibleServiceAccesses:* $\mathbb{P}$ ((*USER* $\times$ *RESOURCE*) $\times$ (*LOCATION* $\times$ *LOCATION*) $\times$ (*NAMEDSERVICE* $\times$ seq *NODE*)): contains the service accesses that holds the three policy models;
- *FeasibleServiceAccesses:* $\mathbb{P}$ ((*USER* $\times$ *RESOURCE*) $\times$ (*LOCATION* $\times$ *LOCATION*) $\times$ (*NAMEDSERVICE* $\times$ seq *NODE*)): contains the service accesses that hold the three policy models and that includes in their paths at least one firewall capable to implement the corresponding named service.
- *FirewallPermissions: Firewall* $\leftrightarrow$ $\mathbb{P}$ ((*USER* $\times$ *RESOURCE*) $\times$ (*LOCATION* $\times$ *LOCATION*) $\times$ *NAMEDSERVICE*): contains the permissions that must be configured in each firewall of the network.

5.3 Demonstration Rationale

The theorem I states that “Every access allowed by the high level policy should be supported by the low level policy (if they can be correctly enforced)”. In the proposed framework, an access in high level policy corresponds to an access permitted by mandatory, discretionary and security property policies where at least one of the firewalls in the path from the source and destination is able to correctly implement the corresponding Named Service.

In order to make the demonstration clearer it is split in the following lemmas:

Lemma 1: If the discretionary, mandatory and security property policies specify a permission, then this permission must have a corresponding service access belonging to the *PossibleServiceAccesses* set. This lemma is split into four lemmas:

Lemma 1.1: A user from the source is permitted to access a resource at the destination if discretionary policy specifies this permission (i.e., the user from the source location is permitted to access the resource at destination location if the discretionary policy explicitly specify the user, source, resource and destination and that the user can be located at the source and the resource is located at destination).

Lemma 1.2: A user is permitted to access a resource if mandatory policy specifies this permission (i.e., the user clearance is greater than or equal to the resource classification and if the user is associated to the compartments necessary to access the resource).

Lemma 1.3: A path and a named service can be used to access a resource if the security property policy specifies this permission (i.e., the overall security assumption of the path (including the source location) and the named service are greater than or equal to the security requirement of the resource).

Lemma 1.4: If a permission is defined in discretionary, mandatory and security property policies, then there must be a corresponding service access belonging to the possible service accesses set.

Lemma 2: If a service access belonging to the *PossibleServiceAccesses* set has its named service implemented by at least one firewall in the associated path, then this service access belongs to the *FeasibleServiceAccesses* set.

Lemma 3: If a service access is a member of *FeasibleServiceAccesses*, then there must be a corresponding firewall permission in the *FirewallPermissions* set for each firewall present in the associated path.

Lemma 1 assures that, if there exist such a permission in the high level, the corresponding user, resource, source, destination, named service and path will be present in a service access in the *PossibleServiceAccesses* set. This is a sufficient condition for any granted high level permission to be included in the *PossibleServiceAccesses* set. Lemma 1 is split into 4 lemmas, which corresponds to the three policy models and the intersection between them. A permission can only exist at low level if it is specified in the three policy models. These lemmas are presented in Figure 5.12 to 5.17.

Lemma 2 assures for any high level permission, if it exists at the high level and if the corresponding shortest path includes at least one firewall implementing the necessary named service, the permission is included in the *FeasibleServiceAccesses* set. This is a sufficient

condition for any granted high level permission, which is protected by a firewall in the corresponding shortest path, to be included in the *FeasibleServiceAccesses* set.

Finally, Lemma 3 assures for any high level permission, if it exists at high level and the corresponding named service can be implemented by at least one firewall in the path, this permission is included in the set of permissions that must be configured in the firewalls present in the corresponding path.

Lemma 2 and Lemma 3 are directly demonstrated by a corresponding Z specification, presented in Figure 5.16 and Figure 5.17.

On the other hand, theorem II states that "No action allowed by the low level policy should be forbidden by the high level policy". In the proposed framework, this means that if the firewall is configured with some service access, then there must be an access permitted by mandatory, discretionary and security property policies and at least one firewall in the path corresponding to this service access implements the necessary named services.

Similarly to Theorem I, this theorem is split in the following lemmas:

Lemma 4: For each firewall permission belonging to *FirewallPermissions*, there is a corresponding service access belonging to the *FeasibleServiceAccesses* set.

Lemma 5: For each service access belonging to *FeasibleServiceAccesses* set, there is a corresponding service access belonging to *PossibleServiceAccesses* set.

Lemma 6: The service accesses belonging to *PossibleServicesAccesses* set must hold the mandatory, discretionary and security property policies. This lemma was split into 3 lemmas:

Lemma 6.1: The service access belonging to *PossibleServicesAccesses* set must hold the discretionary policy, i.e., the rule must specify the user and the source, which are related by *UserLocatedAt*, and must specify the resource and the destination, which are related by *ResourceLocatedAt*.

Lemma 6.2: The service access belonging to *PossibleServicesAccesses* set must hold the mandatory policy, i.e., the user clearance is greater than or equal to than resource classification and the user is associated to the compartments necessary to access the resource.

Lemma 6.3: The service access belonging to *PossibleServicesAccesses* set must hold security property policy, i.e., the overall security assumption of the path and named service are greater than or equal to the resource security requirement.

These lemmas correspond to the inverse analysis of the refinement. In other words, they validates that if there is some permission in the result of the processing, then this permission must be defined in high level. In this way, the goal of the lemmas is to demonstrate the results of each step correspond to the desired behavior of the step, i.e., that the output doesn't include any permission that is not defined in the input of the step. Lemma 4 demonstrates the relation between the firewall permissions and feasible service accesses, i.e., the existence of a corresponding feasible service access is a necessary condition for the existence of the firewall permission. Lemma 5 demonstrates the relation between feasible service accesses and possible service accesses. This is the necessary condition for a high level permission to be present in the feasible service accesses, which means that at least one firewall in the corresponding path is able to implement the named service and that the three policy models specifies the permission. Lemma 6 demonstrates the relation between possible service access and the three policies, meaning that the necessary condition to a service access be a member of possible service access is that the three policy models specify the permission. Therefore, if these lemmas are proved, then the theorem is proved.

Lemmas 4 and 5 are directly demonstrated in Figure 5.18 and Figure 5.19. Lemma 6 is split into 3 lemmas, corresponding to the three policy models, which are presented in Figure 5.20 to 5.23.

Note that the application of the named service libraries for the generation of the firewall rules/scripts is not in scope of this specification, since they are dependent of firewall model/vendor and are designed by network administrators. This specification corresponds to the processing performed during the high-level to low-level conversion, and thus demonstrates the overall transformations performed by the algorithm.

5.4 Z Specification

In Z, it is necessary to define the set types (also named given sets), which defines all possible values for a set of a restricted kind. The construction of elements in a given types is not provided in a specification, usually because that level of detail is not necessary for the purposes of the specification. By convention, given types are written in all capital letters. For example, the type *USER* refers to all users. The following types were defined:

$$[USER, RESOURCE, NAMEDSERVICE, ACTION, NODE, SC, COMPARTMENT]$$

An element of a type is declared using a colon (:). So it is possible to write *user: USER* and read this as "user is of type *USER*", meaning that user is an element in the set of values defined by *USER*. The name *user* is called a variable, that is to say it is not know the denotation of the name *user*, only that it denotes some undetermined value of type *USER*. Since *USER* is a also a set, we could also write $user \in USER$, using the set membership function $\in$. Z uses the : notation when a variable is declared and $\in$ to express predicates over variables. Note that the meaning of a variable in mathematics is not the same as the meaning of variable in programming. In programming, a variable is a store in which different values can appear from time to time. In mathematics a variable does not change its value, but its value might not be determined.

New types can also be defined by constructing them from primitive types, using the following type constructors:

- $\mathbb{P}X$ is the set of all subsets with elements from type X, also called the powerset of X.
- $X \times Y$ is the type consisting of all ordered pairs (x, y) whose first element is of type X and whose second element is of type Y, also called the cross-product of X and Y.
- seq X is the set of all sequences, or lists, of elements from X, including empty and infinite sequences. The representation $\langle a_1, \ldots, a_n \rangle$ is a shorthand for the set $\{1 \mapsto a_1, \ldots, n \mapsto a_n\}$, so the function *ran* returns the range of a sequence, that is a set of the elements of the path. Note that a sequence is ordered, while a set isn't ordered. The functions *head* and *last* represent the first and the last elements of a sequence. The empty sequence $\langle\rangle$ is an alternative notation for the empty function $\varnothing$ from $\mathbb{N}$ to X.
- Relations and functions between types identify special subsets of the cross-product type. In this thesis, the following are used:
    - $X \leftrightarrow Y$ is the set of all relations between domain type X and range type Y. A relation is simply a subset of $X \times Y$, i.e., $X \leftrightarrow Y == \mathbb{P}(X \times Y)$.
    - $X \rightarrow Y$ is the set of all total functions. Total functions are defined on all elements of the domain type: $X \rightarrow Y == \{f: X \leftrightarrow Y \mid \forall x: X \bullet \exists_1 y : Y \bullet (x,y) \in f\}$.

After defining the types, it is possible to specify the basic sets. For example, the *FIREWALL* set of elements of type *NODE* represents the collection of registered firewalls. With the types defined, it is possible to define derived sets, optionally including a constraint on their contents. Both *FIREWALL* and *LOCATION* are sets of nodes, and are defined as:

$$FIREWALL: \mathbb{P}\ NODE$$
$$LOCATION: \mathbb{P}\ NODE$$

The left bar is used to represent an axiomatic description. An axiomatic description introduces one or more global variables, and optionally specifies a constraint on their values. This means that the set *FIREWALL* and LOCATION are global sets.

After defining the basic types and axioms, the next step it to define the schemas that represent the behavior of the algorithm. A schema is a piece of mathematical text that specifies some aspect of the software system that is being discussed. The name of a schema will be used elsewhere in the document to refer to the mathematical text. A schema contains two parts: the signature (or declaration part), which is a collection of variables, and the predicates, which define the properties of the algorithm. When there are more than one line in the predicate, it is understood the conjunction between the predicates.

Figure 5.1 shows the *Inventory* schema. Lines 1 to 12 are the signature, and lines 13 to 20 are the predicates. Lines 1 to 5 define the network registered users, resources, named services, locations and firewalls sets. As explained earlier, the notation $\mathbb{P}USER$ is a set of all subsets with elements from type *USER*, in other words, the notation $Users: \mathbb{P}USER$ means that *Users* is a set of elements of type *USER*. Lines 6 to 10 represent the sets of named services associated to resources, the locations of the users, the locations of the resources, the named services associated to the firewalls and the paths of the network. Since we use Dijkstra's algorithm to compute the paths, the mathematical relations between locations, connections and paths won't be represented here. Line 12 defines the generic named service, as an element of type *NAMEDSERVICE*.

Lines 13 to 20 define the relations between the defined sets. For example, line 13 states that the domain of *UserLocatedAt* set is the *User* set, which means that the users that are related to locations registered in *UserLocatedAt* set must be valid network users. Line 20 states that the range of *FirewallNamedServices* is the *NamedServices* set, which means that the firewalls only can be associated to valid network named services.

Inventory

1 $Users: \mathbb{P}\ USER$
2 $Resources: \mathbb{P}\ RESOURCE$
3 $NamedServices: \mathbb{P}\ NAMEDSERVICE$
4 $Locations: \mathbb{P}\ LOCATION$
5 $Firewalls: \mathbb{P}\ FIREWALL$
6 $ResourceNamedServices: RESOURCE \rightarrow \mathbb{P}\ NAMEDSERVICE$
7 $UserLocatedAt: USER \leftrightarrow LOCATION$
8 $ResourceLocatedAt: RESOURCE \leftrightarrow LOCATION$
9 $FirewallNamedServices: FIREWALL \leftrightarrow NAMEDSERVICE$
10 $NetworkPaths: \mathbb{P}\ (\text{seq}\ NODE)$
11 $Actions: \mathbb{P}\ ACTION$
12 $Generic: NAMEDSERVICE$

---

13 $Users = \text{dom}\ UserLocatedAt$
14 $Locations = \text{ran}\ UserLocatedAt$
15 $Resources = \text{dom}\ ResourceLocatedAt$
16 $Locations = \text{ran}\ ResourceLocatedAt$
17 $Resources = \text{dom}\ ResourceNamedServices$
18 $\mathbb{P}\ NamedServices = \text{ran}\ ResourceNamedServices$
19 $Firewalls = \text{dom}\ FirewallNamedServices$
20 $NamedServices = \text{ran}\ FirewallNamedServices$

Figure 5.1: Inventory Schema

Figure 5.2 presents the *Policy* schema. Line 1 includes the *Inventory*, which means that the declarations of schema *Inventory* are included in declaration part of *Policy* schema. Line 2 defines the structure of a single discretionary rule. Since we are considering only *accept* action, it is not necessary to consider the relation between two or more rules. A permission would exist at low-level only if one or more high-level rules specifies this permission.

A discretionary rule is a tuple *(users, resources, sources, destinations, action)*. *Users*, *resources*, *sources* and *destinations* are sets of elements; for example, *users* can specify a single user or several users. The exception is the *action*, which allows only a single action for each discretionary rule. Lines 3 to 6 define the clearance, classification and compartments of mandatory model. Clearance and classification are functions from *user* and *resource* to a natural number. *UserCompartments* and *ResourceCompartments* are functions that maps *users* and *resources* to a set of *compartments*. This means that each user and each resource can be associated to none, one or many compartments. Lines 7 to 9 represent the security property model. SR is a function that maps resources to security requirements (security classes), and

SANamedService and SANode are functions that maps named services and nodes to security assumptions (security classes). Similarly to Inventory, lines 10 to 16 state relations between the sets defined by the policy.

Policy

1 *Inventory*
2 *Rule:* $\mathbb{P}$ *USER* $\times$ $\mathbb{P}$ *RESOURCE* $\times$ $\mathbb{P}$ *LOCATION* $\times$ $\mathbb{P}$ *LOCATION* $\times$ *ACTION*
3 *Clearance: USER* $\rightarrow$ $\mathbb{N}$
4 *Classification: RESOURCE* $\rightarrow$ $\mathbb{N}$
5 *UserCompartments: USER* $\rightarrow$ $\mathbb{P}$ *COMPARTMENT*
6 *ResourceCompartments: RESOURCE* $\rightarrow$ $\mathbb{P}$ *COMPARTMENT*
7 *SR: RESOURCE* $\rightarrow$ *SC*
8 *SANamedService: NAMEDSERVICE* $\rightarrow$ *SC*
9 *SANode: NODE* $\rightarrow$ *SC*

---

10 *Users* = dom *Clearance*
11 *Resources* = dom *Classification*
12 *Users* = dom *UserCompartments*
13 *Resources* = dom *ResourceCompartments*
14 *Resources* = dom *SR*
15 *NamedServices* = dom *SANamedService*
16 *Locations* $\cup$ *Firewalls* = dom *SANode*

Figure 5.2: Policies Schema

Figure 5.3 shows the discretionary model schema. This schema is responsible for representing the mathematical relations corresponding to the processing of discretionary policy. Line 1 includes the *Policies* schema (which implicitly includes *Inventory*) while lines 2 to 6 define elements and sets used in the discretionary model specification. Line 2 defines the type of the *RulePermissions* set, which contains the tuples *((user, resource), (source, destination))* that hold the discretionary policy. Lines 3 to 6 defines the elements that are used in the mapping of the rule (line 7).

Line 7 presents the mapping of *Rule* to *users*, *resources*, *sources*, *destinations* and *action*. Lines 8 to 16 presents the definition of *RulePermissions* set, which contains tuples *((user, resource), (source, destination))* such that the *user* belongs to the set of users specified by the rule, the *resource* belongs to the set of resources specified by the rule, the *source* belongs to the set of sources specified by the rule, the *destination* belongs to the set of destinations specified by the rule, the *(user, source)* pair belongs to the *UserLocatedAt* set (meaning that the user can

initiate an access from the source) and that the *(resource, destination)* pair belongs to the *ResourceLocatedAt*, meaning the resource is delivered at destination.

DiscretionarySchema

1 *Policy*
2 $RulePermissions: \mathbb{P}((USER \times RESOURCE) \times (LOCATION \times LOCATION))$
3 $users: \mathbb{P}\ USER$
4 $resources: \mathbb{P}\ RESOURCE$
5 $sources, destinations: \mathbb{P}\ LOCATION$
6 $action: ACTION$

---

7 $Rule = (users, resources, sources, destinations, action)$
8 $RulePermissions$
9 $= \{\ user: Users; resource: Resources; source, destination: Locations$
10 $\mid user \in users$
11 $\land resource \in resources$
12 $\land source \in sources$
13 $\land destination \in destinations$
14 $\land (user, source) \in UserLocatedAt$
15 $\land (resource, destination) \in ResourceLocatedAt$
16 $\bullet ((user, resource), (source, destination))\ \}$

Figure 5.3: Discretionary Schema

Figure 5.4 presents the schema with the mathematical specification of mandatory model. Line 1 includes the DiscretionarySchema, i.e., the definitions of the discretionary schema are included in this schema. Line 2 defines the *Accesses* set, which contains *(user, resource)* pairs that hold mandatory model. Lines 3 to 7 shows the mathematical relation that defines the content of *Accesses*: given a user belonging to *Users* and a resource belonging to *Resources*, such that the user clearance is greater than or equal to the resource classification and resources' compartments are a subset of the users' compartments, then the pair *(user, resource)* will belong to set *Accesses*.

MandatorySchema

1 *DiscretionarySchema*
2 *Accesses:* $\mathbb{P}$ (*USER* $\times$ *RESOURCE*)

3 *Accesses*
4 = { *user: Users; resource: Resources*
5 | *Clearance user* $\geqslant$ *Classification resource*
6 $\wedge$ *ResourceCompartments resource* $\subseteq$ *UserCompartments user*
7 • (*user*, *resource*) }

Figure 5.4: Mandatory Schema

Figure 5.5 presents the schema for *ResourcesPaths*. This set is an intermediate step in computation of security property model. This set contains pairs *(resource, path)*, meaning that an access can be performed through this path to reach the resource. A pair *(resource, path)* will belong to *ResourcesPaths* if the path is not null and is a network path, if the resource is a network registered resource; if the resource is associated to a destination in set *ResourceLocatedAt*; and if the destination is the last node in the path.

ResourcesPathsSchema

1 *MandatorySchema*
2 *ResourcesPaths:* $\mathbb{P}$ (*RESOURCE* $\times$ seq *NODE*)
3

4 *ResourcesPaths*
5 = { *resource: RESOURCE; destination: Locations; path:* seq *NODE*
6 | *path* $\neq \langle\rangle$
7 $\wedge$ *path* $\in$ *NetworkPaths*
8 $\wedge$ *resource* $\in$ *Resources*
9 $\wedge$ (*resource*, *destination*) $\in$ *ResourceLocatedAt*
10 $\wedge$ *destination* = *last path* • (*resource*, *path*) }

Figure 5.5: ResourcesPaths Schema

Figure 5.6 presents the definitions for the functions ESA, OSA, SUP and INF, and the relation GOSA. The SUP is a function that returns a security class with the greater values of security levels of the two security classes, and the INF function returns a security class with the lower values from a set of security classes (see section 4.3.4). GOSA is a relation between two security classes, where the first element of the tuple is greater than or equal to the second element (for a tuple (SC_1, SC_2), SC_1 is greater than or equal to SC_2). Since the security class is

defined as a type in Z, SUP, INF and GOSA are not explicitly specified in Z. The ESA function returns the effective security assumption for a named service and a node, and OSA function returns the overall security assumption for a named service and a sequence of nodes (path).

SecurityPropertyFunctionsSchema

1 *ResourcesPathsSchema*
2 $ESA: NAMEDSERVICE \times NODE \rightarrow SC$
3 $OSA: NAMEDSERVICE \times \text{seq } NODE \rightarrow SC$
4 $SUP: SC \times SC \rightarrow SC$
5 $INF: \mathbb{P}\, SC \rightarrow SC$
6 $GOSA: SC \leftrightarrow SC$

---

7 ESA
8 $= \{\ namedservice: NAMEDSERVICE; node: NODE; sc: SC$
9 $\mid sc = SUP\ ((SANamedService\ namedservice), (SANode\ node))$
10 $\bullet\ ((namedservice, node), sc)\ \}$
11 OSA
12 $= \{\ namedservice: NAMEDSERVICE; path: \text{seq } NODE; scseq: \mathbb{P}\, SC; scres: SC$
13 $\mid scseq$
14 $= \{\ sc: SC; node: \text{ran } path \mid sc = ESA\ (namedservice, node)$
15 $\bullet\ sc\ \}$
16 $\wedge\ scres = INF\ scseq \bullet\ ((namedservice, path), scres)\ \}$

Figure 5.6: SecurityPropertyFunctions Schema

Figure 5.7 presents the schema for security property model. The set *PossibleVias* contains all tuples *(resource, namedservice, path)* such that there is a *(resource, path)* that belongs to *ResourcesPaths*, the named service is associated to the *resource* and that the overall security assumption of the *namedservice* and *path* is greater than or equal to the security requirement of the *resource*. In other words, all the tuples in *PossibleVias* hold security property policy.

SecurityPropertySchema

1 *SecurityPropertyFunctionsSchema*
2 *PossibleVias:* ℙ (*RESOURCE* × *NAMEDSERVICE* × seq *NODE*)

---

3 *PossibleVias*
4 = { *resource: RESOURCE; namedservice: NAMEDSERVICE; path:* seq *NODE*
5 | (*resource*, *path*) ∈ *ResourcesPaths*
6 ∧ *namedservice* ∈ *ResourceNamedServices resource*
7 ∧ (*OSA* (*namedservice*, *path*), *SR resource*) ∈ *GOSA*
8 • (*resource*, *namedservice*, *path*) }

Figure 5.7 : SecurityProperties Schema

Figure 5.8 presents the schema for *PossibleServiceAccesses* set. This set contains the service accesses that hold the discretionary, the mandatory and the security property models, i.e., the permissions that are allowed by the three policies. This set contains the tuples *((user, resource), (source, destination),(namedservice, path))*, where ((*user, resource), (source, destination))* tuple belongs to *RulePermissions*, *(user, resource)* tuple belongs to *Accesses* set, *(resource, namedservice, path)* tuple belongs to *PossibleVias* and that *source* and *destination* are the first and the last nodes of the path.

PossibleServiceAccessesSchema

1 *SecurityPropertySchema*
2 *PossibleServiceAccesses:* ℙ ((*USER* × *RESOURCE*) × (*LOCATION* × *LOCATION*) ×
3 (*NAMEDSERVICE* × seq *NODE*))

---

4 *PossibleServiceAccesses*
5 = { *user: USER; resource: RESOURCE; namedservice: NAMEDSERVICE; source,*
6 *destination: LOCATION; path: NetworkPaths*
7 | ((*user*, *resource*), (*source*, *destination*)) ∈ *RulePermissions*
8 ∧ (*user*, *resource*) ∈ *Accesses*
9 ∧ (*resource*, *namedservice*, *path*) ∈ *PossibleVias*
10 ∧ *source* = *head path*
11 ∧ *destination* = *last path*
12 • ((*user*, *resource*), (*source*, *destination*), (*namedservice*, *path*)) }

Figure 5.8 : PossibleServiceAccesses Schema

Figure 5.9 presents the schema for *FeasibleServiceAccesses* set. This set contains the service accesses that hold the discretionary, the mandatory and the security property models (which are the service accesses that belong to *PossibleServiceAccesses* set) and that the associated path has

at least one firewall capable of implementing the named service (i.e., exists a firewall *auxFirewall* in the path such that the *(auxFirewall, namedservice)* belongs to *FirewallNamedServices*).

FeasibleServiceAccessesSchema

1 *PossibleServiceAccessesSchema*
2 *FeasibleServiceAccesses:* $\mathbb{P}$ ((*USER* $\times$ *RESOURCE*) $\times$ (*LOCATION* $\times$ *LOCATION*) $\times$
3 (*NAMEDSERVICE* $\times$ seq *NODE*))

---

4 *FeasibleServiceAccesses*
5 = { *user: USER; resource: RESOURCE; namedservice: NAMEDSERVICE; source,*
6 *destination: LOCATION; path:* seq *NODE*
7 | ((*user*, *resource*), (*source*, *destination*), (*namedservice*, *path*))
8 $\in$ *PossibleServiceAccesses*
9 $\wedge$ ($\exists$*auxFirewall:* ran *path*
10 $\bullet$ (*auxFirewall*, *namedservice*) $\in$ *FirewallNamedServices*)
11 $\bullet$ ((*user*, *resource*), (*source*, *destination*), (*namedservice*, *path*)) }

Figure 5.9 : FeasibleServiceAccess Schema

Figure 5.10 presents the schema definition for *FirewallPermissions* set. This set contains the tuples with the firewalls and the corresponding service accesses. The firewall permissions are determined in the following way: the permission that must be configured in a firewall must correspond to a service access belonging to *FeasibleServiceAccess*, the firewall must be a node of the path corresponding to the feasible service access and the firewall must implement the named service or the generic named service.

FirewallPermissionsSchema

1 *FeasibleServiceAccessesSchema*
2 *FirewallPermissions: FIREWALL*
3 $\leftrightarrow \mathbb{P}$ ((*USER* $\times$ *RESOURCE*) $\times$ (*LOCATION* $\times$ *LOCATION*) $\times$
4 *NAMEDSERVICE*)

---

5 *FirewallPermissions*
6 = { *firewall: Firewalls;*
7 *servs:* $\mathbb{P}$ ((*USER* $\times$ *RESOURCE*) $\times$ (*LOCATION* $\times$ *LOCATION*) $\times$ *NAMEDSERVICE*)
8 | *servs*
9 = { *user: USER; resource: RESOURCE; source, destination: LOCATION;*
10 *namedservice: NAMEDSERVICE; path:* seq *NODE;*
11 *fw_service: NAMEDSERVICE*
12 | ((*user*, *resource*), (*source*, *destination*),
13 (*namedservice*, *path*)) $\in$ *FeasibleServiceAccesses*
14 $\wedge$ *firewall* $\in$ ran *path*
15 $\wedge$ ((*firewall*, *namedservice*) $\in$ *FirewallNamedServices*
16 $\wedge$ *fw_service* = *namedservice*
17 $\vee \neg$ (*firewall*, *namedservice*) $\in$ *FirewallNamedServices*
18 $\wedge$ (*firewall*, *generic*) $\in$ *FirewallNamedServices*
19 $\wedge$ *fw_service* = *generic*)
20 • ((*user*, *resource*), (*source*, *destination*), *fw_service*) }
21 • (*firewall*, *servs*) }

Figure 5.10: FirewallPermissions Schema

5.5 Demonstration of Theorem I

As mentioned earlier in this chapter, Z/Eves proofs work on a predicate called the *goal*. Named proof goals in Z/Eves are established by theorem declarations or by domain checking conditions for paragraphs. Due to this characteristic of Z/Eves, the lemmas are represented by Z theorem declarations.

One important concept used in this specification is the Skolemization [72][73]. Skolemization is a procedure frequently used in automated theorem proving. It is a process of eliminating existentially quantified variables from a formula by replacing them with Skolem constants and Skolem functions. A formula of first-order logic is in Skolem normal form if it is in conjunctive prenex normal form with only universal first-order quantifiers. Every first-order formula can be converted into Skolem normal form while not changing its satisfiability via a process called Skolemization. The resulting formula is not necessarily equivalent to the original one, but is equisatisfiable with it: it is satisfiable if and only if the original one is.

The simplest form of Skolemization is for existentially quantified variables which are not inside the scope of universal quantifiers. These can simply be replaced by creating new constants. For example, $\exists x \bullet P(x)$ can be changed to $P(c)$, where c is a new constant. More generally, Skolemization is performed by replacing every existentially quantified variable y with a term $f(x_1, \ldots, x_n)$ whose function symbol f is new (does not occur anywhere else in the formula), where $x_1, \ldots, x_n$ are the variables that are universally quantified and whose quantifiers precede that of y.

The Skolemization process is used to simplify the predicates and thus allow the use of tools for automated theorem proving. This process is used in some schemas of this formal specification, in order to allow the complete prove using Z/Eves.

Figure 5.11 presents the Skolemized FeasibleServiceAccesses Schema. Lines 9 and 10 of Figure 5.9 contain the existencial quantifier $\exists auxFirewall$, which is substituted by a constant with same name in lines 4, 10 and 11 of Figure 5.11.

FeasibleServiceAccessesSchema

```
1   PossibleServiceAccessesSchema
2   FeasibleServiceAccesses: ℙ ((USER × RESOURCE) × (LOCATION × LOCATION) ×
3                               (NAMEDSERVICE × seq NODE))
4   auxFirewall: FIREWALL
    ──────────────────────────────
5   FeasibleServiceAccesses
6     = { user: USER; resource: RESOURCE; namedservice: NAMEDSERVICE; source,
7       destination: LOCATION; path: seq NODE
8         | ((user, resource), (source, destination), (namedservice, path))
9             ∈ PossibleServiceAccesses
10          ∧ auxFirewall ∈ ran path
11          ∧ (auxFirewall, namedservice) ∈ FirewallNamedServices
12        • ((user, resource), (source, destination), (namedservice, path)) }
```

Figure 5.11: Skolemized FeasibleServiceAccess Schema

The specification of theorems in Z/Eves can be done in two ways: if the right side of implication contains only one predicate, the structure is that presented in Figure 5.12; otherwise, in the situations where more than one predicate is present in the right side, it is necessary to define a schema with the predicates and then use the named of the schema as the right side of implication, as presented in Figure 5.18. The use of first or second structure depends only on the

number of predicates, and both are used for the validation of the theorems presented in this thesis.

Figure 5.12 presents the Z specification for Lemma 1.1. It is responsible for the validation of the discretionary policy. If a *user* belongs to the set of users specified by the rule, a *resource* belongs to the set of resources specified by the rule, a *source* belongs to the set of sources specified by the rule and a *destination* belongs to the set of destinations specified by the rule, and the *(user, source)* pair belongs to the *UserLocatedAt* set (meaning that the user can initiate an access from the source) and that the *(resource, destination)* pair belongs to the *ResourceLocatedAt*, meaning the resource is delivered at destination, then the tuple *((user, resource), (source, destination))* must be a member of *RulePermissions* set.

theorem rule *Lemma11*
DiscretionarySchema
$\wedge$ *user* $\in$ *users*
$\wedge$ *resource* $\in$ *resources*
$\wedge$ *source* $\in$ *Locations*
$\wedge$ *destination* $\in$ *Locations*
$\wedge$ *source* $\in$ *sources*
$\wedge$ *destination* $\in$ *destinations*
$\wedge$ (*user*, *source*) $\in$ *UserLocatedAt*
$\wedge$ (*resource*, *destination*) $\in$ *ResourceLocatedAt*
$\Rightarrow$ ((*user*, *resource*), (*source*, *destination*)) $\in$ *RulePermissions*

Figure 5.12: Z specification of Lemma 1.1

Figure 5.13 presents the Z specification for Lemma 1.2. It validates the permission of a user to access a resource in the mandatory policy. The resulting set *Accesses* contains *(user, resource)* pairs that are permitted by mandatory policies. Its Z specification is: if a user is member of *Users* set, a resource is member of *Resources* set, and if the user's clearance is greater than or equal to the resource's classification and resources' compartments are a subset of the users' compartments, then the pair *(user, resource)* must be member of the *Accesses* set, meaning that this user is allowed to have access to the resource.

theorem rule *Lemma12*
MandatorySchema
$\wedge$ *user* $\in$ *Users*
$\wedge$ *resource* $\in$ *Resources*
$\wedge$ *Clearance user* $\geqslant$ *Classification resource*
$\wedge$ *ResourceCompartments resource* $\subseteq$ *UserCompartments user*
$\Rightarrow$ (*user*, *resource*) $\in$ *Accesses*

Figure 5.13: Z specification of Lemma 1.2

Figure 5.14 presents the Z specification for Lemma 1.3. It validates the permission in security property policy. The resulting set *PossibleVias* contains *(resource, namedservice, path)* tuples that are permitted by security property policy. Its Z specification is: given a network path and a named service, if the resource is located at the destination node, the destination is the last node of the path, the named service is a named service associated to the resource, and the OSA of the pair *(namedservice, path)* are greater than or equal to the security requirement of the resource, then the pair *(resource, namedservice, path)* must be a member of *PossibleVias*.

theorem rule *Lemma13*
SecurityPropertySchema
$\wedge$ *destination* $\in$ *Locations*
$\wedge$ *resource* $\in$ *Resources*
$\wedge$ *path* $\neq \langle\rangle$
$\wedge$ *path* $\in$ *NetworkPaths*
$\wedge$ (*resource*, *destination*) $\in$ *ResourceLocatedAt*
$\wedge$ *destination* = *last path*
$\wedge$ *namedservice* $\in$ *NAMEDSERVICE*
$\wedge$ *namedservice* $\in$ *ResourceNamedServices resource*
$\wedge$ (*OSA* (*namedservice*, *path*), *SR resource*) $\in$ *GOSA*
$\Rightarrow$ (*resource, namedservice*, *path*) $\in$ *PossibleVias*

Figure 5.14: Z specification of Lemma 1.3

Figure 5.15 presents the specification for Lemma 1.4. If the tuple *((user, resource), (source, destination))* belongs to *RuleRulePermissions*, the *(user, resource)* pair belongs to *Accesses* set, and the *(service, path)* pair belongs to *PossibleVias* set, then the tuple *((user, resource), (source, destination), (namedservice, path))* must belong to *PossibleServiceAccesses*.

theorem rule *Lemma14*
PossibleServiceAccessesSchema
$\wedge$ ((*user*, *resource*), (*source*, *destination*)) $\in$ *RulePermissions*
$\wedge$ (*user*, *resource*) $\in$ *Accesses*
$\wedge$ (*resource, namedservice*, *path*) $\in$ *PossibleVias*
$\wedge$ *source* = *head path*
$\wedge$ *destination* = *last path*
$\Rightarrow$ ((*user*, *resource*), (*source*, *destination*), (*namedservice*, *path*))
$\in$ *PossibleServiceAccesses*

Figure 5.15: Z specification of Lemma 1.4

Figure 5.16 presents the specification for Lemma 2. It validates the computation of the *FeasibleServiceAccesses* set. This set contains the tuples that represent the service accesses permitted by discretionary, mandatory and security property policies, and that can be implemented by at least one firewall of the selected path. In Z, if the service access *((user*, *resource*), (*source*, *destination*), (*namedservice*, *path))* is member of *PossibleServiceAccess* and there is a firewall in the associated path that implements the named service, then this service access must be a member of *FeasibleServiceAccesses*.

theorem rule *Lemma2*
FeasibleServiceAccessesSchema
$\wedge$ *user* $\in$ *USER*
$\wedge$ *resource* $\in$ *RESOURCE*
$\wedge$ *namedservice* $\in$ *NAMEDSERVICE*
$\wedge$ *destination* $\in$ *LOCATION*
$\wedge$ *source* $\in$ *LOCATION*
$\wedge$ *path* $\in$ seq *NODE*
$\wedge$ ((*user*, *resource*), (*source*, *destination*), (*namedservice*, *path*))
$\in$ *PossibleServiceAccesses*
$\wedge$ *auxFirewall* $\in$ ran *path*
$\wedge$ (*auxFirewall*, *namedservice*) $\in$ *FirewallNamedServices*
$\Rightarrow$ ((*user*, *resource*), (*source*, *destination*), (*namedservice*, *path*))
$\in$ *FeasibleServiceAccesses*

Figure 5.16: Z specification of Lemma 2

Figure 5.17 presents the specification for Lemma 3. This lemma validates the attribution of the permissions to the firewalls. Each feasible service access requires the configuration of a number of firewalls (obtained from the associated path) in order to implement the access. This

validation includes the relaxation for the firewalls that doesn't recognize a given named service, where the specific named service is substituted by the generic named service.

theorem rule *Lemma3*
FirewallPermissionsSchema
$\wedge$ *firewall* $\in$ *Firewalls*
$\wedge$ *servs* $\in \mathbb{P}$ ((*USER* $\times$ *RESOURCE*) $\times$ (*LOCATION* $\times$ *LOCATION*) $\times$ *NAMEDSERVICE*)
$\wedge$ *servs*
= { *user: USER; resource: RESOURCE; source, destination: LOCATION;*
namedservice: NAMEDSERVICE; path: seq *NODE; fw_service: FIREWALL*
| ((*user*, *resource*), (*source*, *destination*), (*namedservice*, *path*))
$\in$ *FeasibleServiceAccesses*
$\wedge$ *firewall* $\in$ ran *path*
$\wedge$ ((*firewall*, *namedservice*) $\in$ *FirewallNamedServices*
$\wedge$ *fw_service* = *namedservice*
$\vee$ $\neg$ (*firewall*, *namedservice*) $\in$ *FirewallNamedServices*
$\wedge$ (*firewall*, *Generic*) $\in$ *FirewallNamedServices*
$\wedge$ *fw_service* = *Generic*)
• ((*user*, *resource*), (*source*, *destination*), *fw_service*) }
$\Rightarrow$ (*firewall*, *servs*) $\in$ *FirewallPermissions*

Figure 5.17: Z specification of Lemma 3

5.6 Demonstration of Theorem II

Figure 5.18 presents the specification for Lemma 4. This lemma validates that if a firewall permission exists, then there must be a corresponding feasible service access that has this firewall in the corresponding path.

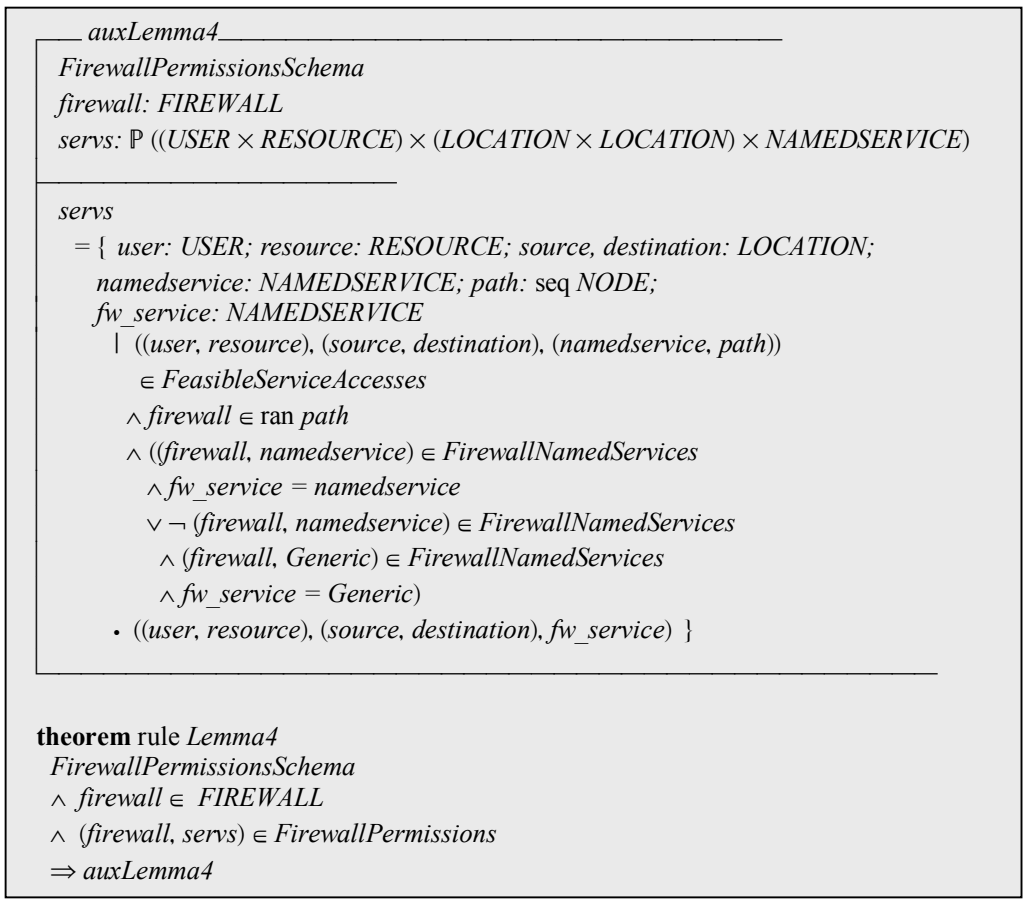

Figure 5.18: Z specification of Lemma 4

Figure 5.19 presents the specification for Lemma 5. It validates that the feasible service accesses corresponds to a permission that is allowed by discretionary, mandatory and security property models and that there is at least one firewall in the associated path that implements the named service. In Z, if a service access is a member of *FeasibleServiceAccesses*, then this service access must be a member of *PossibleServiceAccesses* and there must be a firewall in the associated path that implements the named service.

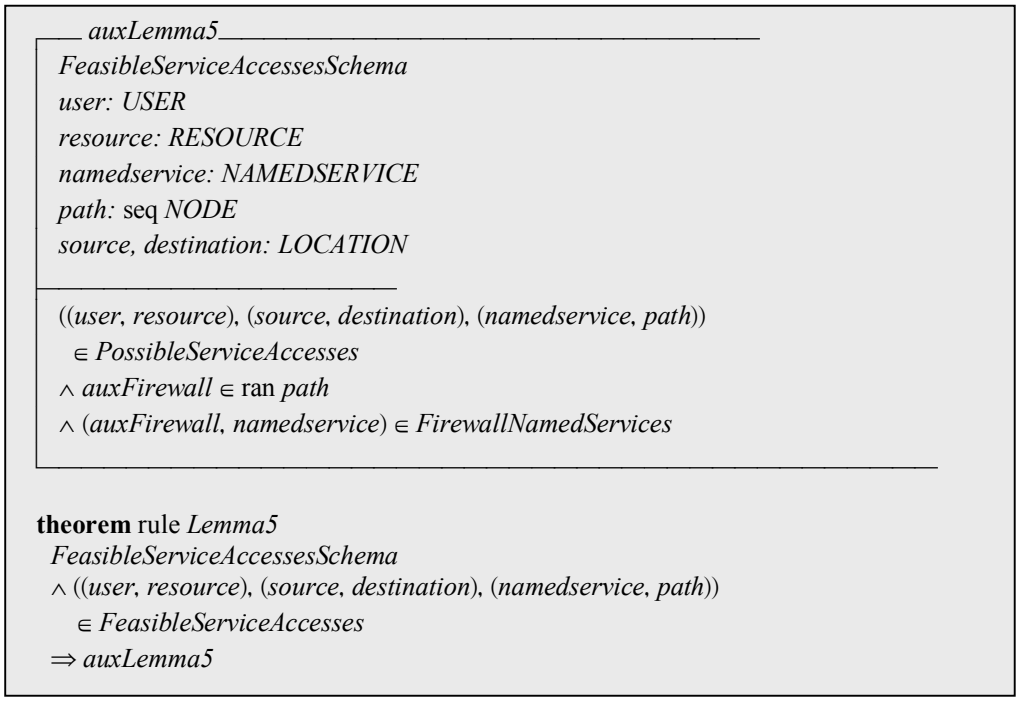

Figure 5.19: Z specification of Lemma 5

Lemma 6 validates that the possible service accesses corresponds to a permission that is allowed by discretionary, mandatory and security property models. This lemma was split into 3 lemmas. Figure 5.20 presents the specification for Lemma 6.1, which validates that the possible service access must be permitted by the discretionary policy. If the service access belong to possible service access set, then the user, resource, source and destination must be specified by the rule (i.e., the tuple *((user, resource), (source, destination))* must belong to *RulePermissions*) and the user and source must be related in *UserLocatedAt* set and resource and destination must be related in *ResourceLocatedAt* set.

auxLemma61

DiscretionarySchema
user: USER
resource: RESOURCE
source, destination: LOCATION

$user \in users$
$\wedge\ resource \in resources$
$\wedge\ source \in Locations$
$\wedge\ destination \in Locations$
$\wedge\ source \in sources$
$\wedge\ destination \in destinations$
$\wedge\ (user, source) \in UserLocatedAt$
$\wedge\ (resource, destination) \in ResourceLocatedAt$

theorem rule *Lemma61*
PossibleServiceAccessesSchema
$\wedge\ ((user, resource), (source, destination), (namedservice, path))$
$\in PossibleServiceAccesses$
$\Rightarrow auxLemma61$

Figure 5.20: Z specification of Lemma 6.1

Figure 5.21 presents the specification for Lemma 6.2, which validates that the possible service access must be permitted by the mandatory policy. If the service access belong to possible service access set, then the user clearance must be greater than or equal to the resource classification and the resource's compartments must be a subset of the user's compartments (i.e., the user must have the necessary compartments to access the resource).

Figure 5.22 presents the specification for Lemma 6.3, which validates that the possible service access must be permitted by the security property policy. If the service access belong to *PossibleServiceAccesses* set, then the resource must located at the destination, the named service must be associated to the resource and the overall security assumption of the service path must be greater than or equal to resource security requirement.

auxLemma62

PossibleServiceAccessesSchema
user: USER
resource: RESOURCE

Clearance user $\geqslant$ *Classification resource*
$\wedge$ *ResourceCompartments resource* $\subseteq$ *UserCompartments user*

theorem rule *Lemma62*
PossibleServiceAccessesSchema
$\wedge$ ((*user*, *resource*), (*source*, *destination*), (*namedservice*, *path*))
$\in$ *PossibleServiceAccesses*
$\Rightarrow$ *auxLemma62*

Figure 5.21: Z specification of Lemma 6.2

auxLemma63

PossibleServiceAccessesSchema
resource: RESOURCE
namedservice: NAMEDSERVICE
path: seq *NODE*
destination: LOCATION

---

path $\in$ *NetworkPaths*
$\wedge$ (*resource*, *destination*) $\in$ *ResourceLocatedAt*
$\wedge$ *namedservice* $\in$ *ResourceNamedServices resource*
$\wedge$ (*OSA* (*namedservice*, *path*), *SR resource*) $\in$ *OSA*

theorem rule *Lemma63*
PossibleServiceAccessesSchema
$\wedge$ ((*user*, *resource*), (*source*, *destination*), (*namedservice*, *path*))
$\in$ *PossibleServiceAccesses*
$\Rightarrow$ *auxLemma63*

Figure 5.22: Z specification of Lemma 6.3

5.7 Conclusion

This chapter presented a formal approach to the validation of the high-level to low-level translation process. In order to perform the validation, the algorithm was represented mathematically using the Z notation, and the two properties were selected to ensure the

correctness of the refinement: completeness and consistency. Completeness means that the desired behavior specified in an abstract level is completely implemented at lower level. Consistency means that all actions enabled at the lower level do not contradict the high level undesired behavior specification.

Two theorems that corresponds to these properties were defined, which was split in number of lemmas in order to make the proof process less complex. The theorems were represented in Z notation, using the Z/Eves tool.

The validation of the proofs was performed using the Z/Eves tool, which is a tool that allows the automated validation of goals. Z/Eves proofs work on a predicate called the *goal* (the defined theorem to be proved is the first goal); each step transforms the goal into a new goal that is equivalent. Transforming a goal to *true* thus completes a proof. The defined theorems were proved correctly.

Chapter 6

Performance Analysis and Evaluation

6.1 Introduction

The goal of this chapter is to perform the complexity analysis of the refinement algorithm, to determine its scalability by identifying the necessary resources in terms of fundamental operations and the size of input data. Commonly, the resources analyzed are time – the number of steps taken to solve an instance of the problem – and space – the amount of memory required by the algorithm. In this work we will follow the asymptotic notation, also called Big-Oh notation, to represent the time complexity. The refinement algorithm performs the following operations: computation of the paths between sources and destinations, computation of the permissions declared in each model (discretionary, mandatory and security property) and the computation of the resulting permissions.

The performance analysis demonstrates the behavior of the algorithm for the worst case and for the variation of the input variables, but doesn't give a real idea of the time spent for computation of real scenarios. This chapter also includes an evaluation of the implementation of the algorithm, in order to give real measures of processing time, considering scenarios that are more common in real situations.

6.2 Complexity Analysis

Considering a graph with E edges and V vertices, the complexity of the shortest path determination (by using Dijkstra's algorithm) from one vertex (source) to all other vertices (destinations) is [17]:

$$O(E + V \log V + kV) \tag{3}$$

In our network model, the vertices are locations that represent subnets and the edges are the connections (firewalls) between them. Considering the complexity of Dijkstra's algorithm, and considering *SN* subnets, *C* connections, and that any subnet can be the source of a path, the complexity for determining all paths from any source to all destinations is:

$$O\big(SN(C + SN \log SN + kSN)\big) =$$

$$O(SN \times C + SN^2 \log SN + SN^2) =$$

$$O(SN \times C + SN^2 \log SN) \tag{4}$$

A discretionary rule specifies a set of users that has access permission to a set of resources located at a set of destinations from a set of sources. Letting *U*, *R*, *S*, *D* to be the number of users, resources, sources and destinations referred in the currently evaluated rule, the complexity to evaluate the rule is:

$$O(U \times R \times S \times D) \tag{5}$$

Although this could seem to be somewhat complex, note that equation (5) just considers the elements addressed by a single rule. Therefore, even if the system has many registered users, resources and locations, each rule makes reference to just a few elements. For example, commonly a user has access to one or few resources, and resources are located at a few servers. Letting *Ru* to be the number of rules defined in the discretionary model, the complexity to compute the set of permissions defined by this model is:

$$O\big((U \times R \times S \times D) \times Ru\big) \tag{6}$$

The mandatory model represents user clearances and resource classifications. Again, considering *U* users and *R* resources, the complexity to evaluate the set of permissions defined by mandatory model is:

$$O(U \times R) \tag{7}$$

The security property model represents the security requirements assigned to resources and the security assumptions assigned to locations and services. Letting *Lp* to be the number of locations and firewalls within a path, *Sv* to be the number of services per resource and *R* to be the number of resources, the complexity to compute the set of permissions defined by security property model is:

$$O(Lp \times Sv \times R) \tag{8}$$

Equation 6 represents the complexity of analyzing the discretionary policy. This processing result in a number of permissions, that depends on the policy and the inventory information. Equations 7 and 8 represent the complexity of analyzing the mandatory and security property policies. The algorithm needs to evaluate the permissions obtained from the three models to compute the resulting set of permissions. The resulting complexity is:

$$O(((U \times R \times S \times D) \times Ru) \times (U \times R) \times (Lp \times Sv \times R)) =$$

$$O(U^2 \times R^3 \times S \times D \times Ru \times Lp \times Sv) \tag{9}$$

The overall complexity of the refinement algorithm is:

$$O(SN \times C + SN^2 \log SN) + O(U \times R \times S \times D \times Ru) + O(U \times R) +$$
$$O(Lp \times Sv \times R) + O(U^2 \times R^3 \times S \times D \times Ru \times Lp \times Sv) =$$

$$O(SN \times C + SN^2 \log SN) + O(U^2 \times R^3 \times S \times D \times Ru \times Lp \times Sv) \tag{10}$$

As before, Equation (10) represents the complexity for the worst case. For real situations, the following considerations apply:

(i) The time complexity in equation (10) considers all (user, resource) pair in the mandatory model, and all (resource, service, location) triple in the security property model. However, in the algorithm implementation, when processing the mandatory model only the users and resources allowed by discretionary model are taken into account. Also, when processing the security property model only the sources and destinations allowed by discretionary model and resources allowed by mandatory model are considered.

(ii) When computing the paths in the network, *SN* represents the number of subnets, independently of the number of hosts located in each subnet. This is because the path between a source and a destination network is the same for every host in the source network and every host in the destination network. In this way, although the complexity is high, isn't a major issue because the number of elements isn't high.

(iii) Each rule contains information about permissions given to users located at sources to access resources located at destinations. Considering that each user can be located at *X* different hosts, and that resources are located at *Y* different hosts, the complexity can be evaluated by:

$$O((X \times U \times Y \times R) \times Ru) \;=\; O((XY \times U \times R) \times Ru) \tag{11}$$

(iv) Another point is that not all rules include generic references, i.e., in most cases they reference only few users and resources. Considering that each rule references *W* users and *Z* resources, we have:

$$O((X \times Y \times W \times Z) \times Ru) \tag{12}$$

Then, the complexity for processing the discretionary rules can be more generally evaluated by the number of discretionary permissions per rule (*NDP*) multiplied by the number of rules:

$$O(NDP \times Ru) \tag{13}$$

(v) When processing the security property model, the algorithm must consider the resources and the services that they represent. Normally, there is a limited number of services per resource. Resources also include the information about its location (through the relationship Resource Located At). Letting *Srv* to be the number of services per resource, as the path includes the information of the destination (the last node of the path), the equation (6) can be evaluated as:

$$O(Lp \times Svr) \tag{14}$$

(vi) Each policy model represents a set of permissions. The number of permissions is very important for the refinement algorithm scalability, because their combination is an expensive operation, since permission only becomes a firewall rule if it is defined in the three models. Considering equations 7, 13 and 14, the Equation (9) can be alternatively represented by:

$$O(NDP \times Ru \times U \times R \times Lp \times Svr) \tag{15}$$

Equation 10 represents the complexity in terms of the basic objects. Taking in account the considerations presented above, the overall complexity can be represented as:

$$O(SN \times C + SN^2 \log SN) + O(NDP \times Ru \times U \times R \times Lp \times Svr) \tag{16}$$

Note that the equation 16 represents the same complexity as represented by equation 10. They differ in the input variables and some facts about the policies in real scenarios. The objective of equation 16 is to demonstrate the influence of each input element in the complexity of the algorithm for real situations: in equation 10, the number of resources seems to be very problematic (since it is R^3), but equation 16 demonstrates that its influence is R multiplied by the number of discretionary permissions per rule (the number of discretionary permissions will probably be lower the number of resources, since the rule restricts the allowed locations for resources). The same idea applies to the number of resources and services: equation 10 considers that the number of resources is directly multiplied by the number of services, which means that each resource includes all services (see equation 8). Equation 16 considers that the resources normally include only one or few services, using a average number of services per resources.

6.3 Evaluation of the Algorithm Implementation

The complexity analysis presents the behavior of the algorithm considering how the computation time changes with respect to the input data in the worst case. However, as it doesn't consider a real input data, it can not provide information about the real time necessary to process a particular input. The algorithm has been implemented in C# in order to perform its performance evaluation. We first performed a sensibility analysis experiment, followed by a load stress analysis on the critical component. The experiments have been done in a computer with Intel Core 2 Duo 2.53 GHz CPU and 2 GB RAM.

The sensibility analysis scenario starts with 10 users, 10 resources, 10 locations, 10 firewalls and 1 rule. The users and resources are equally distributed among the locations, that is, the users are located at one half and resources are located at the other half. In order to emulate the worst case condition, the scenario includes the rule "any user from any location is allowed to access any resource at any location". Note that, in this situation, increasing the number of users, resources or locations impacts directly in computation time, since the rules include every new element. The sensibility analysis variables are the number of each component type. The effect on the time to refine all rules has been observed separately by increasing each one individually.

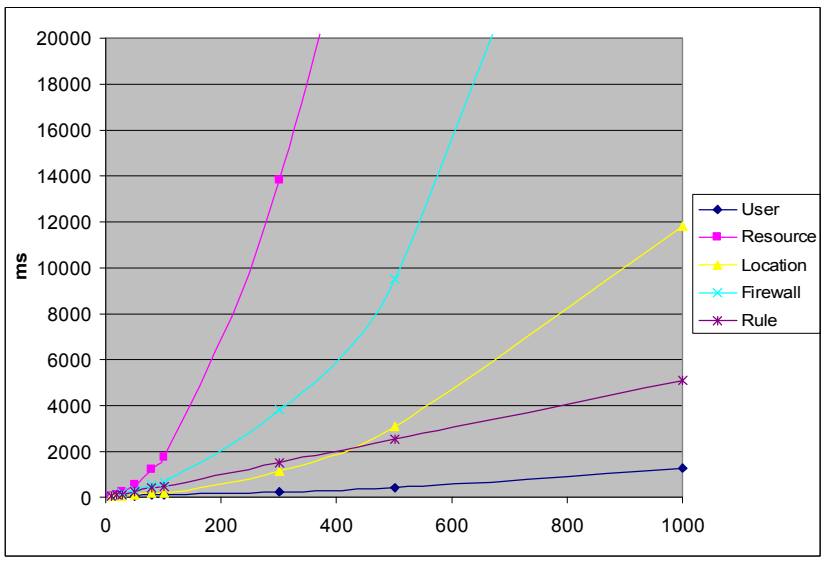

Figure 6.1: Sensibility analysis

Figure 6.1 shows the sensibility analysis results. It can be observed that the number of resources is the critical component, followed by of the number of firewalls. This result can be explained because the resource is the only element that is referenced in the three policy models. From the observed values, we have used the least squares method to fit the corresponding equations, as presented in Table 1:

Table 6.1: Time consuming and complexity equations for scenario 1

Variable	Equation (milliseconds)	Complexity
Resources	$0.1457R^2+1.8243R+65.1442$	$O(0.1457\times R^2)$
Firewalls	$0.0464F^2-3.8038F+331.1513$	$O(0.0464\times F^2)$
Locations	$0.0115L^2+0.2779L+50.6049$	$O(0.0115\times L^2)$
Users	$0.0008U^2+0.3904U+51.9187$	$O(0.0008\times U^2)$
Rules	$5.0986Ru-8.7489$	$O(5.0986\times Ru)$

As expected, it can be observed that the complexity analysis provides an upper bound for the processing time. In other words, in more realistic situations, the impact of the input size is smaller than that stated in the complexity equations. This can be verified by looking at each component equation individually. For example, in equation (10), the complexity equation corresponding to the number of resources is $O(R^3)$, but the implementation evaluation shows a $O(R^2)$ complexity.

The second scenario is intended to simulate a realistic situation. It starts with 500 users, 200 resources, 100 locations, 100 firewalls and 10 rules. Because the scenario is supposed to emulate a realistic situation, the rules are different from the rules used in the first analysis. In this case, the rules consider only 5 users and 1 resource, instead of any user from any location in the sensitivity analysis. For example, “users U1, U2, U3, U4 and U5 from any location are allowed to access resource R1 at any location”. As both users and resources have its locations constrained by User Located At and Resource Located At, it is not necessary to specify these restrictions in the rule.

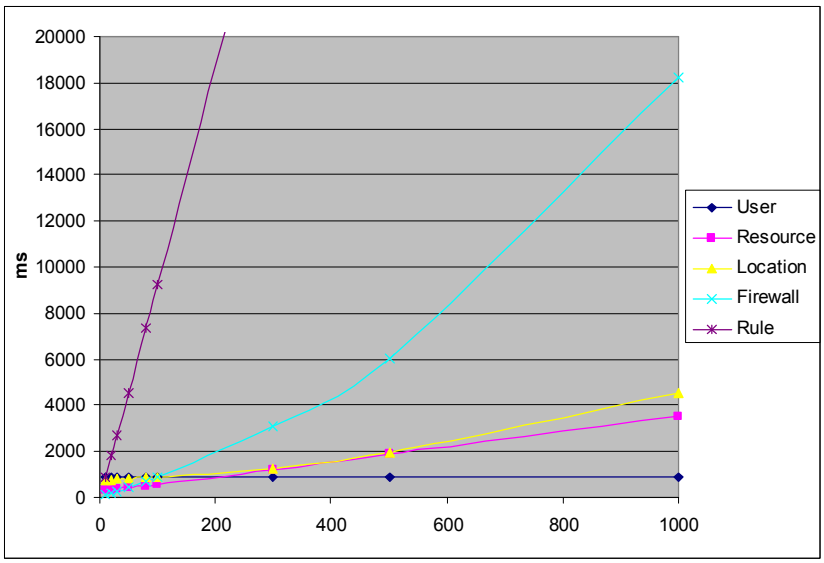

Figure 6.2: Performance evaluation

Figure 6.2 shows the results for the load stress scenario. It can be observed that increasing the number of users doesn't cause any impact to the computation time. Increasing resources causes a minimum impact, mainly because the time necessary to search the classification and security requirement of the resources and the security assumptions of the named services. This is because in the simulated scenario the number of services is dependent on the number of resources. It is important to note that the small impact observed for users and resources is because the inclusion of a new user or new resource doesn't imply in a new service access. Another reason for the lower impact for users and resources is because the rule is very specific, unlike to the first scenario, specifying only 5 users and 1 resource. The equations obtained for scenario 2 are presented in Table 2.

Table 6.2: Time consuming and complexity equations for scenario 2

Variable	Equation (milliseconds)	Complexity
Resources	$3.2476R+258.5567$	$O(3.2476\times R)$
Firewalls	$0.0122F^2+5.9329F-106.6788$	$O(0.0122\times F^2)$
Locations	$0.0029L^2+0.8776L+769.585$	$O(0.0029\times L^2)$
Users	$0.0232U+890.542$	$O(0.0232\times U)$
Rules	$94.4009Ru-168.6364$	$O(94.4009\times Ru)$

The elements that cause more impact in this scenario are firewalls, followed by locations and rules. As the number of firewalls and locations isn't very large in real networks, the element that is more important to be analyzed is the number of rules. With this in mind, we evaluated the time necessary to process 500 users, 200 resources, 100 locations, 100 firewalls and 10,000 rules. The processing time for this scenario was approximately 16 minutes.

6.4 Conclusion

This chapter demonstrated the complexity analysis and presented a performance evaluation of the algorithm implementation.

The theoretical analysis demonstrated that the complexity formula is:

$$O(SN \times C + SN^2 \log SN) + O(U^2 \times R^3 \times S \times D \times Ru \times Lp \times Sv)$$

This formula has two parts: the first part demonstrated the complexity of the path computing, that is the complexity of Dijkstra's algorithm. The second part corresponds to the processing of the three policy models.

This formula represents the worst case for the algorithm: every rule referencing all users, resources, sources and destinations. Since in real scenarios the rules references few users, resources, sources and destinations, the formula can be represented as:

$$O(SN \times C + SN^2 \log SN) + O(NDP \times Ru \times U \times R \times Lp \times Svr)$$

The latter formula represents better real situations. It can be noticed that the complexity concerning the discretionary rule is represented by $O(NDP \times Ru)$, which corresponds to the average number of permissions per rule multiplied by the number of rules.

In order to measure the real time necessary to process the policies, some scenarios were evaluated using the implementation of the algorithm in C#. The values obtained within these scenarios were used to construct separate formulas for each input variable.

The obtained formulas showed that the processing time increases linearly for the increasing of number of users, resources and rules, and increases quadraticly for locations and firewalls.

Chapter 7

Example

7.1 Introduction

According to our approach, only the framework has the credentials necessary to create and modify rules or scripts in the firewalls. The policy administrators need to use the framework in order to manage the security policy. To illustrate the use of the framework we consider the scenario depicted in Figure 7.1. This sample scenario represents a fictitious corporate network, which is representative of a real corporate network. The network is subdivided into 5 subnets, and includes a connection to the Internet. There are 4 firewalls connecting the Internet and subnets: firewall 1 connects Internet, DMZ and Commercial subnets; firewall 2 connects Commercial and Administration subnets; firewall 3 connects Commercial and Engineering subnets; and firewall 4 connects Engineering and Production subnets. Firewall 1 is a VPN server, i.e., the users at the Internet must connect to the VPN server and the IP addresses it assigns correspond to external users. Guest is the only external user that doesn't connect to the VPN server. The example illustrates the information registered in the inventory, the mandatory, discretionary and security property policies and how they are processed. It also illustrates the operation performed at each step of the algorithm. For sake of simplicity, the security assumption and requirement vectors have been reduced to two dimensions: [confidentiality, accountability].

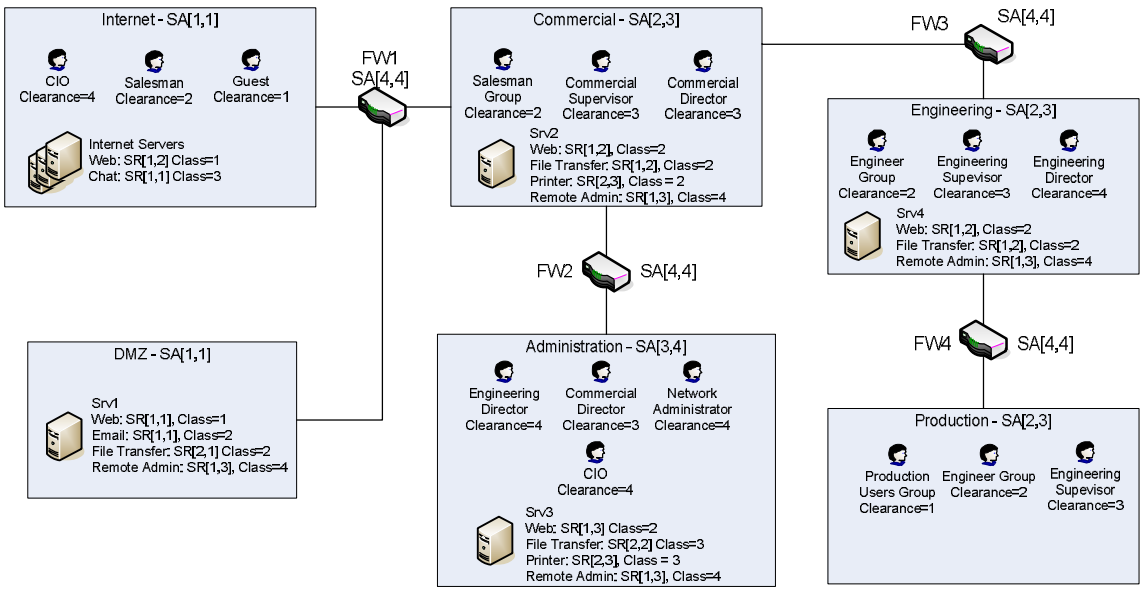

Figure 7.1: Example network

All subnets have one server, except the production subnet. There are four user groups (sales, engineering, production and guest) and six individual users (CIO, commercial supervisor, commercial director, engineering supervisor, engineering director, and network administrator). The users are identified by their locations. In real situations they are translated to the corresponding IP address when the firewall rules are created, but in this example we will keep the host names for better legibility. There are 4 servers inside the network and a group of servers at the Internet, where the resources are available for user access. For this example network, Figure 7.2, 7.3 and 7.4 presents the information represented in the inventory, Figure 7.5 and Figure 7.6 represent the mandatory policy and Figure 7.7, 7.7 and 7.8 represent the security property policy. The discretionary policy is presented in the table 7.1. Appendix A presents the complete file with the Inventory and the three policy models.

7.2 High Level

Figure 7.2, 7.3 and 7.4 presents the information in the Inventory. Figure 7.2 represents the list of users (or user groups) and their locations. The locations may represent hosts or subnets, which are translated to the corresponding IP address/mask. The names of the subnets in parenthesis are only present to help the understanding. Guest represents a group of users, any user that is located at the Internet (not connected through the VPN).

```
user_located_at(CIO, [VPN1,HostA4]).
user_located_at(Salesman, [VPN2,VPN3,VPN4,HostH1]).
user_located_at(CommercialSupervisor, [HostC2]).
user_located_at(CommercialDirector, [HostC3,HostA2]).
user_located_at(Engineer, [HostE1,HostP2]).
user_located_at(EngineeringSupervisor, [HostE2,HostP3]).
user_located_at(EngineeringDirector, [HostA1,HostE3]).
user_located_at(ProductionUser, [HostP1]).
user_located_at(NetworkAdministrator, [HostA3]).
user_located_at(Guest, [Internet]).
```

Figure 7.2: Users and its locations (Inventory)

Figure 7.3 represents the list of named services. Note that in this example we use the name of protocols for some named services, but this is not mandatory. The Named Service Library must implement all named services supported by a firewall as registered in the Inventory. In this example all the firewalls implement the named services represented in Figure 7.3.

```
named_service(http).
named_service(https).
named_service(smtp).
named_service(pop3).
named_service(imap).
named_service(ftp).
named_service(ipp).
named_service(msnMessenger).
named_service(msnFileTransfer).
named_service(vnc).
```

Figure 7.3: Available named services (Inventory)

Figure 7.4 represents the resources available in the network, its locations and the named services associated with them. In this example, the named services have names similar to the names of the protocols they represent, but this is not mandatory.

```
resource(WebInternet, [http, https]).
resource(Chat, [msnMessenger,msnFileTransfer]).
resource(WebDMZ, [http, https]).
resource(EmailDMZ, [smtp, pop3, imap]).
resource(FileTransferDMZ, [ftp]).
resource(WebCommercial, [http, https]).
resource(FileTransferCommercial, [ftp]).

resource(PrinterCommercial, [ipp]).
resource(WebAdmin, [http, https]).
resource(FileTransferAdmin, [ftp]).
resource(PrinterAdmin, [ipp]).
resource(WebEngineering, [http, https]).
resource(FileTransferEngineering, [ftp]).

resource(RemoteAdmin, [vnc]).
```

```
resource_located_at(WebInternet, [Internet]).
resource_located_at(Chat, [Internet]).
resource_located_at(WebDMZ, [Srv1]).
resource_located_at(EmailDMZ, [Srv1]).
resource_located_at(FileTransferDMZ, [Srv1]).
resource_located_at(WebCommercial, [Srv2]).
resource_located_at(FileTransferCommercial,
                    [Srv2]).
resource_located_at(PrinterCommercial, [Srv2]).
resource_located_at(WebAdmin, [Srv3]).
resource_located_at(FileTransferAdmin, [Srv3]).
resource_located_at(PrinterAdmin, [Srv3]).
resource_located_at(WebEngineering, [Srv4]).
resource_located_at(FileTransferEngineering,
                    [Srv4]).
resource_located_at(RemoteAdmin,
                    [Srv1, Srv2, Srv3, Srv4]).
```

Figure 7.4: Available resources and corresponding locations (Inventory)

Figure 7.5 and Figure 7.6 present the Mandatory Policy information. Figure 7.5 shows the clearance and the compartments of each user/user group. Note that guest users have the lowest clearance, while network administrator and CIO have the greatest clearances. Note also that although CIO and Network Administrator have the same clearance, they have different compartments, corresponding to their need-to-know information.

```
clearance(CIO, 4, [Commercial, Administration, Engineering, Production]).
clearance(Salesman, 2, [Commercial]).
clearance(CommercialSupervisor, 3, [Commercial]).
clearance(CommercialDirector, 3, [Commercial, Administration]).
clearance(Engineer, 2, [Administration, Engineering]).
clearance(EngineeringSupervisor, 3, [Engineering]).
clearance(EngineeringDirector, 4, [Administration, Engineering]).
clearance(ProductionUser, 1, []).
clearance(NetworkAdministrator, 4, [NetworkAdministration]).
clearance(Guest, 1, []).
```

Figure 7.5: Users' clearance (Mandatory Policy)

Figure 7.6 shows the resources' classification. The Internet servers and the Srv1 (DMZ) have the mandatory classification 1, as they can be accessed by any user. On the other hand, Remote Administration has a high classification, since this is a sensitive resource and must be used only by few users. Note that, although the CIO has clearance level sufficient to have access

to RemoteAdmin resource, this access will not be permitted since CIO isn't associated to NetworkAdministration compartment.

```
classification(WebInternet, 1, []).
classification(Chat, 3, []).
classification(WebDMZ, 1, []).
classification(EmailDMZ, 2, []).
classification(FileTransferDMZ, 2, []).
classification(RemoteAdmin, 4, [NetworkAdministration]).
classification(WebCommercial, 2, []).
classification(FileTransferCommercial, 2, [Commercial]).
classification(PrinterCommercial, 2, []).
classification(WebAdmin, 2, []).
classification(FileTransferAdmin, 3, [Administration]).
classification(PrinterAdmin, 3, [Administration]).
classification(WebEngineering, 2, []).
classification(FileTransferEngineering, 2, [Engineering]).
```

Figure 7.6: Resources classification (Mandatory Policy)

Figure 7.7, 7.7 and 7.8 present the Security Property Policy information. Figure 7.7 shows the resources' Security Requirements. Resources at DMZ and Internet have low security classes, while resources at internal servers have high.

```
security_requirement(WebInternet, [1,2]).
security_requirement(Chat, [1,1]).
security_requirement(WebDMZ, [1,1]).
security_requirement(FileTransferDMZ, [2,1]).
security_requirement(EmailDMZ, [1,1]).
security_requirement(RemoteAdmin, [1,3]).
security_requirement(WebCommercial, [1,2]).
security_requirement(FileTransferCommercial, [1,2]).
security_requirement(PrinterCommercial, [2,3]).
security_requirement(WebAdmin, [1,3]).
security_requirement(FileTransferAdmin, [2,2]).
security_requirement(PrinterAdmin, [2,3]).
security_requirement(WebEngineering, [1,2]).
security_requirement(FileTransferEngineering, [1,2]).
```

Figure 7.7: Resources Security Requirements (Security Property Policy)

Figure 7.8 represents Security Assumptions of named services. Note that https have a stronger confidentiality level when compared to http.

```
security_assumption(http, [1,1]).
security_assumption(https, [4,1]).
security_assumption(smtp, [2,1]).
security_assumption(pop3, [2,1]).
security_assumption(imap, [2,1]).
security_assumption(ftp, [2,1]).
security_assumption(ipp, [2,1]).
security_assumption(msnMessenger, [2,1]).
security_assumption(msnFileTransfer, [1,1]).
security_assumption(vnc, [1,1]).
```

Figure 7.8: Services Security Assumption (Security Property Policy)

Figure 7.9 represents the Security Assumptions assigned to locations and firewalls. Internet and DMZ have the lowest security class, since they are very insecure. The firewalls have the highest security classes, because we are considering that they are safe. Note that, although the accesses coming from VPNs are initiated from the Internet, the corresponding SA is high since the users are authenticated and the connection is ciphered.

```
security_assumption(Internet, [1,1]).
security_assumption(DMZ, [1,1]).
security_assumption(Commercial, [1,3]).
security_assumption(Administration, [1,4]).
security_assumption(Engineering, [1,3]).
security_assumption(Production, [1,3]).
security_assumption(VPN1, [3,3]).
security_assumption(VPN2, [3,3]).
security_assumption(VPN3, [3,3]).
security_assumption(VPN4, [3,3]).

security_assumption(fw1, [4,4]).
security_assumption(fw2, [4,4]).
security_assumption(fw3, [4,4]).
security_assumption(fw4, [4,4]).
```

Figure 7.9: Locations and Firewalls Security Assumption (Security Property Policy)

Table 7.1 presents the list of discretionary rules created for this example. The table presents the rule in the format used by the algorithm and the description of the rule.

Table 7.1: Discretionary Policy

Num	Rule	Description
1	rule([any], [WebDMZ], [any], [Srv1], permit).	Any user is allowed to access Web at Srv1 (DMZ) from any location.
2	rule([insiderUsers], [EmailDMZ], [any], [Srv1], permit).	Company users are allowed to access Email at Srv1 (DMZ) from any location.
3	rule([NetworkAdministrator], [FileTransferDMZ], [any], [Srv1], permit).	Network Administrator is allowed to access File Transfer at Srv1 (DMZ) from any location.
4	rule([NetworkAdministrator], [RemoteAdmin], [any], [any], permit).	Network Administrator is allowed to access Remote Admin at any location from any location.
5	rule([NetworkAdministrator], [FileTransferAdmin,FileTransferCommercial], [any], [any], permit).	Network Administrator is allowed to access File Transfer at Srv3 (Administration) and Srv2 (Commercial) from any location.
6	rule([Salesman,CommercialSupervisor], [WebCommercial,PrinterCommercial], [any], [Srv2], permit).	Salesman and Commercial Supervisor are allowed to access Web and Printer at Srv2 (Commercial) from any location.
7	rule([Engineer,EngineeringSupervisor, EngineeringDirector], [WebEngineering, FileTransferEngineering], [any], [Srv4], permit).	Engineer, Engineering Supervisor and Engineering Director are allowed to access Web and File Transfer at Srv4 (Engineering) from any location.
8	rule([CIO,EngineeringDirector,CommercialDirector], [WebAdmin,PrinterAdmin], [any], [Srv3], permit).	CIO and Directors are allowed to access Web and Printer at Srv3 (Administration) from any location.
9	rule([NetworkAdministrator], [PrinterAdmin], [Administration], [Srv3], permit).	Network Administrator is allowed to access the Printer at Administration subnet from Administration subnet.
10	rule([CIO,EngineeringDirector,CommercialDirector], [Chat], [any], [Internet], permit).	CIO and Directors are allowed to access Chat at Internet Servers from any location.
11	rule([any], [any], [any], [Internet], permit).	Any user is allowed to access Internet Servers from any location.
12	rule([Salesman], [PrinterAdmin], [any], [Srv3], permit).	Salesman is allowed to access Printer at Srv3 (Administration) from any location.

Table 7.1: Discretionary Policy (Cont.)

13	rule([ProductionUser], [WebEngineering], [any], [Srv4], permit).	Production Users are allowed to access Web at Srv4 (Engineering) from any location.
14	rule([CIO], [WebAdmin], [Internet], [Administration], permit).	CIO is allowed to access resource Web at Srv3 (Administration) from the Internet.

The above describe scenario was used as input for an implementation of the refinement algorithm, producing the corresponding firewall rules. For example, considering the mandatory, security properties models and rule 1, the following firewall rules for firewall FW3 are created:

- Permit engineering director, engineering supervisor and engineers to access the Web resource (services http and https) at Srv1 from their hosts at engineering subnet;
- Permit engineering supervisor, engineers and production users to access the Web resource (services http and https) at Srv1 from their hosts at production subnet.

Note that, although rule 1 specifies any user, the firewall rules generated by the algorithm are very specific in terms of the hosts/users that are allowed to access the Srv1. In this case, the algorithm has obtained the information about which users are allowed to use the Engineering and Production subnets from the *UserLocatedAt* inventory class.

It is possible that one discretionary rule doesn't generate a firewall rule. This situation can occur in the following cases:

- There is no firewall between source and destination: for example, rule 9 specify that the network administrator can only access the resource Printer at Srv3 from Administration subnet. Since there isn't a firewall to be configured, the rules aren't generated. This can be solved by including a firewall between source and destination, of by using a personal firewalls in the hosts [75][76].
- Mandatory policy denies the access: for example, rule 12 specify that salesman are allowed to access printer at Srv3, but the salesman clearance is not sufficient to grant this access;
- Security property policy denies the access: for example, rule 14 specifies that the CIO is allowed to access resource Web at Srv3 from the Internet, but the OSA of the path

Internet-FW1-Commercial-FW2-Administration combined with any named service is weaker than the SR of the Web resource at Srv3.

In order to demonstrate the steps performed by the algorithm, table 7.2 presents the results produced at each step. For sake of simplicity, we present only the results for discretionary rule 8.

Table 7.2: Results of each step performed by the algorithm for rule 8

Step	**Results**
1	Users: [CIO,EngineeringDirector,CommercialDirector] Sources: [HostA4,VPN1, HostE3, HostA1, HostA2, HostC3]
2	Resources: [WebAdmin, PrinterAdmin] Destinations: [Srv3]
3	(CIO, WebAdmin) (CIO, PrinterAdmin) (EngineeringDirector, WebAdmin) (EngineeringDirector, PrinterAdmin) (CommercialDirector, WebAdmin) (CommercialDirector, PrinterAdmin)
4	[VPN1, fw1,Commercial, fw2, Srv3] [HostE3, fw3, Commercial, fw2, Srv3] [HostC3, fw2, Srv3]
5	(http, [HostE3, fw3,Commercial,fw2, Srv3]) SR: [1,3] OSA: [1,3] (https, [HostE3, fw3,Commercial,fw2,Srv3]) SR: [1,3] OSA: [4,3] (ipp, [HostE3, fw3,Commercial,fw2,Srv3]) SR: [2,3] OSA: [2,3] (http, [HostC3, fw2,Srv3]) SR: [1,3] OSA: [1,3] (https, [HostC3, fw2,Srv3]) SR: [1,3] OSA: [4,3] (ipp, [HostC3, fw2,Srv3]) SR: [2,3] OSA: [2,3]
6	(http, [HostE3, fw3,Commercial,fw2,Srv3]) SR: [1,3] OSA: [1,3] (https, [HostE3, fw3,Commercial,fw2,Srv3]) SR: [1,3] OSA: [4,3] (ipp, [HostE3, fw3,Commercial,fw2,Srv3]) SR: [2,3] OSA: [2,3] (http, [HostC3, fw2,Srv3]) SR: [1,3] OSA: [1,3] (https, [HostC3, fw2,Srv3]) SR: [1,3] OSA: [4,3] (ipp, [HostC3, fw2,Srv3]) SR: [2,3] OSA: [2,3]
7	permission(fw2,[8],[EngineeringDirector,CommercialDirector], [WebAdmin,PrinterAdmin],[HostE3,HostC3],[Srv3],[http,https,ipp]). permission(fw3,[8],[EngineeringDirector], [WebAdmin,PrinterAdmin],[HostE3],[Srv3],[http,https,ipp]).

Steps 1 and 2 select the set of users, sources, resources and destinations that are permitted by the discretionary rule. Step 3 selects the (user, resource) pairs that are permitted by mandatory policy. Step 4 computes the paths between the sources and destinations obtained in steps 1 and 2 using the Dijkstra algorithm. Step 5 computes the possible service accesses, calculating the OSA and selecting the named services and paths which OSA is greater than or equal to the resource's security requirement. Step 6 computes the feasible service accesses, where at least one firewall in the associated path implements the named service. Finally, step 7 computes the firewall permissions, resulting in the permissions that must be configured in each firewall of the network.

Table 7.3 presents the resulting optimized firewall permissions obtained from the processing of the algorithm on the network example.

Table 7.3: Optimized Firewall Permissions for Example Network

FW	**Optimized Firewall Permissions**	
FW1	permission(fw1,[HostA3],[Srv1],[http,https,smtp,pop3,imap,ftp,vnc]).	(a)
	permission(fw1,[HostE3,HostA1,HostP2,HostE1,HostA2,HostC3,HostA4,VPN1, HostP3,HostE2,VPN2,VPN3,VPN4,HostC2],[Srv1],[http,https,smtp,pop3,imap]).	(b)
	permission(fw1,[Internet,HostP1],[Srv1],[http,https]).	(c)
FW2	permission(fw2,[HostA3],[Srv1],[http,https,smtp,pop3,imap,FTP,vnc]).	(d)
	permission(fw2,[HostA1,HostA2,HostA4],[Srv1],[http,https,smtp,pop3,imap]).	(e)
	permission(fw2,[HostA3],[Srv4,Srv2],[vnc]).	(f)
	permission(fw2,[HostA1],[Srv4],[http,https,ftp]).	(g)
	permission(fw2,[HostE3,HostC3],[Srv3],[http,https,ipp]).	(h)
FW3	permission(fw3,[HostE3,HostP2,HostE1,HostP3,HostE2],[Srv1],[http,https,smtp, pop3,imap]).	(i)
	permission(fw3,[HostP1],[Srv1],[http,https]).	(j)
	permission(fw3,[HostA3],[Srv4],[vnc]).	(k)
	permission(fw3,[HostA1],[Srv4],[http,https,ftp]).	(l)
	permission(fw3,[HostE3],[Srv3],[http,https,ipp]).	(m)
FW4	permission(fw4,[HostP2,HostP3],[Srv1],[http,https,smtp,pop3,imap]).	(n)
	permission(fw4,[HostP1],[Srv1],[http,https]).	(o)
	permission(fw4,[HostP2,HostP3],[Srv4],[http,https,ftp]).	(p)

The firewalls are configured so that the users can access the resources correctly. For example, FW1 is configured to permit the access from the internal hosts and from Internet to Srv1 server (a)(b)(c). Note that the firewalls FW2, FW3 and FW4 also must be configured to permit this access from internal subnets. FW2 is configured with scripts (d) and (e) that together with (b) allows the access from the hosts of Administration subnet. FW3 is configured with scripts (i) and (j) to allow the access to Srv1 from Engineering and Production subnets, and FW4

is configured with scripts (n) and (o) to allow the access to Srv1 from Production subnet. Note that the scripts (b), (i) and (n) allows that HostP2 and HostP3 have access to Srv1 using http, https, smtp, pop3 and imap, while scripts (c), (j) and (o) allows that HostP1 have access to Srv1 using only http and https.

Other example is the access of the Network Administrator from HostA3 to servers Srv1, Srv2, Srv3 and Srv4 using the named service VNC. Note that the scripts (a) and (d) allows the access to Srv1 from HostA3, script (f) allows the access to Srv2 from HostA3 and scripts (f) and (k) allows the access to Srv4. As mentioned before, no script is generated for the access to Srv3 from HostA3, since there is no firewall between HostA3 and Srv3.

The complete example processing time was 33.17ms, resulting in 164 permissions that were grouped into 16 optimized permissions and approximately 700 script rules.

7.3 Low Level

The last step of the algorithm is the generation of the scripts or rules for firewall configuration. The example presented in this chapter supposes that the firewalls are based on Linux IPTables. To demonstrate how firewalls with different characteristics can be configured, we also consider that the firewalls includes the L7-filter [18], a software package that provides a classifier for Linux's Netfilter subsystem. L7-filter allows categorizing Internet Protocol packets based on their application layer data. For instance, the Telnet service could be defined as TCP with destination port 23 and any source port, or, when using the L7-filter, it could be defined as a search pattern [19] independently of port numbers. As second example, considers the http protocol. It can be identified by the destination port 80 or by inspecting the http headers. Simply allowing connections to server port 80 allows the http connections to work properly, but also allows other protocols to be used at this port. For instance, it would be possible to use this port for P2P connections, but if the implementation checks the http headers, only valid http connections would be allowed.

Figure 7.10 presents an example of configuration script for IPTables with L7-filter for the http protocol. The L7-filter inspects the first 10 packets or 2k bytes by default, before deciding if the connection matches or not. It marks the connections it is still trying to match as "unset". The first few packets of all TCP connections will be marked as unset. After the first 10 packets or 2k bytes of a connection have been inspected, if the connection is still marked as unset, then it marks the connection as "unknown".

The L7-filter documentation of the HTTP pattern specifies that it matches the HTTP response. In order to allow the L7-filter to inspect a HTTP response, it is necessary to create rules that permit the TCP handshake packets and HTTP requests. To implement this, we use the "unset" mark (lines 3 and 4 of the script). Due to fact that the first packets of any connection match "unset", it is important to specify the server port 80.

```
iptables -A FORWARD -s 10.1.4.3/32 -d 10.1.3.254/32 -p tcp --dport 80 -m layer7 --l7proto http
        -m state --state NEW,ESTABLISHED -j ACCEPT
iptables -A FORWARD -s 10.1.3.254/32 -d 10.1.4.3/32-p tcp --sport 80 -m layer7 --l7proto http
        -m state --state ESTABLISHED -j ACCEPT
iptables -A FORWARD -s 10.1.4.3/32 -d 10.1.3.254/32 -p tcp --dport 80 -m layer7 --l7proto unset
        -m state --state NEW,ESTABLISHED -j ACCEPT
iptables -A FORWARD -s 10.1.3.254/32 -d 10.1.4.3/32 -p tcp --sport 80 -m layer7 --l7proto unset
        -m state --state ESTABLISHED -j ACCEPT
```

Figure 7.10: IPTables with L7-filter Configuration Rules

Considering the resulting optimized firewall permissions for the rule 8 in the Table 7.2, table 7.4 presents the generated scripts from the template.

Table 7.4: IPTables with L7-Filter scripts generated for rule 8

Firewall	**Script**
FW2	iptables -F FORWARD iptables -F INPUT iptables -F OUTPUT iptables -P FORWARD DROP #User: [EngineeringDirector,CommercialDirector] Source: [HostE3,HostC3] Resource: [WebAdmin,PrinterAdmin] Dest: [Srv3] Service: [http,https,ipp] iptables -A FORWARD -s 10.1.4.3/32 -d 10.1.3.254/32 -p tcp --dport 80 -m layer7 --l7proto http -m state --state NEW,ESTABLISHED -j ACCEPT iptables -A FORWARD -s 10.1.3.254/32 -d 10.1.4.3/32 -p tcp --sport 80 -m layer7 --l7proto http -m state --state ESTABLISHED -j ACCEPT iptables -A FORWARD -s 10.1.4.3/32 -d 10.1.3.254/32 -p tcp --dport 80 -m layer7 --l7proto unset -m state --state NEW,ESTABLISHED -j ACCEPT iptables -A FORWARD -s 10.1.3.254/32 -d 10.1.4.3/32 -p tcp --sport 80 -m layer7 --l7proto unset -m state --state ESTABLISHED -j ACCEPT iptables -A FORWARD -s 10.1.2.3/32 -d 10.1.3.254/32 -p tcp --dport 80 -m layer7 --l7proto http -m state --state NEW,ESTABLISHED -j ACCEPT iptables -A FORWARD -s 10.1.3.254/32 -d 10.1.2.3/32 -p tcp --sport 80 -m layer7 --l7proto http -m state --state ESTABLISHED -j ACCEPT iptables -A FORWARD -s 10.1.2.3/32 -d 10.1.3.254/32 -p tcp --dport 80 -m layer7 --l7proto unset -m state --state NEW,ESTABLISHED -j ACCEPT iptables -A FORWARD -s 10.1.3.254/32 -d 10.1.2.3/32 -p tcp --sport 80 -m layer7 --l7proto unset -m state --state ESTABLISHED -j ACCEPT iptables -A FORWARD -s 10.1.4.3/32 -d 10.1.3.254/32 -p tcp --dport 443 -m layer7 --l7proto ssl -m state --state NEW,ESTABLISHED -j ACCEPT iptables -A FORWARD -s 10.1.3.254/32 -d 10.1.4.3/32 -p tcp --sport 443 -m layer7 --l7proto ssl

	-m state --state ESTABLISHED -j ACCEPT iptables -A FORWARD -s 10.1.4.3/32 -d 10.1.3.254/32 -p tcp --dport 443 -m layer7 --l7proto unset -m state --state NEW,ESTABLISHED -j ACCEPT iptables -A FORWARD -s 10.1.3.254/32 -d 10.1.4.3/32 -p tcp --sport 443 -m layer7 --l7proto unset -m state --state ESTABLISHED -j ACCEPT iptables -A FORWARD -s 10.1.2.3/32 -d 10.1.3.254/32 -p tcp --dport 443 -m layer7 --l7proto ssl -m state --state NEW,ESTABLISHED -j ACCEPT iptables -A FORWARD -s 10.1.3.254/32 -d 10.1.2.3/32 -p tcp --sport 443 -m layer7 --l7proto ssl -m state --state ESTABLISHED -j ACCEPT iptables -A FORWARD -s 10.1.2.3/32 -d 10.1.3.254/32 -p tcp --dport 443 -m layer7 --l7proto unset -m state --state NEW,ESTABLISHED -j ACCEPT iptables -A FORWARD -s 10.1.3.254/32 -d 10.1.2.3/32 -p tcp --sport 443 -m layer7 --l7proto unset -m state --state ESTABLISHED -j ACCEPT iptables -A FORWARD -s 10.1.4.3/32 -d 10.1.3.254/32 -p tcp --dport 631 -m layer7 --l7proto ipp -m state --state NEW,ESTABLISHED -j ACCEPT iptables -A FORWARD -s 10.1.3.254/32 -d 10.1.4.3/32 -p tcp --sport 631 -m layer7 --l7proto ipp -m state --state ESTABLISHED -j ACCEPT iptables -A FORWARD -s 10.1.4.3/32 -d 10.1.3.254/32 -p tcp --dport 631 -m layer7 --l7proto unset -m state --state NEW,ESTABLISHED -j ACCEPT iptables -A FORWARD -s 10.1.3.254/32 -d 10.1.4.3/32 -p tcp --sport 631 -m layer7 --l7proto unset -m state --state ESTABLISHED -j ACCEPT iptables -A FORWARD -s 10.1.2.3/32 -d 10.1.3.254/32 -p tcp --dport 631 -m layer7 --l7proto ipp -m state --state NEW,ESTABLISHED -j ACCEPT iptables -A FORWARD -s 10.1.3.254/32 -d 10.1.2.3/32 -p tcp --sport 631 -m layer7 --l7proto ipp -m state --state ESTABLISHED -j ACCEPT iptables -A FORWARD -s 10.1.2.3/32 -d 10.1.3.254/32 -p tcp --dport 631 -m layer7 --l7proto unset -m state --state NEW,ESTABLISHED -j ACCEPT iptables -A FORWARD -s 10.1.3.254/32 -d 10.1.2.3/32 -p tcp --sport 631 -m layer7 --l7proto unset -m state --state ESTABLISHED -j ACCEPT
FW3	iptables -F FORWARD iptables -F INPUT iptables -F OUTPUT iptables -P FORWARD DROP #User: [EngineeringDirector] Source: [HostE3] Resource: [WebAdmin,PrinterAdmin] Dest: [Srv3] Service: [http,https,ipp] iptables -A FORWARD -s 10.1.4.3/32 -d 10.1.3.254/32 -p tcp --dport 80 -m layer7 --l7proto http -m state --state NEW,ESTABLISHED -j ACCEPT iptables -A FORWARD -s 10.1.3.254/32 -d 10.1.4.3/32 -p tcp --sport 80 -m layer7 --l7proto http -m state --state ESTABLISHED -j ACCEPT iptables -A FORWARD -s 10.1.4.3/32 -d 10.1.3.254/32 -p tcp --dport 80 -m layer7 --l7proto unset -m state --state NEW,ESTABLISHED -j ACCEPT iptables -A FORWARD -s 10.1.3.254/32 -d 10.1.4.3/32 -p tcp --sport 80 -m layer7 --l7proto unset -m state --state ESTABLISHED -j ACCEPT iptables -A FORWARD -s 10.1.4.3/32 -d 10.1.3.254/32 -p tcp --dport 443 -m layer7 --l7proto ssl -m state --state NEW,ESTABLISHED -j ACCEPT iptables -A FORWARD -s 10.1.3.254/32 -d 10.1.4.3/32 -p tcp --sport 443 -m layer7 --l7proto ssl -m state --state ESTABLISHED -j ACCEPT iptables -A FORWARD -s 10.1.4.3/32 -d 10.1.3.254/32 -p tcp --dport 443 -m layer7 --l7proto unset -m state --state NEW,ESTABLISHED -j ACCEPT iptables -A FORWARD -s 10.1.3.254/32 -d 10.1.4.3/32 -p tcp --sport 443 -m layer7 --l7proto unset -m state --state ESTABLISHED -j ACCEPT iptables -A FORWARD -s 10.1.4.3/32 -d 10.1.3.254/32 -p tcp --dport 631 -m layer7 --l7proto ipp -m state --state NEW,ESTABLISHED -j ACCEPT iptables -A FORWARD -s 10.1.3.254/32 -d 10.1.4.3/32 -p tcp --sport 631 -m layer7 --l7proto ipp -m state --state ESTABLISHED -j ACCEPT iptables -A FORWARD -s 10.1.4.3/32 -d 10.1.3.254/32 -p tcp --dport 631 -m layer7 --l7proto unset

	-m state --state NEW,ESTABLISHED -j ACCEPT iptables -A FORWARD -s 10.1.3.254/32 -d 10.1.4.3/32 -p tcp --sport 631 -m layer7 --l7proto unset -m state --state ESTABLISHED -j ACCEPT

As can be observed, the resulting scripts were generated for configuration of IPtables firewalls with the L7-filter module. As the proposed architecture supports the translation of multi vendor firewalls, the same policy can be used for configuration of IPTables without the use of L7-filter and CheckPoint Firewall-1. The inventory contains the information about the firewall models and points to the library that should be used accordingly. For CheckPoint Firewall-1, the scripts generated are presented in Table 7.5.

Table 7.5: INSPECT scripts generated for rule 8

Firewall	**Script**

FW2	prolog_services = { <99999,99999>, <21,21>, <111,111> };
	tcp_services = { <79, 80>, <443, 443>, <636, 636>,};
	tcp_fastmode_services = { <0, 0> };
	udp_services = { };
	
	#include "tcpip.def"
	#include "fwui_head.def"
	#include "base.def"
	
	// User: [EngineeringDirector,CommercialDirector] Source: [HostE3,HostC3] Resource: [WebAdmin,PrinterAdmin] Dest: [Srv3] Service: [http,https,ipp]
	accept tcp, http, ip_src = 10.1.4.3/32, ip_dst = 10.1.3.254/32;
	accept tcp, http, ip_src = 10.1.2.3/32, ip_dst = 10.1.3.254/32;
	accept tcp, https, ip_src = 10.1.4.3/32, ip_dst = 10.1.3.254/32;
	accept tcp, https, ip_src = 10.1.2.3/32, ip_dst = 10.1.3.254/32;
	accept tcp, ip_src = 10.1.4.3/32, ip_dst = 10.1.3.254/32, dport = 631;
	accept tcp, ip_src = 10.1.4.3/32, ip_dst = 10.1.3.254/32, sport = 631;
	accept tcp, ip_src = 10.1.2.3/32, ip_dst = 10.1.3.254/32, dport = 631;
	accept tcp, ip_src = 10.1.2.3/32, ip_dst = 10.1.3.254/32, sport = 631;
	
	#include "fwui_trail.def"

FW3	prolog_services = { <99999,99999>, <21,21>, <111,111> };
	tcp_services = { <79, 80>, <443, 443>, <636, 636>,};
	tcp_fastmode_services = { <0, 0> };
	udp_services = { };
	
	#include "tcpip.def"
	#include "fwui_head.def"
	#include "base.def"
	
	// User: [EngineeringDirector] Source: [HostE3] Resource: [WebAdmin,PrinterAdmin] Dest: [Srv3] Service: [http,https,ipp]
	accept tcp, http, ip_src = 10.1.4.3/32, ip_dst = 10.1.3.254/32;
	accept tcp, https, ip_src = 10.1.4.3/32, ip_dst = 10.1.3.254/32;
	accept tcp, ip_src = 10.1.4.3/32, ip_dst = 10.1.3.254/32, dport = 631;
	accept tcp, ip_src = 10.1.4.3/32, ip_dst = 10.1.3.254/32, sport = 631;
	
	#include "fwui_trail.def"

7.4 Conclusion

This chapter presented an example of the application of the proposed framework for the definition of a network security policy. The example network contains 5 private subnets and a connection to the internet. The subnets and internet are connected by 4 firewalls, which must be configured in order to enforce the global security policy.

The example presented the inventory description and the three policy models, in order to demonstrate how the three models restrict the users' accesses. The firewalls were defined as IPTables with L7-Filter, and the resulting scripts were presented for rule 8. The actual implementation has named service libraries for IPTables, IPTables with L7-Filter and CheckPoint Firewall-1 (INSPECT language), thus it is possible to configure the network with any of these firewalls.

Chapter 8

Conclusion and Future Work

This thesis proposes a new approach for network policy definition, capable of handling a multi-constraint security policy model. The policy can be simultaneously defined in mandatory, discretionary and security property models. This approach enables the cooperation of multiple security staff for the policy definition.

Mandatory model allows the representation of coarse grained policies, in terms of which resources the users are permitted to access. Users must have a clearance greater than or equal to the classification of the resource and must have the need-to-know compartments necessary to access the resource.

Security property model allows the representation of location and path dependent policies, where an access is permitted or denied based on the location of the user and the path that connects the user to the resource. The concept of security properties are directly connected to the path that packet traverse from the source to destination. Then, the location from where the access is initiated (source) and the path between the source and the destination are decisive to determine if the access is permitted or not.

Finally, the discretionary model allows the representation of fine grained permissions. With this model, it is possible to specify from very specific rules, such as “Engineering group is allowed to access the Projects Resource at Main Server from the technology lab” to very generic rules, such as “Any user from Internal Users Group is allowed to access any resource at Main Server”.

The proposed framework includes an algorithm that is capable to process the three policy models, and compute the set of resulting permissions. Note that the resulting permissions are

always the "most restrictive" from the three policy models, i.e., a permission will only exist if it is defined by the three policy models.

The framework has an inventory, which contains the information necessary to define the three policy models and to represent the topology of the network. Having the topology represented separately from the policies turns the framework in topology independent, i.e., it is possible to change the topology, without demanding changes in the global security policy.

The inventory also contains the information about the named services implemented by the firewalls of the network. A named service is a high-level secured network service; in other words, it represents an abstract firewall protection service. The translation of a named service to the specific firewall configuration script/rules is done using a Named Service Library. The Named Service Library contains the mapping between the named services and the firewall scripts/rules. Each firewall model/vendor will have its own library. The advantage of having a library is that the library can be implemented by experts, in order to fully use the corresponding firewall technology.

The combination of the Named Service and Named Service Library enables the framework to be device independent, since the topology and policies can be defined independently of the specific firewall models/vendors that are present in the network.

One important aspect of this approach is that even simple firewalls can be present in the network, since the algorithm determines the permissions that can be securely applied, without violating the high level security policy. The algorithm includes a step named Relaxation. When a specific service access must be configured from the source to the destination, it is possible that more than one firewall is present in the path. If at least one firewall implements the necessary named service, the algorithm is capable to create the firewall scripts/rules for all the firewall in the path in order to enforce that service access. The relaxation procedure is possible because the packets of a particular service access will cross all the firewalls along the corresponding path. If at least one of the firewalls blocks the offending packets, the access is protected.

In chapter 5, it is presented a formal approach for analysis and validation of the proposed framework. The framework has been formalized using the Z-notation, which is a mathematical notation that allows the representation and validation of the algorithm. The Z/Eves tool was used for the representation and validation, in order to demonstrate that the transformations performed by the algorithm don't cause violation of the policies.

A performance analysis and evaluation are presented, in order to demonstrate the behavior of the proposed algorithm for large scenarios. The complexity analysis demonstrated the

variation of the computation time in terms of the size of the input variables (number of rules, users, resources, services, locations and firewalls). It is also presented a performance evaluation, to demonstrate how the algorithm behaves in real scenarios.

Finally, an example is presented. The example represents a small, but yet realist, network. The network contains 5 subnets and the internet, and 4 firewalls connecting them. The mandatory, discretionary and security properties were defined, and the implementation processed the policies and inventory. The resulting scripts for configuration of IPTables with L7-filter are presented.

In summary, the main contributions of this thesis are:

- Specification of three-dimensional framework for network security policy definition, which allows different security staff to be responsible for the description of each dimension (Chapter 4);
- Specification of a high-level security policy language, able to handle mandatory, discretionary and security property policies, independently of topology and firewall models/vendors (Chapter 4, section 4.4);
- Definition of an information model that comprises the network topology (Chapter 4, section 4.3.1);
- Definition of the Named Service concept, which is a high-level representation of the firewall protection services (Chapter 4, section 4.2);
- Description of the algorithm that process the three policies specifications, responsible for translating the high-level security policy to low-level scrips/rules for firewall configuration; (Chapter 4, section 4.5);
- Demonstration of the formal validation of the algorithm using a mathematical language (Chapter 5)
- Scalability Study of the proposed framework (Chapter 6).

This work can be extended and enhanced in a number of ways. The actual framework includes named service libraries for IPTables, IPTables with L7-Filter and CheckPoint Firewall 1 (INSPECT language). New named service libraries can be constructed for other firewall models, in order to increase the applicability of the proposed framework.

The representation of IP address and mask can be improved, in order to allow easier representation of groups of locations that not necessarily includes the entire subnet (for example, declaring groups that doesn't some hosts of the subnet).

The strategy "anything not explicitly allowed is forbidden" is somewhat restrictive, since in some situations may be necessary to represent negative policies. The framework can be extended to include the possibility of defining negative policies. However, it is necessary to consider the situations where conflicts exist, since the interaction of positive and negative authorizations can become extremely complicated [57].

The current framework considers a security assumption for each named service. This could be improved, in order that the named service would have more than one security assumption, one for each firewall model depending on how the firewall enforce the named service. For example, the implementation of the HTTP protocol would receive a low security assumption when enforce by a packet filter, and a greater security assumption when it is enforce by a stateful firewall (and even greater if the stateful firewall has deep packet inspection). This would turn the security property policy more powerful.

Finally, the framework could be improved in order to allow incremental processing. In this way, the framework would only generate the firewall rules or scripts necessary to reflect the high level change, instead of reprocessing the entire policy and inventory and generating the full set of configuration rules or scripts.

Appendix A

Example Inventory and Policy

A.1 Inventory

```
firewall_library(fw1, iptableswithl7filterlib).
firewall_library(fw2, iptableswithl7filterlib).
firewall_library(fw3, iptableswithl7filterlib).
firewall_library(fw4, iptableswithl7filterlib).

named_service_library(fw1, [http, https, smtp, pop3, imap, ftp, ipp,
msnMessenger, msnFileTransfer, vnc]).
named_service_library(fw2, [http, https, smtp, pop3, imap, ftp, ipp,
msnMessenger, msnFileTransfer, vnc]).
named_service_library(fw3, [http, https, smtp, pop3, imap, ftp, ipp,
msnMessenger, msnFileTransfer, vnc]).
named_service_library(fw4, [http, https, smtp, pop3, imap, ftp, ipp,
msnMessenger, msnFileTransfer, vnc]).

user(CIO).
user(Salesman).
user(CommercialSupervisor).
user(CommercialDirector).
user(Engineer).
user(EngineeringSupervisor).
user(EngineeringDirector).
user(ProductionUser).
user(NetworkAdministrator).
user(Guest).

user_group(insiderUsers, [CIO, Salesman, CommercialSupervisor,
CommercialDirector, Engineer, EngineeringSupervisor,
EngineeringDirector, ProductionUser, NetworkAdministrator]).

user_located_at(CIO, [VPN1,HostA4]).
user_located_at(Salesman, [VPN2,VPN3,VPN4,HostH1]).
```

```
user_located_at(CommercialSupervisor, [HostC2]).
user_located_at(CommercialDirector, [HostC3,HostA2]).
user_located_at(Engineer, [HostE1,HostP2]).
user_located_at(EngineeringSupervisor, [HostE2,HostP3]).
user_located_at(EngineeringDirector, [HostA1,HostE3]).
user_located_at(ProductionUser, [HostP1]).
user_located_at(NetworkAdministrator, [HostA3]).
user_located_at(Guest, [Internet]).

location(VPN1,'10.1.2.201/32').
location(VPN2,'10.1.2.202/32').
location(VPN3,'10.1.2.203/32').
location(VPN4,'10.1.2.204/32').
location(HostC1,'10.1.2.1/32').
location(HostC2,'10.1.2.2/32').
location(HostC3,'10.1.2.3/32').
location(HostA1,'10.1.3.1/32').
location(HostA2,'10.1.3.2/32').
location(HostA3,'10.1.3.3/32').
location(HostA4,'10.1.3.4/32').
location(HostE1,'10.1.4.1/32').
location(HostE2,'10.1.4.2/32').
location(HostE3,'10.1.4.3/32').
location(HostP1,'10.1.5.1/32').
location(HostP2,'10.1.5.2/32').
location(HostP3,'10.1.5.3/32').
location(Srv1, '10.1.1.254/32').
location(Srv2, '10.1.2.254/32').
location(Srv3, '10.1.3.254/32').
location(Srv4, '10.1.4.254/32').

location(DMZ, '10.1.1.0/24').
location(Commercial, '10.1.2.0/24').
location(Administration, '10.1.3.0/24').
location(Engineering, '10.1.4.0/24').
location(Production, '10.1.5.0/24').
location(Internet, '200.0.0.0/8').

location_group(Internet,'200.0.0.0/8',[VPN1,VPN2,VPN3,VPN4]).
location_group(DMZ,'10.1.1.0/24',[Srv1]).
location_group(Commercial,'10.1.2.0/24',[HostC1,HostC2,HostC3,Srv2]).
location_group(Administration,'10.1.3.0/24',[HostA1,HostA2,HostA3,HostA
4,Srv3]).
location_group(Engineering,'10.1.4.0/24',[HostE1,HostE2,HostE3,Srv4]).
location_group(Production,'10.1.5.0/24',[HostP1,HostP2,HostP3]).

firewall(fw1, [ifa, ifb, ifc]).
firewall(fw2, [ifa, ifb]).
firewall(fw3, [ifa, ifb]).
firewall(fw4, [ifa, ifb]).

connected(Internet, fw1, ifa, 1).
connected(DMZ, fw1, ifb, 1).
connected(Commercial, fw1, ifc, 1).
connected(Commercial, fw2, ifa, 1).
```

```
connected(Administration, fw2, ifb, 1).
connected(Commercial, fw3, ifa, 1).
connected(Engineering, fw3, ifb, 1).
connected(Engineering, fw4, ifa, 1).
connected(Production, fw4, ifb, 1).

named_service(http).
named_service(https).
named_service(smtp).
named_service(pop3).
named_service(imap).
named_service(ftp).
named_service(ipp).
named_service(msnMessenger).
named_service(msnFileTransfer).
named_service(vnc).

resource(WebInternet, [http, https]).
resource(Chat, [msnMessenger,msnFileTransfer]).
resource(WebDMZ, [http, https]).
resource(EmailDMZ, [smtp, pop3, imap]).
resource(FileTransferDMZ, [ftp]).
resource(RemoteAdmin, [vnc]).
resource(WebCommercial, [http, https]).
resource(FileTransferCommercial, [ftp]).
resource(PrinterCommercial, [ipp]).
resource(WebAdmin, [http, https]).
resource(FileTransferAdmin, [ftp]).
resource(PrinterAdmin, [ipp]).
resource(WebEngineering, [http, https]).
resource(FileTransferEngineering, [ftp]).

resource_located_at(WebInternet, [Internet]).
resource_located_at(Chat, [Internet]).
resource_located_at(WebDMZ, [Srv1]).
resource_located_at(EmailDMZ, [Srv1]).
resource_located_at(FileTransferDMZ, [Srv1]).
resource_located_at(RemoteAdmin, [Srv1, Srv2, Srv3, Srv4]).
resource_located_at(WebCommercial, [Srv2]).
resource_located_at(FileTransferCommercial, [Srv2]).
resource_located_at(PrinterCommercial, [Srv2]).
resource_located_at(WebAdmin, [Srv3]).
resource_located_at(FileTransferAdmin, [Srv3]).
resource_located_at(PrinterAdmin, [Srv3]).
resource_located_at(WebEngineering, [Srv4]).
resource_located_at(FileTransferEngineering, [Srv4]).
```

A.2 Mandatory Policy

```
clearance(CIO, 4, [Commercial, Administration, Engineering,
Production]).
clearance(Salesman, 2, [Commercial]).
clearance(CommercialSupervisor, 3, [Commercial]).
```

```
clearance(CommercialDirector, 3, [Commercial, Administration]).
clearance(Engineer, 2, [Administration, Engineering]).
clearance(EngineeringSupervisor, 3, [Engineering]).
clearance(EngineeringDirector, 4, [Administration, Engineering]).
clearance(ProductionUser, 1, []).
clearance(NetworkAdministrator, 4, [NetworkAdministration]).
clearance(Guest, 1, []).

classification(WebInternet, 1, []).
classification(Chat, 3, []).
classification(WebDMZ, 1, []).
classification(EmailDMZ, 2, []).
classification(FileTransferDMZ, 2, []).
classification(RemoteAdmin, 4, [NetworkAdministration]).
classification(WebCommercial, 2, []).
classification(FileTransferCommercial, 2, [Commercial]).
classification(PrinterCommercial, 2, []).
classification(WebAdmin, 2, []).
classification(FileTransferAdmin, 3, [Administration]).
classification(PrinterAdmin, 3, [Administration]).
classification(WebEngineering, 2, []).
classification(FileTransferEngineering, 2, [Engineering]).
```

A.3 Security Property Policy

```
security_assumption(fw1, [2,3]).
security_assumption(fw2, [4,4]).
security_assumption(fw3, [4,4]).
security_assumption(fw4, [4,4]).

security_assumption(Internet, [1,1]).
security_assumption(DMZ, [1,1]).
security_assumption(Commercial, [1,3]).
security_assumption(Administration, [1,4]).
security_assumption(Engineering, [1,3]).
security_assumption(Production, [1,3]).
security_assumption(VPN1, [3,3]).
security_assumption(VPN2, [3,3]).
security_assumption(VPN3, [3,3]).
security_assumption(VPN4, [3,3]).

security_assumption(http, [1,1]).
security_assumption(https, [4,1]).
security_assumption(smtp, [2,1]).
security_assumption(pop3, [2,1]).
security_assumption(imap, [2,1]).
security_assumption(ftp, [2,1]).
security_assumption(ipp, [2,1]).
security_assumption(msnMessenger, [2,1]).
security_assumption(msnFileTransfer, [1,1]).
security_assumption(vnc, [1,1]).

security_requirement(WebInternet, [1,2]).
```

```
security_requirement(Chat, [1,1]).
security_requirement(WebDMZ, [1,1]).
security_requirement(FileTransferDMZ, [2,1]).
security_requirement(EmailDMZ, [1,1]).
security_requirement(RemoteAdmin, [1,3]).
security_requirement(WebCommercial, [1,2]).
security_requirement(FileTransferCommercial, [1,2]).
security_requirement(PrinterCommercial, [2,3]).
security_requirement(WebAdmin, [1,3]).
security_requirement(FileTransferAdmin, [2,2]).
security_requirement(PrinterAdmin, [2,3]).
security_requirement(WebEngineering, [1,2]).
security_requirement(FileTransferEngineering, [1,2]).
```

A.4 Discretionary Policy

```
rule(1, [any], [WebDMZ], [any], [DMZ], permit).
rule(2, [insiderUsers], [EmailDMZ], [any], [DMZ], permit).
rule(3, [NetworkAdministrator], [FileTransferDMZ], [any], [DMZ],
permit).
rule(4, [NetworkAdministrator], [RemoteAdmin], [any], [any], permit).
rule(5, [NetworkAdministrator],
[FileTransferAdmin,FileTransferCommercial], [any], [any], permit).
rule(6, [Salesman,CommercialSupervisor],
[WebCommercial,PrinterCommercial], [any], [Commercial], permit).
rule(7, [Engineer,EngineeringSupervisor,EngineeringDirector],
[WebEngineering,FileTransferEngineering], [any], [Engineering],
permit).
rule(8, [CIO,EngineeringDirector,CommercialDirector],
[WebAdmin,PrinterAdmin], [any], [Administration], permit).
rule(9, [NetworkAdministrator], [PrinterAdmin], [Administration],
[Administration], permit).
rule(10, [CIO,EngineeringDirector,CommercialDirector], [Chat], [any],
[Internet], permit).
rule(11, [any], [any], [any], [Internet], permit).
rule(12, [Salesman], [PrinterAdmin], [any], [Administration], permit).
rule(13, [ProductionUser], [WebEngineering], [any], [Engineering],
permit).
rule(14, [CIO], [WebAdmin], [Internet], [Administration], permit).
```

Publications

[Kropiwiec et al. 2006] Kropiwiec, C. D.; Jamhour, E.; Fonseca, M. S. P.; Enembreck, F.; Pujolle, G. Uma Abordagem Baseada em Programação Declarativa para Configuração de Firewalls em Ambientes Heterogêneos. *24º Simpósio Brasileiro de Redes de Computadores – Curitiba, SBRC 2006*, v. 1, 2006.

[Kropiwiec et al. 2008] Kropiwiec, C. D.; Jamhour, E.; Penna, M. C.; Pujolle, G. Multi-Constraint Security Policies for Delegated Firewall Administration. *IFIP/IEEE International Workshop on Distributed Systems: Operations and Management (DSOM 2008)*, Lecture Notes in Computer Science (LNCS 5273/2008), v. 5273. pp. 123-135, 2008.

References

[1] D. Lin, P. Rao, E. Bertino, N. Li, and J. Lobo. Policy Decomposition for Collaborative Access Control. *Proceedings of the 13th ACM Symposium on Access Control Models and Technologies - SACMAT'08*, Estes Park, Colorado, USA. pp. 103-112, (2008)

[2] E. Al-Shaer, and H. Hamed. Discovery of Policy Anomalies in Distributed Firewalls. *23rd Conference of the IEEE Communications Society (INFOCOMM)*, pp. 2605-2616 (2004)

[3] Cisco Systems Inc.: Cisco PIX Firewall Command Reference. Available at: http://www.cisco.com , (2004)

[4] Cisco Systems Inc.: Cisco IOS Reference Guide. Available at: http://www.cisco.com, (2004)

[5] CheckPoint Software Technologies Ltd.: Stateful Inspection Technology. Available at: http://www.checkpoint.com/products, (2005)

[6] T. K. Lee, S. Yusuf, W. Luk, M. Sloman, E. Lupu, and N. Dulay. Compiling Policy Descriptions into Reconfigurable Firewall Processors. *11th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, pp. 39-48, (2003)

[7] N. Damianou, N. Dulay, E. Lupu, and M. Sloman. The Ponder Policy Specification Language. *Policy 2001: Workshop on Policies for Distributed Systems and Networks*, pp. 18-39, (2001)

[8] E. Winjum, and B. K. Mølmann. A Multidimensional Approach to Multilevel Security. *Information Management & Computer Security* - Vol. 16 No. 5, pp. 436-448, (2008)

[9] X. Ou, S. Govindavajhala, and A. W. Appel. Network security management with high-level security policies". *Technical report TR-714-04, Computer Science Dept, Princeton University*, (2004)

[10] B. Zhang, E. Al-Shaer, R. Jagadeesan, J. Riely, and C. Pitcher. Specifications of a High-Level Conflict-Free Firewall Policy Language for Multi-Domain Networks. *Proceedings of the 12th ACM Symposium on Access Control Models and Technologies*, 185-194, (2007)

[11] J. D. Guttman. Filtering postures: local enforcement for global policies. *IEEE Symposium on Security and Privacy*, pp. 120-129, (1997)

[12] Y. Bartal, A. J. Mayer, K. Nissin, and A. Wool. Firmato: A novel firewall management toolkit. *ACM Transactions on Computer Systems*, vol. 22, no. 4, pp. 381-420, (2004)

[13] DOD: Trusted Computer Security Evaluation Criteria. DOD 5200.28-STD. *Department of Defense*, (1985)

[14] J. P. Albuquerque, H. Krumm, and P. L. Geus. Policy Modeling and Refinement for Network Security Systems. *IEEE 6th International Workshop on Policies for Distributed Systems and Networks*, pp. 24-33, (2005)

[15] J. M. Spivey. The Z notation: a reference manual. Prentice Hall International (UK) Ltd., Hertfordshire, UK, (1992)

[16] M. Saaltink. The Z/EVES system. *J. P. Bowen, M. G. Hinchey, and D. Till, (eds.), ZUM 1997 LNCS*, vol. 1212, pp. 72–85, Springer-Verlag, (1997)

[17] E.W. Dijkstra. A note on two problems in connection with graphs. Numer. Math., 1:269–271, 1959.

[18] L7-filter Application Layer Packet Classifier for Linux. Available at http://l7-filter.sourceforge.net/

[19] Telnet pattern, available at http://l7-filter.sourceforge.net/layer7-protocols/protocols/telnet.pat

[20] J. Guttman, and A. Herzog. Rigorous automated network security management. *International Journal of Information Security 4*, pp. 29-48, (2005)

[21] D. Haixin, W. Jianping, and L. Xing. Policy-Based Access Control Framework for Large Networks. *Eighth IEEE International Conference on Networks*, pp. 267-273, (2000)

[22] T. Jaeger, X. Zhang, and A. Edwards. Policy Management Using Access Control Spaces. *ACM Transactions on Information and System Security*, Vol. 6, No. 3, August 2003, pp 327–364, (2003).

[23] R. Loew, I. Stengel, U. Bleimann, and A. McDonald. Security Aspects of Enterprise-Wide Network Architecture. *Internet Research: Electronic Networking Applications and Policy*, Volume 9, Number 1 - pp. 8–15 (1999)

[24] G. N. Stone, B. Lundy, and G. G. Xie. Network Policy Languages: A Survey and a New Approach - *IEEE Network*, Volume 15, Issue 1, pp: 10 – 21, (2001).

[25] J. Burns, A. Cheng, P. Gurung, S. Rajagopalan, P. Rao, D. Rosenbluth, A. V. Surendran, and D. M. Martin Jr. Automatic Management of Network Security Policy. *DARPA Information Survivability Conference and Exposition*, vol. II, pp. 12-26, (2001)

[26] T. Markham, and C. Payne. Security at the Network Edge: A Distributed Firewall Architecture. *DARPA Information Survivability Conference and Exposition (DISCEX II'01)*, pp. 279, vol. I, (2001)

[27] The netfilter.org "iptables" project: http://www.netfilter.org/projects/iptables/index.html

[28] Extensible access control markup language (XACML) version 2.0. *OASIS Standard*, 2005.

[29] R. S. Sandhu, and P. Samarati, P. Access Controls: Principles and practice. *IEEE Communications*, 32(9), September 1994

[30] J. B. D. Joshi, W. G. Aref, A. Ghafoor, and E. H. Spafford. Security Models for Web-Based Applications. *Communications of the ACM*. Volume 44 , Issue 2 , pp. 38-44, February 2001

[31] D. E. Bell, and L. J. LaPadula. (1975), "Secure computer systems, mathematical foundations*", Mitre Corp. Report No. MTR-2547*, Mitre Corp., Bedford, MA

[32] D. E. Denning., A lattice model of secure information flow. *Communications of the ACM*, Vol. 19 No. 5, pp. 236-43, (1976.

[33] K. J. Biba. Integrity considerations for secure computer systems*, Mitre Corp. Report No. MTR-3153*, Mitre Corp., Bedford, MA, Bedford, MA, 1977.

[34] R. S. Sandhu. Lattice-Based Access Control Models. *IEEE Compute*r, vol. 26, no. 11, pp. 9-19, Nov. 1993, doi:10.1109/2.241422

[35] E. Al-Shaer, and H. Hamed. Firewall Policy Advisor for Anomaly Detection and Rule Editing. *Proceedings of IEEE/IFIP Integrated Management Conf. (IM'2003)*, 2003.

[36] L. Yuan, H. Chen, J. Mai, C. Chuah, Z. Su, and P. Mhapatra. FIREMAN: A Toolkit for FIREwall Modeling an Analysis. *IEEE Symposium on Security and Privacy*, Berkeley/Oakland, CA, pp 199-213, 2006.

[37] A. A. Hassan. Algorithms for verifying firewall and router access lists. *IEEE International Symposium on Micro-NanoMechatronics and Human Science*, pp. 512-515, 2003.

[38] T. Uribe, and S. Cheung. Automatic analysis of firewall and network intrusion detection system configurations. *Journal of Computer Security*, Vol. 15, Issue 6, pp. 691-715, 2007.

[39] A. Liu and M. G. Gouda. Diverse Firewall Design. *IEEE Transactions on Parallel and Distributed Systems*, Vol. 19, Issue 9, pp. 1237-1251, 2008.

[40] A. El-Atawy, T. Samak, Z. Wali, E. Al-Shaer, F. Lin, C. Pham, and S. Li. An Automated Framework for Validating Firewall Policy Enforcement. *Proceedings of the Eighth IEEE International Workshop on Policies for Distributed Systems and Networks*, pp. 151-160, 2007.

[41] R. S. Sandhu, E. J. Coyne, et al. Role-Based Access Control Models. *IEEE Computer* 29(2), pp. 38-47, 1996.

[42] R. S. Sandhu. Role Activation Hierarchies. *Third ACM/NIST Role Based Access Control Workshop*, Fairfax, Virginia, USA, ACM Press, 1998.

[43] R. D. Sandhu, D. Ferraiolo, and R. Kuhn. The NIST Model for Role-Based Access Control: Towards A Unified Standard. *5th ACM Workshop on Role-Based Access Control*, Berlin, Germany, pp 47-63, 2000.

[44] J. F. Barkley, K. Beznosov, et al. Supporting Relationships in Access Control Using Role Based Access Control. *Fourth ACM Workshop on Role-Based Access Control*, Fairfax, Virginia, USA, 1999.

[45] R. K. Thomas. Team-based Access Control (TMAC): A Primitive for Applying Role-based Access Controls in Collaborative Environments. *Second ACM/NIST Role Based Access Control Workshop*, Fairfax, Virginia, USA, ACM Press, 1997.

[46] C.P. Pfleeger: Security in Computing (second edition). Prentice-Hall, Inc. 1997.

[47] C. Hare, and K. Siyan. Internet Firewalls and Network Security (Second Edition). New Riders Publishing, 1996.

[48] E. S. Al-Shaer, and H. H. Hamed. Modeling and Management of Firewall Policies. *IEEE Transactions on Network and Service Management*, Vol. 1, Issue 1, pp. 2-10, 2004.

[49] J. Wack, K. Cutler, and J. Pole. Guidelines on Firewalls and Firewall Policy. *NIST Recommendations, SP 800-41*, Jan. 2002.

[50] C. Scott, P. Wolfe, and M. Erwin. Virtual Private Networks (Second Edition). O'Reilly & Associates, 1998.

[51] P. Srisuresh, and K. Egevang. Traditional IP Network Address Translator (Traditional NAT). *RFC 3022, Internet Engineering Task Force (IETF)*, 2001.

[52] W. R. Cheswick, S. M. Bellovin, and A. D. Rubin. Firewalls and Internet Security: Repelling the Wily Hacker (Second Edition). Addison-Wesley, 2003.

[53] S. Osborn, R. Sandhu, and Q. Munawer. Configuring role-based access control to enforce mandatory and discretionary access control policies. *ACM Transactions on Information and System Security (TISSEC)*, Vol. 3, Issue 2, pp. 85-106, 2000.

[54] R. Sandhu. Role hierarchies and constraints for lattice-based access controls. *Proceedings of the Conference on Computer Security (ESORICS 96, Rome, Italy)*, Eds. Springer-Verlag, New York, NY, 65–79, 1996.

[55] R. Sandhu, and Q. Munawer. How to do discretionary access control using roles. *Proceedings of the Third ACM Workshop on Role-Based Access Control (RBAC '98, Fairfax, VA, Oct. 22–23)*, ACM Press, New York, NY, 47–54, 1998.

[56] Sun Microsystems (2008), "Sun Microsystems", available at: http://www.sun.com/.

[57] E. Bertino, P; Samarati, and S. Jajodia. Authorizations in relational database management systems. 1st *ACM Conference on Computer and Communications Security*, pp. 130-139, Fairfax, VA, November, 3-5, 1993.

[58] K. Scarfone, and P.Mell. Guide to Intrusion Detection and Prevention Systems (IDPS). *NIST Recommendations, SP 800-94*, 2007.

[59] S. Garfinkel, and G. Spafford. Practical Unix & Internet Security – Second Edition. O'Reilly & Associates, Sebastopol, CA, 1996.

[60] J. Klensin. Simple Mail Transfer Protocol. RFC *2821, Internet Engineering Task Force (IETF)*, 2001.

[61] S. Kent, and K. Seo. Security Architecture for the Internet Protocol. *RFC 4301. Internet Engineering Task Force*, 2005.

[62] J. Postel. Transmission Control Protocol. *RFC 793. Internet Engineering Task Force*, 1981.

[63] J. Postel, and J. Reynolds. File Transfer Protocol. *RFC 959. Internet Engineering Task Force*, 1985.

[64] Common Criteria Project. Common Criteria for Information Technology Security Evaluation (CC 2.2), Part 2: Security Functional Requirements, January 2004.

[65] ITU-T Telecommunications Standardization Sector of International Telecommunication Union. X.810 – Security Frameworks for Open Systems: Overview, 1995.

[66] Sendmail Consortium. http://www.sendmail.org/

[67] J. D. Moffet . Control Principles and Role Hierarchies. *Third ACM/NIST Role Based Access Control Workshop*, Fairfax, Virginia, USA, ACM Press, 1998.

[68] R. Laborde, B. Nasser, F. Grasset, F. Barrère, and A. Benzekri. Network Security Management: A Formal Evaluation Tool based on RBAC Policies. *IFIP Network Control and Engineering for QoS, Security and Mobility, III*, pp. 69-80, 2005.

[69] M. Saaltink. The Z/Eves Users' Guide. *ORA Canada Technical Report TR-97-5493-06*, 1997.

[70] J. C. P. Woodcock, and J. Davies. Using Z: Specification, Proof and Refinement. Prentice Hall Internation Series in Computer Science, 1996.

[71] B. F. Potter, J. E. Sinclair, and D. Till. An Introduction to Formal Specification and Z. Prentice Hall International Series in Computer Science, 2nd edition, 1996.

[72] M. Fitting. First-order logic and automated theorem proving, 2nd Edition. Springer, 1996.

[73] M. Newborn. Automated theorem proving: theory and practice. Springer, 2001.

[74] CERT Advisory CA-2003-12 Buffer Overflow in Sendmail, 2003. Available at http://www.cert.org/advisories/CA-2003-12.html.

[75] W. Noonan, and I. Dubrawsky. Firewall Fundamentals. Cisco Press, 2006.

[76] L. Yeo. Personal Firewalls. Prentice Hall Professional Technical Reference, 2002.

List of Acronyms

ACL	*Access Control List*
DAC	*Discretionary Access Control*
DMZ	*Demilitarized Zone*
ESA	*Effective Security Assumption*
FTP	*File Transfer Protocol*
ICMP	*Internet Control Message Protocol*
IDS	*Intrusion Detection System*
IETF	*Internet Enginneering Task Force*
IMAP	*Internet Message Access Protocol*
ITU-T	*International Telecommunication Union - Telecommunication Standardization Sector*
HTTP	*Hypertext Transfer Protocol*
LIP6	*Laboratoire d'Informatique de Paris VI*
MAC	*Mandatory Access Control*
MLS	*Multilevel Security*
NAT	*Network Address Translation*
OSA	*Overall Security Assumption*
OSI	*Open Systems Interconnection*
P2P	*Peer to Peer*
PDP	*Policy Decision Point*
PEP	*Policy Enforcement Point*
POP3	*Post Office Protocol version 3*
RBAC	*Role Based Access Control*
SA	*Security Assumption*
SMTP	*Simple Mail Transfer Protocol*

SR *Security Requirement*

TCP *Transmission Control Protocol*

UDP *User Datagram Protocol*

VPN *Virtual Private Network*

Glossary

Accesses	Contains (user, resource) pairs, where the user is permitted to access the resource by the mandatory policy;
Compartment	Also known as need-to-know, describes the restriction of data which is considered very sensitive. Under need-to-know restrictions, even if one has all the necessary official approvals (such as a security clearance) to access certain information, one would not be given access to such information unless one has a specific need to know; that is, access to the information must be necessary for the conduct of one's official duties.
Classification	Security level that are claimed from the individual that wants to access the classified information.
Clearance	Security level granted to individuals allowing them access to classified information.
Dijkstra algorithm	Graph search algorithm that solves the single-source shortest path problem for a graph with nonnegative edge path costs, producing a shortest path tree.
Feasible Service Access	Contains all the service accesses that hold the three policy models and that includes in their paths at least one firewall capable to implement the corresponding named service.
Firewall Permission	Contains the permissions that must be configured in each firewall of the network.
Named Service	Models a “high-level” secured network service that is supposed to be configured in the firewalls along the network, i.e., it is an abstract representation for the firewall protection services
Named Service Library	Contains the information necessary to translate optimized firewall

	permission to rules or scripts used to configure the physical firewalls.
Possible Service Access	Contains all the service accesses that holds the three policy models;
PossibleVias	Contains tuples (resource, namedservice, path) that holds the security property policy
Security Assumption	Security class associated to a node or a named service,
Security Class	Vector of security levels, representing the properties confidentiality, integrity, availability and accountability.
Security Level	A natural number ranging from one to four, expressing the "strength" of one of the following security properties: confidentiality, integrity, availability and accountability.
Security Requirement	Security class necessary to access the resource
Service Access	Corresponds to the tuple ((user, resource), (source, destination), (named service, path)), which means that the user from the source location is allowed to access the resource at the destination location through the path using the named service.

List of Figures

List of Tables