SECURITY POLICY INTEGRATION AND CONSISTENCY VALIDATION

BY

HAMED OKHRAVI

B.Eng., Sharif University of Technology, 2003

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Electrical and Computer Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2007

Urbana, Illinois

ACKNOWLEDGMENTS

I would like to sincerely thank my advisor, Prof. David M. Nicol, who always supported me with his effective guidance and brilliant ideas without which this project would not have been possible. Also thanks to my family who endured this process with me, always offering support and love.

TABLE OF CONTENTS

LIST OF FIGURES vi

LIST OF TABLES vii

1. INTRODUCTION 1
1.1 Problem of Consistency 1
1.2 Consistency Checking Frameworks 2
1.3 The New Framework 3
1.4 Contributions 3
1.5 Thesis Organization 4

2. RELATED WORKS 5
2.1 The Global Policy Languages 5
2.1.1 Ponder 5
2.1.2 Security policy language (SPL) 7
2.1.3 Policy description language (PDL) 9
2.1.4 Data access list (DACL) 9
2.1.5 Security assertion markup language (SAML) 9
2.1.6 eXtensible access control markup language (XACML) 10
2.1.7 CDSFramework & SEFramework 11
2.2 Composition and Decomposition of Policies 13
2.3 Consistency of Policies 13

3. THE SECURITY-ENHANCED LINUX (SE-LINUX) POLICY 15
3.1 Access Control 15
3.1.1 Discretionary access control (DAC) 16
3.1.2 Mandatory access control (MAC) 16
3.2 The SE-Linux Policy Language 17
3.2.1 Type enforcement (TE) policies 17
3.2.2 Role-based access control (RBAC) policies 19
3.2.3 User identity (UI) 20
3.3 The SE-Linux Models 21
3.3.1 The state-machine model 21
3.3.2 The SE-Linux access control (SELAC) model 22

4. POLICY INTEGRATION AND GLOBAL ACCESSIBILITY SPACES 25
4.1 The Integration Scheme 25
4.2 The SE-Linux Network Configuration 26
4.3 The Global Accessibility Spaces 27
4.4 Translation and Consistency Checking 35

5. FORMAL MODEL 40
5.1 Formal Mapping 40
5.1.1 XACML features 40
5.1.2 XACML model 42
5.2 Safety Property 45

6. SECURITY METRIC DEVELOPMENT 49
6.1 High-Level View 49
6.2 Definition of the Metric 50
6.2.1 Primitive version 50
6.2.2 Extending the metric 54

7. CONCLUSION AND FUTURE WORK 57

REFERENCES 58

LIST OF FIGURES

Figure 2.1 - Policy hierarchy 5
Figure 2.2 - An XACML policy example.....11

Figure 4.1 - Tunnel server in front of firewall (picture from www.microsoft.com)....26
Figure 4.2 - Tunnel server behind firewall (picture from www.microsoft.com).....26
Figure 4.3 - Shared and nonshared objects and accessible sets.....32
Figure 4.4 - Mapping high-level entities into low-level ones.....35
Figure 4.5 - Translation bits and partitioning the entity spaces.....37

LIST OF TABLES

Table 2.1 - Ponder Authorization Policy Syntax 6
Table 2.2 - Ponder Information Filtering Syntax 6
Table 2.3 - Ponder Delegation Policy Syntax 7
Table 2.4 - An SPL Entity Hierarchy Definition 8
Table 2.5 - Example of SPL External Entities and Sets 8
Table 2.6 - SPL Rule Syntax 8
Table 2.7 - The DACL Policy Language 9
Table 2.8 - XACML Normative Elements 12

Table 3.1 - Different Permissions for Important Some Object Classes 19
Table 3.2 - The Sets Used in the SELAC Model 23

Table 4.1 - Notations Used in the Algorithm 29
Table 4.2 - The Algorithm for Building Connectivity Matrix 29
Table 4.3 - The Connectivity Matrix Convolution Algorithm 30
Table 4.4 - The Algorithm for Building Global Accessibility Sets 31
Table 4.5 - A Translation Table for Objects 36
Table 4.6 - Parts of the Framework Capturing each Policy Type 38

1. INTRODUCTION

Cyber-security in modern large enterprise networks can be achieved by enforcing a security policy by the means of a number of policy enforcement points (PEPs). Large enterprises and organizations usually have English security policy which is the high-level description of the desired security objectives and goals.

In a top-down security design approach, high-level security objectives are described in the form of a high-level security policy by the chief security officer of the organization and the policy is stated using a high-level policy language [1]. This policy is then decomposed manually into many local low-level policies or configurations and the low-level policies are then loaded into the policy enforcement points (PEPs).

The problem with this approach is that the decomposition process is error prone. As there are large number of low-level policies and each of them is very detailed and complex, it is impossible to make sure that the overall policy which the PEPs are actually implementing is the global high-level policy that was intended by the enterprise. In other words it may be possible for an intruder to hack into these systems without violating any low-level security policy and without being noticed by the security measures.

This work provides a formal framework for translating the high-level global policy into low-level local policies and checking the consistency between these local policies and their compliance with the global policy. Furthermore, our framework provides a basis measuring the degree of security of an enterprise which is usually referred to as "Security Metric." With the definition of the metric, it is possible to determine tactics to make the enterprise "more" secure and also to achieve the "best" security policy.

1.1 Problem of Consistency

The problem with the traditional manual policy implementation and analysis is that it is virtually impossible to check for consistency it the low-level policies.

Consider the following example: A large company network consists of some departments (e.g., engineering, accounting, administrative, etc.) In this company, the global security policy defines the high-level access controls and the normal traffic flow within the network. For example, different rules in the global policy may look like: "Only trusted users within the engineering department can access accounting records," "Anyone in the administrative department can access engineering plans," or "Any traffic between administrative and accounting departments must be confidential and integral." This global policy is implemented in the network using firewalls [2], [3], OS policies, and application configurations. However, it is virtually impossible for the administrator of the network to figure out inconsistencies in the policies by just looking at the low-level configurations.

As a result, there may be many inconsistencies in the low-level policies and the overall global policy that they implement. These inconsistencies are dangerous for the security of the enterprise inasmuch as an intruder does not even need to violate any policy to damage the system. An access which complies with all the enforcement policies may directly violate the intended global policy and might be malicious. Hence a formal automated framework for consistency check and policy validation is needed in these systems. The framework should be capable of integrating different types of low-level policies and validate it against a global high-level policy and point out the inconsistencies.

In the next section, we will briefly introduce some of frameworks that can validate policies, and we will discuss their shortcomings.

1.2 Consistency Checking Frameworks

There has been some work on the integration of different security policies and configuration for consistency checking and validation. We will discuss these frameworks in detail in the next chapter, and introduce the concept here.

Some frameworks have been designed for the integration of access control policies [4]-[8]. In these frameworks different access control policies are modeled under a single formalism and one can integrate different types of access control policies into a unified view. The problem with these frameworks is that they are capable of integrating only access control policies. The global policy is usually much broader than just access control and as a result these frameworks lack expressiveness

for a general global policy. For example, traffic shaping policies cannot be expressed in these frameworks.

Another type of framework is capable of integrating traffic policies and can check them for consistency [9]. This type of framework usually gets as its input the topology of the network and also policies from traffic controlling devices. These devices include network-based firewalls, host-based firewalls, intrusion detection systems (IDS), and routers. Because many unauthorized accesses and attacks have the same traffic patterns and use the same protocols as the authorized accesses, and because in these frameworks there is no notion of access, they fail to formalize a major part of global policy.

There are also some frameworks that formalize the complex OS policies (e.g., SE-Linux policy) to answer questions like: "Can a specific subject in the machine access an object in a specific mode?" The obvious problem with these frameworks is that they can only integrate OS policies within a single machine and they do not have a networked view of the system. As a result, they cannot express access control between different machines or traffic policies.

In the next section we briefly introduce our framework for integration of different policies and consistency checking. The new framework has all the advantages of the above mentioned frameworks and it is more expressive.

1.3 The New Framework

The framework that we propose integrates both traffic policies and access control policies and checks them against a global high-level policy. We use the idea of access control spaces [10] to describe OS policies (SE-Linux policies). Particularly, we use SE-Linux Access Control (SELAC) spaces [11] and extend the spaces to describe access control policies between different machines and to integrate firewall and traffic policies in the framework.

The new framework will be discussed in detail in Chapter 4, but before that we list some contributions of the new framework in the next section.

1.4 Contributions

The framework that we propose uses the objects and the subjects as defined by SE-Linux policies and builds global access control spaces. These spaces are built

using the SE-Linux policies, topology, and the rules inside traffic policy enforcement devices (network-based firewalls, host-based firewalls, etc.) Consequently, this framework is capable of integrating OS policies with firewall rules. To the best of our knowledge, this is the first work that combines SE-Linux policies and firewall rules under a single formalism.

The objects and subject that we use are direct resources of SE-Linux. As they come directly from the real-world entities, the model is fitted already to the real-world system. This is usually not the case for many of the formalisms and models for policy integration.

In addition, the fine granularity of the objects and the subjects in our framework allow a wide variety of global high-level policies to be described and checked for consistency.

Another important contribution of this work is the definition of the security metric which again comes from the fine granularity of the entities and the integrated security policy. Not only does security metric give a measure of the security of a system, but also it gives directions on how to make the system more secure. We will explain in detail later how each of these goals is achieved.

1.5 Thesis Organization

The rest of the thesis is organized as follows. Chapter 2 discusses previous work done in the area of policy specification, composition and decomposition of policies, consistency checking, and modeling different types of policy. Basics of access control policies and a brief overview of the SE-Linux policies are provided in Chapter 3. In Chapter 4, we explain the new integration framework for different policy types and provide details on how low-level policies and configurations can be checked and validated against a global high-level policy. Chapter 5 gives formal definition of desired properties and formalizes the approach described in Chapter 4. It is also proven that the new integration and consistency checking framework is safe. Chapter 6 describes the security metric and the steps taken in development of it. It also explains how the metric provides insights and directions for achieving "more" security and the "optimal" security. We finally conclude the thesis in Chapter 7.

2. RELATED WORKS

This chapter discusses the previous work done in the area of policies and modeling them. Each section is dedicated to a different aspect of this problem.

2.1 The Global Policy Languages

An essential part of a policy driven system is the global policy. Global policy is a set of high-level rules that are specified by the highest security authority within each organization. This policy is first written in English and then specified as rules using a high-level policy language. Global policy determines the high-level security requirements of an organization; the low-level rules and configuration must all comply with it. Figure 2.1 shows this hierarchy.

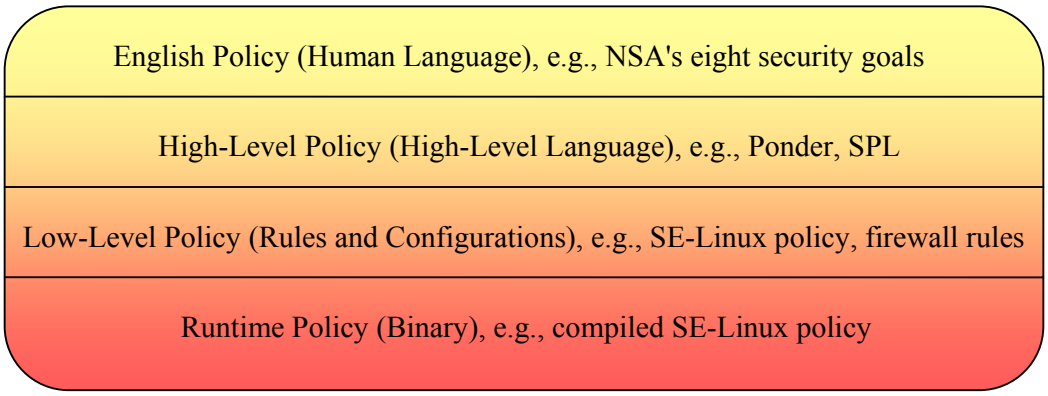

Figure 2.1 - Policy hierarchy

In this section we briefly describe a few commonly used high-level policy languages. These languages are different in their expressiveness, simplicity, and extendibility. Based on the specific application, one can choose any of them to express high-level security requirements for a system.

2.1.1 Ponder

Ponder is a declarative, object-oriented language for security policy specifications [12]. It is rich in declarative constructs which makes it suitable for different

implementation mechanisms including firewalls, operating systems, and Java firewalls, operating systems, and Java [12].

Ponder is capable of expressing access control, constraints, and obligation policies. Access control is supported by providing:

- Authorization policies
- Information filtering policies (suitable for firewalls)
- Delegation policies (temporary transfer of rights)
- Refrain policies

Obligation policies are event-based and they specify activities that must be done when some event is triggered. Tables 2.1 – 2.3 show the Ponder syntax for different policy types.

Table 2.1 shows the general syntax of authorization policies in Ponder. Ponder keywords "auth+" and "auth-" represent positive and negative authorizations, respectively. Each instance of an authorization policy specifies the "actions" that "subject" is allowed to perform on "target" (object). Furthermore, an authorization can be constrained with a Boolean expression optionally expressed with the "when" keyword.

The syntax of information filtering policies is shown in Table 2.2. Each filter executes transformations provided that the optional condition is true. "In" and "out" keywords are used to specify input and output parameters of the action on which the filter is specified. Finally, "result" keyword can be used to transform the returned value of the action.

Table 2.1 - Ponder Authorization Policy Syntax

```
inst ( auth+ | auth- ) policyName  "{"
      subject [<type>] domain-Scope-Expression;
      target [<type>] domain-Scope-Expression;
      action action-list;
   [when constraint-Expression ; ]  "}"
```

Table 2.2 - Ponder Information Filtering Syntax

```
actionName { filter }
filter = [ if condition ] "{" { ( in parameterName =
expression ; | out parameterName = expression ; |
result = expression ; ) } "}"
```

Table 2.3 - Ponder Delegation Policy Syntax

```
inst deleg+ "(" associated-auth-policy ")" policyName "{"
grantee [<type>] domain-Scope-Expression ;
[ subject [<type>] domain-Scope-Expression ; ]
[ target [<type>] domain-Scope-Expression ; ]
[ action action-list ; ]
[ when constraint-Expression ; ]
[ valid constraint-Expression ; ] "}"
```

Ponder also supports delegation policies which can be used to temporarily transfer access rights. Table 2.3 depicts the syntax of such delegation policies. "Grantee" is the subject who transfers a right to another subject. This delegation can be time limited using the keyword "valid" followed by time duration. All other keywords in this policy have the same meaning as those of authorization policy.

2.1.2 Security policy language (SPL)

SPL is a policy-oriented constraint-based language. It is capable of expressing permissions, prohibitions, and some restricted forms of obligations.

SPL has four major components: entities, sets, constraint rules, and policies. SPL entities are typed objects with an explicit interface through which their properties can be queried [13]. SPL also supports sets of entities to achieve compactness, generalization, and scalability. The constraint rules specify the restrictions that are imposed on the interactions between entities. These are the main rules defined by the language. Finally, the constraint rules can be grouped into policies.

SPL is also capable of using current or past events when ruling about interactions. As a result, it can keep a history of past events and determine a constraint. Obligation and invariant constraints are also expressible under SPL. Tables 2.4-2.6 show different components of SPL.

Table 2.4 depicts a sample hierarchy of entities. SPL supports custom definition of entities in an object-oriented fashion. Each entity is defined using the "type" keyword and has some properties. For example, an object's properties include its name, owner, and type while an event has properties such as time, action, etc. "Extends" keyword is used whenever an entity is derived from a root entity.

Table 2.4 - An SPL Entity Hierarchy Definition

```
type object {
      string name;       // The name of the object
      user owner;        // The owner of the object
      string type;       // A string identifying the type
      object set groups;// The sets containing the object
      string homeHost;  // The host where the user
}                        // is defined

type user extends object {
      rule set userPolicy; // User private policies
}
type operation extends object {
      number ID; // operation Id
}
type event extends object {
      user author;           // The author of the event
      object target;         // The target of the event
      operation action;      // The performed action
      object set parameter; // The set of parameters
      number time;           // The time instant
      object task;           // The task to which the event
}
```

Entities can be internal or external to the security service. External entities can include files, users, and events. To define external entities, SPL uses the "external" keyword as shown in Table 2.5. Queries on the interface of external entities are translated into method or function calls on the services of those entities.

Table 2.5 - Example of SPL External Entities and Sets

```
external string localhost;          // An external entity
external user set AllUsers;         // All the users
                                    // in the system
external object set AllObjects;     // All the objects
external operation set AllActions; // All the actions
external event set AllEvents;       // All the events,
                                    // past and future
```

Table - 2.6 SPL Rule Syntax

```
[label :] domain-expression :: decide-expression
```

Finally, Table 2.6 shows the syntax of an SPL rule. Each rule has a domain-expression which describes the domain of entities in terms of their properties. This domain is the set of entities about which the rule is making a proposition. The decide-expression specifies the decision made about the domain. This decision can either be a true/false or a Boolean expression putting some constraints on some other properties.

2.1.3 Policy description language (PDL)

PDL is an event-based policy language [14]. As actions in PDL are only event-triggered, the language is limited in expressiveness for access control or traffic restricting policies. Every rule in PDL looks like:

Event **causes** *action* **if** *condition*

in which events can be either system defined or policy defined. Actions are predefined and are given to the policy writer. A condition puts another constraint on the execution of the action. There can be a variety of constraints including the time of the event, past events, and the state of the system.

2.1.4 Data access list (DACL)

DACL is a high-level language specially designed to express access control policies. It specifies what operation a subject (also known as a *principal*) can perform on objects (*Data*). Table 2.7 shows the syntax of DACL.

Table 2.7 - The DACL Policy Language

```
Principal   ::= p | p ; Principal
     Data   ::= d | d ; Data
       Op   ::= read | write
DataAccRule ::= allow Principal Op Data
       DACL ::= DataAccRule | DataAccRule ;DACL
```

As DACL does not have notions of traffic and/or information flow, it is not useful for expressing the configurations of traffic control devices such as firewall rules.

2.1.5 Security assertion markup language (SAML)

SAML is an XML-based policy language for exchanging security information. There are three main assertion categories that can be expressed using SAML:

- Authentication assertions: Specify how a user has been authenticated
- Attribute assertions: Associate attributes with subjects in the system
- Authorization assertions: Specify whether or not a subject is authorized to do an operation

SAML also has a protocol which is a simple request-response protocol for exchanging elements.

It is important to note that SAML specifies how information should be exchanged and it is not for actually specifying the security policy. SAML must be complemented with a language for specifying the security policy. In most cases this language is XACML which we introduce in the next section.

2.1.6 eXtensible access control markup language (XACML)

XACML is again an XML-based language for specifying the security policy [15]. The XACML policy language has three major components: rules, policies, and policy sets. Figure 2.2 provides a sample policy written in XACML. Each rule has three parts:

- Target: is the entity about which the rule gives a statement
- Effect: is the decision made about the target (this can be a "permit" or "deny")
- Condition: puts an extra constraint on the target of the rule

A policy has four different components: a target, a rule-combining algorithm-identifier, a set of rules, and some obligations. Similarly, a policy set is made up of a set of policies. Extensibility makes XACML even more flexible for describing different policies in different systems, and potentially makes the language expressive for any type of security policy.

XACML has normative elements (key words) which define the structure of the policy. These are components such as subjects, resources, actions, etc. A comprehensive list of normative components is shown in the first column of Table 2.8 [15]. The second column shows whether each component is mandatory or optional to implement. Any other element in an XACML policy is context specific. For example, if the policy is used to control accesses of different people to various resources in a university, subjects can be defined as "faculty members," "students," "staff," etc., whereas in an operating system subjects may be "administrators," "account operators," "guest users," "remote users," etc. So, nonnormative elements in an XACML policy are platform dependent and should be defined based on the specific application for which the policy is being used. Detailed description of the normative elements of XACML can be found in [15].

More details about the language will be presented later in Chapter 5 when the formal model is discussed. We also provide a sample customization of the XACML for our framework in Chapter 5.

```
   <? xml version="1.0" encoding="UTF-8"?>
   <Policy
   xmlns="urn:oasis:names:tc:xacml:2.0:policy:schema:os"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="urn:oasis:names:tc:xacml:2.0:policy:schema:os
   http://docs.oasis-open.org/xacml/access_control-xacml-2.0-policy-
   schema-os.xsd"
   PolicyId="urn:oasis:names:tc:example:SimplePolicy1"
   RuleCombiningAlgId="identifier:rule-combining-algorithm:deny-
overrides">
   <Description>
   Medi Corp access control policy
   </Description>
   <Target/>
    <Rule
    RuleId= "urn:oasis:names:tc:xacml:2.0:example:SimpleRule1"
    Effect="Permit">
     <Description>
     Any subject with an e-mail name in the med.example.com domain
     can perform any action on any resource.
     </Description>
     <Target>
      <Subjects>
       <Subject>
        <SubjectMatch
        MatchId="urn:oasis:names:tc:xacml:1.0:function:rfc822Name-
match">
        <AttributeValue
        DataType="http://www.w3.org/2001/XMLSchema#string">
        med.example.com
        </AttributeValue>
        <SubjectAttributeDesignator
        AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
        DataType="urn:oasis:names:tc:xacml:1.0:data-
type:rfc822Name"/>
        </SubjectMatch>
       </Subject>
      </Subjects>
     </Target>
    </Rule>
   </Policy>
```

Figure 2.2- An XACML policy example

2.1.7 CDSFramework & SEFramework

These two frameworks provide a high level language for describing security policies. They are designed specifically for SE-Linux policies and any policy expressed in these frameworks can be directly compiled to an SE-Linux type enforcement policy [16].

The basic concepts in these frameworks are domains and interactions between them. *Domain* is a set of objects and subjects with the same security sensitivity.

Domains interact **only** through *shared resources* which are objects that are shared between two or more domains. Each domain is allowed to have only specific types of *access* on a shared resource. There is also the concept of *domain decomposition* in these frameworks, meaning that each domain can be decomposed to many domains and shared resources.

Table 2.8 – XACML Normative Elements

Element Name	Mandatory/Optional
Action	Mandatory
ActionAttributeDesignator	Mandatory
ActionMatch	Mandatory
Actions	Mandatory
AnyAction	Mandatory
AnyResource	Mandatory
AnySubject	Mandatory
Apply	Mandatory
AttributeAssignment	Optional
AttributeSelector	Optional
AttributeValue	Mandatory
Condition	Mandatory
Description	Mandatory
EnvironmentAttributeDesignator	Mandatory
Function	Mandatory
Obligation	Optional
Obligations	Optional
Policy	Mandatory
PolicyDefaults	Optional
PolicyIdReference	Mandatory
PolicySet	Mandatory
PolicySetDefaults	Optional
PolicySetIdReference	Mandatory
Resource	Mandatory
ResourceAttributeDesignator	Mandatory
ResourceMatch	Mandatory
Resources	Mandatory
Rule	Mandatory
Subject	Mandatory
SubjectMatch	Mandatory
Subjects	Mandatory
Target	Mandatory
XPathVersion	Optional

There are also other XML-based languages which are trust-based and use credentials and certificates (such as X.509) to specify decisions (e.g., Trust Policy Language-TPL and Akenti). Theses languages are the most useful ones for representing trust-based policies. As trust-based policies are not the main focus of this work, we will not describe the details involved in these languages.

2.2 Composition and Decomposition of Policies

There have been some works on composition and decomposition of policies. They mainly focus on one type of policy and try to formally compose/decompose different policies instances.

Bonatti et al. [17] and Jajodia et al. [6] propose a modular approach for composing access control policies. They formalize policies in an algebra and define valid operators for composing policies expressed in such a framework. Bhatti et al. [4] introduce a framework for expressing and composing Role-Based Access Control (RBAC) policies in federated systems using an XML based language. Bonatti et al. [5] and Di Vimercati and Samarati [18] deal with the general authorization problem in federated systems and explain models for specification and enforcement of such authorization policies.

Jajodia et al. [6] provide another framework for composing and simultaneously enforcing different access control policies in a system. They introduce a framework called flexible authorization manager (FAM) and formalize secure rules for composing different instances of policy.

Siewe et al. [7] discuss another model for composing access control policies in a web server to provide multiple services to each user in the service oriented computing (SoC) context. Taylor and Murty [8] develop another model for RBAC policies when sharing information on the web for applications such as GIS or biological databases.

Hale et al. [19] propose a ticket-based authorization model to ensure global consistency in heterogeneous systems with different access control policies.

Finally, [20] deals with the reverse problem, namely decomposing a global policy into multiple rule-sets for distributed policy enforcement points (PEPs).

2.3 Consistency of Policies

As mentioned earlier, there have been some works on consistency-checking for policies. They usually model a special type of security policy and try to formalize it. In the formalized form, a policy can be validated and checked for consistency either by looking for anomalies and logical inconsistencies or by validating against a global higher-level policy.

Most of the works mentioned in Section 2.2 for composition and decomposition of policies provide consistency checking functionality as well.

Different frameworks for policy consistency checking are described in [5], [6], [17]-[27]. They differ in their models and formalisms from propositional logic to special algebra. They also differ in their target policy from role-based access control policies to web-server service policy. However, they all propose models to formalize the policy and then check for consistency in that context. The advantage of this approach is that the model is usually mathematically sound and useful security and consistency properties can be proved about the policies.

There have also been some works on consistency checking for firewalls and network security devices. This problem is tackled using approaches such as modeling and data mining as in [28]-[31]. Uribe and Cheung [9] devise a framework for modeling firewall and network intrusion detection system (NIDS) rules to automatically check for consistency and optimization.

Nevertheless, the generic modeling approach used in the above frameworks has some subtleties. The most important problem with this approach is that not all real entities can be easily described in the model and there usually exists a strong trade-off between the simplicity, expressiveness, and accuracy of the model. As a result, these models are either hard to implement or very limited in their scope. Our work avoids this problem by using real-world entities (subjects and objects as defined in a SE-Linux operating system) which we discuss in the next sections. As a result, no extra modeling attempt is needed to map between real entities and model abstractions.

3. THE SECURITY-ENHANCED LINUX (SE-LINUX) POLICY

In this chapter, the two different types of access control – i.e., discretionary access control (DAC) and mandatory access control (MAC) – are reviewed and the major differences between them are discussed. We then briefly introduce one of the famous implementations of mandatory access control, which is the NSA SE-Linux. Type enforcement (TE), role-based access control (RBAC), and user identity (UI) will be discussed as different policy types in the SE-Linux.

After the SE-Linux is introduced, different modeling attempts and formalisms will be discussed in this regard. Our new framework will be based on the access control space model by G. Zanin and L. Mancini [11]. The new framework is discussed in the next chapter.

3.1 Access Control

Access control is one of the most famous and efficient security policies which can be implemented in various systems and environments. Access control defines in a fine-grained manner what active entities (subjects) can access a passive entity (object).

Subjects in a system can refer to different active entities depending on the level of abstraction (e.g., departments in a company, users in a system, etc.), but at the lowest abstraction level, they usually refer to processes which act on behalf of users.

Objects, on the other hand, are passive (or sometimes active) entities that subjects interact with. They can be anything from a file, database, log-file, socket, or network interface at the lowest level to different resources inside a corporation at a higher level.

Because in most systems subjects can also interact with other subjects, the set of different subjects in a system is a subset of the set of objects. This means that a subject in a system can be an object as well, but not vice versa.

Access control in a system is usually defined in the form of an access control matrix which specifies what operations a subject can perform on an object. In this notation, rows of the access control matrix are different subjects, columns are different objects, and each element is the rights that the subject has over the object.

There are two different types of access control which will be introduced in the following two sections.

3.1.1 Discretionary access control (DAC)

In discretionary access control, each subject (process) in the system is bound to an identity and the policy determines what resources in the system (objects) it can access. The main characteristic of DAC is that one user can pass its privileges to other users in the system. For example, consider the access control system implemented in the UNIX operating system. In this system, each file has an "owner" and the owner not only can have all different rights when accessing the file, but also it can pass its rights to any other user.

Discretionary access control has different problems depending on the specific implementation including "decidability" and "computational feasibility," but there is a more important problem that exists in all implementations of the DAC policy system, which is "unauthorized flow of information" (or information leakage). DAC assumes that the processes running on behalf of users always have perfect bound to users themselves and this is where the problem arises. To clarify, consider an example. Assume that in a company, an employee is trusted not to disclose sensitive information to non-trusted users. Each time this employee wants to work with sensitive data, he/she runs a program to do so. Now if anyone modifies this program to copy the sensitive data into a nonsensitive file which can be accessed by anyone, the next time the trusted employee runs the program, the information will leak to nontrusted users in the system. This problem exists because, although a user may be trusted, the process that works on behalf of him/her can be a spy ware or a Trojan horse. As DAC assumes any process running on behalf of a trusted user is itself trusted as well, the "information leakage" problem always exists in such a system.

The problem motivated the introduction of mandatory access control (MAC) as a means to enforce information flow constraints.

3.1.2 Mandatory access control (MAC)

In mandatory access control (MAC), all rights that a subject has or may gain to access an object are explicitly defined in the policy. As opposed to DAC, in mandatory access control, a user can NOT pass its rights to another user unless

explicitly stated in the policy. In this sense, the information can only flow to entities that are specified in the policy and Trojan horses cannot leak information.

When implementing a MAC system, the administrator of the system should specify all different valid accesses. After the policy is loaded into the system, no user can modify the policy or pass its rights to others.

There are two different types of MAC policies: "targeted" (open-world) policy and "strict" (closed-world) policy. In open-world policies, any access is allowed unless restricted by the policy. On the other hand, in closed-world policies, every access which is not explicitly allowed by the policy is denied. In most cases, a MAC system is built, tested, and debugged in open-world mode, and then migrated into a closed-world system to minimize false access denials. SE-Linux is a real world example of a mandatory access control implementation which will be discussed in the next section.

3.2 The SE-Linux Policy Language

This section reviews the main features of the SE-Linux policy language including its different policy types. For each type, different language constructs will be introduced and some example policies will be given wherever needed. A more detailed description of the language can be found in the NSA manuals for SE-Linux [32]. Implementation of the real-world policies using SE-Linux as well as the analysis of the sample policies can be found in [33] and [34].

3.2.1 Type enforcement (TE) policies

The type enforcement (TE) policy is one kind of policy in the SE-Linux policy language. In this policy, we associate a type with each subject (often called the "domain" of the subject) and each subject can only access the objects of the same type. Below, we describe the main language constructs used for type enforcement in the SE-Linux:

- Attribute: Defines different attributes for types. An attributes works as a macro and allows a rule governing many different types to be expressed in a compact form. For example,

  ```
  attribute domain;
  ```

defines an attribute to be assigned to processes.

- Type: Assigns names to different domains of subject and types of object. For example, the rule

  `type sshd_exec_t,file_type,exec_type,sysadmfile;`

  assigns type "sshd_exec_t" to all subjects and objects having attributes "file_type", " `exec_type` ", and "`sysadmfile`".

- Allow: The main construct of the TE policy, defines what object types a subject can access and what permissions it has over them. The syntax of this construct is as follows:

  `allow subject_domain object_type : object_class {permissions};`

  E.g.,

  `allow user_t bin_t : file { read execute } ;`

  allows each subject from "user_t" domain to "read" or "execute" "files" of type "bin_t".
- Constrain: Puts extra limitation on accesses. It states that an access can be granted only if a Boolean condition is true. This construct improves the expressiveness and compactness of the language in the sense that one can deny all the unwanted accesses and only allow access when a specific condition is true using a single rule instead of many different rules.
- Type transition: Has two different usages. It defines the type of newly generated objects and also the legitimate transitions of type for a process. E.g.,

  `type_transition sshd_t tmp_t:dir sshd_tmp_t;`

  says that when a "directory" of the type "tmp_t" is being created by a process of type "sshd_t", it will have the new type "sshd_tmp_t".

  `type_tansition initrc_t sshd_exec_t:process sshd_t;`

  allows a process of the type "initrc_t" to change its type to "sshd_t" when run by an executable of the type "sshd_exec_t".

In the SE-Linux, there are different permissions associated with each class of object. Table 3.1 lists different permissions for "file," "socket," and "dir" classes. Note that many of the permissions are common between different classes. For example, a directory has all the permission modes that a file has. In fact, the directory class *inherits* file permissions while it has some specific permission modes of its own.

Table 3.1 - Different Permissions for Important Some Object Classes

File	Socket	Dir
ioctl	ioctl	ioctl
read	read	read
write	write	write
create	create	create
getattr	getattr	getattr
setattr	setattr	setattr
lock	lock	lock
relabelfrom	relabelfrom	relabelfrom
relabelto	relabelto	relabelto
append	append	append
unlink	bind	unlink
link	connect	link
rename	listen	rename
execute	accept	execute
swapon	getopt	swapon
quotaon	setopt	quotaon
mounton	shutdown	mounton
	recvfrom	add_name
	sendto	remove_name
	recv_msg	reparent
	send_msg	search
	name_bind	rmdir

3.2.2 Role-based access control (RBAC) policies

Role-based access control is another type of policy implemented in the SE-Linux. In an RBAC system, there should be two different mappings: the first is the mapping between users and roles, and the second is the mapping between roles and permissions. In the SE-Linux RBAC, the first mapping is defined using the "user" construct, but the second mapping uses the type enforcement (TE) as its basis. As a result, there is a mapping between roles and types and allow rules are defined only based on types (and not roles). Note that roles are only assigned to subjects. Objects all have the generic role of "object_r". The major RBAC constructs used in SE-Linux are as follows:

- User: Assigns allowable roles for a user in the system. For example,
  `user root roles {user_r, sysadm_r};`
  says that user "root" can have roles "user_r" or "sysadm_r". Note that a user can have one and only one role at a time.
- Role: Defines the types (domains) that can be entered by each role. This is the middle ring of the chain that connects roles to permissions. As an example, the rule
  `role sysadm_r types {sysadm_t run_init_t};`
  states that the role "sysadm_r" is allowed to have types "`sysadm_t`" and "`run_init_t`".

- Dominance: Is the construct used to define user hierarchies in RBAC. It defines the roles which a higher role dominates. With domination, a higher role can enter any domain that its dominated role can. In this sense, it can **gain** access to any object which the dominated role has access to. An example of this rule is as follows:
  `dominance {role sysadm_r {role user_r}};`
  which says that the "sysadm_r" role can enter all domains that "user_r" can.
- Allow: This is the second usage of the allow construct. It can be used to define the set of legitimate roles that a specific role can change to. For example, the rule
  `allow sysadm_r {user_r logger_r};`
  means that a user with "sysadm_r" role can change its role to either "user_r" or "logger_r".

3.2.3 User identity (UI)

An important advantage of SE-Linux over the traditional UNIX system is that the identity of users in SE-Linux is **not** the standard user category (root, user, etc.). In the SE-Linux framework, the identity of a user is the role it has. As the roles are defined in the policy and legitimate role transitions are explicitly stated in the rules, a user cannot gain illegitimate rights by changing its category to a privileged one.

3.3 The SE-Linux Models

To the best of our knowledge, there have been two major modeling attempts for the SE-Linux policy system so far. These attempts try to formalize the SE-Linux policy into a framework from which they can derive interesting security properties. The first attempt has been made by Archer et al. [35]-[37] using the state machines. Zanin and Mancini [11] proposed the second framework based on access control spaces [10]. In the subsequent sections, we introduce these models.

3.3.1 The state-machine model

In this approach, the SE-Linux system is modeled as a state machine. In this model, a state is defined as "all objects in the system plus their security contexts (type, user, role, etc.)." As a result, a real SE-Linux state machine is composed of a large number of states.

The initial state of the system is defined as the state of Linux after initialization is done and the system is in a state ready for the user to start using it. The initialization period is skipped to avoid complexities of the system in this phase and to capture the system in a more useful state.

Transitions of the state machine are thus defined as access control decisions (creation of new objects, type or role transitions, etc.) made in the system plus modifications of the security contexts. In essence, the state machine changes state whenever a user changes its role, a new object is created, a process changes its type, a security context is modified, etc. State transitions in this framework are usually in the form of system calls.

Archer et al. [35]-[37] use TAME [38] and PVS [39] to prove properties about the state machine. Interesting security properties are mapped to invariant properties of the state machine and they are then checked for correctness using TAME. For example, the NSA's high level security goal "protect raw accesses to data" will be translated to the state-invariant property: "In any state, only subjects with security context c1 can access objects with security context c2." As another example, the NSA's high-level goal "Protect integrity of kernel, configurations and logs" can be captured in a transition-invariant property which looks like: "If an action changes the content of an

object with security context c1, the action must have resulted from a successful request of a subject with security context c2."

The problem with this model is that a real system consists of very large number of states and handling such a state machine requires a large amount of memory and is time consuming. Another more important problem is that not every high-level security goal can be translated easily to such invariant properties. As a result, although the model promises to map high-level security goals into simple properties of the state machine, in a real system, it is difficult and sometimes impossible to do so.

In the following section, we introduce the other model for the SE-Linux which uses simpler basis for its analysis (i.e., sets instead of state-machine). We believe it can be extended to include other device configurations and to integrate different policies.

3.3.2 The SE-Linux access control (SELAC) model

This model is a customized version of the access control spaces first introduced by [10]. The model is specialized for the SE-Linux policies by [11]. In this model, based on the rules inside the policy, one tries to build the set of accessible objects for a specific subject to answer the question: "Can subject S access object O in mode P?"

Note that the answer to this question is by no means trivial. A typical policy in SE-Linux is made up of tens of thousands of rules. What makes the situation worse is that usually there are many allowed type transitions, role changes, constraints, and role dominations in a typical SE-Linux policy. This makes the question hard to answer as a subject may not currently have access to a particular object, but it can **gain access** simply by an allowed role change, role domination, or a type transition.

The SELAC builds the accessible object set for a particular subject considering all the changes that may legitimately take place under the policy. As a result, if a subject has or may gain access to an object in a specific mode, the algorithm returns "true." On the other hand, if it does not have access and may not explicitly gain access to the object, the algorithm returns "false."

One may consider this model a conservative approach to the problem as it returns "true" even when the subject currently does not have access to the object, but it is worth noting that gaining access in most cases means a single legitimate system call

[11]. Consequently, although the subject may not currently have a specific access to an object, it is legitimate under the policy for it to gain access.

For each language construct in the policy, the SELAC model builds a set corresponding to it. By working with these sets, it specifies the set of accessible objects for a specific subject. We briefly review these sets in Table 3.2 using the same notation as [11] to avoid any confusion. The first column shows the name of the set, the second shows the rules or the sets used in building it, and the third gives a short description of it. One can refer to the paper itself for a more detailed discussion about each of these sets.

Table 3.2 - The Sets Used in the SELAC Model

Set Name	Rules or Sets	Description
A	Attribute	The set of all attributes defined for subject or objects
T	Type	The set of all different types
A(t)	Type	The set of attributes for a specific type
α(a)	Type	The set of types with a given attribute
R	Role	The set of all different roles
U	User	The set of all different user identities
O	U, R, T	The set of all security contexts UxRxT
D(r)	Dominance	The set of dominated roles for a specific role
R(u)	User	The set of roles that a specific user can have
T(r)	Role	The set of types that a specific role can enter
S	U, R(u), T(r)	The set of all valid security contexts for subjects UxR(u)xT(r)
C	Class	The set of all different classes
P(c)	Class	The set of permissions applicable for a specific class
P	P(c), C	The set of all permissions
M(s)	Allow	The set of permissions with respect to type for a subject domain
Do(s)	Type_transition	The set of labeling decisions made for newly created object by a specific subject
Dp(s)	Type_transition	The set of types that a specific subject can transform to
$R^{\rightarrow}(s)$	Allow	The set of allowed roles for a subject
$T^{\rightarrow}(s)$	Allow	The set of allowed types that a subject domain can transform to
O(c)	P(c), T, M(t)	The set of all valid security contexts for objects
Ω(c)	O(c), P(c)	The universe of a specific class O(c)xP(c)
∑(s,c)	Ω(c), M(s)	The space of specified permissions
X(s,c)	Ω(c), Constraint	The space of constrained permissions
Δ(s,c)	∑(s,c), X(s,c)	The space of authorized permissions ∑ \ X

In this context, if a pair (object o, mode m) is a member of $\Delta(s,c)$, the subject s can access the object o from the class c in mode m; otherwise, it cannot. In the actual implementation, the algorithm is recursive as it considers dominated roles and allowed type transitions as well.

This model has lower complexity than the previous one as it works with sets instead of state-machines. Although these sets might have a large number of members, working with sets is inherently easy. Translating a high-level goal into this model would be a series of membership tests that reveals whether a specific object is accessible to a subject or not. For confidentiality-related goals, we are mostly concerned about access modes used for getting data (read, getattr, listen, getopt, search, etc.) and for integrity-related ones, we are concerned about those modes used for writing data (write, setattr, connect, send_msg, sendto, create, append, etc.). As this framework is easy to implement and high-level security goals can be expressed in the form of low-complexity operations (i.e., membership tests), we build our integration framework described in the next chapter based on this model.

4. POLICY INTEGRATION AND GLOBAL ACCESSIBILITY SPACES

This chapter explains the new framework proposed for the integration of security policies. We focus on firewall rules and SE-Linux policies as examples of network and OS security policies, although the framework works for other types of policies as well. The framework can integrate network topology and firewall rules with SE-Linux policies to get an integrated view of the implemented security policy. This policy can then be validated against a global security policy specified for the network.

4.1 The Integration Scheme

In the new framework, we work with subjects and objects as defined in the SE-Linux framework. As these entities have fine-granularity and in fact for security purposes they present the finest grain one is interested in, they provide a strong basis for integration of security policies. They also allow the definition of security metric as is discussed in Chapter 6.

In our analysis, we consider a corporate networked computer system which has different departments and sections isolated by firewalls (network or host-based). Furthermore, some machines are protected by the SE-Linux policies. We are interested to know whether the implemented policy complies with the high-level policy or not. To do so, we assume that we have all the policy files from the machines with the SE-Linux and also the firewall rules as well as the complete network topology.

It is assumed, through our analysis, that the information about all the tunnels in the network is in hand. There are two tunnel configurations possible. The first one is when the tunnel server is in front of the firewalls. As, in this configuration, the firewall can still filter unwanted packets, the information about these tunnels is not important to our analysis. The second configuration, in which the tunnel server is behind the firewalls, poses problem though. As filtering based on ports cannot happen in this situation, for these tunnels, we have to assume full port access to the other subnet. Figures 4.1 and 4.2 show these two configurations.

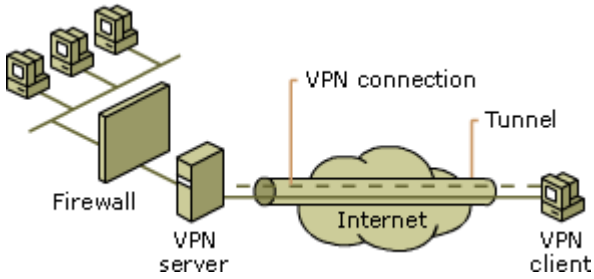

Figure 4.1 - Tunnel server in front of firewall (picture from www microsoft.com)

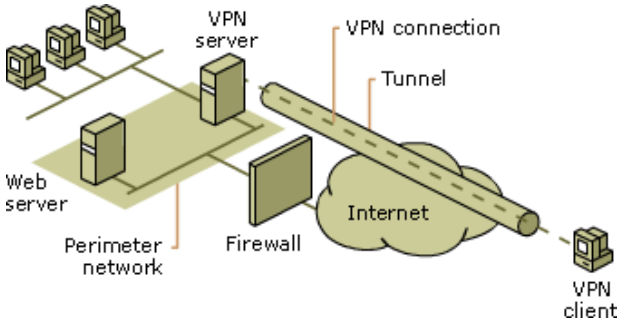

Figure 4.2 - Tunnel server behind firewall (picture from www microsoft.com)

In our framework, there is a network analysis engine that inputs the topology, the firewall rules, and the tunnel configurations and formally analyzes these data to answer questions like: "Can machine m1 send/receive packets to machine m2 on port p?" Since there are proposed methods in the literature for such an engine [2], [3], [9], [29]-[31], we do not talk about its details. Anytime we need to answer such a question, we simply call a function from that engine.

Based on the connectivity of the machines in the system and their network configuration, we build global accessibility spaces to know what resources (objects) on different machines in the network a specific user (subject) can access. This is done based on the SE-Linux network configuration policies as well as the access control policies. The latter is discussed in detail in Chapter 3. The former will be introduced in the next section.

4.2 The SE-Linux Network Configuration

SE-Linux has three constructs for network configuration: portcon, netifcon, and nodecon.

Portcon defines a type for each port number used in the system. A subject can access a port (listen, connect, send data, etc.) only if it is allowed explicitly in the policy to access that type of object (TE or RBAC). In this manner, SE-Linux protects ports from unauthorized accesses by applications.

Netifcon associates two types to each of the network interfaces on the machine. One of these types is used for labeling the messages that arrive to the interface, and the other is for the interface itself. Again, a subject can only use the network interface if it is allowed to access that specific object type.

Nodecon, in a similar fashion, assigns types to different nodes in the network. An application can send packets destined to a node only if it is allowed to access that type. This prevents some applications from communicating with specific nodes in the network (e.g., secure database server can be protected from unauthorized accesses). Below are sample rules from a network configuration.

```
# Ports
      portcon tcp 80   system_u:object_r:http_port_t
      portcon tcp 8080 system_u:object_r:http_port_t
# Network interfaces
      Netifcon eth0 system_u:object_r:netif_eth0_t
      system_u:object_r:netmsg_eth0_t
# Nodes
      Nodecon 10.1.2.30 255.255.255.255
      system_u:object_r:node_db_server_t
      Nodecon 10.1.2.0 255.255.255.0 system_u:object_r:node_any_t
```

4.3 The Global Accessibility Spaces

This section describes how global accessibility spaces are built using the network analysis engine, the SE-Linux network configurations, and the accessibility spaces of the SELAC model. These spaces specify the objects a subject can access in any mode on any machine in the network. In this sense, they present an integrated view of different policy types across the whole network. This is where the framework integrates various policies into a single global picture.

The logic behind a global accessibility set is as follow. Each network-enabled subject in a node has a set of accessible object-permission pairs on that node. This is called $\Delta(s,c)$ in the SELAC model. This subject can talk to some of the other subjects on other machines through the network. On the other hand, it cannot communicate

with some other subjects because the traffic is being filtered by the firewalls. For those subjects that it can talk to, it can also potentially gain access to their shared objects. To clarify, consider the following distinction. A database daemon (subject) shares the information in the database (shared objects) with the authorized entities (other subjects), but it does not normally give any information about the password file (private object) it has access to.

In the SELAC model, the accessible object sets for a subject are expressed for different classes of objects. For simplicity, we consider the union of all these sets with respect to classes to get one accessible object set for each subject (called Γ).

$$\Gamma(s) = \bigcup_{c \in C} \Delta(s,c) \tag{4.1}$$

The notation described in Table 4.1 will be used in the algorithms. To build the global accessibility sets, we have to consider the scenario in which an object travels multiple hops before reaching the destination. So a subject on machine m_1 may send an object to another subject on machine m_2 which may in turn send the object to a third subject on machine m_3. Note that this scenario is not captured in the network analysis engine because machines m_1 and m_3 may not be allowed to communicate directly. To account for such scenarios, a connectivity matrix is built using the algorithm in Table 4.2. The rows and columns of this matrix denote machines in the network. The element in row *a* and column *b* is a set which contains (sender subject, receiver subject) pairs if machine *a* has at least one subject that can send packets to another subject in machine *b* through ports and protocols accessible to them. Note that the communication is one-directional, so the matrix may not be symmetric.

Now we have to convolute the matrix to figure out where an object can ultimately reach using the algorithm in Table 4.3. The new matrix is denoted by Π^k in which *K* is the number of machines in the network. Note that for a middle hop to be able to pass an object there should exist a type with which the receiver subject marks the incoming objects and the sender subject can access.

Table 4.1 - Notations Used in the Algorithm

M	The set of all machines in the network
S(m)	The set of all subjects in a machine
IsConnected(IP_1,p_1,IP_2,p_2, protocol)	The network analyzer engine. The function returns "True" if IP address IP_1 can send packets from port p_1 to IP_2 on port p_2 using "protocol" and "False" otherwise. Protocol may be TCP, UDP, ICMP, etc.
PortType(p)	The function returns the type defined for the port number with "portcon" construct in the policy
GS(s,m)	The Global Accessibility Set for subject s on machine m
IsShared(o,p)	The function returns "True" if the object is a shared object across the network with permission p and "False" otherwise
Filter((m_1,s_1), (m_2,s_2))	The function returns "True" if the possibility of the two subjects communicating is filtered out by the administrator
Protocol(s)	The function returns the protocols over which subject s communicates. It can be TCP, UDP, etc.
$\Pi(m_1, m_2)$	The connectivity matrix. $\Pi(m_1, m_2)$ is the set of all subject pairs (s_1,s_2) such that s1 in machine m1 can send messages from port p1 to subject s2 in machine m2 on port p2 using some protocol and the communication is allowed by the firewall rules and the SE-Linux policy.
NewType(s)	The set of new types with which the subject s marks new objects. They are types defined by in the policy using "type_transition" construct.
P(s)	The set of all permissions allowed when s creates an object
Relabeling_allowed(m,o)	The function returns "true" if any subject in machine m can relabel object o. This comes directly from the "allow" construct in the SE-Linux policy.
O(m)	The set of all object types in machine m

Table 4.2 - The Algorithm for Building Connectivity Matrix

```
Π=0;
For each m_k∈M, m_j∈M
  For each s_t∈S(m_k), s_i∈S(m_j)
     For each pair of ports p_1,p_2
        For each protocol prot
             If (IsConnected(m_k,p_1,m_j,p_2,prot_1)∧
                (PortType(p_2),"recv_msg")∈Γ(s_i) ∧
                (PortType(p_1),"send_msg")∈Γ(s_t) ∧
                  prot∈Protocol(s_t) ∧
                  prot∈Protocol(s_i))
             {
                  Π(k,j)= Π(k,j)∪(s_t,s_i);
              }
```

Relabeling of objects must be allowed explicitly in the policy; otherwise, they are blocked by the operating system. If any subject in machine *m* can relabel an object, the object can become available to any sender subject. This is reflected in the "if" statement in Table 4.3. A sender subject s_3 can send an object o if it has access to it or any subject can relabel the object.

The convolution algorithm works as follows. First, note that Π contains one-hop connectivies; i.e., each entry contains the set of communicating subjects in adjacent machines. Π^k is initialized by Π which means, at the first iteration, it contains all one-hop communications. Next, for each two pairs of communicating subject on three machines ((s_1,s_2) and (s_3,s_4)), if the receiving subject in the middle machine can create an object which a sender subject on the same machine can read, the first and third machines are connected. In this case, we add the (initial sender, final receiver) pair to the appropriate element of the connectivity matrix. Note that the port and the communication protocol may change in the middle machine; i.e., the receiver subject (s_2) uses a different port and protocol for communication than the sender one (s_3). We are sure that the sender in the middle (s_3) can communicate with the subject in the third machine (machine *j*) using some port and protocol because (s_3,s_4) is in the corresponding element of Π initially.

Note that the first iteration of the convolution algorithm creates two-hop connectivity matrix; i.e., it indicates where an object can go after two hops of communication. After *K* iterations, the matrix indicates ultimate connectivity; i.e., it shows the ultimate places an object can reach in the network.

Table 4.3 - The Connectivity Matrix Convolution Algorithm

```
Π^k = Π;
Π^k_new= Π;
For convolution_count = 1 to K
   Π^k=Π^k_new;
   For each k = 1 to K
      For each u = 1 to K
         For each (s1,s2)∈ Π^k(k,u)
            For j = 1 to K
                For each (s3,s4)∈ Π(u,j)
                      For each type∈NewType(s2)
                         If ((type,"read")∈Γ(s3)  ∨
                              Relabeling_allowed(mu,type))
                         {
                          Π^k_new(k,j)= Π^k_new(k,j) ∪ (s1,s4);
                         }
```

The algorithm to build the global accessibility set for a specific subject (s_i) inside a specific machine (m_j) in pseudo-code is shown in Table 4.4. For simplicity, we have assumed that the network configuration does not restrict access to nodes and that each machine has one network interface.

First, for each two subjects, if they can communicate through one or more hops, all the accessible objects of the first subject are added to the second subject's accessible set. This set is called the "Global Accessibility Set." We do not know the semantics of the subjects and GSs are built only based on the fact that the rules (i.e., firewall rules and the OS access control rules) allow the two subjects to exchange their objects.

Table 4.4 - The Algorithm for Building Global Accessibility Sets

```
For each m_k∈M, m_j∈M
   For each (s_t,s_i)∈ Π^k(k,j)
         If (Filter((m_k,s_t),(m_j,s_i))= False)
         {
              For each o∈O(m_k)
                If(((o,"read")∈Γ(s_t) ∧ IsShared(o,"read"))
                     ∨ Relabeling_allowed(m_k,o))
                {
                GS(s_i,m_j)=GS(s_i,m_j) ∪ (m_k,o);
                }
          }
```

For each object and permission pair, only when the object is a "shared" object, it will be added to the "Global Set" of accessible objects for the subject. This set indicates the objects that are currently made available to the subject by the remote subject. Theses are objects which can be accessed legitimately through the policies and configurations without any attack. Note that pairs of objects and permissions are considered here. As a result, an object coupled with "read" permission might be "shared," but the same object with "write" permission can be considered as "non-shared." This situation happens very often for usual services. For example a web server or an ftp server shares many of its files just for "reading" and does not consider "writing" to those files a legitimate operation. As another example, a (log-file, append) object-permission pair might be "shared" as apposed to (log-file, read). Figure 4.3 depicts this concept.

Note that some of the information used in the analysis is taken from auxiliary files. This includes whether a subject shares an object over the network (IsShared function) and also the protocols that each subject uses for communication (Protocol

function). The auxiliary files can be provided to the analysis engine by the network administrator.

The "Filter" function above is a manually defined mask which the network administrator defines to make the analysis more meaningful. He can run the analysis first without any mask and reviews the results. There may be many false positive inconsistencies reported at this stage. Then he defines the mask to filter out communications which cannot occur semantically and reruns the analysis. The process continues until there is no inconsistency left. The filter adds some flexibility to the framework as not all subjects that are allowed to communicate under the rules, actually can communicate considering their semantics. It can be defined as a square matrix with subjects as rows and columns and all elements set to 1 initially. The administrator can zeroize each element if he thinks that the two subjects cannot possibly communicate.

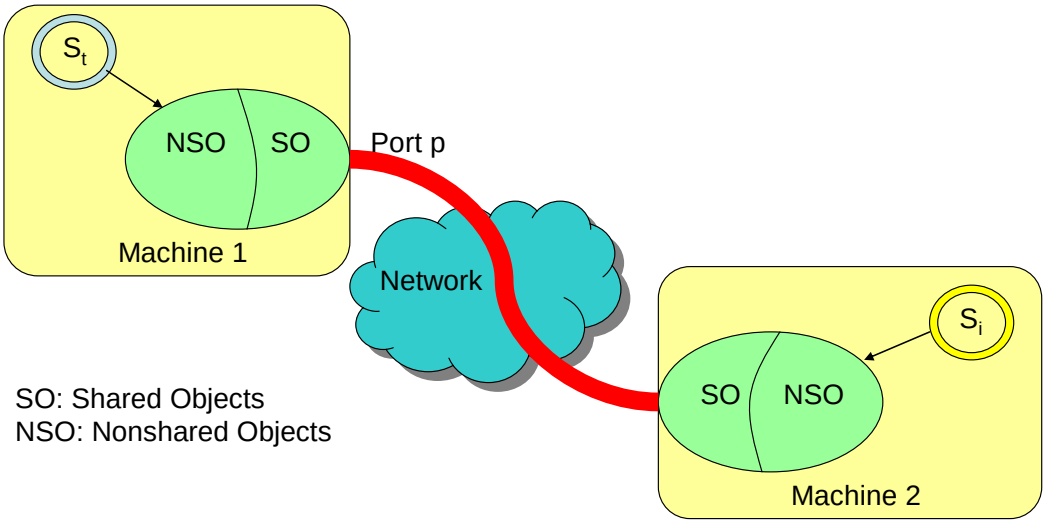

Figure 4.3- Shared and nonshared objects and accessible sets

For this analysis, one needs:

- The firewall rules and the configurations of the traffic shaping devices in addition to the topology for the network analyzer engine
- SE-Linux access control policies for building local accessible object sets
- SE-Linux network configurations for determining subjects' access to ports

It is important here to distinguish between events that initiate reanalysis and those which do not. By reanalysis, we mean the global accessibility sets must be rebuilt after some specific events. These events are as follows:

- Policy change in any policy enforcement point

- Physical topology change
- Static route modification (through routers)
- Rule change in any firewall (network and host F/Ws)
- Change in tunnels' status (i.e., up or down)

On the other hand, there are events which do NOT initiate reanalysis. This is either because they have already been included in the analysis or they have no effect on the analysis. The most important ones are as follows:

- Role change
- Type transitions
- Dynamic route change
- Activation or deactivation of tunnels outside firewalls
- Access change through domination

The following theorems formalize the three algorithms used for building global accessibility sets.

Theorem 4.1: After the K^{th} convolution, the convoluted connectivity matrix contains all the communicating pairs of subjects in the machines which are up to K-hops apart from each other. In other words, a subject pair (s_1,s_2) is a member of $\Pi^k(x,y)$ if and only if there exist a path of the length K or less of machines with communicating subjects that can pass an object considering the SE-Linux policy and firewall rules.

Proof: We prove the theorem by induction on convolution_count (see Table 4.3). For convolution_count = 1, the convoluted connectivity matrix is the same as the connectivity matrix. When building connectivity matrix, we check whether the two subjects are allowed to communicate using the specific protocol and ports and we include all the subject pairs that satisfy these conditions. So, the theorem holds for convolution_count = 1. Assume that the theorem holds for convolution_count = K-1; i.e., after convolving the connectivity matrix K-1 times, the convoluted matrix contains all subjects in machines which are up to K-1 hops apart and can pass objects. We prove that the theorem holds for convolution_count = K. Consider the subject s_α can send objects to the subject s_β and the two subjects are in machines which are K-1 hops apart (m_α and m_β). By the assumption, (s_α,s_β) is in $\Pi^{k-1}(\alpha,\beta)$. Now if s_β can send objects to any other subject in any machine (m_γ), the pair is included in the connectivity matrix $\Pi(\beta,\gamma)$. At the K^{th} step, as $(s_\alpha,s_\beta) \in \Pi^{k-1}(\alpha,\beta)$ and $(s_\beta,s_\gamma) \in \Pi(\beta,\gamma)$,

we add (s_α,s_γ) to $\Pi^k(\alpha,\gamma)$. Note that machine m_γ is K hops or less than K hops apart from m_α. Because we add all such subjects to Π^k, the theorem holds for convolution_count = K.

Theorem 4.2: Object o in machine m_1 is in the global accessibility set of subject s_{rN} in machine m_N if and only if there exist a subject s_{s1} in machine m_1 that can read o and share it over the network and also there exist a sequence of machines (hops) $\{m_1,\ldots,m_{i,\ldots,}\ m_N\}$ (numbered 1 to N without loss of generality) with a subject pair (s_{ri},s_{si}) for each machine m_i such that s_{si} can access an object type created by s_{ri} considering SE-Linux access control policy and s_{si} in machine m_i can send the object to $s_{r(i+1)}$ in machine m_{i+1} considering the network rules (firewall rules) and SE-Linux network configuration.

Proof: We prove each side of the theorem separately.

"If": Consider the sequence of hops $\{m_1,\ldots, m_i,\ldots, m_N\}$. s_{si} can talk to $s_{r(i+1)}$ considering their ports and protocols by the assumption. So after building the connectivity matrix, $(s_{si}, s_{r(i+1)}) \in \Pi(i,i+1)$ for i = 1 to N-1. s_{si} can access an object type created by s_{ri} (both in machine m_i). As a result, after convoluting Π using the algorithm in Table 4.3, $(s_{s1}, s_{rN}) \in \Pi^K(1,N)$. s_{s1} can read o, so (o, "read") $\in \Gamma(s_{s1})$. Now considering algorithm in Table 4.4, (o, "read") $\in \Gamma(s_{s1})$, $(s_{s1}, s_{rN}) \in \Pi^K(1,N)$, and o is shared; as a result, o will be added to the global accessibility set of $s_{rN.}$

"Only if": Object o in machine m_1 is in the global accessibility set of s_{rN} in machine m_N. Because of the algorithm in Table 4.4, (o, "read") $\in \Gamma(s_{s\alpha})$ and $(s_{s\alpha}, s_{rN}) \in \Pi^K(\alpha,N)$ in which $s_{s\alpha}$ is a subject in machine m_α that shares the object o. Using theorem 4.1, $s_{s\alpha}$ and s_{rN} are up to K hops apart and $s_{s\alpha}$ can send messages to s_{rN}. If $s_{s\alpha}$ and s_{rN} are 1 hop apart, $s_{s\alpha}$ can directly send the object to $s_{rN,}$ so the object can end up in the destination and the proof is done. If they are more than one hop apart (2 to K hops), theorem 4.1 asserts that there is a path of machines that can send the object one by one to the destination considering firewall and SE-Linux rules. Call these machines $\{m_\alpha, m_{\alpha2,} m_{\alpha3,\ \ldots,} m_{\alpha N}\}$ (N<=K). By corresponding the machines in theorem 4.1 to the machines in this theorem we have $m_1=m_{\alpha,}$ $m_{2=}$ $m_{\alpha2,}$ …, $m_N=m_{\alpha N}$ and the proof is done. As we have shown that for any object o is in the global accessibility set of s_{rN} such a path exist, an object is in the global accessibility set of a subject only if it can really be sent to it through multiple hops.

Finally, note that algorithms shown in Table 4.2, 4.3, and 4.4 are not ideally how one would implement the framework because of their high complexity. Future works will explore optimization of the algorithm.

4.4 Translation and Consistency Checking

Translation is the process of mapping the high-level security policy into low-level rules and configurations. The high-level security policy expresses the desired security properties of the system in terms of "high-level entities." For example, a high-level rule may state that "Only trusted users from the accounting department may access design plans inside the engineering department of a company." In this rule, "trusted users" and "design plans" are high-level entities, each of which refers to a set of lower level entities in the system. Translation, in this sense, is a one-to-many mapping defined on such entities. Figure 4.4 depicts this mapping process. These entities can be either objects or subjects.

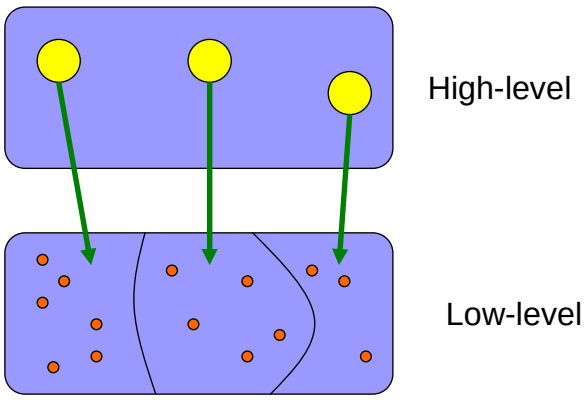

Figure 4.4 - Mapping high-level entities into low-level ones

To do such a translation, one has to partition the set of low-level entities into meaningful subsets. These subsets for objects indicate the measure of "criticality" of the objects. Depending on the granularity needed, the number of subsets can be any number from two to many different subsets. At the lowest granularity level, objects in a system are partitioned into "critical" and "noncritical" objects or "sensitive" and "nonsensitive" data. At higher granularity, one defines a criticality for each object in the system and this criticality measure can potentially be quantized into many different levels.

Different subsets for subjects represent their level of "trustworthiness." "Trustworthiness" here refers to a few different parameters, the most important of which are: the level of trust that can be put on the underlying user associated with the subject, the number of vulnerabilities in the subject, the ability of the subject to determine authenticity and integrity of its users and activities, the number of different attacks and exploits applicable to it, and the amount of interest one might have in hacking into the subject. The same discussion about the granularity of the measure applies for "trustworthiness."

Although the above scheme represents a meaningful mapping between high-level and low-level entities, the mapping can also be any arbitrary partitioning of the low-level entity space into any number of subsets. For example, if the high-level global policy is role-based access control, each subset of the subject space is associated with a role defined in the policy.

The information needed for translation can be stored in a few different ways. One way is to have a translation database. This database assigns a subset to each of the objects and subjects in the system. A sample table for objects is shown in Table 4.5. In this example there are only two different criticality levels available in the system.

TABLE 4.5 - A Translation Table for Objects

Objects	Criticality level
O1: kernel_t	Subset 1 (critical
O2: initrc_t	Subset 1 (critical)
O3: user_netscape_t	Subset 2 (non-critical)
O4: sysadm_t	Subset 1 (critical)
…	…

This information can also be stored as a few extra bits in the security context of the objects and the subjects. Figure 4.5 shows an example of such partitioning for 1-bit and 2-bit translations.

Note that the global high-level policy describes the high-level entities in the system. In this framework, having defined the translation mapping, the consistency checking process will be reduced to mostly set operations on the (conservative) global accessibility sets. In this context, of each rule in the global policy, high-level entities are translated into low-level ones. Then based on the type of rule (positive or negative authorization), compliance will depend on whether the low-level objects are members

of the subjects' global accessibility sets. As a result, one high-level rule may be expanded into many lower-level ones, each of which has low complexity (i.e., set operations).

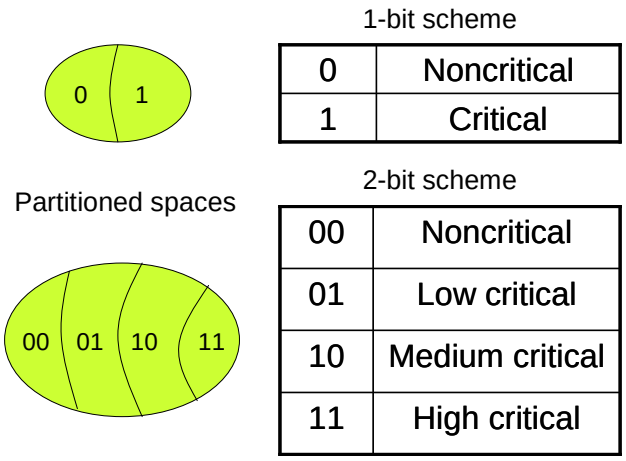

Figure 4.5 - Translation bits and partitioning the entity spaces

The consistency checking algorithm is as follows:

i. For each subject S in machine m_j in the high-level policy, determine the objects O_i , (i=1 to K) in machines m_i over which it has permission P_i (i = 1 to K). This is stated in the high-level policy and can be determined from the syntax of the policy language as described for XACML in Chapter 5. Note that S, O_i, and P_i are individual high-level entities and at the same time sets of low-level entities. For example, a high-level policy may restrict "guest users" to access "kernel modules". Here, "guest users" and "kernel modules" are high-level entities which are sets of low-level usernames and module files as well.

ii. Build K sets of object-permission pairs as follows: $L_i=\{m_i\}\times O_i \times P_i$.

iii. For every $s \in S$ and L_i (i=1 to K) IF $L_i \subset GS(s,m_j)$ THEN $\Lambda(s,m_j)= \Lambda(s,m_j) \cup L_i$ ELSE $\Theta(s,m_j)= \Theta(s,m_j) \cup (L_i / GS(s,m_j))$.

iv. Consider the union of all subjects S in the high-level, call it Ξ. Ξ^c (compliment of the set Ξ) denotes any subject in the system about which the high-level policy has no rule.

v. Now $\Theta(s,m)$ contains accesses that are allowed in the high-level policy and denied in the actual implementation and $GS(s,m)/\Lambda(s,m)$ contains those accesses which are denied by the high-level policy and denied in the actual

implementation. If $\Theta(s,m)$ and $GS(s,m)/\Lambda(s,m)$ are empty sets for all subjects $s \in \Xi$ and $GS(s,m)$ is empty for all $s \in \Xi^c$, then the low-level implementation is consistent with the global high-level policy and vice versa. Otherwise, report them as inconsistencies.

Note that O_i's and P_i's are the objects and permissions on every machine on the network, not just local objects.

Now we analyze the complexity of the algorithm described above. Assume that there are M machines on the network with S subjects, R objects, and P different permissions on each machine. Also assume that each subject is network-enabled with probability $p_{network}$ and two machines are connected with probability $p_{connectivity}$ (on any port). So, each network-enabled subject has $O(M \cdot R \cdot P \cdot p_{connectivity})$ objects-permissions pairs in its global accessibility set. There are totally $M \cdot S \cdot p_{network}$ network-enabled subjects in the network. Each union operation costs $O(M \cdot R \cdot P \cdot p_{connectivity} \cdot \log(M \cdot R \cdot P \cdot p_{connectivity}))$. Hence, the algorithm has overall complexity of $O(M^2 \cdot R \cdot P \cdot S \cdot p_{network} \cdot p_{connectivity} \cdot \log(M \cdot R \cdot P \cdot p_{connectivity}))$. The complexity of the algorithm depends strongly on the fraction of machines which are connected and the fraction of subjects that can communicate over the network.

The global policy might be richer than just high-level access controls; for instance, it may contain rules for the traffic flows or for the protocols used in the network, but most of these rules can be expressed in terms of one or more components of the framework. As a case in point, traffic shaping policies can be fully expressed in terms of firewall rules and topology. Table 4.6 summarizes the parts of the framework which capture each type of global policy.

Table 4.6 - Parts of the Framework Capturing each Policy Type

Type of global policy	Part of framework
Authorization & access control	Sets
Traffic shaping	Topology & Firewalls
Application layer policy	Sets & Firewalls
Authentication policy	Sets
Protocol policy	Firewalls
Quantitative security requirement	Sets & Metric & Firewalls

The next chapter formalizes the framework and the algorithm used here and we show that it is safe to map the high-level policies into the global accessibility spaces. Chapter 6 discusses how this framework can be used in the development of a security metric.

5. FORMAL MODEL

In this chapter, a formal model for mapping from high-level security policy to the global accessibility spaces is presented. The model shows how a real-world high-level security policy can be safely reduced to accessibility spaces. Formal definitions and proofs are presented to show that such a mapping is safe (safety is formally defined in Section 5.2). Without loss of generality, we focus on XACML as a high-level language and translate its constructs into global accessibility spaces.

5.1 Formal Mapping

An XACML rule consists of a target, an effect, and optionally a condition. Although other terms may be used in different policy languages, almost every language has these components. Target of a rule includes object (resource), subject (user), action, and optionally environment. To clarify the terms used here, consider the following example: assume that a rule gives read access to the student's transcript to the student's advisor if the system auditor is up for logging such an access. Here, subject is the *advisor*, resource is the *transcript*, action is *read*, environment is *whether or not the auditor is up*, and finally the effect is *allow*. The first four components are the target of the rule. Note that the difference between a condition and an environment is that the former limits accesses based on some properties of the subject, object, action, or a combination of them; whereas, the latter uses any other property. For example, here, whether or not the auditor is up is categorized as an environment because it is not a property of the subject, object, or action.

5.1.1 XACML features

Different rules in XACML can be combined into a policy. In XACML, a policy has a target, a rule combining algorithm, and a set of rules. The target of policy is the union of its rule targets. Rule combing algorithms specify how to consider the effect of different rules in a policy. Different rules target different sets of subjects and objects which can have intersections. What happens if rules have different effects with respect to an entity? This is where rule combining algorithms are applied. They resolve (possibly) contradictory effects of the rules into a final decision. Standard rule

combing algorithms include "deny-overrides," "permit-overrides," "first-applicable," "only-one-applies," etc. Consider the following example:

```
Policy
     Resource: university computing resources
     Subject: university people
     Action: any
          Rule1 (allow)
               Resource: Workstation machines
               Subject: students
               Action: login & use
          Rule2 (allow)
               Resource: university cluster machine
               Subject: faculty & RAs
               Action: initiate process
          Rule3 (deny)
               Resource: university cluster machine
               Subject: students
               Action: initiate process
```

Here, for simplicity and readability, the policy is written using generic terms but without formatting. Note that student research assistants are the intersection between rule 2 and rule 3. Now if the rule-combining algorithm is "permit-overrides" a student research assistant can initiate a process on the cluster machine; on the other hand, if it is "deny-overrides," such an access is denied.

In this work, we limit our model to rules with subject, resource (we use resource and object interchangeably here), action (also referred to as permission), and effect. For simplicity, conditions and environment are not modeled here, but the same steps can be taken to model those as well. Policies with different rule-combining algorithms are also formalized in this work. Policy sets can be built in the same fashion although they are not considered here for the sake of space.

Normative elements related to subjects, resources, action, and effect from Table 2.8 are used here (Action, AnyAction, AnyResource, AnySubject, Policy, PolicySet, Resource, Rule, Subject, and Target). *Effect* is an attribute of the *Rule* tag and can be either "Permit" or "Deny". *RuleCombiningAlgId* is an attribute of the *Policy* tag and can have any of these values: "deny-overrides," "permit-overrides," "first-applicable," or "only-one-applicable" as defined in [15]. For nonnormative parts of the XACML policy (e.g., subject names, object names, and permissions), the key-words should be defined by the administrator considering the desired granularity. For instance, at the highest granularity, subject names may simply be "Privileged users" and "Normal

users" each of which refers to a large number of subjects in SE-Linux context (examples of privileged subjects in SE-Linux are kernel_t, init_exec_t, init_t, etc.). At higher granularity level, subject names may be "Privileged users," "Network operators, " "Account operators," "Normal users," etc.

The same discussion applies for the object names. One customization of XACML can have object names as follows: "Kernel Objects" (maps to kernel_t), "Account Objects" (maps to user_t, passwd_t, passwd_exec_t, shadow_t, etc.), "Executables" (maps to bin_t), "Network Resources" (maps to socket_t, http_port_t, netif_intranet_t), etc.

Actions can also be defined at different granularities. In a simple case, actions can be "Read," "Write," and "Execute." They can be mapped to "ioctl, read, getattr, bind, getopt, recv_msg, listen, etc.," "write, create, append, setattr, lock, relabelto, setopt, send_msg, etc.," and "execute, shutdown, execmod, etc." in SE-Linux context respectively. In more fine-grained policies, actions may be defined differently.

5.1.2 XACML model

A rule in XACML can be easily mapped into global accessibility spaces considering its effect. The high-level policy expressed in XACML has different entities (subjects, resources, etc.), which themselves are sets of entities in a low-level policy (see Figure 4.4). Denote high-level subjects, objects, and permissions with S, O, and P, respectively. These are sets of low-level entities. For example, S in a high-level rule can be "faculty members" which consists of many low-level usernames.

If a rule permits some actions from a set of subjects over a set of resources, the cross product of the set of resources and that of permissions is added to the global accessibility space of all of the subjects in the rule; i.e., for each rule of the form

```
<Rule Effect="Permit">
<Target>
   <Subject>
        S
   </Subject>
   <Resource>
        O
   </Resource>
   <Action>
        P
   </Action>
</Target>
</Rule>
```

we have

$$\forall s \in S : \Delta(s,c) = \Delta(s,c) \cup O \times P \tag{5.1}$$

On the other hand, if the effect is "deny," all combinations of the objects and permissions should be removed from the accessibility space; i.e., for each rule of the form

```
<Rule Effect="Deny">
<Target>
   <Subject>
        S
   </Subject>
   <Resource>
        O
   </Resource>
   <Action>
        P
   </Action>
</Target>
</Rule>
```

we do

$$\forall s \in S : \Delta(s,c) = \Delta(s,c) / O \times P \tag{5.2}$$

Note that here we assumed that all objects are of the same class. If otherwise is true, the operations are done per class. As is evident, the mapping between single rules and accessibility spaces is fairly straightforward. But, in reality, the situation is not as simple because a policy has different rules and the rule-combining algorithm makes the final decision.

To model a policy, consider a generic XACML policy as below:

```
<Policy RuleCombiningAlgId="algorithm">
<Target>
   <Subject>
        S
   </Subject>
   <Resource>
        O
   </Resource>
   <Action>
        P
   </Action>
```

```
</Target>

<Rule1 Effect="Deny"> (S1,O1,P1)
</Rule>
<Rule2 Effect="Permit"> (S2,O2,P2)
</Rule>
<Rule3 Effect="Permit"> (S3,O3,P3)
</Rule>
…
</Policy>
```

Note that compact format is used to show policy and rule features for more readability. So, (S_1,O_1,P_1) means that the rule target is subject S_1, object O_1, and action (permission) P_1. Also note that different rules can (and probably do) intersect. As a result, one cannot model the rules one by one and independently. To consider the effect of the rule-combining algorithm and all the rules, the following algorithm is used to generate the accessibility spaces:

$$\begin{aligned} &\forall s \in S : \Delta(s,c) = \phi \\ &\forall s \in S, o \in O, p \in P: \\ &\textit{if } (\textit{decision} = \textit{deny}) \Rightarrow \Delta(s,c) = \Delta(s,c)/(o,p) \\ &\textit{if } (\textit{decision} = \textit{permit}) \Rightarrow \Delta(s,c) = \Delta(s,c) \cup (o,p) \end{aligned} \tag{5.3}$$

In words, for each entity in the policy, if the overall effect of different rules considering the rule-combining algorithm is to allow an access, the pair of (object, permission) is added to the accessibility set. Otherwise, it is subtracted from the set. To decide whether the decision is "deny" or "permit", one has to iterate through the rules combining their effects using the rule-combining algorithm. Generic codes for combining rules can be found in [15]. For example "deny-overrides" can be implemented with the following algorithm [15]:

```
   Decision denyOverridesRuleCombiningAlgorithm(Rule
rule[])
   {
      Boolean atLeastOneError = false;
      Boolean potentialDeny = false;
      Boolean atLeastOnePermit = false;
      for( i=0 ; i < lengthOf(rules) ; i++ )
      {
         Decision decision = evaluate(rule[i]);
         if (decision == Deny)
         {
```

```
            return Deny;
         }
         if (decision == Permit)
         {
            atLeastOnePermit = true;
            continue;
         }
         if (decision == NotApplicable)
         {
            continue;
         }
         if (decision == Indeterminate)
         {
            atLeastOneError = true;
            if (effect(rule[i]) == Deny)
            {
               potentialDeny = true;
            }
         continue;
         }
      }
      if (potentialDeny)
      {
      return Indeterminate;
      }
      if (atLeastOnePermit)
      {
      return Permit;
      }
      if (atLeastOneError)
      {
      return Indeterminate;
      }
      return NotApplicable;
   }
```

5.2 Safety Property

In the previous section, a formal model for mapping a high-level policy into global accessibility spaces was provided in detail. Following the same semantics and analogous terms, it is possible to model access control policies of this type into the global accessibility framework. But, what guarantees that this approach is safe? More importantly, what is safety?

In this section, formal definitions and proofs are provided to show that modeling a policy using the method described above is secure. For simplicity, we assume that

there are many rules and one policy in the system. The same approach can be used for policy sets and policy-combing algorithms.

We start by defining some terms. Some of the terms and definitions used here are similar to those in [40].

Definition 5.1: A mapping from one representation of a policy (e.g., high-level representation) into another representation of the same policy (e.g., low-level) is said to *leak* right r if a subject gets the right r over an object o under the second representation which it does not have under the first one.

Definition 5.2: A mapping is *safe* with respect to right r if it does not leak it. Otherwise, it is called *unsafe* with respect to right r.

In the proofs involving access control models, the term *safety* is used instead of security to differentiate between the abstract model and its implementation. Safety is the property of the former whereas security is of the latter.

To prove that the mapping is safe, let M denote the system that maps the high-level policy represented using XACML into the low-level policy represented using the global accessibility sets using the algorithm (5.3) described in the previous section. We have:

Theorem 5.1: System M is safe with respect to all rights $r \in P$.

Proof: Proof by contradiction. Assume that M is not safe with respect to some right $r \in P$. It means that a subject s can get right r over an object o in the "global accessibility sets" framework that it does not have in the XACML high-level framework. If the decision is "deny," we do not add any member to the accessibility sets, so this operation cannot leak any right. We also know that the sets are empty at the beginning. So, every right is added when the decision is "permit." Without loss of generality, assume that the rule-combining algorithm is "deny-overrides." As the final decision is "permit," at least one rule has the effect of "permit" and the others have the effect of either "permit" or "Not-Applicable" (see the algorithm (DenyOverrides) in the previous section). But this is a contradiction because, in this case, the high-level policy allows the access and subject s has had the right r over object o before the mapping. Same reasoning applies for any other rule-combining algorithm as well; for example, for "permit-overrides," all rule effects have to be either "permit" or "Not-Applicable." Hence, system M is safe with respect to any right $r \in P$.

Theorem 5.1 proves that any mapping from a single policy instance to the global accessibility framework is safe with respect to all rights. In a complex system, high-

level requirements may be expressed in terms of a policy-set which are instances of policies combined using a policy-combining algorithm. The same reasoning applies for policy-combining algorithms and, as a result, mapping policy-sets does not leak rights either. Corollary 5.2 formalizes the above statement.

Corollary 5.2: If a high-level policy set Ψ denies subject S permission P over object O and the global accessibility sets are built using the algorithm (5.3), there is no member of the form (o, p) with $o \in O$ and $p \in P$ in the accessibility set of any subject $s \in S$.

Proof: Follows from Theorem 5.1.

Theorem 5.3: A low-level set of firewall rules and OS (SE-Linux) access control policy consistently implement a global policy expressed in XACML provided that they pass the consistency checking algorithm (page 36). In other words, if conditions of the consistency checking algorithm are satisfied, the low-level policy implies the high-level global policy and vice versa.

Proof: First we assume that the high-level policy expressed using XACML is not ambiguous if it contains any required element described in the XACML standard [15]. Specifically, the policy must specify rule combining algorithms for each instance of policy and a policy combining algorithm for the policy set. If we use algorithm (5.3) to map the high-level policy into our framework, the *decision* is always known. As mentioned earlier, we go through the rules and policies one by one combining them using rule combining algorithms and policy combining algorithms, respectively, to determine the accessible objects for each subject. We continue by proving the two implications.

"The high-level policy implies the low-level policy": This implication means that any access that is allowed by the high-level policy is also allowed by the low-level policy. As they passed the consistency checking algorithm, the set Φ is empty. As a result, no access allowed by the high-level policy is denied by the low-level one.

"The low-level policy implies the high-level policy": This part is a little more demanding. First, because of Corollary 5.2, when the high-level policy is mapped to the low-level one, there is no leakage of rights. Second, as they pass the consistency checking algorithm, the set GS/Λ is empty for all subjects in the high-level policy. So for any subject in the high-level policy, the low-level policy does not allow any access which is denied by the former. The only remaining concerns are subjects about which the global policy does not have any statement. Because the conditions of the

consistency checking algorithm are satisfied, the set GS is empty for all those subjects and the statement follows.

In this chapter, the formal model of the framework and the mapping process are provided and it is shown that the two policies are consistent if they pass the consistency checking algorithm. As a result, this chapter provides the formal foundation for consistency checking described in Chapter 4.

6. SECURITY METRIC DEVELOPMENT

This chapter explains the ideas about a security metric as well as the role of the new framework in its development. It will be shown that because of fine granularity of the entities in the framework, one can develop a measure of security based on the properties of these entities, their interactions, and access restrictions. It is important to note that this metric is applicable in any security framework with arbitrary resolution in which the core of the policy can be expressed in terms of access control.

6.1 High-Level View

A security metric is a quantitative or qualitative measure of how much a system meets it security objectives. Security objectives are usually stated in the form of basic security components, i.e., confidentiality, integrity, and availability. A real system needs all or some of these basic components with different priorities. For example, a military system usually focuses more on confidentiality while a commercial system requires integrity more than other components. This shows that a good security metric should be system specific.

A good security metric must have some high-level properties which are described below:

- **Well-defined:** Security metric must be well-defined and meaningful. Being well-defined refers to the fact that the metric strictly increases when correctly applying security safeguards. In addition, it must truly refer to the extent to which the system complies with the security requirements. That means a system which has a higher rank with respect to the metric must be more secure in reality.
- **Comprehensive:** A good security metric captures the effect of all security safeguards in a system. This may include but is not limited to OS security, firewalls, intrusion detection systems, secure protocols, secure applications, authentication, encryption, etc.
- **Simple output:** It may be desirable that a security metric can express the degree to which a system is secure in the form of a simple output (e.g., a

scalar). Therefore, a detailed log file that lists all the security holes in the system is not considered a simple output in this sense.

- **Directive:** A good metric should suggest how to increase security of the system by showing the least secure components of it. It should also be possible to infer some information about the new safeguards to put in the system.
- **Easy to compute:** It should not take lots of effort and computational power to compute the value of a security metric. Note that the computational complexity of finding a security value for a system depends on the complexity of the system, the amount of details involved in the computation, and the granularity of the analysis. Having said that, it must be possible to find that value with a reasonable amount of effort and computation with relative precision and granularity.
- **Scalable:** The security metric must be defined in way which makes it useable for large systems. This can be done by balancing the granularity-scalability tradeoff. In small systems, the analysis is done on fine-grained entities, whereas in larger systems it runs on coarser and coarser entities.
- **System specific:** As mentioned earlier, a useful metric must consider the specifics about a system, its goals, and its security objectives to be meaningful and flexible for different systems.

6.2 Definition of the Metric

This section describes the details of the proposed security metric. We first present a primitive version of the metric and then enhance it with more details in order to better quantify the security measure of the system.

6.2.1 Primitive version

The new integration framework expresses different security policies in terms of the interaction between processes or users (subject) and resources (objects). It combines firewall rules with SE-Linux policies and network configurations to get a global view of such an interaction inside a network. The Global Accessibility Sets described in the previous chapter answer the basic question "What objects can a

subject access and in which modes?" These sets contain enough information to describe what actual policy the low-level rules and configurations are implementing inside a network.

It is intuitive to derive the global access control matrix (ACM) from the sets. Rows of ACM refer to different subjects and columns to objects. Each entry contains different modes of access that the subject has on the object. For simplicity, assume that there is only one mode of access which makes the ACM a binary matrix. Below is a sample ACM.

Every access that is allowed under the policy may not happen in reality. To capture this, we allow the administrator to define a filter matrix, F_{nxm}, which will be used to mask accesses that cannot occur under process semantics. It is a binary matrix with each element set to 1 initially and zeroized by the administrator selectively. The filter matrix adds some flexibility to the framework by masking unrealistic entries in the access control matrix. The Hadamard product (entrywise product) of the access control matrix (ACM) and the filter matrix gives the effective access control matrix (referred to as EACM hereafter).

Now we consider the effect of different types of objects with different security sensitivities. Loosely speaking, critical objects are those which, when attacked, severely endanger the overall security of the system whereas less-critical objects have smaller overall security impact. Examples of critical objects can be kernel modules or password files while noncritical objects can include some user configuration files or log files. Note that criticality of objects varies from system to system. In fact, non-critical objects in a system might be considered critical in others. For example, log files in a server may be considered critical because they are used to detect attacks and track attackers.

$$\begin{array}{c} \\ s_1 \\ s_2 \\ s_3 \\ s_m \end{array} \begin{array}{c} \begin{array}{cccc} o_1 & o_2 & o_3 & o_n \end{array} \\ \begin{bmatrix} 1 & 0 & 1 & \ldots \\ 0 & 0 & 1 & \ldots \\ 1 & 1 & 0 & \ldots \\ \ldots & \ldots & \ldots & \ldots \end{bmatrix} \end{array} \tag{6.1}$$

These criticality values can be the same values as those used when translating a high-level policy to low-level one. These values can be specified either by the chief

security officer of the network or by using an automated algorithm which assigns a criticality value to each object considering its type, its closeness to the kernel, and some heuristics. We arrange these values in a column vector and call it the "criticality vector." Below is such a vector in a 2-bit ranking scheme.

$$C_{m\times 1} = \begin{matrix} o_1 \\ o_2 \\ o_3 \\ o_4 \\ \dots \end{matrix} \begin{bmatrix} 2 \\ 0 \\ 2 \\ 3 \\ \dots \end{bmatrix} \tag{6.2}$$

In the same way, subjects in the system are ranked with respect to their "trustworthiness" with smaller numbers showing more trustworthy subjects. Trustworthiness of a subject depends, among other things, on its importance to the overall security of the system, its usage of secure protocols, and the number of known vulnerabilities for it. These values are arranged in a row vector, called "trustworthiness vector."

$$\begin{matrix} & s_1 & s_2 & s_3 & s_4 & \dots \end{matrix}$$
$$T_{1\times n} = \begin{bmatrix} 3 & 1 & 0 & 2 & \dots \end{bmatrix} \tag{6.3}$$

Multiplying these four matrices together, one gets a scalar value which shows the overall vulnerability of the system. We call this value the "vulnerability scalar" (VS). The symbol "$\bullet$ " denotes the Hadamard product (entrywise product) of *A* (ACM) and *F* (filter matrix).

$$T_{1\times n} \times (A_{n\times m} \bullet F_{n\times m}) \times C_{m\times 1} = \text{Vulnerability Scalar} \tag{6.4}$$

Multiplying the trustworthiness vector by each EACM column gives a number showing how subjects with different trustworthiness values in the system interact with a specific object. If access control limits the access to an object to trustworthy subjects only (with t-value = 0), then this product is zero. The more untrustworthy subjects can access the object, the greater the product will be. In the same fashion,

multiplying each row of the access matrix by the criticality vector gives a number showing the interaction of a subject with different objects in the system.

In a real-world system, all of the entries in the access control matrix may not be specified. This happens either because the policy has no statement about a subject-object pair or the policy maker is open regarding that type of access. These "don't cares" in the policy give the system some degrees of freedom to implement unspecified portions of policy arbitrarily. Some implementations permit all unspecified accesses whereas others deny such accesses.

A "closed-world" implementation (which is called the "strict-mode" in SE-Linux) minimizes the vulnerability scalar. This means that it denies any access which is not specified and effectively puts zeros in every place in the ACM that is a "don't care."

$$\min\left\|T \times (A \bullet F) \times C\right\| \tag{6.5}$$

On the other hand, an "open-world" implementation allows any access which is not explicitly restricted, which means putting one in any empty entry in the ACM. This implementation is called the "targeted-mode" in the context of SE-Linux. This implementation maximizes the vulnerability scalar.

$$\max\left\|T \times (A \bullet F) \times C\right\| \tag{6.6}$$

A better approach to deal with unspecified policies is to maximize accessibility of the system while minimizing its vulnerability. One approach is to minimize the vulnerability per accessibility as in (6.7). This solution minimizes the ratio of the VS and the norm of ACM, i.e.:

$$\min\frac{\left\|T \times (A \bullet F) \times C\right\|}{\left\|A\right\|} \Rightarrow \min\frac{\left\|\sum_{i=1}^{n.m} \alpha_i \cdot x_i\right\|}{\left\|\sum_{i=1}^{n.m} x_i\right\|}, x_i \in \{0,1\} \tag{6.7}$$

In the above problem, x_i's are the elements of the effective access control matrix (EACM). This problem is a mixed integer linear programming (MILP) problem and can be solved using different MILP techniques.

6.2.2 Extending the metric

In the previous section, a primitive version of the security metric was defined. This metric gives a measure of security using three major components: criticality of objects, trustworthiness of subjects, and access restrictions. The metric defined has a linear additive form with respect to criticality and trustworthiness values. In defining such a metric, there is a strong assumption which may not hold in real systems; i.e., the system is equally sensitive to all objects/subjects and also all objects/subjects are independent of each other. In a real system, there are correlations between objects/subjects, and the system also has varying degrees of sensitivity with respect to each of these entities.

Different sensitivities with respect to objects (or subjects) just scale the criticality (or trustworthiness) matrix. So, the effect of sensitivity can be reflected in C and T matrices objectively (by the administrator of the network).

Correlation between objects and/or subjects is another complexity that exists in real systems. The security of one object may improve the security of the other, a breached shared resource may deteriorate the security of objects sharing it, etc. Also, in some cases, objects are organized in a layered fashion with those in the lower layers providing some service to the higher ones. These are real situations in which objects/subjects are NOT independent of each other and in fact there is a strong correlation between them.

The simple form of the vulnerability scalar in (6.4) is linear with respect to c_i's. Each term contains one c_i and the vulnerability scalar is the summation of these terms. Because of its linearity, the simple version is not capable of capturing the correlation between objects. To capture the effect of different correlations, the definition of the VS is augmented with nonlinear terms with unknown multipliers to be solved using desired correlation values.

Nonlinear terms are those which contain two or more c_i's. They are added to the VS definition to express correlation between objects and introduce nonlinearity. All combinations of two or more criticality values are added with unknown multipliers to the definition of VS. To find these multipliers, we have to solve a system of equations with desired correlation values. To clarify, we group terms with the same number of c_i's together; i.e.,

$$VS = \sum \alpha_i . c_i + \sum \beta_{ij} . c_i . c_j + \sum \delta_{ijk} . c_i . c_j . c_k + ... \tag{6.8}$$

Denoting sensitivity of the VS with respect to c_i by S_{ci} and correlation between c_i and c_j by $S_{ci,cj}$, it follows that

$$Sc_i = \frac{\partial f}{\partial c_i} = \alpha_i + \sum \beta_{ij} . c_j + \sum \delta_{ijk} . c_j . c_k + ... \tag{6.9}$$

$$Sc_i , c_j = \frac{\partial f}{\partial c_i \partial c_j} = \beta_{ij} + \sum \delta_{ijk} . c_k + . \tag{6.10}$$

Note that β_{ij} is not necessarily the same as β_{ji} as objects may be correlated asymmetrically. For example, assume that the first object is a file containing a cryptographic key and the second one is a log file encrypted with that key. If the key is attacked, the log file is not secure any more; the reverse, however, is not true.

In the worst case, these relations give a system of 2^m equations with 2^m unknowns. One can solve the system to get all unknown multipliers. However, note that in real cases, the problem may be simpler than this. In real situations, if one is not interested in correlations beyond two or three terms, the number of equations to solve can be far less than 2^m. The total number of equations to solve is

$$\text{Number of Equations} = \sum_{i=1}^{k} \binom{m}{i} \tag{6.11}$$

where k is the maximum depth of correlation we are interested in. For example in a system in which all entities are correlated in pairs, the correlation depth $k = 2$.

An example here clarifies the process of finding unknown multipliers. Consider a system which has two subjects: a trustworthy user (trustworthiness-values = 1) and an untrustworthy user (trustworthiness-values = 3). For simplicity, assume that there are only four objects in the system: a kernel module, a file containing two keys, a file containing usernames/passwords encrypted by one of the keys, and a log file encrypted by the other key (number these objects 1 to 4, respectively). Also assume

that the criticality -values of these objects are 4, 5, 2, and 3, respectively. The access control policy defines the effective access control matrix (EACM) as

$$A = \begin{bmatrix} 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}$$

If we do not consider the effect of correlations (hence, using the simple version of the metric), we have

$$VS = \begin{bmatrix} 1 & 3 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix} \times \begin{bmatrix} 4 \\ 5 \\ 2 \\ 3 \end{bmatrix} = 42$$

Now consider the effect of correlations. Security of the log file and the username file is dependent completely on the security of the key file. So, we have

$$Sc_3, c_2 = \beta_{23} = 1$$
$$Sc_4, c_2 = \beta_{24} = 1$$

Any higher order term is zero. The new vulnerability scalar becomes

$$VS = \sum \alpha_i . a_i . c_i + \sum \beta_{ij} . c_i . c_j = 42 + (1.2.5) + (1.3.5) = 67$$

The new vulnerability scalar is higher than the simple version. This is because the system is less secure than the same system with independent objects. Note that here, if the key file is attacked, both the username file and the log file are no longer secure. So the system is less secure than what is shown by the simple version of the metric (hence, higher vulnerability scalar value).

Correlation between objects can come from the functional or logical dependence. This, for example, means that one object must be attacked before the other one can be endangered. Such correlation happens when there is a hierarchical or functional dependence between the two objects. In such cases, the insecurity of an object either makes the other object completely insecure or degrades the security of the other one. The examples for these cases are when the insecurity of a key makes the encrypted message insecure or when insecurity of a folder degrades the security of the files it contains.

The logical correlation can be described and quantified using functional graphs and hierarchies of objects in the system. Derivation of exact correlation values is beyond the scope of this work.

7. CONCLUSION AND FUTURE WORK

This work uses the concept of access control spaces to integrate different types of policy in an autonomous system. Using a single underlying framework based on fine grained entities in the system, it is possible to get an integrated view of the policy enforced in the system. This integrated view can be used to validate the enforced policy upon a global high-level policy and perform consistency checking.

The model assumes that the policy enforcement mechanism is secure (tamper resistant) while any other services in the system can be insecure. The introduction of Conservative Global Accessibility Sets makes it possible to check for potential future attack on services that currently have no vulnerability or are assumed to be secure.

A formal algorithm for mapping between a high-level policy language and the global accessibility sets is discussed, and formal proofs are provided to show that such a mapping is safe and global accessibility spaces capture all that is needed for consistency checking.

Finally, the work defines a security metric based on the accessibility of entities and their criticality and trustworthiness ranking. This metric can be used in any system in which the policy can be described in terms of access control. Entities in the metric can have different resolutions depending on the level of detail desired. Entities can be low-level objects and subjects as those defined in the SE-Linux system, or they can be large subsets of them as defined in a high-level policy.

Future extension of this project includes expansion of the framework to include different policy models such as multilevel security (MLS) or multicategory security (MCS). The work can be extended to express policies from other network security or traffic shaping devices such as intrusion detection systems (IDS). Also, the work on security metric is open-ended, and future work can address details for definition of sensitivities and correlations, ranking objects/subjects, and evaluation of different choices against real scenarios.

REFERENCES

[1] X. Ou, S. Govindavajhala, and A. Appel, "Network security management with high-level security policies," Computer Science Dept., Princeton University, Technical Report TR-714-04, 2004.

[2] S. M. Bellovin and W. R. Cheswick, "Network firewalls," *Communications Magazine, IEEE*, vol. 32, no. 9, pp. 50-57, Sept. 1994.

[3] K. Ingham and S. Forrest, "A history and survey of network firewalls," Department of Computer Science, University of New Mexico, Albuquerque, NM, Technical Report 2002-37, 2002.

[4] R. Bhatti, E. Bertino, and A. Ghafoor, "A policy framework for access management in federated information sharing," in *Proceedings of 2005 IFIP TC-11 WG 11.1 & WG 11.5 Joint Working Conference on Security Management, Integrity, and Internal Control in Information Systems*, December 2005.

[5] P. Bonatti, S. De Capitani di Vimercati, and P. Samarati, "An algebra for composing access control policies," *ACM Trans. on Inf. System Security*, vol. 5, pp. 1-35, Feb. 2002.

[6] S. Jajodia, P. Samarati, V. S. Subrahmanian, and E. Bertino, "A unified framework for enforcing multiple access control policies," in *Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data*, Tucson, Arizona, May 11 - 15, 1997, pp. 474-485.

[7] F. Siewe, H. Janicke, and K. Jones, "Dynamic access control policies and web service composition," in *Proceedings of the 1st Young Researchers Workshop on Service Oriented Computing (YR-SOC 05)*, 2005.

[8] K. Taylor and J. Murty, "Implementing role based access control for federated information systems on the web," in *Proceedings of the Australasian Information Security Workshop Conference on ACSW Frontiers 2003*, vol. 34, Adelaide, Australia, pp. 87-95, 2003.

[9] T. E. Uribe and S. Cheung, "Automatic analysis of firewall and network intrusion detection system configurations," in *Proceedings of the 2004 ACM Workshop on Formal Methods in Security Engineering*, Washington DC, October 29 - 29, 2004, pp. 66 - 74.

[10] T. Jaeger, A. Edwards, and X. Zhang, "Managing access control policies using access control spaces," in *Proceedings of the Seventh ACM Symposium on Access Control Models and Technologies*, Monterey, California, June 03 - 04, 2002, pp. 3-12.

[11] G. Zanin and L. V. Mancini, "Towards a formal model for security policies specification and validation in the selinux system," in *Proceedings of the Ninth ACM Symposium on Access Control Models and Technologies*, Yorktown Heights, New York, June 02 - 04, 2004, pp. 136-145.

[12] N. Damianou, N. Dulay, E. Lupu, and M. Sloman, "The ponder policy specification language," in *Proceedings of the International Workshop on Policies for Distributed Systems and Networks*, January 29 - 31, 2001, pp. 18-38.

[13] C. Ribeiro and P. Guedes, "SPL: An access control language for security policies with complex constraints," INESC, Technical Report RT/0001/99, Jan. 1999.

[14] J. Lobo, R. Bhatia, and S. Naqvi, "A policy description language," in *Proceedings of the Sixteenth National Conference on Artificial Intelligence and the Eleventh Innovative Applications of Artificial Intelligence Conference Innovative Applications of Artificial intelligence*, Orlando, Florida, July 18 - 22, 1999.

[15] E. Moses, *eXtensible Access Control Markup Language (XACML) Version 2.0*, OASIS, Feb. 2005.

[16] C. Sellers, J. Athey, S. Shimko, A. Wilson, F. Mayer, and K. MacMillan, "Experiences implementing a higher-level policy language for SE-Linux," Tresys Technology, White Paper, March 2006.

[17] P. Bonatti, S. De Capitani di Vimercati, and P. Samarati, "A modular approach to composing access control policies," in *Proceedings of the 7th ACM Conference on Computer and Communications Security*, Athens, Greece, November 01 - 04, 2000, pp. 164-173.

[18] S. D. Di Vimercati and P. Samarati, "Authorization specification and enforcement in federated database systems," *Journal of Computer Security*, vol. 5, no. 2, pp. 155-188, Jan. 1997.

[19] J. Hale, P. Galiasso, M. Papa, and S. Shenoi, "Security policy coordination for heterogeneous information systems," in *Proceedings of the 15th Annual Computer Security Applications Conference*, Washington, DC, December 06 - 10, 1999, p. 219.

[20] R. Hull, B. Kumar, and D. Lieuwen, "Towards federated policy management," in *Proceedings of the 4th IEEE Iinternational Workshop on Policies For Distributed Systems and Networks*, Washington, DC, June 04 - 06, 2003, p. 183.

[21] S. Castano, S. D. di Vimercati, and M. G. Fugini, "Automated derivation of global authorizations for database federations," *Journal of Computer Security*, vol. 5, pp. 271-302, 1997.

[22] C. Ribeiro, A. Z. Uquete, P. Ferreira, and P. Guedes, "Security policy consistency," INESC, Technical Report RT/03/00, 2000.

[23] E. Bertino, S. Castano, and E. Ferrari, "On specifying security policies for web documents with an XML-based language," in *Proceedings of the Sixth ACM Symposium on Access Control Models and Technologies*, SACMAT '01, Chantilly, Virginia, 2001, pp. 57-65.

[24] Y. Deng, J. Wang, J. J. Tsai, and K. Beznosov, "An approach for modeling and analysis of security system architectures," *IEEE Transactions on Knowledge and Data Engineering* vol. 15, no. 5, pp. 1099-1119, Sep. 2003.

[25] J. D. Guttman, "Filtering postures local enforcement for global policies," in *Proceedings of IEEE Symposium on Security and Privacy*, May 4-7, 1997, pp. 120 – 129.

[26] S. Ioannidis, "Security policy consistency and distributed evaluation in heterogeneous environments," PhD dissertation, University of Pennsylvania in Partial, 2005.

[27] J. Porto de Albuquerque, H. Krumm, and P. Licio de Geus, "Policy modeling and refinement for network security systems," in *Proceedings of the Sixth IEEE International Workshop on Policies for Distributed Systems and Networks, Policy'05,* Washington, DC, June 06 - 08, 2005, pp. 24-33.

[28] F. Caldeira and E. Monteiro, "Policy-based networking: Applications to firewall management," *Annals of Telecommunications*, CNET, vol. 59, no 1, pp. 38-54 2003.

[29] E. Al-Shaer, H. Hamed, R. Boutaba, and M. Hasan, "Conflict classification and analysis of distributed firewall policies," *IEEE Journal on Selected Areas in Communications*, vol. 23, no. 10, pp. 2069- 2084, Oct. 2005.

[30] E. Al-Shaer and H. Hamed, "Modeling and management of firewall policies," *IEEE Trans. Network and Service Management*, vol. 1, no. 1, Apr. 2004.

[31] K. Golnabi, R. K. Min, L. Khan, and E. Al-Shaer, "Analysis of firewall policy rule using data mining techniques," in *Proceedings of the 10th IEEE/IFIP Network Operations and Management Symposium*, NOMS, 2006, pp. 305- 315.

[32] National Security Agency (NSA), 2006, www.nsa.gov.

[33] T. Jaeger, R. Sailer, and X. Zhang, "Analyzing integrity protection in the SELinux example policy," in *Proceedings of the 11th USENIX Security Symposium*, USENIX, August 2003, pp. 59-74.

[34] P. A. Loscocco and S. D. Smalley, "Meeting critical security objectives with security-enhanced Linux," in *Proceedings of the 2001 Ottawa Linux Symposium*, July 2001.

[35] M. Archer, E. Leonard, and E. Pradella, "Towards a methodology and tool for the analysis of security-enhanced Linux security policies," NRL, Wash. DC, Tech. Rep. NRL/MR/5540-- 02-8629, August 16, 2002.

[36] M. Archer, E. Leonard, and M. Pradella, "Analyzing security-enhanced Linux policy specifications," in *Proceedings of the 4th IEEE international Workshop on Policies For Distributed Systems and Networks,* Washington, DC, June 04 - 06, 2003, p. 158.

[37] M. Archer, E. Leonard, and M. Pradella, "Modeling security-enhanced Linux policy specifications for analysis," in *Proceedings of DARPA Information Survivability Conference and Exposition,* April 22-24, 2003, pp. 164-169.

[38] M. Archer, "TAME: Using PVS strategies for special-purpose theorem proving," *Annals of Mathematics and Artificial Intelligence*, vol. 29, pp. 139-181, Jan. 2001.

[39] S. Owre, N. Shankar, J. M. Rushby, *User Guide for the PVS Specification and Verification System*, Computer Science Laboratory, SRI International, Menlo Park, CA, Feb. 1993.

[40] M. Bishop, *Computer Security: Art and Science*, Singapore: Pearson Education (Singapore) Pte. Ltd., 2003.